Co-funded by the
Erasmus+ Programme
of the European Union

Code4SP E-book

WP3:

Code4SP Training Materials

Prepared by:

Project Information

Project Acronym: Code4SP

Project Title: Coding for Social Promotion

Project Reference: 621417-EPP-1-2020-1-PT-EPPKA3-IPI-SOC-IN

Project website: www.code4sp.eu

Authoring Partner: CodeDoor, CEPROF, SHA and CIP

Document Version: 2

Date of Preparation: 15/04/2022

Document History			
Date	Version	Author	Description
04/04/2022	1	Various Authors	Draft
15/04/2022	2	Various Authors	Final Version

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Introduction

This e-book is part of the Code4SP KA3, reference number 621417-EPP-1-2020-1-PT-EPPKA3-IPI-SOC-IN, co-funded by the Erasmus+ Programme, KA3 – Policy Reform, of the European Commission.

The project is coordinated by SPEL and the consortium is composed of seven more partners (CEPROF, CIP – Citizens in Power, CSI - Center for Social Innovation Ltd., CODEDOOR. ORG EV, ZAUG, Action Sinergy SA and Social Hackers Academy), from four countries (Cyprus, Greece, Germany and Portugal).

This project aims to:

- Generate socio-economic promotion, by providing job-market-oriented training on computer programming.

- Transfer currently established good practices on non-formal education, on computer programming, to the Southern European countries, that are, simultaneously, deemed more economically vulnerable and subjected to an unprecedented exposure to migratory waves, of people with low socio-economic condition.

The e-book is drawn from the content of the implementation guide previously elaborated in the scope of the project in order to compile the training materials for coaches/trainers, so that they can implement Code4SP methodology in their countries. The training materials are composed of lesson scenarios, PPTs, ready-made tutorials and any other adjusted curriculum that will have emerged from the original CodeDoor methodology, as well as from successful best practices of each national context.

Table of Contents

1. Computer Programming and its basic concepts

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Topic Information

Topic:

1. Computer Programming and its basic concepts

Prerequisites:

Basic computer literacy, basic software installed, and basic knowledge of working with files.

Workload:

5 hours.

Description:

In this topic, we give a brief introduction to computer programming and its basic concepts. We handle hardware and software concepts as well as how computers tore data. In the end we take a lot at how a program works and what type of programming languages are out there.

Learning outcomes:

- Recognize the concept of HyperText Markup Language (HTML) in the family of document description languages.
- Distinguish between the structure, content and styles of a page.
- Use HTML in the construction of pages for the web.

Material required:

- Computer or laptop
- Internet connection

Lesson Scenario:

The total time for this topic is 5 hours, and it will be up to the trainer/coach to decide how much time to dedicate to teaching each subtopic. In order to make the most of all the time available, we propose the use of the training materials produced by the project

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

(PPT presentations), which were designed with an effective use of time in mind. These presentations are composed of the following elements:

- Development of the subtopic and main ideas to retain;
- Proposed Activities/Exercises.

That said, if the trainer/coach follows the logical sequence of the PPTs, he/she will certainly be able to complete the session within the stipulated time limit. These presentations can also be made available to learners for individual study.

Subtopics:

1.1. Introduction to computers and programming

1.2. Hardware and Software Concept

1.3. How computers store data

1.4. How a program works

1.5. Programs and Program Languages

Additional resources:

- Khan Academy: useful resources computer topics in various languages
- Marc Andreessen: https://future.a16z.com/software-is-eating-the-world/

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

1.1. Introduction to computers and programming

Computers are machines that can be programmed to carry out a sequence of instructions. Programming is the process of designing those instructions. Programs are written in a particular language which provides a structure for the programmer and uses specific instructions to control the sequence of operations that the computer carries out. „Software is eating the world“ (Marc Andreessen, https://future.a16z.com/software-is-eating-the-world/) so make your learners aware of the fact that nowadays nearly everything is computerized and programmed (see also: Internet of Things).

What are some of the ways you use computers?

Let your learners do a brainstorming or create a mind map on how they are actually using computers. There are many ways you can use computers in your everyday life. Here are a few of the most common ways:

- To surf the internet
- To check your email
- To do research for a project
- To play games
- To watch movies or TV shows
- To listen to music
- To work on a document or spreadsheet
- To read the news

What other devices are computers?

Computers are not just the devices that sit on our desks or in our pockets. They are also the devices that run our cars, planes, and boats. They are in our televisions, refrigerators, and even our watches. In short, computers are everywhere.

What software have you used?

Let your learners do a brainstorming or create a mind map on what software they have already used. Try to make them aware that every Software was written by some developers. Even the Software that resides on the smartphones, cash-dispensers or even televisions they use everyday.

1.2. Hardware and Software Concept

Definition of hardware and software

Hardware refers to the physical components of a computer system, while software refers to the instructions and data that make the computer work.

The main components of a computer and its functions

A computer has four main components: the central processing unit (CPU), the memory, the input devices, and the output devices. The CPU is the part of the computer that performs the calculations and controls the other parts. The memory is where the computer stores the data it is working on. The input devices are the devices that the user uses to enter data into the computer, such as the keyboard and the mouse. The output devices are the devices that the computer uses to display the results of its calculations, such as the monitor and the printer.

1.3 How computers store data

Concept: the binary system

A binary system is a way of representing information using two symbols: 0 and 1. Binary is the simplest form of information representation and is used in computer systems to store and process information. Binary is a convenient way to represent information because it is very simple and can be processed by computer systems very easily.

Storing numbers, characters

ASCII

Computer systems store text and numbers in a variety of ways, each with its own benefits and drawbacks. The most common way to store text is as ASCII (American

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Standard Code for Information Interchange) characters. In ASCII, each character is represented by a number, from 0 to 127. This number is called the character's ASCII code. When a computer stores text as ASCII characters, it simply stores the ASCII codes for each character in the text.

Unicode

Another way to store text is as Unicode characters. Unicode is an international standard that defines a unique number for every character in every language. When a computer stores text as Unicode characters, it stores the Unicode code for each character in the text.

UTF-8

UTF-8 is a character encoding that can store text and numbers in a single Unicode character. This encoding is widely used on the internet because it can encode all characters in a variety of languages. UTF-8 is a variable-length encoding, which means that it can encode characters of different sizes. The smallest encoding is 1 byte, and the largest encoding is 4 bytes. This encoding is backwards compatible with ASCII, which means that ASCII text will be encoded in UTF-8 using 1 byte.

Numbers

The most common way to store numbers is as binary integers. In binary, each number is represented by a string of 0s and 1s. For example, the number 12 can be represented as: 01001000 The number 12 can also be represented in hexadecimal, which is a base 16 numbering system. In hexadecimal, each number is represented by a string of hexadecimal digits. For example, the number 12 can be represented as: C When a computer stores a number in binary or hexadecimal, it stores the number's integer value.

Other types of data

Computers are often referred to as digital devices. The term digital can be used to describe anything that uses binary numbers. Binary data is data that is stored in binary, and a digital device is any device that works with binary data. In this section we have discussed how numbers and characters are stored in binary, but computers also work

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

with many other types of digital data. For example, consider the pictures that you take with your digital camera. These images are composed of tiny dots of color known as pixels. (The term pixel stands for picture element.) Each pixel in an image is converted to a numeric code that represents the pixel’s color. The numeric code is stored in memory as a binary number.

The music that you play on your CD player, iPod or MP3 player is also digital. A digital song is made up of small pieces of music called samples. Each sample is turned into a binary number, which can be stored in a computer's memory. The more samples that a song has, the more like the original music it will sound when it's played back. A CD quality song has more than 44,000 samples per second!

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

1.4. How a program works

There are many different types of computer programs, but they all have the same basic components: a user interface, a processor, and memory. The user interface allows the user to input information and instructions into the program, the processor carries out the instructions, and the memory stores the program and the data it processes. Most computer programs are written in a high-level programming language, which is a language that is designed to be easy for humans to read and write. However, the processor can only understand machine code, which is a series of ones and zeroes. So, before a program can be run, it must be converted into machine code. This is done by a program called a compiler. The compiler reads the program and converts it into machine code. It then stores the machine code in a file called an executable. When the user runs the program, the executable is loaded into memory and the processor carries out the instructions.

The fetch-decode-execute cycle

The fetch-decode-execute cycle is the basic process that a computer uses to carry out instructions. The cycle begins when the computer fetches an instruction from memory. It then decodes the instruction to determine what it is supposed to do. Finally, it executes the instruction. The cycle then repeats, fetching the next instruction from memory.

From machine language to assembly language

As programming in machine language, which consist only of binary code is too complicated for a human being, assembly language was created. Assembly language is a low-level programming language for a computer, microprocessor, or other programmable device, in which the programmer uses assembly language instructions to control the operation of the device. Assembly language is specific to a certain microprocessor or family of microprocessors. It consists of a series of mnemonic codes, symbolic names for the operations that the microprocessor can perform, and

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

the operands (data) upon which these operations are to be performed. Assembly language is converted into machine code, a form of binary code that is specific to a particular type of computer and can be understood by the computer's processor. Machine code is the only form of code that the processor can directly execute.

Even assembly language programming is easier than machine language programming it was not handy enough to produce fast and easy to read source code. Therefore high-level programming languages (such as C# or python) where created.

Today there are many high-level programming languages. Some of the more common ones are Java, Python, and Ruby. High-level programming languages are easier to use than low-level programming languages. They allow you to focus on the task at hand, rather than on the details of the computer. This makes them ideal for creating applications and programs. High-level programming languages also tend to be more forgiving than low-level programming languages. If you make a mistake when writing code in a high-level language, the compiler will usually be able to correct it for you. This can save you a lot of time and frustration when coding.

Key Words, Operators, and Syntax: an overview

There are many high-level programming languages available today. Each has its own unique set of keywords, operators, and syntax. In order to be effective with a high-level programming language, it is important to be familiar with the specific keywords, operators, and syntax used by that language. Some of the most common keywords used in high-level programming languages include: *if, then, else, while, for, do, break, continue*. These keywords are used to control the flow of program execution. Operators are symbols that represent operations that can be performed on values. The most common operators include: + *(addition), - (subtraction), * (multiplication), / (division), and % (modulus)*. These operators can be used to calculate the results of expressions. The syntax of a programming language is the set of rules that govern how code must be written in order to be interpreted by the compiler or interpreter. The syntax of a high-level programming language is typically more forgiving than the syntax of a lower-level language. This can make it easier for beginners to learn how to program.

Compilers and Interpreters

Computer compilers and interpreters are important tools for software developers. A compiler takes code written in one language and converts it into code that can be run on a different machine. An interpreter takes code written in one language and runs it as it is, without compiling it first. Compilers are typically used for languages that have a lot of structure, like C or Java. Interpreters are typically used for languages that are more flexible, like Python or Ruby. Compilers usually produce faster code than interpreters. However, interpreters are typically more portable, meaning they can run on more types of machines. Which tool to use depends on the situation. If speed is important, a compiler is a better choice. If portability is important, an interpreter is a better choice.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

1.5. Programs and Program Languages

Programming languages are used to create programs, which are used to control the behavior of a machine, typically a computer. A programming language provides a structure for the programmer to give instructions to the machine, and a way to communicate those instructions to other programmers. There are many programming languages in use today. The most popular ones are C, Java, Python, and JavaScript.

Types of languages

There are dozens of programming languages in use today, but they can be broadly classified into five categories:

- **Low-level programming languages:** These programming languages are very close to the hardware and are used to program microprocessors and other low-level devices. They are not easy to learn and are not popular for general-purpose programming. Examples: Assembly language, C programming language, and Low-level assembly language.
- **High-level programming languages:** These programming languages are designed to be easy to learn and use. They are popular for general-purpose programming. Examples: Java, C++, and Python.
- **Scripting languages:** Scripting languages are designed to be easy to use and are popular for scripting purposes. Examples: Python, Ruby, and JavaScript.
- **Domain-specific languages:** Domain-specific languages are designed for a specific task or industry. They are not easy to learn and are not popular for general-purpose programming. Examples: MATLAB, SQL, and FORTRAN.
- **Object-oriented programming languages:** These programming languages are based on the object-oriented programming paradigm. Examples: Java, C++, and Python.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

From a high-level program to an executable file

When a computer program is written in a high-level language, it is first translated into a lower-level language, which is more easily understood by machines. The lower-level language is then compiled into an executable file, which can be run on a computer.

IDEs (Integrated Development Environments)

An Integrated Development Environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE typically consists of a source code editor, build automation tools, and a debugger. The source code editor allows the programmer to write code, while the build automation tools automate the process of compiling that code into a form the computer can run. The debugger allows the programmer to step through the code, examining the state of the program at each point in its execution. IDEs are often used in conjunction with a version control system, which allows different programmers working on the same project to share and merge their changes seamlessly.

The common elements in programming languages

Computer programming languages share a number of common elements, despite their differences. All programming languages have a way of representing instructions to the computer in a form that the computer can understand. This is usually called code, or source code. Programmers use code to create software programs and applications. All programming languages also have a way of organizing instructions so that they can be reused, modified, or shared with other programmers. This is usually called a library or module. Libraries and modules allow programmers to create complex programs by building on the work of other programmers. Finally, all programming languages have a way of conveying information to the user about what the program is doing and how it is performing. This is usually called output or debug information. Output and debug information helps programmers understand and fix problems with their programs.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Procedural and object-oriented programming

There are two main types of programming: procedural and object-oriented. Procedural programming involves a step-by-step process, while object-oriented programming involves creating objects that interact with one another. Procedural programming is often seen as simpler than object-oriented programming. It is easy to learn the steps required to complete a task, and it is easy to change the order of those steps without affecting the outcome. However, procedural programming can be less efficient because it can be difficult to reuse code that has been written for a specific task. Object-oriented programming is more complex than procedural programming, but it allows for more flexibility and reuse of code. Objects can be created for specific tasks and then reused as needed. In addition, object-oriented code is often easier to read and understand than procedural code. However, object-oriented programming can be more difficult to learn and may be less efficient than procedural programming.

2. HTML – HyperText Markup Language

Topic Information

Topic:

2. HTML

Prerequisites:

Basic computer literacy, basic software installed, and basic knowledge of working with files.

Workload:

10 hours.

Description:

In this topic, we cover the basics of HTML, to get learners up to speed in the world of programming, after having acquired some basic concepts. We define the elements, attributes and all the other important terms they may have heard and where these terms fit into the language. We also show how an HTML element is structured, how a typical HTML page is structured and explain other important basic features of the language.

Learning outcomes:

- Recognize the concept of HyperText Markup Language (HTML) in the family of document description languages.
- Distinguish between the structure, content and styles of a page.
- Use HTML in the construction of pages for the web.

Material required:

- Computer or laptop
- Internet connection
- Online website builder (https://sites.google.com/new)
- Online text editor (https://www.w3schools.com/html/default.asp)

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Lesson Scenario:

The total time for this topic is 10 hours, and it will be up to the trainer/coach to decide how much time to dedicate to teaching each subtopic. In order to make the most of all the time available, we propose the use of the training materials produced by the project (PPT presentations), which were designed with an effective use of time in mind. These presentations are composed of the following elements:

- Development of the subtopic and main ideas to retain;
- Proposed Activities/Exercises.

That said, if the trainer/coach follows the logical sequence of the PPTs, he/she will certainly be able to complete the session within the stipulated time limit. These presentations can also be made available to learners for individual study.

Subtopics:

2.1. The basics of HTML

2.2. HTML advanced concepts

2.3. HTML5 features

2.4. HTML5 references

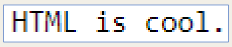
Additional resources:

- HTML reference guide
- W3Schools - Guide for every HTML element and CSS rule, and examples for each one of them
- Khan Academy: useful resources and videos on coding HTML & CSS, in many different languages

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

2.1. The basics of HTML

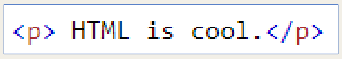
HTML (Hypertext Markup Language) **is not** a programming language. It is a markup language that communicates web browsers how to structure the web pages visited. It can be as complex or as simple as the web developer wants it to be. HTML comprises a series of elements, which you use to enclose, wrap, or rise different parts of content to make it appear or act in a certain way. The enclosing tags can make content into a hyperlink to connect to another page, italicize words, and so on. For example, considering the following line of text:



```
HTML is cool.
```

Figure 1 *- Line of text "HTML is cool" (**Source:** Author)*

If one wanted the text to stand by itself, one could specify that it is a paragraph by enclosing it in a paragraph (<p>) element:



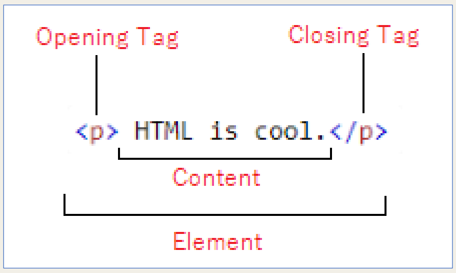
```
<p> HTML is cool.</p>
```

Figure 2 *– Coding for the paragraph "HTML is cool" (**Source:** Author)*

Note: Tags in HTML are not case-sensitive. This means they can be written in uppercase or lowercase. For example, a <title> tag could be written as <title>, <TITLE>, <Title>, <TiTlE>, etc., and it will work. Nevertheless, it is best practice to write all tags in lowercase for consistency and readability.

Anatomy of an HTML element

Let's further explore our paragraph element from the previous section:



```
<p> HTML is cool.</p>
```

***Figure 3** – Anatomy of an HTML element (**Source:** Author)*

Therefore, the anatomy of the element is composed of:

The opening tag: the name of the element (in this example, *p* for paragraph), wrapped in opening and closing angle brackets. This opening tag marks where the element begins or starts to take effect. In this example, it comes first, at the start of the paragraph text.

The content: This is the content of the element. In this example, it is the paragraph text.

The closing tag: This is the same as the opening tag, except that it includes a forward slash before the element name. This marks where the element ends. Failing to include a closing tag is a common beginner error that can produce peculiar results.

The **element** is the opening tag, followed by content, followed by the closing tag.

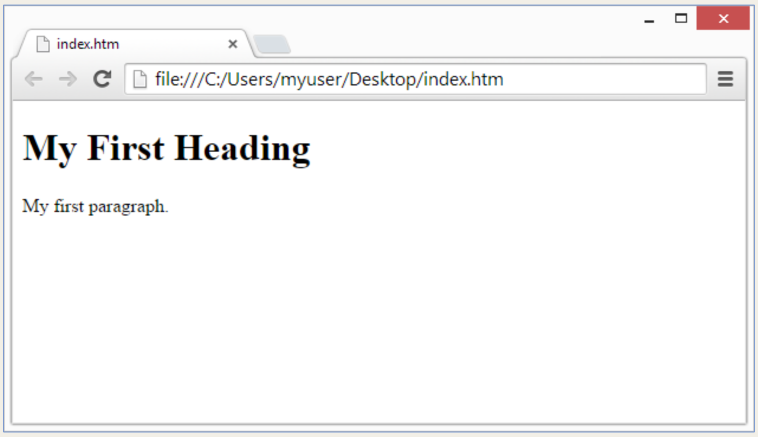
Note: Some HTML elements have no content (as the <br> element). These elements are named “empty elements”. They do not have an end tag!

Web Browsers

The purpose of a web browser (Chrome, Edge, Firefox, Safari) is to read HTML documents and display them properly.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

A browser does not display the HTML tags. It uses them to determine how to display the document:



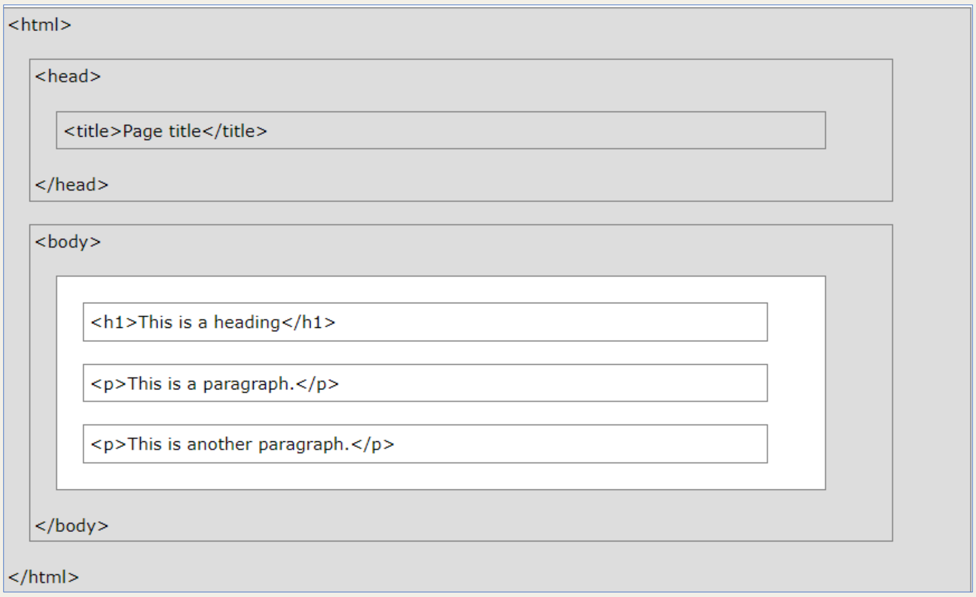
My First Heading

My first paragraph.

Figure 4 *– An HTML document determined in a web browser (**Source:** https://www.w3schools.com/html/html_intro.asp)*

HTML Page Structure

An HTML page should be structured as follows:



```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
      <h1>This is a heading</h1>
      <p>This is a paragraph.</p>
      <p>This is another paragraph.</p>
  </body>
</html>
```

Figure 5 *– Structure of an HTML page (**Source:** https://www.w3schools.com/html/html_intro.asp)*

The content inside the <body> section (the white area above) will be displayed in a browser. The content inside the <title> element will be shown in the browser's title bar or in the page's tab.

HTML History

Since the early days of the World Wide Web, there have been many versions of HTML:

Year	Version
1989	Tim Berners-Lee invented www
1991	Tim Berners-Lee invented HTML
1993	Dave Raggett drafted HTML+
1995	HTML Working Group defined HTML 2.0
1997	W3C Recommendation: HTML 3.2
1999	W3C Recommendation: HTML 4.01
2000	W3C Recommendation: XHTML 1.0
2008	WHATWG HTML5 First Public Draft
2012	WHATWG HTML5 Living Standard
2014	W3C Recommendation: HTML5
2016	W3C Candidate Recommendation: HTML 5.1
2017	W3C Recommendation: HTML5.1 2nd Edition
2017 -	W3C Recommendation: HTML5.2

***Table 1** – HTML story (**Source:** https://www.w3schools.com/html/html_intro.asp)*

This manual follows the latest HTML5 standard.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

HTML Editors

A simple text editor is all you need to learn HTML.

Learn HTML Using Notepad or TextEdit

Web pages can be created and modified by using professional HTML editors. However, for learning HTML a simple text editor like Notepad (PC) or TextEdit (Mac) is recommended. Using a simple text editor can be a good way to learn HTML
The steps below should be followed to create learners' first web page with Notepad or TextEdit.

Step 1: Open Notepad (PC)

(Windows 8 or later: Open the Start Screen (the window symbol at the bottom left on your screen). Type Notepad.

Windows 7 or earlier: Open Start > Programs > Accessories > Notepad)

- Open TextEdit (Mac)
- Open Finder > Applications > TextEdit
- Get the application to save files correctly. In Preferences > Format > choose "Plain Text"
- Then under "Open and Save", check the box that says "Display HTML files as HTML code instead of formatted text".
- Then **open a new document** to place the code.

Step 2: Writing Some HTML

- Write or copy the following HTML code into Notepad:

```
<!DOCTYPE html>

<html>
```

```
<body>
<h1> My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

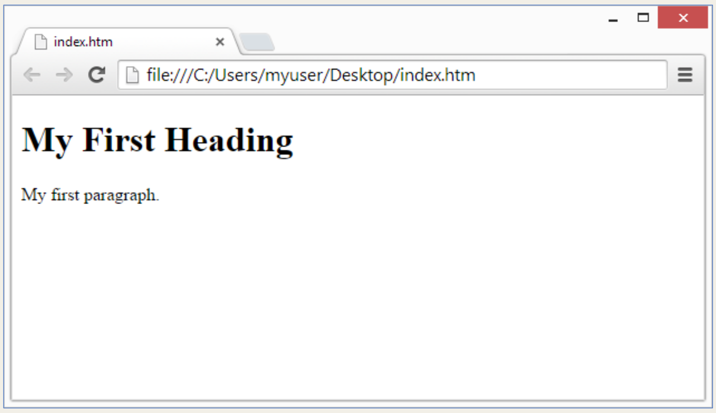
Step 3: Save the HTML Page

- Save the file on your computer.
- Select File > Save as in the Notepad menu.
- Name the file "index.htm" and set the encoding to UTF-8 (which is the preferred encoding for HTML files).

Step 4: View the HTML Page in Your Browser

- Open the saved HTML file in your favorite browser (double click on the file, or right-click - and choose "Open with").

The result will be similar to the following:



My First Heading

My first paragraph.

*Figure 6 – An HTML document determined in a web browser (**Source:** https://www.w3schools.com/html/html_intro.asp)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

HTML Basic Examples

In this section, some basic HTML examples will be shared.

HTML Documents

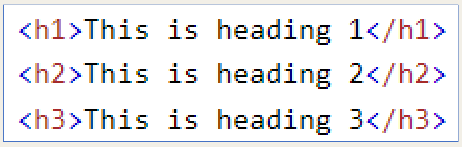
All HTML documents must start with a document type declaration: <!DOCTYPE html>.

The <!DOCTYPE> declaration represents the document type and helps browsers to display web pages correctly. It must only appear once, at the top of the page (before any HTML tags). It is not case sensitive.

The <!DOCTYPE> declaration for HTML5 is: `<!DOCTYPE html>`

HTML Headings

HTML headings are defined with the <h1> to <h6> tags.
<h1> defines the most important heading. <h6> defines the least important heading:

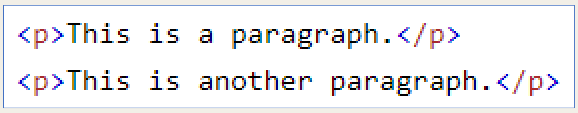


```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
```

***Figure 7** – HTML headings (**Source:** https://www.w3schools.com/html)*

HTML Paragraphs

HTML paragraphs are defined with the <p> tag:

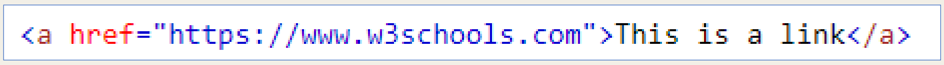


```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
```

***Figure 8** – HTML paragraphs (**Source:** https://www.w3schools.com/html)*

HTML Links

HTML links are defined with the <a> tag:



```
<a href="https://www.w3schools.com">This is a link</a>
```

***Figure 9** – HTML link (**Source:** https://www.w3schools.com/html)*

The link's destination is specified in the href attribute.

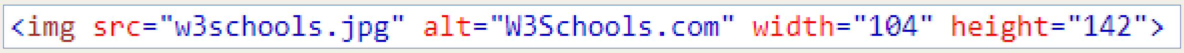
Attributes are used to provide additional information about HTML elements.

More about attributes will be taught at a later stage.

HTML Images

HTML images are defined with the <img> tag.

The source file (src), alternative text (alt), width, and height are provided as attributes:



```
<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">
```

***Figure 10** – HTML images (**Source:** https://www.w3schools.com/html)*

How to View HTML Source?

View HTML Source Code:

- Right-click in an HTML page and select "View Page Source" (in Chrome) or "View Source" (in Edge), or similar in other browsers. This will open a window containing the HTML source code of the page.

Inspect an HTML Element:

- Right-click on an element (or a blank area) and choose "Inspect" or "Inspect Element" to see what elements are made up of (you will see both the HTML and the CSS). You can also edit the HTML or CSS on-the-fly in the Elements or Styles panel that opens.

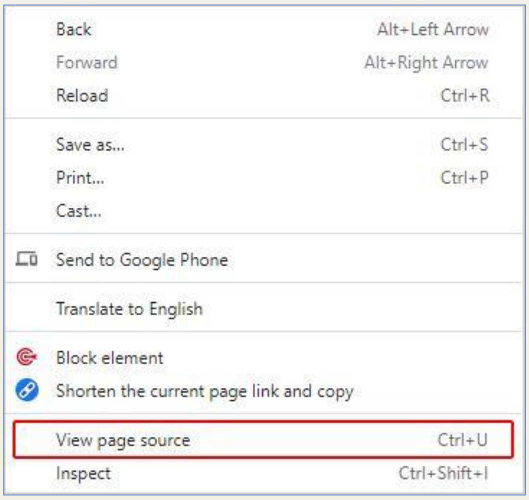
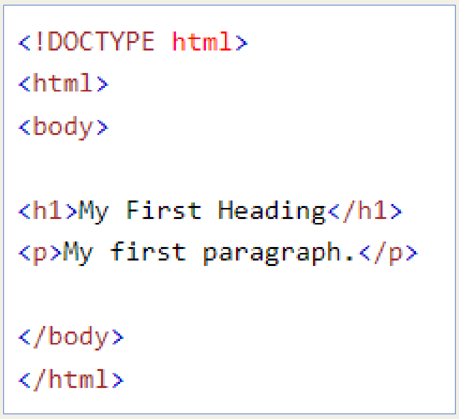


***Figure 11** – How to view page source (**Source:**Author)*

The structure of a HTML document

The HTML document itself begins with <html> and ends with </html>. The visible part of the HTML document is between <body> and </body>.



```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

***Figure 12** – Document type declaration, HTML and visible part of the HTML document (**Source:** https://www.w3schools.com/html)*

HTML Attributes

HTML attributes provide additional information about HTML elements, and all HTML elements can have them. Attributes provide additional information about elements, being always specified in the start tag. They usually come in name/value pairs like: name="value".

List of common attributes:

- **href** – the <a> tag defines a hyperlink. The href attribute specifies the URL of the page the link goes to, as follows:

```
<a href="https://www.w3schools.com">Visit W3Schools</a>
```

- **src** – the <img> tag is used to embed an image in an HTML page. The src attribute specifies the path to the image to be displayed, as seen below:

```
<img src="img_girl.jpg">
```

There are two ways to specify the URL in the src attribute:

1. **Absolute URL** - Links to an external image that is hosted on another website. E.g.: *src="https://www.w3schools.com/images/img_girl.jpg".*

Notes: External images might be under **copyright**. If one does not get permission to use it, he/she may be in violation of copyright laws. In addition, external images cannot be controlled; it can abruptly be removed or changed.

2. **Relative URL** - Links to an image that is hosted within the website. Here, the URL does not include the domain name. If the URL begins without a slash, it will be relative to the current page. E.g*: src="img_girl.jpg"*. If the URL begins with a slash, it will be relative to the domain. E.g.: *src="/images/img_girl.jpg".*

Hint: It is almost always best to use relative URLs. They will not break if the domain changes.

- **width and height attributes** – the <img> tag should also comprise the width and height attributes, which stipulates the width and height of the image (in pixels):

```
<img src="img_girl.jpg" width="500" height="600">
```

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- **alt –** the required alt attribute for the <img> tag specifies an alternate text for an image, if the image cannot be displayed for some reason. This can be due to slow connection, or an error in the src attribute, or if the user uses a screen reader.

  ```
  <img src="img_girl.jpg" alt="Girl with a jacket">
  ```

  If we try to display an image that does not exist, the value of the alt attribute will be displayed instead, as follows:

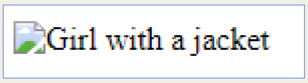
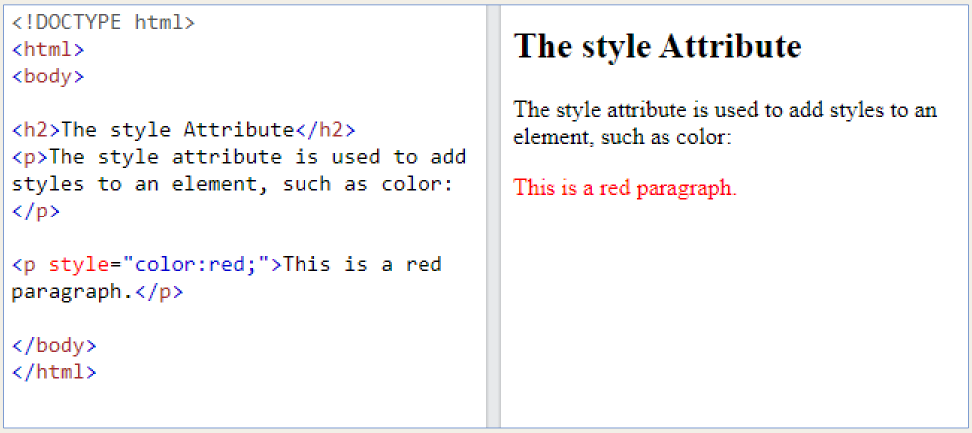

Girl with a jacket

__Figure 13__ – Alt value if image does not exist (__Source:__ https://www.w3schools.com/html)

- **style** – the style attribute is used to add styles to an element, such as color, font, size, and more.

```
<p style="color:red;">This is a red paragraph.</p>
```

Result:



```
<!DOCTYPE html>
<html>
<body>

<h2>The style Attribute</h2>
<p>The style attribute is used to add
styles to an element, such as color:
</p>

<p style="color:red;">This is a red
paragraph.</p>

</body>
</html>
```

The style Attribute

The style attribute is used to add styles to an element, such as color:

This is a red paragraph.

__Figure 14__ – Code for coloring a paragraph (__Source:__ https://www.w3schools.com/html)

- **lang** – the lang attribute should always be included inside the <html> tag, to declare the language of the Web page. This is meant to support search engines and browsers. The example below stipulates English as the language in use:

```
<!DOCTYPE html>
<html lang="en">
```

```
<body>
…
</body>
</html>
```

Country codes can also be added to the language code in the lang attribute. So, the first two characters express the **language** of the HTML page, and the last two characters outline the **country**.

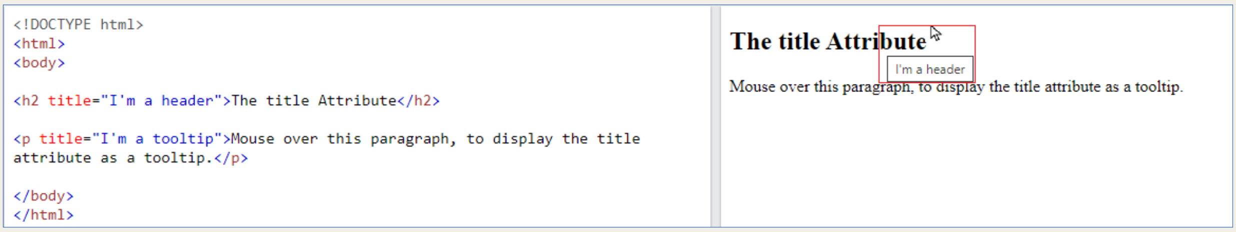
The following example specifies Portuguese as the language and Portugal as the country:

```
<!DOCTYPE html>
<html lang="pt-PT">
<body>
...
</body>
</html>
```

[HTML Language Code Reference](#) includes all the language codes.

- **title –** this attribute describes some additional information about an element.

Its value will be displayed as a tooltip when the mouse pointer goes over the element, as follows:



```
<!DOCTYPE html>
<html>
<body>

<h2 title="I'm a header">The title Attribute</h2>

<p title="I'm a tooltip">Mouse over this paragraph, to display the title
attribute as a tooltip.</p>

</body>
</html>
```

The title Attribute

Mouse over this paragraph, to display the title attribute as a tooltip.

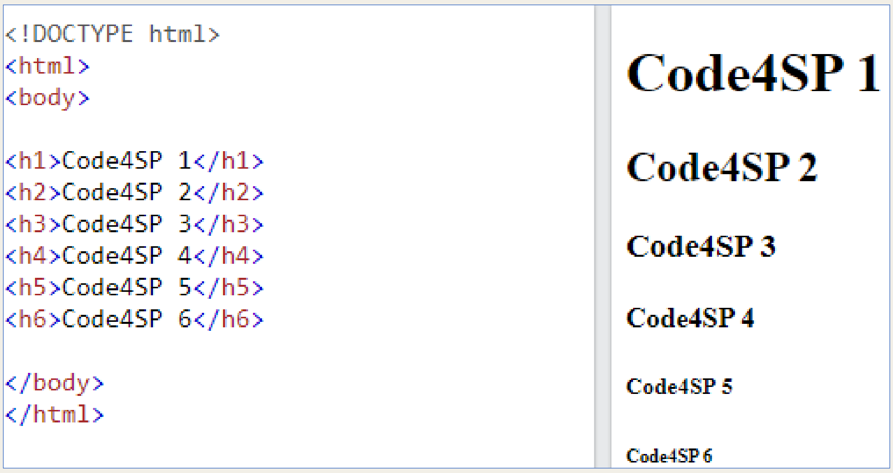
*Figure 15 – Code for displaying a 'title' tooltip (**Source:** Author)*

Recommendation: It is highly recommended to always use lowercase attributes and to always quote attribute values for stricter document types like XHTML.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

HTML Headings

HTML headings are titles or subtitles that one wants to display on a webpage.



```
<!DOCTYPE html>
<html>
<body>

<h1>Code4SP 1</h1>
<h2>Code4SP 2</h2>
<h3>Code4SP 3</h3>
<h4>Code4SP 4</h4>
<h5>Code4SP 5</h5>
<h6>Code4SP 6</h6>

</body>
</html>
```

```
Code4SP 1
Code4SP 2
Code4SP 3
Code4SP 4
Code4SP 5
Code4SP 6
```

***Figure 16** – Code for adding headings (**Source:** Author)*

HTML headings are delineated with the <h1> to <h6> tags. <h1> identifies the most important heading. <h6> outlines the least important heading. <h1> headings should be used for main headings, followed by <h2> headings, then the less important <h3>, and so on.

Headings are of upmost importance since search engines use them to index the structure and content of webpages. Users often browse a page by its headings. It is important to use headings to exhibit the document structure.

It is important to state that, by definition, browsers automatically add a margin before and after a heading.

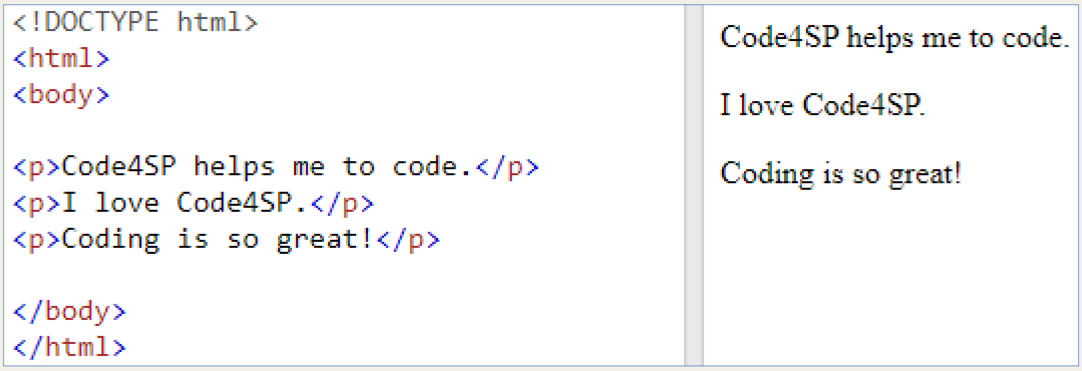
Recommendation: It is highly recommended to use HTML headings for headings only, not to make text big or bold.

Moreover, in the CSS topic, it will be taught that headings’ size can be specified using the **style** attribute, using the CSS font-size property, as follows:

```
<h1 style="font-size:60px;">Heading 1</h1>
```

HTML Paragraphs

A paragraph constantly starts on a new line and is typically a block of text. It is defined by the HTML <p> element and, like headings, browsers automatically add some margin before and after a paragraph.



```
<!DOCTYPE html>
<html>
<body>

<p>Code4SP helps me to code.</p>
<p>I love Code4SP.</p>
<p>Coding is so great!</p>

</body>
</html>
```

Code4SP helps me to code.

I love Code4SP.

Coding is so great!

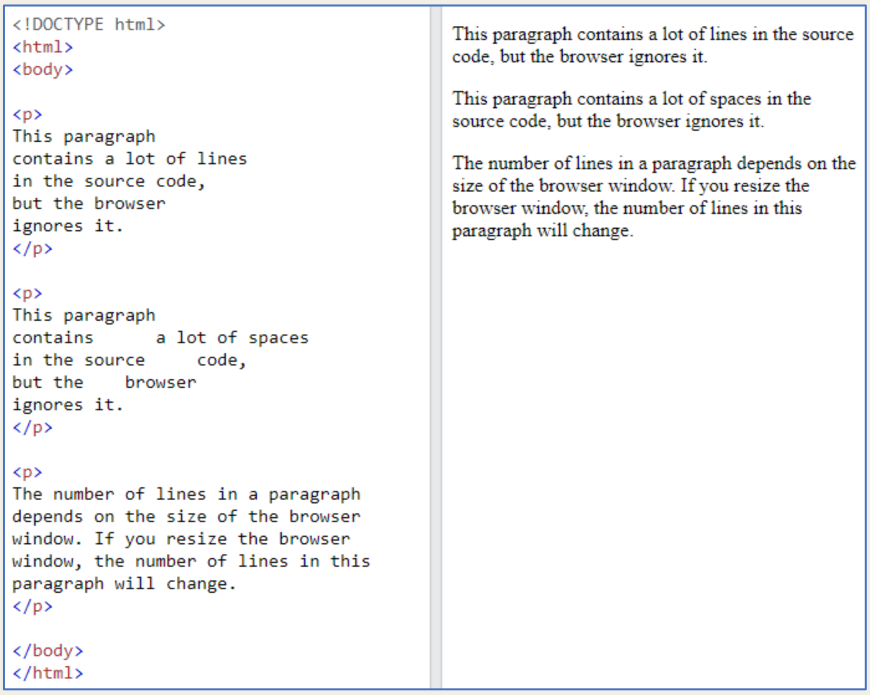
***Figure 17** – Code for adding paragraphs (**Source:** Author)*

HTML Display

One cannot be sure how HTML will be presented, as it can vary from screen to screen. With HTML, the display cannot be changed by adding extra spaces or extra lines in the HTML code.

The browser will automatically remove any extra spaces and lines when the page is displayed, as seen in the following example:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<!DOCTYPE html>
<html>
<body>

<p>
This paragraph
contains a lot of lines
in the source code,
but the browser
ignores it.
</p>

<p>
This paragraph
contains      a lot of spaces
in the source     code,
but the    browser
ignores it.
</p>

<p>
The number of lines in a paragraph
depends on the size of the browser
window. If you resize the browser
window, the number of lines in this
paragraph will change.
</p>

</body>
</html>
```

This paragraph contains a lot of lines in the source code, but the browser ignores it.

This paragraph contains a lot of spaces in the source code, but the browser ignores it.

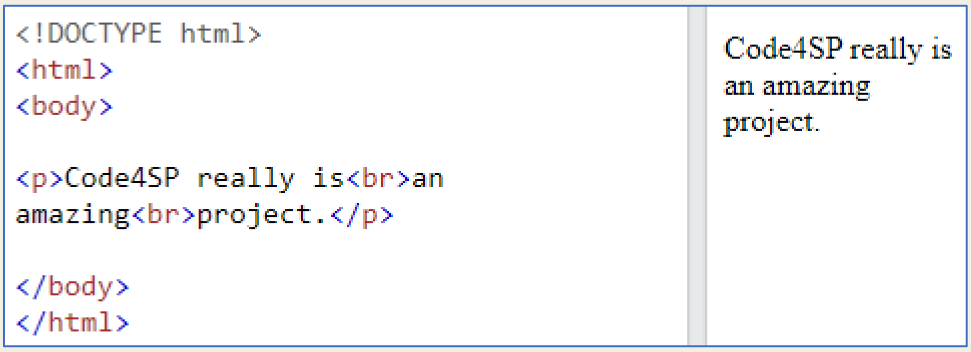
The number of lines in a paragraph depends on the size of the browser window. If you resize the browser window, the number of lines in this paragraph will change.

***Figure 18** – Example of how browsers tend to ignore spaces (**Source:** https://www.w3schools.com/html)*

HTML Line Breaks

The HTML <br> element defines a line break. It is an empty tag, which means that it has no end tag.

<br> should be used if one wants a new line without starting a new paragraph:



```
<!DOCTYPE html>
<html>
<body>

<p>Code4SP really is<br>an
amazing<br>project.</p>

</body>
</html>
```

Code4SP really is
an amazing
project.

***Figure 19** – The <br> element (**Source:** Author)*

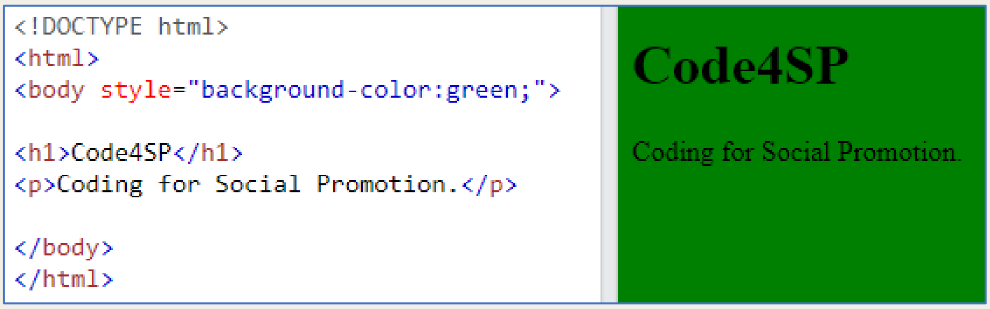
HTML Styles

The HTML style attribute is used to add styles to an element, such as color, font, size, etc. To set the style of an HTML element, the style attribute must be used. It has the following syntax (it should be noted that *property* and *value* are CSS features, to be learned later).

```
<tagname style="property:value;">
```

- **Background Colour**

The CSS *background-color* property specifies the background colour for an HTML element.



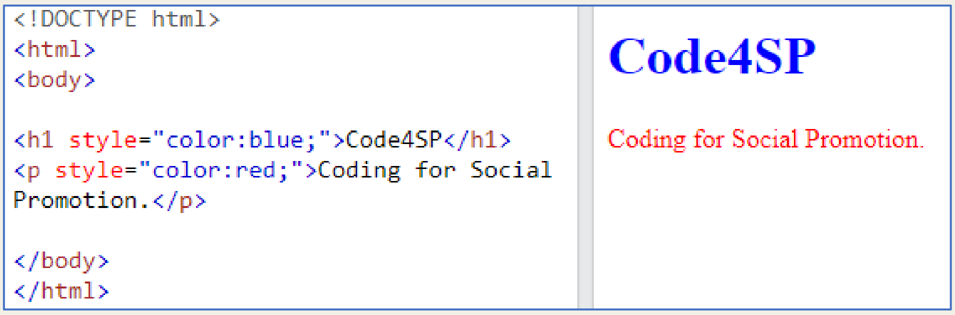
```
<!DOCTYPE html>
<html>
<body style="background-color:green;">

<h1>Code4SP</h1>
<p>Coding for Social Promotion.</p>

</body>
</html>
```

```
Code4SP

Coding for Social Promotion.
```

***Figure 20** – Setting a background colour (**Source:** Author)*

- **Text Colour**

The CSS *color* property outlines the text colour for an HTML element:



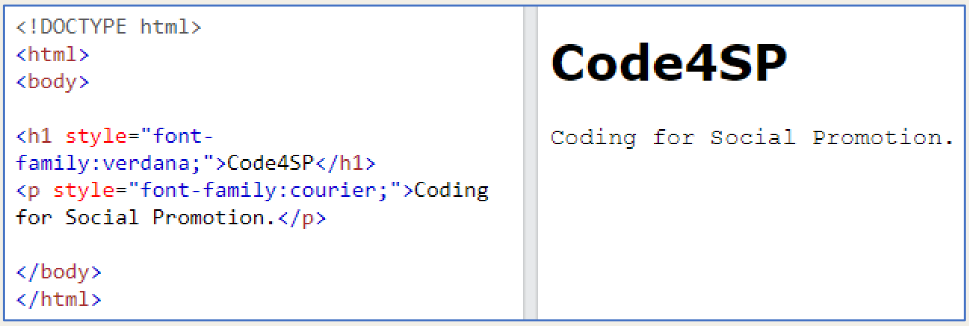
```
<!DOCTYPE html>
<html>
<body>

<h1 style="color:blue;">Code4SP</h1>
<p style="color:red;">Coding for Social
Promotion.</p>

</body>
</html>
```

```
Code4SP

Coding for Social Promotion.
```

***Figure 21** – Setting a background colour (**Source:** Author)*

- **Fonts**

The CSS *font-family* property defines the font to be used for an HTML element:



```
<!DOCTYPE html>
<html>
<body>

<h1 style="font-
family:verdana;">Code4SP</h1>
<p style="font-family:courier;">Coding
for Social Promotion.</p>

</body>
</html>
```

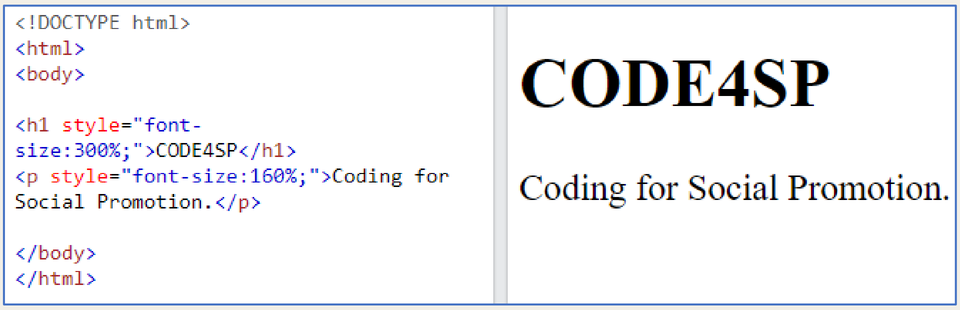
Code4SP

Coding for Social Promotion.

Figure 22 *– Setting the font to be used for an HTML element (**Source:** Author)*

- **Text Size**

The CSS *font-size* property identifies the text size for an HTML element:



```
<!DOCTYPE html>
<html>
<body>

<h1 style="font-
size:300%;">CODE4SP</h1>
<p style="font-size:160%;">Coding for
Social Promotion.</p>

</body>
</html>
```

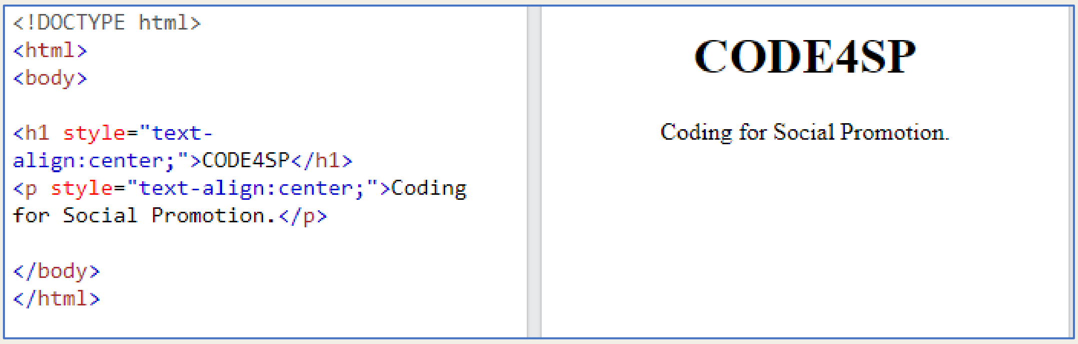
CODE4SP

Coding for Social Promotion.

Figure 23 *– Setting the text size for an HTML element (**Source:** Author)*

- **Text Alignment**

The CSS *text-align* feature defines the horizontal text alignment for an HTML element:



```
<!DOCTYPE html>
<html>
<body>

<h1 style="text-
align:center;">CODE4SP</h1>
<p style="text-align:center;">Coding
for Social Promotion.</p>

</body>
</html>
```

CODE4SP

Coding for Social Promotion.

***Figure 24** – Text Alignment feature (**Source:** Author)*

HTML Text Formatting

HTML comprises various elements for defining text with a special implication (bold, italic, subscript, superscript, etc.).

The following are the **HTML formatting elements**:

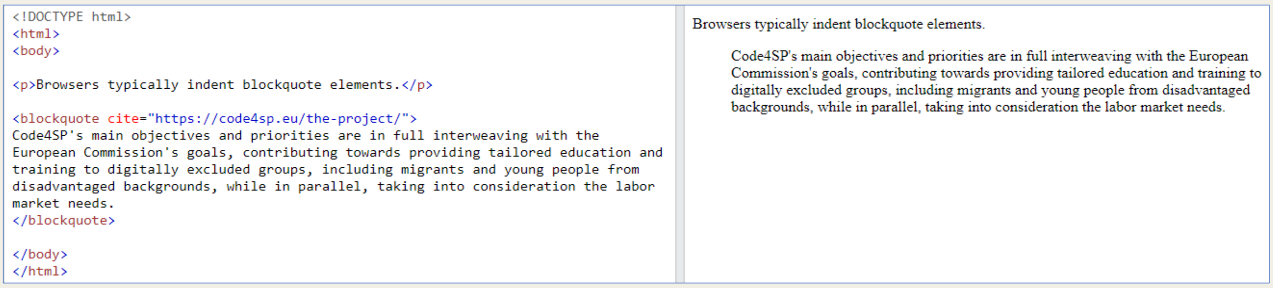
Property	Outcome	Definition	Example
`<b>`	Bold text	The HTML <b> element specifies bold text, without any extra importance.	**Bold text**
`<strong>`	Important text	The HTML <strong> element describes text with strong importance. The content inside is usually displayed in bold.	**Important text**
`<i>`	Italic text	The HTML <i> element defines a part of text in an alternate voice or mood. The content inside is typically displayed in italic.	*Italic text*
`<em>`	Emphasized text	The HTML <em> element defines emphasized text. The content inside is typically displayed in italic.	*Emphasized text*
`<mark>`	Marked text	The HTML <mark> element defines text that should be marked or highlighted.	Marked text

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

<small>	Smaller text	The HTML <small> element defines smaller text.	Smaller text
<del>	Deleted text	The HTML <del> element defines text that has been deleted from a document. Browsers will usually strike a line through deleted text.	Deleted ~~text~~
<ins>	Inserted text	The HTML <ins> element defines a text that has been inserted into a document. Browsers will usually underline inserted text:	~~Inserted~~ text.
<sub>	Subscript text	The HTML <sub> element expresses subscript text. Subscript text appears half a character below the normal line and is sometimes rendered in a smaller font. Subscript text can be used for Chemistry, like H_2O.	Su$_b$script text
<sup>	Superscript text	The HTML <sup> element specifies superscript text. Superscript text appears half a character above the normal line and is sometimes rendered in a smaller font. Superscript text can be used for footnotes, as WWW [1]:	Su^{p}erscript text

HTML <blockquote> for Quotations

The HTML <blockquote> element identifies a section that is quoted from another source. Browsers typically indent <blockquote> elements, as can be seen below:



```
<!DOCTYPE html>
<html>
<body>

<p>Browsers typically indent blockquote elements.</p>

<blockquote cite="https://code4sp.eu/the-project/">
Code4SP's main objectives and priorities are in full interweaving with the
European Commission's goals, contributing towards providing tailored education and
training to digitally excluded groups, including migrants and young people from
disadvantaged backgrounds, while in parallel, taking into consideration the labor
market needs.
</blockquote>

</body>
</html>
```

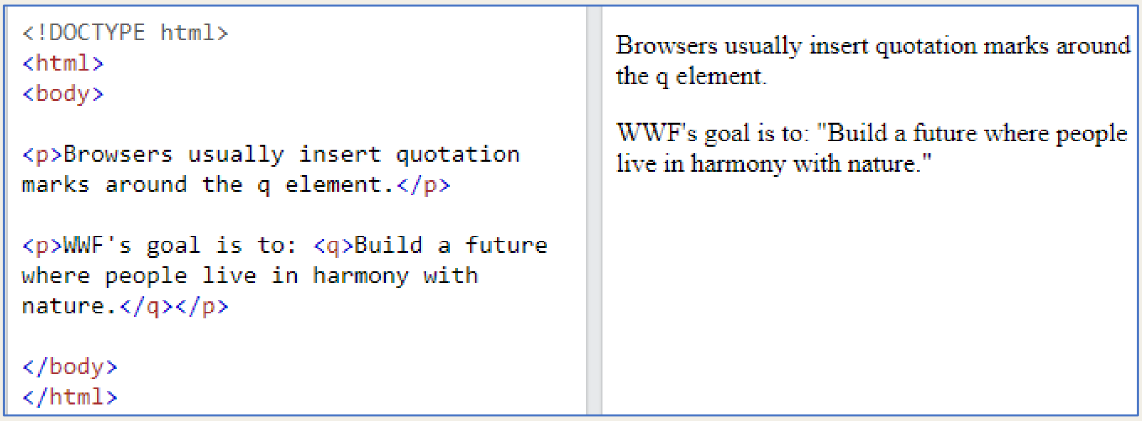
Browsers typically indent blockquote elements.

> Code4SP's main objectives and priorities are in full interweaving with the European Commission's goals, contributing towards providing tailored education and training to digitally excluded groups, including migrants and young people from disadvantaged backgrounds, while in parallel, taking into consideration the labor market needs.

***Figure 25** – The blockquote element (**Source:** Author)*

HTML <q> for Short Quotations

The HTML <q> tag specifies a short quotation. Browsers generally insert quotation marks around the quotation, as follows:



```
<!DOCTYPE html>
<html>
<body>

<p>Browsers usually insert quotation
marks around the q element.</p>

<p>WWF's goal is to: <q>Build a future
where people live in harmony with
nature.</q></p>

</body>
</html>
```

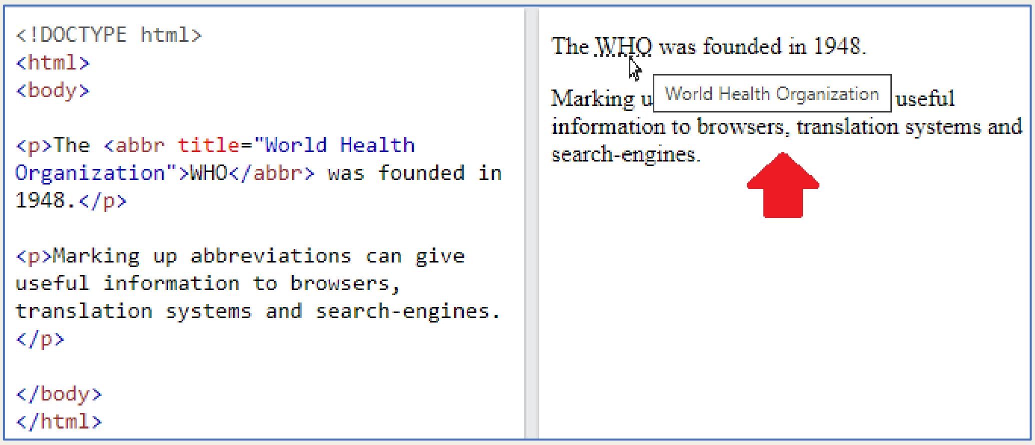
Browsers usually insert quotation marks around the q element.

WWF's goal is to: "Build a future where people live in harmony with nature."

***Figure 26** – The short quotation's element (**Source:** https://www.w3schools.com/html/html_quotation_elements.asp)*

HTML <abbr> for Abbreviations

The HTML <abbr> tag specifies an abbreviation or an acronym, like "HTML", "CSS", "Mr.", "Dr.", "ASAP", etc. Marking abbreviations can give valuable information to browsers, translation systems and search engines, as seen previously. In case one does not know the meaning of any given abbreviation, he/she could use the global title attribute to show the description for the abbreviation/acronym when mousing over the element, as seen below:



```
<!DOCTYPE html>
<html>
<body>

<p>The <abbr title="World Health
Organization">WHO</abbr> was founded in
1948.</p>

<p>Marking up abbreviations can give
useful information to browsers,
translation systems and search-engines.
</p>

</body>
</html>
```

The WHO was founded in 1948.

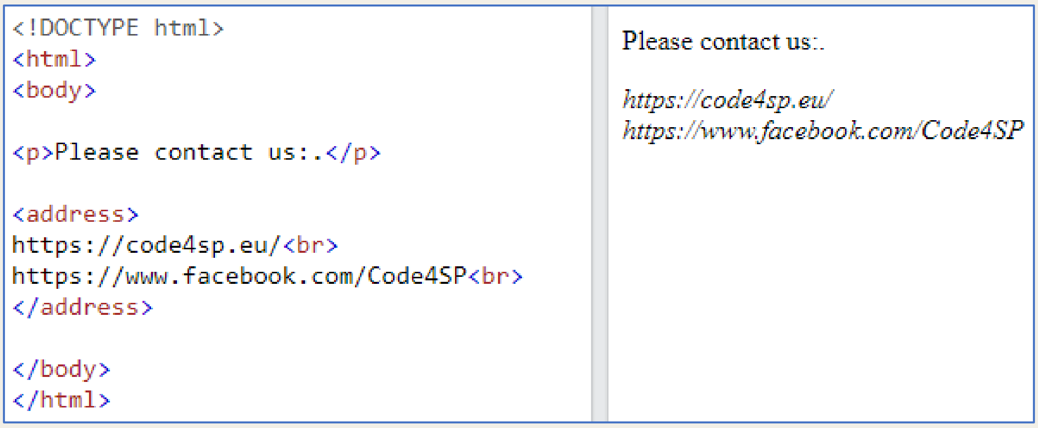
Marking u World Health Organization useful information to browsers, translation systems and search-engines.

Figure 27 – The <abbr> element and the mouse over function (Source: Author)

HTML <address> for Contact Information

The HTML <address> tag defines the contact information for the author/owner of a document or an article. It can be an email address, URL, physical address, phone number, etc. The text comprised in the <address> element is typically presented in italic, and browsers will always add a line break before and after it, as follows:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<!DOCTYPE html>
<html>
<body>

<p>Please contact us:.</p>

<address>
https://code4sp.eu/<br>
https://www.facebook.com/Code4SP<br>
</address>

</body>
</html>
```

Please contact us:.

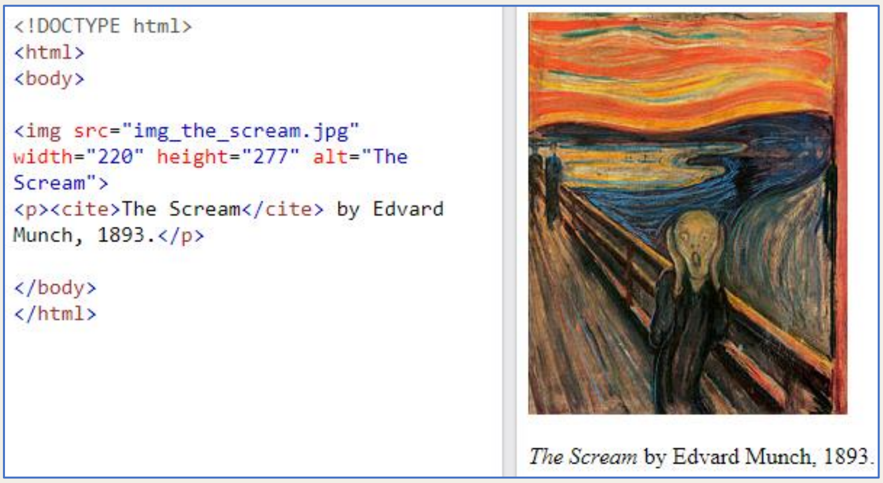
https://code4sp.eu/
https://www.facebook.com/Code4SP

***Figure 28** – The <address> element (**Source:** Author)*

HTML <cite> for Work Title

The HTML <cite> tag defines the title of a book, a poem, a song, a movie, a painting, and all creative works. It should be stated that the author's name is not the title of a work.

As in the aforementioned tags, the text in the <cite> element normally renders in italic:



```
<!DOCTYPE html>
<html>
<body>

<img src="img_the_scream.jpg"
width="220" height="277" alt="The
Scream">
<p><cite>The Scream</cite> by Edvard
Munch, 1893.</p>

</body>
</html>
```

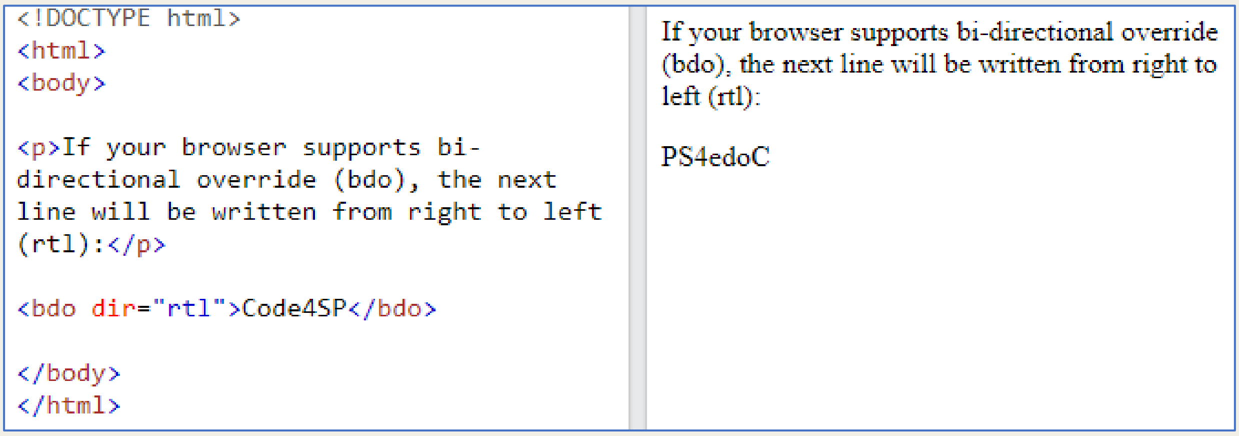
The Scream by Edvard Munch, 1893.

***Figure 29** – The <cite> element (**Source:** https://www.w3schools.com/html/html_quotation_elements.asp)*

HTML <bdo> for Bi-Directional Override

HTML <bdo> tag stands for "bidirectional override" which is used to override the current/default text direction. This tag sets the direction of content within it to execute on browser from left to right or right to left (*rtl* – right to left; *ltf* – left to right).

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<!DOCTYPE html>
<html>
<body>

<p>If your browser supports bi-
directional override (bdo), the next
line will be written from right to left
(rtl):</p>

<bdo dir="rtl">Code4SP</bdo>

</body>
</html>
```

If your browser supports bi-directional override (bdo), the next line will be written from right to left (rtl):

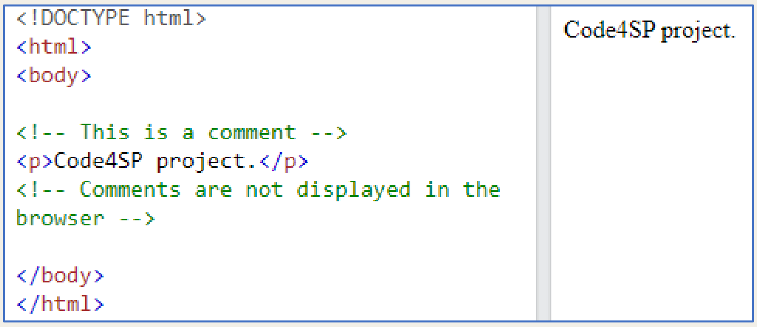
PS4edoC

__Figure 30__ – The <bdo> element (__Source:__ https://www.w3schools.com/html/html_quotation_elements.asp)

HTML Comments

Add Comments

This element is used to add a comment to an HTML document. An HTML comment begins with <!— and closes with —>. HTML comments are visible to anyone that views the page source code but are not rendered when the HTML document is rendered by a browser. It should be noted that there is an exclamation point in the start tag, but not in the end tag. This feature is especially useful for placing notifications and reminders in the HTML code:



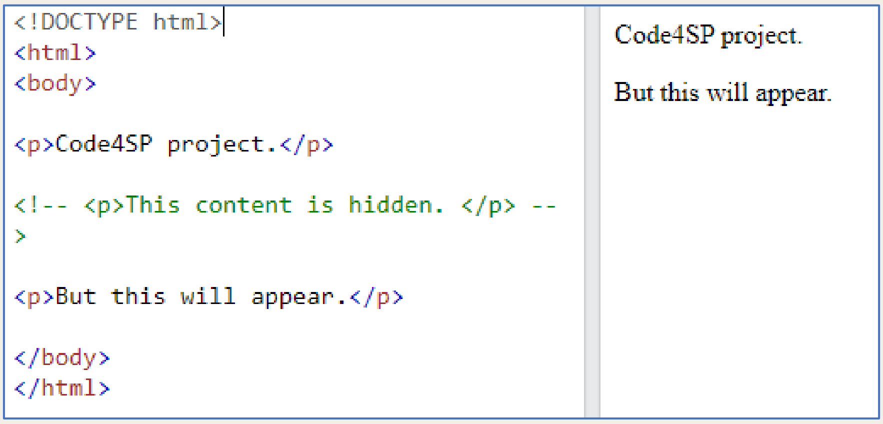
```
<!DOCTYPE html>
<html>
<body>

<!-- This is a comment -->
<p>Code4SP project.</p>
<!-- Comments are not displayed in the
browser -->

</body>
</html>
```

Code4SP project.

__Figure 31__ – The Comments feature (__Source:__ Author)

Hide Content

Comments can also be used to hide content, and that can be helpful if one hides it for the moment. You can also hide more than one line, everything between the <!-- and the --> will be hidden from the display. Comments are also great for debugging HTML, because one can comment out HTML lines of code, one at a time, to search for errors.



```
<!DOCTYPE html>
<html>
<body>

<p>Code4SP project.</p>

<!-- <p>This content is hidden. </p> --
>

<p>But this will appear.</p>

</body>
</html>
```

Code4SP project.

But this will appear.

__Figure 32__ – The hide content feature (__Source:__ Author)

HTML Colours

HTML colours are stipulated with predefined colour names, or with RGB, HEX, HSL, RGBA, or HSLA values.

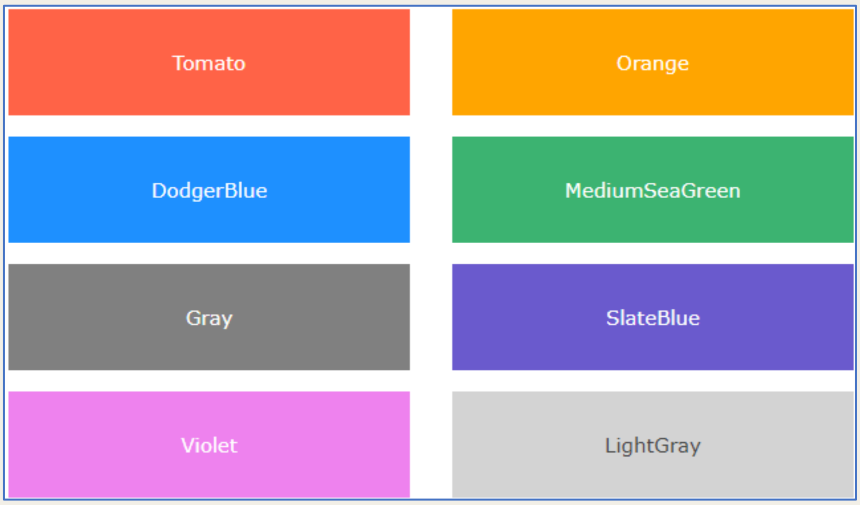
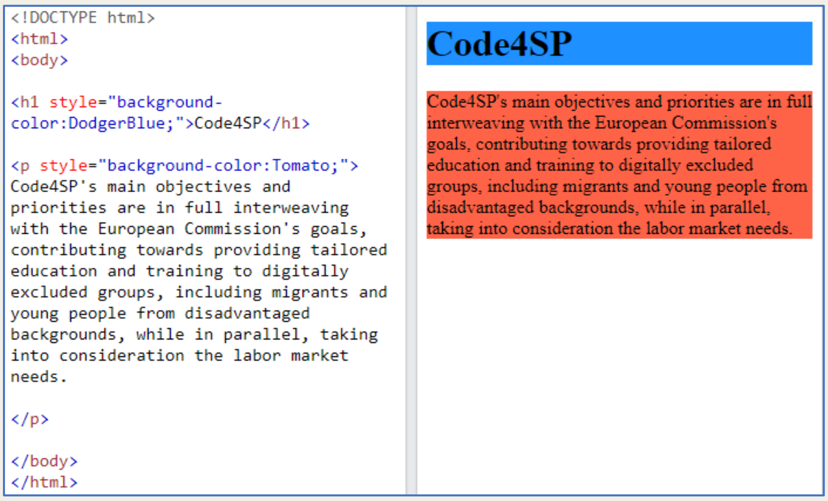


__Figure 33__ – Some colour names one can define (__Source:__ https://www.w3schools.com/html/html_colors.asp)

Colours can also be set for the **page background**:



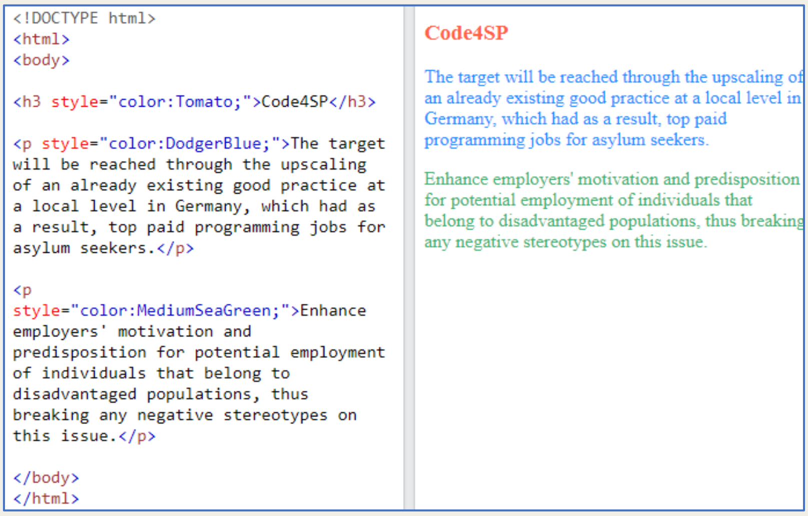
```
<!DOCTYPE html>
<html>
<body>

<h1 style="background-
color:DodgerBlue;">Code4SP</h1>

<p style="background-color:Tomato;">
Code4SP's main objectives and
priorities are in full interweaving
with the European Commission's goals,
contributing towards providing tailored
education and training to digitally
excluded groups, including migrants and
young people from disadvantaged
backgrounds, while in parallel, taking
into consideration the labor market
needs.

</p>

</body>
</html>
```

Code4SP

Code4SP's main objectives and priorities are in full interweaving with the European Commission's goals, contributing towards providing tailored education and training to digitally excluded groups, including migrants and young people from disadvantaged backgrounds, while in parallel, taking into consideration the labor market needs.

***Figure 34** – Defining a webpage's background colour (**Source:**Author)*

The same principle can be applied to the **text colour**:



```
<!DOCTYPE html>
<html>
<body>

<h3 style="color:Tomato;">Code4SP</h3>

<p style="color:DodgerBlue;">The target
will be reached through the upscaling
of an already existing good practice at
a local level in Germany, which had as
a result, top paid programming jobs for
asylum seekers.</p>

<p
style="color:MediumSeaGreen;">Enhance
employers' motivation and
predisposition for potential employment
of individuals that belong to
disadvantaged populations, thus
breaking any negative stereotypes on
this issue.</p>

</body>
</html>
```

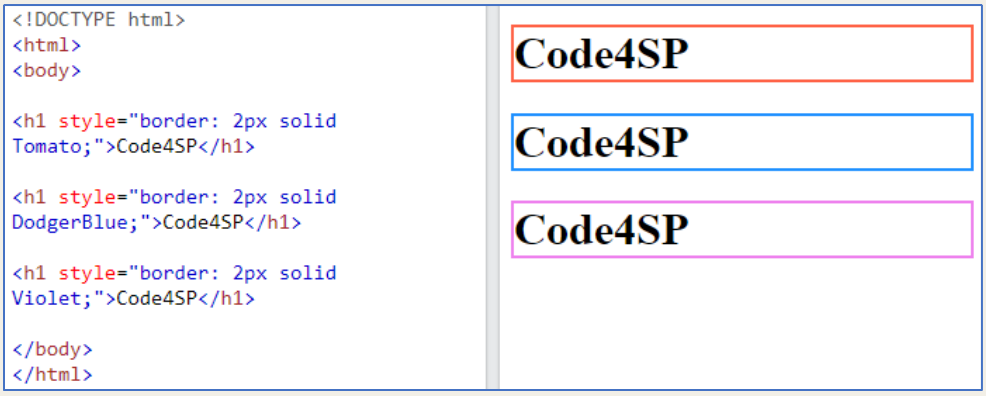
Code4SP

The target will be reached through the upscaling of an already existing good practice at a local level in Germany, which had as a result, top paid programming jobs for asylum seekers.

Enhance employers' motivation and predisposition for potential employment of individuals that belong to disadvantaged populations, thus breaking any negative stereotypes on this issue.

***Figure 35** – Defining the text colour (**Source:**Author)*

Also, for the colour of **borders**:



```
<!DOCTYPE html>
<html>
<body>

<h1 style="border: 2px solid
Tomato;">Code4SP</h1>

<h1 style="border: 2px solid
DodgerBlue;">Code4SP</h1>

<h1 style="border: 2px solid
Violet;">Code4SP</h1>

</body>
</html>
```

Code4SP

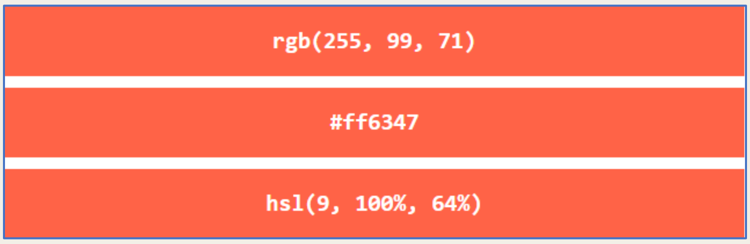
Code4SP

Code4SP

Figure 36 *– Adding borders and defining their colour (**Source:**Author)*

Colour Values

As stated above, HTML colours are stipulated with predefined colour names, or with RGB, HEX, HSL, RGBA, or HSLA values. The following three <div> elements have their background color set with RGB, HEX, and HSL values:



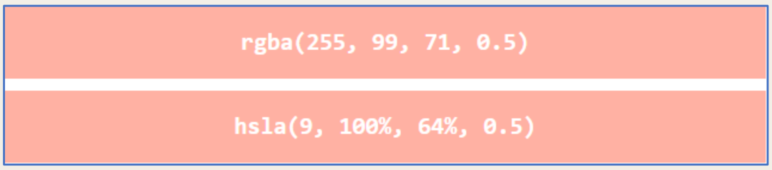
rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

Figure 37 *– RGB, HEX, and HSL values for the Tomato colour (**Source:** https://www.w3schools.com/html/html_colors.asp)*

Transparency is also a feature that could be added when defining a colour, by adding an Alpha channel to it. The following example as 50% transparency:

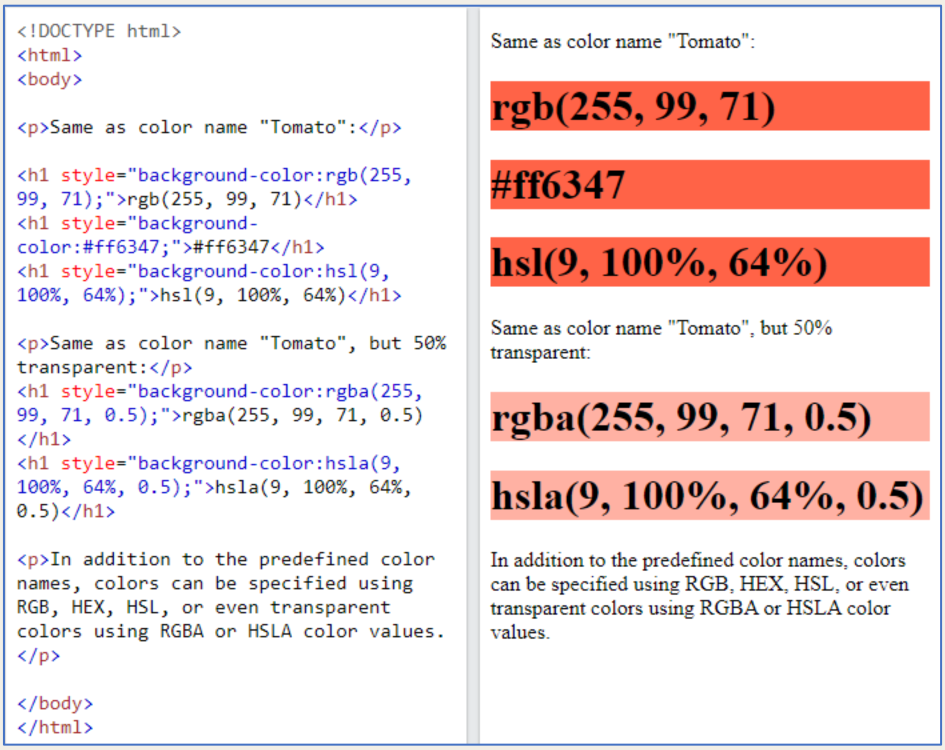


rgba(255, 99, 71, 0.5)

hsla(9, 100%, 64%, 0.5)

Figure 38 *– Setting transparency values for the Tomato colour (**Source:** https://www.w3schools.com/html/html_colors.asp)*

The code for setting up both features will ease learners' understanding. As seen, it comprises CSS features to be explored later:



```
<!DOCTYPE html>
<html>
<body>

<p>Same as color name "Tomato":</p>

<h1 style="background-color:rgb(255,
99, 71);">rgb(255, 99, 71)</h1>
<h1 style="background-
color:#ff6347;">#ff6347</h1>
<h1 style="background-color:hsl(9,
100%, 64%);">hsl(9, 100%, 64%)</h1>

<p>Same as color name "Tomato", but 50%
transparent:</p>
<h1 style="background-color:rgba(255,
99, 71, 0.5);">rgba(255, 99, 71, 0.5)
</h1>
<h1 style="background-color:hsla(9,
100%, 64%, 0.5);">hsla(9, 100%, 64%,
0.5)</h1>

<p>In addition to the predefined color
names, colors can be specified using
RGB, HEX, HSL, or even transparent
colors using RGBA or HSLA color values.
</p>

</body>
</html>
```

Same as color name "Tomato":

rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

Same as color name "Tomato", but 50% transparent:

rgba(255, 99, 71, 0.5)

hsla(9, 100%, 64%, 0.5)

In addition to the predefined color names, colors can be specified using RGB, HEX, HSL, or even transparent colors using RGBA or HSLA color values.

***Figure 39** – Setting colour and transparency of the Tomato colour (**Source:** https://www.w3schools.com/html/html_colors.asp)*

More information concerning RGB, HEX, HSL, RGBA, or HSLA values can be found at https://www.w3schools.com/html/html_colors.asp

HTML Links

Links can be found in nearly all webpages. They allow internet users to navigate from page to page. It does not have to be text, as it can be an image or any other HTML element.

spel CITIZENS IN POWER Center for Social Innovation CDDR ZAUG gGmbH 4Action social hackers academy EPE

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Hyperlinks

HTML links are hyperlinks, which can be clicked, jumping to another document. When the mouse is moved over a link, the mouse arrow will turn into a little hand.

Syntax

The HTML <a> tag defines a hyperlink. It has the following syntax:

```
<a href="url">link text</a>
```

The most significant attribute of the <a> element is the href attribute, which indicates the link's destination. The link text is the part that will be visible to the user. By clicking on the link text, the reader will be redirected to the required URL address.

By default, links will be seen in all browsers as follows:

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

Note: Link colours can be changed by using CSS features.

The target attribute

By default, the linked page will be shown in the current browser window. To modify this, learners must indicate another target for the link.

The target attribute indicates where to open the linked document. It can have one of the following values:

- _self - Default. Opens the document in the same window/tab as it was clicked
- _blank - Opens the document in a new window or tab
- _parent - Opens the document in the parent frame

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- _top - Opens the document in the full body of the window

Using an image as a Link

To use an image as a link, just put the <img> tag inside the <a> tag, as it can be checked over the following tutorial: https://www.w3schools.com/html/tryit.asp?filename=tryhtml_links_image

Linking an email address

To create a link that opens the user's email software (letting them send a new email), mailto: should be added inside the href attribute, following the next example:

.

```
<a href="mailto:someone@example.com">Send email</a>
```

Create a Bookmark in HTML

HTML links can be applied to make bookmarks, so that readers can jump to particular parts of a web page, and it can be truly useful if the web page is very long. This process is composed of two very simple steps:

- To create a bookmark - first the bookmark should be created, and then a link should be added to it. To create, the id attribute should be used (e.g.: `<h2 id="C1">Chapter 1</h2>),` then, a link to the bookmark, from within the same page, should be added (e.g.: `<a href="#C4">Jump to Chapter 4</a>)`
- When the link is clicked, the page will scroll to the location with the bookmark.

HTML Images

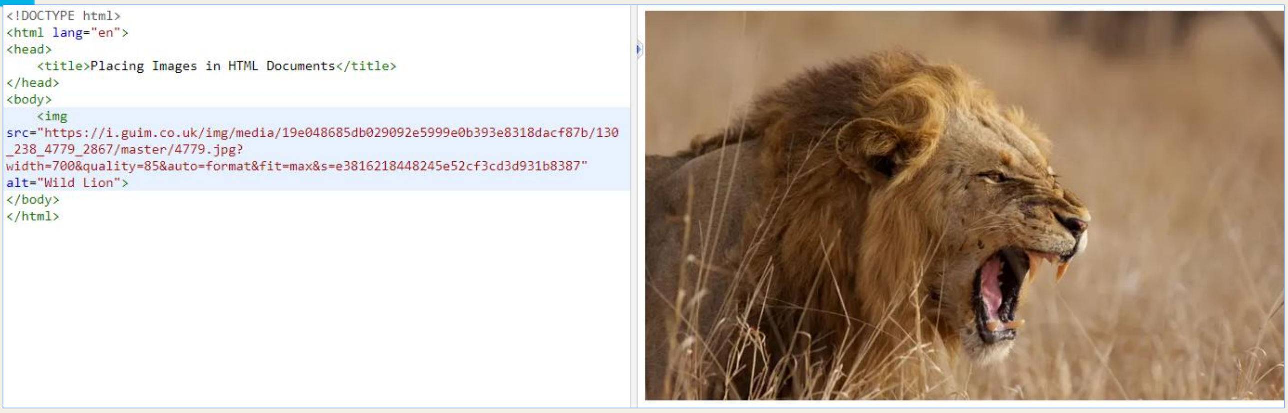
Inserting images into webpages

Images improve visual look of the web pages by making them more appealing and colourful. The <img> tag is used to add images in the HTML pages. It is an empty element and contains attributes only.

Each image must have at least two attributes: the src and alt attributes. The src attribute informs the browser where to find the image, being its value the URL of the image file. The alt attribute provides an alternative text for the image if it is inaccessible or cannot be displayed for some reason (slow connection, image is not available at the specified URL, or if the user uses a screen reader or non-graphical browser). Its value should be a meaningful substitute for the image, preferably a suggestive text.

Images improve visual look of the web pages by making them more appealing and colourful. The <img> tag is used to add images in the HTML pages. It is an empty element and contains attributes only.

Each image must have at least two attributes: the src and alt attributes. The src attribute informs the browser where to find the image, being its value the URL of the image file. The alt attribute provides an alternative text for the image if it is inaccessible or cannot be displayed for some reason (slow connection, image is not available at the specified URL, or if the user uses a screen reader or non-graphical browser). Its value should be a meaningful substitute for the image, preferably a suggestive text.

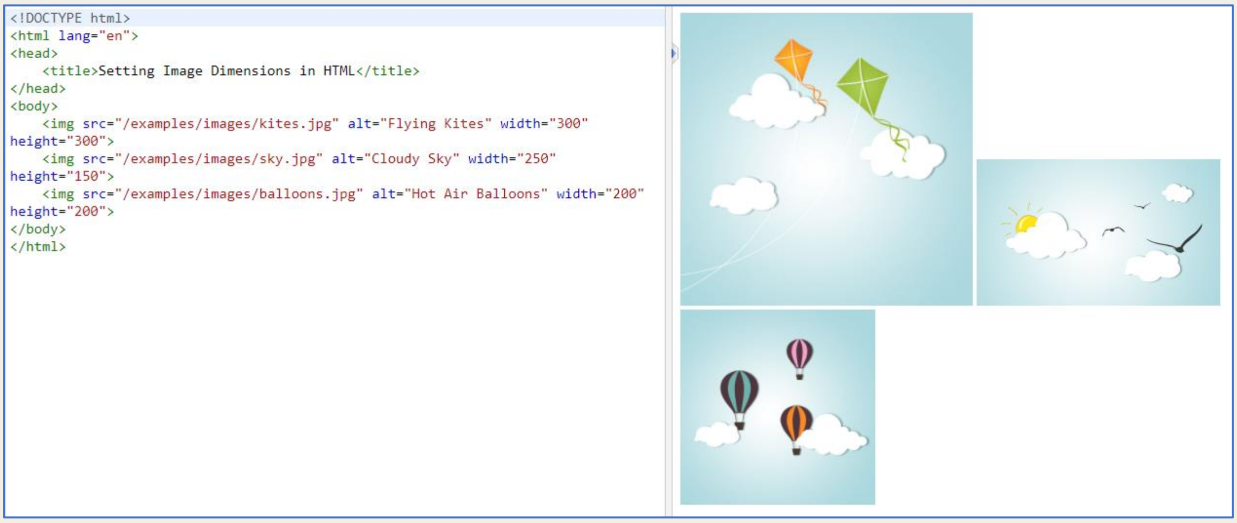


```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Placing Images in HTML Documents</title>
</head>
<body>
    <img
src="https://i.guim.co.uk/img/media/19e048685db029092e5999e0b393e8318dacf87b/130
_238_4779_2867/master/4779.jpg?
width=700&quality=85&auto=format&fit=max&s=e3816218448245e52cf3cd3d931b8387"
alt="Wild Lion">
</body>
</html>
```

Figure 40 *– Adding an image to a HTML document, using the src and alt attributes (**Source:**Author)*

Setting the Width and Height of an Image

The width and height attributes are used to indicate the width and height of an image. The values of these attributes are interpreted in pixels by default. It's a good practice to specify both the width and height attributes, so that browser can assign that much of space for the image before it is transferred.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Setting Image Dimensions in HTML</title>
</head>
<body>
    <img src="/examples/images/kites.jpg" alt="Flying Kites" width="300"
height="300">
    <img src="/examples/images/sky.jpg" alt="Cloudy Sky" width="250"
height="150">
    <img src="/examples/images/balloons.jpg" alt="Hot Air Balloons" width="200"
height="200">
</body>
</html>
```

Figure 41 *– Setting the height and width of an image (**Source:***
https://www.tutorialrepublic.com/codelab.php?topic=html&file=specify-dimensions-for-images)

The style attribute can also be used to indicate width and height. It prevents style sheets from changing the image size by accident, because inline style has the highest priority.

Using the HTML5 Picture Element

Now and then, scaling an image up or down to fit different devices (or screen sizes) does not work as expected. In addition, reducing the image dimension using the width and height attribute does not decrease the initial file size. In order to solve these problems, HTML5 has introduced the <picture> tag that allows to define multiple versions of an image to target different types of devices.

The <picture> element contains zero or more <source> elements, each referring to different image source, and one <img> element at the end. Likewise, each <source> element has the media attribute which specifies a media condition that is used by the browser to determine when a particular source should be used.

Image Maps

An image map allows one to define hotspots on an image that acts out just like a hyperlink. The key idea behind creating an image map is to give an simple way of linking various parts of an image without dividing it into separate image files. For example, a map of a country may have each city hyperlinked to more information about that city.

The following example is pretty accurate about these features:
https://www.tutorialrepublic.com/codelab.php?topic=html&file=image-maps

The name attribute of the <map> tag is used to reference the map from the <img> tag using its usemap attribute. The <area> tag is used inside the <map> element to define the clickable areas on an image. Any number of clickable areas can be defined within an image.

HTML Favicon

A favicon is a small image displayed to the left of the page title in the browser tab:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

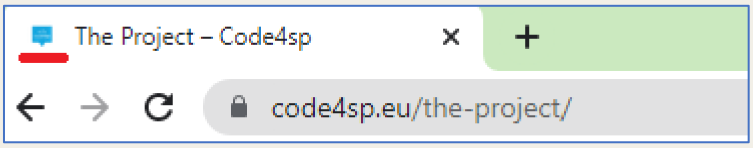


*Figure 42 – Code4SP's favoticon (**Source:** Author)*

To add a favicon to a website, a favicon image should be saved to the root directory of webserver. Other way is by creating a folder in the root directory called images, then saving the favicon image in this folder. A common name for a favicon image is "favicon.ico".

Next, a <link> element should be added to the "index.html" file, after the <title> element, as follows:

```
<!DOCTYPE html>
<html>
<head>
  <title>Code4SP</title>
  <link rel="icon" type="image/x-icon" href="/images/favicon.ico">
</head>
<body>
<h1>Our project</h1>
<p>Co-funded by the E+ fund of the EC.</p>
</body>
</html>
```

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Creating Tables in HTML

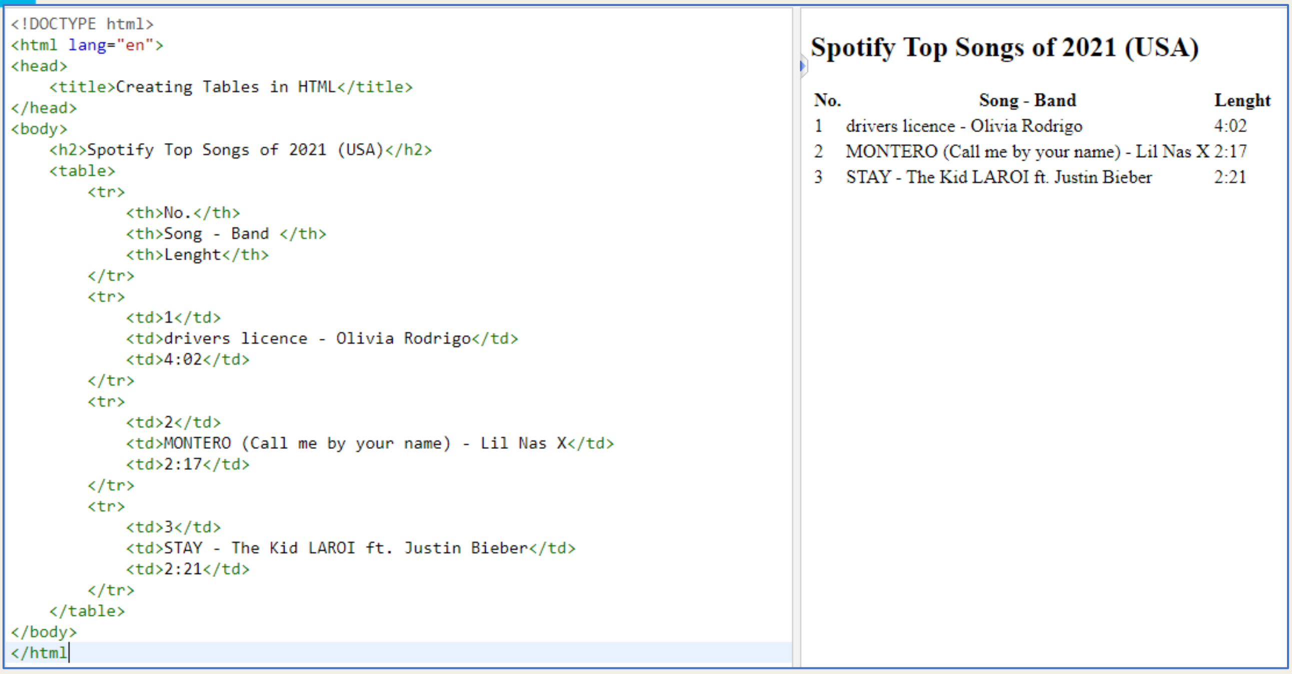
HTML tables allow to arrange data into rows and columns. They are generally used to display tabular data like product listings, customer's details, financial reports, etc.

A table can be created using the <table> element. Inside the <table> element, the <tr> elements can be utilized to create rows, and to create columns inside a row the <td> elements can be used. A cell can be defined as a header for a group of table cells using the <th> element.

Tables do not have any borders by default. The CSS border property can be used to add borders to the tables. Furthermore, table cells are sized just large enough to fit the contents by default. To add more space around the content in the table cells, the CSS padding property can be used.

By default, borders around the table and their cells are separated from each other. But they can be collapsed into one by using the border-collapse property on the <table> element. Additionally, text inside the <th> elements is displayed in bold font, aligned horizontally center in the cell by default. To change the default alignment, the CSS text-align property can be used. To do so, it is suggested that learners reach the CSS topic first. Most of the <table> element's attribute such as border, cellpadding, cellspacing, width, align, etc. for styling table appearances in earlier versions has been dropped in HTML5, so they should be avoided. **CSS should be privileged to style HTML tables.**

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Creating Tables in HTML</title>
</head>
<body>
    <h2>Spotify Top Songs of 2021 (USA)</h2>
    <table>
        <tr>
            <th>No.</th>
            <th>Song - Band </th>
            <th>Lenght</th>
        </tr>
        <tr>
            <td>1</td>
            <td>drivers licence - Olivia Rodrigo</td>
            <td>4:02</td>
        </tr>
        <tr>
            <td>2</td>
            <td>MONTERO (Call me by your name) - Lil Nas X</td>
            <td>2:17</td>
        </tr>
        <tr>
            <td>3</td>
            <td>STAY - The Kid LAROI ft. Justin Bieber</td>
            <td>2:21</td>
        </tr>
    </table>
</body>
</html>
```

Spotify Top Songs of 2021 (USA)

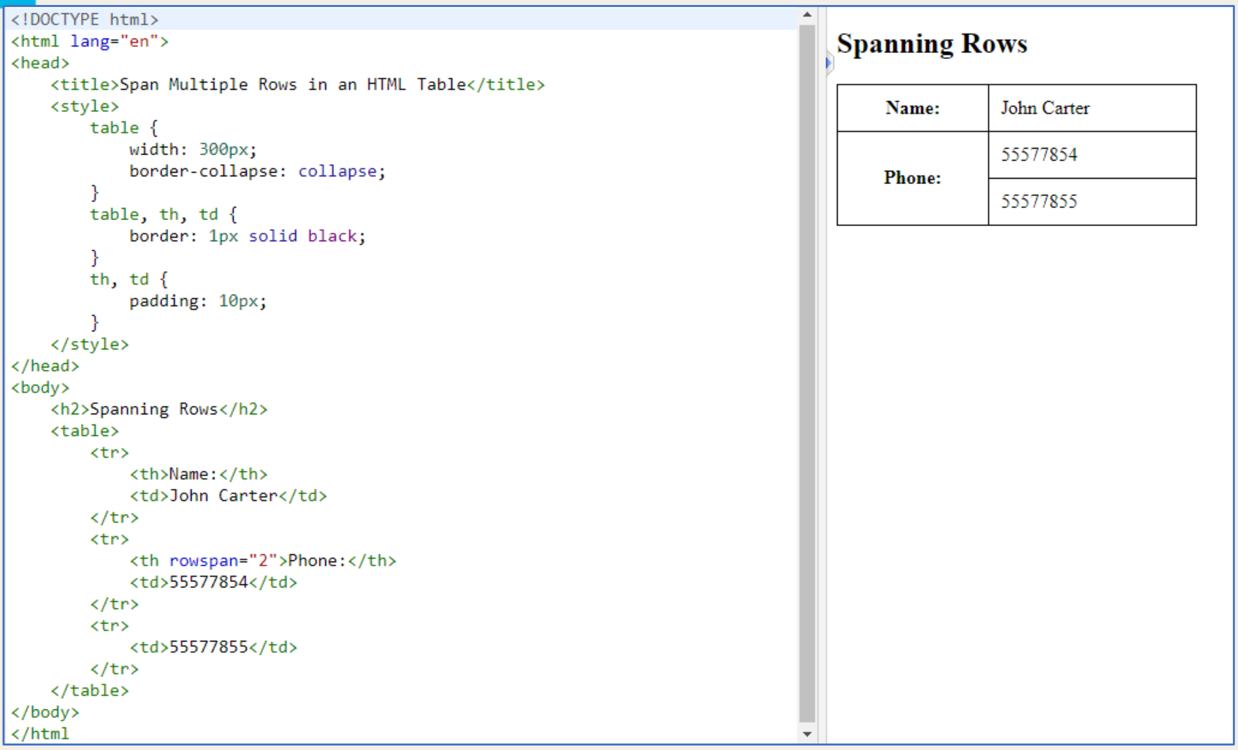
No.	Song - Band	Lenght
1	drivers licence - Olivia Rodrigo	4:02
2	MONTERO (Call me by your name) - Lil Nas X	2:17
3	STAY - The Kid LAROI ft. Justin Bieber	2:21

***Figure 43** – A basic table (**Source:** Author)*

Spanning Multiple Rows and Columns

Spanning allows to extend table rows and columns across multiple other rows and columns. Usually, a table cell cannot pass over into the space below or above another table cell. However, the rowspan or colspan attributes can be used to span multiple rows or columns in a table.

Likewise, the rowspan attribute can be used to create a cell that spans more than one row, as follows:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Span Multiple Rows in an HTML Table</title>
    <style>
        table {
            width: 300px;
            border-collapse: collapse;
        }
        table, th, td {
            border: 1px solid black;
        }
        th, td {
            padding: 10px;
        }
    </style>
</head>
<body>
    <h2>Spanning Rows</h2>
    <table>
        <tr>
            <th>Name:</th>
            <td>John Carter</td>
        </tr>
        <tr>
            <th rowspan="2">Phone:</th>
            <td>55577854</td>
        </tr>
        <tr>
            <td>55577855</td>
        </tr>
    </table>
</body>
</html
```

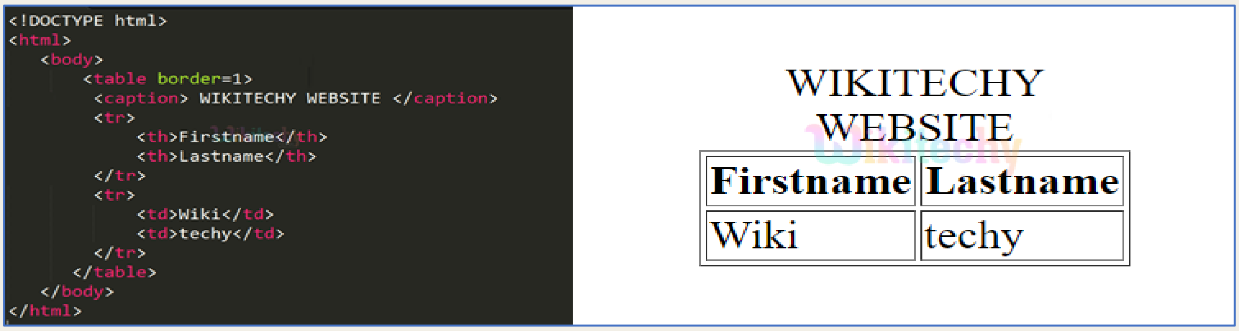
Spanning Rows

Name:	John Carter
Phone:	55577854
	55577855

Figure 44 – The rowspan attribute (Source: https://www.tutorialrepublic.com/html-tutorial/html-tables.php)

Table captions

A caption (or title) for tables can be created by using the <caption> element. This element should be placed directly after the (opening) <table> tag. By default, caption appears at the top of the table, but this can be changed by using the CSS caption-side property.



```
<!DOCTYPE html>
<html>
   <body>
       <table border=1>
        <caption> WIKITECHY WEBSITE </caption>
        <tr>
            <th>Firstname</th>
            <th>Lastname</th>
        </tr>
        <tr>
            <td>Wiki</td>
            <td>techy</td>
        </tr>
      </table>
   </body>
</html>
```

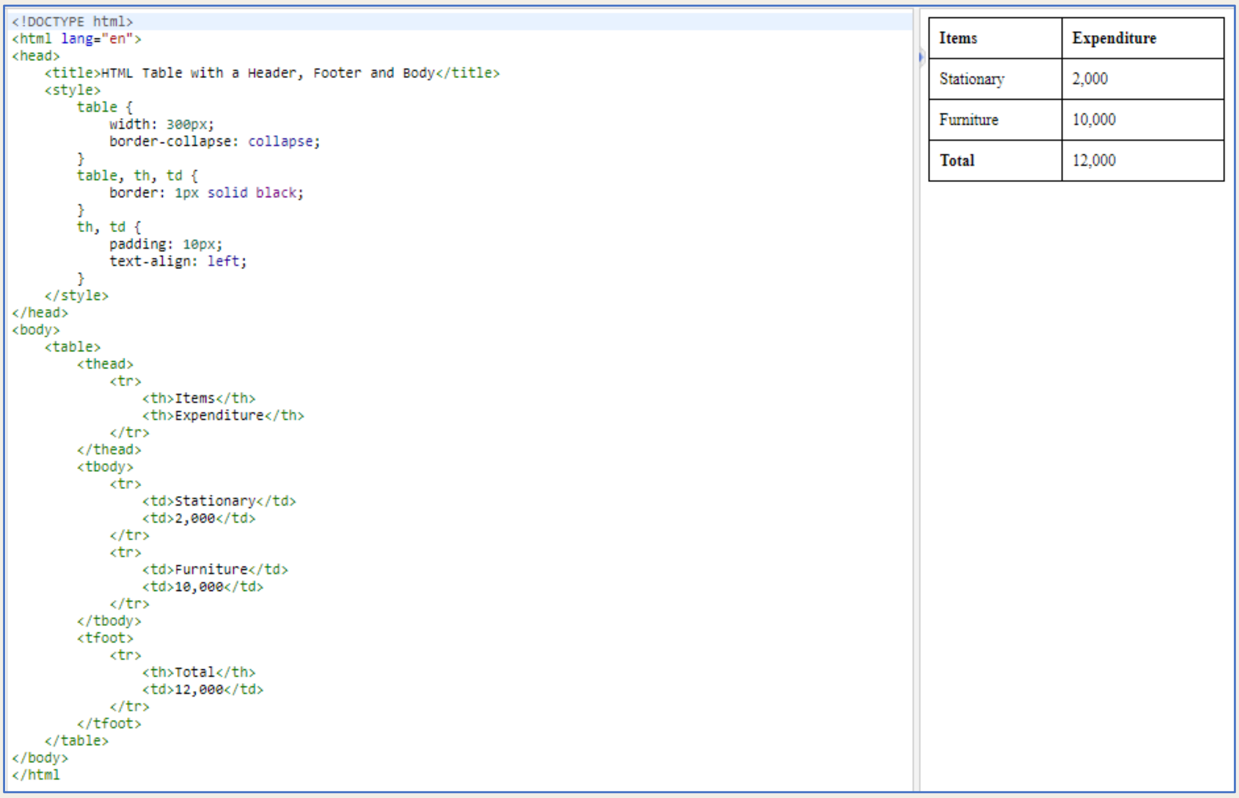
WIKITECHY WEBSITE

Firstname	Lastname
Wiki	techy

Figure 45 – Adding table captions (Source: https://www.wikitechy.com)

Defining a Table Header, Body, and Footer

HTML provides a series of tags <thead>, <tbody>, and <tfoot> that aid learners to create more coordinated tables, by defining header, body and footer regions, in that order.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML Table with a Header, Footer and Body</title>
    <style>
        table {
            width: 300px;
            border-collapse: collapse;
        }
        table, th, td {
            border: 1px solid black;
        }
        th, td {
            padding: 10px;
            text-align: left;
        }
    </style>
</head>
<body>
    <table>
        <thead>
            <tr>
                <th>Items</th>
                <th>Expenditure</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Stationary</td>
                <td>2,000</td>
            </tr>
            <tr>
                <td>Furniture</td>
                <td>10,000</td>
            </tr>
        </tbody>
        <tfoot>
            <tr>
                <th>Total</th>
                <td>12,000</td>
            </tr>
        </tfoot>
    </table>
</body>
</html
```

Items	**Expenditure**
Stationary	2,000
Furniture	10,000
Total	12,000

***Figure 46** – Adding table captions (**Source:** https://www.tutorialrepublic.com/html-tutorial/html-tables.php)*

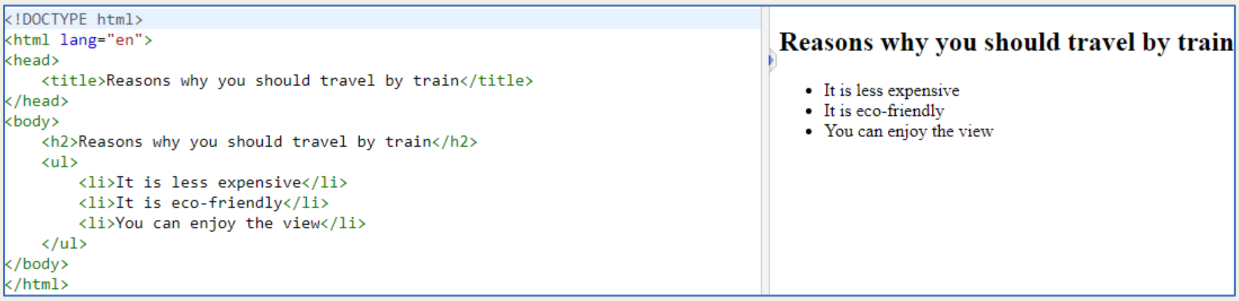
HTML Lists

HTML lists are applied to present information in a well-formed and semantic way. Inside of them, one can add text, images, links, line breaks, etc. There are three different types of lists in HTML and each one has a specific purpose and meaning:

- **A) Unordered lists** — Used to create a list of related items, in no particular order.
- **B) Ordered lists** — Used to create a list of related items, in a certain order.
- **C) Description lists** — Used to create a list of terms and their descriptions.

A) Unordered lists

An unordered list created using the <ul> element, and each list item begins with the <li> element. It is marked with bullets, as follows:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Reasons why you should travel by train</title>
</head>
<body>
    <h2>Reasons why you should travel by train</h2>
    <ul>
        <li>It is less expensive</li>
        <li>It is eco-friendly</li>
        <li>You can enjoy the view</li>
    </ul>
</body>
</html>
```

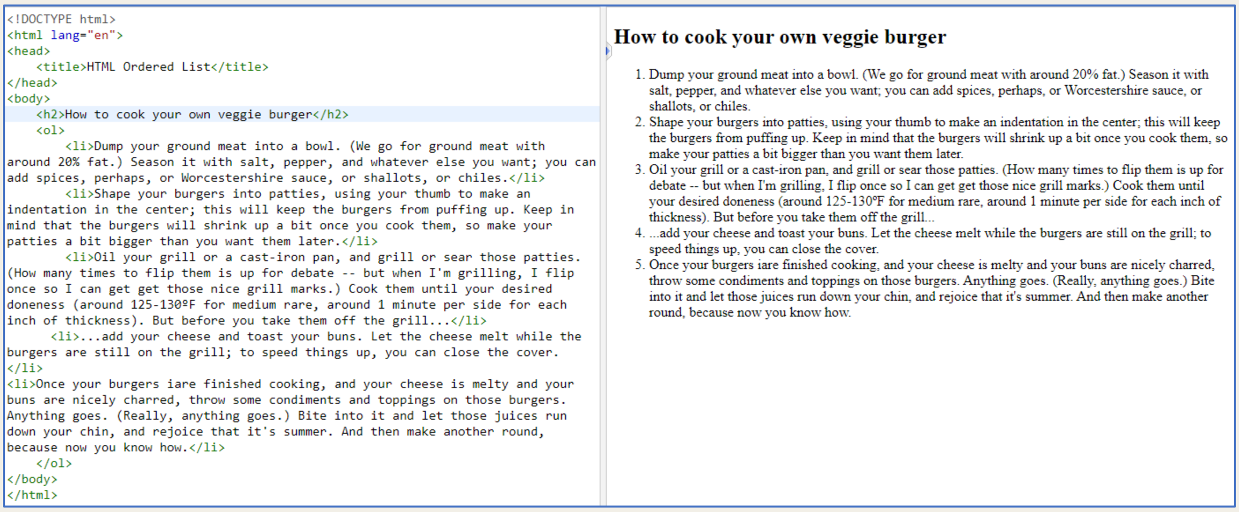
Reasons why you should travel by train

- It is less expensive
- It is eco-friendly
- You can enjoy the view

Figure 47 *– Unordered Lists (**Source:** Author)*

B) Ordered lists

An ordered list is created by using the <ol> element, and each list item starts with the <li> element. Ordered lists are used when the order of the list's items is critical. It is marked with numbers, as follows:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML Ordered List</title>
</head>
<body>
    <h2>How to cook your own veggie burger</h2>
    <ol>
        <li>Dump your ground meat into a bowl. (We go for ground meat with
around 20% fat.) Season it with salt, pepper, and whatever else you want; you can
add spices, perhaps, or Worcestershire sauce, or shallots, or chiles.</li>
        <li>Shape your burgers into patties, using your thumb to make an
indentation in the center; this will keep the burgers from puffing up. Keep in
mind that the burgers will shrink up a bit once you cook them, so make your
patties a bit bigger than you want them later.</li>
        <li>Oil your grill or a cast-iron pan, and grill or sear those patties.
(How many times to flip them is up for debate -- but when I'm grilling, I flip
once so I can get get those nice grill marks.) Cook them until your desired
doneness (around 125-130ºF for medium rare, around 1 minute per side for each
inch of thickness). But before you take them off the grill...</li>
      <li>...add your cheese and toast your buns. Let the cheese melt while the
burgers are still on the grill; to speed things up, you can close the cover.
</li>
<li>Once your burgers iare finished cooking, and your cheese is melty and your
buns are nicely charred, throw some condiments and toppings on those burgers.
Anything goes. (Really, anything goes.) Bite into it and let those juices run
down your chin, and rejoice that it's summer. And then make another round,
because now you know how.</li>
    </ol>
</body>
</html>
```

How to cook your own veggie burger

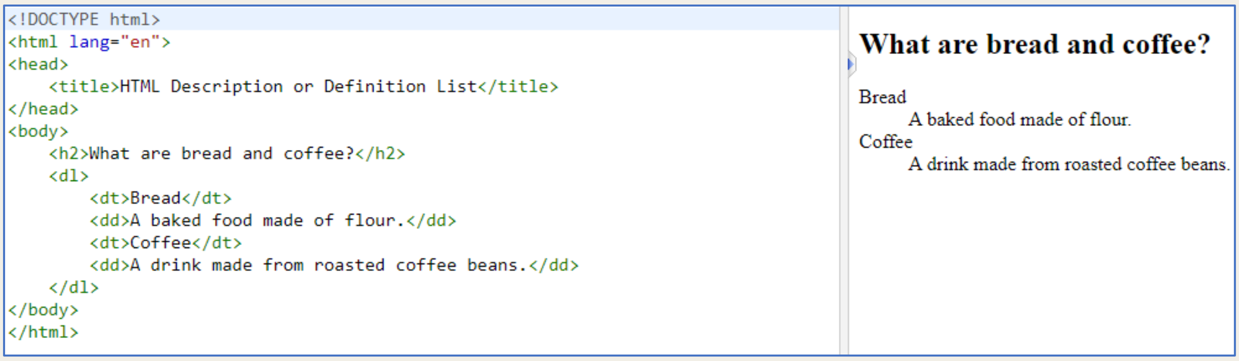
1. Dump your ground meat into a bowl. (We go for ground meat with around 20% fat.) Season it with salt, pepper, and whatever else you want; you can add spices, perhaps, or Worcestershire sauce, or shallots, or chiles.
2. Shape your burgers into patties, using your thumb to make an indentation in the center; this will keep the burgers from puffing up. Keep in mind that the burgers will shrink up a bit once you cook them, so make your patties a bit bigger than you want them later.
3. Oil your grill or a cast-iron pan, and grill or sear those patties. (How many times to flip them is up for debate -- but when I'm grilling, I flip once so I can get get those nice grill marks.) Cook them until your desired doneness (around 125-130ºF for medium rare, around 1 minute per side for each inch of thickness). But before you take them off the grill...
4. ...add your cheese and toast your buns. Let the cheese melt while the burgers are still on the grill; to speed things up, you can close the cover.
5. Once your burgers iare finished cooking, and your cheese is melty and your buns are nicely charred, throw some condiments and toppings on those burgers. Anything goes. (Really, anything goes.) Bite into it and let those juices run down your chin, and rejoice that it's summer. And then make another round, because now you know how.

Figure 48 – Ordered Lists (Source: Author)

C) Description Lists

A description list is a list of items with a description or definition of each item. The description list is created using <dl> element. The <dl> element is used in conjunction with the <dt> element which specify a term, and the <dd> element which specify the term's definition.

Browsers usually render the definition lists by placing the terms and definitions in separate lines, where the term's definitions are slightly indented.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML Description or Definition List</title>
</head>
<body>
    <h2>What are bread and coffee?</h2>
    <dl>
        <dt>Bread</dt>
        <dd>A baked food made of flour.</dd>
        <dt>Coffee</dt>
        <dd>A drink made from roasted coffee beans.</dd>
    </dl>
</body>
</html>
```

What are bread and coffee?

Bread
    A baked food made of flour.
Coffee
    A drink made from roasted coffee beans.

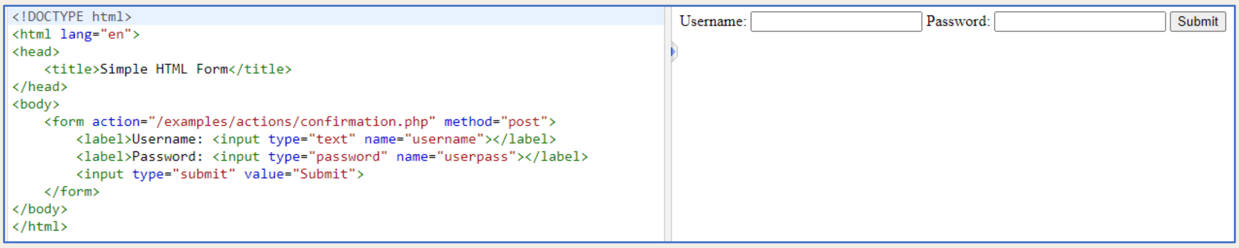
Figure 49 – Description Lists (Source: Author)

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

HTML Forms

HTML Forms are expected to collect different types of user inputs, such as contact details (name, email, phone number, bank account, etc.). Forms include special elements known as controls like input box, checkboxes, radio-buttons, submit buttons, etc. To this end, users fill in a form with this data, either via text or box selection, submitting it later to a webserver for processing this data.

The <form> tag is used to generate an HTML form. A simple example of a login form would be as follows:

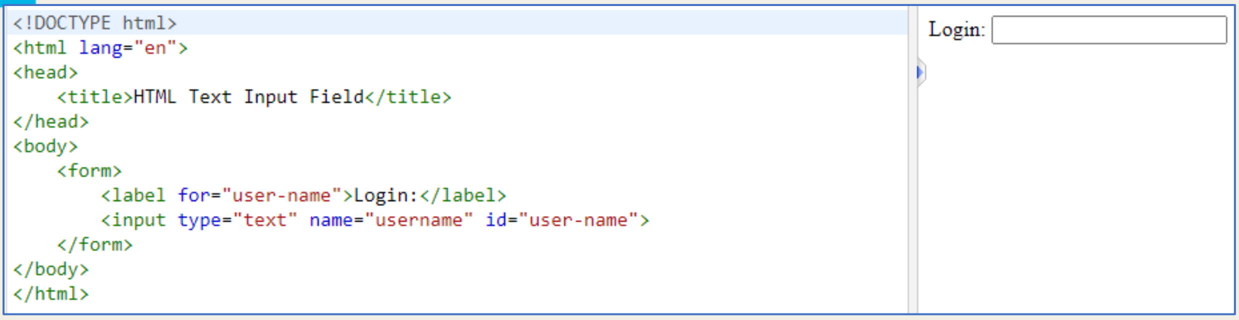


```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Simple HTML Form</title>
</head>
<body>
    <form action="/examples/actions/confirmation.php" method="post">
        <label>Username: <input type="text" name="username"></label>
        <label>Password: <input type="password" name="userpass"></label>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

***Figure 50** – Example of a login form (**Source:** https://www.tutorialrepublic.com/html-tutorial/html-forms.php)*

There are different types of controls to be applied in a HTML form, being **input elements** the most frequently used ones. They identify various types of user input fields, depending on the type attribute. Input elements can be **text fields, password fields, checkboxes, submit buttons, reset buttons, file select boxes**, as well as several new input types introduced in HTML5 (they can be checked out here).

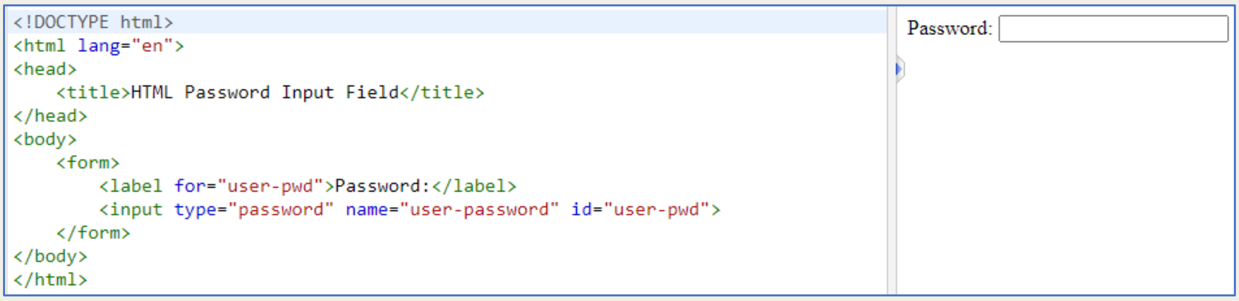
Text fields are areas that let users add text. These are created by using an <input> element, whose type attribute has a value of text. It should be noted that the <label> tag is used to identify the labels for <input> elements. If the webmaster wants his/her users to enter several lines, <textarea> should be added instead.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML Text Input Field</title>
</head>
<body>
    <form>
        <label for="user-name">Login:</label>
        <input type="text" name="username" id="user-name">
    </form>
</body>
</html>
```

Figure 51 – Text field example (Source: Author)

Password fields are like text fields, being the only difference characters in a password field are hidden, so they are displayed as asterisks or dots. Likewise, this procedure prevents someone else from reading the password on the screen. This is as well a single-line text input control generated using an <input> element whose type attribute has a value of password.

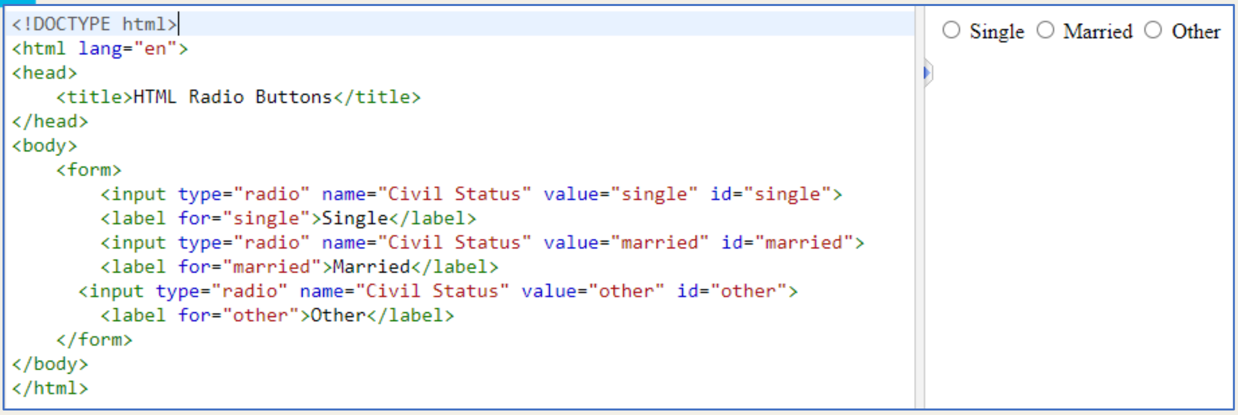


```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML Password Input Field</title>
</head>
<body>
    <form>
        <label for="user-pwd">Password:</label>
        <input type="password" name="user-password" id="user-pwd">
    </form>
</body>
</html>
```

Figure 52 – Password field example (Source: https://www.tutorialrepublic.com/html-tutorial/html-forms.php)

Radio buttons are applied to allow the user select exactly one option from a pre-specified set of options. It is generated using an <input> element whose type attribute has a value of radio.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

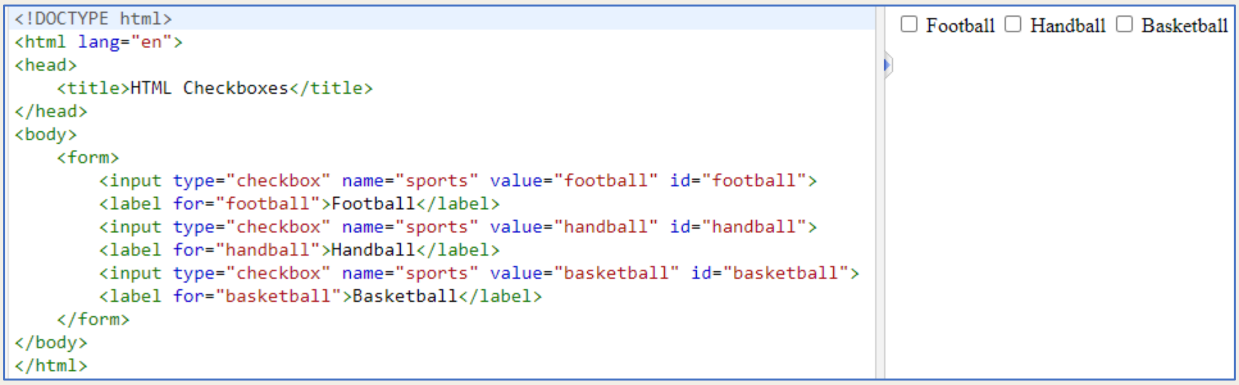


```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML Radio Buttons</title>
</head>
<body>
    <form>
        <input type="radio" name="Civil Status" value="single" id="single">
        <label for="single">Single</label>
        <input type="radio" name="Civil Status" value="married" id="married">
        <label for="married">Married</label>
      <input type="radio" name="Civil Status" value="other" id="other">
        <label for="other">Other</label>
    </form>
</body>
</html>
```

○ Single ○ Married ○ Other

__Figure 53__ – Code for setting up radio buttons (__Source:__ Author)

Checkboxes make available to the user one or more choices from a pre-defined set of options. It is generated using an <input> element whose type attribute has a value of checkbox. If one prefers to generate a checkbox (or radio button) selected by default, one simply must add the attribute checked to the input element (`<input type="checkbox" checked>`).



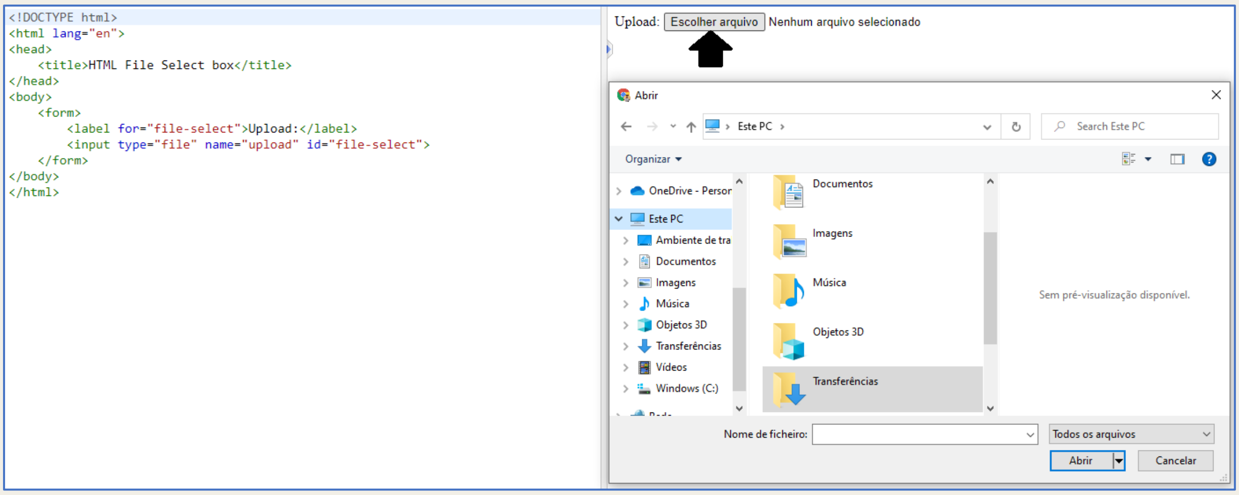
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML Checkboxes</title>
</head>
<body>
    <form>
        <input type="checkbox" name="sports" value="football" id="football">
        <label for="football">Football</label>
        <input type="checkbox" name="sports" value="handball" id="handball">
        <label for="handball">Handball</label>
        <input type="checkbox" name="sports" value="basketball" id="basketball">
        <label for="basketball">Basketball</label>
    </form>
</body>
</html>
```

□ Football □ Handball □ Basketball

__Figure 54__ – Code for setting up checkboxes (__Source:__ Author)

File select boxes allow a user to browse for a local file and send it as an attachment with the form data. E.g., Google Chrome provides a file select input field with a ‘Browse’ button that permits the user to select a file from his/her hard drive.

File select boxes are also created using an <input> element, whose type attribute value is set to file.

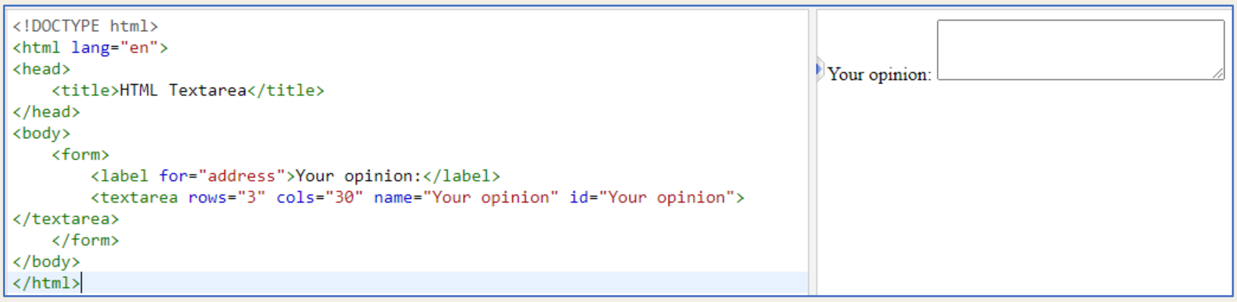


```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML File Select box</title>
</head>
<body>
    <form>
        <label for="file-select">Upload:</label>
        <input type="file" name="upload" id="file-select">
    </form>
</body>
</html>
```

Upload: Escolher arquivo Nenhum arquivo selecionado

*Figure 55 – Code for setting up file select boxes (**Source:** Author)*

Textarea can be defined as a multiple-line text input control that allows to enter more than one line of text. These controls are created using an <textarea> element, as follows:

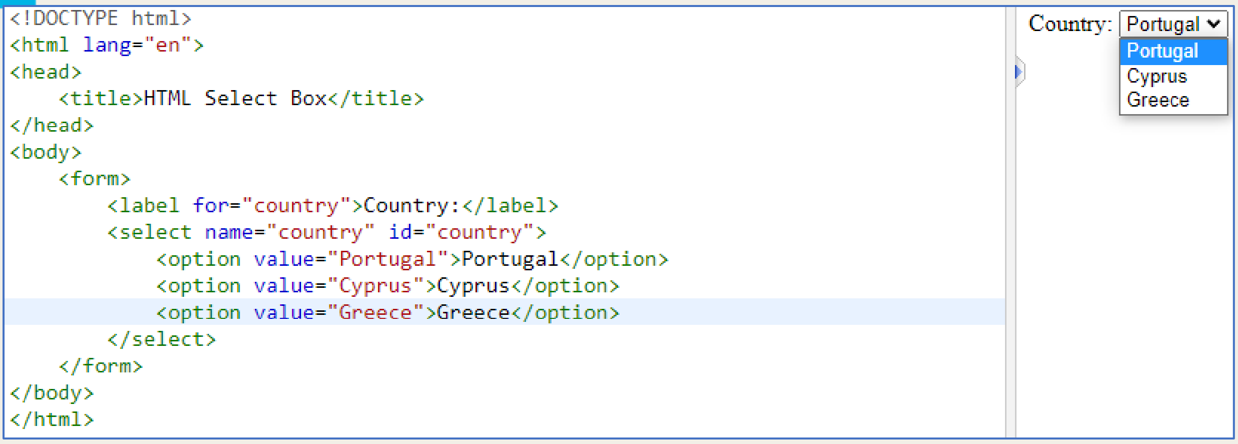


```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML Textarea</title>
</head>
<body>
    <form>
        <label for="address">Your opinion:</label>
        <textarea rows="3" cols="30" name="Your opinion" id="Your opinion">
</textarea>
    </form>
</body>
</html>
```

Your opinion:

*Figure 56 – Code for setting up a multiple-line text input control (**Source:** Author)*

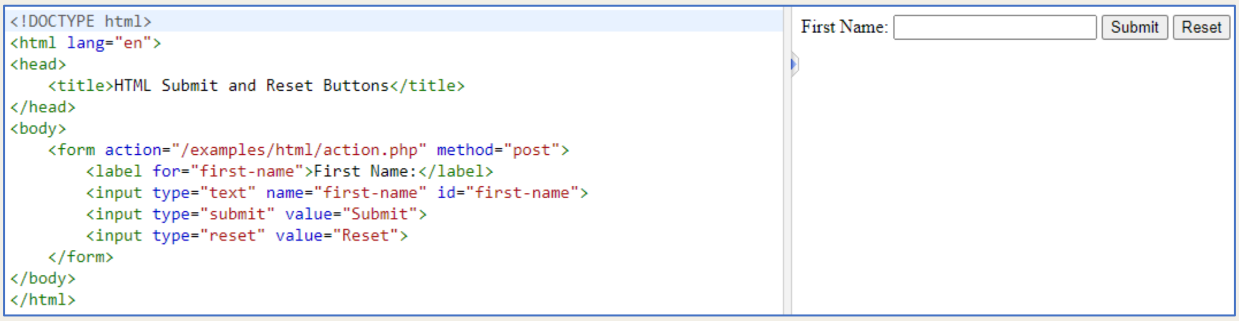
Select boxes are dropdown lists of options in which a user can select one or more options from a pull-down menu. They are created by using the <select> element and <option> element. The <option> elements within the <select> element define each list item.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML Select Box</title>
</head>
<body>
    <form>
        <label for="country">Country:</label>
        <select name="country" id="country">
            <option value="Portugal">Portugal</option>
            <option value="Cyprus">Cyprus</option>
            <option value="Greece">Greece</option>
        </select>
    </form>
</body>
</html>
```

***Figure 57** – Code for setting up select boxes (**Source:** Author)*

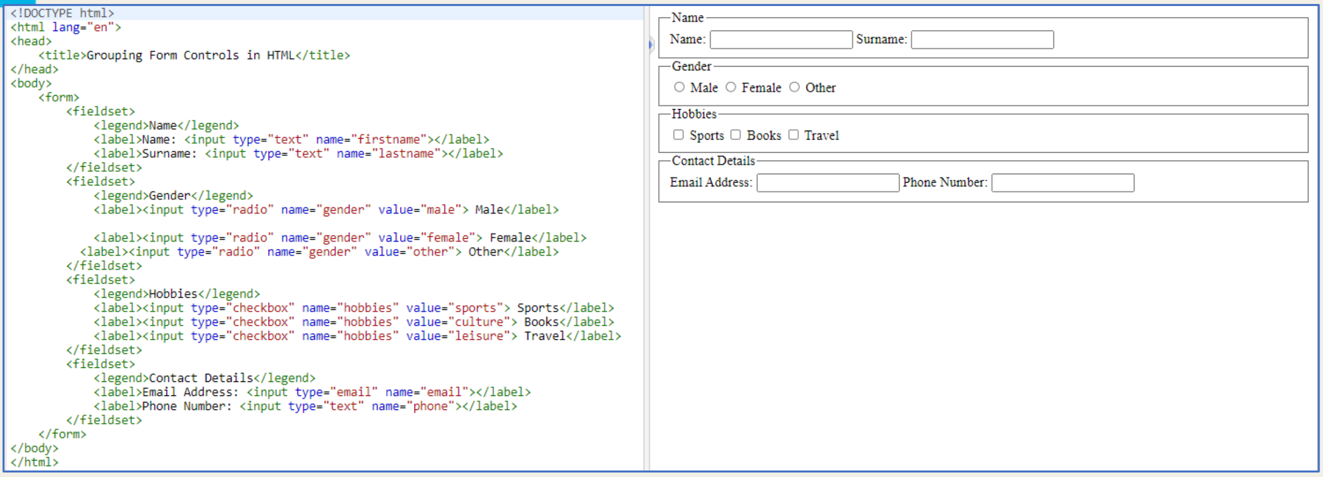
Submit and reset buttons are very common in most of the websites. Submit buttons are used to send (form) data to a web server, while reset buttons are created to reset the form to default values. When the user clicks the submit button, the form data is sent to the file specified in the form's action attribute to process the submitted data. Buttons can also be created using the <button> element. They have the same purpose as buttons created with the input element. However, they offer more rendering possibilities, as they allow the embedding of other HTML elements.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML Submit and Reset Buttons</title>
</head>
<body>
    <form action="/examples/html/action.php" method="post">
        <label for="first-name">First Name:</label>
        <input type="text" name="first-name" id="first-name">
        <input type="submit" value="Submit">
        <input type="reset" value="Reset">
    </form>
</body>
</html>
```

***Figure 58** – Code for setting up submit and reset buttons (**Source:** Author)*

Grouping form controls are a great tool for users to locate a control, making the form more accessible. The <legend> element is key for creating logically related controls, as seen in the picture below:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Grouping Form Controls in HTML</title>
</head>
<body>
    <form>
        <fieldset>
            <legend>Name</legend>
            <label>Name: <input type="text" name="firstname"></label>
            <label>Surname: <input type="text" name="lastname"></label>
        </fieldset>
        <fieldset>
            <legend>Gender</legend>
            <label><input type="radio" name="gender" value="male"> Male</label>

            <label><input type="radio" name="gender" value="female"> Female</label>
          <label><input type="radio" name="gender" value="other"> Other</label>
        </fieldset>
        <fieldset>
            <legend>Hobbies</legend>
            <label><input type="checkbox" name="hobbies" value="sports"> Sports</label>
            <label><input type="checkbox" name="hobbies" value="culture"> Books</label>
            <label><input type="checkbox" name="hobbies" value="leisure"> Travel</label>
        </fieldset>
        <fieldset>
            <legend>Contact Details</legend>
            <label>Email Address: <input type="email" name="email"></label>
            <label>Phone Number: <input type="text" name="phone"></label>
        </fieldset>
    </form>
</body>
</html>
```

Name

Name: ▭ Surname: ▭

Gender

○ Male ○ Female ○ Other

Hobbies

□ Sports □ Books □ Travel

Contact Details

Email Address: ▭ Phone Number: ▭

Figure 59 *– Code for setting up grouping form controls (**Source:** Author, check this for better definition)*

Below there is a list of frequently used form element's attributes:

Attribute	Description
`name`	Specifies the name of the form.
`action`	Specifies the URL of the program or script on the web server that will be used for processing the information submitted via form.
`method`	Specifies the HTTP method used for sending the data to the web server by the browser. The value can be either get (the default) and post.
`target`	Specifies where to display the response that is received after submitting the form. Possible values are _blank, _self, _parent and _top.
`enctype`	Specifies how the form data should be encoded when submitting the form to the server. Applicable only when the value of the method attribute is post.

Table 2 *– List of frequently used form element's attributes (**Source:** https://www.tutorialrepublic.com/html-tutorial/html-forms.php)*

More information regarding other attributes can be found here.

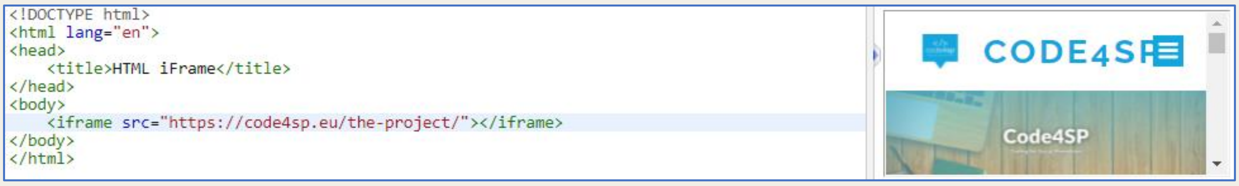
HTML iFrame

An iframe (*or inline frame*) is used to exhibit external objects within a web page, including other web pages. An iframe pretty much performs like a mini web browser within a web browser. Likewise, the content inside an iframe occurs separate from the adjacent elements.

The basic syntax for adding this feature to a web page is as follows:

```
<iframe src="URL"></iframe>
```

The URL specified in the src attribute indicates the location of an external object or a web page.

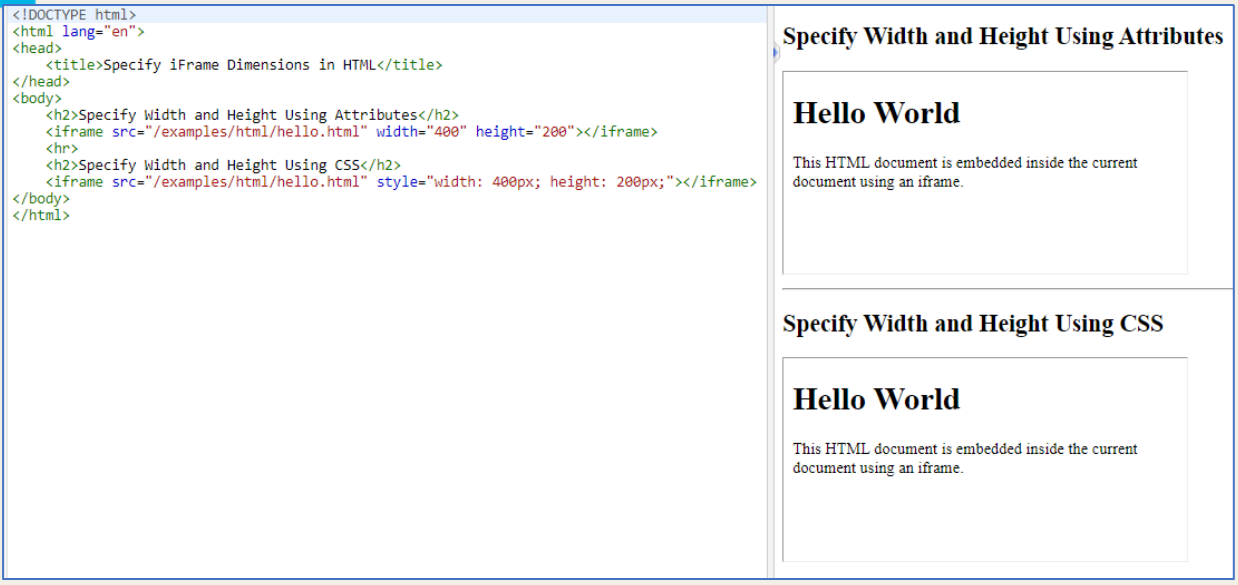


```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML iFrame</title>
</head>
<body>
    <iframe src="https://code4sp.eu/the-project/"></iframe>
</body>
</html>
```

Figure 60 *– Code for setting up an inline frame (**Source:** Author)*

The **height** and **width** of the iframe can be defined by applying the code disposed on *Figure 61* below. The width and height attribute values are stipulated in pixels by default, but users can also set these values in percentage, such as 50%, 100%, etc.. The default width of an iframe is 300 pixels, while the default height is 150 pixels.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Specify iFrame Dimensions in HTML</title>
</head>
<body>
    <h2>Specify Width and Height Using Attributes</h2>
    <iframe src="/examples/html/hello.html" width="400" height="200"></iframe>
    <hr>
    <h2>Specify Width and Height Using CSS</h2>
    <iframe src="/examples/html/hello.html" style="width: 400px; height: 200px;"></iframe>
</body>
</html>
```

Specify Width and Height Using Attributes

Hello World

This HTML document is embedded inside the current document using an iframe.

Specify Width and Height Using CSS

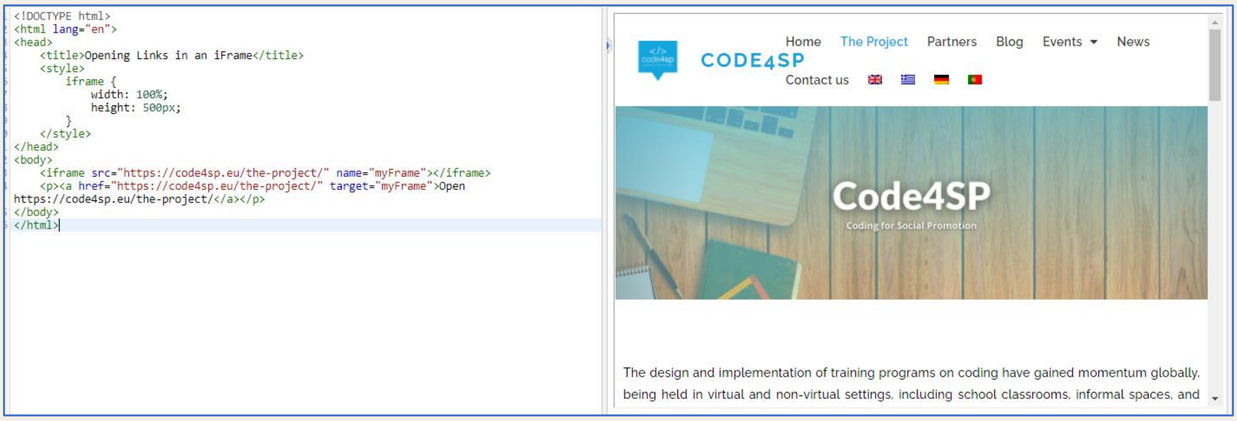
Hello World

This HTML document is embedded inside the current document using an iframe.

*Figure 61 – Code for setting up height and width of an inline frame (**Source:** Author)*

As it could be verified, iframe has a border around it set by default. To modify or remove it, the best way is to use the CSS border feature (to be taught in the CSS topic).

An iframe can also be used as a target for the hyperlinks. It can be named using the name attribute. This means that when a link with a target attribute (with that name set as value) is clicked, the linked source will open in the same inline frame, as follows:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Opening Links in an iFrame</title>
    <style>
        iframe {
            width: 100%;
            height: 500px;
        }
    </style>
</head>
<body>
    <iframe src="https://code4sp.eu/the-project/" name="myFrame"></iframe>
    <p><a href="https://code4sp.eu/the-project/" target="myFrame">Open
https://code4sp.eu/the-project/</a></p>
</body>
</html>
```

The design and implementation of training programs on coding have gained momentum globally, being held in virtual and non-virtual settings, including school classrooms, informal spaces, and

*Figure 62 – Using iframe as a target for a hyperlink (**Source:** Author)*

2.2. HTML advanced concepts

HTML Doctypes

A Document Type Declaration (DOCTYPE) is an instruction to the web browser concerning the version of markup language in which a web page is created. It appears at the top of a web page before all other elements. According to the HTML specification, every HTML document requires a valid document type declaration to ensure that web pages are displayed the way they are meant to be. The doctype declaration is frequently the first thing defined in an HTML document (even before the opening <html> tag); nevertheless, the doctype declaration itself is not an HTML tag.

The DOCTYPE for HTML5 is very short, concise, and case-insensitive: `<!DOCTYPE html>`.

The following markup can be used as a template to create a new HTML5 document:

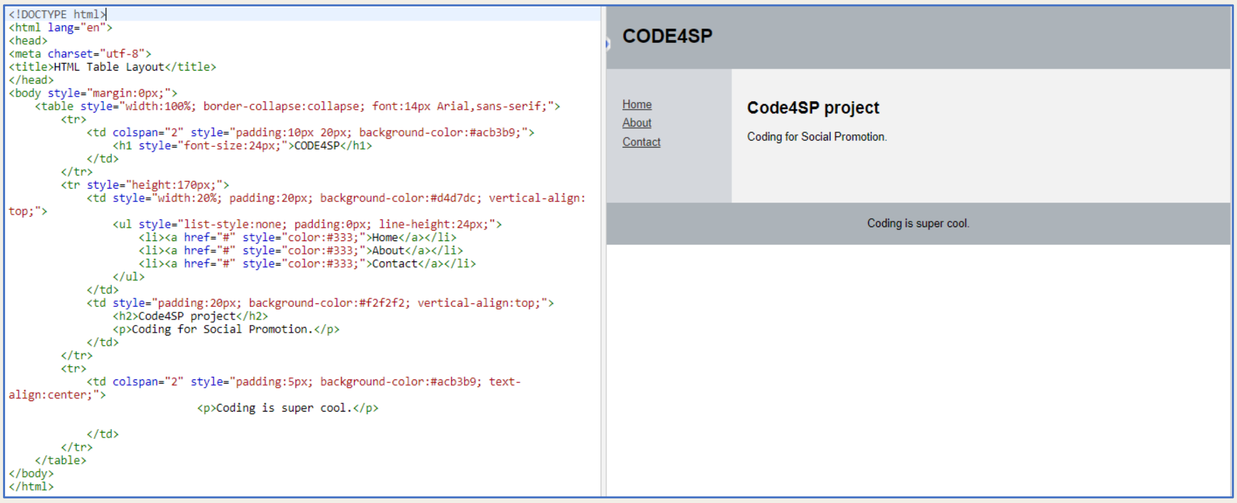
```
<!DOCTYPE html> <html lang="en"> <head> <meta charset="utf-8">
<title><!-- Insert your title here --></title> </head> <body> <!--
Insert your content here --> </body> </html>
```

HTML Layouts

There are different methods of creating a web page layout, positioning the various elements that compose a web page in a well-structured manner and giving an engaging appearance to the website. Most websites usually display their content in multiple rows and columns, configured like a magazine to provide the users with a better reading and writing atmosphere. This can be easily attained by utilizing the HTML tags, such as <table>, <div>, <header>, <footer>, <section>, etc. and combining some CSS styles to them.

The simplest way for creating layouts in HTML is indeed obtained by designing tables. As seen in the previous sections, this usually involves the process of putting contents (text, images, etc.) into rows and columns.

The layout below contains an HTML table with three rows and two columns. It should be noted that the first and last row spans both uses the colspan attribute. It should be stated that the method used for creating layout in this example, although it is not wrong, it is not recommended. Tables and inline styles for creating layouts should be avoided. Layouts created using tables often rendered very slowly. **Tables should only be used to display tabular data.** For creating such layouts, **CSS float techniques** are advised. Users shall learn about this tool moreover.



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>HTML Table Layout</title>
</head>
<body style="margin:0px;">
    <table style="width:100%; border-collapse:collapse; font:14px Arial,sans-serif;">
        <tr>
            <td colspan="2" style="padding:10px 20px; background-color:#acb3b9;">
                <h1 style="font-size:24px;">CODE4SP</h1>
            </td>
        </tr>
        <tr style="height:170px;">
            <td style="width:20%; padding:20px; background-color:#d4d7dc; vertical-align:
top;">
                <ul style="list-style:none; padding:0px; line-height:24px;">
                    <li><a href="#" style="color:#333;">Home</a></li>
                    <li><a href="#" style="color:#333;">About</a></li>
                    <li><a href="#" style="color:#333;">Contact</a></li>
                </ul>
            </td>
            <td style="padding:20px; background-color:#f2f2f2; vertical-align:top;">
                <h2>Code4SP project</h2>
                <p>Coding for Social Promotion.</p>
            </td>
        </tr>
        <tr>
            <td colspan="2" style="padding:5px; background-color:#acb3b9; text-
align:center;">
                              <p>Coding is super cool.</p>

            </td>
        </tr>
    </table>
</body>
</html>
```

CODE4SP

Home
About
Contact

Code4SP project

Coding for Social Promotion.

Coding is super cool.

***Figure 63** – HTML basic layout (**Source:** Author, adapted from Tutorial Republic. Click here for better resolution)*

HTML5 has established new structural elements, for instance <header>, <footer>, <nav>, <section>, etc. to identify the different parts of a web page in a more semantic way. This example applies the new HTML5 structural elements to create the same layout created in *Figure 62*.

To know more about newly introduced tags, learners should visit this source.

HTML Head

The head element is the receptacle for all head elements, which provide extra information about the document or reference to other resources, required for the document to perform properly. It illustrates the properties of the document such as title, deliver meta information like character set, tell the browser where to find the style sheets or scripts that allows to expand the HTML document interactively.

The HTML elements that can be used inside the <head> element are: <title>, <base>, <link>, <style>, <meta>, <script> and <noscript>.

The HTML title Element

The <title> element identifies the title of the document and only one is required in all HTML/XHTML documents to produce a valid document. It must be placed within the <head> element. The title element comprises plain text and entities and it may not include other markup tags. It may be used for different functions:

- To show a title in the browser title bar and in the task bar;
- To deliver a title for the page when it is added to favourites or bookmarked;
- To present a title for the page in search-engine results (e.g., Google search).

It should be **short** and **specific** to the content of the document, as search engines would pay particular attention to the words present in the title – it should have ideally 65 characters.

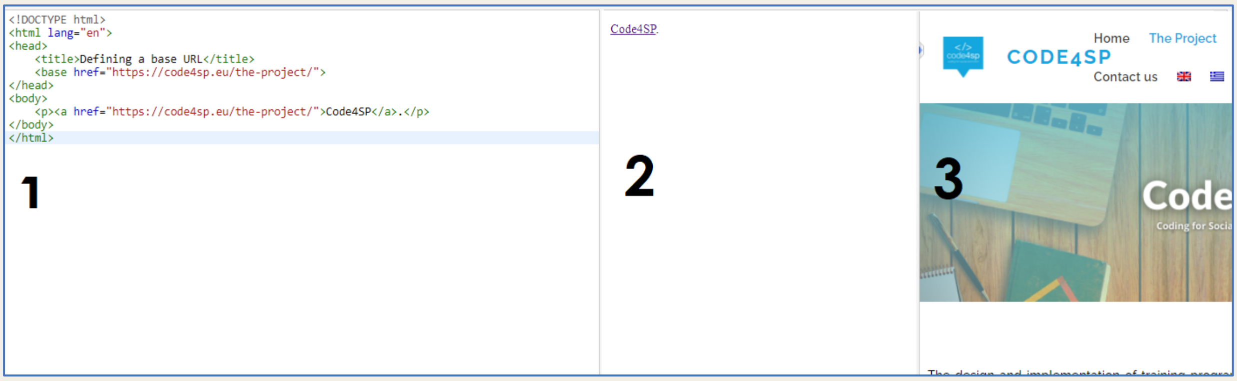
A title in an HTML document should be displayed as follows:

```
<!DOCTYPE html> <html lang="en"> <head> <title>A simple HTML
document</title> </head> <body> <p>Hello World! </p> </body> </html>
```

The HTML base Element

The HTML <base> element is used to identify a base URL for all relative links contained in the document. For example, one can set the base URL once at the top of the page, and then all subsequent relative links will use that URL as a starting point, as follows:

```
<!DOCTYPE  html>  <html  lang="en">  <head>  <title>Defining  a  base
URL</title>  <base  href=" https://code4sp.eu/the-project/  ">  </head>
<body>  <p><a  href=" https://code4sp.eu/the-project/  ">HTML  Head</a>.
</p> </body> </html>
```



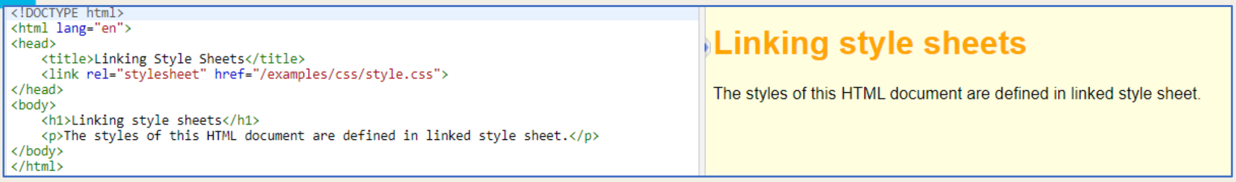
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Defining a base URL</title>
    <base href="https://code4sp.eu/the-project/">
</head>
<body>
    <p><a href="https://code4sp.eu/the-project/">Code4SP</a>.</p>
</body>
</html>
```

__Figure 64__ – HTML base element (__Source:__ Author)

The HTML <base> element must be found before any element that belongs to an external resource. HTML permits only one base element for each document.

The HTML link Element

The <link> element describes the correlation between the current document and an external document or resource. A common use of link element is to connect to external style sheets.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Linking Style Sheets</title>
    <link rel="stylesheet" href="/examples/css/style.css">
</head>
<body>
    <h1>Linking style sheets</h1>
    <p>The styles of this HTML document are defined in linked style sheet.</p>
</body>
</html>
```

Linking style sheets

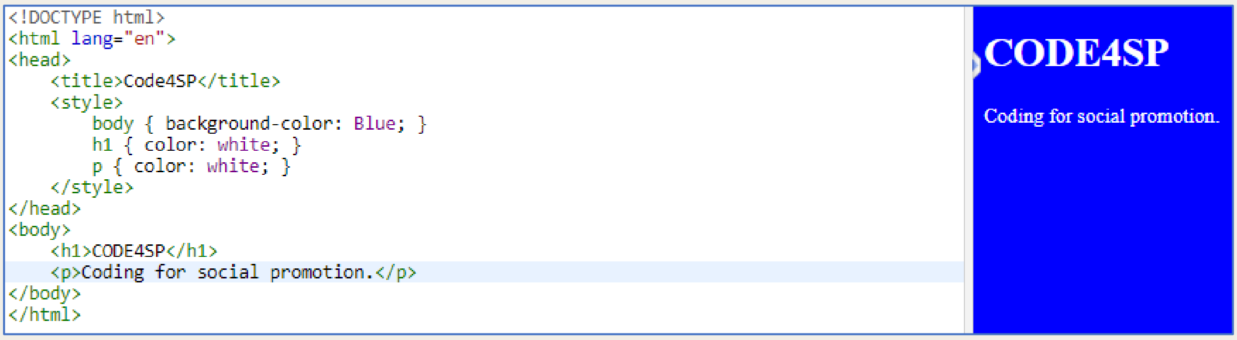
The styles of this HTML document are defined in linked style sheet.

***Figure 65** – HTML base element (**Source:** https://www.tutorialrepublic.com/codelab.php?topic=html&file=linking-style-sheet)*

It should be stated that an HTML document's <head> element may include any number of <link> elements. The <link> element has attributes, but no contents.

The HTML style Element

The <style> element is applied to describe embedded style information for an HTML document. The style rules inside the <style> element indicate how HTML elements should render in a browser. An embedded style sheet should be used when a single document has a unique style. If the same style sheet is used in various documents, then an external style sheet would be more suitable.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Code4SP</title>
    <style>
        body { background-color: Blue; }
        h1 { color: white; }
        p { color: white; }
    </style>
</head>
<body>
    <h1>CODE4SP</h1>
    <p>Coding for social promotion.</p>
</body>
</html>
```

CODE4SP

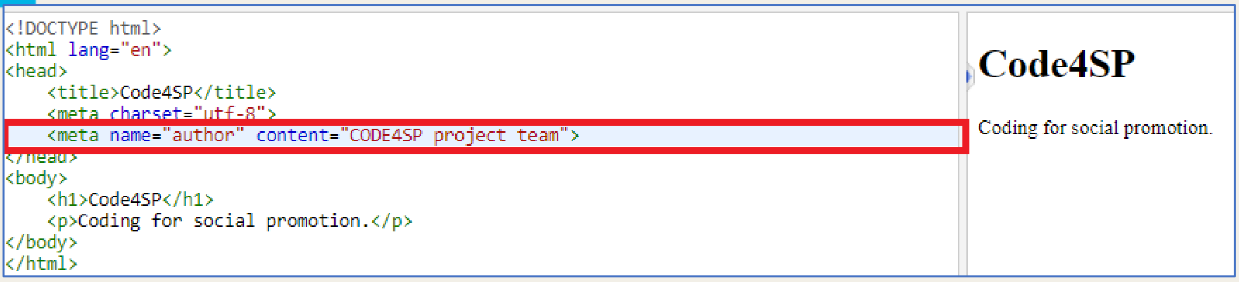
Coding for social promotion.

***Figure 66** – Coding various style elements (**Source:** Author)*

The HTML meta Element

The <meta> element provides metadata about the HTML document, which is a set of data that describes and gives information about other data. <meta> tags always appear inside the <head> element, and are typically used to specify character set, page description, keywords, author of the document, and viewport settings.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Code4SP</title>
    <meta charset="utf-8">
    <meta name="author" content="CODE4SP project team">
</head>
<body>
    <h1>Code4SP</h1>
    <p>Coding for social promotion.</p>
</body>
</html>
```

Code4SP

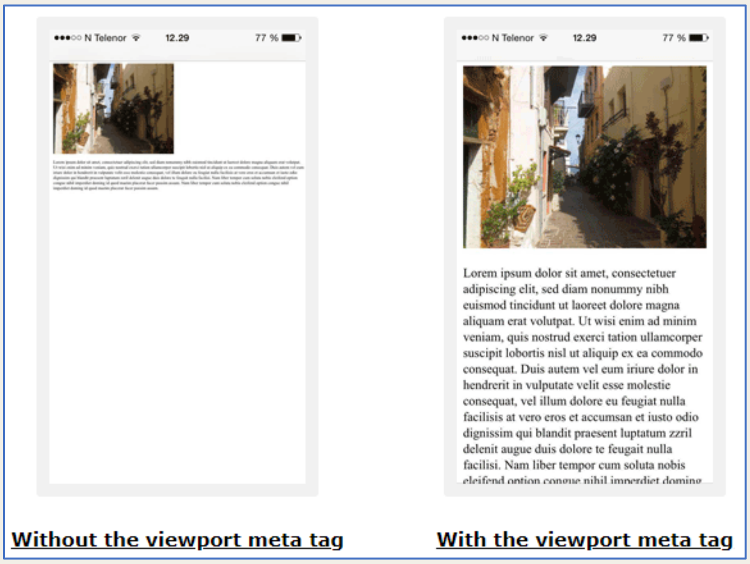
Coding for social promotion.

*Figure 67 – The meta element (**Source:** Author)*

As seen above, meta tags include information about a web page. It is not visible in the browser (it is machine parsable though). Metadata is utilised by browsers (how to display content or reload page), search engines (keywords), and other web services. Moreover, there is a technique to let web designers rule over the **viewport**, through the <meta> tag. The viewport is the user's visible area of a web page. It differs from device to device (it will be bigger on a computer screen than on a mobile phone).

The following <meta> element should be included in all web pages, as it will give the browser guidelines on how to assume the page dimensions and scaling: `<meta name="viewport" content="width=device-width, initial-scale=1.0">`
The `width=device-width` part sets the width of the page to follow the screen-width of the device (which will vary depending on the screen). The `initial-scale=1.0` part sets the initial zoom level when the page is initially loaded by the browser.

The difference between webpages with or without viewport meta tag is quite visible in the following example:



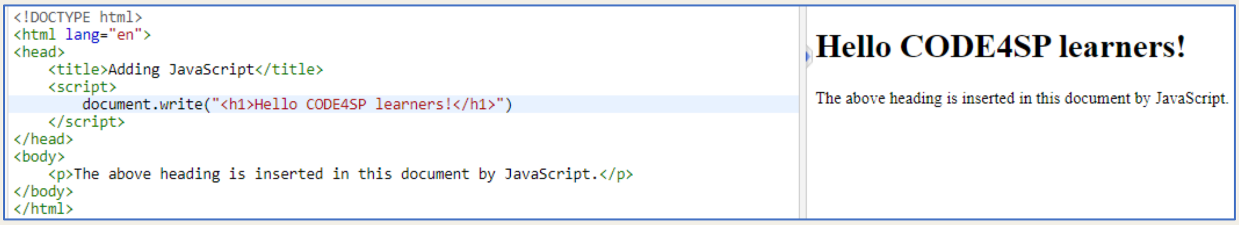
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming

Without the viewport meta tag **With the viewport meta tag**

Figure 68 *– With or without viewport meta tags examples (**Source:** https://www.w3schools.com/tags/tag_meta.asp)*

The HTML script Element

The <script> element is used to define client-side script, such as JavaScript in HTML documents.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Adding JavaScript</title>
    <script>
        document.write("<h1>Hello CODE4SP learners!</h1>")
    </script>
</head>
<body>
    <p>The above heading is inserted in this document by JavaScript.</p>
</body>
</html>
```

Hello CODE4SP learners!

The above heading is inserted in this document by JavaScript.

Figure 69 *– The script element (**Source:** Author)*

Working with Client-side Script

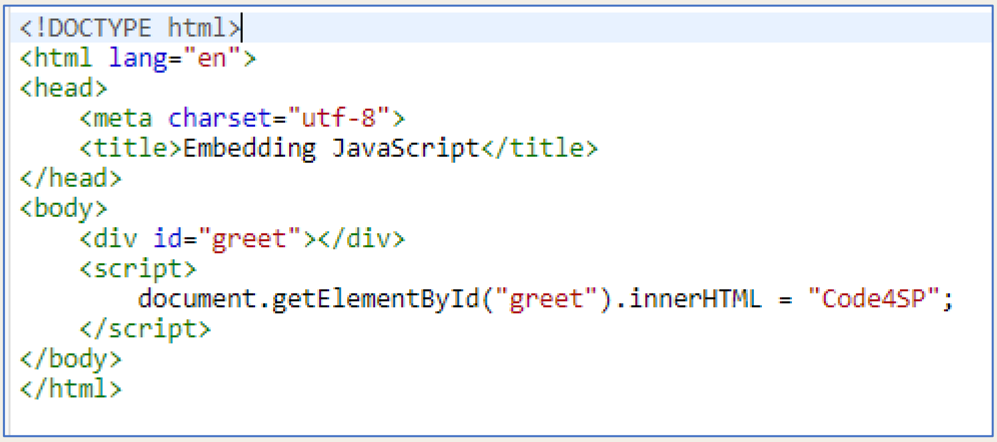
Client-side scripting is linked to the type of computer programs that are performed by the user's web browser. **JavaScript (JS)** is the most widespread client-side scripting language on the web. The <script> element is used to embed or reference JavaScript within an HTML document to add interactivity to web pages and to create a more user-friendly experience. Some of the most common uses of JavaScript are form validation, generating alert messages, creating image gallery, show hide content, DOM manipulation, etc.

JavaScript can be **embedded** following two ways:

1. Directly inside the HTML page; or
2. Placed in an external script file and referenced inside the HTML page.

Note: both techniques use the <script> element.

To embed JS in an HTML file, the user must add the code as the content of the <script> element, following the example of *Figure 66*. Preferably, script elements should be placed at the end of the page, before the closing body tag (</body>), because when browser encounters a script, it pauses rendering the rest of the page until it deconstructs the script that may drastically impact the website performance.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Embedding JavaScript</title>
</head>
<body>
    <div id="greet"></div>
    <script>
        document.getElementById("greet").innerHTML = "Code4SP";
    </script>
</body>
</html>
```

***Figure 70** – Embedding JS in an HTML doc (**Source:** Author)*

JavaScript codes can also be placed into a separate file, calling up a .js extension, while corresponding it in the HTML document by using the src attribute of the <script> tag. This can be especially useful if the web designer wants to make the same script available for multiple documents, so he/she does not need to perform the same tasks repeatedly. When the src attribute is indicated, the <script> element should be empty, so the web designer cannot use the same <script> element to both embed the JavaScript and to link to an external JavaScript file in an HTML document.

If a browser, for some reason, does not support client-side scripting, or users have disabled JS in their browser, the <no script> element can be used to provide an alternative content. This element can include any HTML elements, except <script>, as it can be included in the <body> element of a HTML page.

HTML Entities

Most learners will certainly be curious about how to display special characters and symbols in their programming processes. Hence, this subchapter is intended to explain how they can be successful in doing such thing.

First, it is important to understand what a HTML entity is. As perceived in the previous chapters, some characters are quite reserved in HTML. For instance, one cannot use signs such as '<' or '>', as the browser could mistake them for a markup. Moreover, some characters are just not available in the keyboard (e.g., the copyright symbol).

These special characters can be displayed by simply being replaced with the character entities (or just entities), then solving the aforementioned troubles. Below is a list of the most frequently used entities in HTML:

Result	Description	Entity Name	Numerical reference
	non-breaking space	&nbsp;	&#160;
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	ampersand	&amp;	&#38;

"	quotation mark	&quot;	&#34;
'	apostrophe	&apos;	&#39;
¢	cent	&cent;	&#162;
£	pound	&pound;	&#163;
¥	yen	&yen;	&#165;
€	euro	&euro;	&#8364;
©	copyright	&copy;	&#169;
®	registered trademark	&reg;	&#174;
™	trademark	&trade;	&#8482;

***Table 3** – The most frequent entities in HTML (**Source:** https://www.tutorialrepublic.com/html-tutorial/html-entities.php)*

Numeric character references can also be used as an alternative for entity names, especially because they have stronger browser support, and can be used to specify any Unicode character, however entities are limited to a subgroup of this.

HTML URL

URL: how many times has this abbreviation appeared in virtual environments? It stands for Uniform Resource Locator, and it works as the global address of documents and other resources on the World Wide Web. Its goal is to identify the location of a document and other resources available online, while specifying the mechanism for accessing it using a web browser. For instance, https://code4sp.eu/ is an URL.

The general syntax of URLs can be described as follows: `scheme://host:port/path?query-string#fragment-id`

URLs have a linear structure and are composed of the following components:

- **Scheme name** — The scheme recognizes the protocol to be used to access a resource on the Internet. The scheme names are followed by the three characters :// (a colon and two slashes). The most used protocols are http://, https://, ftp://, and mailto://.
- **Host name** — identifies the host where the resource is situated. A hostname is a domain name given to a host computer. This is usually a combination of the host's local name with its parent domain's name. For example, www.code4sp.eu/ consists of host's machine name www and the domain name code4sp.eu.
- **Port Number** — Servers often deliver more than one type of service, so they must be told what service is being requested. These requests are made by port number. Well-known port numbers for a service are normally omitted from the URL. For example, web service HTTP runs by default over port 80, HTTPS runs by default over port 443.
- **Path** — identifies the specific resource within the host that the user wants to access. For example, /html/html-url.php, /news/technology/, etc.
- **Query String** — contains data to be passed to server-side scripts, running on the web server. For instance, parameters for a search. The query string preceded by a question mark (?), is usually a string of name and value pairs separated by ampersand (&), for example, ?first_name=John&last_name=Corner, q=mobile+phone, and so on.
- **Fragment identifier** —specifies a location within the page. Browser may scroll to display that part of the page. The fragment identifier introduced by a hash character (#) is the optional last part of a URL for a document.

HTML URL Encoding

It is well-known that sometimes data is not safely transmitted over the internet. This happens mostly because URLs are not fully or accurately encoded and causes some misunderstandings among internet users.

According to RFC 3986, the characters in a URL only restricted to a defined set of reserved and unreserved US-ASCII characters. Any other characters are not allowed in a URL (that is why some Latin and Cyrillic characters are not seen in URL). But URL often comprises characters outside the US-ASCII character set, so they must be converted to a valid US-ASCII format for worldwide interoperability. URL-encoding is a process of converting URL information so that it can be safely transmitted over the internet.

To map the large range of characters used globally, a two-step method is followed:

- Firstly, the data is encoded in relation to the UTF-8 character encoding.
- Then only those bytes that do not correspond to characters in the unreserved set should be percent-encoded like %HH, where HH is the hexadecimal value of the byte.

For instance, look at this Portuguese popular saying:

"Quem vê caras, não vê corações." ["Faces we see, hearts we do not know."]

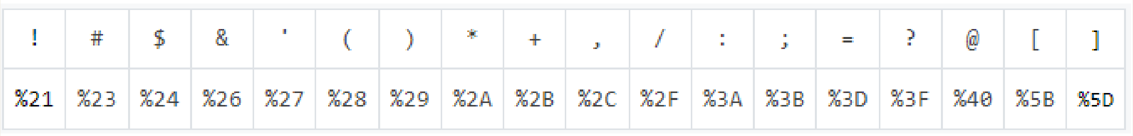
This sentence would be encoded as:

Quem%20v%C3%AA%20caras%2C%20n%C3%A3o%20v%C3%AA%20cora%C3%A7%C3%B5es.

Ç, ç (c-cedilla) is a Latin script letter, as well as well as the accents used (~ and ^).

Certain characters are restricted from use in a URL, as they may (or may not) be defined as delimiters by the generic syntax in a particular URL scheme (for instance, forward slash / characters are used to separate different parts of a URL).

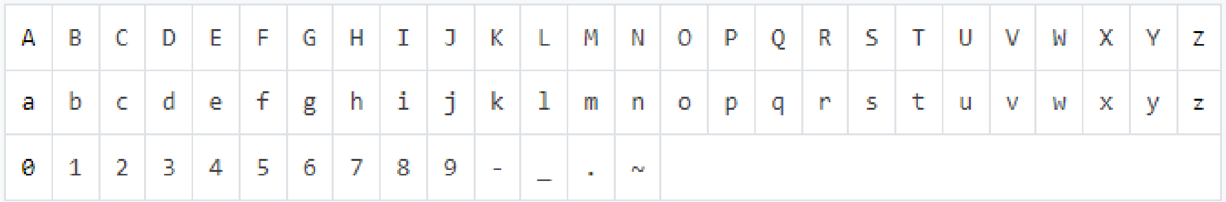
Reserved characters in a URL are as follows:



!	#	$	&	'	(	)	*	+	,	/	:	;	=	?	@	[	]
%21	%23	%24	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

Figure 71 *– Reserved characters in a URL (**Source:** https://www.tutorialrepublic.com/html-tutorial/html-url-encode.php)*

However, there are also characters that, despite of being allowed in a URL, do not have a reserved purpose – that is why they are called "unreserved characters".
These include uppercase and lowercase letters, decimal digits, hyphen, period, underscore, and tilde. The following table lists all the unreserved characters in a URL:



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	-	_	.	~												

Figure 72 *– Unreserved characters in a URL (**Source:** https://www.tutorialrepublic.com/html-tutorial/html-url-encode.php)*

For encoding/decoding characters, users may use this converter.

HTML Validation

To make sure a HTML code follows the current web standards, free from errors, it is of upmost importance to figure out how to validate a HTML code. Beginners often will make mistakes when writing HTML codes, and incorrect or non-standard codes will certainly cause unexpected results in how a web page is displayed in a browser.

In order to prevent this to happen, users can test their codes within the formal guidelines and standards set by the Wide Web Consortium (W3C) for HTML/XHTML web pages. There is an online tool that automatically checks HTML codes and points out any problems/errors, like missing closing tags or missing quotes around attributes.

The process of validating a web page is ensuring the respect for the norms/standards defined by the W3C, so it is very important. Some reasons for validating a web page are:

- It helps to create web pages that are cross-browser, cross-platform compatible. Most likely they will be compatible with the future version of web browsers and web standards.
- Standards compliant web pages increase the search engine spiders and crawlers visibility, as a result your web pages will more likely be appear in search results.
- It will reduce unexpected errors and make your web pages more accessible to the visitor.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

2.3. HTML5 Features

In this subchapter, the features of the fifth (and last) major HTML version recommended by the W3C will be explained.

HTML5 New Input Types

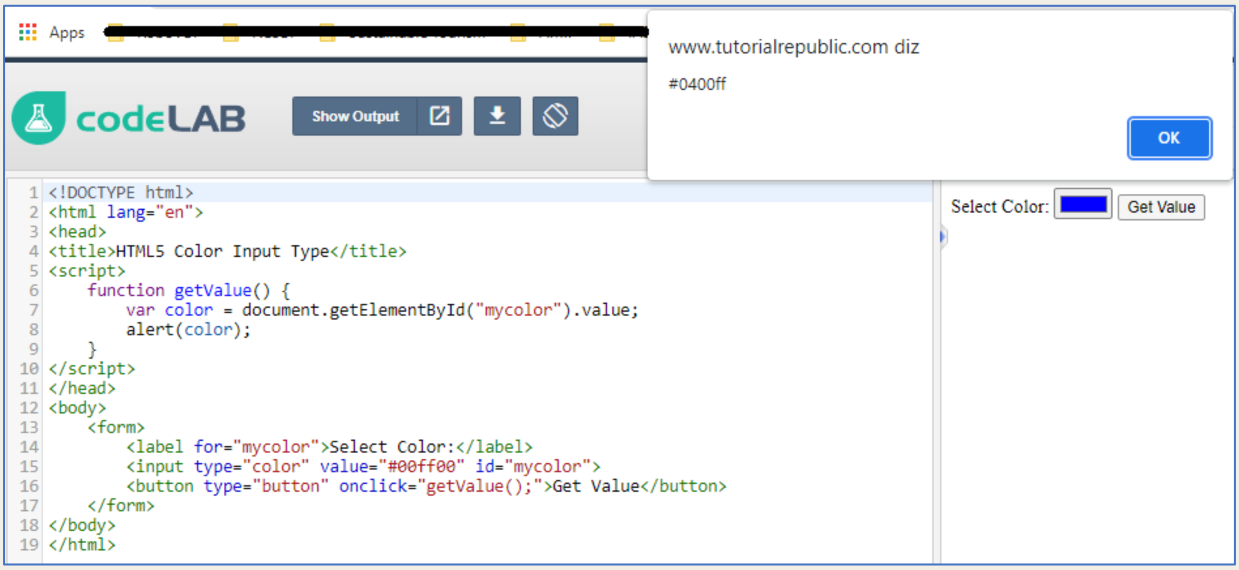
HTML5 introduces several new <input> types like email, date, time, color, range, and so on, with the goal of improving the user experience and to make the forms more interactive. Nevertheless, if a browser failed to recognize these new input types, it will consider them a normal text box.

The following are some new input types:

- color
- date
- datetime-local
- email
- month
- number
- range
- search
- tel
- time
- url
- week

Regarding the color input, it allows to select a colour from a colour picker and gives information about the colour value in hexadecimal format (e.g., #000000, which is black, the default colour if the user does not specify a value), as verified in *Figure 73* below.

It should be noted that the color input is supported in all major modern web browsers (Firefox, Chrome, Opera, Safari (12.1+), Edge (14+)), but it is not supported by the Microsoft Internet Explorer and older versions of Apple Safari browsers.



[redacted]

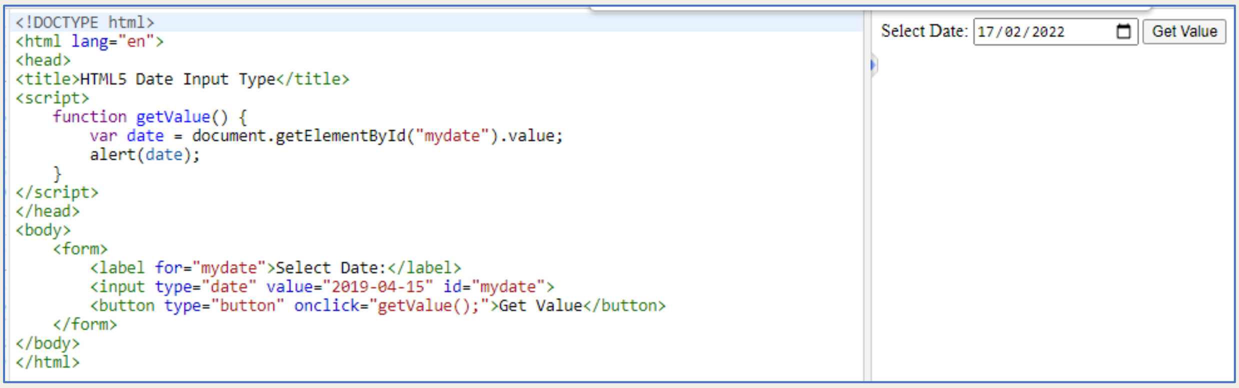
www.tutorialrepublic.com diz

#0400ff

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Color Input Type</title>
<script>
    function getValue() {
        var color = document.getElementById("mycolor").value;
        alert(color);
    }
</script>
</head>
<body>
    <form>
        <label for="mycolor">Select Color:</label>
        <input type="color" value="#00ff00" id="mycolor">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```

***Figure 73** – Picking a colour using the color input (**Source:** https://www.tutorialrepublic.com/codelab.php?topic=html5&file=color-input-type)*

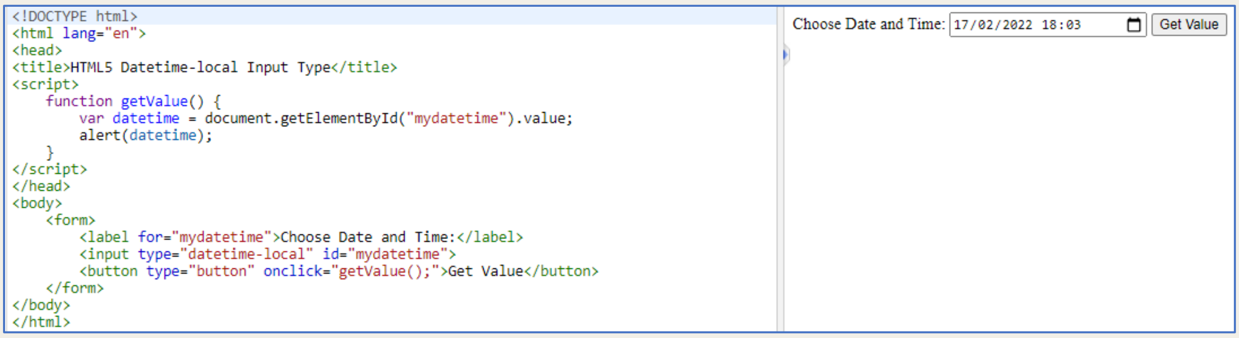
Concerning the date input type, it allows the user to select a date from a drop-down calendar, in which he/she may choose the year, month and day (but not time). This feature is also supported by most browsers, except for Internet Explorer and Safari.



```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Date Input Type</title>
<script>
    function getValue() {
        var date = document.getElementById("mydate").value;
        alert(date);
    }
</script>
</head>
<body>
    <form>
        <label for="mydate">Select Date:</label>
        <input type="date" value="2019-04-15" id="mydate">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```

***Figure 74** – The date input type (**Source:** https://www.tutorialrepublic.com/codelab.php?topic=html5&file=color-input-type)*

The datetime-local input type makes available to the user to select both local date and time, including the year, month, and day including the time in hours and minutes. This input is supported by Safari, Firefox and IE, but not by Chrome, Edge and Opera.

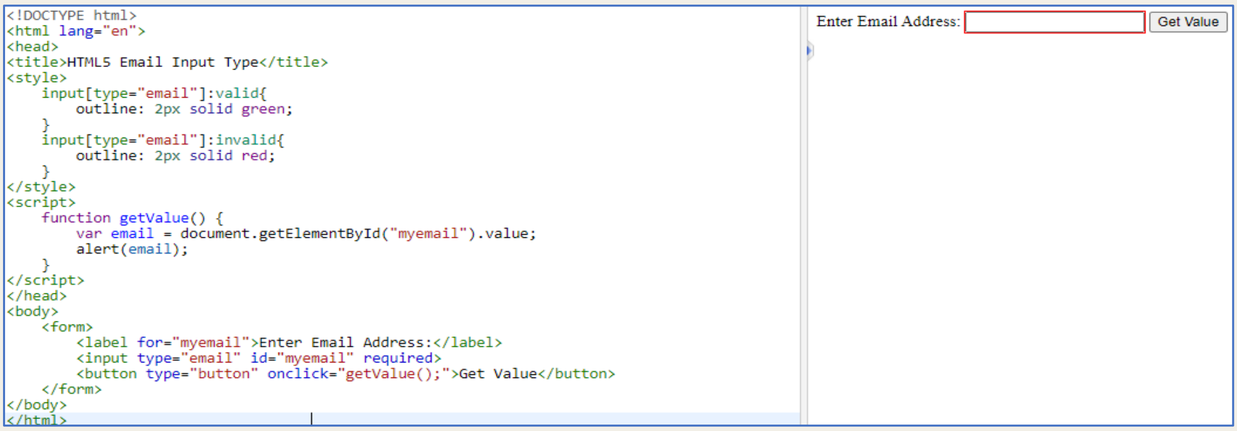


```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Datetime-local Input Type</title>
<script>
    function getValue() {
        var datetime = document.getElementById("mydatetime").value;
        alert(datetime);
    }
</script>
</head>
<body>
    <form>
        <label for="mydatetime">Choose Date and Time:</label>
        <input type="datetime-local" id="mydatetime">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```

Choose Date and Time: 17/02/2022 18:03

***Figure 75** – The datetime-local input type (**Source:** https://www.tutorialrepublic.com/codelab.php?topic=html5&file=color-input-type)*

To allow the user to enter his/her e-mail address, email is the preferred input type. It is quite like a standard text input type, but if it is applied in combination with the required attribute, browsers may search for the patterns to guarantee a properly formatted e-mail address should be entered. The email input field can be styled for different validation states, when a value is entered using the :valid, :invalid or :required pseudo-classes. This input type is supported by all major browsers.

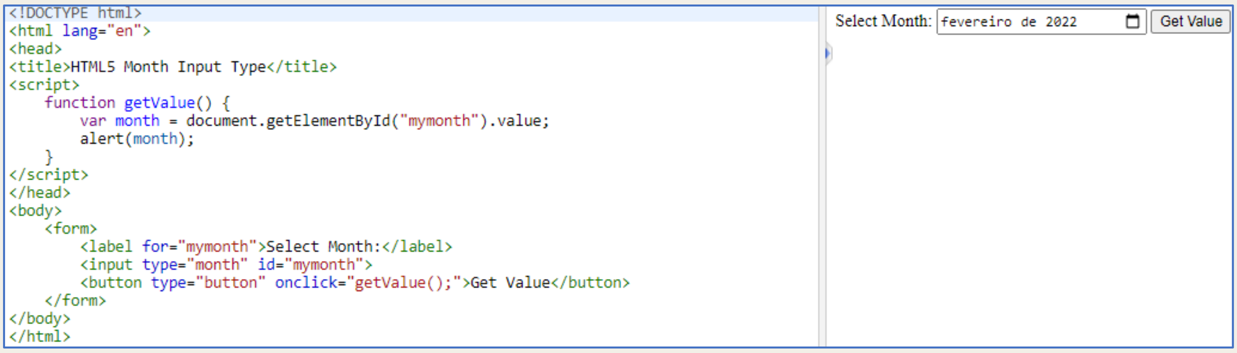


```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Email Input Type</title>
<style>
    input[type="email"]:valid{
        outline: 2px solid green;
    }
    input[type="email"]:invalid{
        outline: 2px solid red;
    }
</style>
<script>
    function getValue() {
        var email = document.getElementById("myemail").value;
        alert(email);
    }
</script>
</head>
<body>
    <form>
        <label for="myemail">Enter Email Address:</label>
        <input type="email" id="myemail" required>
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```

Enter Email Address:

***Figure 76** – The email input type (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

In what regards month Input type, this is a very similar feature to the previous time ones since it allows to select a month and year from a drop-down calendar (being 'YYYY' the year and ''MM'' the month. It should be noted that this is not supported by Firefox, Safari and Internet Explorer. Only Chrome, Edge and Opera browsers support it.

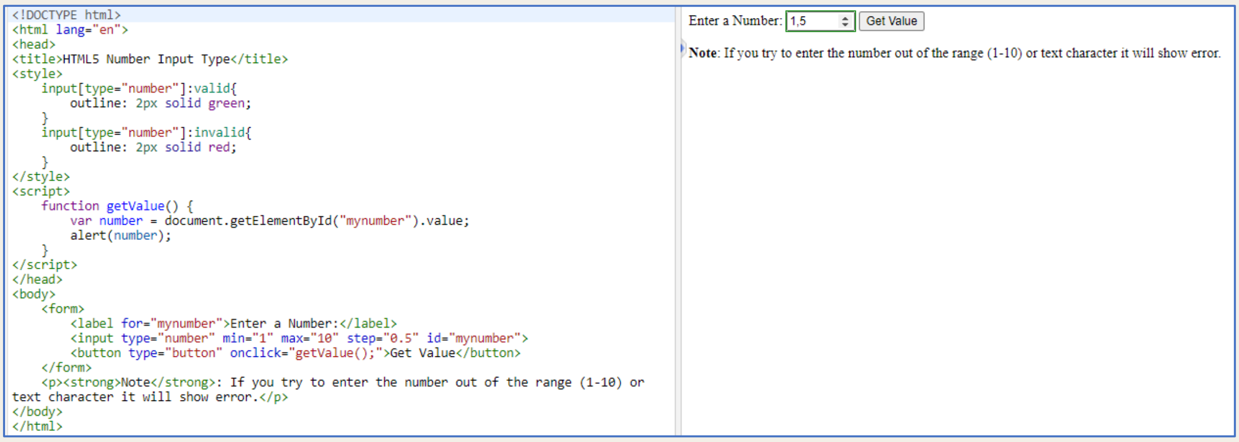


```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Month Input Type</title>
<script>
    function getValue() {
        var month = document.getElementById("mymonth").value;
        alert(month);
    }
</script>
</head>
<body>
    <form>
        <label for="mymonth">Select Month:</label>
        <input type="month" id="mymonth">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```

Select Month: fevereiro de 2022 Get Value

***Figure 77** – The month input type (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Concerning the number input type, it is used for inserting a numerical value. The web designer can also limit the user to enter only acceptable values. For this to happen, the additional attributes min, max, and step should be used. This feature is supported by all major web browsers.



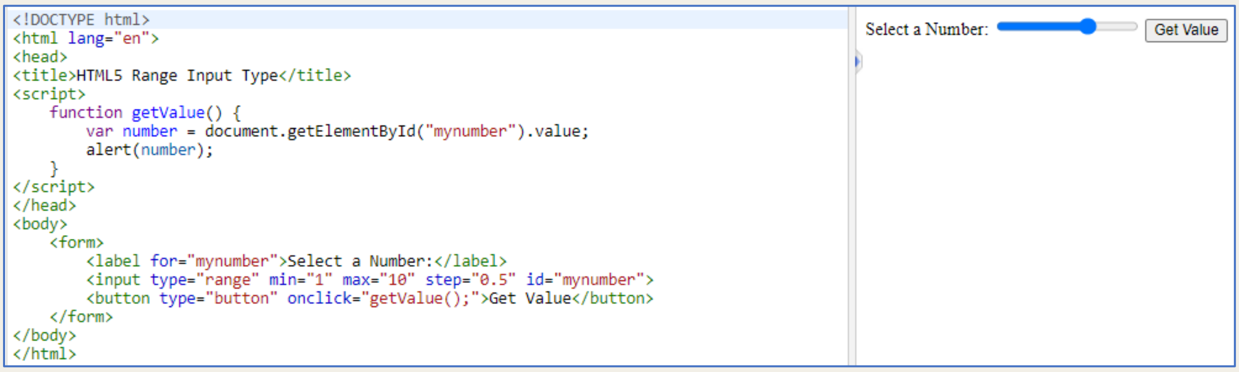
```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Number Input Type</title>
<style>
    input[type="number"]:valid{
        outline: 2px solid green;
    }
    input[type="number"]:invalid{
        outline: 2px solid red;
    }
</style>
<script>
    function getValue() {
        var number = document.getElementById("mynumber").value;
        alert(number);
    }
</script>
</head>
<body>
    <form>
        <label for="mynumber">Enter a Number:</label>
        <input type="number" min="1" max="10" step="0.5" id="mynumber">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
    <p><strong>Note</strong>: If you try to enter the number out of the range (1-10) or
text character it will show error.</p>
</body>
</html>
```

Enter a Number: 1,5 Get Value

Note: If you try to enter the number out of the range (1-10) or text character it will show error.

***Figure 78** – The number input type (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

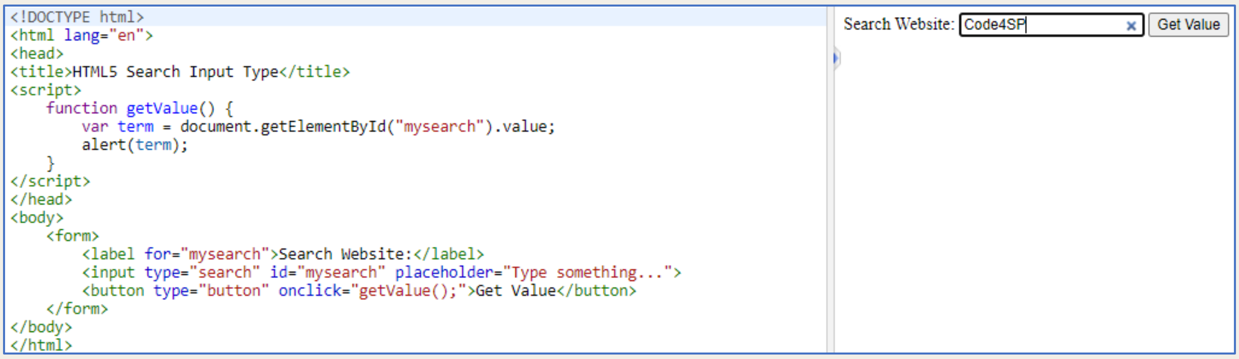
The range input type can be applied for registering a numerical value within a specific range. It works very similar to number input above, although it presents an easier way for inserting a number. This input type is supported by all major web browsers.



```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Range Input Type</title>
<script>
    function getValue() {
        var number = document.getElementById("mynumber").value;
        alert(number);
    }
</script>
</head>
<body>
    <form>
        <label for="mynumber">Select a Number:</label>
        <input type="range" min="1" max="10" step="0.5" id="mynumber">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```

***Figure 79** – The range input type (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

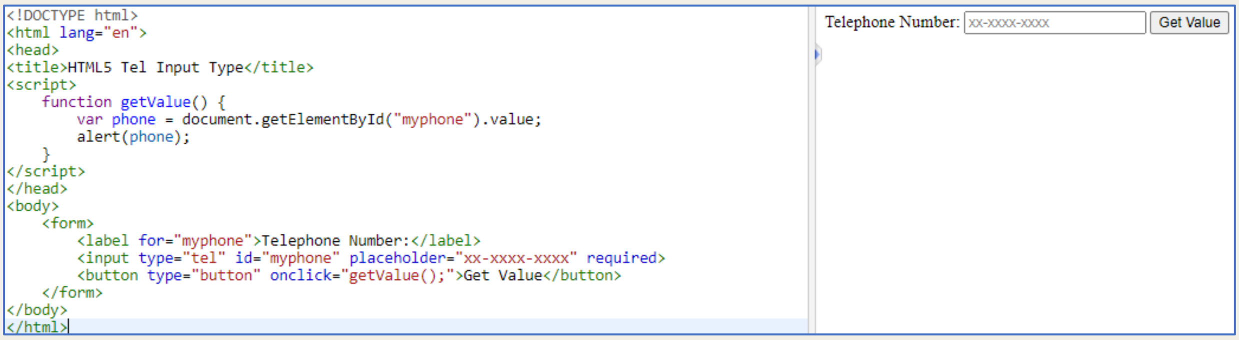
The search input type is suitable for generating search input fields. It must be stated that, in some browsers (i.e., Chrome and Safari), as soon as the user begins to type in the search box, a tiny cross emerges on the right side of the field, which can be useful for clearing the entire search field. It is supported by all major browsers.



```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Search Input Type</title>
<script>
    function getValue() {
        var term = document.getElementById("mysearch").value;
        alert(term);
    }
</script>
</head>
<body>
    <form>
        <label for="mysearch">Search Website:</label>
        <input type="search" id="mysearch" placeholder="Type something...">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```

***Figure 80** – The search input type (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

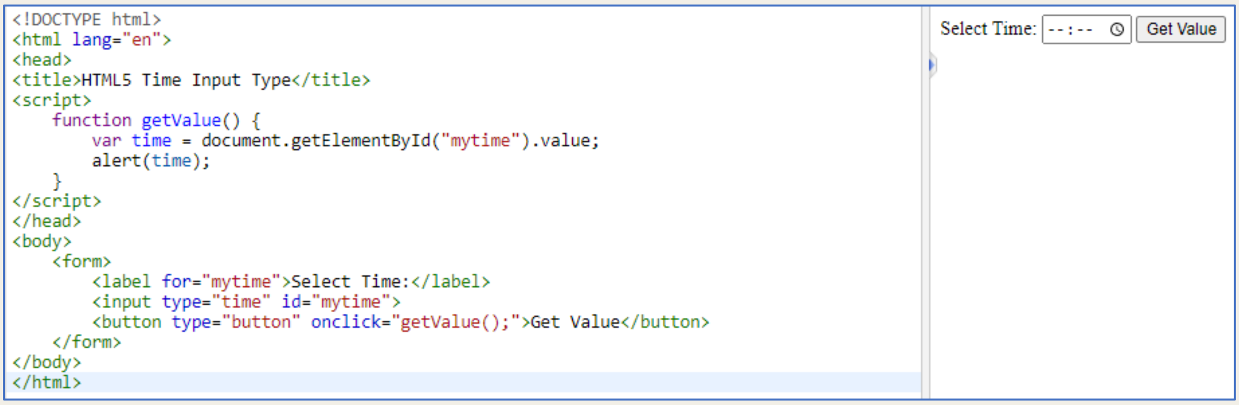
Concerning the tel input type, it is especially useful for inserting a phone number. As browsers do not support tel input validation by default, the placeholder attribute can be used to help users inserting the correct format for their phone number or indicate a regular expression to validate the user's input by applying the pattern attribute. This feature is not supported by any browser, as phone numbers vary a lot worldwide.



```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Tel Input Type</title>
<script>
    function getValue() {
        var phone = document.getElementById("myphone").value;
        alert(phone);
    }
</script>
</head>
<body>
    <form>
        <label for="myphone">Telephone Number:</label>
        <input type="tel" id="myphone" placeholder="xx-xxxx-xxxx" required>
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```

Figure 81 *– The tel input type (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

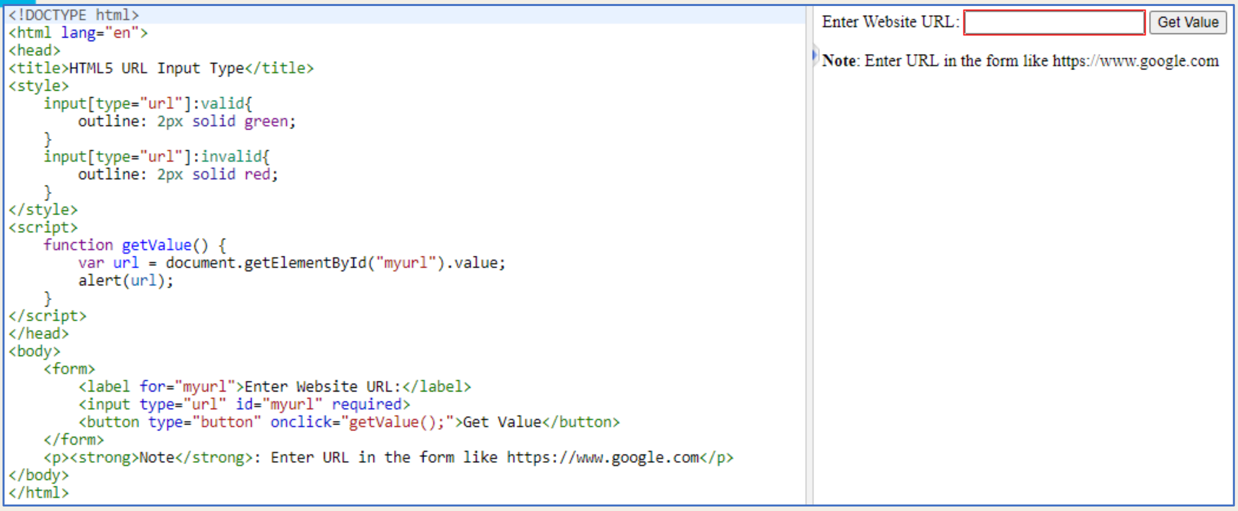
Regarding the time input type, it can be used for entering any given time (hours and minutes), and the browser can use both hour formats (12 or 24-hour) for inserting times, depending on the region. This input is not supported by IE and Safari browsers.



```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Time Input Type</title>
<script>
    function getValue() {
        var time = document.getElementById("mytime").value;
        alert(time);
    }
</script>
</head>
<body>
    <form>
        <label for="mytime">Select Time:</label>
        <input type="time" id="mytime">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```

Figure 82 *– The time input type (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Concerning the url input type, it can be used for inserting URL's or web addresses. The multiple attribute can be used for inserting more than one URL. Moreover, if required attribute is restricted, the browser will automatically perform validation to ensure that only text that meets the standard format for URLs goes into the input box. All major browsers support this input type.

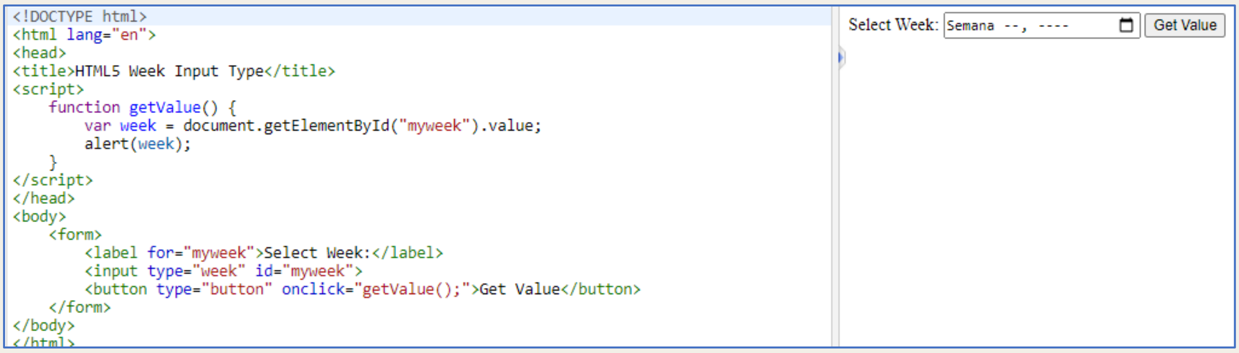


```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 URL Input Type</title>
<style>
    input[type="url"]:valid{
        outline: 2px solid green;
    }
    input[type="url"]:invalid{
        outline: 2px solid red;
    }
</style>
<script>
    function getValue() {
        var url = document.getElementById("myurl").value;
        alert(url);
    }
</script>
</head>
<body>
    <form>
        <label for="myurl">Enter Website URL:</label>
        <input type="url" id="myurl" required>
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
    <p><strong>Note</strong>: Enter URL in the form like https://www.google.com</p>
</body>
</html>
```

Note: Enter URL in the form like https://www.google.com

***Figure 83** – The url input type (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Finally, the week input type enables the user to choose a week and year from a drop-down calendar. This feature is not supported by Firefox, Safari and IE, but it is currently supported by Chrome, Edge and Opera browsers.



```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Week Input Type</title>
<script>
    function getValue() {
        var week = document.getElementById("myweek").value;
        alert(week);
    }
</script>
</head>
<body>
    <form>
        <label for="myweek">Select Week:</label>
        <input type="week" id="myweek">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```

***Figure 84** – The week input type (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

HTML5 Canvas

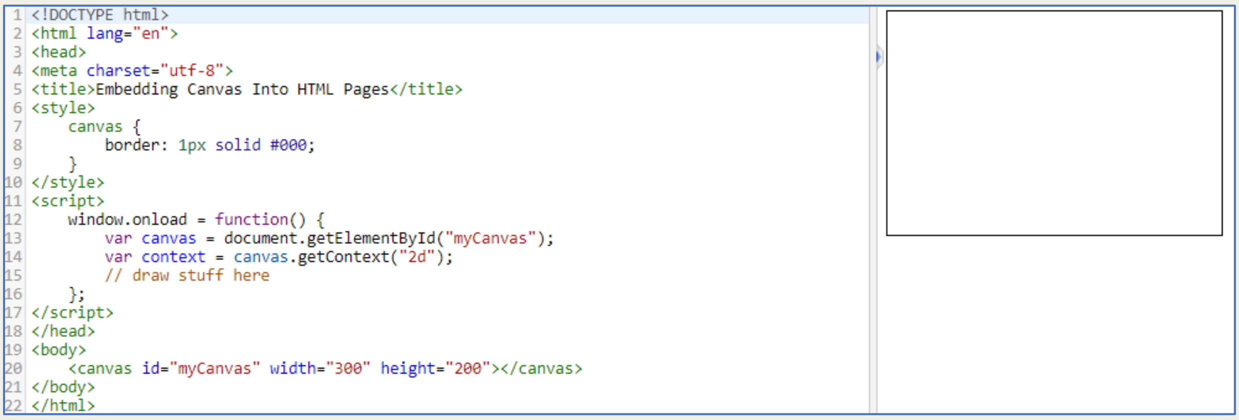
This subchapter will be essentially useful for learning how to draw graphics using the HTML5 canvas element. It was originally created by Apple for the Mac OS dashboard widgets and to enable graphics in Safari. Moreover, it was adopted by other browsers, like Firefox, Google Chrome and Opera.

By default, the <canvas> element has 300px of width and 150px of height without any border and content. Nevertheless, custom width and height can be specified using the CSS height and width feature.

The canvas is a two-dimensional four-sided area. The coordinates of the top-left corner of the canvas are (0, 0), identified as origin, and the coordinates of the bottom-right corner are (*canvas width, canvas height*), as can be seen using the interactive tool available here.

To draw basic paths and shapes utilizing the recently introduced HTML5 canvas element and JavaScript, it will be useful to take a look at several templates.

Firstly, the base template for drawing paths and shapes onto the 2D HTML5 canvas:

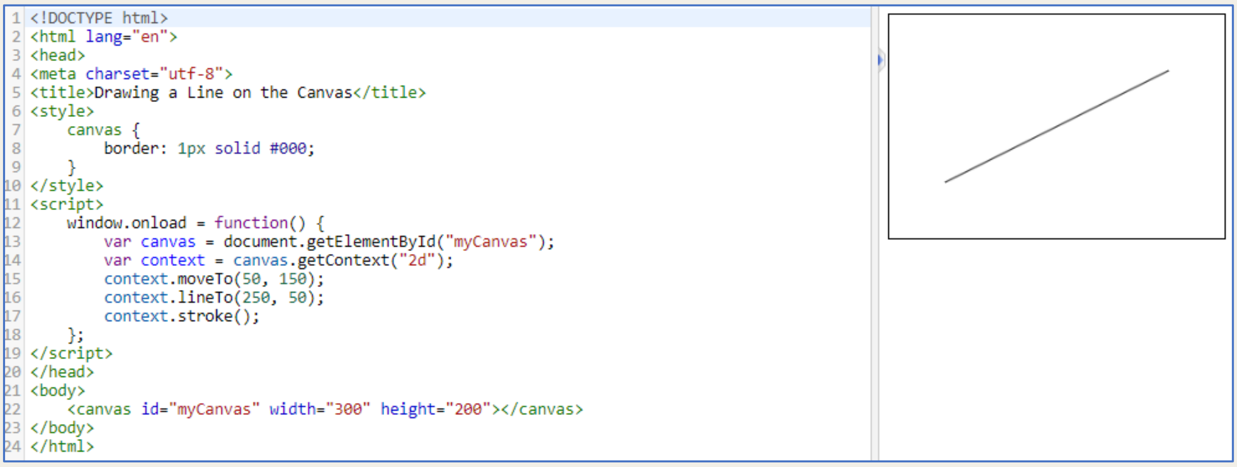


```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Embedding Canvas Into HTML Pages</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        // draw stuff here
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Figure 85 *– The base template for drawing paths and shapes onto the 2D HTML5 canvas (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

All the lines except those from 7 to 11 are quite straight forward and easy to interpret. The anonymous function affixed to the window.onload event will execute when the page loads. As soon as the page is loaded, one can access the canvas element with document.getElementById() method. Later, a 2D canvas context is defined by passing 2d into the getContext() method of the canvas object.

The initial step to draw on canvas is a **straight line**. The most important procedures used for this purpose are moveTo(), lineTo() and the stroke(). The moveTo() method identifies the position of drawing cursor onto the canvas, while the lineTo() method used to define the coordinates of the line's end point, and finally the stroke() method is used to make the line visible:



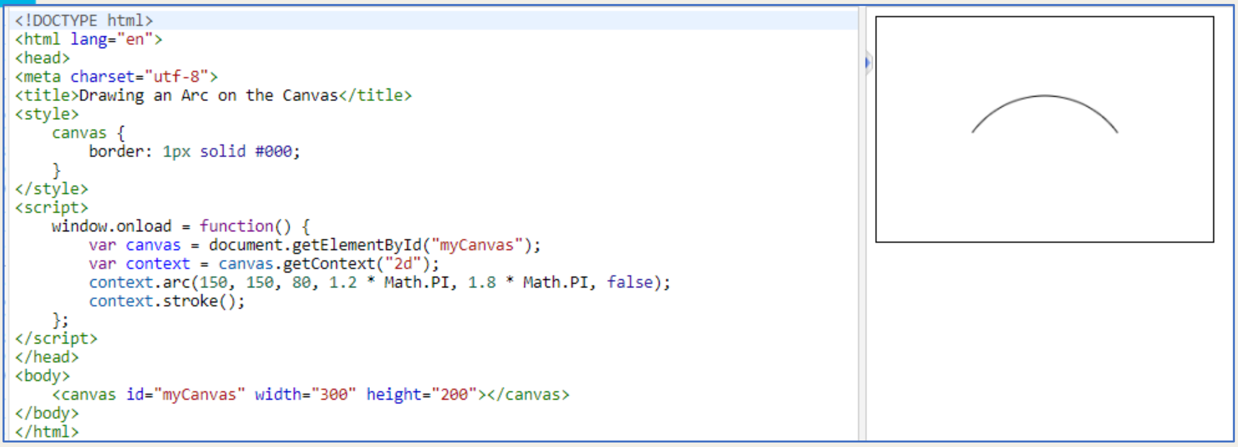
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Drawing a Line on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.moveTo(50, 150);
        context.lineTo(250, 50);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

***Figure 86** – The moveTo(), lineTo() and the stroke() procedures (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

What about **drawing an arc**? It can be obtained by simply using the arc() method, which syntax is:

```
context.arc(centerX, centerY, radius, startingAngle, endingAngle,
counterclockwise);
```

In the example above, an arc was drawn on canvas by inserting a JavaScript code:



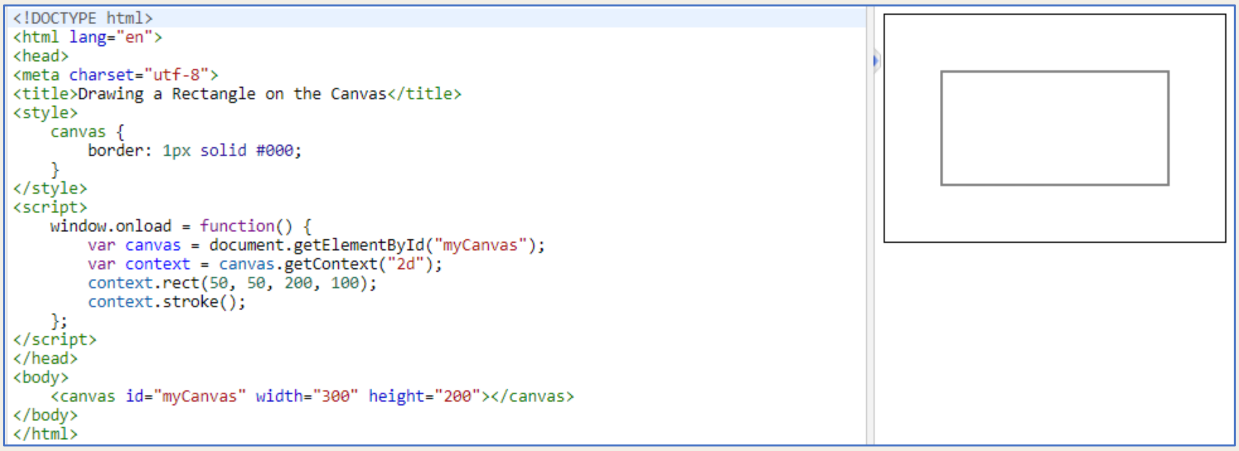
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Drawing an Arc on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.arc(150, 150, 80, 1.2 * Math.PI, 1.8 * Math.PI, false);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

***Figure 87** – Drawing an arc using a JS code (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

To draw a **rectangle and square shapes**, the rect() method is the way to go. It entails four parameters: x, y positions of the rectangle and its width and height. The basic syntax of the rect() method is the following:

```
context.rect(x, y, width, height);
```

To draw it using a JS code:

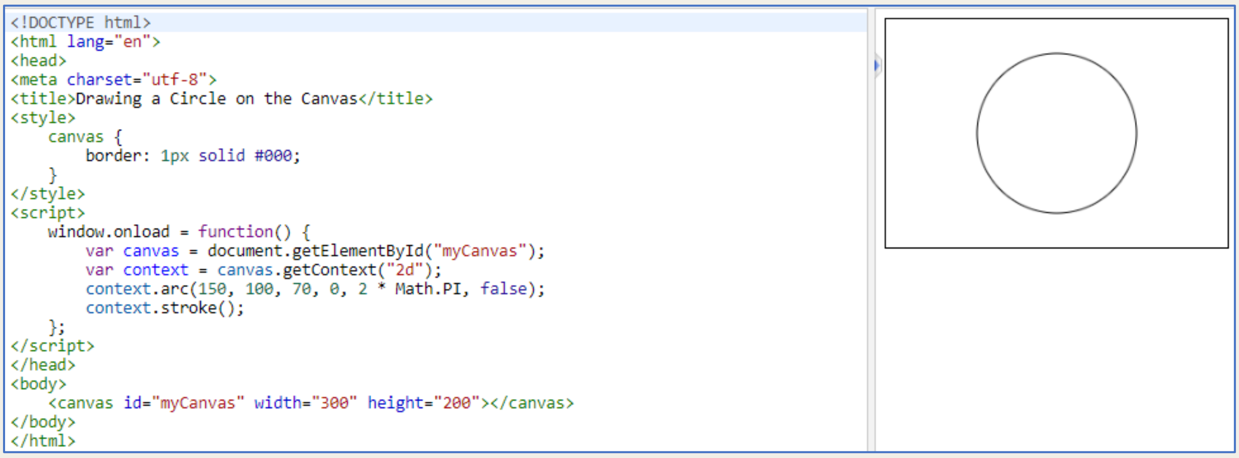


```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Drawing a Rectangle on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.rect(50, 50, 200, 100);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

***Figure 88** – Drawing a rectangle using a JS code (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

In opposite to the rect() method, there is no specific procedure for drawing a circle. However, the result can be obtained by creating a fully enclosed arc, by using the arc() method. The synthax for drawing a complete circle using the arc() method is the following:

```
context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);
```



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Drawing a Circle on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.arc(150, 100, 70, 0, 2 * Math.PI, false);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

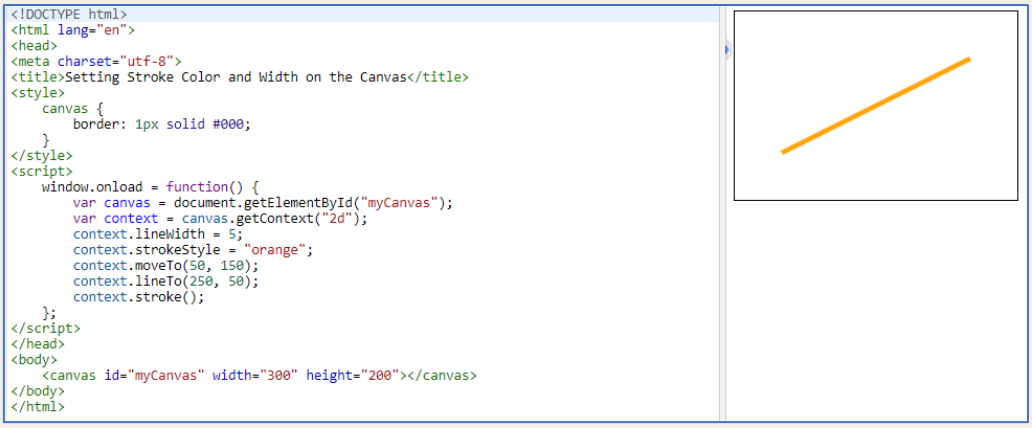
Figure 89 – Drawing a circle on HTML5 canvas (Source: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)

In regard to **styles and colours on stroke**, the default colour of the stroke is black, being its thickness 1 pixel. However, these attributes can be changed using the strokeStyle and lineWidth properties, as follows on *Figure 86*.

In *Figure 87*, it is possible to set the cap style for the lines by using the lineCap property, with three styles available: butt, round, and square.

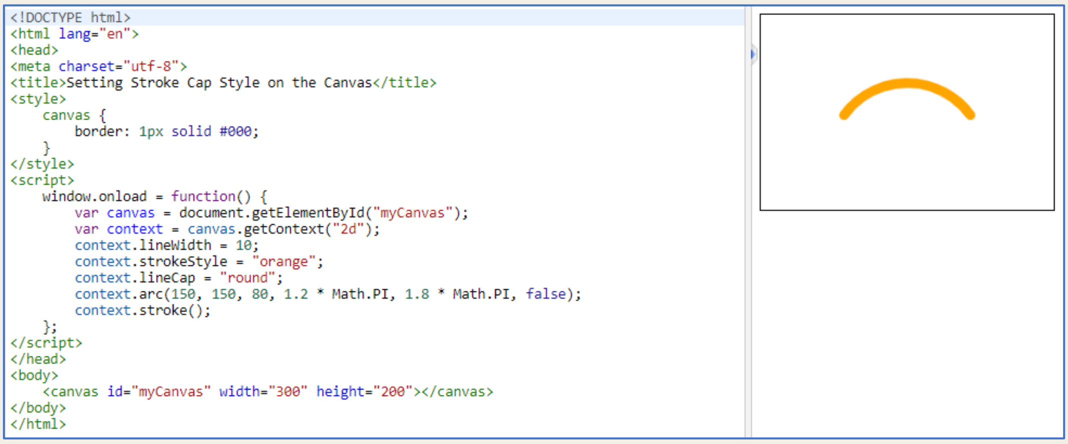
One can also fill colour inside the canvas shapes by using the fillStyle() approach. *Figure 88* shows how to fill up a solid colour within a rectangle shape. While designing the shapes on canvas, it is suggested to use the fill() method before the stroke() method to render the stroke appropriately.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



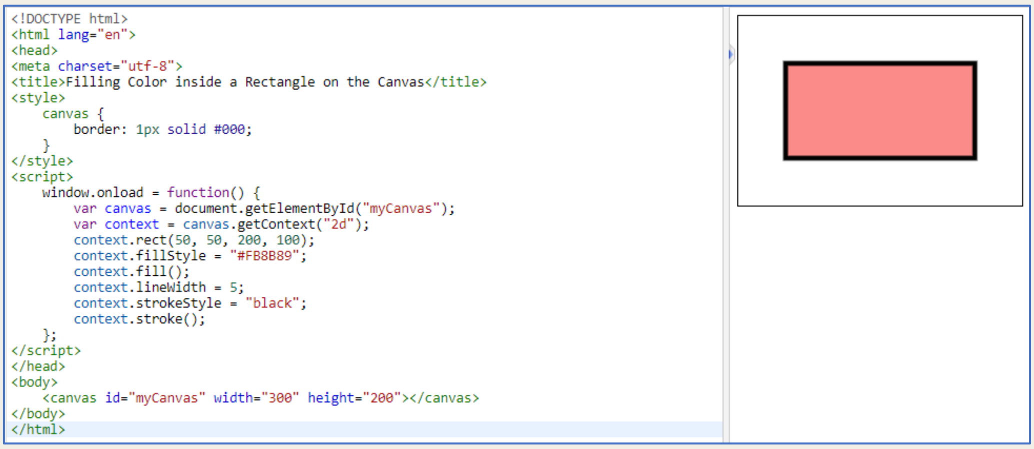
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Setting Stroke Color and Width on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.lineWidth = 5;
        context.strokeStyle = "orange";
        context.moveTo(50, 150);
        context.lineTo(250, 50);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

***Figure 90** – Setting the styles and colours on stroke using strokeStyle and lineWidth properties (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Setting Stroke Cap Style on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.lineWidth = 10;
        context.strokeStyle = "orange";
        context.lineCap = "round";
        context.arc(150, 150, 80, 1.2 * Math.PI, 1.8 * Math.PI, false);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

***Figure 91** – Setting the cap style for the lines using the lineCap property (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Filling Color inside a Rectangle on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.rect(50, 50, 200, 100);
        context.fillStyle = "#FB8B89";
        context.fill();
        context.lineWidth = 5;
        context.strokeStyle = "black";
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

***Figure 92** – Filing colour inside the canvas shapes by using the fillStyle() approach (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

It is also possible to fill gradient colour (a smooth visual transition from one colour to another) within the canvas shapes. There are two types of gradients here: linear and radial.

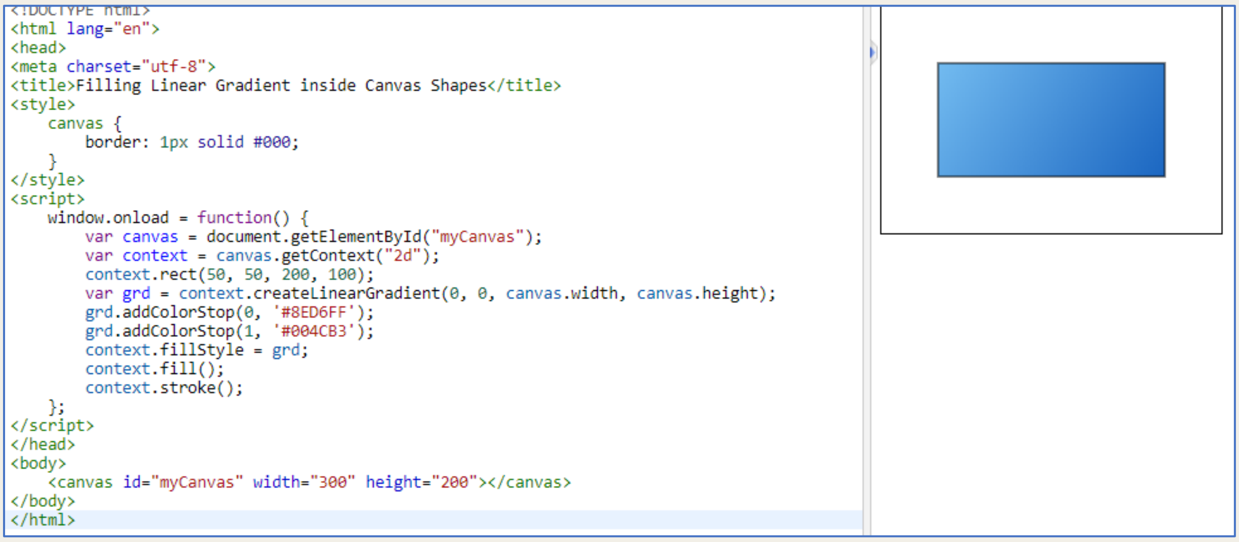
The basic syntax for creating a **linear gradient** is:

```
var grd = context.createLinearGradient(startX, startY, endX, endY);
```

The basic syntax for creating a **radial gradient** is:

```
var grd = context.createRadialGradient(startX, startY, startRadius,
endX, endY, endRadius);
```

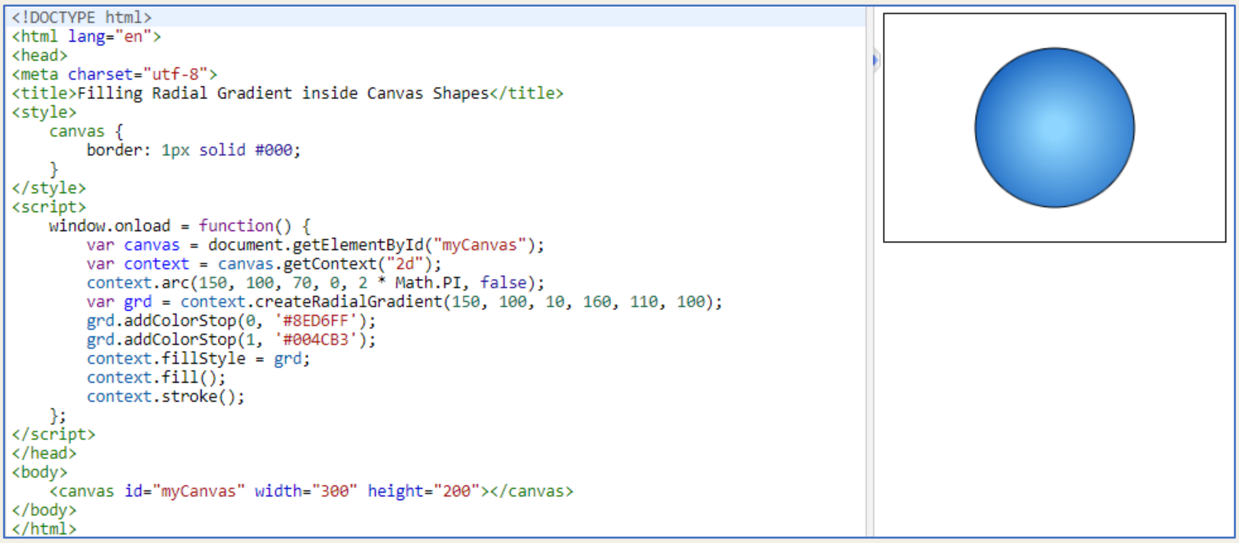
Figure 89 shows how to fill a **linear gradient** colour inside a rectangle using the createLinearGradient() method:



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Filling Linear Gradient inside Canvas Shapes</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.rect(50, 50, 200, 100);
        var grd = context.createLinearGradient(0, 0, canvas.width, canvas.height);
        grd.addColorStop(0, '#8ED6FF');
        grd.addColorStop(1, '#004CB3');
        context.fillStyle = grd;
        context.fill();
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Figure 93 *– Filing a linear gradient colour inside a rectangle using the createLinearGradient() method (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Figure 90 demonstrates how to fill a radial gradient colour inside a circle through the createRadialGradient() method:



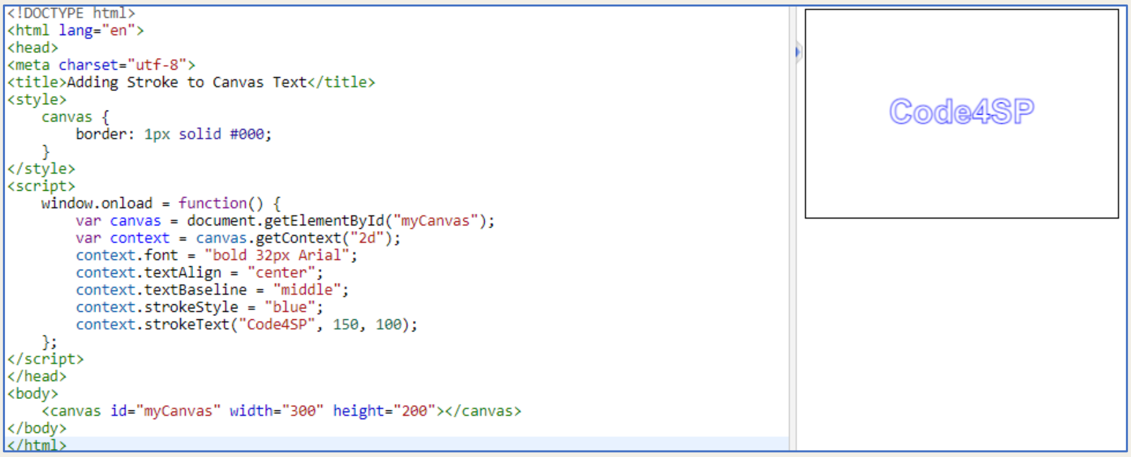
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Filling Radial Gradient inside Canvas Shapes</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.arc(150, 100, 70, 0, 2 * Math.PI, false);
        var grd = context.createRadialGradient(150, 100, 10, 160, 110, 100);
        grd.addColorStop(0, '#8ED6FF');
        grd.addColorStop(1, '#004CB3');
        context.fillStyle = grd;
        context.fill();
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

***Figure 94** – Filling a radial gradient colour inside a circle through the createRadialGradient() method (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

It is also possible to **draw text** on canvas (containing Unicode characters only), as well as adding it **colour and alignment**, and even apply stroke on text using the strokeText() method, which will colour the perimeter of the text instead of filling it. Nonetheless, if the web designer intends to set both the fill and stroke on the text element, he/she can use the fillText() and the strokeText() methods together. it is suggested to use the fillText() method before the strokeText() method to render the stroke accurately.

spel CITIZENS IN POWER Center for Social Innovation CDDR ZAUG gGmbH 4Action social hackers academy EPE

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Adding Stroke to Canvas Text</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.font = "bold 32px Arial";
        context.textAlign = "center";
        context.textBaseline = "middle";
        context.strokeStyle = "blue";
        context.strokeText("Code4SP", 150, 100);
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

***Figure 95** – Drawing text on canvas (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

HTML5 SVG

This subchapter is expected to be clear on how to use HTML5 svg elements to draw vector graphics on a web page. To do so, firstly it is important to define **SVG**.

SVG stands for Scalable Vector Graphics, and it is an XML-based image format that is used to define two-dimensional vector-based graphics for the web. Distinct from raster image (e.g. .jpg, .gif, .png, and other two-dimensional formats), a vector image can be scaled up or down to any extent without losing the quality of the image. Vector images are comprised of a series of shapes defined by math, while raster images are composed of a fixed set of dots (pixels). Like other topics discussed along this chapter, SVG is also a W3C recommendation.

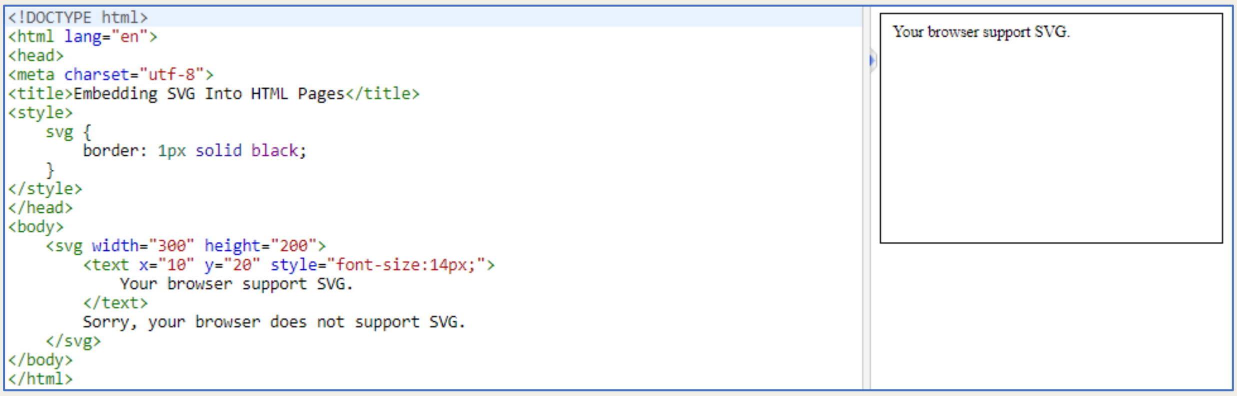
An SVG image is built by using a sequence of statements that go along with the XML schema, so, **SVG images** can be created and edited with a text editor like Notepad. There are numerous advantages of using SVG images rather than other image formats, as follows:

- They can be searched, indexed, scripted, and compressed.
- They can be created and modified using JavaScript in real time.
- They can be printed with high quality at any resolution.

- They can be animated using the built-in animation elements.
- They can contain hyperlinks to other documents.

SVG graphics can be embedded directly in a document by using the HTML5 <svg> element (see *Figure 96* below).

All the major modern web browsers (Chrome, Firefox, Safari, and Opera), as well as Internet Explorer 9 and above are compatible with inline SVG rendering.



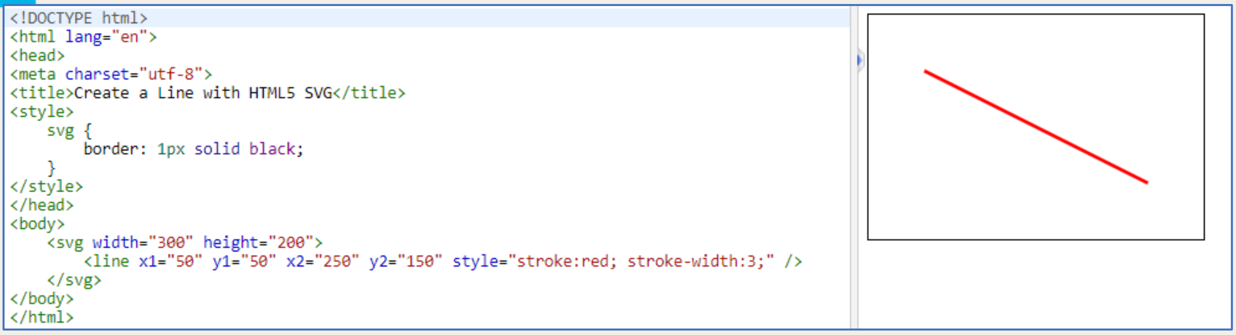
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Embedding SVG Into HTML Pages</title>
<style>
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <text x="10" y="20" style="font-size:14px;">
            Your browser support SVG.
        </text>
        Sorry, your browser does not support SVG.
    </svg>
</body>
</html>
```

Your browser support SVG.

Figure 96 *– Embedding SVG graphics directly by using the svg element (**Source:*** *https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)*

Basic vector-based paths and shapes on the webpages can be drawn by utilizing the HTML5 <svg> element.

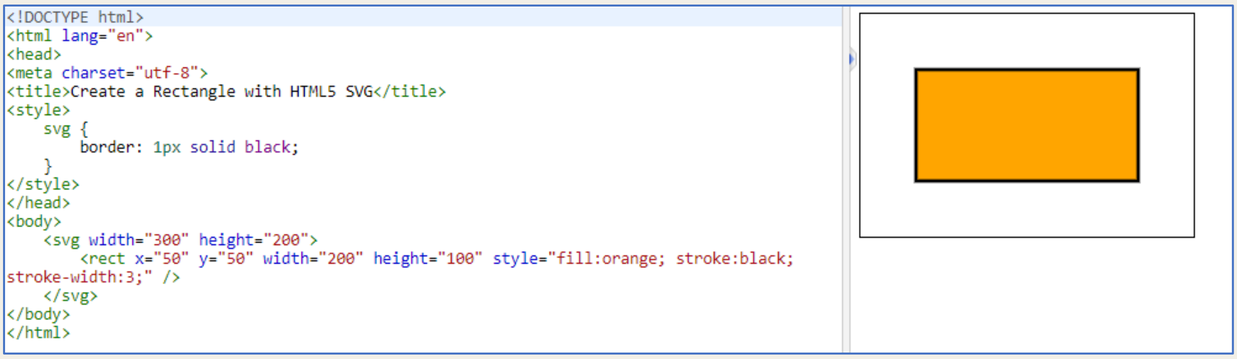
The most basic path to work with SVG is to **draw a straight line**. For that to happen, the SVG <line> element should be used. As can be seen in *Figure 93*, the attributes x1, x2, y1 and y2 of the <line> element draw a line from (x1,y1) to (x2,y2).



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Create a Line with HTML5 SVG</title>
<style>
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <line x1="50" y1="50" x2="250" y2="150" style="stroke:red; stroke-width:3;" />
    </svg>
</body>
</html>
```

Figure 97 *– Drawing a straight line with SVG <line> element (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)*

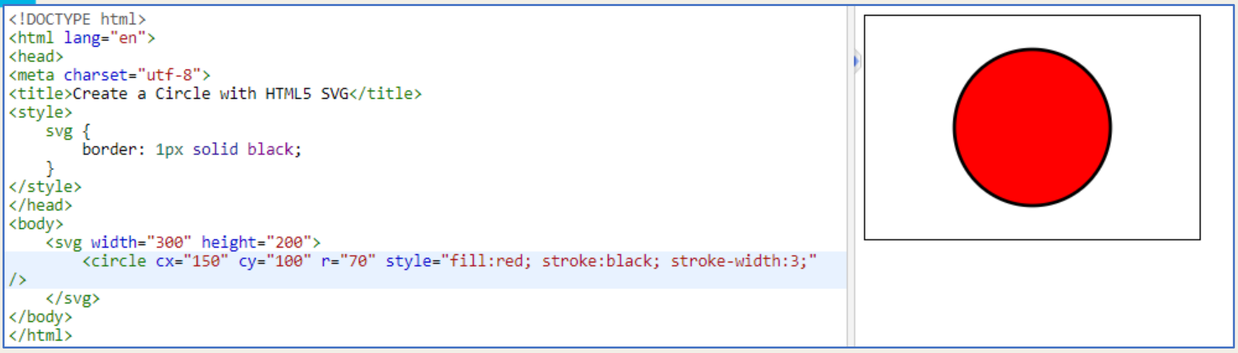
For drawing a **rectangle** and squares, the <rect> SVG element is the most appropriate way. The attributes x and y of <rect> element specify the co-ordinates of the top-left corner of the rectangle. The attributes width and height specify the width and height of the shape.



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Create a Rectangle with HTML5 SVG</title>
<style>
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <rect x="50" y="50" width="200" height="100" style="fill:orange; stroke:black;
stroke-width:3;" />
    </svg>
</body>
</html>
```

Figure 98 *– Drawing a rectangle with SVG <rect> element (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)*

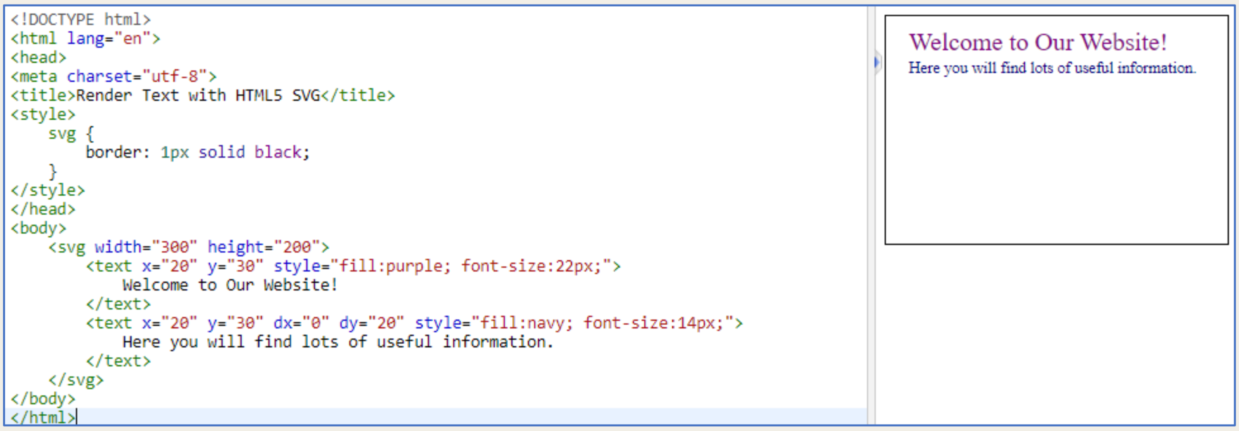
If a **circle** is the chosen shape, the SVG element <circle> is the most suitable. The attributes cx and cy of the <circle> element specifies the co-ordinates of the center of the circle and the attribute r identifies the radius of the circle. Still, if the attributes cx and cy are absent or not specified, the center of the circle is set to (0,0).



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Create a Circle with HTML5 SVG</title>
<style>
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <circle cx="150" cy="100" r="70" style="fill:red; stroke:black; stroke-width:3;"
/>
    </svg>
</body>
</html>
```

__Figure 99__ – Drawing a circle with SVG <circle> element (__Source:__ https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)

It is also possible to **draw text** with SVG. The text in SVG is rendered as a graphic so the web designer can use all the graphic transformation to it, but it still performs as text, so it can be selected and copied as text by the user. The attributes x and y of the <text> element identify the location of the top-left corner in absolute terms although the attributes dx and dy indicates the relative location.

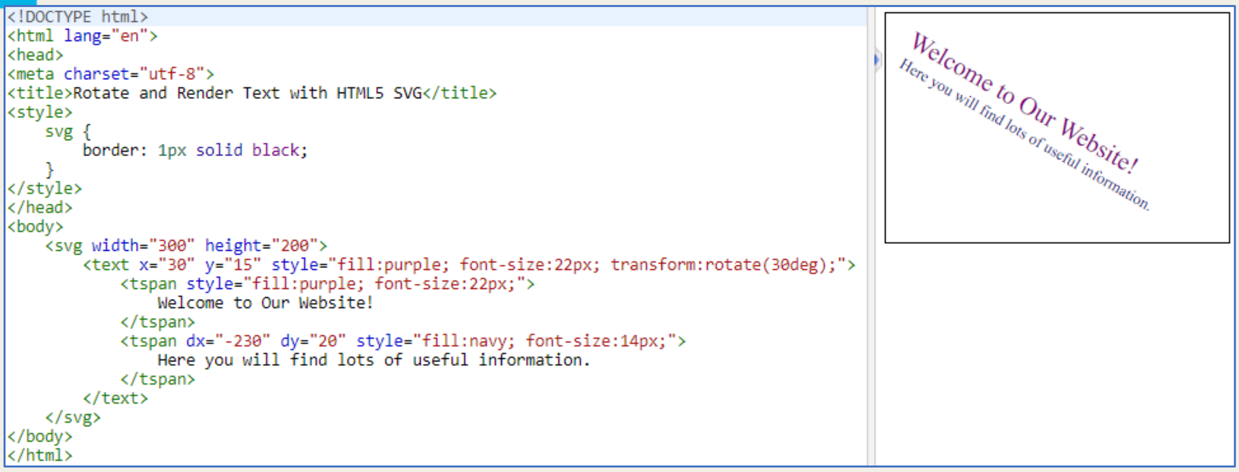


```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Render Text with HTML5 SVG</title>
<style>
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <text x="20" y="30" style="fill:purple; font-size:22px;">
            Welcome to Our Website!
        </text>
        <text x="20" y="30" dx="0" dy="20" style="fill:navy; font-size:14px;">
            Here you will find lots of useful information.
        </text>
    </svg>
</body>
</html>
```

Welcome to Our Website!
Here you will find lots of useful information.

__Figure 100__ – Drawing text with SVG <text> element (__Source:__ https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)

In alternative, web designers may use the <tspan> element to resize or relocate the span of text included within a <text> element. The text is included in separate tspans, but inside the same text element can all be selected at the same moment — when clicking and dragging to select the text. Yet, the text in separate text elements cannot be picked at the same time.



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Rotate and Render Text with HTML5 SVG</title>
<style>
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <text x="30" y="15" style="fill:purple; font-size:22px; transform:rotate(30deg);">
            <tspan style="fill:purple; font-size:22px;">
                Welcome to Our Website!
            </tspan>
            <tspan dx="-230" dy="20" style="fill:navy; font-size:14px;">
                Here you will find lots of useful information.
            </tspan>
        </text>
    </svg>
</body>
</html>
```

***Figure 101** – Drawing text with SVG <tspan> element (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)*

Even though the new graphical elements <canvas> and <svg> have been introduced by HTML 5 in order to create high-quality graphics on the web, they differ quite a lot. In the table below, the differences between both are summarized, and this will help learners on how to use them in an appropriate, effective way:

SVG	Canvas
Vector based (composed of shapes)	Raster based (composed of pixel)
Multiple graphical elements, which become the part of the page's DOM tree	Single element similar to <img> in behavior. Canvas diagram can be saved to PNG or JPG format
Modified through script and CSS	Modified through script only
Good text rendering capabilities	Poor text rendering capabilities
Give better performance with smaller number of objects or larger surface, or both	Give better performance with larger number of objects or smaller surface, or both

spel CITIZENS IN POWER Center for Social Innovation CDDR ZAUG gGmbH 4Action social hackers academy EPE

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

SVG	Canvas
Better scalability. Can be printed with high quality at any resolution. Pixelation does not occur	Poor scalability. Not suitable for printing on higher resolution. Pixelation may occur

***Figure 102** – The differences between SVG and canvas (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)*

HTML 5 Audio

This subchapter is intended to explain how to embed audio in an HTML document.

As web browsers did not have a uniform standard for embedding media files as audio, it was not an easy task to perform in the past. However, there are many ways to embed sound in a webpage, from simply use a simple link to using the HTML5 <audio> element. This element provides a standard way to insert audio in web pages. Since the audio element is somewhat new, it runs in most of the modern web browsers.

There are many ways of inserting an audio into a HTML5 document. One of them is by using the browser's default group of controls, with one source defined by the src attribute, as it can be verified in this code, in which it is possible to hear a group of birds singing.

Another way can be achieved by using the <object> element, which is used for embedding different types of media files. This example embeds an audio file into a web page following the aforementioned method. It should be stated that the <object> element is not supported broadly and it depends on the type of the object that is being implanted. Other methods like HTML5 <audio> element or third-party HTML5 audio players could be a better option in a lot of cases.

Finally, the <embed> element can also be another way of inserting media in an HTML document, following this example. Even though the <embed> element is superbly supported in nowadays browsers and defined as standard in HTML5, the inserted audio might not be played due to absence of browser support for that file format or inaccessibility of plugins.

HTML5 Video

It is now time to learn how to embed video content into an HTML document.

Pretty much like sound, video contents were also difficult to insert into a web page, and for the same reason (web browsers did not have a uniform standard for defining embedded media files like video). In the next paragraphs, many ways of inserting these contents will be explained.

The newly introduced <video> element works in most of the modern browsers. This example explains how to simply insert a video into the HTML document, using the browser default set of controls, with one source defined by the src attribute.

The <object> element is also used to embed different types of media files. Following this example, one can understand how to embed a Flash video into a webpage (only browsers/applications supporting Flash will be able to play it). It should be noted that the <object> element is not supported extensively and depends a lot on the type of the object embedded. Other methods could be a better choice in many cases as, for instance, iPad and iPhone device cannot display Flash videos.

And what about **embedding YouTube videos**? That is actually the most simple and common way of embedding video files in webpages nowadays. The web designer must simply upload the video on YouTube and insert HTML code to display the video on his/her webpage. Here is a step-by-step mini guide:

Step 1 – Upload a video on YouTube.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Step 2 – After uploading a video on YouTube, the web designer should look for the 'Share' button, which is located below a video running on the platform's video player, just like as follows:

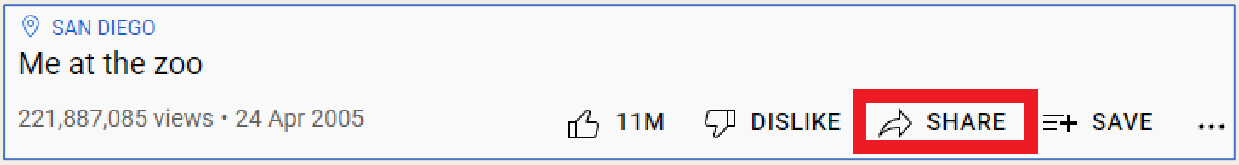
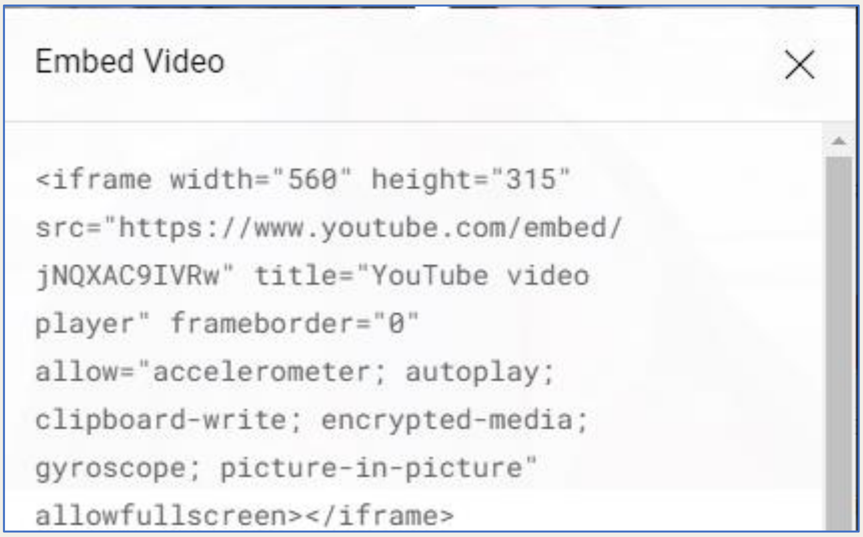


***Figure 103** – 'Share' button on YouTube (**Source:** Author)*

When clicking the 'Share' button, a share panel will open exhibiting some more options. Now, the '**Embed**' button should be clicked, as it will generate the HTML code to directly embed the video into the web page. For that to happen, the web designer should copy and paste that code into the HTML document.



```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/
jNQXAC9IVRw" title="YouTube video
player" frameborder="0"
allow="accelerometer; autoplay;
clipboard-write; encrypted-media;
gyroscope; picture-in-picture"
allowfullscreen></iframe>
```

***Figure 104** – 'Embed' option on YouTube (**Source:** Author)*

This code can be furtherly customized, and for that to happen the web designer just needs to select the customization option given just below the embed-code input box.

The insertion of a YouTube video on a webpage is explained in this example.

HTML5 Web Storage

Ever wondered how to use HTML5 web storage feature to store data on user's browser? The following paragraphs will be useful for fully understanding this.

Firstly, it is important to understand the implications of 'web storage'.

With web storage, web applications can store data locally within the user's browser. Prior to HTML5, application data had to be stored in cookies, incorporated in every server request. Web storage is more secure, and substantial amounts of data can be stored locally, without impacting website performance. The information kept in the web storage is not sent to the web server as opposite to the cookies where data is delivered to the server with every request. Moreover, cookies only allow to store a small amount of data (nearly 4KB), while the web storage permits to store up to 5MB.

There are two types of web storage:

- **Local storage** — makes use of the localStorage object to store data for the entire website on a *permanent basis*. That being said, the stored local data will be available on the next day, the next week, or the next year unless it is removed.
- **Session storage** — it uses the sessionStorage object to store data on a *temporary basis*, for a single browser window or tab. The data vanishes when session ends, for instance when the user shuts that browser window or tab.

As regards the **local storage**, each piece of data is collected in a key/value pair. The key identifies the name of the information (i.e. 'first_name'), and the value is the value related to the same key (i.e. 'Peter'). The following JS code expresses the following:

- localStorage.setItem(key, value) stores the value associated with a key.
- localStorage.getItem(key) saves the value associated with the key.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

It is also possible to remove a particular item from the storage, by passing the key name to the removeItem() method, i.e. localStorage.removeItem("first_name").

However, if the web designer intends to remove the complete storage, he/she should use the clear() method, i.e. localStorage.clear(). The clear() method simply clears all key/value pairs from localStorage at once, **so it must be used cautiously**. The web storage data will not be accessible between different browsers.

Finally, the sessionStorage object works similarly to localStorage, except that it stores the data only for one session. The following example is quite explanatory on how this works.

HTML5 Application Cache

During the subchapter, learners can get closer on how to create offline applications using **HTML5 caching feature**.

It is well-known that most web-based application will not work if the web designer is offline. However, HTML5 brings in an application cache mechanism that allows the browser to automatically save the HTML file and all the other resources that requires to display it properly on the local machine, that way the browser can still access the web page and its resources without establishing a connection to the internet. This is supported in all major modern web browsers (Firefox, Chrome, Opera, Safari, and Internet Explorer 10 and above.

There are several advantages in using this feature:

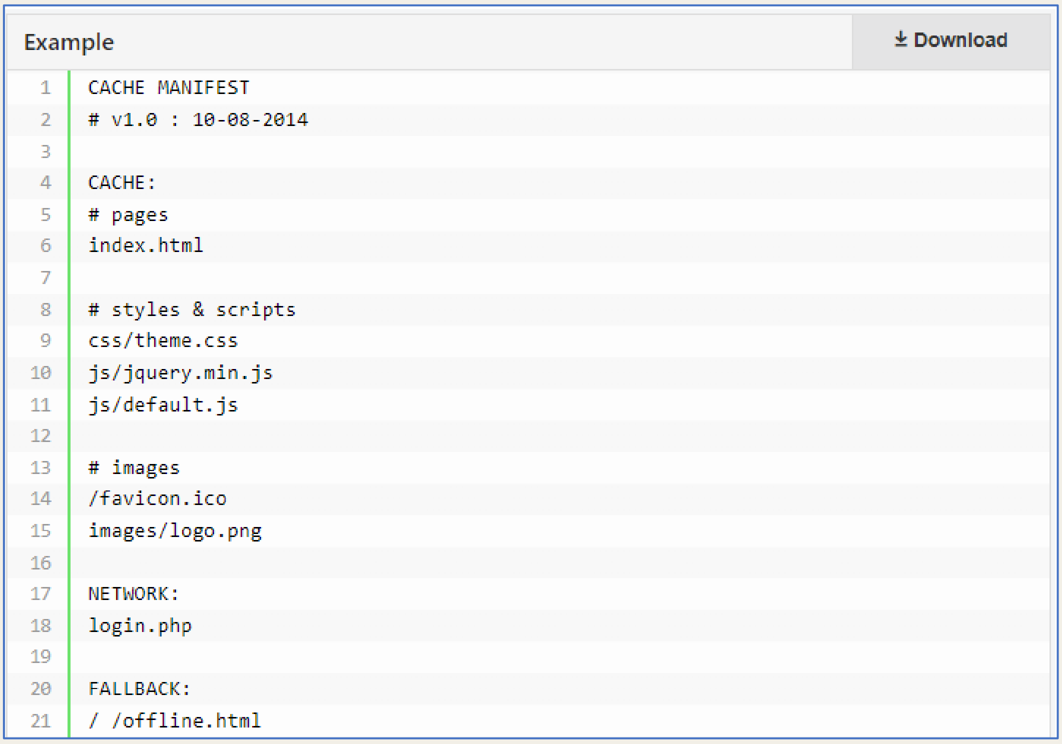
- **Offline browsing** — Visitors can use the application even when they are not online or there are unexpected interruptions in the network connection.

- **Improve performance** — Cached resources load directly from the user's machine instead of the remote server so web pages load quicker and perform better.
- **Reduce HTTP request and server load** — The browser will only have to download the updated/changed resources from the remote server that reduce the HTTP requests and saves valuable bandwidth as well as decrease the load on the web server.

There are a few steps to go through in order to cache the files for offline use:

STEP 1 – Create a Cache Manifest file. This is a special text file that informs browsers what files should they store (and not), and what files to replace. It always starts with the words CACHE MANIFEST (always in uppercase).

Figure 105 below is an example of a manifest file:



```
CACHE MANIFEST
# v1.0 : 10-08-2014

CACHE:
# pages
index.html

# styles & scripts
css/theme.css
js/jquery.min.js
js/default.js

# images
/favicon.ico
images/logo.png

NETWORK:
login.php

FALLBACK:
/ /offline.html
```

***Figure 105** – Example of a manifest file (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-application-cache.php)*

It is now time to explain the coding of *Figure 98*.

- Firstly, it is important to understand that manifest files can have three different sections: **CACHE, NETWORK**, and **FALLBACK**.
- The files listed under the CACHE: section header (or right after the CACHE MANIFEST line) are clearly cached after they are downloaded for the first time;
- Files under the NETWORK: section header are white-listed resources that are never cached and are only available online. It means users cannot never access login.php page when they're offline;
- The FALLBACK: section specifies alternative pages the browser should use in case the connection to the server cannot be done. Each entry in this section lists two URIs — first is the primary resource, the second is the fallback. For instance, in *Figure 98* case, offline.html page will be displayed if the user is offline. Also, both URIs must be from the same origin as the manifest file.
- It should be noted that lines starting with '#' (hash symbol) are comment lines.

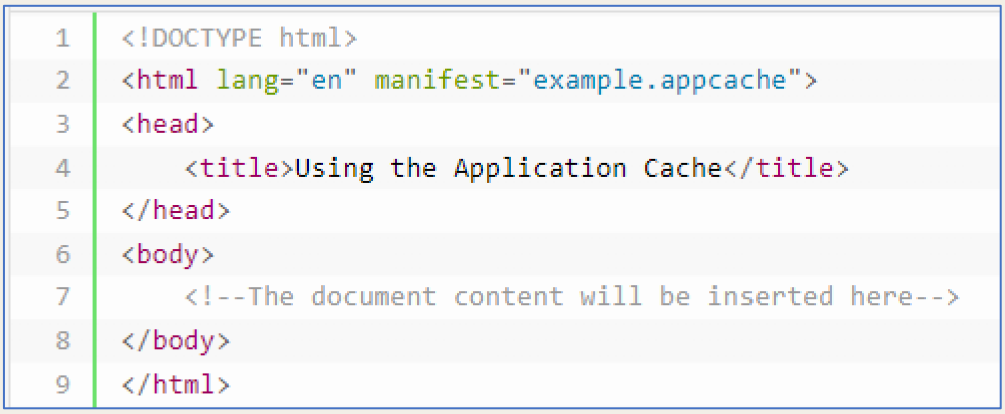
Therefore, if an application cache is there, the browser loads the document and its associated resources straight from the cache, without accessing the network. Then browser checks to find out whether the manifest file has been updated on the server. If it has been updated, the browser downloads the new version of the manifest and the resources listed in it.

It is important to note that the manifest file itself should not be specified in the cache manifest file. If so, it will be quite difficult to notify the browser that a new manifest is available.

STEP 2 – Use the cache manifest file. After creating it, the web designer should upload the cache manifest file on the web server — making sure the web server is configured to serve the manifest files with the MIME type text/cache-manifest.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

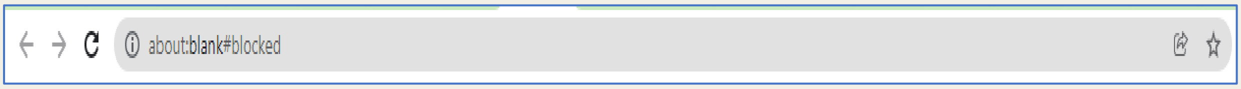
To make the cache manifest work, the web designer will need to enable it in the web pages, by adding the manifest attribute to the root <html> element, as shown by *Figure 106* below:



```
<!DOCTYPE html>
<html lang="en" manifest="example.appcache">
<head>
    <title>Using the Application Cache</title>
</head>
<body>
    <!--The document content will be inserted here-->
</body>
</html>
```

***Figure 106** – Making the cache manifest work (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-application-cache.php)*

If the user is online, the result for this code will be the following:



about:blank#blocked

***Figure 107** – about_blank#blocked (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-application-cache.php)*

HTML5 Web Workers

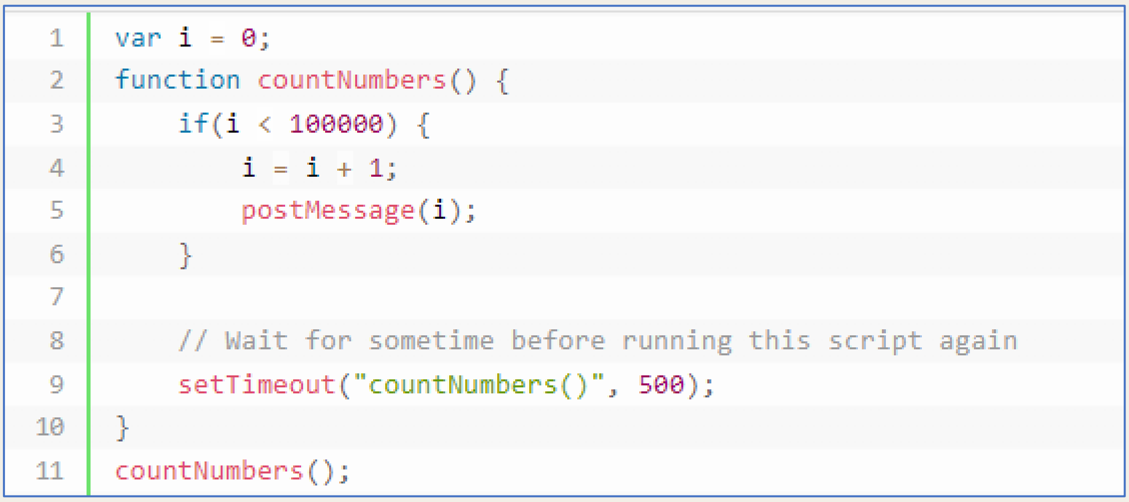
This subtopic will be essentially useful for further learning on JS topics, as it will teach how to use HTML5 web worker to run JS code in the background. So, learners must come back if they experience any difficulties.

If one tries to perform intensive, time-consuming, and heavy calculations demanding tasks with JavaScript, it probably will freeze up the webpages and will prevent users from doing anything until the job is done. Why? Well, because JS code always runs in

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

the foreground. However, HTML5 has a new technology (‘web worker’) created to perform background work apart from other user-interface scripts, without impacting the performance of the page. Distinct from normal JS operations, web worker does not interrupt the user and the web page stays responsive because they are running the tasks in the background.

Web workers are specially useful for performing a time-consuming task. So, in the first, example, a simple JS task that counts from zero to 100 000 will be created *(name of the file should be worker.js)*, as follows on *Figure 101:*

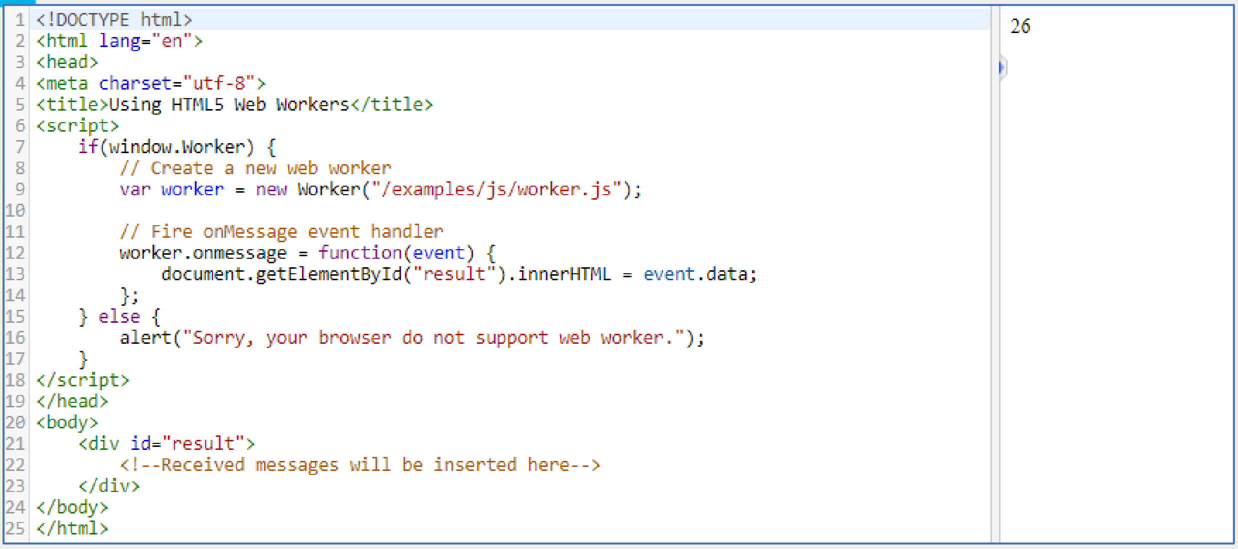


```
var i = 0;
function countNumbers() {
    if(i < 100000) {
        i = i + 1;
        postMessage(i);
    }

    // Wait for sometime before running this script again
    setTimeout("countNumbers()", 500);
}
countNumbers();
```

***Figure 108** – Creating a JS task counting from 0 to 100 000 (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-web-workers.php)*

Note: For a better understanding, it is advisable to download the code from Fig. 108 and follow every step of this chapter.

So, now that the web worker file has been created, it is time to start the web worker from an HTML document that runs the code inside the file named "worker.js" in the background and gradually shows the result on the web page. It should be noted that the number in the right will always be growing until it reaches 100 000.



```
 1 <!DOCTYPE html>
 2 <html lang="en">
 3 <head>
 4 <meta charset="utf-8">
 5 <title>Using HTML5 Web Workers</title>
 6 <script>
 7     if(window.Worker) {
 8         // Create a new web worker
 9         var worker = new Worker("/examples/js/worker.js");
10 
11         // Fire onMessage event handler
12         worker.onmessage = function(event) {
13             document.getElementById("result").innerHTML = event.data;
14         };
15     } else {
16         alert("Sorry, your browser do not support web worker.");
17     }
18 </script>
19 </head>
20 <body>
21     <div id="result">
22         <!--Received messages will be inserted here-->
23     </div>
24 </body>
25 </html>
```

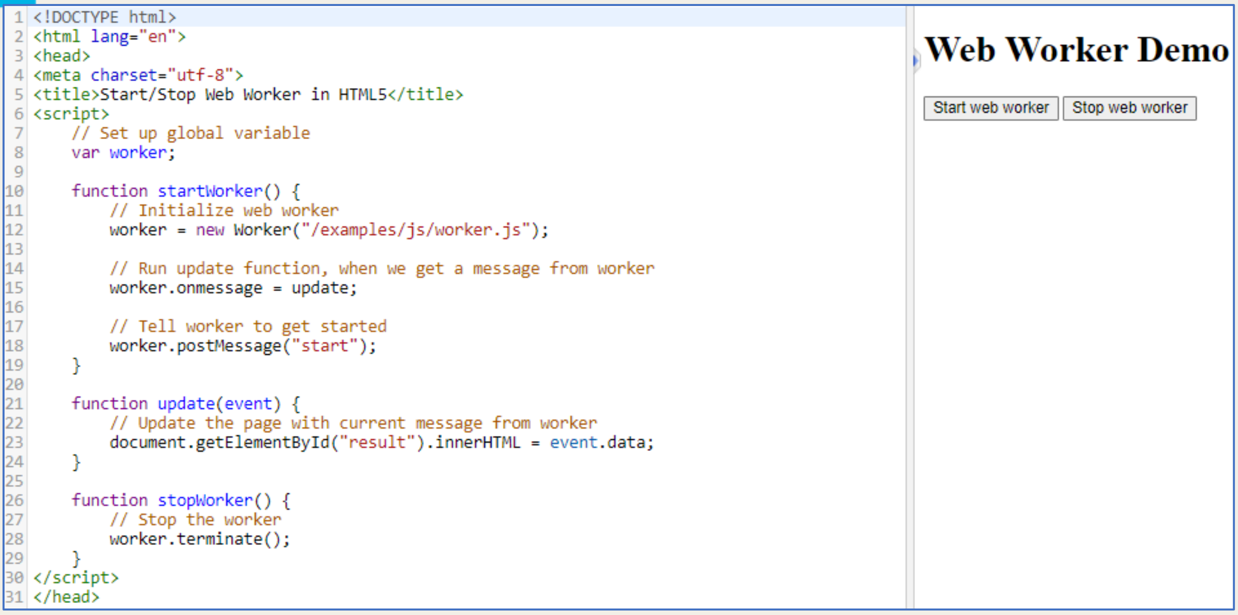
```
26
```

Figure 109 *– Starting the web worker (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-web-workers.php)*

Now explaining what is going on the example from above, the statement **var worker = new Worker("worker.js");** creates a new web worker object, which is used to communicate with the web worker. When the worker posts a message, it triggers the **onmessage** event handler (line 14) that permits the code to receive messages from the web worker. The **event.data** element comprises the message sent from the web worker. For the record, the code that a worker runs is always stored in a separate JavaScript file to prevent web developer from writing the web worker code that makes an attempt to use global variables or directly open the elements on the web page.

It is also possible to **put an end to a running worker** in the middle of the operation, following the example below:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Start/Stop Web Worker in HTML5</title>
<script>
    // Set up global variable
    var worker;

    function startWorker() {
        // Initialize web worker
        worker = new Worker("/examples/js/worker.js");

        // Run update function, when we get a message from worker
        worker.onmessage = update;

        // Tell worker to get started
        worker.postMessage("start");
    }

    function update(event) {
        // Update the page with current message from worker
        document.getElementById("result").innerHTML = event.data;
    }

    function stopWorker() {
        // Stop the worker
        worker.terminate();
    }
</script>
</head>
```

Web Worker Demo

***Figure 110** – Stopping the running worker (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-web-workers.php)*

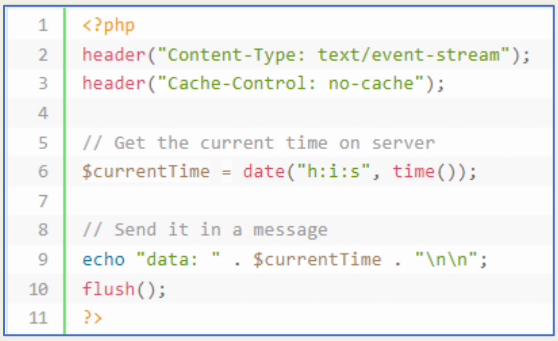
The example above shows how to start and stop worker from a web page by simply clicking on HTML buttons.

HTML5 Server-Sent Events

This subchapter will be useful for understanding how to use the HTML5 server-sent events feature to make a unidirectional and permanent connection between a web page and a server.

HTML5 server-sent event is an innovative way for the web pages to communicate with a web server. Nevertheless, there are some circumstances where web pages need a longer-term connection to the web server, for example, on stock quotes on finance websites where price updated automatically or game tickers running on various sports websites. Such things are feasible with the HTML5 server-sent events, as it makes available for a web page to hold an open connection to a web server, in a way that the web server can send a new response mechanically at whatever time. At this point, there is no need to reconnect and run the same server script from the beginning repeatedly.

For a better understanding of the aforementioned concepts, a PHP[1] file named "server_time.php" and type the following script into it. This file will merely report the present time of the web server's built-in clock in regular intervals:



```php
<?php
header("Content-Type: text/event-stream");
header("Cache-Control: no-cache");

// Get the current time on server
$currentTime = date("h:i:s", time());

// Send it in a message
echo "data: " . $currentTime . "\n\n";
flush();
?>
```

***Figure 111** – server_time.php example (Source https://www.tutorialrepublic.com/html-tutorial/html5-server-sent-events.php)*

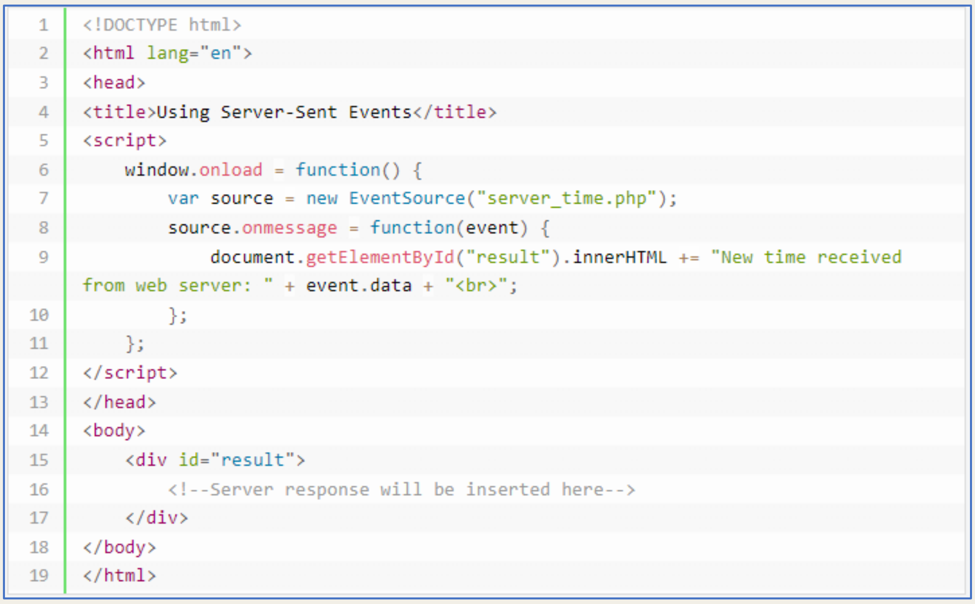
[1] A PHP file is a plain-text file which contains code written in the PHP programming language. Since PHP is a server-side (back-end) scripting language, the code written on it is executed on the server. In fact, a PHP file may contain plain text, HTML tags, or code as per the PHP syntax. PHP is often used to develop web applications that are processed by a PHP engine on the web server.

The first two line of the PHP script (*Fig. 111*) sets two crucial headers. Number one, it sets the MIME type to text/event-stream, which is needed by the server-side event standard. The second line informs the web server to turn off caching or else the output of the script may be cached.

Every message send through HTML5 server-sent events must start with the text data: followed by the actual message text and the new line character sequence (\n\n).

And lastly, the PHP flush() function has been used to ensure that the data is sent immediately, rather than buffered until the PHP code is complete.

Regarding on **how to process messages in a web page**, the EventSource object is used to receive server-sent event messages. In the example below, learners will see how a HTML document simply receives the current time reported by the web server and displays it to the web page visitors. For a better understanding, an HTML document named “demo_sse.htlm” will be created and furtherly placed in the same project directory where “server_time.php” is located. The download of the following code can be done in the source’s link available below.



```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Using Server-Sent Events</title>
<script>
    window.onload = function() {
        var source = new EventSource("server_time.php");
        source.onmessage = function(event) {
            document.getElementById("result").innerHTML += "New time received from web server: " + event.data + "<br>";
        };
    };
</script>
</head>
<body>
    <div id="result">
        <!--Server response will be inserted here-->
    </div>
</body>
</html>
```

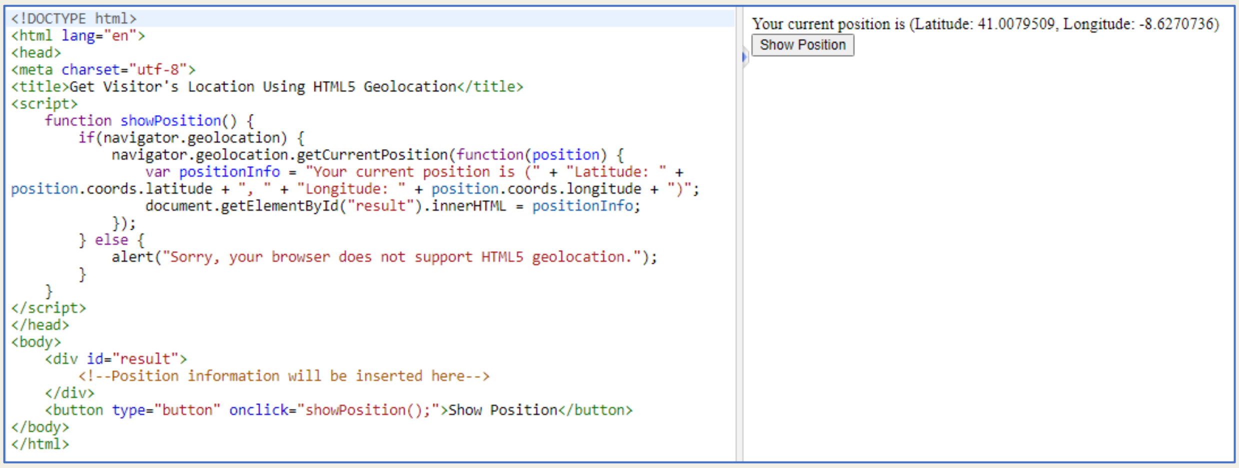
Figure 112 *– How to process messages in a web page (Source https://www.tutorialrepublic.com/html-tutorial/html5-server-sent-events.php)*

HTML 5 Geolocation

Through this subtopic, learners will get a few insights on how to use HTML5 geolocation feature for detecting user’s location. This feature lets the programmer find out the geographic coordinates (latitude and longitude) of the website visitor’s current location. It is especially useful for providing the best browsing experience for the visitor, since, for instance, this tool can show search results that are physically close to the user’s location.

Receiving the position information of the website visitor using the HTML5 geolocation API is not difficult. It exploits the three methods that are packed into the navigator.geolocation object — getCurrentPosition(), watchPosition() and clearWatch().

After the user agrees to let the browser tell the web server about his/her position (web browsers will not share the visitor location with a webpage unless the user agrees to do so), the geolocation process should occur as follows:

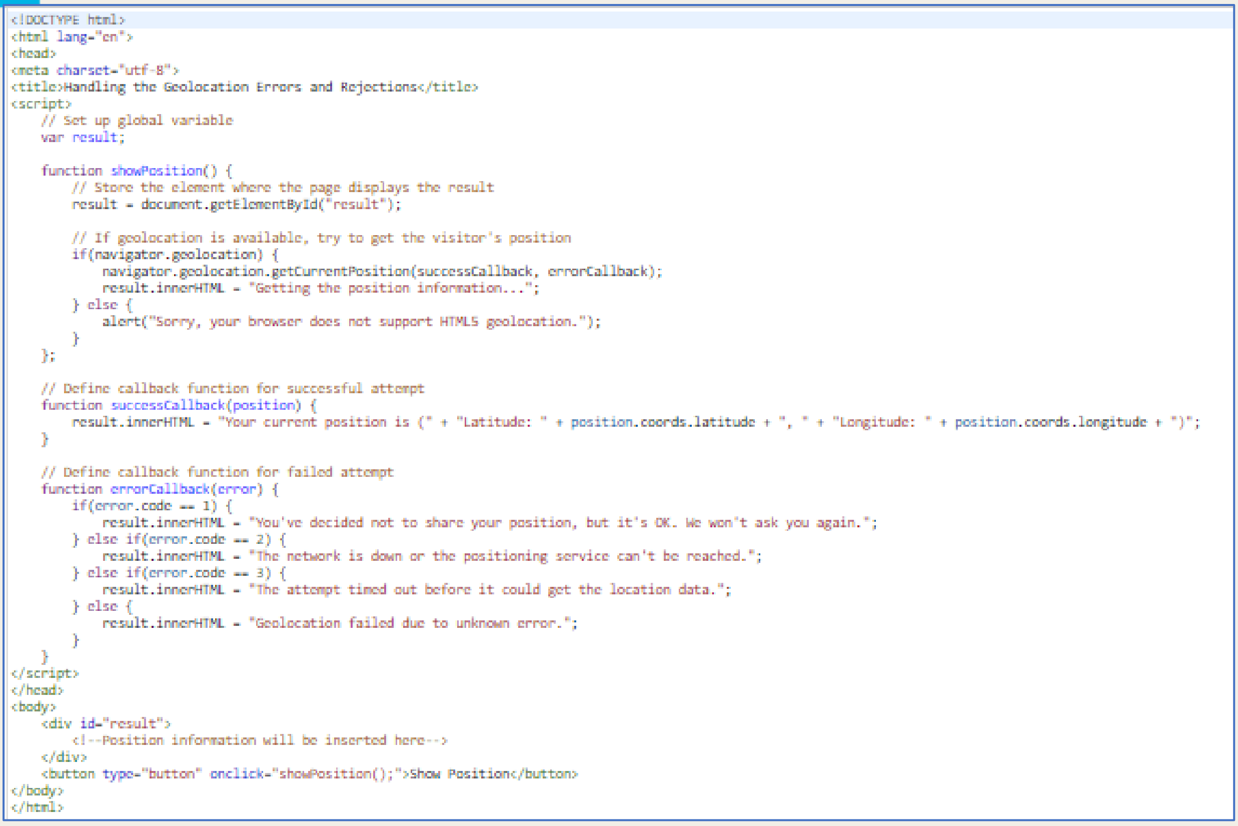


```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Get Visitor's Location Using HTML5 Geolocation</title>
<script>
    function showPosition() {
        if(navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(function(position) {
                var positionInfo = "Your current position is (" + "Latitude: " +
position.coords.latitude + ", " + "Longitude: " + position.coords.longitude + ")";
                document.getElementById("result").innerHTML = positionInfo;
            });
        } else {
            alert("Sorry, your browser does not support HTML5 geolocation.");
        }
    }
</script>
</head>
<body>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
    <button type="button" onclick="showPosition();">Show Position</button>
</body>
</html>
```

```
Your current position is (Latitude: 41.0079509, Longitude: -8.6270736)
```

Figure 113 – *The Geolocation feature process (****Source:*** *https://www.tutorialrepublic.com/html-tutorial/html5-geolocation.php)*

In case a user does not intend to share his/her location data with the website, the programmer can supply two functions when calling the getCurrentLocation() function. First function is called in case geolocation attempt is successful, whereas the second is called if the geolocation attempt fails.



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Handling the Geolocation Errors and Rejections</title>
<script>
    // Set up global variable
    var result;

    function showPosition() {
        // Store the element where the page displays the result
        result = document.getElementById("result");

        // If geolocation is available, try to get the visitor's position
        if(navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(successCallback, errorCallback);
            result.innerHTML = "Getting the position information...";
        } else {
            alert("Sorry, your browser does not support HTML5 geolocation.");
        }
    };

    // Define callback function for successful attempt
    function successCallback(position) {
        result.innerHTML = "Your current position is (" + "Latitude: " + position.coords.latitude + ", " + "Longitude: " + position.coords.longitude + ")";
    }

    // Define callback function for failed attempt
    function errorCallback(error) {
        if(error.code == 1) {
            result.innerHTML = "You've decided not to share your position, but it's OK. We won't ask you again.";
        } else if(error.code == 2) {
            result.innerHTML = "The network is down or the positioning service can't be reached.";
        } else if(error.code == 3) {
            result.innerHTML = "The attempt timed out before it could get the location data.";
        } else {
            result.innerHTML = "Geolocation failed due to unknown error.";
        }
    }
</script>
</head>
<body>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
    <button type="button" onclick="showPosition();">Show Position</button>
</body>
</html>
```

***Figure 114** – Applying two functions to the getCurrentLocation() function (**Source:** https://www.tutorialrepublic.com/html-tutorial/html5-geolocation.php)*

There are many interesting functions to explore with geolocation data, as showing the user's location on Google Maps. Based on the latitude and longitude data retrieved through the HTML5 geolocation feature, this example shows the reader's current location. This simply shows a static image showing user's location, although an interactive Google maps with dragging, zoom in/out and other features, as this example shows.

All the aforementioned examples have been based on the getCurrentPosition() method. Nevertheless, the geolocation function has another technique –watchPosition() that allows to track the visitor's movement by returning the updated position as the location changes. watchPosition() has the same input parameters as getCurrentPosition(). Yet, watchPosition() may activate the success function multiple times — when it gets the location for the first time, and again, every time it spots a new position, as this example suggests.

HTML5 Drag and Drop

It is a common procedure in the online daily routine to drag and drop an element to another location in a website. The HTML5 Drag and Drop feature allows to do so, and any element can be dragged and dropped. This w3schools example sets a simple drag and drop example learners may try to get more familiar with this concept. Even though the code seems difficult to understand, it is quite simple and logic:

- Firstly, to make an element draggable, the draggable attribute must be true:

```
<img draggable="true">
```

- Then, it should be specified what should happen once the element is dragged. In the w3schools example given above, the ondragstart attribute calls a function (drag (event)) that specifies what data will be dragged. The dataTransfer.setData() process sets the data type and the value of the dragged data:

```
function drag(ev) {
 ev.dataTransfer.setData("text",ev.target.id);
}
```

In this example, the data type is “text”, being the value the id of the draggable element (“drag1”).

- The ondragover event stipulates where the dragged data can be dropped. By default, data/elements are unable to be dropped in other elements. To permit a drop, the web designer should prevent the default handling of the element, by calling the event.preventDefault() technique for the ondragover event:

```
event.preventDefault()
```

- As soon as the dragged data is dropped, a drop event happens. In the example given previously, the ondrop attribute calls a function, drop(event):

```
function drop(ev) {
  ev.preventDefault();
  var data=ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
```

- ✓ Call preventDefault() to prevent the browser default handling of the data (default is open as a link on drop);
- ✓ The programmer gets the dragged data with the dataTransfer.getData() technique. This technique will return any data that was set to the same type in the setData() technique;
- ✓ The dragged data is the id of the dragged element ("drag1")
- ✓ The programmer should also attach the dragged element into the drop element.

2.4. HTML5 **References**

For a detailed, comprehensive list of elements concerning **HTML5 Tags/Elements, HTML5 Global Attributes**, **HTML5 Event Attributes, HTML5 Language Codes, HTML5 Character Entities**, **HTTP Status Codes, HTML5 Color Picker,** and other useful references, learners should refer to the following link (HTML5 References section).

3. CSS – Cascading Style Sheets

Topic Information

Topic:

3. CSS

Prerequisites:

Basic computer literacy, basic software installed, basic knowledge of working with files, and HTML basics (study Introduction to HTML.)

Workload:

10 hours.

Description:

Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML or XML (including XML dialects such as SVG, MathML or XHTML). CSS describes how elements should be rendered on screen, on paper, in speech, or on other media.

Learning outcomes:

Learners will get knowledge on how to define CSS, use basic CSS syntax, set up web pages with CSS, use CSS for styling text, font, and properties, use CSS for styling page backgrounds, style lists in CSS, use CSS classes and IDs, use borders and height and width CSS properties, use CSS pseudo elements, position elements with CSS and validate CSS and HTML.

Material required:

- Computer or laptop
- Internet connection
- Text editor (online or offline): Sublime Text/Brackets/W3Schools online editor

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Lesson Scenario:

The total time for this topic is 10 hours, and it will be up to the trainer/coach to decide how much time to dedicate to teaching each subtopic. To make the most of all the time available, we propose the use of the training materials produced by the project (PPT presentations), which were designed with an effective use of time in mind. These presentations are composed of the following elements:

- Development of the subtopic and main ideas to retain;
- Proposed Activities/Exercises.

That said, if the trainer/coach follows the logical sequence of the PPTs, he/she will certainly be able to complete the session within the stipulated time limit. These presentations can also be made available to learners for individual study.

Subtopics:

- 3.1. CSS Intro
- 3.2. CSS Advanced
- 3.3. CSS Responsive Web Design
- 3.4. CSS Grid
- 3.5. CSS SASS

Additional resources:

- CSS tutorial: w3schools
- Online Course on HTML and CSS: CodeAcademy

3.1. CSS Intro

What is CSS?

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

CSS handles the look and feel part of a web page. Using CSS, you can control the colour of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colours are used, as well as a variety of other effects.

CSS is easy to learn and understand but it provides a powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.

Advantages of CSS

- **CSS saves time** - You can write CSS once and then reuse the same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many web pages as you want.
- **Pages load faster** - If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So, less code means faster download times.
- **Easy maintenance** - To make a global change, simply change the style, and all the elements in all the web pages will be updated automatically.
- **Superior styles to HTML** - CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
- **Multiple Device Compatibility** - Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different

versions of a website can be presented for handheld devices such as PDAs and cellphones or for printing.

- **Global web standards** – Now HTML attributes are being deprecated and it is being recommended to use CSS. So, it is a good idea to start using CSS in all the HTML pages to make them compatible with future browsers.

Who Creates and Maintains CSS?

CSS is created and maintained through a group of people within the W3C called the CSS Working Group. The CSS Working Group creates documents called specifications. When a specification has been discussed and officially ratified by the W3C members, it becomes a recommendation.

These ratified specifications are called recommendations because the W3C has no control over the actual implementation of the language. Independent companies and organizations create that software.

NOTE: The World Wide Web Consortium or W3C is a group that makes recommendations about how the Internet works and how it should evolve.

CSS Versions

CSS was originally released in 1996 and consists of properties for adding font properties such as typeface and emphasis color of text, backgrounds, and other elements. CSS2 was released in 1998 with added styles for other media types so that it can be used for page layout designing. CSS3 was released in 1999 and presentation-style properties were added in it that allows you to build a presentation from documents.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

CSS Syntax

A CSS comprises of style rules that are interpreted by the browser and then applied to the corresponding elements in your document. A style rule is made of three parts:

- **Selector**: A selector is an HTML tag at which a style will be applied. This could be any tag like <h1> or <table> etc.
- **Property**: A property is a type of attribute of HTML tag. Put simply, all the HTML attributes are converted into CSS properties. They could be color, border, etc.
- **Value**: Values are assigned to properties. For example, color property can have the value either red or #F1F1F1 etc.

CSS Selectors

CSS selectors are used to "find" (or select) the HTML elements you want to style.

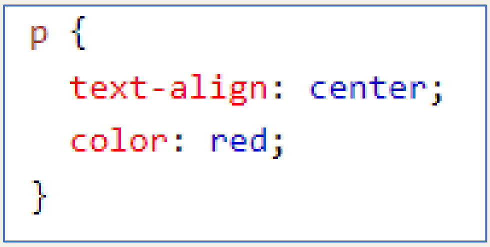
We can divide CSS selectors into five categories:

- **Simple selectors** (select elements based on name, id, class)
- **Combinator selectors** (select elements based on a specific relationship between them)
- **Pseudo-class selectors** (select elements based on a certain state)
- **Pseudo-elements selectors** (select and style a part of an element)
- **Attribute selectors** (select elements based on an attribute or attribute value)

The CSS element Selector

The element selector selects HTML elements based on the element name.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



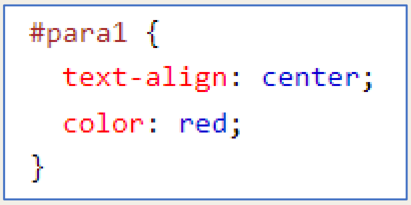
```
p {
  text-align: center;
  color: red;
}
```

*Figure 1 – The CSS element selector (**Source**: https://www.w3schools.com/css/css_selectors.asp)*

The CSS ID Selector

The id selector uses the id attribute of an HTML element to select a specific element. The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.



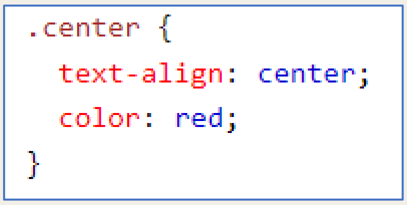
```
#para1 {
  text-align: center;
  color: red;
}
```

*Figure 2 – The CSS ID Selector (**Source**: https://www.w3schools.com/css/css_selectors.asp)*

The CSS class Selector

The class selector selects HTML elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the class name.

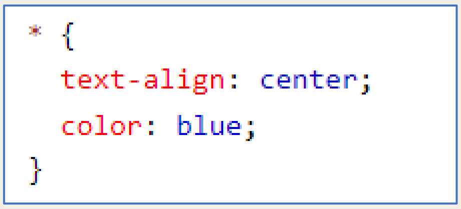


```
.center {
  text-align: center;
  color: red;
}
```

*Figure 3 – The CSS class Selector (**Source**: https://www.w3schools.com/css/css_selectors.asp)*

The CSS universal Selector

The universal selector (*) selects all HTML elements on the page.

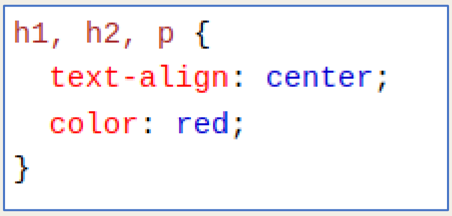


```
* {
  text-align: center;
  color: blue;
}
```

Figure 4 *– The CSS universal Selector (**Source**: https://www.w3schools.com/css/css_selectors.asp)*

Grouping selectors

You can apply a style to many selectors if you like. Just separate the selectors with a comma, as given in the following example:

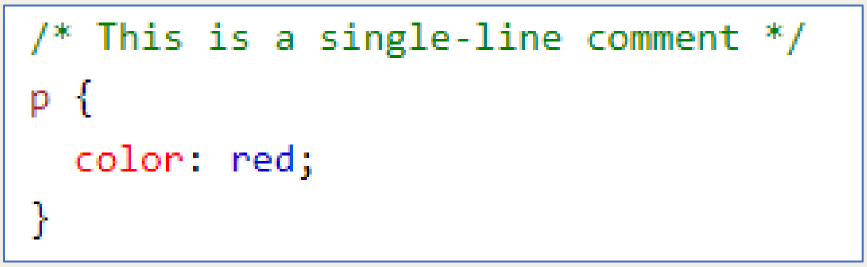


```
h1, h2, p {
  text-align: center;
  color: red;
}
```

Figure 5 *– Grouping Selectors (**Source**: https://www.w3schools.com/css/css_selectors.asp)*

CSS Comments

- Comments are used to explain the code and may help when you edit the source code at a later date.
- Comments are ignored by browsers.
  A CSS comment starts with /* and ends with */



```
/* This is a single-line comment */
p {
  color: red;
}
```

***Figure 6** – CSS comments (**Source**: https://www.w3schools.com/css/css_comments.asp)*

CSS Colours

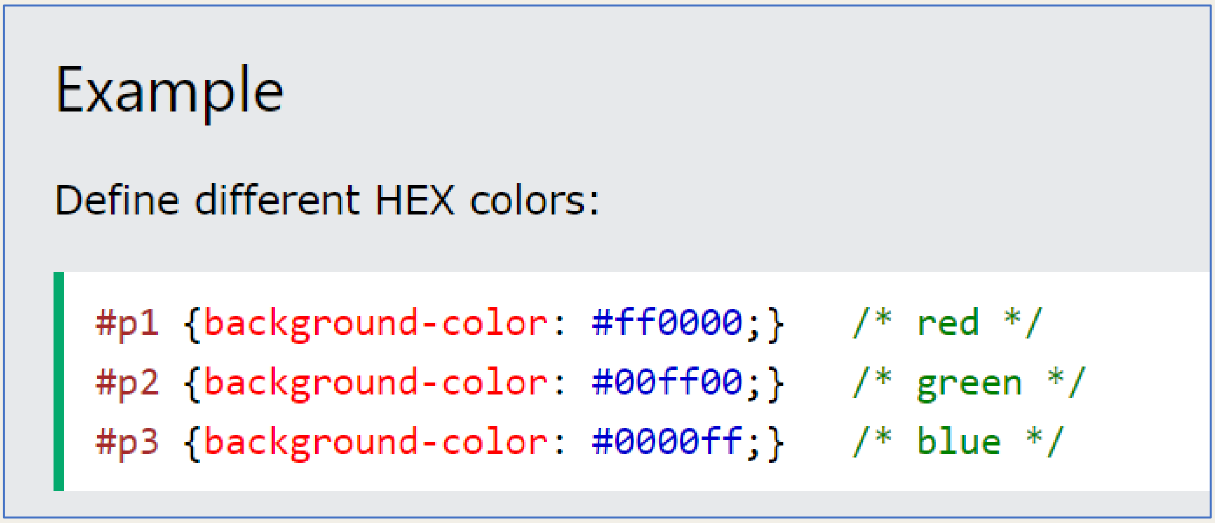
Colours in CSS can be specified by the following methods:

- Hexadecimal colors
- Hexadecimal colors with transparency
- RGB colours
- RGBA colours
- HSL colours
- HSLA colours
- Predefined/Cross-browser color names
- With the current color keyword

Hexadecimal Colours

A hexadecimal colour is specified with: #RRGGBB, where the RR (red), GG (green) and BB (blue) hexadecimal integers specify the components of the colour. All values must be between 00 and FF.

For example, the #0000ff value is rendered as blue, because the blue component is set to its highest value (ff) and the others are set to 00.



Example

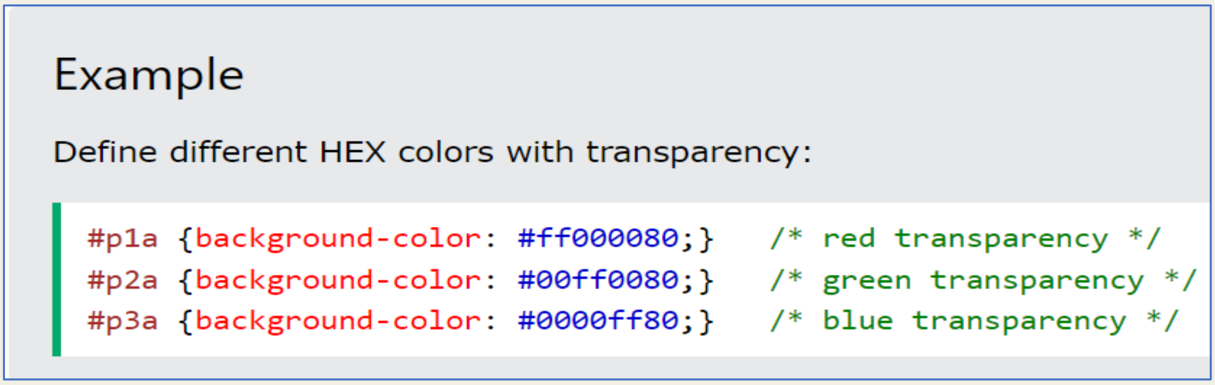
Define different HEX colors:

```
#p1 {background-color: #ff0000;}   /* red */
#p2 {background-color: #00ff00;}   /* green */
#p3 {background-color: #0000ff;}   /* blue */
```

***Figure 7** – Hex Colours (**Source**: https://www.w3schools.com/css/css_colors.asp)*

Hexadecimal Colours With Transparency

A hexadecimal colour is specified with: #RRGGBB. To add transparency, add two additional digits between 00 and FF.



Example

Define different HEX colors with transparency:

```
#p1a {background-color: #ff000080;}   /* red transparency */
#p2a {background-color: #00ff0080;}   /* green transparency */
#p3a {background-color: #0000ff80;}   /* blue transparency */
```

***Figure 8** – Hex Colors with transparency (**Source**: https://www.w3schools.com/css/css_colors.asp)*

RGB Colours

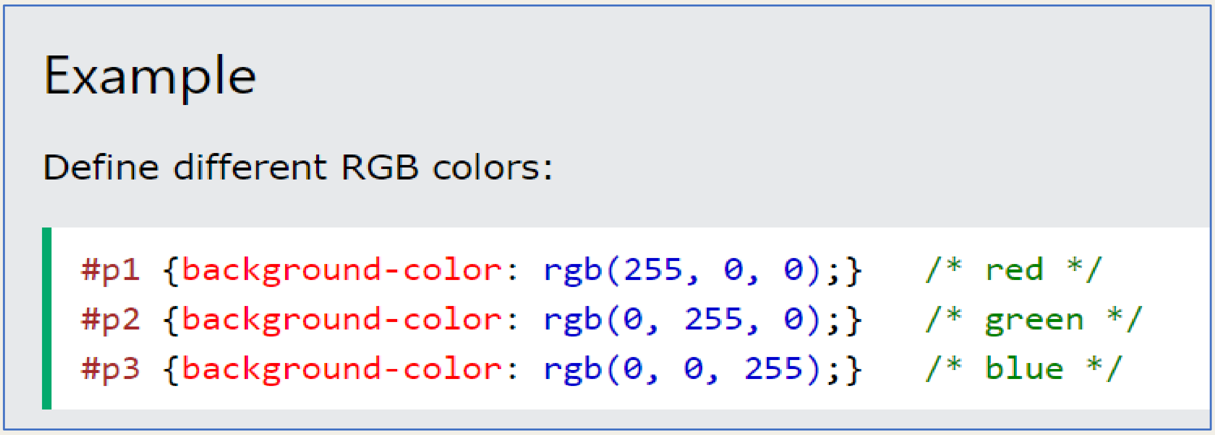
An RGB colour value is specified with the rgb() function, which has the following syntax:

```
rgb(red, green, blue)
```

Each parameter (red, green, and blue) defines the intensity of the colour and can be an integer between 0 and 255 or a percentage value (from 0% to 100%).

For example, the rgb(0,0,255) value is rendered as blue, because the blue parameter is set to its highest value (255) and the others are set to 0.

Also, the following values define equal colour: rgb(0,0,255) and rgb(0%,0%,100%).



Example

Define different RGB colors:

```
#p1 {background-color: rgb(255, 0, 0);}   /* red */
#p2 {background-color: rgb(0, 255, 0);}   /* green */
#p3 {background-color: rgb(0, 0, 255);}   /* blue */
```

***Figure 9** – RGB Colors (**Source**: https://www.w3schools.com/css/css_colors.asp)*

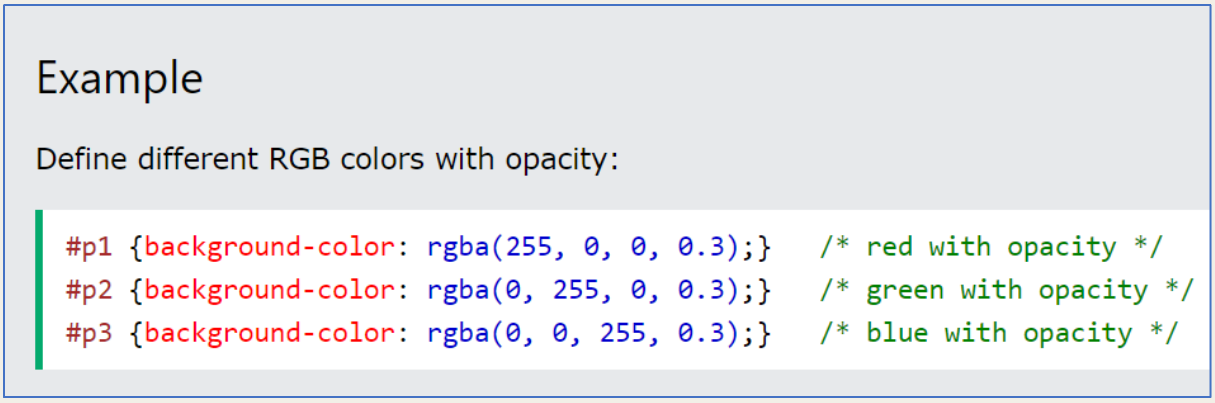
RGBA Colours

RGBA colour values are an extension of RGB colour values with an alpha channel - which specifies the opacity of the object.

An RGBA color is specified with the rgba() function, which has the following syntax:

```
rgba(red, green, blue, alpha)
```

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).



Example

Define different RGB colors with opacity:

```
#p1 {background-color: rgba(255, 0, 0, 0.3);}   /* red with opacity */
#p2 {background-color: rgba(0, 255, 0, 0.3);}   /* green with opacity */
#p3 {background-color: rgba(0, 0, 255, 0.3);}   /* blue with opacity */
```

***Figure 10** – RGB Colors with opacity (**Source**: https://www.w3schools.com/css/css_colors.asp)*

HSL Colours

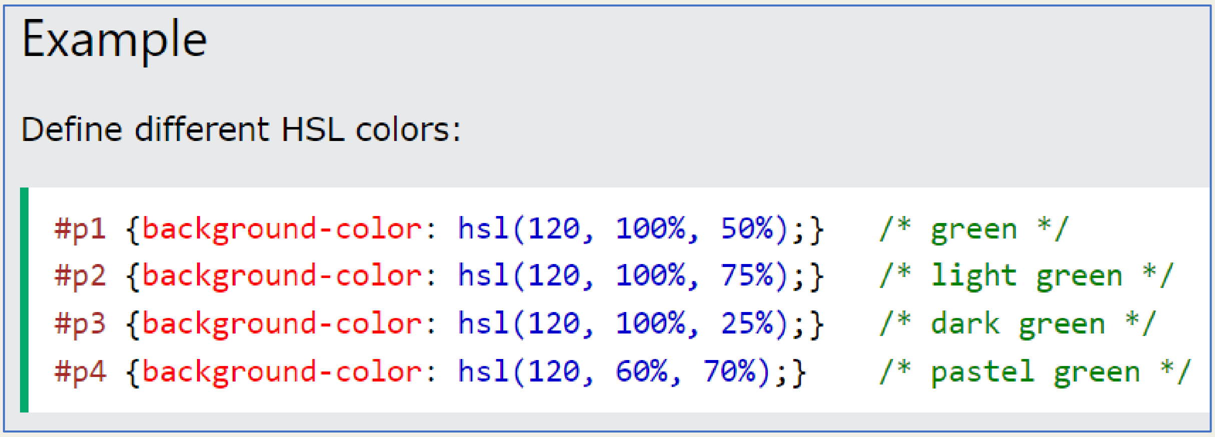
HSL stands for hue, saturation, and lightness - and represents a cylindrical-coordinate representation of colours.

An HSL color value is specified with the hsl() function, which has the following syntax:

```
hsl(hue, saturation, lightness)
```

Hue is a degree on the color wheel (from 0 to 360) - 0 (or 360) is red, 120 is green, 240 is blue. Saturation is a percentage value; 0% means a shade of gray and 100% is the full color. Lightness is also a percentage; 0% is black, 100% is white.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



Example

Define different HSL colors:

```
#p1 {background-color: hsl(120, 100%, 50%);}   /* green */
#p2 {background-color: hsl(120, 100%, 75%);}   /* light green */
#p3 {background-color: hsl(120, 100%, 25%);}   /* dark green */
#p4 {background-color: hsl(120, 60%, 70%);}    /* pastel green */
```

***Figure 11** – HSL Colors (**Source**: https://www.w3schools.com/css/css_colors.asp)*

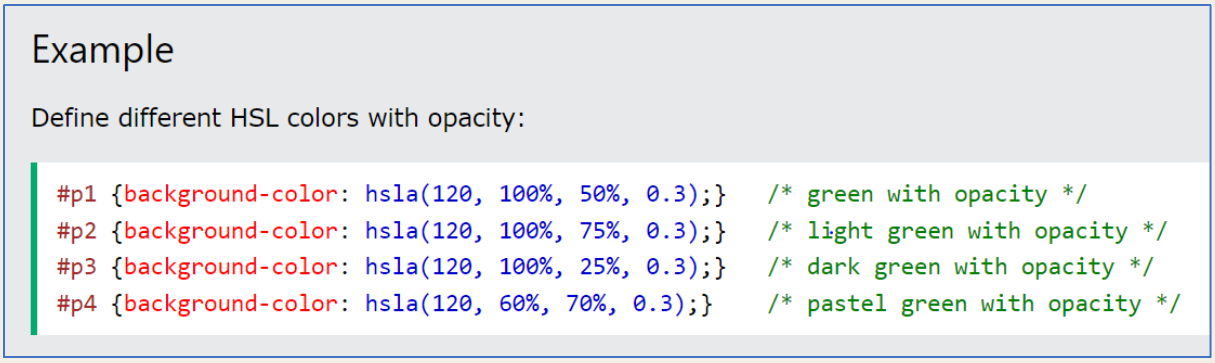
HSLA Colors

HSLA color values are an extension of HSL color values with an alpha channel - which specifies the opacity of the object.

An HSLA color value is specified with the hsla() function, which has the following syntax:

```
hsla(hue, saturation, lightness, alpha)
```

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).



Example

Define different HSL colors with opacity:

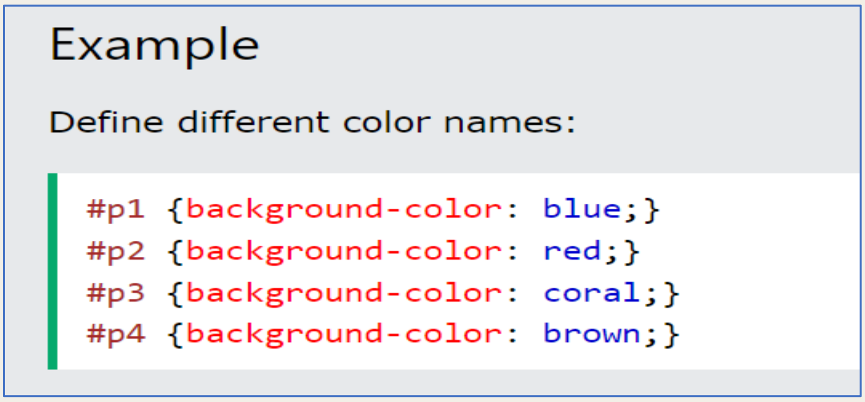
```
#p1 {background-color: hsla(120, 100%, 50%, 0.3);}   /* green with opacity */
#p2 {background-color: hsla(120, 100%, 75%, 0.3);}   /* light green with opacity */
#p3 {background-color: hsla(120, 100%, 25%, 0.3);}   /* dark green with opacity */
#p4 {background-color: hsla(120, 60%, 70%, 0.3);}    /* pastel green with opacity */
```

Figure 12 *– HSL Colors with opacity (**Source**: https://www.w3schools.com/css/css_colors.asp)*

Predefined/Cross-browser Color Names

140 color names are predefined in the HTML and CSS color specification.

For example: blue, red, coral, brown, etc:



Example

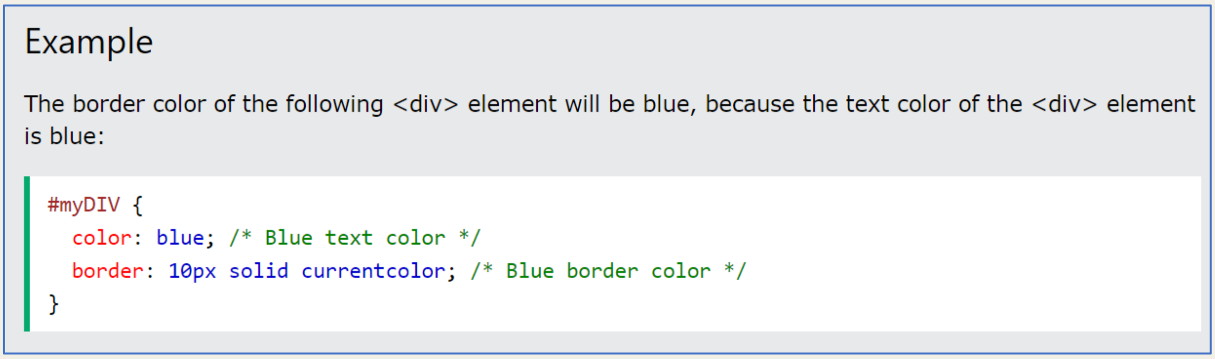
Define different color names:

```
#p1 {background-color: blue;}
#p2 {background-color: red;}
#p3 {background-color: coral;}
#p4 {background-color: brown;}
```

Figure 13 *– Cross-browser Colors (**Source**: https://www.w3schools.com/css/css_colors.asp)*

The currentcolor Keyword

The currentcolor keyword refers to the value of the color property of an element.



Example

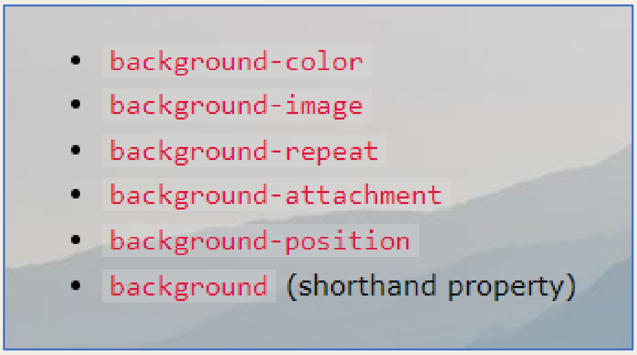
The border color of the following <div> element will be blue, because the text color of the <div> element is blue:

```
#myDIV {
  color: blue; /* Blue text color */
  border: 10px solid currentcolor; /* Blue border color */
}
```

Figure 14 *– The currentcolor keyword (**Source**: https://www.w3schools.com/colors/colors_currentcolor.asp)*

CSS Backgrounds

- The CSS background properties are used to add background effects for elements.
- Some background properties:

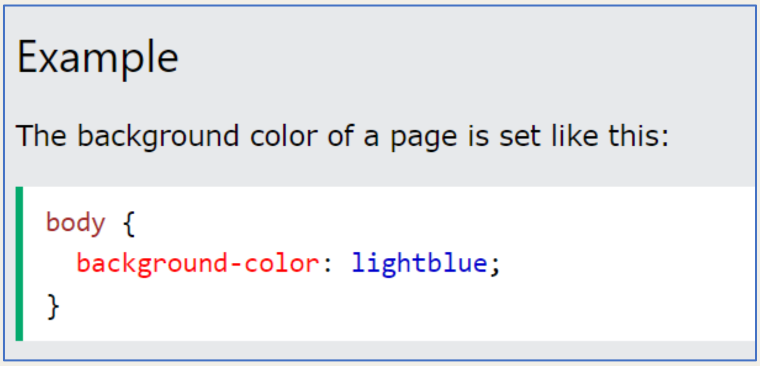


- background-color
- background-image
- background-repeat
- background-attachment
- background-position
- background (shorthand property)

Figure 15 *– Background (**Source**: https://www.w3schools.com/css/css_background.asp)*

CSS background-color

The background-color property specifies the background color of an element.



Example

The background color of a page is set like this:

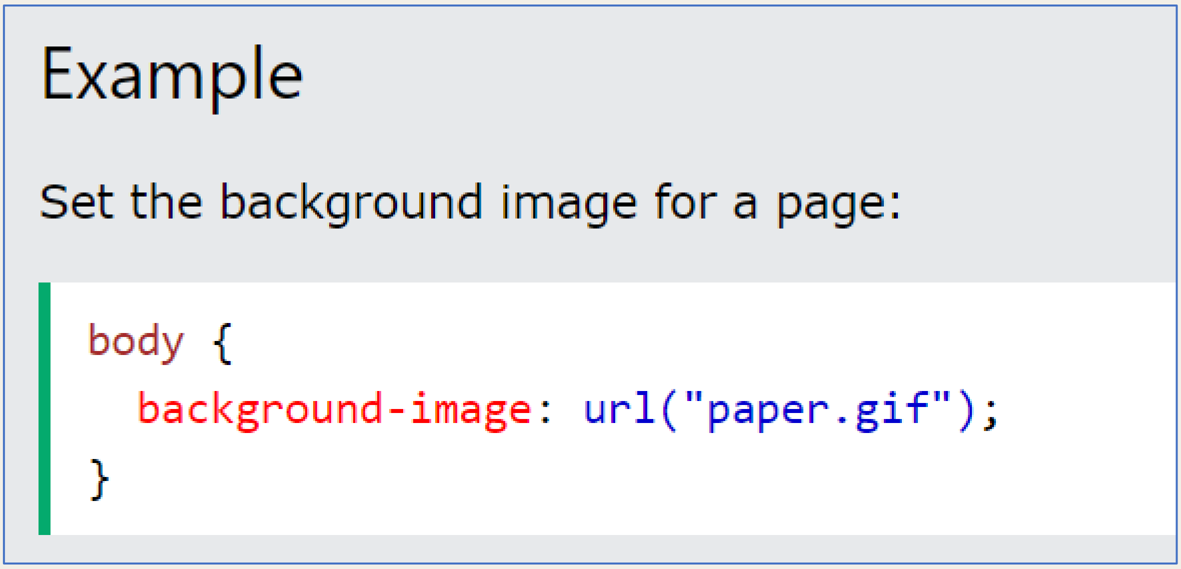
```
body {
  background-color: lightblue;
}
```

***Figure 16** – Background color (**Source**:https://www.w3schools.com/css/css_background.asp)*

CSS background-image

The background-image property specifies an image to use as the background of an element.

By default, the image is repeated so it covers the entire element.



Example

Set the background image for a page:

```
body {
  background-image: url("paper.gif");
}
```

***Figure 17** – Background image (**Source**: https://www.w3schools.com/css/css_background.asp)*

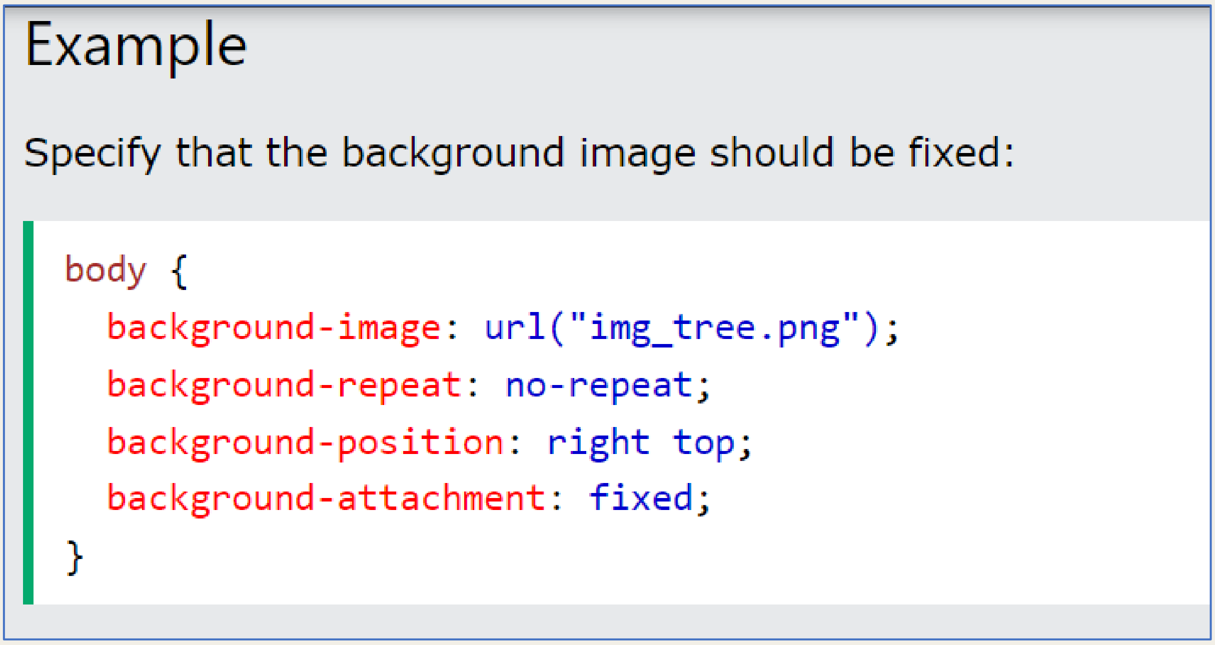
CSS background-repeat

By default, the background-image property repeats an image both horizontally and vertically.

Some images should be repeated only horizontally or vertically, or they will look strange.

CSS background-attachment

The background-attachment property specifies whether the background image should scroll or be fixed.

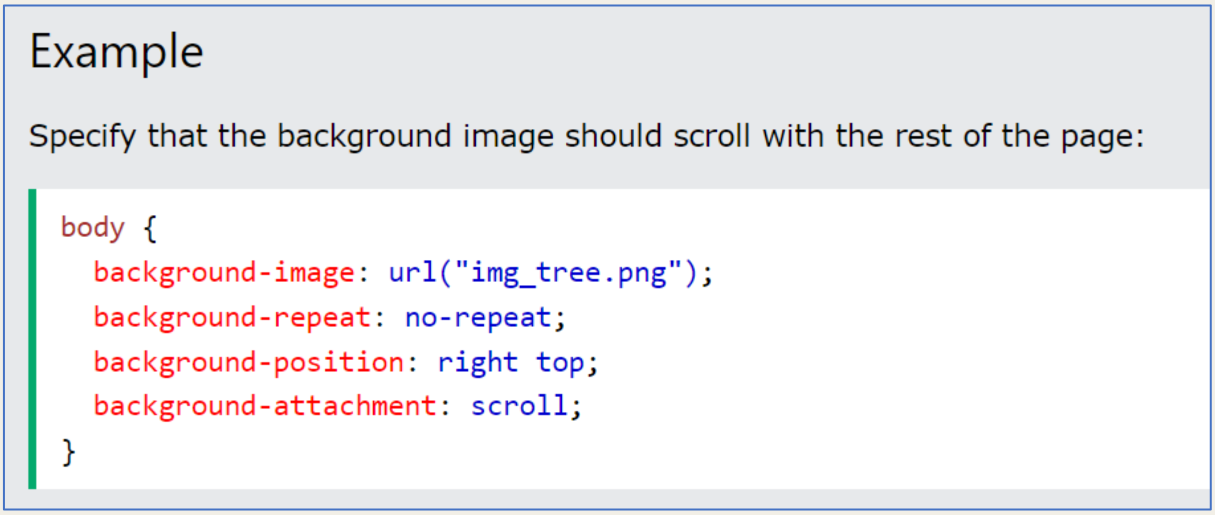


Example

Specify that the background image should be fixed:

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
  background-attachment: fixed;
}
```

Figure 18 *– Background-attachement (**Source**: https://www.w3schools.com/css/css_background_attachment.asp)*



Example

Specify that the background image should scroll with the rest of the page:

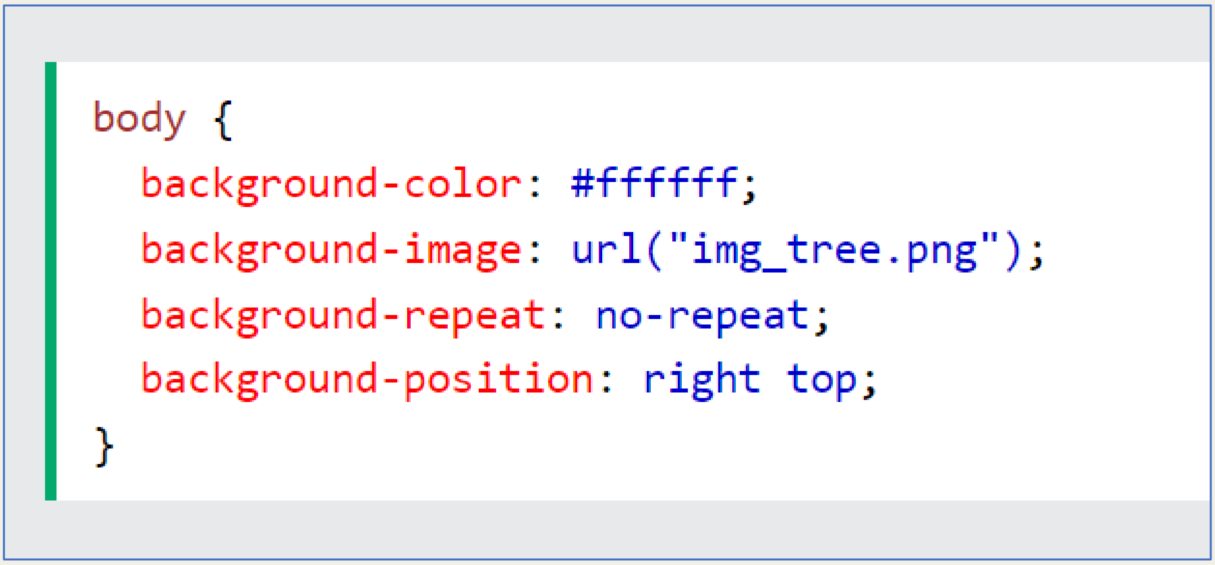
```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
  background-attachment: scroll;
}
```

***Figure 19** – Background-attachement (**Source**: https://www.w3schools.com/css/css_background_attachment.asp)*

CSS background - Shorthand property

To shorten the code, it is also possible to specify all the background properties in one single property. This is called a shorthand property.

Instead of writing:

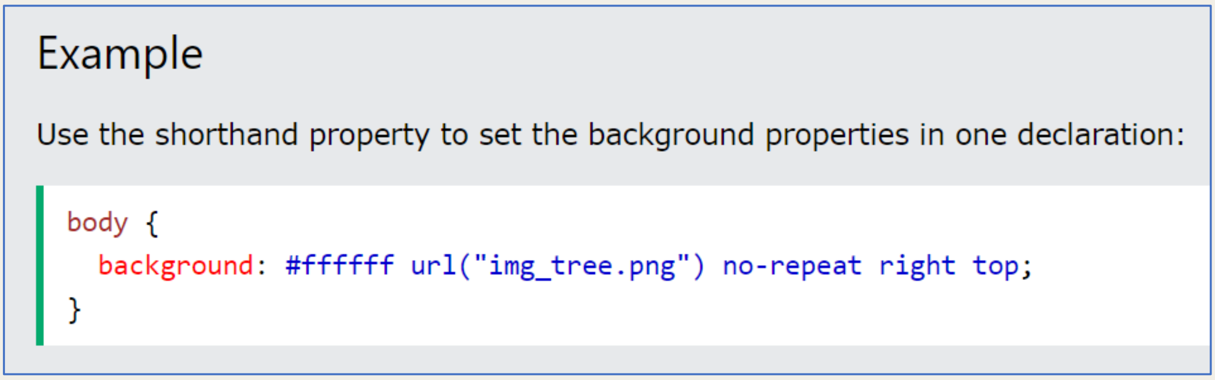


```
body {
  background-color: #ffffff;
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
}
```

***Figure 20** – Background shorthand property (**Source**:https://www.w3schools.com/css/css_background_shorthand.asp)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

You can use the shorthand property background:



Example

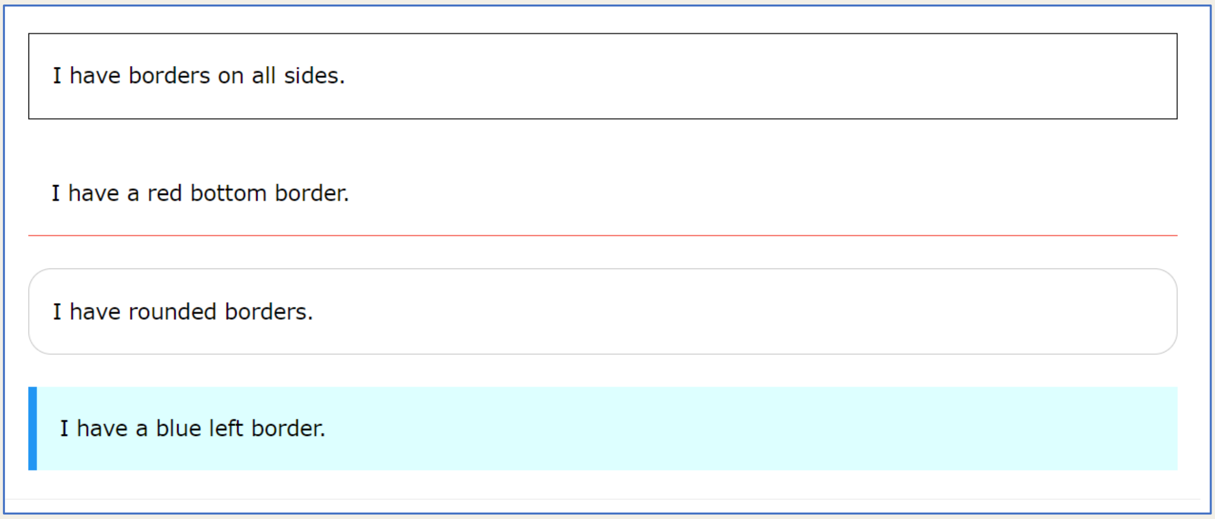
Use the shorthand property to set the background properties in one declaration:

```
body {
  background: #ffffff url("img_tree.png") no-repeat right top;
}
```

Figure 21 – Background shorthand property
*(**Source**:https://www.w3schools.com/css/css_background_shorthand.asp)*

CSS Border

The CSS border properties allow you to specify the style, width, and color of an element's border.



I have borders on all sides.

I have a red bottom border.

I have rounded borders.

I have a blue left border.

*Figure 22 – Border (**Source**: https://www.w3schools.com/css/css_border.asp)*

CSS Margins

Margins are used to create space around elements, outside of any defined borders.

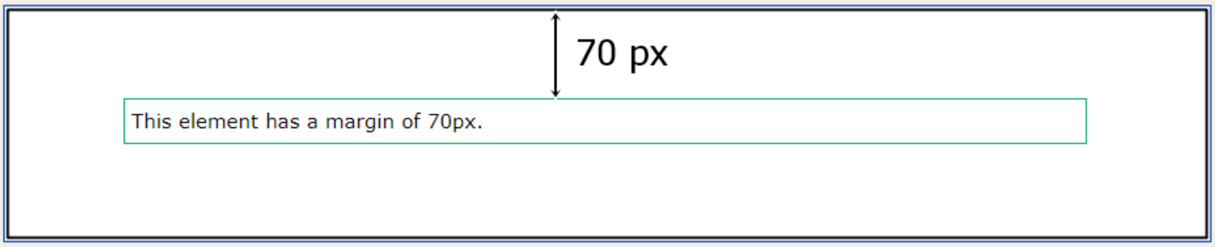
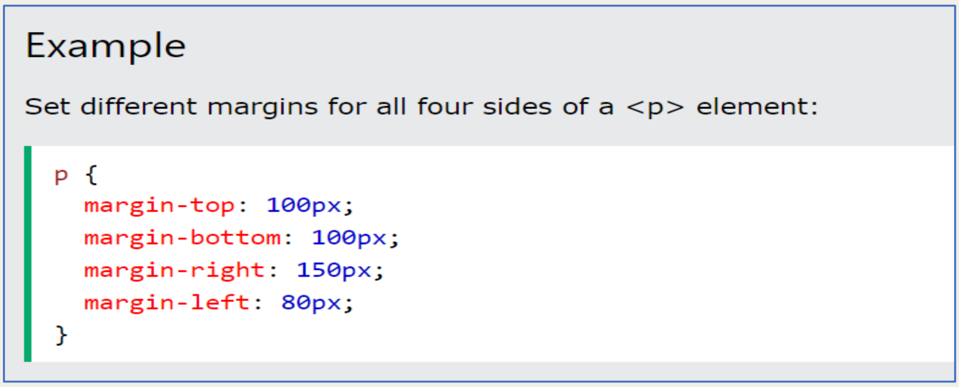


*Figure 23 – Margin (**Source**:https://www.w3schools.com/css/css_margin.asp)*

With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).



Example

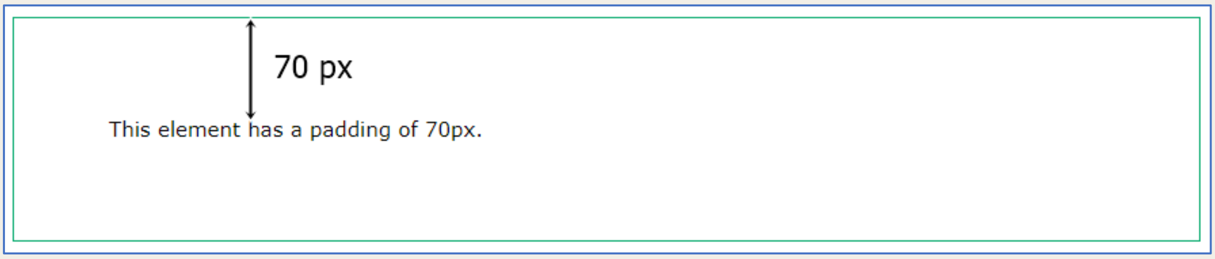
Set different margins for all four sides of a <p> element:

```
p {
  margin-top: 100px;
  margin-bottom: 100px;
  margin-right: 150px;
  margin-left: 80px;
}
```

*Figure 24 – Margin (**Source**:https://www.w3schools.com/css/css_margin.asp)*

CSS Padding

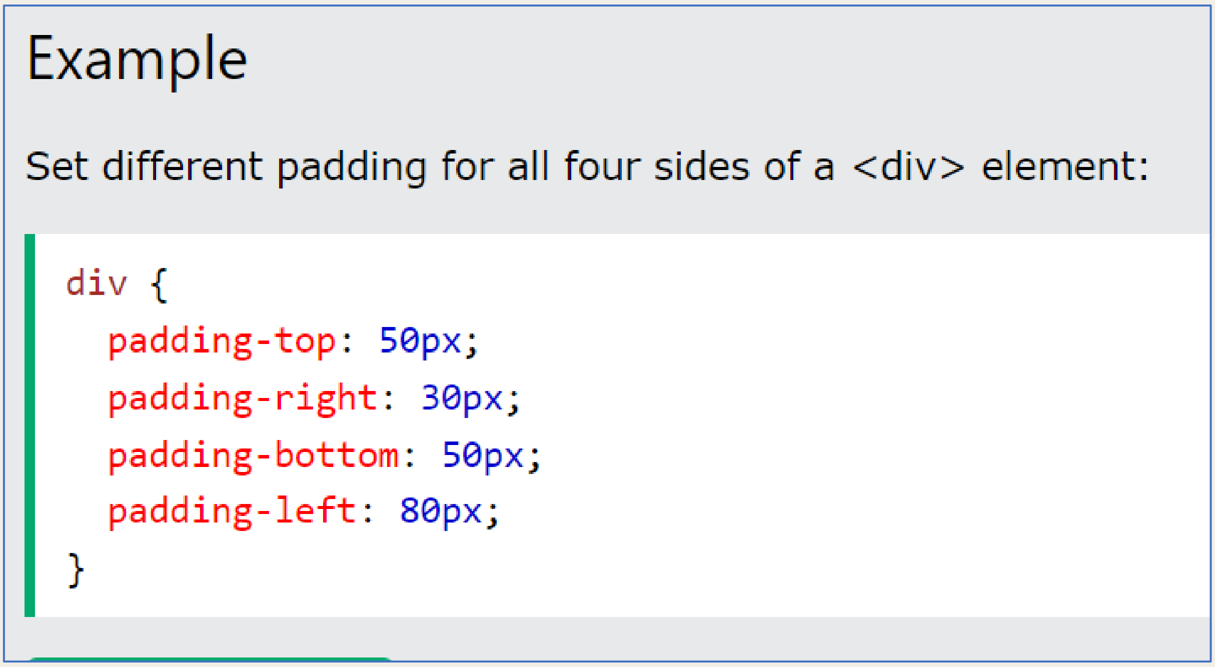
Padding is used to create space around an element's content, inside of any defined borders.



This element has a padding of 70px.

__Figure 25__ – Padding (__Source__:https://www.w3schools.com/css/css_padding.asp)

With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).



Example

Set different padding for all four sides of a <div> element:

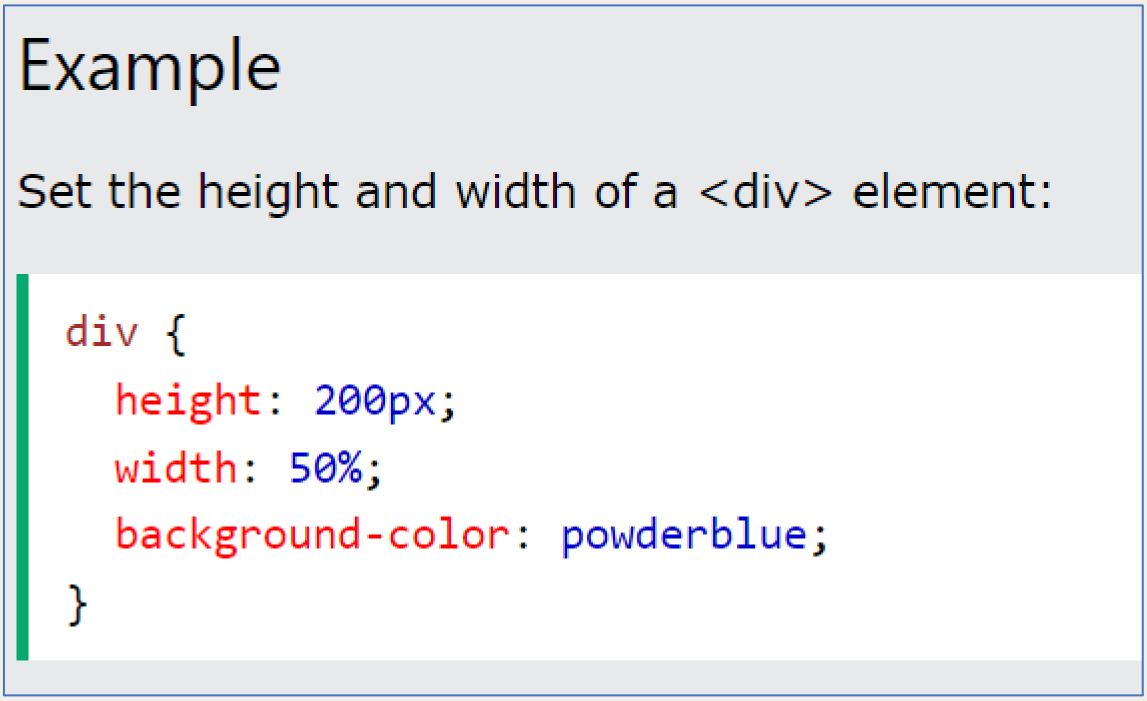
```
div {
  padding-top: 50px;
  padding-right: 30px;
  padding-bottom: 50px;
  padding-left: 80px;
}
```

__Figure 26__ – Padding (__Source__:https://www.w3schools.com/css/css_padding.asp)

CSS height and width

The height and width properties are used to set the height and width of an element. The height and width properties do not include padding, borders, or margins. It sets the height/width of the area inside the padding, border, and margin of the element. The height and width properties may have the following values:

- auto - This is default. The browser calculates the height and width
- length - Defines the height/width in px, cm etc.
- % - Defines the height/width in percent of the containing block
- initial - Sets the height/width to its default value
- inherit - The height/width will be inherited from its parent value



Example

Set the height and width of a <div> element:

```
div {
  height: 200px;
  width: 50%;
  background-color: powderblue;
}
```

***Figure 27** – Height and width (**Source**: https://www.w3schools.com/css/css_dimension.asp)*

The CSS Box Model

In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:

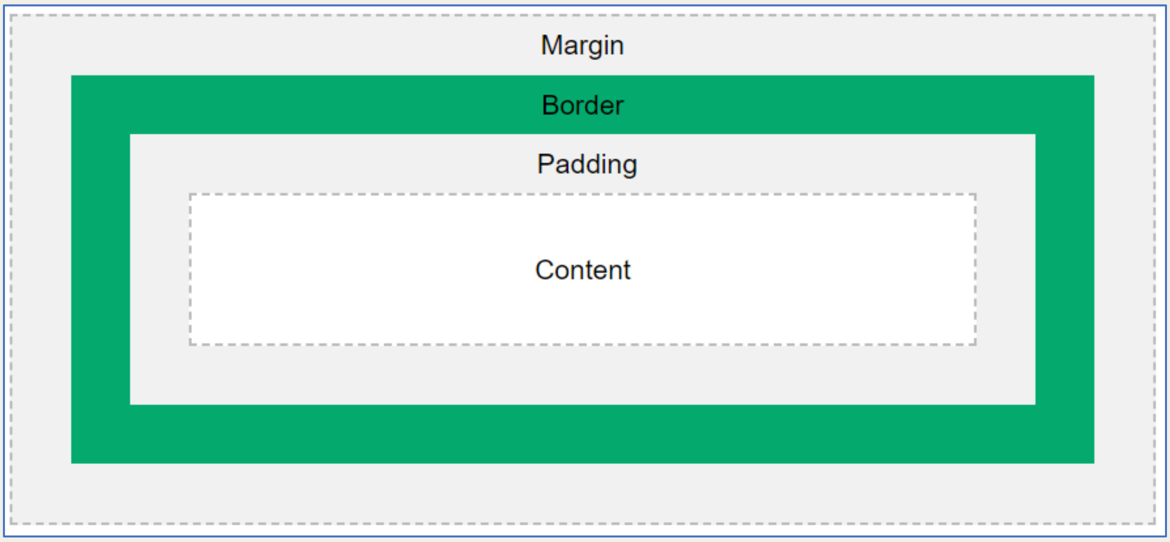


Figure 28 *– Box model (**Source**: https://www.w3schools.com/css/css_boxmodel.asp)*

Explanation of the different parts:

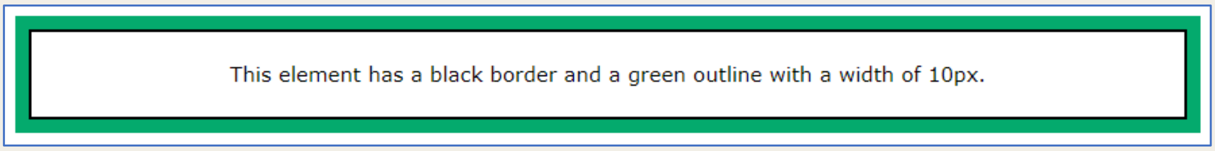
- Content - The content of the box, where text and images appear
- Padding - Clears an area around the content. The padding is transparent
- Border - A border that goes around the padding and content
- Margin - Clears an area outside the border. The margin is transparent

The box model allows us to add a border around elements, and to define space between elements.

CSS Outline

An outline is a line that is drawn around elements, OUTSIDE the borders, to make the element "stand out".

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

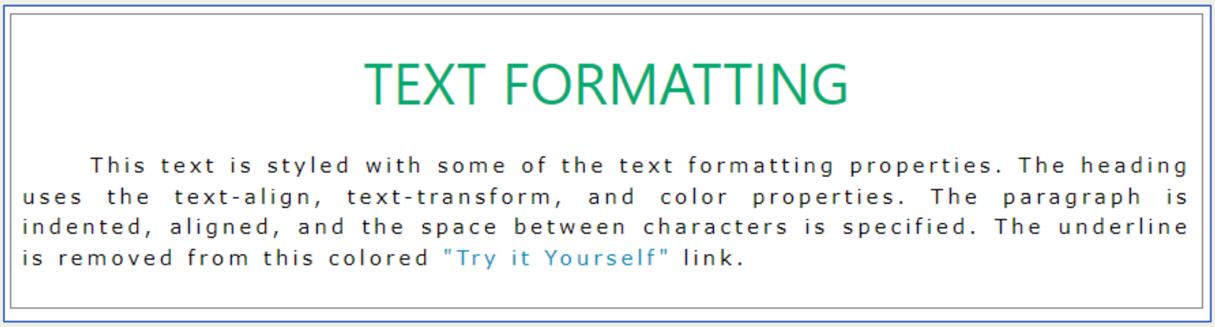


This element has a black border and a green outline with a width of 10px.

*Figure 29 – CSS Outline (**Source**: https://www.w3schools.com/css/css_outline.asp)*

CSS Text

CSS has a lot of properties for formatting text.



TEXT FORMATTING

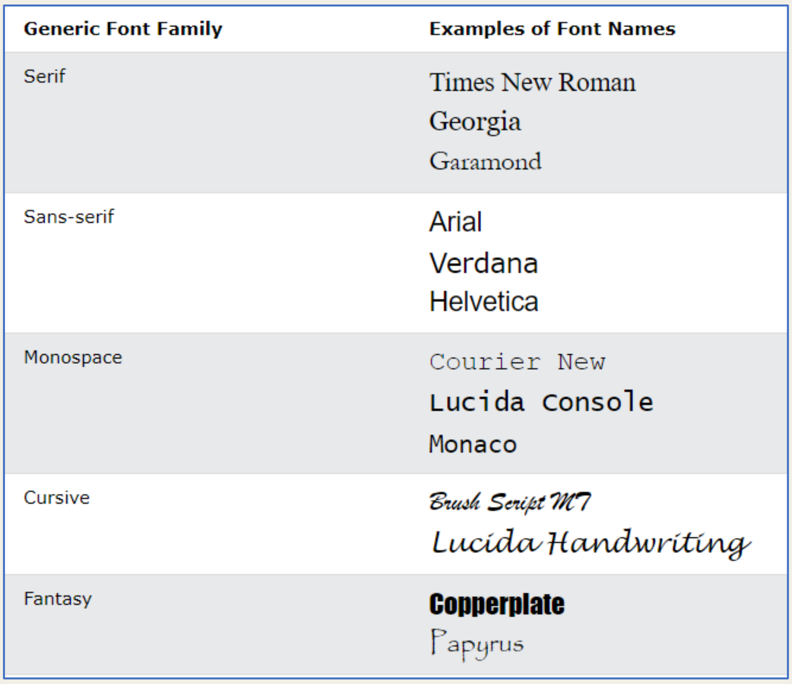
This text is styled with some of the text formatting properties. The heading uses the text-align, text-transform, and color properties. The paragraph is indented, aligned, and the space between characters is specified. The underline is removed from this colored "Try it Yourself" link.

*Figure 30 – CSS Text (**Source**: https://www.w3schools.com/css/css_text.asp)*

- Text Color
- Text Alignment
- Text Decoration
- Text Transformation
- Text Spacing
- Text Shadow

CSS Fonts

- The right font can create a strong identity for your brand.
- Using a font that is easy to read is important. The font adds value to your text. It is also important to choose the correct color and text size for the font.



Generic Font Family	Examples of Font Names
Serif	Times New Roman Georgia Garamond
Sans-serif	Arial Verdana Helvetica
Monospace	Courier New Lucida Console Monaco
Cursive	Brush Script MT Lucida Handwriting
Fantasy	Copperplate Papyrus

***Figure 31** – CSS Fonts (**Source**: https://www.w3schools.com/css/css_font.asp)*

CSS Icons

- The simplest way to add an icon to your HTML page, is with an icon library, such as Font Awesome.
- Add the name of the specified icon class to any inline HTML element (like <i> or <span>).
- All the icons in the icon libraries below, are scalable vectors that can be customized with CSS.

***Figure 32** – CSS Icons (**Source**: https://www.w3schools.com/css/css_icons.asp)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

CSS Links

Links can be styled with any CSS property (e.g. color, font-family, background, etc,).

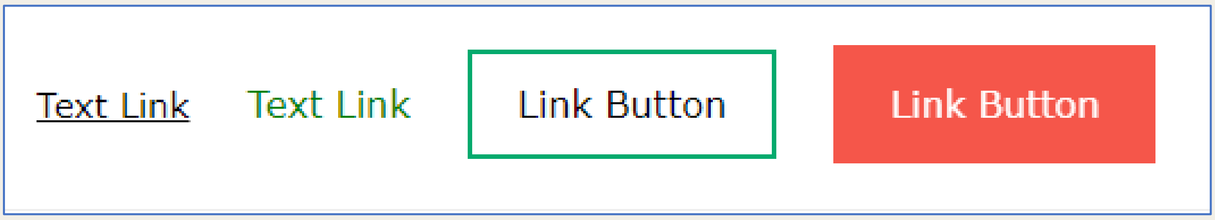


***Figure 33** – Links (**Source**: https://www.w3schools.com/css/css_link.asp)*

In addition, links can be styled differently depending on what state they are in.

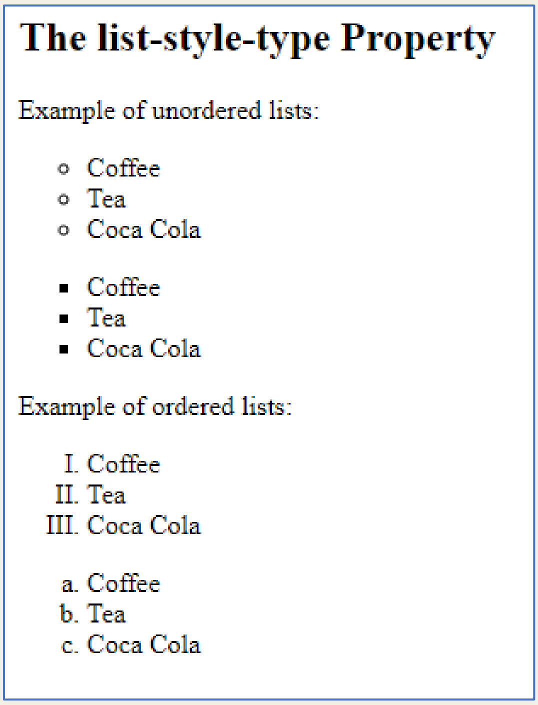
The four links states are:

- a:link - a normal, unvisited link
- a:visited - a link the user has visited
- a:hover - a link when the user mouses over it
- a:active - a link the moment it is clicked.

CSS Lists

The CSS list properties allow you to:

- Set different list item markers for ordered lists
- Set different list item markers for unordered lists
- Set an image as the list item marker
- Add background colors to lists and list items



The list-style-type Property

Example of unordered lists:

○ Coffee
○ Tea
○ Coca Cola

▪ Coffee
▪ Tea
▪ Coca Cola

Example of ordered lists:

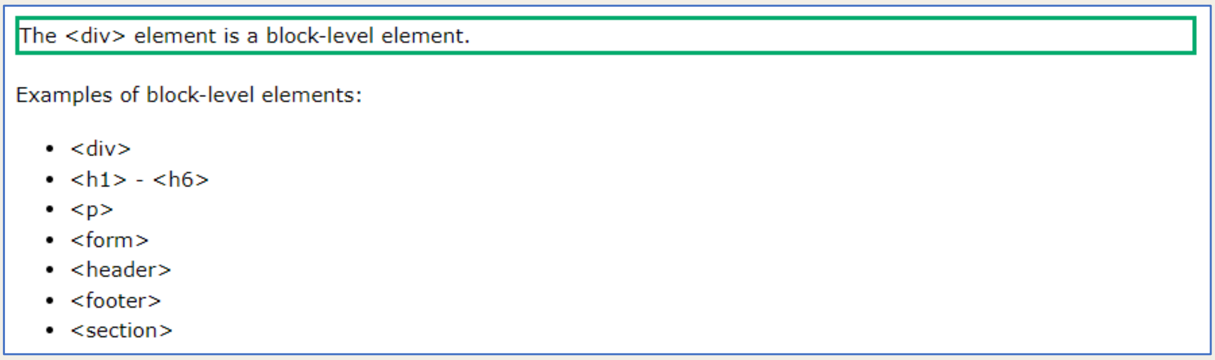
I. Coffee
II. Tea
III. Coca Cola

a. Coffee
b. Tea
c. Coca Cola

***Figure 34** – Lists (**Source**: https://www.w3schools.com/css/css_list.asp)*

CSS Display

- The display property specifies if/how an element is displayed.
- Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is block or inline.



The <div> element is a block-level element.

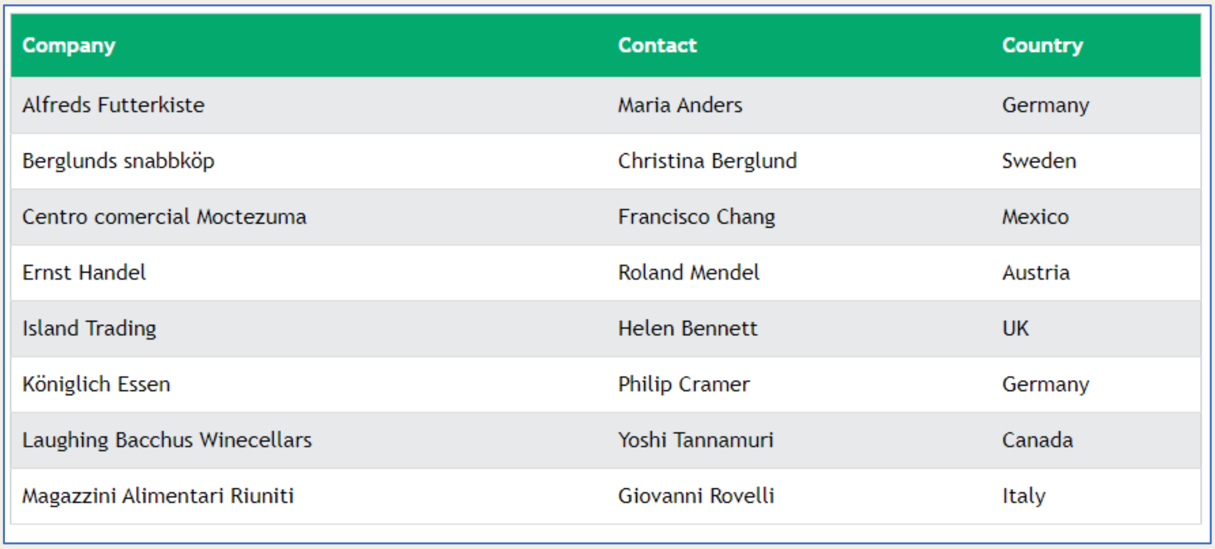
Examples of block-level elements:

- <div>
- <h1> - <h6>
- <p>
- <form>
- <header>
- <footer>
- <section>

***Figure 35** – Display (**Source**: https://www.w3schools.com/css/css_display_visibility.asp)*

CSS Tables

The look of an HTML table can be greatly improved with CSS:



Company	Contact	Country
Alfreds Futterkiste	Maria Anders	Germany
Berglunds snabbköp	Christina Berglund	Sweden
Centro comercial Moctezuma	Francisco Chang	Mexico
Ernst Handel	Roland Mendel	Austria
Island Trading	Helen Bennett	UK
Königlich Essen	Philip Cramer	Germany
Laughing Bacchus Winecellars	Yoshi Tannamuri	Canada
Magazzini Alimentari Riuniti	Giovanni Rovelli	Italy

***Figure 36** – Tables (**Source**: https://www.w3schools.com/css/css_table.asp)*

CSS Max-width

- The problem with the <div> above occurs when the browser window is smaller than the width of the element. The browser then adds a horizontal scrollbar to the page
- Using max-width instead, in this situation, will improve the browser's handling of small windows. This is important when making a site usable on small devices.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

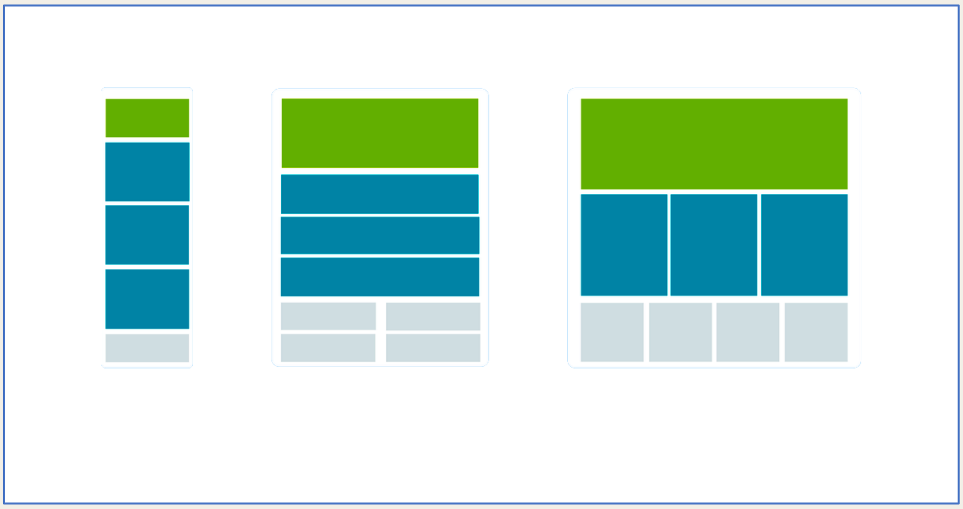


***Figure 37** – Max Width (**Source**: https://www.w3schools.com/css/css_max-width.asp)*

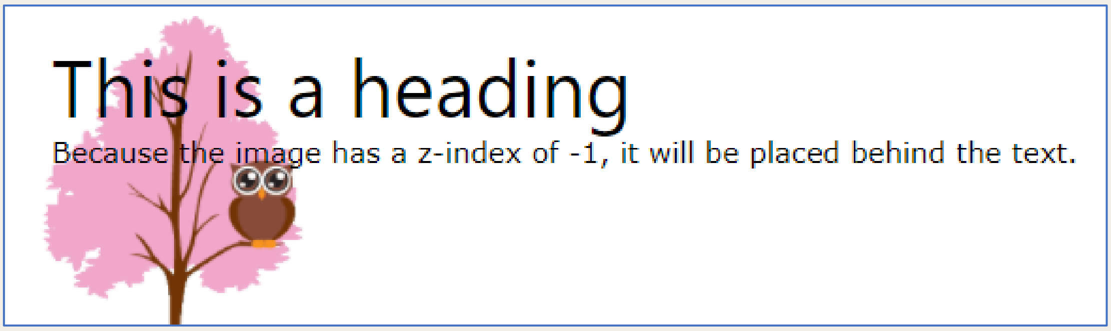
CSS Position

- The position property specifies the type of positioning method used for an element
- There are five different position values:
- static
- relative
- fixed
- absolute
- sticky

CSS Z-index

- The z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others).
- An element can have a positive or negative stack order:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



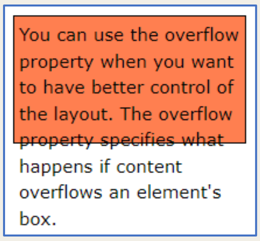
This is a heading

Because the image has a z-index of -1, it will be placed behind the text.

Figure 38 – Z-index (Source: https://www.w3schools.com/css/css_z-index.asp)

CSS Overflow

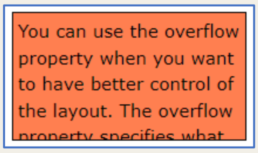
- The overflow property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.
- The overflow property has the following values:
- visible



You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Figure 39 – Overflow (Source: https://www.w3schools.com/css/css_overflow.asp)

- hidden

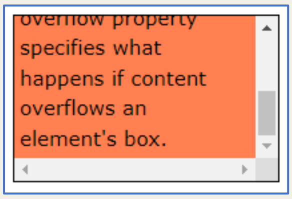


You can use the overflow property when you want to have better control of the layout. The overflow property specifies what

Figure 40 – Overflow (Source: https://www.w3schools.com/css/css_overflow.asp)

- Scroll

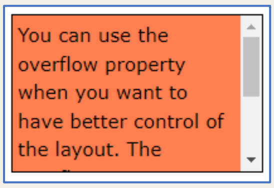
The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



overflow property
specifies what
happens if content
overflows an
element's box.

Figure 41 – *Overflow (**Source**: https://www.w3schools.com/css/css_overflow.asp)*

- auto



You can use the overflow property when you want to have better control of the layout. The

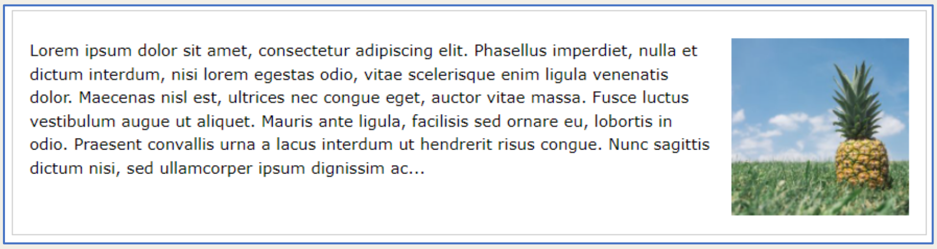
Figure 42 – *Overflow (**Source**: https://www.w3schools.com/css/css_overflow.asp)*

CSS Float

The float property is used for positioning and formatting content

Example - **float: right**

The following example specifies that an image should float to the **right** in a text:

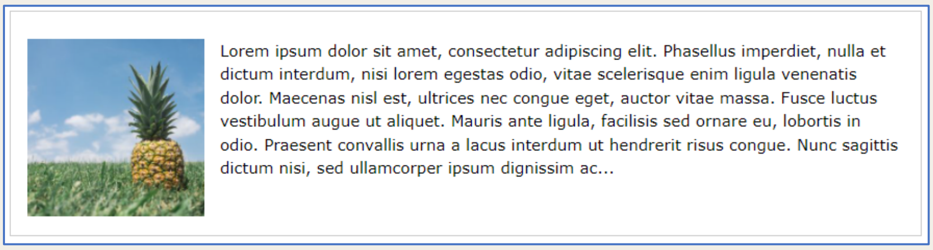


Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

Figure 43 – *Float-right (**Source**: https://www.w3schools.com/css/css_float.asp)*

- Example - **float: left**

  The following example specifies that an image should float to the **left** in a text:

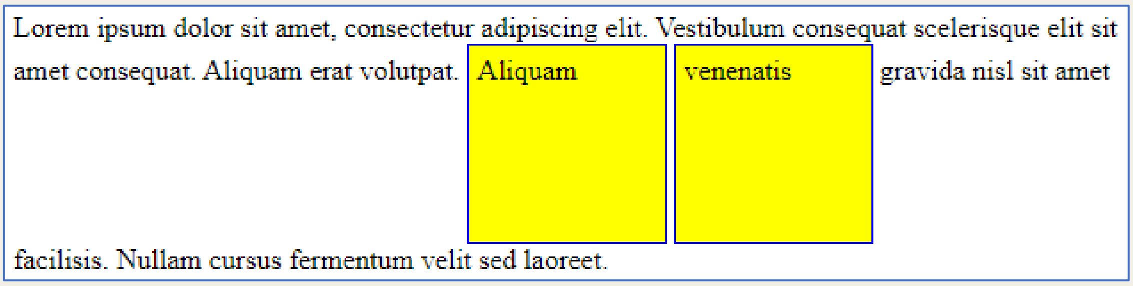


Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

***Figure 44** – Float-left(**Source**: https://www.w3schools.com/css/css_float.asp)*

CSS Inline-block

- Display: inline-block allows to set a width and height on the element.
- With display: inline-block the top and bottom margins/paddings are respected
- Display: inline-block does not add a line-break after the element, so the element can sit next to other elements.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet consequat. Aliquam erat volutpat. Aliquam venenatis gravida nisl sit amet facilisis. Nullam cursus fermentum velit sed laoreet.

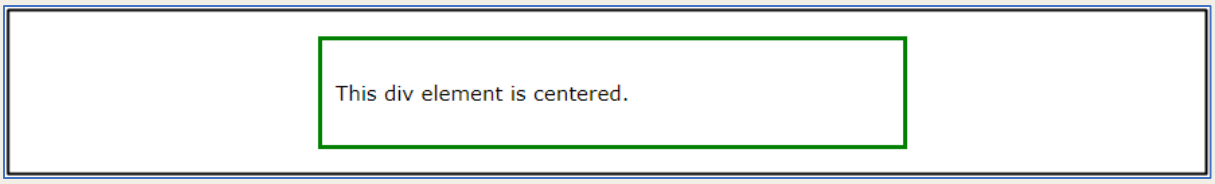
***Figure 45** – Inline-block (**Source**: https://www.w3schools.com/css/css_inline-block.asp)*

CSS Align

- To horizontally center a block element (like <div>), use margin: auto

- Setting the width of the element will prevent it from stretching out to the edges of its container.

- The element will then take up the specified width, and the remaining space will be split equally between the two margins:



This div element is centered.

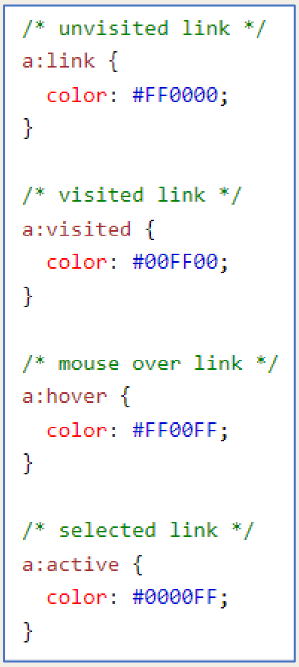
Figure 46 *– Align (**Source**: https://www.w3schools.com/css/css_align.asp)*

CSS Combinators

- A CSS selector can contain more than one simple selector. Between the simple selectors, we can include a combinator.
- There are four different combinators in CSS :
- descendant selector (space) -> e.g. div p {background-color: yellow }
- child selector (>) -> e.g. div > p {background-color: yellow }
- adjacent sibling selector (+) -> e.g. div + p {background-color: yellow }
- general sibling selector (~) -> e.g. div ~ p {background-color: yellow }

CSS Pseudo-class

- A pseudo-class is used to define a special state of an element.
- For example, it can be used to:
- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus

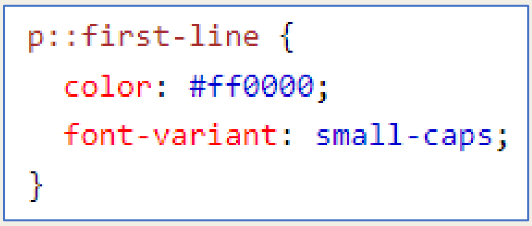


```
/* unvisited link */
a:link {
  color: #FF0000;
}

/* visited link */
a:visited {
  color: #00FF00;
}

/* mouse over link */
a:hover {
  color: #FF00FF;
}

/* selected link */
a:active {
  color: #0000FF;
}
```

***Figure 47** – Pseudo Class (**Source**: https://www.w3schools.com/css/css_pseudo_classes.asp)*

CSS Pseudo-element

- A CSS pseudo-element is used to style specified parts of an element.
- For example, it can be used to:
- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element



```
p::first-line {
  color: #ff0000;
  font-variant: small-caps;
}
```

***Figure 48** – Pseudo element (**Source**: https://www.w3schools.com/css/css_pseudo_elements.asp)*

The ::first-line pseudo-element is used to add a special style to the first line of a text.

CSS Opacity

- The opacity property specifies the opacity/transparency of an element.
- The opacity property can take a value from 0.0 - 1.0. The lower value, the more transparent

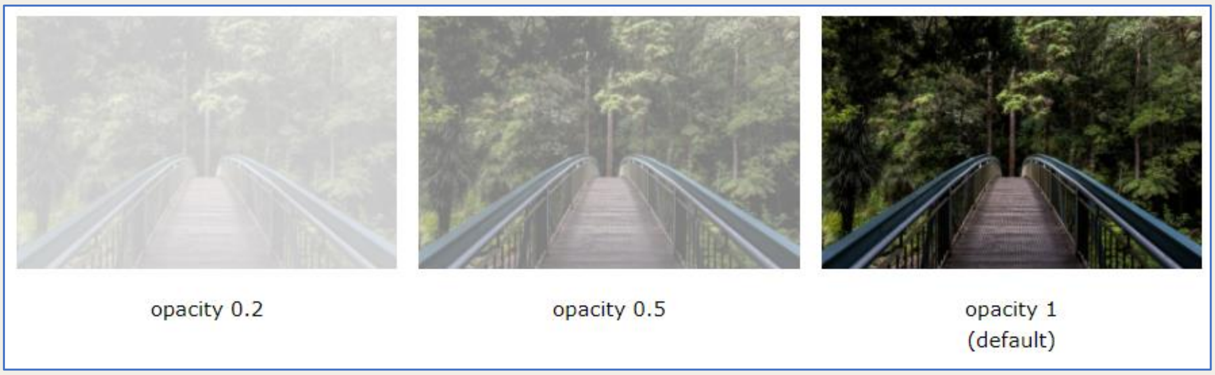


Figure 49 *– CSS Opacity (**Source**: https://www.w3schools.com/css/css_image_transparency.asp)*

CSS Navigation Bar

With CSS you can transform boring HTML menus into good-looking navigation bars.

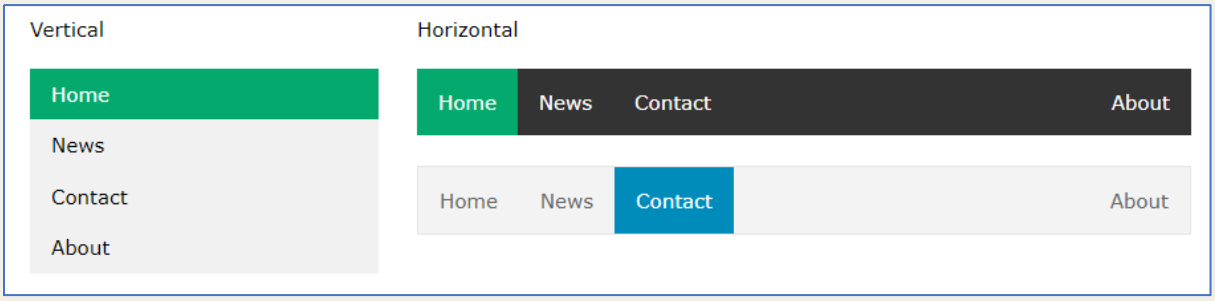


Figure 50 *– Navbar (**Source**: https://www.w3schools.com/css/css_navbar.asp)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

CSS Dropdowns

With CSS you can transform boring HTML menus into good-looking navigation bars.

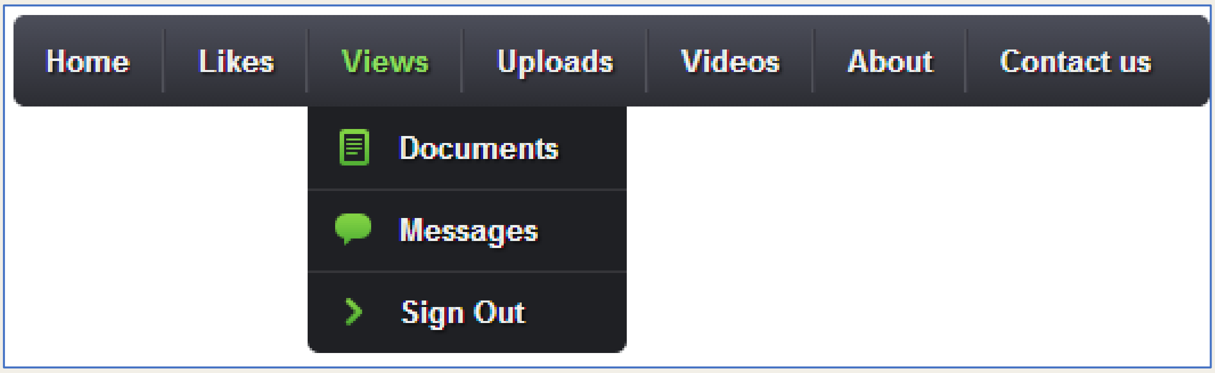


*Figure 51 – Dropdown (**Source**: https://www.w3schools.com/css/css_dropdowns.asp)*

CSS Image Gallery

CSS can be used to create an image gallery.

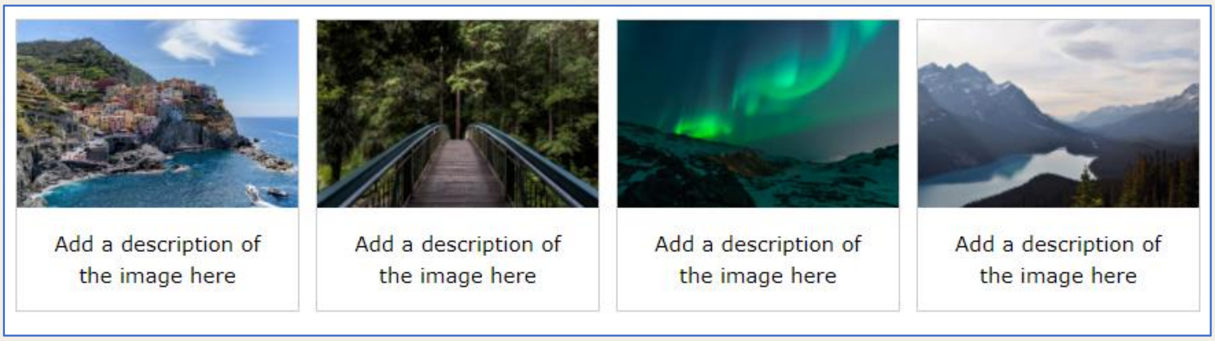

Add a description of the image here
Add a description of the image here
Add a description of the image here
Add a description of the image here

*Figure 52 – Image gallery (**Source**: https://www.w3schools.com/css/css_image_gallery.asp)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

CSS Image Sprites

- An image sprite is a collection of images put into a single image.
- A web page with many images can take a long time to load and generates multiple server requests.
- Using image sprites will reduce the number of server requests and save bandwidth.

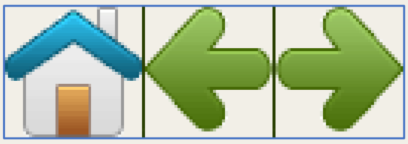
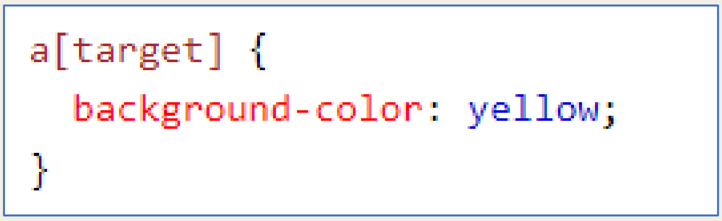


***Figure 53** – Image sprites (**Source**: https://www.w3schools.com/css/css_image_sprites.asp)*

CSS Attr Selectors

- The [attribute] selector is used to select elements with a specified attribute.
- The following example selects all <a> elements with a target attribute:

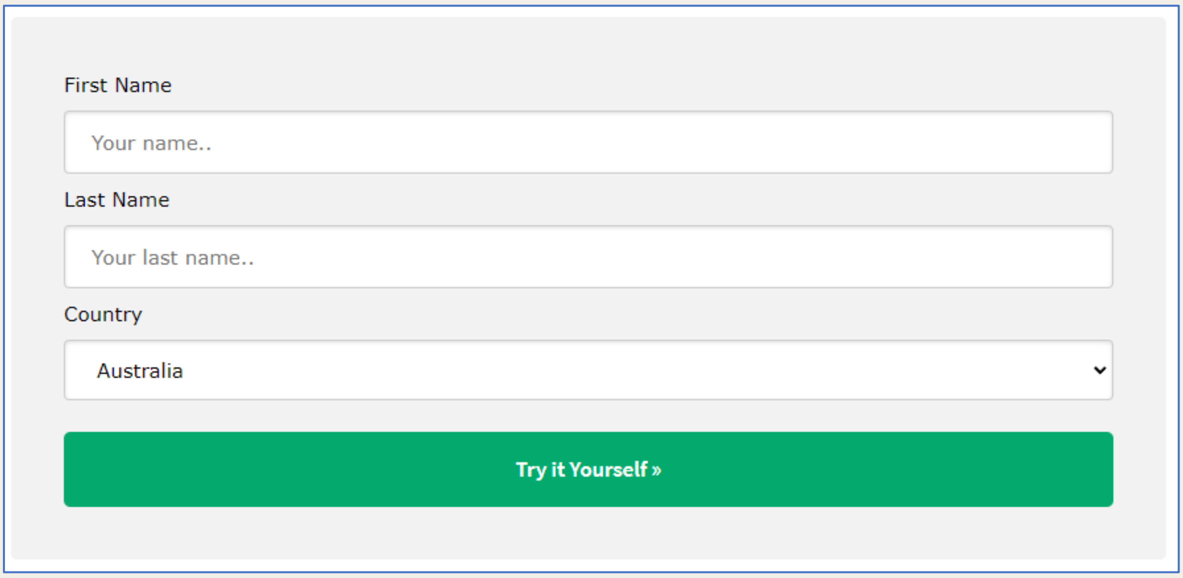


```
a[target] {
  background-color: yellow;
}
```

***Figure 54** – Attr Selector (**Source**: https://www.w3schools.com/css/css_attribute_selectors.asp)*

CSS Forms

The look of an HTML form can be greatly improved with CSS:



First Name

Your name..

Last Name

Your last name..

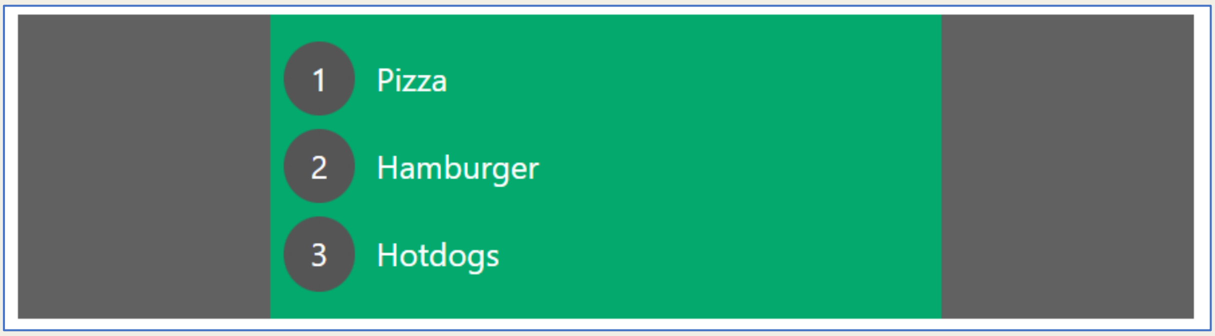
Country

Australia

Figure 55 – *Forms* (**Source**: https://www.w3schools.com/css/css_form.asp)

CSS Counters

CSS counters are "variables" maintained by CSS whose values can be incremented by CSS rules (to track how many times they are used). Counters let you adjust the appearance of content based on its placement in the document.



1 Pizza

2 Hamburger

3 Hotdogs

Figure 56 – *Counter* (**Source**: https://www.w3schools.com/css/css_counters.asp)

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

CSS Website Layout

A website is often divided into headers, menus, content and a footer:

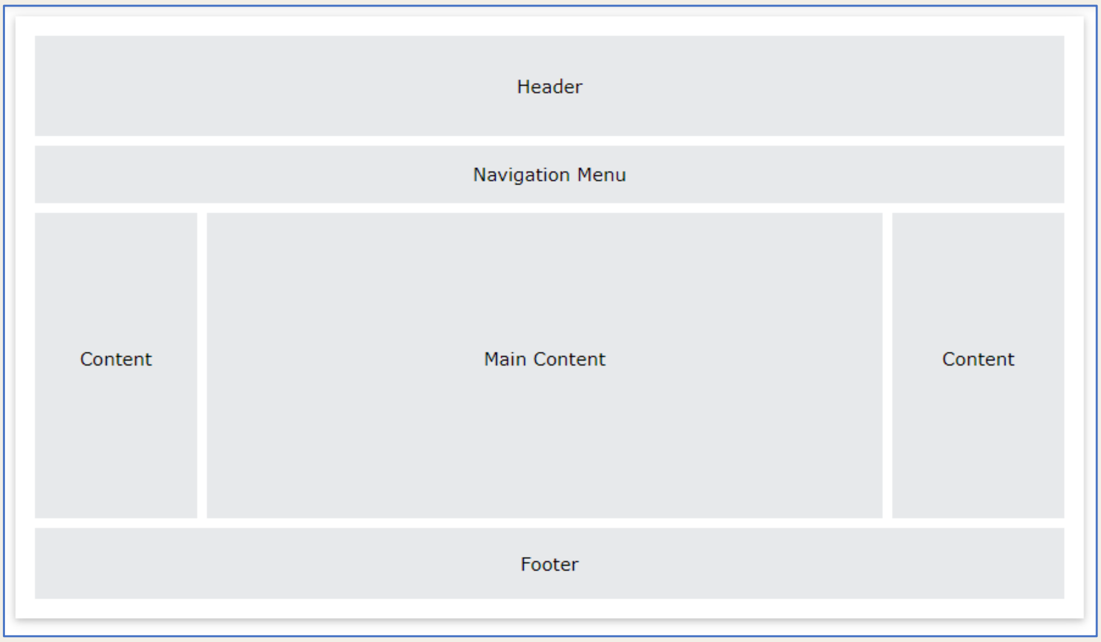
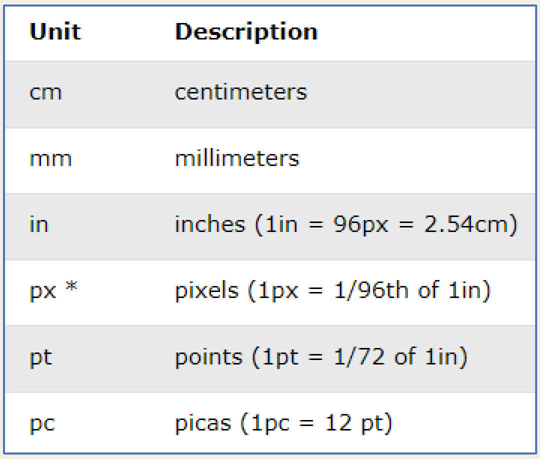


***Figure 57** – Website layout (**Source**: https://www.w3schools.com/css/css_website_layout.asp)*

CSS Units

- CSS has several different units for expressing a length.
- Many CSS properties take "length" values, such as width, margin, padding, font-size, etc,
- Length is a number followed by a length unit, such as 10px, 2em, etc,

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

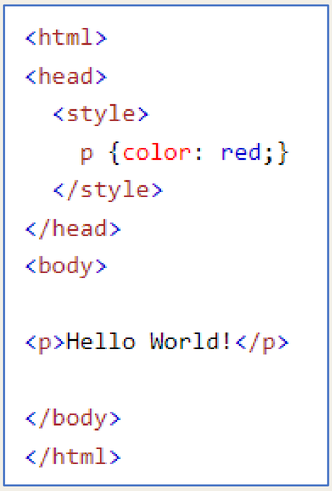


Unit	Description
cm	centimeters
mm	millimeters
in	inches (1in = 96px = 2.54cm)
px *	pixels (1px = 1/96th of 1in)
pt	points (1pt = 1/72 of 1in)
pc	picas (1pc = 12 pt)

Figure 58 *– Units (**Source**: https://www.w3schools.com/css/css_units.asp)*

CSS Specificity

- If there are two or more CSS rules that point to the same element, the selector with the highest specificity value will "win", and its style declaration will be applied to that HTML element.
- Think of specificity as a score/rank that determines which style declaration are ultimately applied to an element.



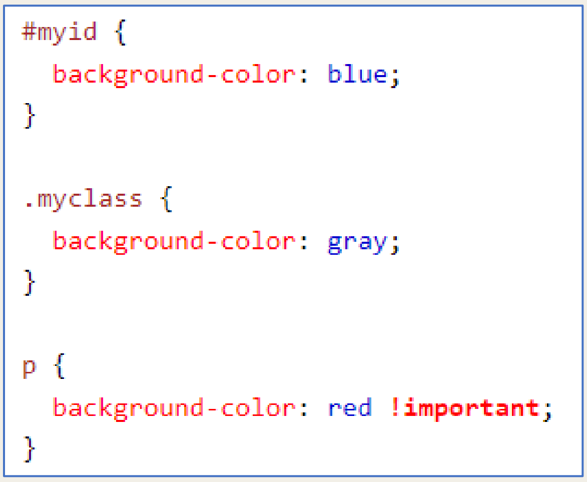
```
<html>
<head>
  <style>
    p {color: red;}
  </style>
</head>
<body>

<p>Hello World!</p>

</body>
</html>
```

Figure 59 *– Specificity (**Source**: https://www.w3schools.com/css/css_specificity.asp)*

In this example, we have used the "p" element as selector and specified a red color for this element. The text will be red.

CSS !important

- The !important rule in CSS is used to add more importance to a property/value than normal.
- In fact, if you use the !important rule, it will override ALL previous styling rules for that specific property on that element!

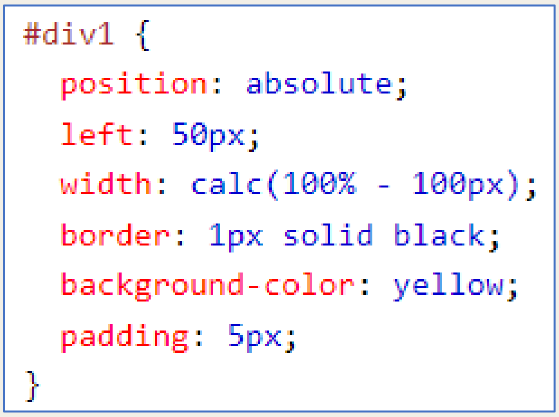


```css
#myid {
  background-color: blue;
}

.myclass {
  background-color: gray;
}

p {
  background-color: red !important;
}
```

Figure 60 *– !important (**Source**: https://www.w3schools.com/css/css_important.asp)*

CSS Math Function

- The CSS math functions allow mathematical expressions to be used as property values. Some Math functions are: calc(), max() and min() functions.
- Example: the use of calc() to calculate the width of a <div> element:



```
#div1 {
  position: absolute;
  left: 50px;
  width: calc(100% - 100px);
  border: 1px solid black;
  background-color: yellow;
  padding: 5px;
}
```

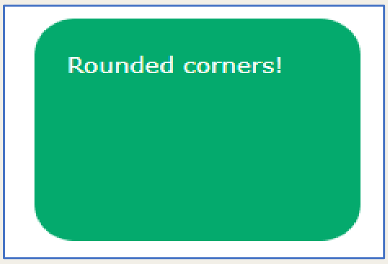
***Figure 61** – Math function (**Source**: https://www.w3schools.com/css/css_math_functions.asp)*

3.2. CSS Advanced

CSS rounded corners

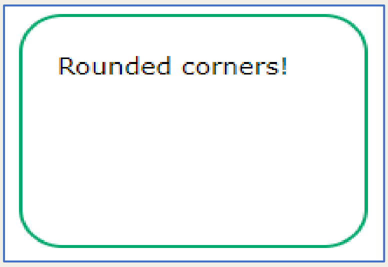
The border-radius property defines the radius of an element's corners.

Tip: This property allows you to add rounded corners to elements!



Rounded corners!

*Figure 62 – Rounded corners (**Source**: https://www.w3schools.com/css/css3_borders.asp)*



Rounded corners!

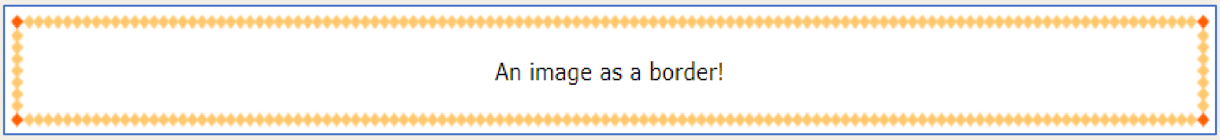
*Figure 63 – Rounded corners (**Source**: https://www.w3schools.com/css/css3_borders.asp)*

CSS Border Images

The CSS border-image property allows you to specify an image to be used instead of the normal border around an element.

The property has three parts:

- The image to use as the border
- Where to slice the image
- Define whether the middle sections should be repeated or stretched



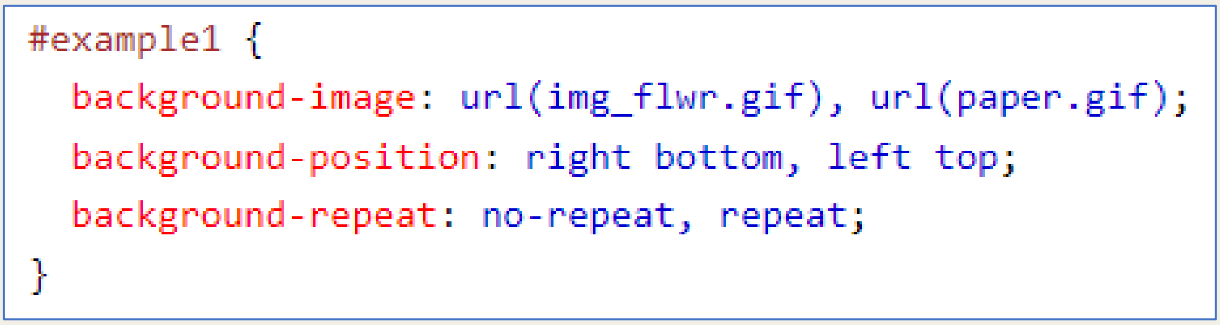
An image as a border!

*Figure 64 – Border image (**Source**: https://www.w3schools.com/css/css3_border_images.asp)*

CSS Backgrounds

CSS allows you to add multiple background images for an element, through the background-image property.

The following example has two background images, the first image is a flower (aligned to the bottom and right) and the second image is a paper background (aligned to the top-left corner).



```
#example1 {
  background-image: url(img_flwr.gif), url(paper.gif);
  background-position: right bottom, left top;
  background-repeat: no-repeat, repeat;
}
```

*Figure 65 – Backgrounds (**Source**: https://www.w3schools.com/css/css3_backgrounds.asp)*

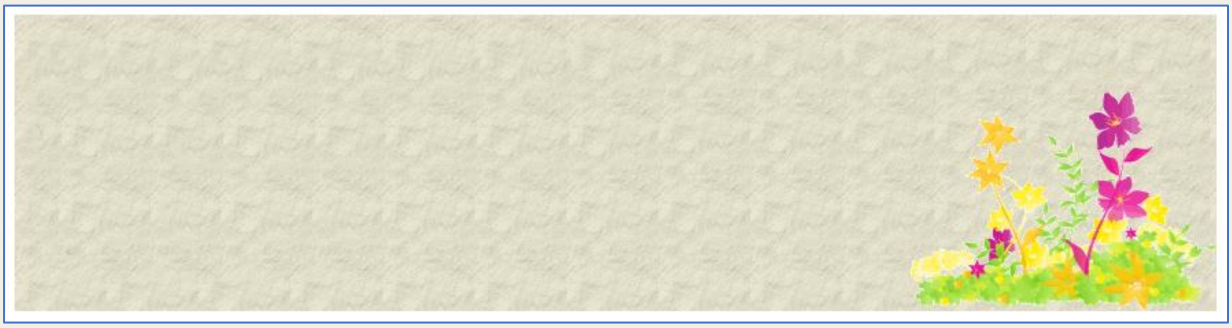
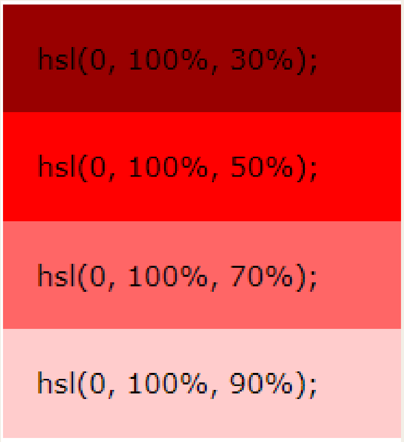


*Figure 66 – Backgrounds (**Source**: https://www.w3schools.com/css/css3_backgrounds.asp)*

CSS Colours

- CSS supports +140 color names, HEX values, RGB values, RGBA values, HSL values, HSLA values, and opacity.

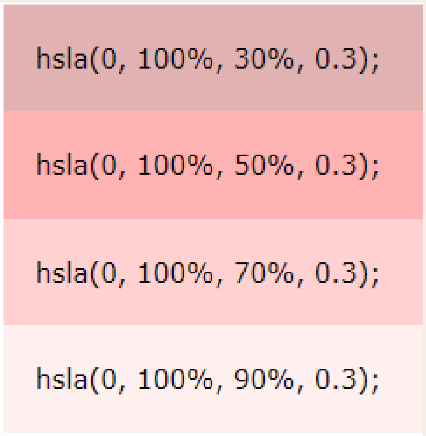


hsl(0, 100%, 30%);

hsl(0, 100%, 50%);

hsl(0, 100%, 70%);

hsl(0, 100%, 90%);



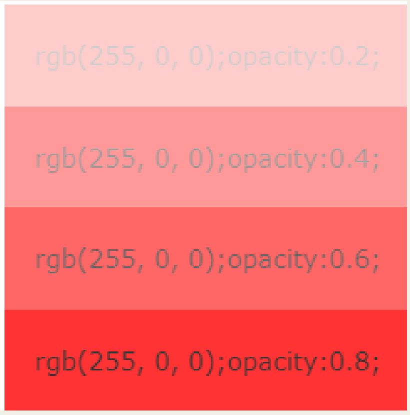
hsla(0, 100%, 30%, 0.3);

hsla(0, 100%, 50%, 0.3);

hsla(0, 100%, 70%, 0.3);

hsla(0, 100%, 90%, 0.3);

Figures 67 and 68 *– CSS Colours (**Source**: https://www.w3schools.com/css/css3_colors.asp)*

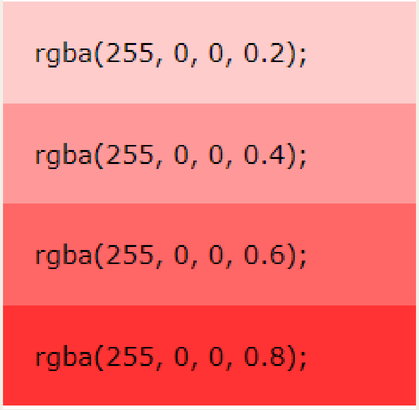


rgb(255, 0, 0);opacity:0.2;

rgb(255, 0, 0);opacity:0.4;

rgb(255, 0, 0);opacity:0.6;

rgb(255, 0, 0);opacity:0.8;



rgba(255, 0, 0, 0.2);

rgba(255, 0, 0, 0.4);

rgba(255, 0, 0, 0.6);

rgba(255, 0, 0, 0.8);

Figures 69 and 70 *– CSS Colours (**Source**: https://www.w3schools.com/css/css3_colors.asp)*

CSS Colour Keywords

This page will explain the transparent, currentcolor, and inherit keywords:

- The transparent keyword is used to make a color transparent. This is often used to make a transparent background color for an element.
- The currentcolor keyword is like a variable that holds the current value of the colour property of an element.

This keyword can be useful if you want a specific colour to be consistent in an element or a page.

The inherit keyword specifies that a property should inherit its value from its parent element.

The inherit keyword can be used for any CSS property, and on any HTML element.

CSS Gradient

CSS gradients let you display smooth transitions between two or more specified colours.

CSS defines three types of gradients:

- Linear Gradients (goes down/up/left/right/diagonally)
- Radial Gradients (defined by their center)
- Conic Gradients (rotated around a center point)

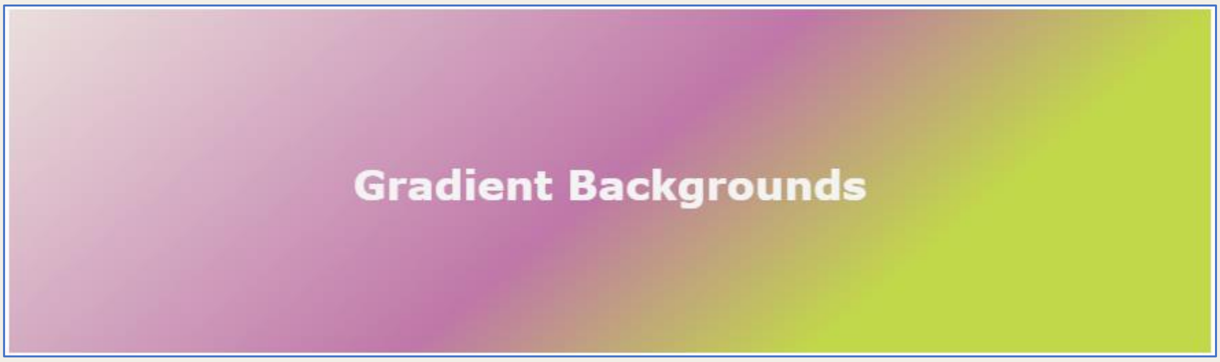


***Figure 71** – Gradient Backgrounds (**Source**: https://www.w3schools.com/css/css3_gradients.asp)*

CSS Shadows

With CSS you can add shadow to text and to elements.

In these chapters you will learn about the following properties :

- Text-shadow
- Box-shadow

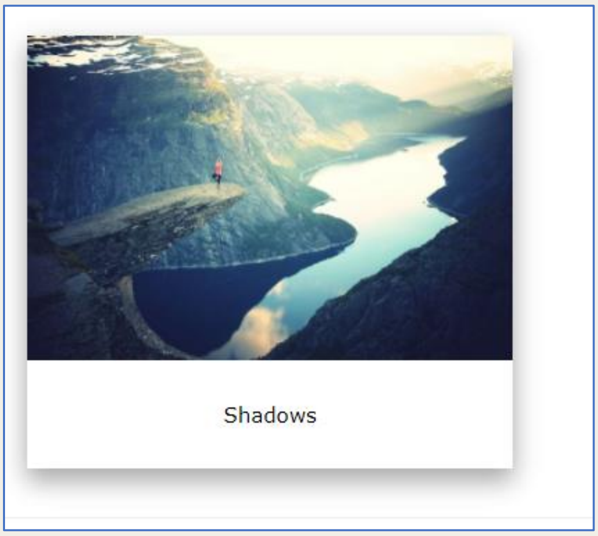
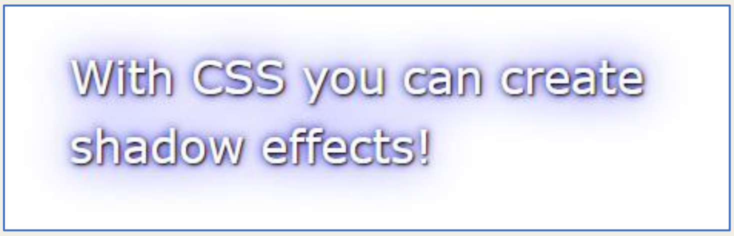


***Figure 72** – Shadows (**Source**: https://www.w3schools.com/css/css3_shadows.asp)*



With CSS you can create shadow effects!

Figure 73 – *Shadows* (**Source**: https://www.w3schools.com/css/css3_shadows.asp)

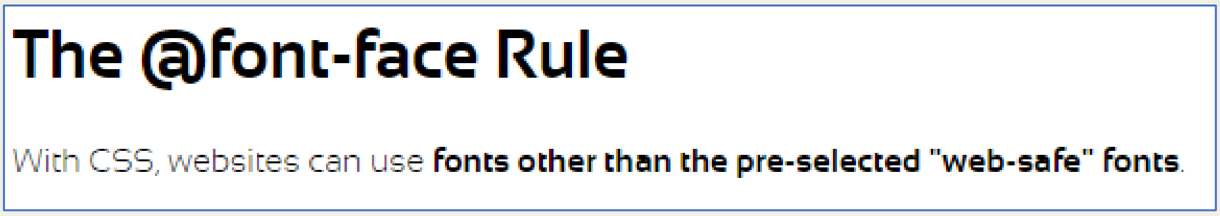
CSS Text Effects

In this chapter you will learn about Text-overflow, word-wrap, word-break, writing-mode:

- The text-overflow property specifies how overflowed content that is not displayed should be signaled to the user.
- The CSS word-wrap property allows long words to be able to be broken and wrap onto the next line.
- The CSS word-break property specifies line breaking rules.
- The CSS writing-mode property specifies whether lines of text are laid out horizontally or vertically.

CSS Web Fonts

- Web fonts allow Web designers to use fonts that are not installed on the user's computer.
- When you have found/bought the font you wish to use, just include the font file on your web server, and it will be automatically downloaded to the user when needed.
- Your "own" fonts are defined within the CSS @font-face rule



The @font-face Rule

With CSS, websites can use **fonts other than the pre-selected "web-safe" fonts**.

Figure 74 – Web fonts (**Source**: https://www.w3schools.com/css/css3_fonts.asp)

CSS 2D transforms

CSS transforms allow you to move, rotate, scale, and skew elements.

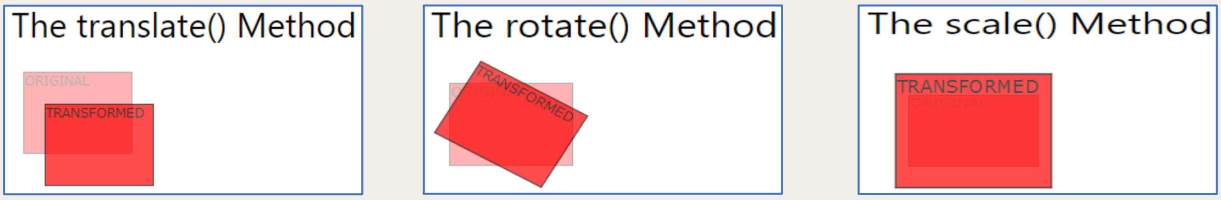


Figure 75 – 2D Transform (**Source**: https://www.w3schools.com/css/css3_2dtransforms.asp)

CSS 3D transforms

With the CSS transform property you can use the following 3D transformation methods:

- RotateX()
- RotateY()
- RotateZ()

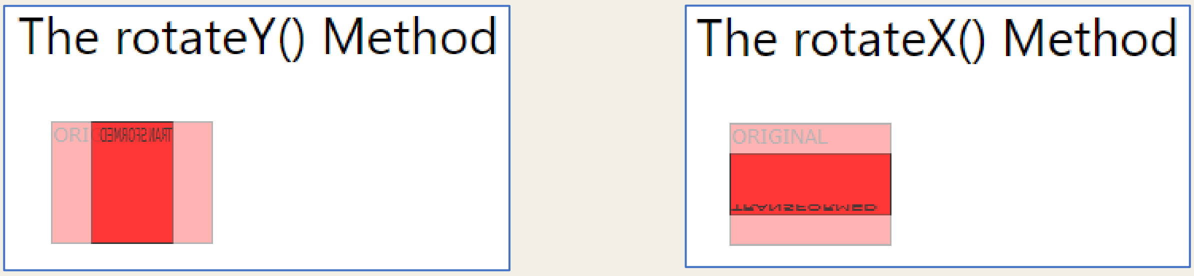
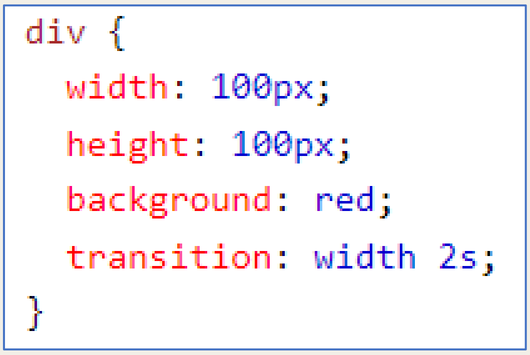


Figure 76 – 3D Transform (**Source**: https://www.w3schools.com/css/css3_3dtransforms.asp)

CSS Transitions

- CSS transitions allows you to change property values smoothly, over a given duration.
- To create a transition effect, you must specify two things:
  - the CSS property you want to add an effect to
  - the duration of the effect
- The following example shows a 100px * 100px red <div> element. The <div> element has also specified a transition effect for the width property, with a duration of 2 seconds:

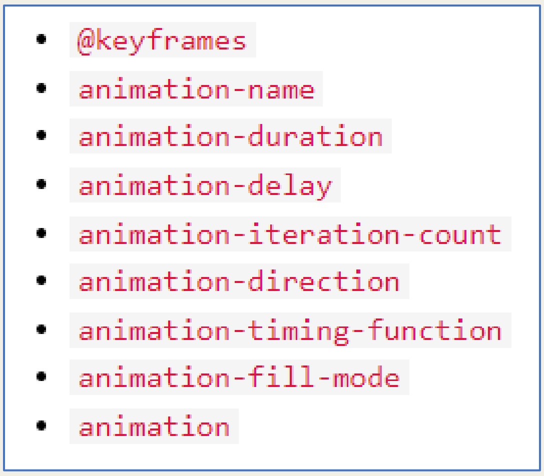


```
div {
  width: 100px;
  height: 100px;
  background: red;
  transition: width 2s;
}
```

Figure 77 *– Transitions (**Source**: https://www.w3schools.com/css/css3_transitions.asp)*

CSS Animations

- An animation lets an element gradually change from one style to another.
- You can change as many CSS properties you want, as many times as you want.
- To use CSS animation, you must first specify some keyframes for the animation.
- Keyframes hold what styles the element will have at certain times.
- These are the main the main animation’s properties :



- `@keyframes`
- `animation-name`
- `animation-duration`
- `animation-delay`
- `animation-iteration-count`
- `animation-direction`
- `animation-timing-function`
- `animation-fill-mode`
- `animation`

***Figure 78** – Animations (**Source**: https://www.w3schools.com/css/css3_animations.asp)*

CSS Tooltips

- A tooltip is often used to specify extra information about something when the user moves the mouse pointer over an element:

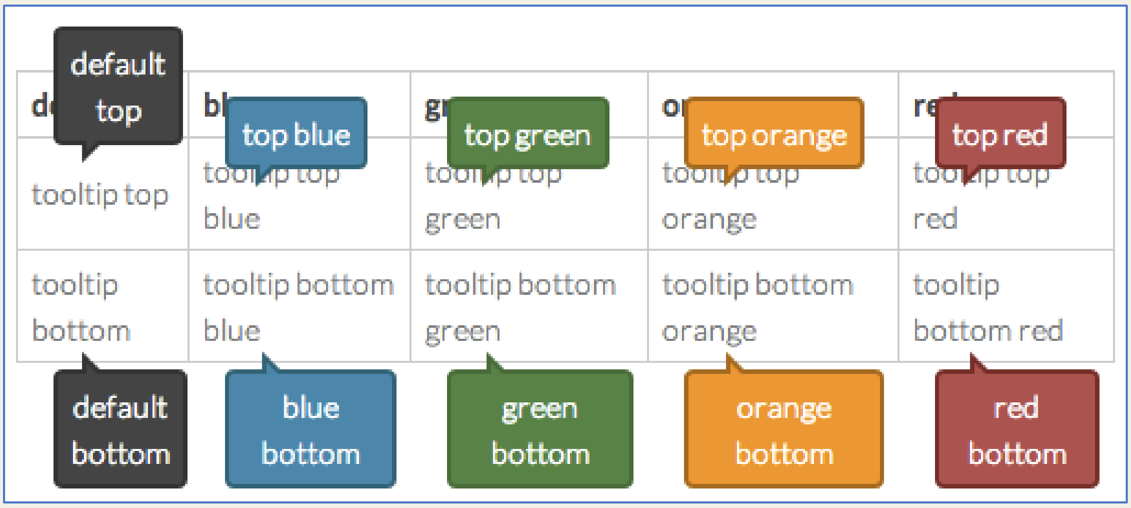


***Figure 79** – Tooltip (**Source**: https://www.w3schools.com/css/css3_animations.asp)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

CSS Style images

- Using CSS to style images allows you to uniformly specify how images should appear across your website with only a few rulesets :

  Add a border, change the shape and size of the image, …

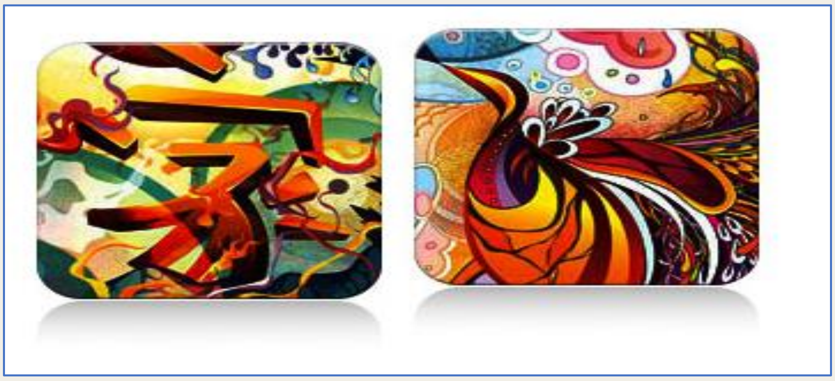


***Figure 80** – Images (**Source:** https://www.w3schools.com/css/css3_image_reflection.asp)*

CSS image reflection

- The box-reflect property is used to create an image reflection.
- The value of the The box-reflect property can be : below, above, left, or right

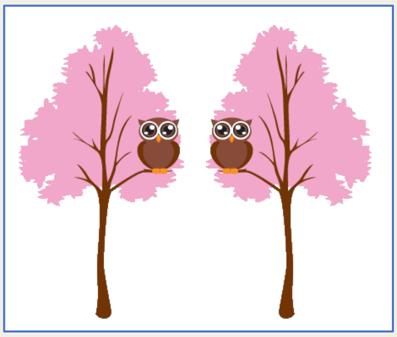


***Figure 81** – Image reflection (**Source**: https://www.w3schools.com/css/css3_image_reflection.asp)*

CSS Object-fit

The CSS object-fit property is used to specify how an <img> or <video> should be resized to fit its container.

This property tells the content to fill the container in a variety of ways; such as "preserve that aspect ratio" or "stretch up and take up as much space as possible".

Look at the following image from Paris. This image is 400 pixels wide and 300 pixels high:

Figure 82 *– Object fit (**Source**: https://www.w3schools.com/css/css3_object-fit.asp)*

CSS Object-position

Let's say that the part of the image that is shown, is not positioned as we want. To position the image, we will use the object-position property

Here we will use the object-position property to position the image so that the great old building is in center:

Figure 83 *– Object position (**Source**: https://www.w3schools.com/css/css3_object-position.asp)*

CSS Masking

- With CSS masking you create a mask layer to place over an element to hide portions of the element partially or fully.
- The CSS mask-image property specifies a mask layer image.
- The mask layer image can be a PNG image, an SVG image, a CSS gradient, or an SVG <mask>element

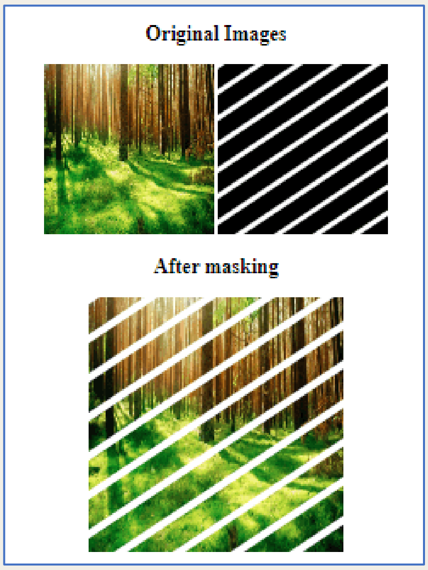


Figure 84 – *Masking (**Source**: https://www.w3schools.com/css/css3_masking.asp)*

CSS Buttons

- There are lots of properties to style a button in CSS
- This is an example:

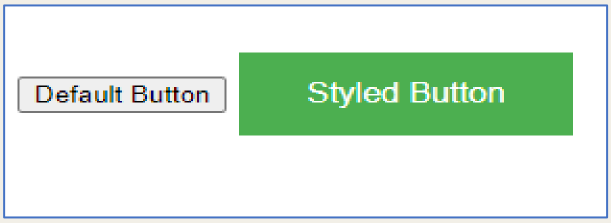
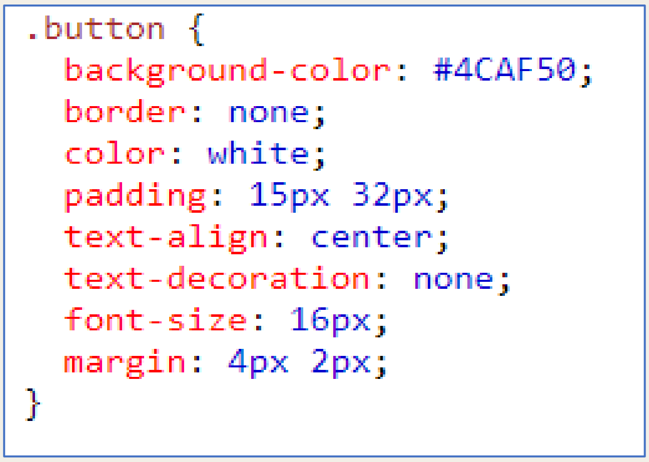


Figure 85 – *Default Button (**Source**: https://www.w3schools.com/css/css3_buttons.asp)*



```
.button {
  background-color: #4CAF50;
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  font-size: 16px;
  margin: 4px 2px;
}
```

Figure 86 – *Button Coding (**Source**: https://www.w3schools.com/css/css3_buttons.asp)*

CSS Pagination

If you have a website with lots of pages, you may wish to add some sort of pagination to each page and these are some examples:

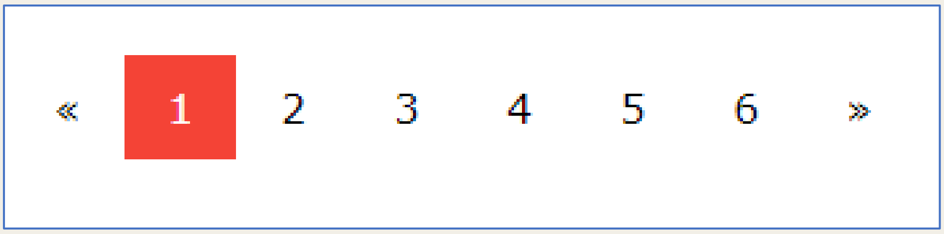


***Figure 87** – Pagination (**Source**: https://www.w3schools.com/css/css3_pagination.asp)*

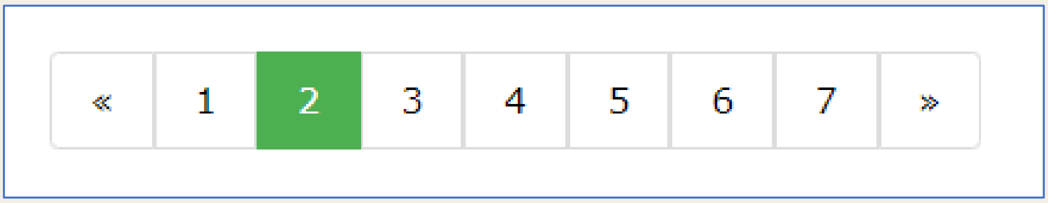
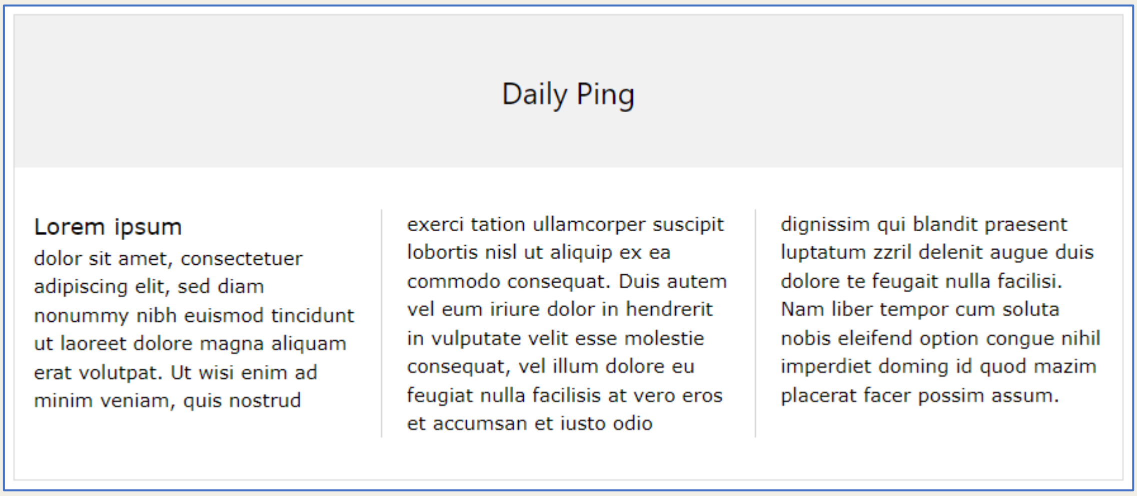


***Figure 88** – Pagination (**Source**: https://www.w3schools.com/css/css3_pagination.asp)*

CSS Multiple Columns

The CSS multi-column layout allows easy definition of multiple columns of text - just like in newspapers:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



Daily Ping

Lorem ipsum

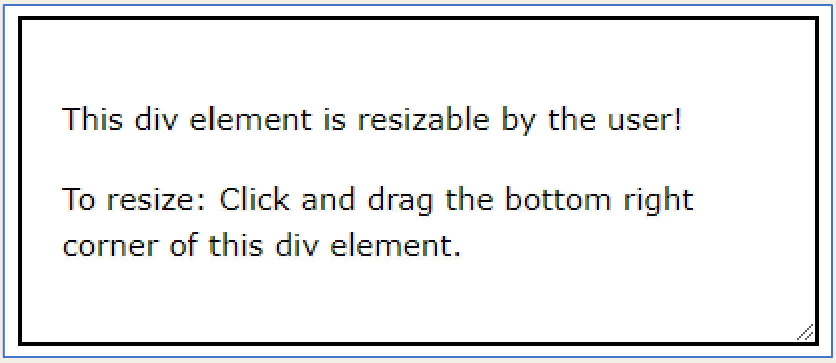
dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum.

***Figure 89** – Multiple Columns (**Source**: https://www.w3schools.com/css/css3_multiple_columns.asp)*

CSS User Interface

In this chapter you will learn about the following CSS user interface properties:

- The resize property specifies if (and how) an element should be resizable by the user.

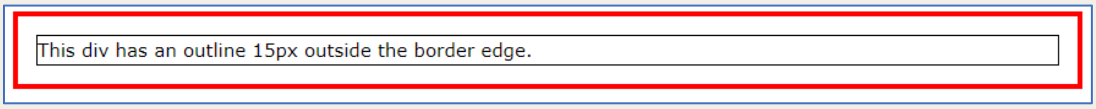


This div element is resizable by the user!

To resize: Click and drag the bottom right corner of this div element.

***Figure 90** – UI (**Source**: https://www.w3schools.com/css/css3_user_interface.asp)*

- The outline-offset property adds space between an outline and the edge or border of an element.



This div has an outline 15px outside the border edge.

Figure 91 – UI (Source: https://www.w3schools.com/css/css3_user_interface.asp)

CSS Variables

- The var() function is used to insert the value of a CSS variable.
- CSS variables have access to the DOM, which means that you can create variables with local or global scope, change the variables with JavnceaScript, and change the variables based on media queries.
- A good way to use CSS variables is when it comes to the colors of your design. Instead of copy and paste the same colors over and over again, you can place them in variables.

CSS Box sizing

- The box-sizing property in CSS defines how the user should calculate the total width and height of an element i.e padding and borders, are to be included or not.

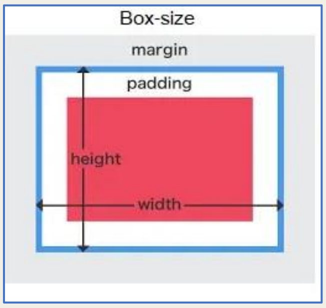


Figure 92 – Box sizing (Source: https://www.w3schools.com/css/css3_box-sizing.asp)

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

CSS Media Queries

Media queries in CSS3 extended the CSS2 media types idea: Instead of looking for a type of device, they look at the capability of the device.

Media queries can be used to check many things, such as:

- width and height of the viewport
- width and height of the device
- orientation (is the tablet/phone in landscape or portrait mode?)
- resolution

Using media queries are a popular technique for delivering a tailored style sheet to desktops, laptops, tablets, and mobile phones (such as iPhone and Android phones).

CSS MQ examples

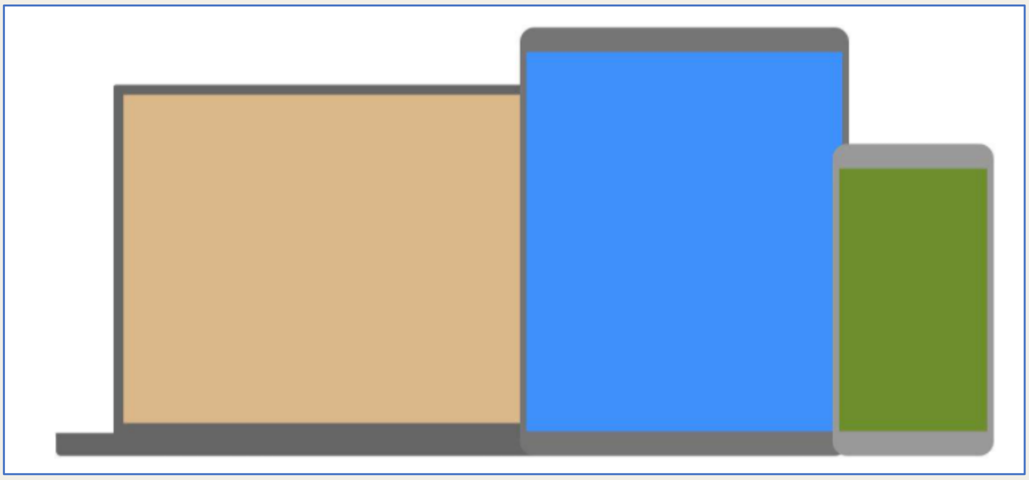


***Figure 93** – MQ Example (**Source**: https://www.w3schools.com/css/css3_mediaqueries_ex.asp)*

CSS Flexbox

The Flexible Box Layout Module makes it easier to design flexible responsive layout structure without using float or positioning.

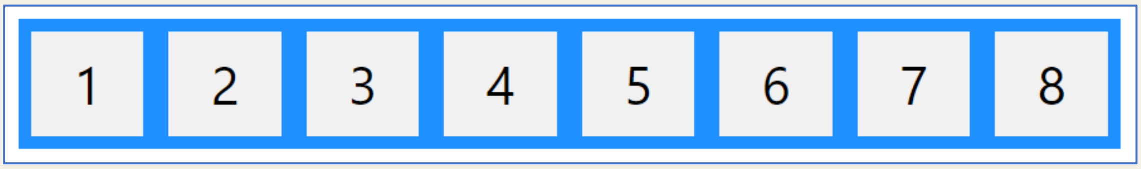


Figure 94 *– Flexbox (****Source****: https://www.w3schools.com/css/css3_flexbox.asp)*

3.3. CSS Responsive Web Design

RWD Intro

Responsive web design makes your web page look good on all devices.

Responsive web design uses only HTML and CSS.

Web pages can be viewed using many different devices: desktops, tablets, and phones. Your web page should look good, and be easy to use, regardless of the device.

Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device:

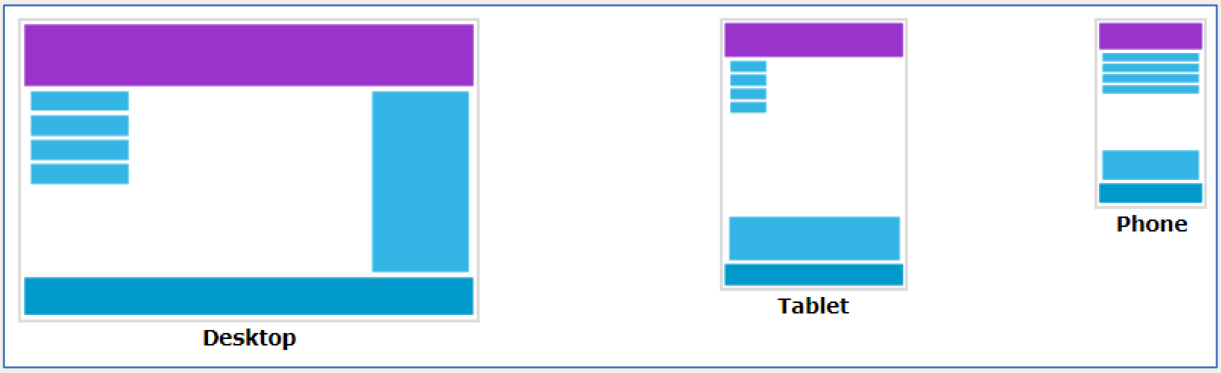


Figure 95 *– RWD (**Source**: https://www.w3schools.com/css/css_rwd_intro.asp)*

It is called responsive web design when you use CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen.

RWD Viewport

The viewport is the user's visible area of a web page.

The viewport varies with the device and will be smaller on a mobile phone than on a computer screen.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Before tablets and mobile phones, web pages were designed only for computer screens, and it was common for web pages to have a static design and a fixed size.

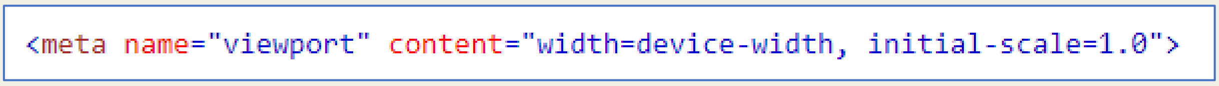
Then, when we started surfing the internet using tablets and mobile phones, fixed size web pages were too large to fit the viewport. To fix this, browsers on those devices scaled down the entire web page to fit the screen.

This was not perfect!! But a quick fix.

Setting the Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the <meta> tag.

You should include the following <meta> viewport element in all your web pages:



```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

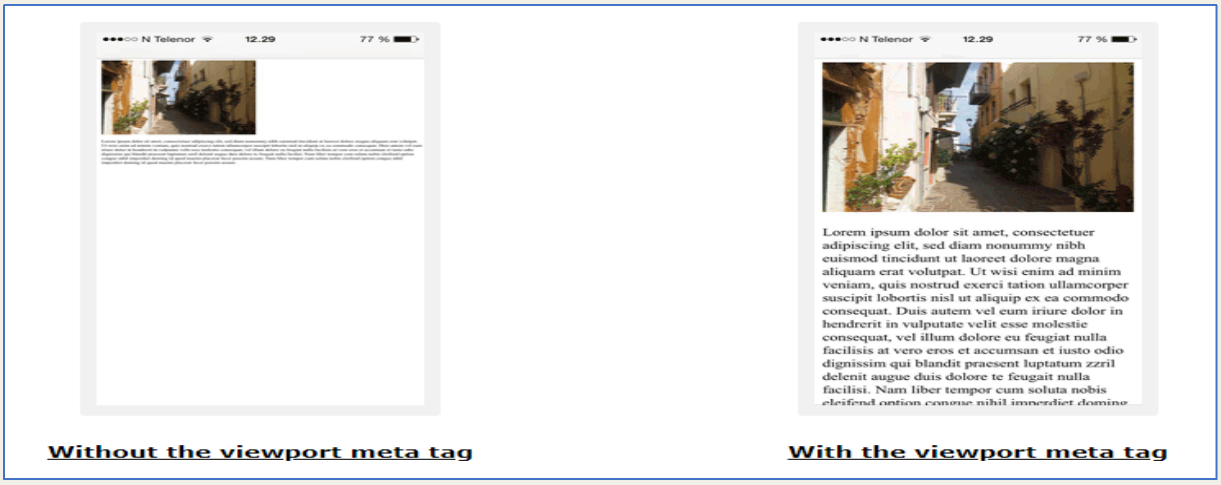
Figure 96 *– Setting Viewport (**Source**: https://www.w3schools.com/css/css_rwd_viewport.asp)*

This gives the browser instructions on how to control the page's dimensions and scaling.

The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser.

Here is an example of a web page without the viewport meta tag, and the same web page with the viewport meta tag:



Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming

Figure 97 *– Viewport (**Source**: https://www.w3schools.com/css/css_rwd_viewport.asp)*

RWD Grid-View

Many web pages are based on a grid-view, which means that the page is divided into columns:

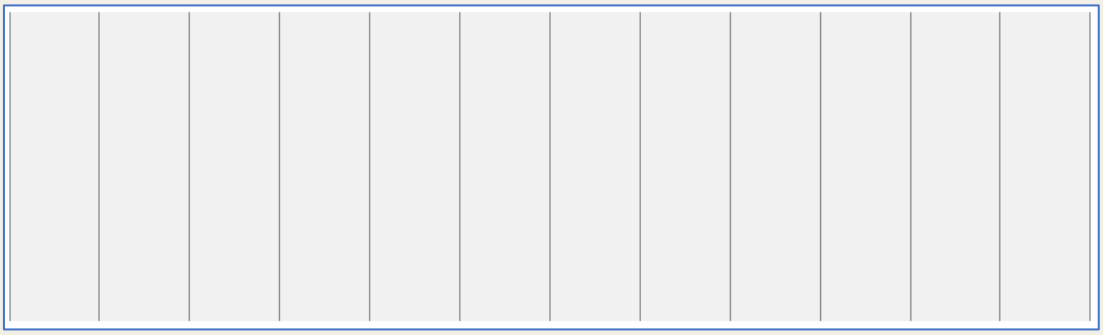


Figure 98 *– Grid-View (**Source**: https://www.w3schools.com/css/css_rwd_grid.asp)*

Many web pages are based on a grid-view, which means that the page is divided into columns:

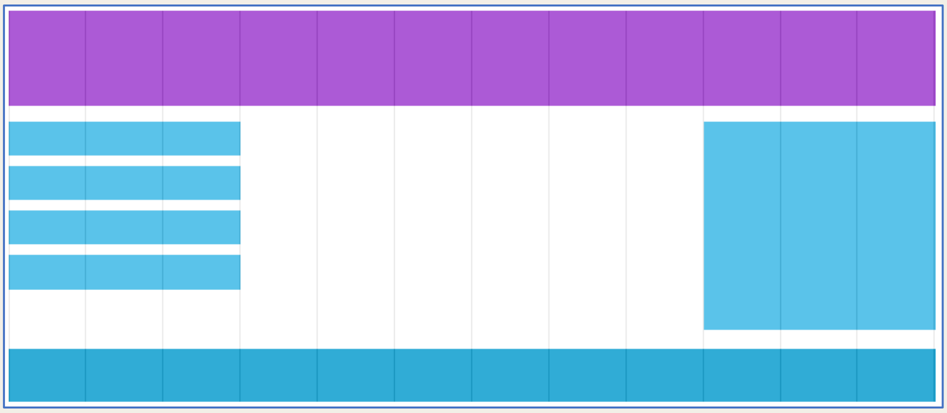
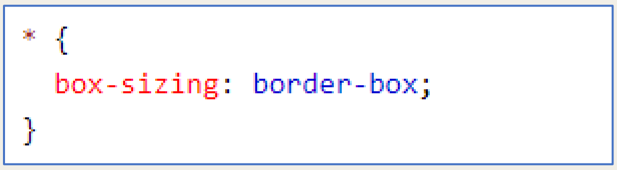


Figure 99 – *Grid-View (Source: https://www.w3schools.com/css/css_rwd_grid.asp)*

Building a responsive Grid-View

First ensure that all HTML elements have the box-sizing property set to border-box. This makes sure that the padding and border are included in the total width and height of the elements.

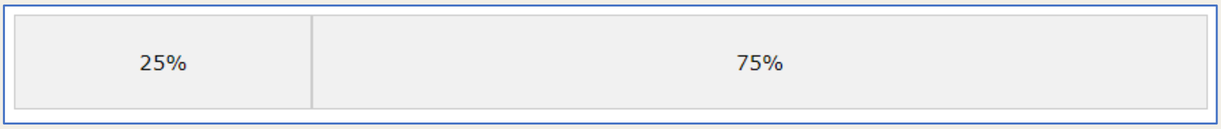
Add the following code in your CSS:



```css
* {
  box-sizing: border-box;
}
```

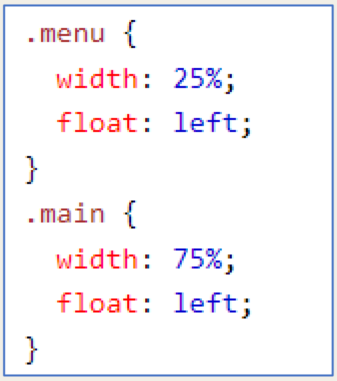
Figure 100 *– Responsive Grid-View (**Source**: https://www.w3schools.com/css/css_rwd_grid.asp)*

The following example shows a simple responsive web page, with two columns:



25% 75%

Figure 101 – *Responsive Grid-View* (**Source**: https://www.w3schools.com/css/css_rwd_grid.asp)

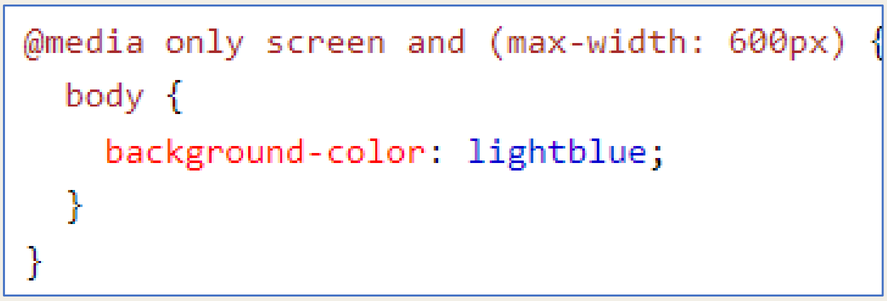


```
.menu {
  width: 25%;
  float: left;
}
.main {
  width: 75%;
  float: left;
}
```

Figure 102 – *Responsive Grid-View* (**Source**: https://www.w3schools.com/css/css_rwd_grid.asp)

RWD Media Queries

- Media query is a CSS technique introduced in CSS3.
- It uses the @media rule to include a block of CSS properties only if a certain condition is true.
- If the browser window is 600px or smaller, the background color will be lightblue:



```
@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

__Figure 103__ – Media query (__Source__: https://www.w3schools.com/css/css_rwd_mediaqueries.asp)

Add a breakpoint

Media queries can help with that. We can add a breakpoint where certain parts of the design will behave differently on each side of the breakpoint.

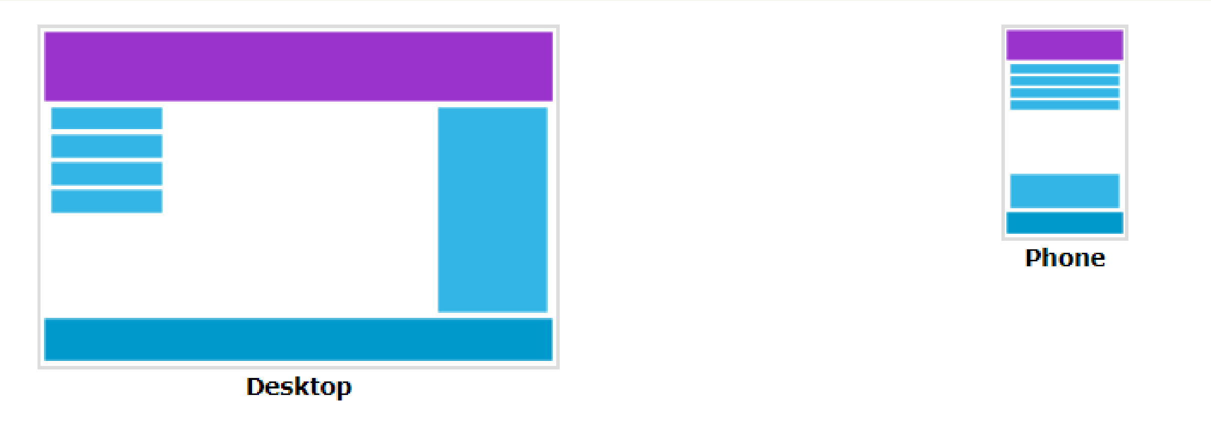
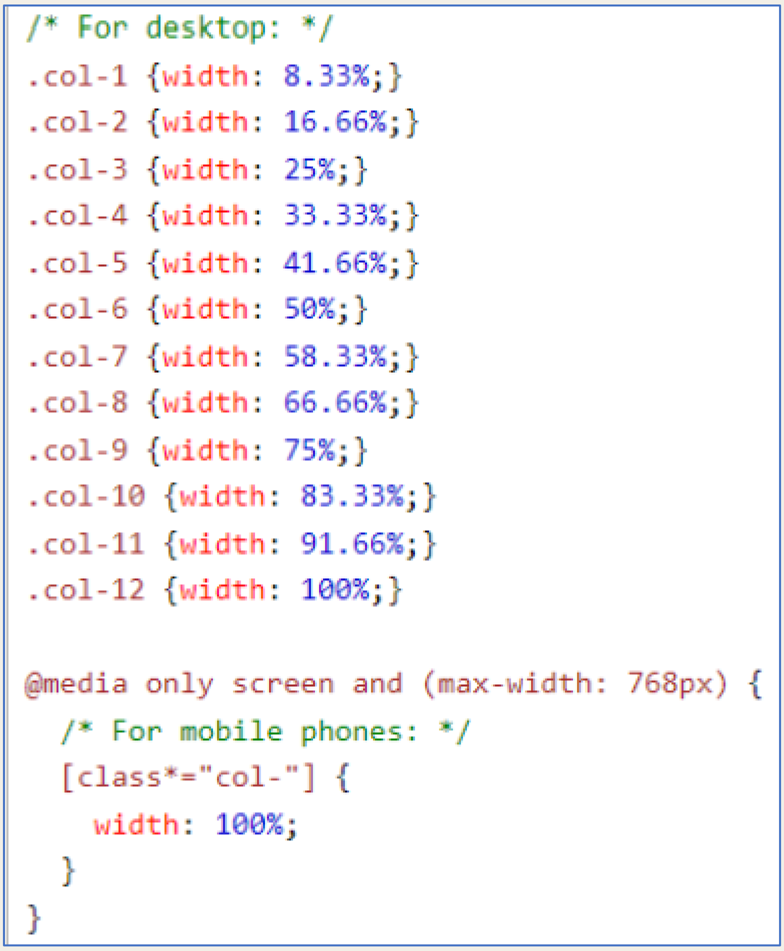


__Figure 104__ – Media query (__Source__: https://www.w3schools.com/css/css_rwd_mediaqueries.asp)

Use a media query to add a breakpoint at 768px:

Example: When the screen (browser window) gets smaller than 768px, each column should have a width of 100%:

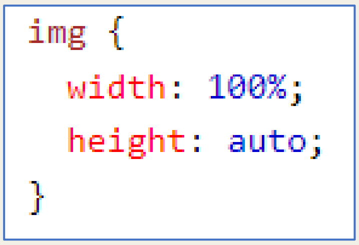


```
/* For desktop: */
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}

@media only screen and (max-width: 768px) {
  /* For mobile phones: */
  [class*="col-"] {
    width: 100%;
  }
}
```

***Figure 105** – Media query (**Source**: https://www.w3schools.com/css/css_rwd_mediaqueries.asp)*

RWD Images

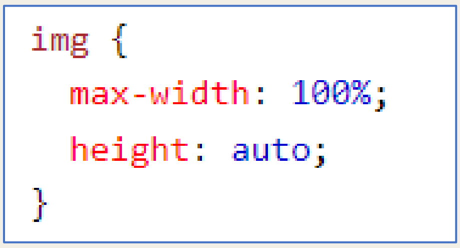
- If the width property is set to a percentage and the height property is set to "auto", the image will be responsive and scale up and down:



```
img {
  width: 100%;
  height: auto;
}
```

***Figure 106** – Image (**Source**: https://www.w3schools.com/css/css_rwd_images.asp)*

- If the max-width property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size:



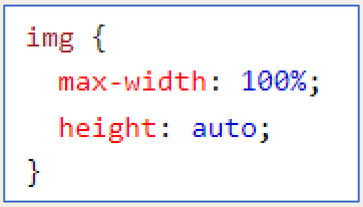
```
img {
  max-width: 100%;
  height: auto;
}
```

***Figure 107** – The max-width property (**Source**: https://www.w3schools.com/css/css_rwd_images.asp)*

Notice that in the example above, the image can be scaled up to be larger than its original size. A better solution, in many cases, will be to use the max-width property instead.

Using the max-width property:

The max-width property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size:

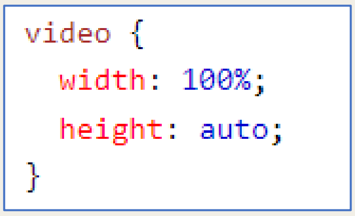


```
img {
  max-width: 100%;
  height: auto;
}
```

***Figure 108** – The max-width property (**Source**: https://www.w3schools.com/css/css_rwd_images.asp)*

RWD Videos

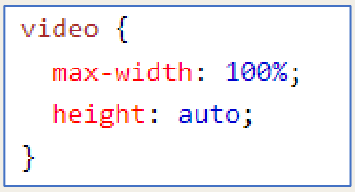
- If the width property is set to 100%, the video player will be responsive and scale up and down:



```
video {
  width: 100%;
  height: auto;
}
```

Figure 109 *– Video (**Source**: https://www.w3schools.com/css/css_rwd_videos.asp)*

- If the max-width property is set to 100%, the video player will scale down if it must, but never scale up to be larger than its original size:



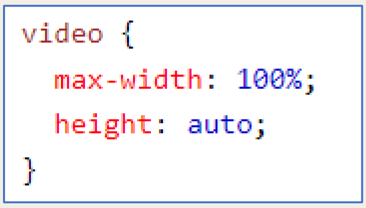
```
video {
  max-width: 100%;
  height: auto;
}
```

Figure 110 *– Video (**Source**: https://www.w3schools.com/css/css_rwd_videos.asp)*

Notice that in the example above, the video player can be scaled up to be larger than its original size. A better solution, in many cases, will be to use the max-width property instead.

Using the max-width property:

If the max-width property is set to 100%, the video player will scale down if it has to, but never scale up to be larger than its original size:



```
video {
  max-width: 100%;
  height: auto;
}
```

Figure 111 *– Video (**Source**: https://www.w3schools.com/css/css_rwd_videos.asp)*

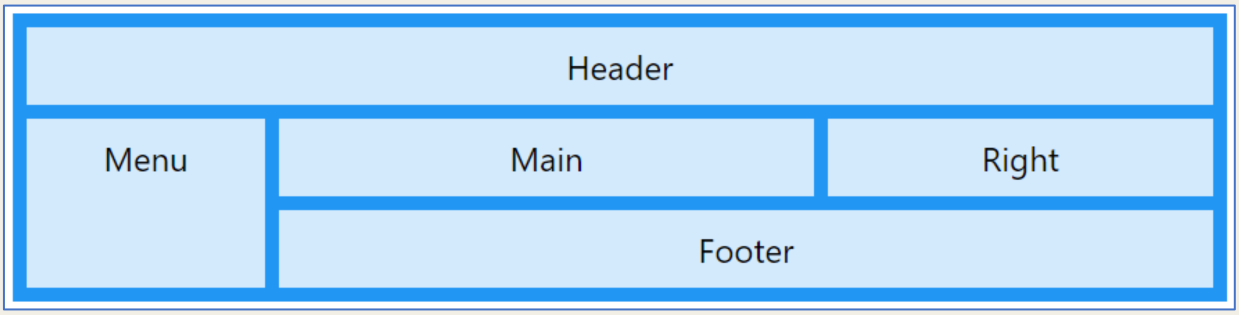
RWD Frameworks

- There are many free CSS Frameworks that offer Responsive Design.
- A great way to create a responsive design, is to use a responsive style sheet, like W3.CSS
- W3.CSS makes it easy to develop sites that look nice at any size!
- Another popular framework is Bootstrap. It uses HTML and CSS to make responsive web pages.

3.4. CSS Grid

Grid intro

- The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

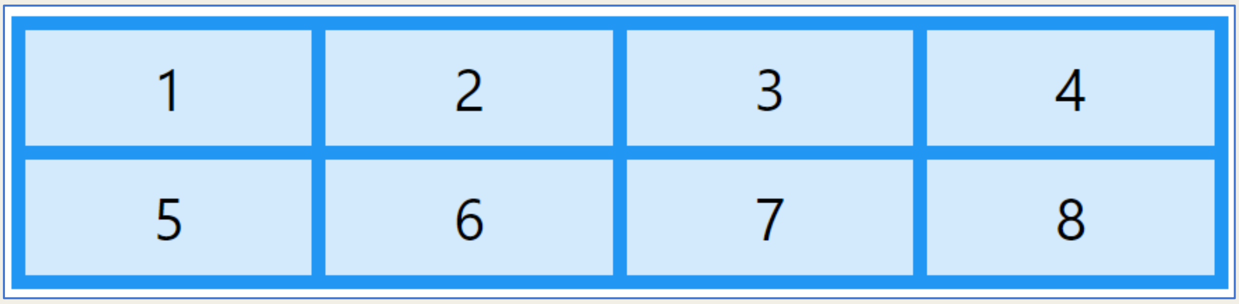


Header
Menu
Main
Right
Footer

***Figure 112** – CSS Grid (**Source**: https://www.w3schools.com/css/css_grid.asp)*

Grid Container

- To make an HTML element behave as a grid container, you have to set the display property to grid or inline-grid.
- Grid containers consist of grid items, placed inside columns and rows.

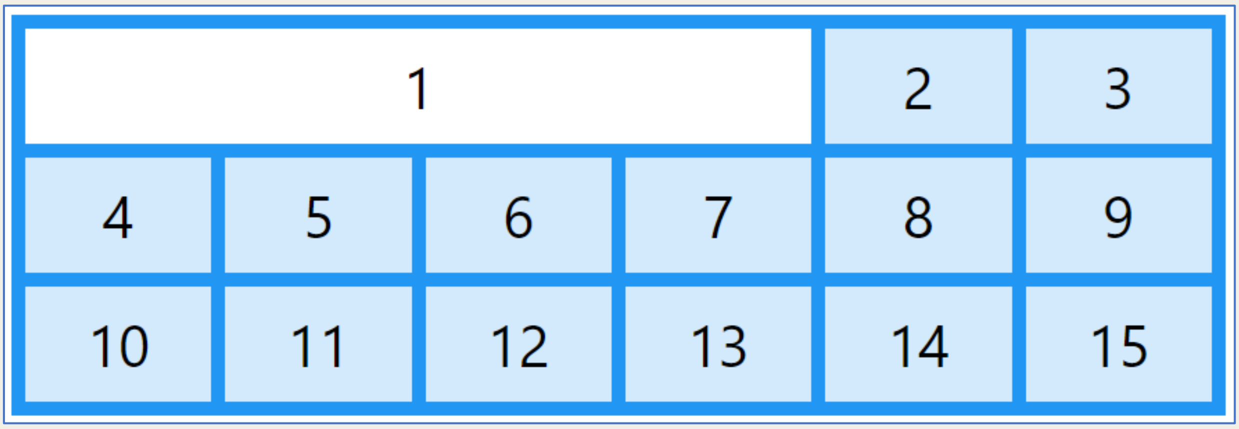


1	2	3	4
5	6	7	8

***Figure 113** – Grid Container (**Source**: https://www.w3schools.com/css/css_grid_container.asp)*

Grid Item

- A grid *container* contains grid *items*.
- By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.
- The grid-column property defines on which column(s) to place an item.
- You define where the item will start, and where the item will end.



1				2	3
4	5	6	7	8	9
10	11	12	13	14	15

***Figure 114** – Grid Items (**Source**: https://www.w3schools.com/css/css_grid_item.asp)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

3.5. CSS SASS

Sass Introduction

What is Sass?

- Sass stands for Syntactically Awesome Stylesheet
- Sass is an extension to CSS
- Sass is a CSS pre-processor
- Sass is completely compatible with all versions of CSS
- Sass reduces repetition of CSS and therefore saves time
- Sass was designed by Hampton Catlin and developed by Natalie Weizenbaum in 2006
- Sass is free to download and use

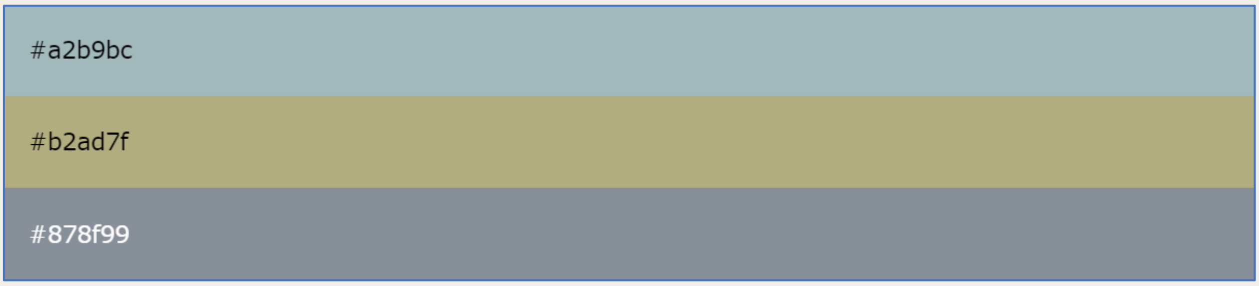
Why Use Sass?

Stylesheets are getting larger, more complex, and harder to maintain. This is where a CSS pre-processor can help.

Sass lets you use features that do not exist in CSS, like variables, nested rules, mixins, imports, inheritance, built-in functions, and other stuff.

A Simple Example why Sass is Useful

Let's say we have a website with three main colors:



#a2b9bc

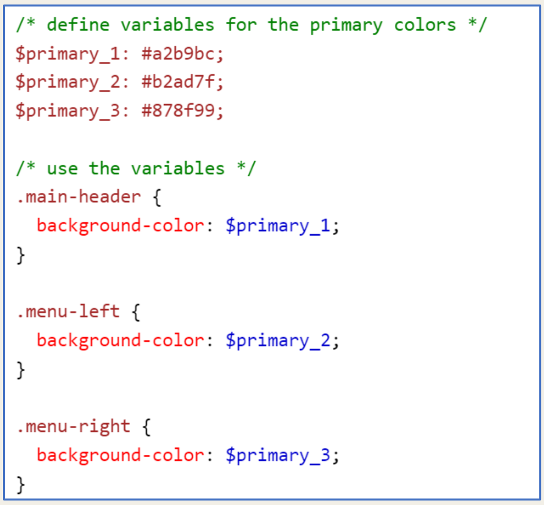
#b2ad7f

#878f99

***Figure 115** – Colours on SASS (**Source**: https://www.w3schools.com/sass/sass_intro.php)*

So, how many times do you need to type those HEX values? A LOT of times. And what about variations of the same colors?

Instead of typing the above values a lot of times, you can use Sass and write this:



```
/* define variables for the primary colors */
$primary_1: #a2b9bc;
$primary_2: #b2ad7f;
$primary_3: #878f99;

/* use the variables */
.main-header {
  background-color: $primary_1;
}

.menu-left {
  background-color: $primary_2;
}

.menu-right {
  background-color: $primary_3;
}
```

***Figure 116** – SASS variable (**Source**: https://www.w3schools.com/sass/sass_variables.php)*

So, when using Sass, and the primary color changes, you only need to change it in one place.

How Does Sass Work?

A browser does not understand Sass code. Therefore, you will need a Sass pre-processor to convert Sass code into standard CSS.

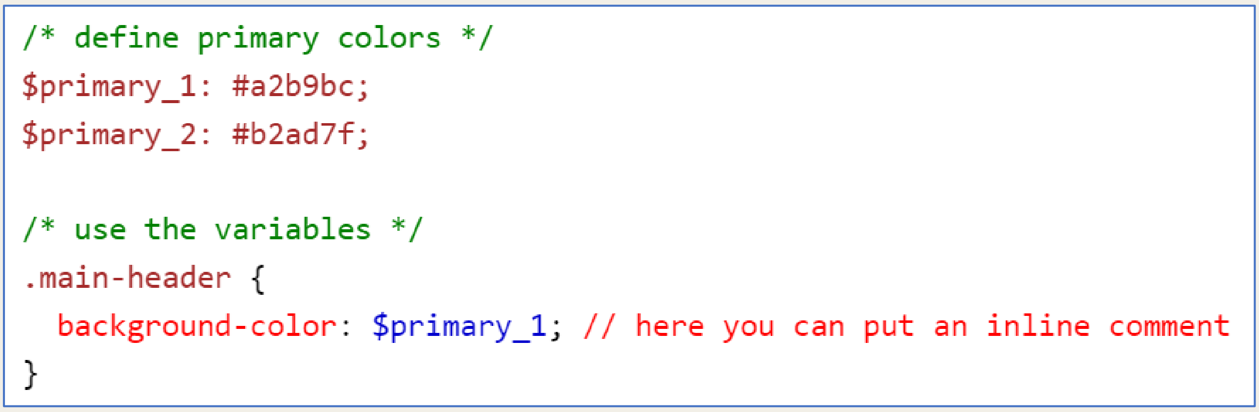
This process is called transpiling. So, you need to give a transpiler (some kind of program) some Sass code and then get some CSS code back.

Sass File Type

Sass files has the ".scss" file extension.

Sass Comments

Sass supports standard CSS comments /* comment */, and in addition it supports inline comments // comment:



```
/* define primary colors */
$primary_1: #a2b9bc;
$primary_2: #b2ad7f;

/* use the variables */
.main-header {
  background-color: $primary_1; // here you can put an inline comment
}
```

***Figure 117** – Comments (**Source**: https://www.w3schools.com/sass/sass_intro.php)*

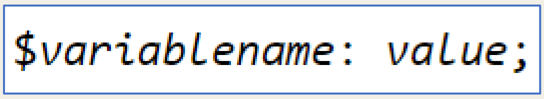
Sass Variables

Variables are a way to store information that you can re-use later.

With Sass, you can store information in variables, like:

- strings
- numbers
- colors
- booleans
- lists
- nulls

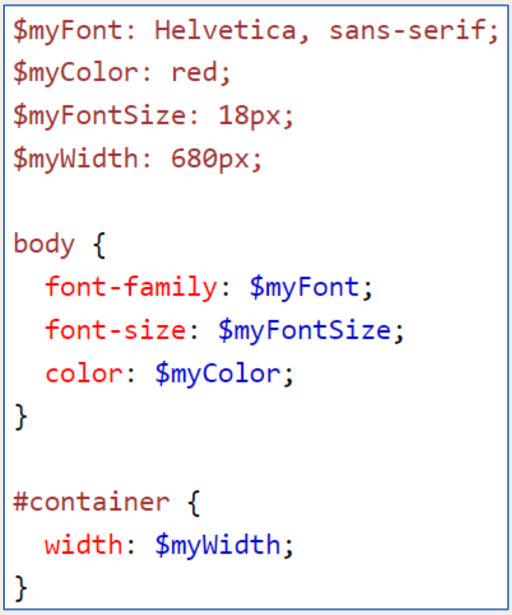
Sass uses the $ symbol, followed by a name, to declare variables:



```
$variablename: value;
```

Figure 118 *– SASS variable (**Source**: https://www.w3schools.com/sass/sass_variables.php)*

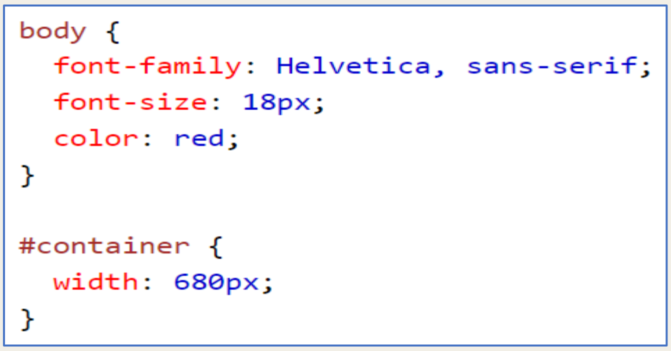
The following example declares 4 variables named myFont, myColor, myFontSize, and myWidth. After the variables are declared, you can use the variables wherever you want:



```
$myFont: Helvetica, sans-serif;
$myColor: red;
$myFontSize: 18px;
$myWidth: 680px;

body {
  font-family: $myFont;
  font-size: $myFontSize;
  color: $myColor;
}

#container {
  width: $myWidth;
}
```

Figure 119 *– SASS variable (**Source**: https://www.w3schools.com/sass/sass_variables.php)*

So, when the Sass file is transpiled, it takes the variables (myFont, myColor, etc.) and outputs normal CSS with the variable values placed in the CSS, like this:



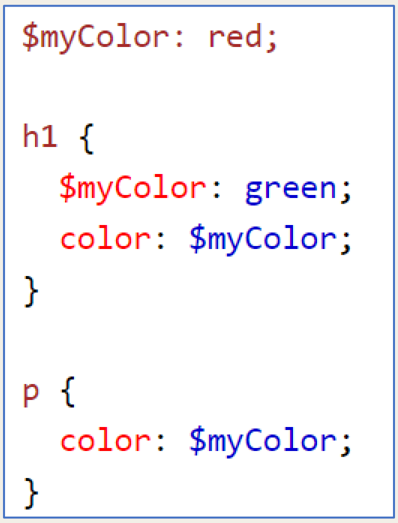
```
body {
  font-family: Helvetica, sans-serif;
  font-size: 18px;
  color: red;
}

#container {
  width: 680px;
}
```

Figure 120 *– SASS variable (**Source**: https://www.w3schools.com/sass/sass_variables.php)*

Sass Variable Scope

Sass variables are only available at the level of nesting where they are defined.

Look at the following example:



```
$myColor: red;

h1 {
  $myColor: green;
  color: $myColor;
}

p {
  color: $myColor;
}
```

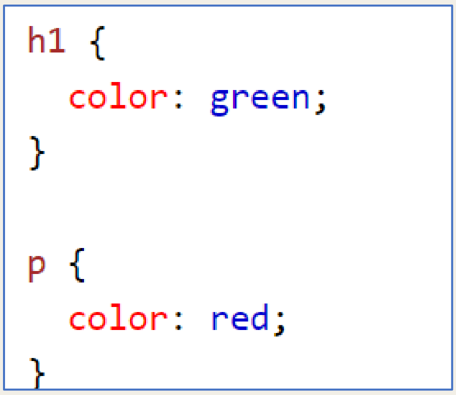
Figure 121 *– SASS variable (**Source**: https://www.w3schools.com/sass/sass_variables.php)*

Will the color of the text inside a <p> tag be red or green? It will be red!

The other definition, $myColor: green; is inside the <h1> rule and will only be available there!

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

So, the CSS output will be:



```
h1 {
  color: green;
}

p {
  color: red;
}
```

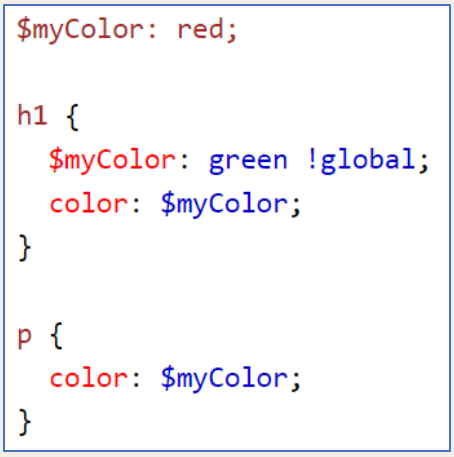
Figure 122 *– SASS variable (**Source**: https://www.w3schools.com/sass/sass_variables.php)*

Ok, that is the default behavior for variable scope.

Using Sass !global

The default behavior for variable scope can be overridden by using the !global switch. !global indicates that a variable is global, which means that it is accessible on all levels.

Look at the following example (same as above; but with !global added):

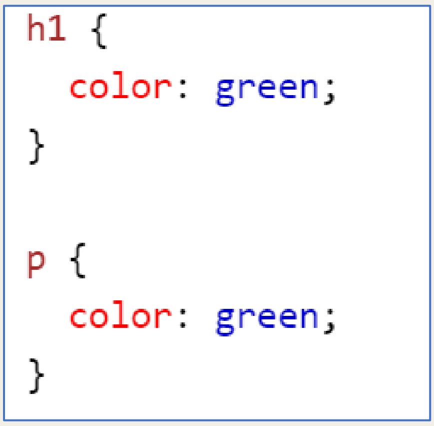


```
$myColor: red;

h1 {
  $myColor: green !global;
  color: $myColor;
}

p {
  color: $myColor;
}
```

Figure 123 *– SASS variable (**Source**: https://www.w3schools.com/sass/sass_variables.php)*

Now the color of the text inside a <p> tag will be green!

So, the CSS output will be:



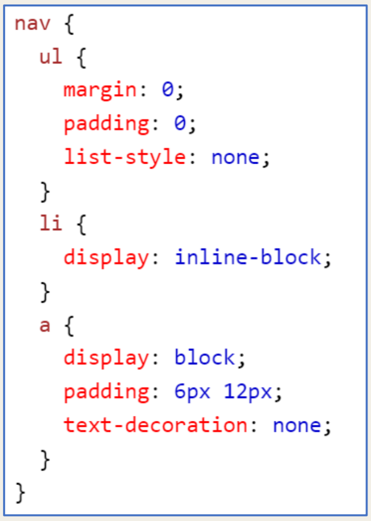
```
h1 {
  color: green;
}

p {
  color: green;
}
```

Figure 124 *– SASS variable (**Source**: https://www.w3schools.com/sass/sass_variables.php)*

Sass Nested Rules

Sass lets you nest CSS selectors in the same way as HTML.

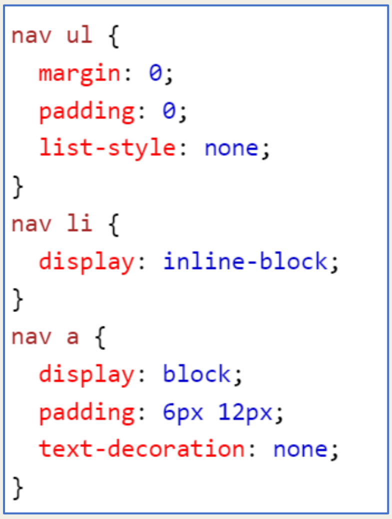
Look at an example of some Sass code for a site's navigation:



```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }
  li {
    display: inline-block;
  }
  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

Figure 125 *– SASS nested (**Source**: https://www.w3schools.com/sass/sass_nesting.php)*

Notice that in Sass, the ul, li, and selectors are nested inside the nav selector.
While in CSS, the rules are defined one by one (not nested):



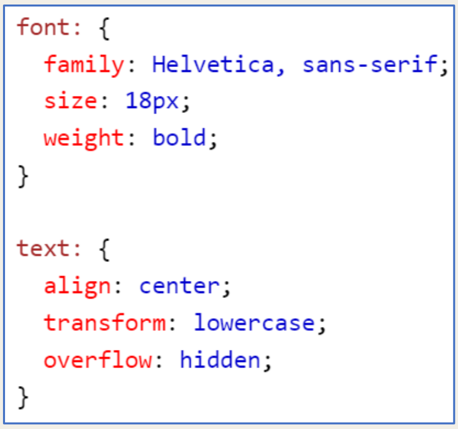
```
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

***Figure 126** – SASS nested (**Source**: https://www.w3schools.com/sass/sass_nesting.php)*

Because you can nest properties in Sass, it is cleaner and easier to read than standard CSS.

Sass Nested Properties

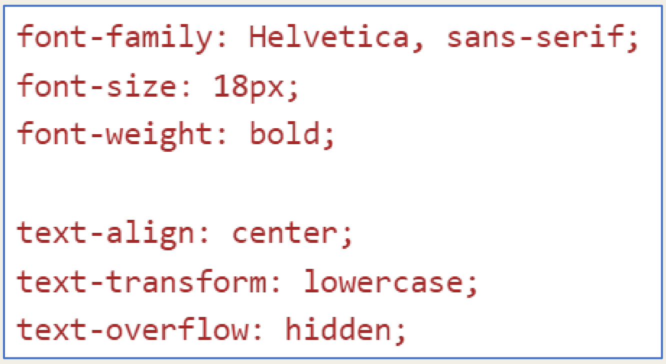
Many CSS properties have the same prefix, like font-family, font-size and font-weight or text-align, text-transform and text-overflow.
With Sass, you can write them as nested properties:



```
font: {
  family: Helvetica, sans-serif;
  size: 18px;
  weight: bold;
}

text: {
  align: center;
  transform: lowercase;
  overflow: hidden;
}
```

Figure 127 *– SASS nested (**Source**: https://www.w3schools.com/sass/sass_nesting.php)*

The Sass transpiler will convert the above to normal CSS:



```
font-family: Helvetica, sans-serif;
font-size: 18px;
font-weight: bold;

text-align: center;
text-transform: lowercase;
text-overflow: hidden;
```

Figure 128 *– SASS nested (**Source**: https://www.w3schools.com/sass/sass_nesting.php)*

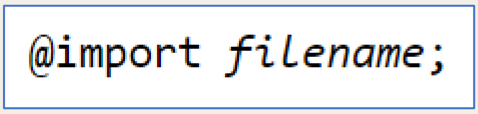
Sass @import and Partials

Sass keeps the CSS code DRY (Don't Repeat Yourself). One way to write DRY code is to keep related code in separate files.

You can create small files with CSS snippets to include in other Sass files. Examples of such files can be: reset file, variables, colors, fonts, font-sizes, etc.

Sass Importing Files

Just like CSS, Sass also supports the @import directive.
The @import directive allows you to include the content of one file in another.
The CSS @import directive has a major drawback due to performance issues; it creates an extra HTTP request each time you call it. However, the Sass @import directive includes the file in the CSS; so no extra HTTP call is required at runtime!

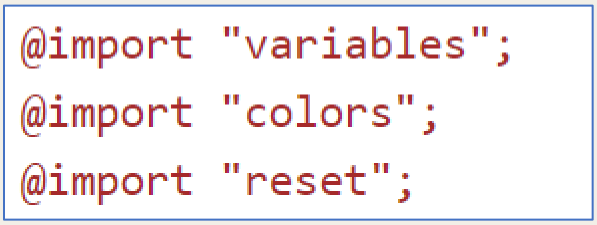


```
@import filename;
```

Figure 129 *– SASS import (**Source**: https://www.w3schools.com/sass/sass_import.php)*

Tip: You do not need to specify a file extension, Sass automatically assumes that you mean a .sass or .scss file. You can also import CSS files. The @import directive imports the file and any variables or mixins defined in the imported file can then be used in the main file.

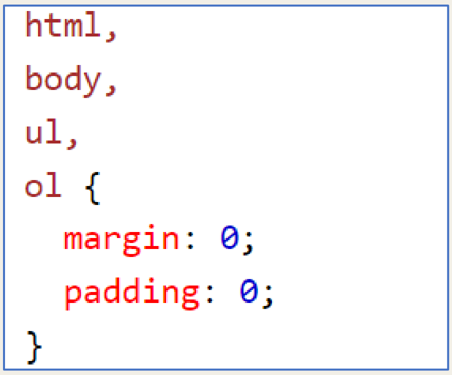
You can import as many files as you need in the main file:



```
@import "variables";
@import "colors";
@import "reset";
```

Figure 130 *– SASS import (**Source**: https://www.w3schools.com/sass/sass_import.php)*

Let's look at an example: Let's assume we have a reset file called "reset.scss", that looks like this:

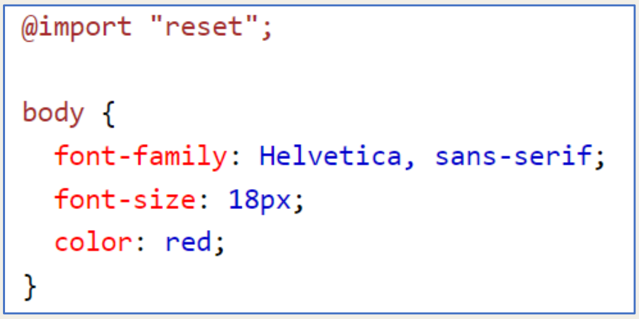


```
html,
body,
ul,
ol {
  margin: 0;
  padding: 0;
}
```

***Figure 131** – SASS import (**Source**: https://www.w3schools.com/sass/sass_import.php)*

And now we want to import the "reset.scss" file into another file called "standard.scss".

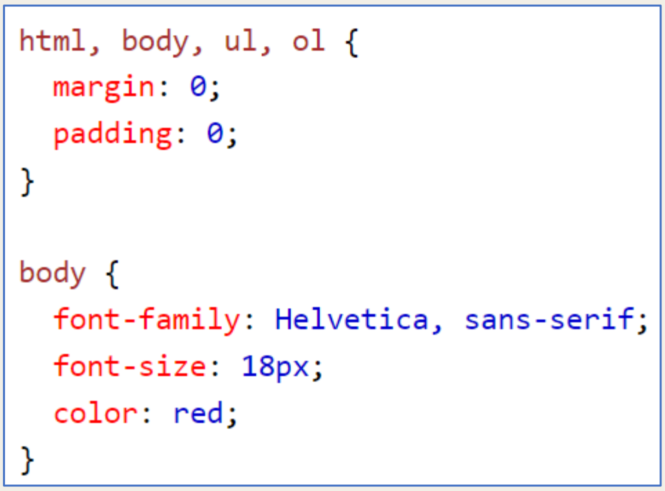
Here is how we do it: It is normal to add the @import directive at the top of a file; this way its content will have a global scope:



```
@import "reset";

body {
  font-family: Helvetica, sans-serif;
  font-size: 18px;
  color: red;
}
```

***Figure 132** – SASS import (**Source**: https://www.w3schools.com/sass/sass_import.php)*

So, when the "standard.css" file is transpiled, the CSS will look like this:



```
html, body, ul, ol {
  margin: 0;
  padding: 0;
}

body {
  font-family: Helvetica, sans-serif;
  font-size: 18px;
  color: red;
}
```

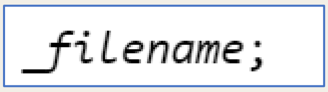
***Figure 133** – SASS import (**Source**: https://www.w3schools.com/sass/sass_import.php)*

Sass Partials

By default, Sass transpiles all the .scss files directly. However, when you want to import a file, you do not need the file to be transpiled directly.

Sass has a mechanism for this: If you start the filename with an underscore, Sass will not transpile it. Files named this way are called partials in Sass.

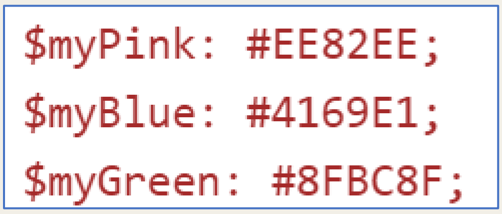
So, a partial Sass file is named with a leading underscore:



```
_filename;
```

***Figure 134** – SASS import (**Source**: https://www.w3schools.com/sass/sass_import.php)*

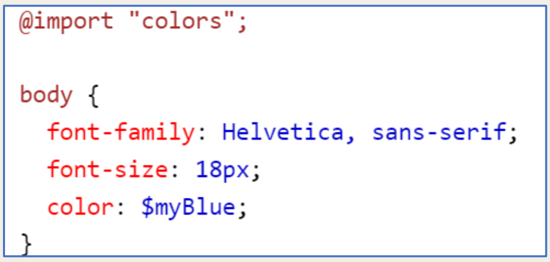
The following example shows a partial Sass file named "_colors.scss". (This file will not be transpiled directly to "colors.css"):



```
$myPink: #EE82EE;
$myBlue: #4169E1;
$myGreen: #8FBC8F;
```

Figure 135 *– SASS import (**Source**: https://www.w3schools.com/sass/sass_import.php)*

Now, if you import the partial file, omit the underscore. Sass understands that it should import the file "_colors.scss":



```
@import "colors";

body {
  font-family: Helvetica, sans-serif;
  font-size: 18px;
  color: $myBlue;
}
```

Figure 136 *– SASS import (**Source**: https://www.w3schools.com/sass/sass_import.php)*

Sass @mixin and @include

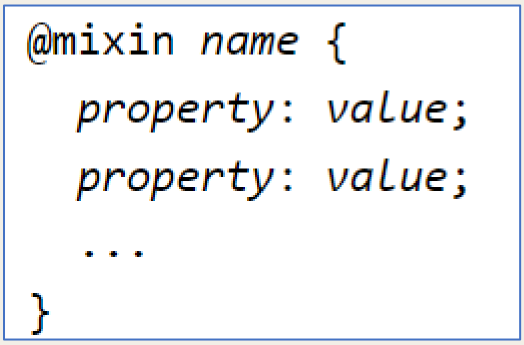
Sass Mixins

The @mixin directive lets you create CSS code that is to be reused throughout the website.

The @include directive is created to let you use (include) the mixin.

Defining a Mixin

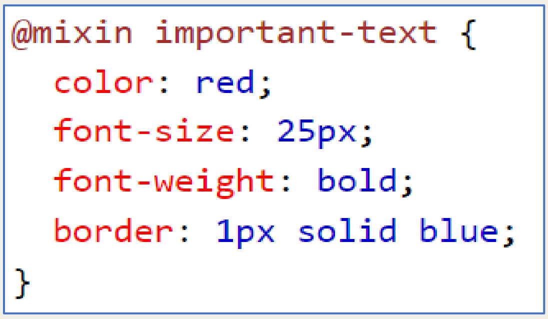
A mixin is defined with the @mixin directive.



```
@mixin name {
  property: value;
  property: value;
  ...
}
```

***Figure 137** – sass @mixin (**Source**: https://www.w3schools.com/sass/sass_mixin_include.php)*

The following example creates a mixin named "important-text":

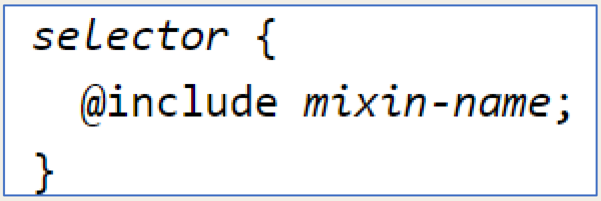


```
@mixin important-text {
  color: red;
  font-size: 25px;
  font-weight: bold;
  border: 1px solid blue;
}
```

***Figure 138** – sass @mixin (**Source**: https://www.w3schools.com/sass/sass_mixin_include.php)*

Using a Mixin

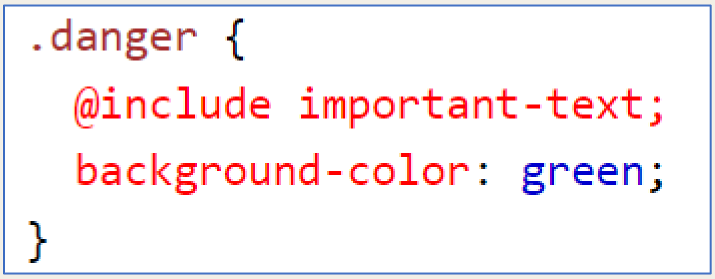
The @include directive is used to include a mixin.



```
selector {
  @include mixin-name;
}
```

***Figure 139** – sass @mixin (**Source**: https://www.w3schools.com/sass/sass_mixin_include.php)*

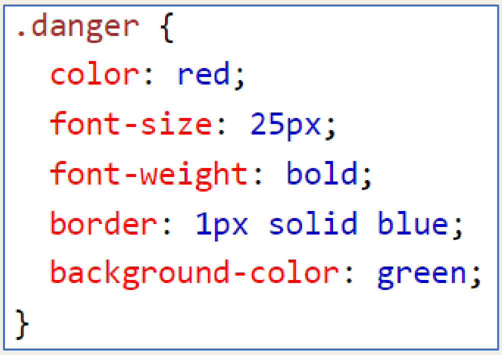
So, to include the important-text mixin created above:



```
.danger {
  @include important-text;
  background-color: green;
}
```

***Figure 140** – sass @mixin (**Source**: https://www.w3schools.com/sass/sass_mixin_include.php)*

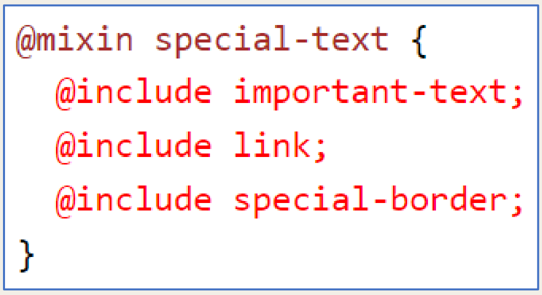
The Sass transpiler will convert the above to normal CSS:



```
.danger {
  color: red;
  font-size: 25px;
  font-weight: bold;
  border: 1px solid blue;
  background-color: green;
}
```

***Figure 141** – sass @mixin (**Source**: https://www.w3schools.com/sass/sass_mixin_include.php)*

A mixin can also include other mixins:



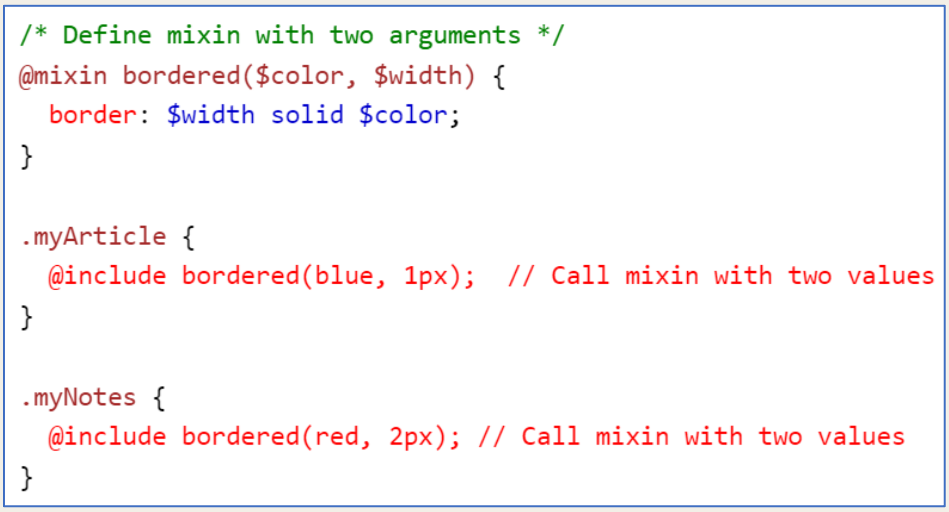
```
@mixin special-text {
  @include important-text;
  @include link;
  @include special-border;
}
```

***Figure 142** – sass @mixin (**Source**: https://www.w3schools.com/sass/sass_mixin_include.php)*

Passing Variables to a Mixin

Mixins accept arguments. This way you can pass variables to a mixin.

Here is how to define a mixin with arguments:

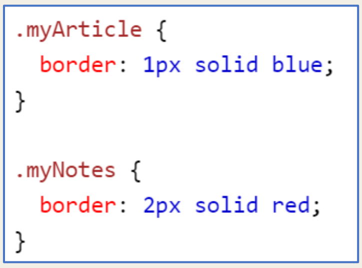


```
/* Define mixin with two arguments */
@mixin bordered($color, $width) {
  border: $width solid $color;
}

.myArticle {
  @include bordered(blue, 1px);  // Call mixin with two values
}

.myNotes {
  @include bordered(red, 2px); // Call mixin with two values
}
```

Figure 143 *– sass @mixin (**Source**: https://www.w3schools.com/sass/sass_mixin_include.php)*

Notice that the arguments are set as variables and then used as the values (color and width) of the border property.

After compilation, the CSS will look like this:



```
.myArticle {
  border: 1px solid blue;
}

.myNotes {
  border: 2px solid red;
}
```

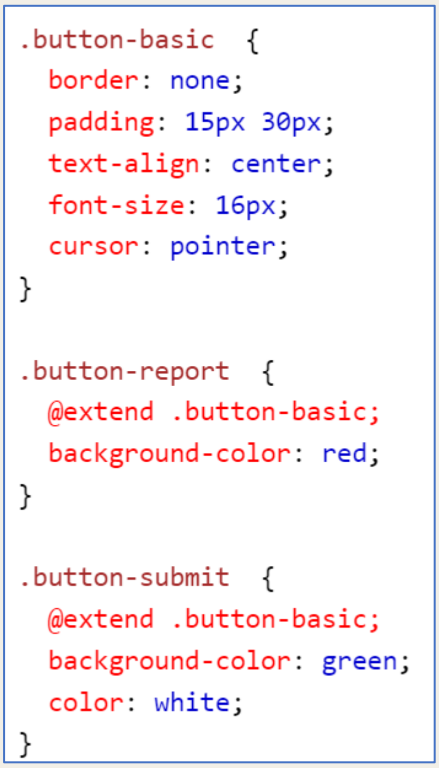
Figure 144 *– sass @mixin (**Source**: https://www.w3schools.com/sass/sass_mixin_include.php)*

Sass @extend Directive

The @extend directive lets you share a set of CSS properties from one selector to another.

The @extend directive is useful if you have almost identically styled elements that only differ in some small details.

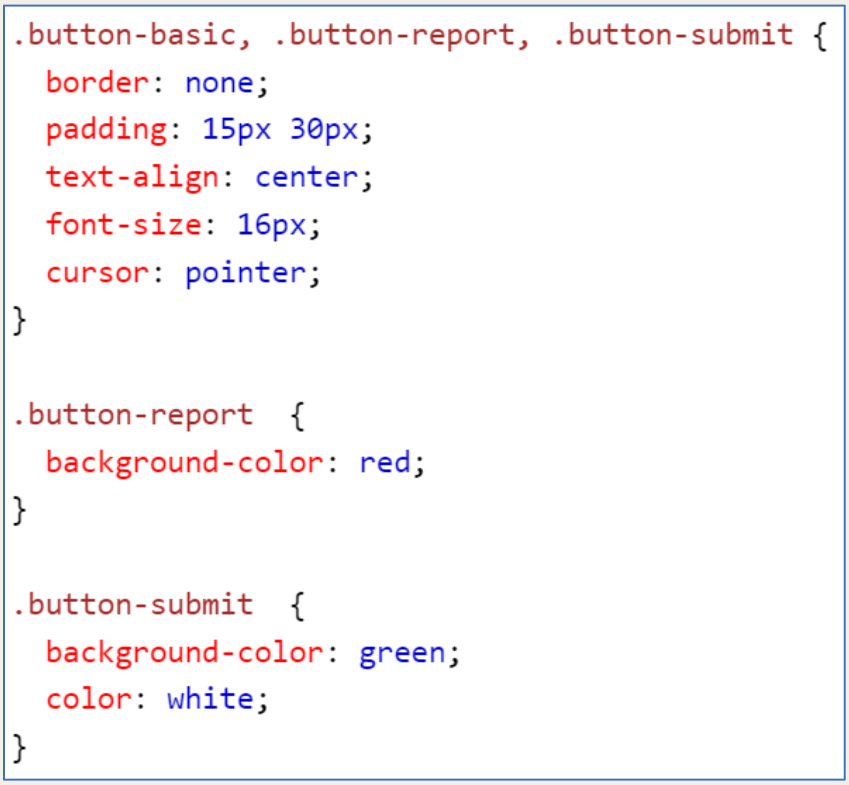
The following Sass example first creates a basic style for buttons (this style will be used for most buttons). Then, we create one style for a "Report" button and one style for a "Submit" button. Both "Report" and "Submit" button inherit all the CSS properties from the .button-basic class, through the @extend directive. In addition, they have their own colors defined:



```
.button-basic  {
  border: none;
  padding: 15px 30px;
  text-align: center;
  font-size: 16px;
  cursor: pointer;
}

.button-report  {
  @extend .button-basic;
  background-color: red;
}

.button-submit  {
  @extend .button-basic;
  background-color: green;
  color: white;
}
```

***Figure 145** – SASS Extend (**Source**: https://www.w3schools.com/sass/sass_extend.php)*

After compilation, the CSS will look like this:



```
.button-basic, .button-report, .button-submit {
  border: none;
  padding: 15px 30px;
  text-align: center;
  font-size: 16px;
  cursor: pointer;
}

.button-report  {
  background-color: red;
}

.button-submit  {
  background-color: green;
  color: white;
}
```

***Figure 146** – SASS Extend (**Source**: https://www.w3schools.com/sass/sass_extend.php)*

By using the @extend directive, you do not need to specify several classes for an element in your HTML code, like this: <button class="button-basic button-report">Report this</button>. You just need to specify .button-report to get both sets of styles.

The @extend directive helps keep your Sass code very DRY.

4. SQL - Structured Query Language

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Topic Information

Topic:

4. SQL

Prerequisites:

Basic computer literacy, basic software installed, and basic knowledge of working with files.

Workload:

10 hours

Description:

In this topic, we cover the basics of SQL to get learners acquainted with the programming world and encourage them to gain more expertise on SQL. We explain the attributes, syntax, and other relevant terms that learners may have heard or be familiar with and how these fit into the programming language. We provide a detailed account of the logic and syntax used in SQL, its structure and other basic and essential functions.

Learning outcomes:

- Recognise the concept and usage of Structured Query Language (SQL) in Relational Database Management Systems (RDBMS)
- Formulate basic syntax of SQL queries
- Use SQL to access, manage, and manipulate RDBMS

Material required:

- Computer or laptop

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- Internet connection
- Online SQL compiler (https://www.mycompiler.io/new/sql)
- Online text editor (https://www.w3schools.com/sql/default.asp)

Lesson Scenario:

The total time dedicated to this topic is 10 hours, and it is up to the judgement of the trainer/coach to decide how much time will be spent on each subtopic. We suggest using the PPT training presentations explicitly created for this topic to ease the teaching process and increase time efficiency. These presentations encompass the following:

- Progressive development of subtopics and core concepts to retain, and
- Recommended Exercises.

Depending on the trainer's/coach's preferences, the progressive development of the presentations allows the completion of the SQL session within the stipulated time, i.e., 10 hours. The presentations can also be made available to learners for self-studying.

Subtopics:

- 4.1. SQL Basics
- 4.2. SQL Databases
- 4.3. SQL References
- 4.4. SQL Examples

Additional resources:

- W3Schools - Guide for every SQL keyword and function, and examples for each of them
- Tutorialspoint – Another detailed guide on SQL keywords and functions, and various examples for each them

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

4.1. SQL Basics

What is SQL?

SQL stands for Structured Query Language. SQL is a standard programming language specifically designed for storing, retrieving, managing or manipulating data found within a Relational Database Management System (RDBMS). A relational database is a collection of data items with pre-defined relationships between them. These items are organised as a set of tables with columns and rows.

SQL became an ISO standard in 1987.

SQL is the most widely-implemented database language supported by popular relational database systems, like MySQL, SQL Server, and Oracle. SQL was initially developed at IBM in the early 1970s. Originally, it was called SEQUEL (Structured English Query Language) and it was later changed to SQL (pronounced as S-Q-L).

Applications of SQL

SQL is one of the most widely used query languages for databases. Some of its many applications are:

- Allowing users to access data in the relational database management systems,
- Allowing users to describe the data,
- Allowing users to define the data in a database and manipulate that data.

SQL Syntax

SQL statements are straightforward, like plain English, but with a specific syntax.

An SQL statement is composed of a sequence of keywords, identifiers etc., terminated by a semicolon (;).

Example:

```
SELECT emp_name, hire_date, salary FROM employees WHERE salary > 5000;
```

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

For better readability, you can also write the same statement, as follows:

```
SELECT emp_name, hire_date, salary
FROM employees
WHERE salary > 5000;
```

Use semicolon at the end of an SQL statement — it terminates the statement or submits the information to the database server.

Case Sensitivity in SQL

Consider another SQL statement that retrieves records from the Employees table:

```
SELECT emp_name, hire_date, salary FROM employees;
```

The same statement can also be written as follows:

```
select emp_name, hire_date, salary from employees;
```

SQL keywords are case-insensitive, which means SELECT is the same as select.

However, the database and table names may be case-sensitive depending on the operating system. In general, Unix or Linux platforms are case-sensitive, whereas Windows platforms aren't.

SQL Select

The SELECT statement selects or retrieves data from one or more tables. You can use this statement to retrieve all the rows from a table in one go or retrieve only those rows that satisfy a specific condition or a combination of conditions.

Suppose we have a table named Employees in our database that contains the following records:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
| emp_id| emp_name| hire_date| salary| dept_id|
+--------+--------------+------------+--------+---------+
|      1 | Ethan Hunt   | 2001-05-01 |   5000 |       4 |
|      2 | Tony Montana | 2002-07-15 |   6500 |       1 |
|      3 | Sarah Connor | 2005-10-18 |   8000 |       5 |
|      4 | Rick Deckard | 2007-01-03 |   7200 |       3 |
|      5 | Martin Blank | 2008-06-24 |   5600 |    NULL |
+--------+--------------+------------+--------+---------+
```

Select All from Table
The following statement will return all the rows from the employees' table.

```
>> SELECT * FROM employees;
```

Select Specific Columns from Table
If you don't require all the data, you can select specific columns, like this:

```
SELECT emp_id, emp_name, hire_date, salary
FROM employees;
```

After executing the above statement, you will get an output like this:

```
| emp_id| emp_name| hire_date| salary|
+--------+--------------+------------+--------+

|      1 | Ethan Hunt    | 1995-10-30 |   5000 |

|      2 | Tony Montana | 1990-07-15 |   6500 |

|      3 | Sarah Connor | 2011-04-13 |   5600 |

|      4 | Rick Deckard | 2005-10-18 |   7200 |

|      5 | Martin Blank | 1996-05-24 |   8000 |

+--------+--------------+------------+--------+
```

SQL Select Distinct

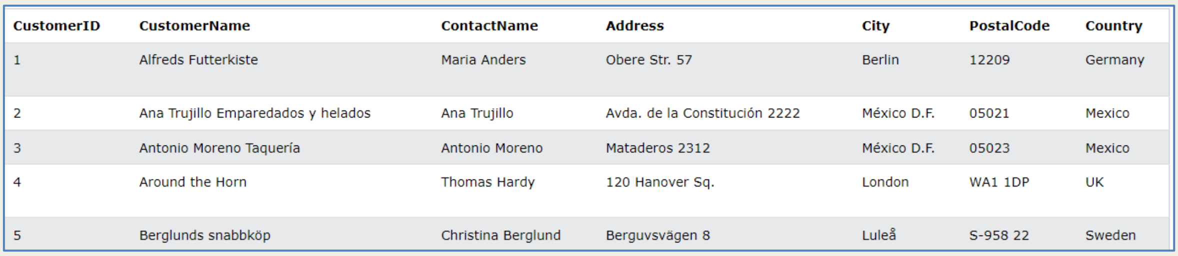
The SELECT DISTINCT statement omits duplicated values when used in a query.

You can find duplicated values inside a table, but sometimes you want to see the "unique" values.

Syntax:

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

In the following examples, we will be using the Customers table that contains data about our customers.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Table 1 - *Customers table in SELECT DISTINCT example* ***(Source****:* https://www.w3schools.com/sql/sql_distinct.asp*)*

To *select all values* from the Country column in the Customers table, you would use the following statement:

```
SELECT Country FROM Customers;
```

However, this statement will include duplicate values.

If you want to omit the duplicate values from your query, use SELECT DISTINCT:

```
SELECT DISTINCT Country FROM Customers;
```

Suppose you wanted to list the number of different customer countries; you would use the following statement:

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

Please note that this example will not work in Firefox since COUNT(DISTINCT column_name) is not supported in MS Access.

To get the equivalent result in Access, use this:

```
>> SELECT Count(*) AS DistinctCountries
    FROM (SELECT DISTINCT Country FROM Customers);
```

SQL Where

Previously, we have learnt how to fetch all the records from a table or columns from a table.

However, in real-world cases, we generally need to select, update or delete only those records which fulfil certain conditions, like users who belong to a particular age group, country, etc.

The WHERE clause is used with the SELECT, UPDATE, and DELETE.

The WHERE clause is used with the SELECT statement to extract only those records that fulfil specified conditions.

The basic syntax is as follows:

```
SELECT column_list
FROM table_name
WHERE condition;
```

Now, let's check out some examples that demonstrate how it works.

Suppose we have a table called Employees in our database with the following records:

```
| emp_id| emp_name| hire_date| salary| dept_id|
+--------+--------------+------------+--------+---------+
|      1 | Ethan Hunt   | 2001-05-01 |   5000 |       4 |
|      2 | Tony Montana | 2002-07-15 |   6500 |       1 |
|      3 | Sarah Connor | 2005-10-18 |   8000 |       5 |
|      4 | Rick Deckard | 2007-01-03 |   7200 |       3 |
+--------+--------------+------------+--------+---------+
```

The following SQL statement will return all employees from the Employees' table *whose salary is greater than 7000*:

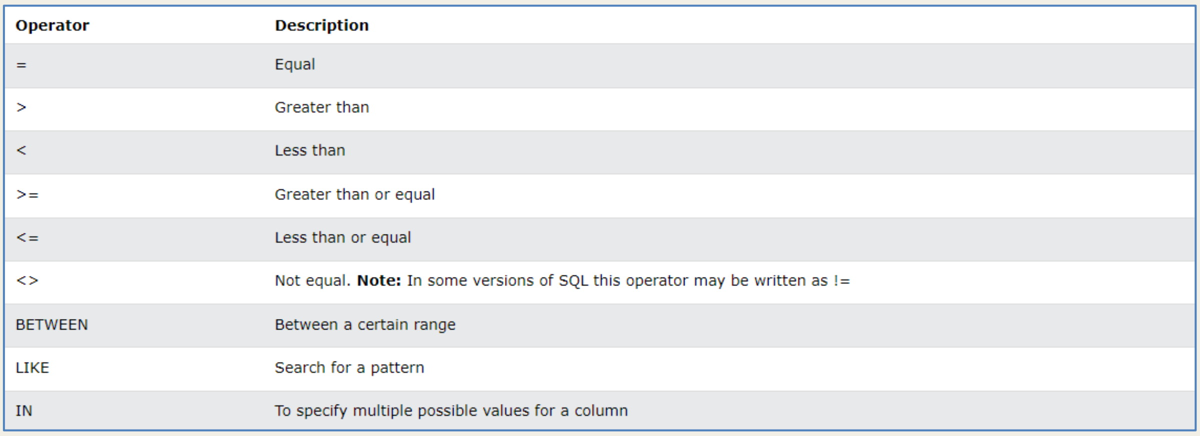
```
SELECT * FROM employees WHERE salary > 7000;
```

The WHERE clause simply filters out the unwanted data.

Another example would be to *select all* employees with *department id =1*:

```
SELECT * FROM employees WHERE dept_id=1;
```

The following table provides the list of operators that can be used with the WHERE clause:



Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Table 2 *-Table of Operators used in WHERE clause (**Source**: https://www.w3schools.com/sql/sql_where.asp)*

SQL And, Or, Not

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition.

The AND operator displays a record if all the conditions that use AND are TRUE.

AND Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Example: Select all fields from Customers table where country is “Germany” AND City is “Berlin”.

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

The OR operator displays a record if any of the conditions that use OR are TRUE.

OR Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Example 1: Select all fields from Customers table where city is “Berlin” or “München”

```
SELECT * FROM Customers.
WHERE City='Berlin' OR City='München';
```

Example 2: Select all fields from Customers table where country is “Germany” or “Spain”.

```
SELECT * FROM Customers
WHERE Country='Germany' OR Country='Spain';
```

The NOT operator displays a record if the condition(s) is NOT TRUE.

NOT Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Example: Select all fields from Customers table where country is NOT “Germany”.

```
SELECT * FROM Customers
WHERE NOT Country='Germany';
```

Combining AND, OR and NOT

Example 1: Select all rows from the Customers table where country is Germany and city must be either Berlin or München.

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

Example 2: Select all rows from the Customers table where country is NOT Germany and NOT USA.

```
SELECT * FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

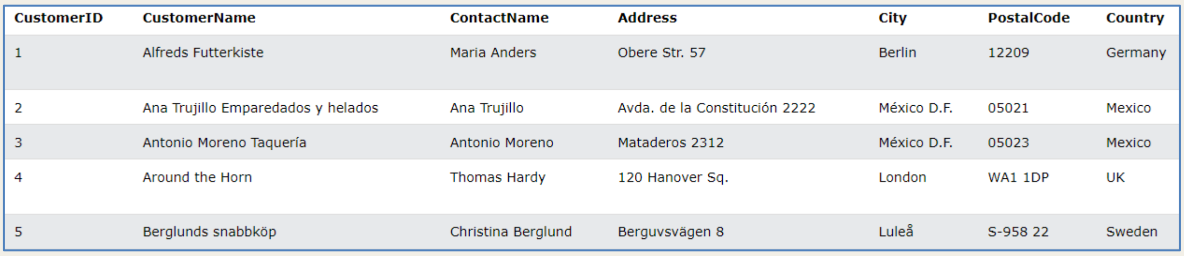
SQL Order By

The ORDER BY keyword sorts the result-set in ascending or descending order. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

In the following examples, we will use the Customers table shown below:



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

***Table 3 -** Customers Table in ORDER BY example (**Source:** https://www.w3schools.com/sql/sql_orderby.asp)*

Example 1: Selects all customers from the Customers table and sorts them by the Country column

```
SELECT * FROM Customers
ORDER BY Country;
```

Example 2: Selects all customers from the same table and sorts them in descending order by the Country column

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

Example 3: Selects all customers from the same table and sorts them by Country and Customer Name.

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

Here, the order is initially sorted by Country. However, if there are some rows that have the same country, then they are sorted by Customer Name.

Example 4: Selects all customers from the same table and sorts them in ascending order by Country and descending order by Customer Name

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

SQL Insert Into

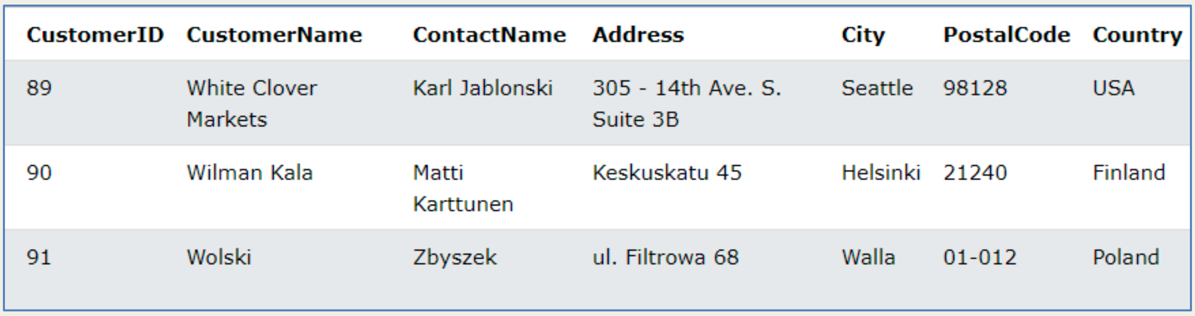
The INSERT INTO statement inserts new records in a table.

For your code to run correctly, specify both the column names and the values that will be inserted.

Syntax:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

***Table 4** - INSERT INTO Example **(Source:**: https://www.w3schools.com/sql/sql_insert.asp)*

For example, to add a new record in your “Customers” table, use the following:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

SQL Null Values

A field with a NULL value is a field with no value. If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value has been **left blank** during record creation!

Testing for NULL Values

It is impossible to test for NULL values with comparison operators, such as =, <, or <>.

Instead, we will have to use the **IS NULL** and **IS NOT NULL** operators.

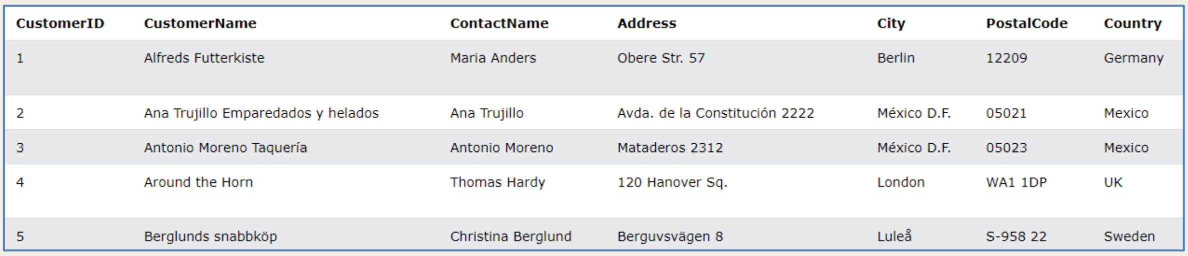
IS NULL Syntax:

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax:

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

The following examples use the table below:



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Table 5 *- Customers Table in NULL Values Example (**Source:*** https://www.w3schools.com/sql/sql_null_values.asp)

Example of IS NULL

Selects all customers with Null values (i.e. empty values) in the Address column:

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

To look for Null values, always use IS NULL.

Example of IS NULL

Selects all customers with NOT Null values (i.e. non-empty values) in the Address column:

```
SELECT CustomerName, ContactName, Address
FROM Customers
```

```
WHERE Address IS NOT NULL;
```

SQL Update

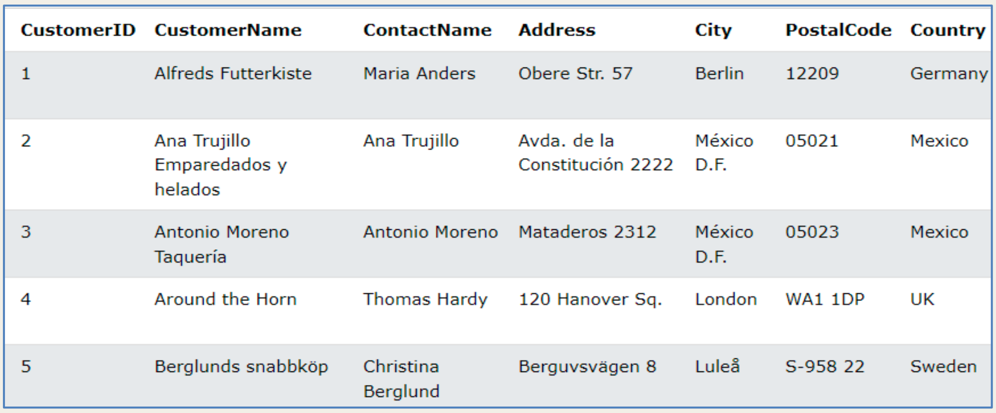
The UPDATE statement modifies the existing records of a table.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) should be updated.

If you **omit** the WHERE clause, all records in the table will be updated!

Example



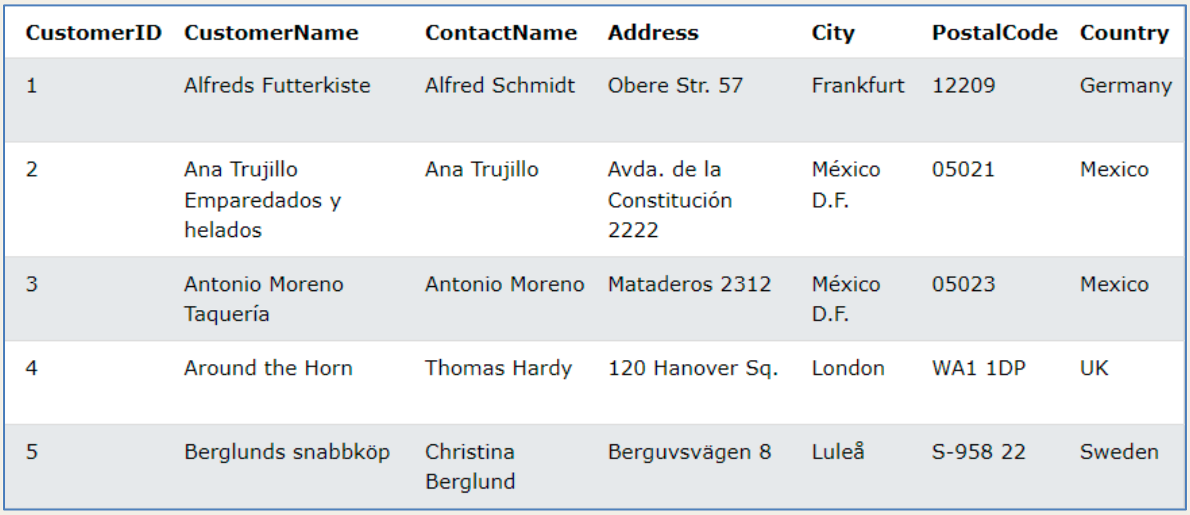
CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

***Table 6 -** Customers table in UPDATE example **(Source:** https://www.w3schools.com/sql/sql_update.asp)*

To update CustomerID=1 from the “Customers” table in the sample database, use the following:

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

And the “Customers” table will now look like this:



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

***Table 7 -** Customers table in UPDATE example (**Source:** https://www.w3schools.com/sql/sql_update.asp)*

The WHERE clause determines how many records will be updated.

The following SQL statement will update the ContactName to "Juan" for all records where the country is "Mexico" in the Customers table:

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

SQL Delete

The DELETE statement deletes existing records in a table.

DELETE FROM table_name WHERE condition;

Keep in mind that you need to be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted.

If you **omit** the WHERE clause, all records in the table will be deleted!

Example:

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Delete All Records

It is possible to delete all rows in a table without deleting the table itself. This means that the table’s structure, attributes, and indexes will stay intact:

```
DELETE FROM table_name;
```

SQL Select Top

The SELECT TOP clause is used to specify the number of records that will be returned.

The SELECT TOP clause is useful on large tables with thousands of records. However, returning a large number of records can impact performance.

Note that not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST n ROWS ONLY and ROWNUM.

SQL Server/MS Access Syntax:

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

MySQL Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
```

```
LIMIT number;
```

Oracle 12 Syntax:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s)
FETCH FIRST number ROWS ONLY;
```

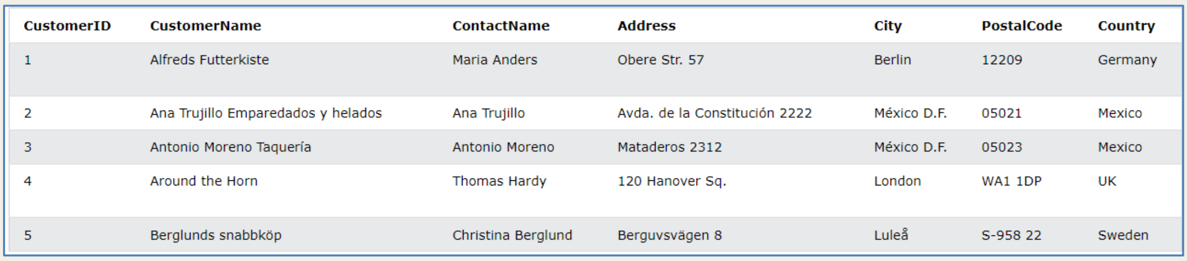
Older Oracle Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE ROWNUM <= number;
```

Older Oracle Syntax with ORDER BY:

```
SELECT *
FROM (SELECT column_name(s) FROM table_name ORDER BY column_name(s))
WHERE ROWNUM <= number;
```

We will use the “Customers” table shown below in the following examples.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Table 8 - *Customers table in SELECT TOP examples* ***(Source:*** *https://www.w3schools.com/sql/sql_top.asp)*

Examples of TOP, LIMIT and FETCH FIRST

To select the first three records from the Customers table in SQL Server/MS Access, use the following:

```
SELECT TOP 3 * FROM Customers;
```

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

To execute a query with the same result as above in MySQL, use the following statement:

```
SELECT * FROM Customers
LIMIT 3;
```

To execute a query with the same result as the two above in Oracle:

```
SELECT * FROM Customers
FETCH FIRST 3 ROWS ONLY;
```

Examples of TOP PERCENT

To select the first 50% of records found in the Customers table, execute the following statement in SQL Server/MS Access:

```
SELECT TOP 50 PERCENT * FROM Customers;
```

Its equivalent in Oracle is the following:

```
SELECT * FROM Customers
FETCH FIRST 50 PERCENT ROWS ONLY;
```

Examples of ADD a WHERE Clause

In the following example, we are selecting the first three records from the Customers table, where the Country is “Germany” for SQL Server/MS Access:

```
>> SELECT TOP 3 * FROM Customers
    WHERE Country='Germany';
```

Its equivalent in MySQL:

```
SELECT * FROM Customers
WHERE Country='Germany'
LIMIT 3;
```

Its equivalent in Oracle:

```
SELECT * FROM Customers
WHERE Country='Germany'
FETCH FIRST 3 ROWS ONLY;
```

SQL Min and Max

The MIN() function returns the smallest values from selected columns.

MIN() Syntax

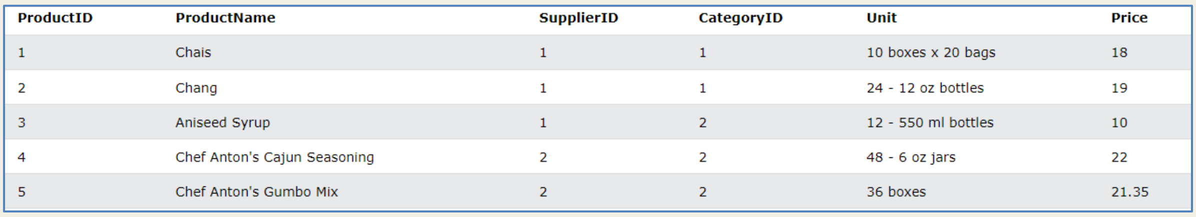
```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

The MAX() function returns the largest value from selected columns.

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

In the following examples, we will use the Products table shown below:



ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

***Table 9 -** Products Table in Min and Max Examples (**Source:** https://www.w3schools.com/sql/sql_min_max.asp)*

Example 1: Finds the price of the cheapest product

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

Example 2: Finds the price of the most expensive product

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

SQL Count, Avg, Sum

The COUNT() function returns the number of rows that match a specified criterion.

COUNT() Syntax

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

The AVG() function returns the average value of a numeric column.

AVG() Syntax

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

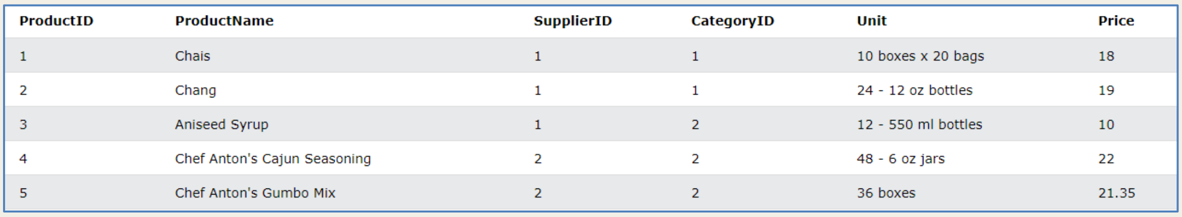
The SUM() function returns the total sum of a numeric column.

SUM() Syntax

```
SELECT SUM(column_name)
```

```
FROM table_name
WHERE condition;
```

We will use the Products table in the following examples below.



ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

Table 10 *- Products table in Count, Avg, Sum Examples (**Source:** https://www.w3schools.com/sql/sql_count_avg_sum.asp)*

Example of COUNT()

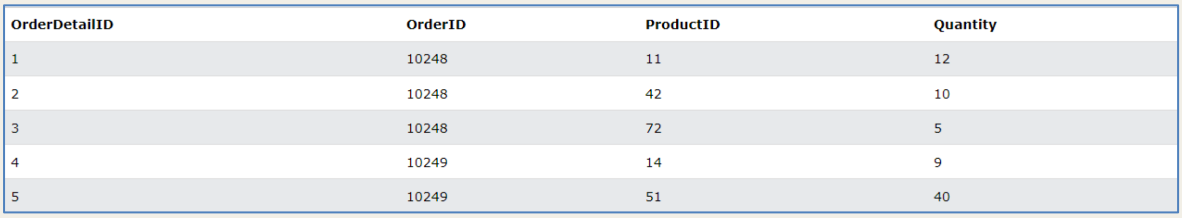
Execute a query to find the number of products:

```
SELECT COUNT(ProductID)
FROM Products;
```

Example of AVG()

Execute a query to find the average price of all products:

```
SELECT AVG(Price)
FROM Products;
```



OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

Table 11 *- OrdersDetails table in Count, Avg, Sum Examples (**Source:** https://www.w3schools.com/sql/sql_count_avg_sum.asp)*

In the following example, we will use the OrdersDetails table, shown above, to find the sum of "Quantity":

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

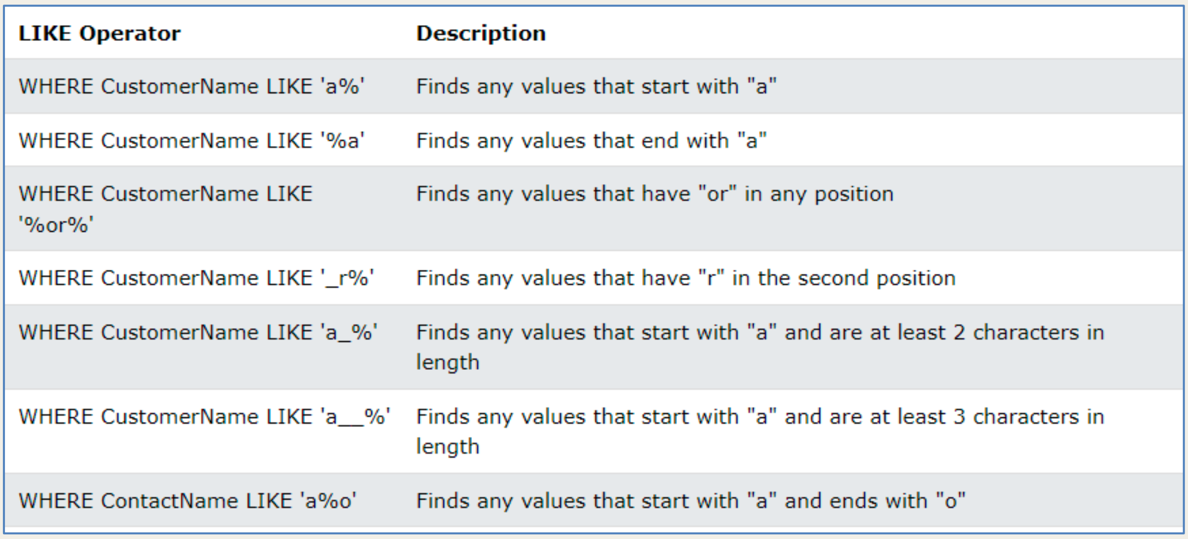
SQL Like

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

LIKE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE column LIKE pattern;
```

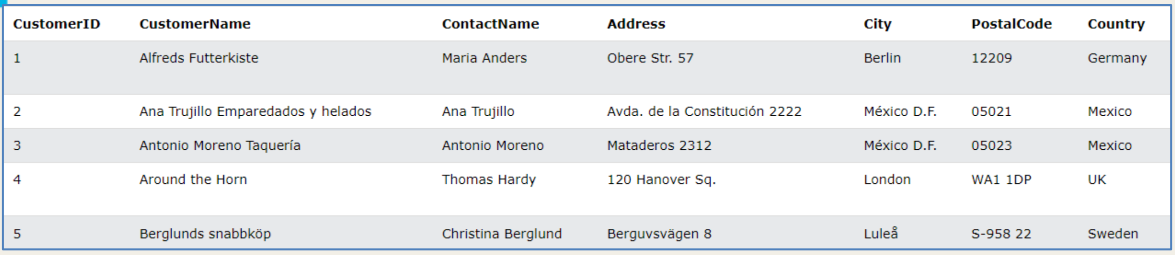
Here are some examples showing different LIKE operators with '%' and '_' wildcards:



LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Table 12 *- LIKE operators (**Source**: https://www.w3schools.com/sql/sql_like.asp)*

We will see some examples that will be using the Customers table shown below.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Table 13 *- Customers table in LIKE examples (**Source:** https://www.w3schools.com/sql/sql_like.asp)*

Example 1: Selects all customers with a Customer Name that starts with ‘a’.

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

Example 2: Selects all Customer names ending with ‘a’.

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

Example 3: Selects all Customer names that contain ‘or’ in their name in any position.

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

Example 4: Selects all Customer names that contain ‘r’ in the second position.

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

Example 5: Selects all Customer names that begin with ‘a’ and have at least three characters in length.

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

Example 6: Selects all Customer names that begin with ‘a’ and end with ‘o’.

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%o';
```

Example 7: Selects all Customer names that do not begin with ‘a’.

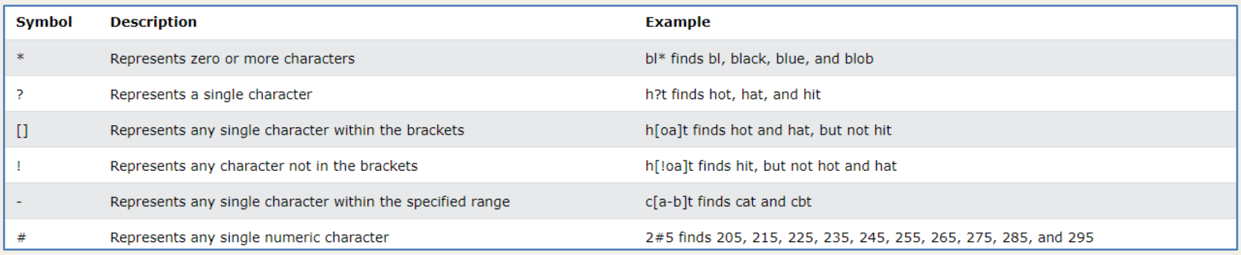
```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

SQL Wildcards

A wildcard character substitutes one or more characters in a string. It is used with the LIKE operator.

The LIKE operator is also used in a WHERE clause to search for a specified pattern in a column, as we have seen in the previous subsection.

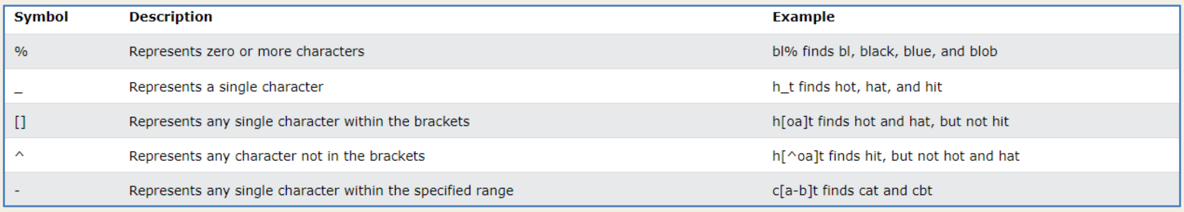
Wildcards in MS Access



Symbol	Description	Example
*	Represents zero or more characters	bl* finds bl, black, blue, and blob
?	Represents a single character	h?t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
!	Represents any character not in the brackets	h[!oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt
#	Represents any single numeric character	2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295

__Table 14__ – Wildcards in Access (__Source:__ https://www.w3schools.com/sql/sql_wildcards.asp)

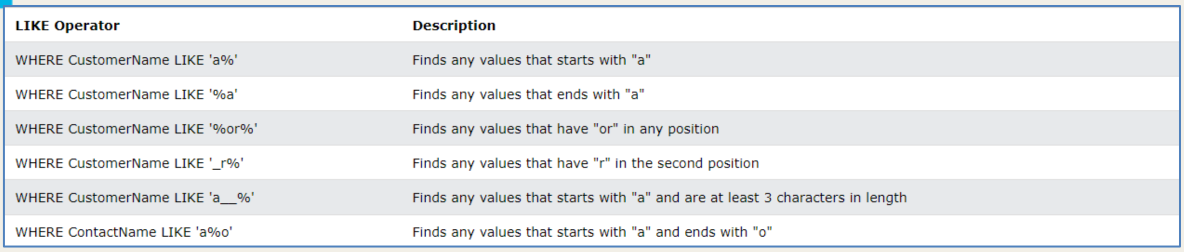
Wildcards in SQL Server



Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt

__Table 15__ – Wildcards in SQL Server (__Source__: https://www.w3schools.com/sql/sql_wildcards.asp)

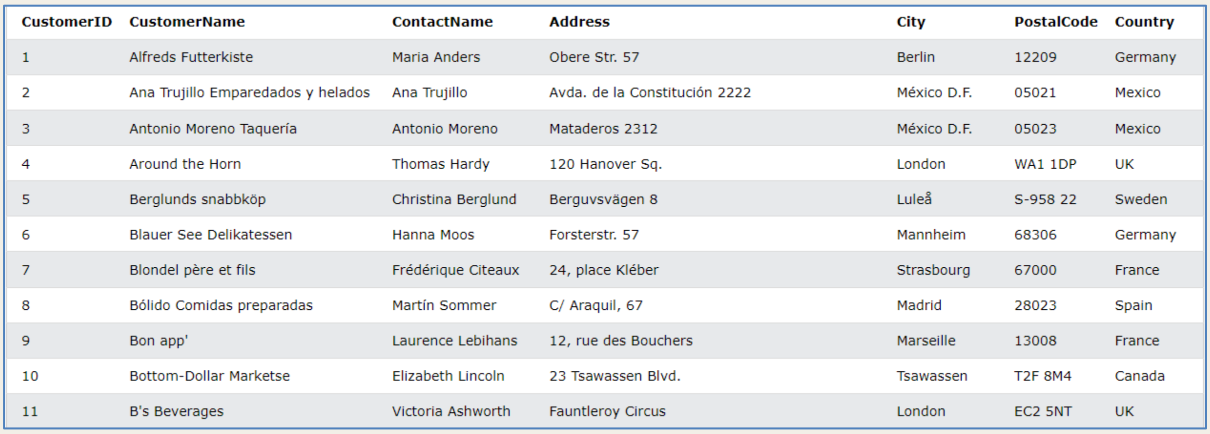
Wildcards can be used in combination; look at some examples at the table below:



LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a__%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

Table 16 *- Examples of Wildcards with % and '_' (**Source**: https://www.w3schools.com/sql/sql_wildcards.asp)*

Let's see some examples with the Customers table.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen	T2F 8M4	Canada
11	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 5NT	UK

Table 17 - *Customers table in WILDCARDS example (**Source**: https://www.w3schools.com/sql/sql_wildcards.asp)*

Examples with the % Wildcard

In this example, we are selecting all customers with a City starting with "ber":

```
SELECT * FROM Customers
WHERE City LIKE 'ber%';
```

In the following example, we are selecting all customers with a City containing "es":

```
SELECT * FROM Customers
WHERE City LIKE '%es%';
```

Examples with the _ Wildcard

Here, we are selecting all customers with a City starting with any character followed by "ondon":

```
SELECT * FROM Customers
```

```
WHERE City LIKE '_ondon';
```

In this example, we are again selecting all customers with a City starting with ‘L’, followed by any character, followed by "n", followed by any character, followed by "on":

```
SELECT * FROM Customers
WHERE City LIKE 'L_n_on';
```

Examples with the [charlist] Wildcard

Here, we are selecting all customers with a City starting with ‘b’,’ s’, or ‘p’:

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

In the following example, we are selecting all customers with City starting with ‘a’, ‘b’, or ‘c’:

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

Examples with the [!charlist] Wildcard

The exclamation mark shows characters that do not contain a specified string. For example, we want to select all customers with a City that does not start with ‘b’, ‘s’, or ‘p’:

```
SELECT * FROM Customers
WHERE City LIKE '[!bsp]%';
```

As an alternative, we can use the following:

```
SELECT * FROM Customers
WHERE City NOT LIKE '[bsp]%';
```

SQL In

The IN operator is used to specify multiple values in a WHERE clause. It can be considered as satisfying various conditions.

Syntax 1:

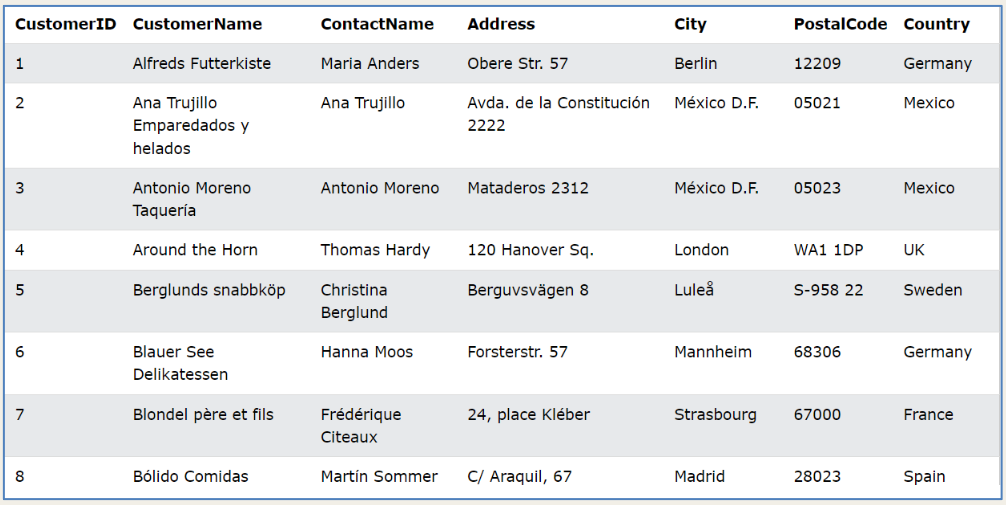
```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Syntax 2:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

There are two ways to use the IN operator, as you have seen.

Suppose that we have a table called “Customers” containing the following columns: CustomerID, CustomerName, ContactName, Address, City, PostalCode and Country.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain

***Table 18** - Customers Table in IN operator example (**Source:** https://www.w3schools.com/sql/sql_in.asp)*

As an example, we want to select all the customers that are located in Germany, France or UK:

```
SELECT * FROM Customers
WHERE Country IN (‘Germany’, ‘France’, ‘UK’)
```

Another example is selecting all the customers that are **not** located in Germany, France or the UK:

```
SELECT * FROM Customers
WHERE Country NOT IN ( ‘Germany’,  ‘France’, ‘UK’)
```

Let’s also consider a third example where we want to select customers that are from the same countries as the suppliers:

```
SELECT * FROM Customers
WHERE Country  IN (SELECT Country FROM Suppliers)
```

SQL Between

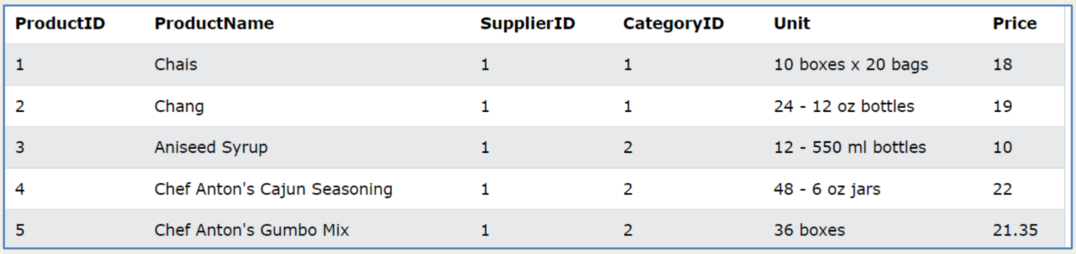
The BETWEEN operator provides a range of values to select from. The values can be text, numbers or dates. The BETWEEN operator includes the starting and end values.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Let’s say that we have the following table, which contains information on different products.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	1	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	1	2	36 boxes	21.35

Table 19 *- BETWEEN operator example (**Source**: https://www.w3schools.com/sql/sql_between.asp)*

There will be a series of examples with the following operators: BETWEEN, NOT BETWEEN, BETWEEN with IN, BETWEEN and NOT BETWEEN with text values and BETWEEN dates.

BETWEEN Example: Selects all products with a price range of 10 to 20.

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

NOT BETWEEN Example: Shows all the products outside the range that we set in the previous example.

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

BETWEEN with IN Example: Selects all products with a price range of 10 to 20 and does not show products with CategoryID 1, 2, or 3.

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

BETWEEN with text values Example: Selects all products with a ProductName between Carnarvon Tigers and Mozzarella di Giovanni.

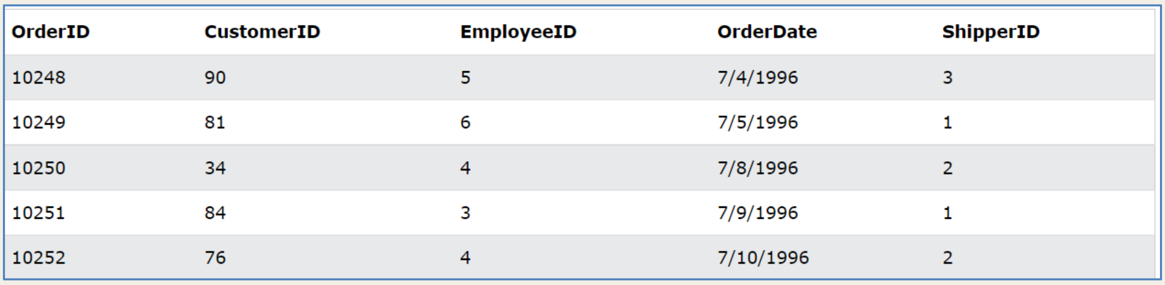
```
SELECT * FROM Products
```

```
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

NOT BETWEEN with text values Example: Selects all products with a ProductName NOT between Carnarvon Tigers and Mozzarella di Giovanni.

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

Assume that we have the following table that contains information on different Orders:



OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	7/4/1996	3
10249	81	6	7/5/1996	1
10250	34	4	7/8/1996	2
10251	84	3	7/9/1996	1
10252	76	4	7/10/1996	2

***Table 20** - BETWEEN operator example (**Source**: https://www.w3schools.com/sql/sql_between.asp)*

BETWEEN Dates Example: Selects all orders with an OrderDate between '01-July-1996' and '31-July-1996'.

There are two ways that this can be done, by either using a hashtag (#) or quote marks (‘’):

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;
```

OR

```
SELECT * FROM Orders
```

```
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

SQL Aliases

Aliases assign a temporary name to a table or a column within a table. An alias exists only for the duration of a query, and it is often used to make column names more readable. An alias is created by using the keyword **AS**.

Syntax for column alias:

```
SELECT column_name AS alias_name
FROM table_name;
```

Syntax for table alias:

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

Column Aliases

Let’s see an example that creates two aliases, one for each column:

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Another example creates two aliases again:

```
SELECT CustomerName AS Customer, ContactName AS [Contact Person]
FROM Customers;
```

Note that it is put in square brackets ([ ]) because the alias contains spaces. Quotation marks can be used as an alternative to square brackets.

You also have the option of creating an alias that contains one or more columns, let's look at the example below to see how it works:

```
SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' + Country AS
Address
FROM Customers;
```

The above statement changes a little bit in MySQL:

```
SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS
Address
FROM Customers;
```

Table Aliases

The following example selects all the orders from the customer table with CustomerID=4 (Around the Horn).

Here aliases are used to shorten the query:

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
```

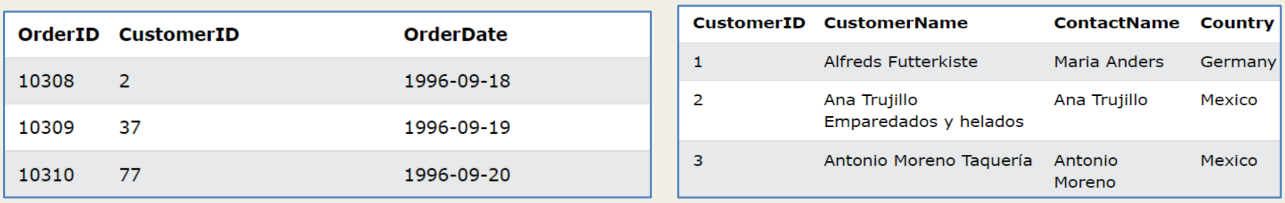
A query without aliases would look something like this:

```
SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName
FROM Customers, Orders
WHERE Customers.CustomerName='Around the Horn' AND
Customers.CustomerID=Orders.CustomerID;
```

SQL Joins

A JOIN clause combines rows from two or more tables based on a related column found in both tables.

Let's look at the Orders table and the Customers table:



OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

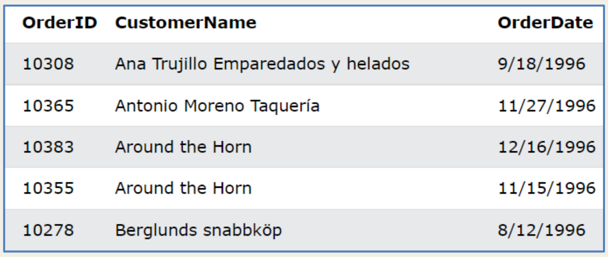
***Tables 21 & 22** - Orders and Customers Tables in JOIN example (**Source:** https://www.w3schools.com/sql/sql_join.asp)*

If you look at the two tables, you will notice a common column called the CustomerID. Based on the common column, we can create an SQL statement that uses an INNER JOIN, which selects records that have matching values in both tables.

Example:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

This command will create something like the following table:



OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

***Table 23** - JOIN example (**Source**: https://www.w3schools.com/sql/sql_join.asp)*

There are four different joins in SQL:

1. **(INNER) JOIN:** Returns records that have matching values in both tables;
2. **LEFT (OUTER) JOIN:** Returns all records from the left table and the corresponding matched records from the right table;
3. **RIGHT (OUTER) JOIN:** Returns all records from the right table and the corresponding matched records from the left table;
4. **FULL (OUTER) JOIN:** Returns all records when there is a match in either the left or the right table.

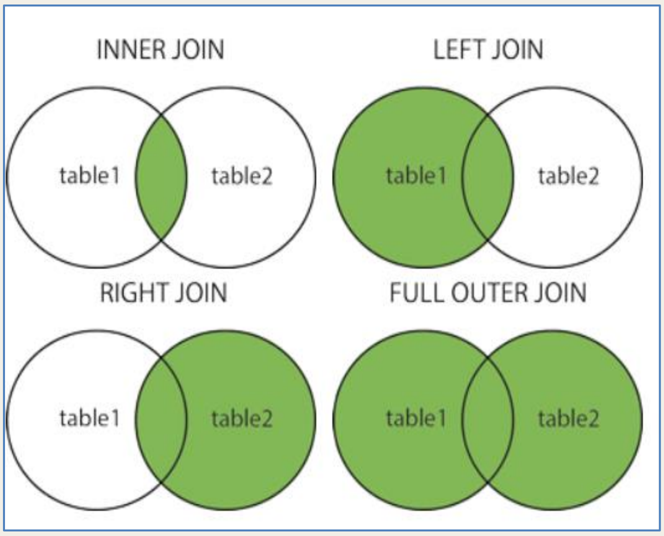


Figure 1 *- Different types of JOINS (**Source**: https://www.w3schools.com/sql/sql_join.asp)*

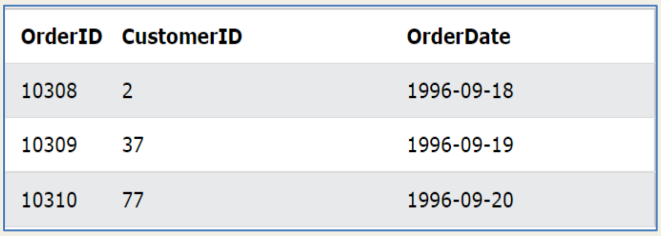
SQL Inner Join

The INNER JOIN keyword selects records that have matching values in both tables.

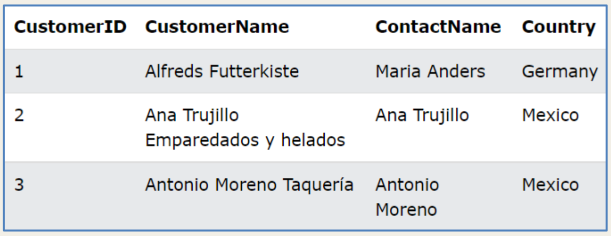
Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

The same tables as the example in the previous subsection are used to perform an inner join.



OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20



CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Tables 24 & 25 *- Orders and Customers Tables in JOIN example (**Source**: https://www.w3schools.com/sql/sql_join_inner.asp)*

In this example, we want to retrieve the names of customers and their corresponding Order IDs:

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Note that the INNER JOIN keyword will select all rows from both tables that match. If records in the Orders table do not have matches in the Customers table, they will not be selected.

In the following example, we will see how to join three tables that contain customer and shipper information:

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

SQL Left Join

The LEFT JOIN keyword returns all records from the left table and the matching records from the right table. If no matches are found, zero records from the right table will be shown as a result.

Syntax:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Note that the LEFT JOIN is called LEFT OUTER JOIN in some databases.

As an example, let's select all customers and any orders that these customers might have:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note that all records from the left table Customers will be returned, even if there are no matches in the right table Orders.

SQL Right Join

The RIGHT JOIN keyword essentially follows the same logic from the right side instead of the left one as described in the previous subsection.

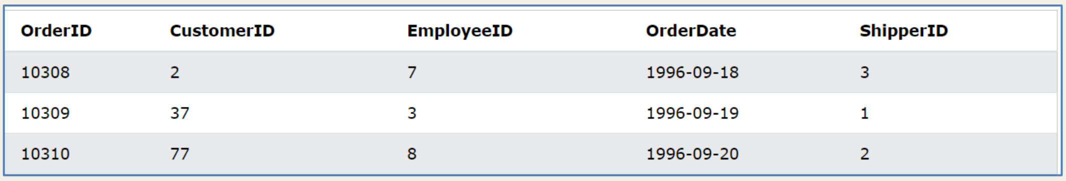
The RIGHT JOIN will return all records from the right table and the matching records from the left table, if there are any.

Syntax:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

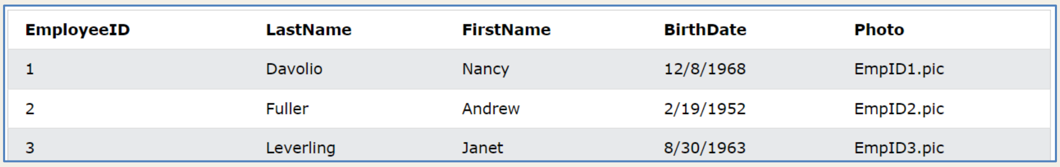
Consider the following two tables, Orders and Employees tables:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

***Table 26** - Orders Table in RIGHT JOIN example (**Source**: https://www.w3schools.com/sql/sql_join_right.asp)*



EmployeeID	LastName	FirstName	BirthDate	Photo
1	Davolio	Nancy	12/8/1968	EmpID1.pic
2	Fuller	Andrew	2/19/1952	EmpID2.pic
3	Leverling	Janet	8/30/1963	EmpID3.pic

***Table 27** - Employees Table in RIGHT JOIN example (**Source:** https://www.w3schools.com/sql/sql_join_right.asp)*

The following example will return all employees and any orders that they might have placed:

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Note that the RIGHT JOIN keyword will return all records from the right table, Employees, even if there are no matches found in the left table, Orders. The same logic applies as in the LEFT JOIN that we saw earlier.

SQL Full Join

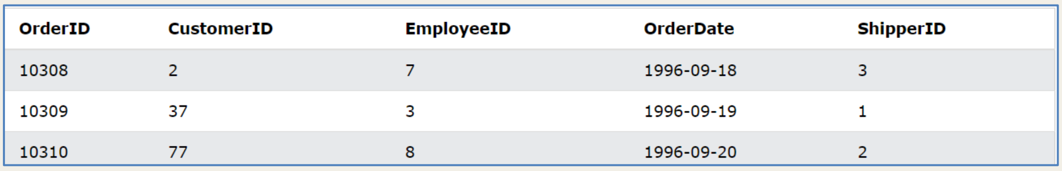
The FULL JOIN keyword returns all records when matching records are found in either the right or the left table.

Note that FULL OUTER JOIN and FULL JOIN are the same thing and a FULL JOIN can potentially return large result-sets.

Syntax:

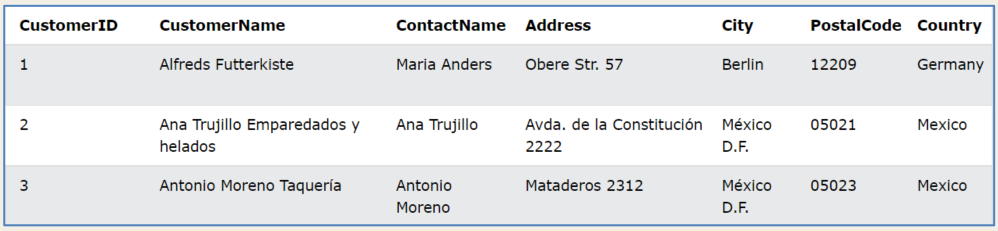
```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Consider the following two tables, the Orders table and the Customers table:



OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

Table 28 *- Orders Table in FULL JOIN example (**Source:** https://www.w3schools.com/sql/sql_join_full.asp)*



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

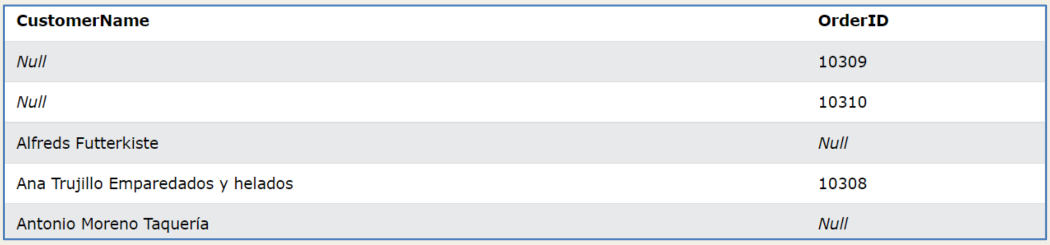
Table 29 *- Customers Table in FULL JOIN example (**Source**: https://www.w3schools.com/sql/sql_join_full.asp)*

An example that selects all customers and orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

The result of this full join can look something like this:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



CustomerName	OrderID
Null	10309
Null	10310
Alfreds Futterkiste	*Null*
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	*Null*

Table 30 *- FULL JOIN on Customers and Orders Tables Example (**Source**: https://www.w3schools.com/sql/sql_join_full.asp)*

Here we can see that it returns all matching records from both tables even if no common matches are found between the two tables. In the case of no common matches, a null value is assigned.

SQL Self Join

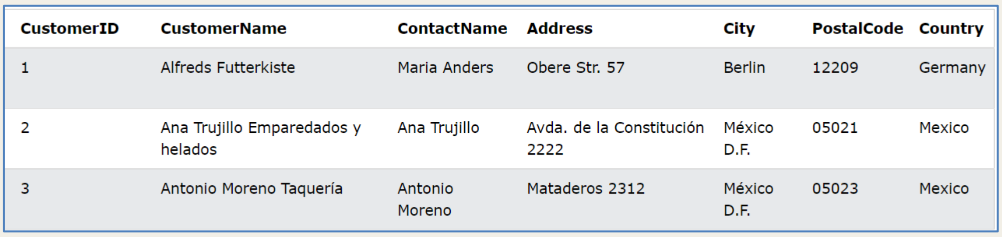
A self-join is considered a regular join, but the table is joined within.

Syntax:

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 and T2 are aliases used for the same table.

Let’s take the Customers table as an example:



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Table 31 *- Customers Table in SELF JOIN example (**Source**: https://www.w3schools.com/sql/sql_join_self.asp)*

Here, we want to select customers that are from the same city:

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS
CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

SQL Union

The UNION operator is used to combine the result-set of two or more SELECT statements.

There are a few requirements to enable a UNION:

1. Every SELECT statement within the UNION must have the same number of columns;
2. The columns must have similar data types;
3. The columns in every SELECT statement must be in the same order.

Syntax:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

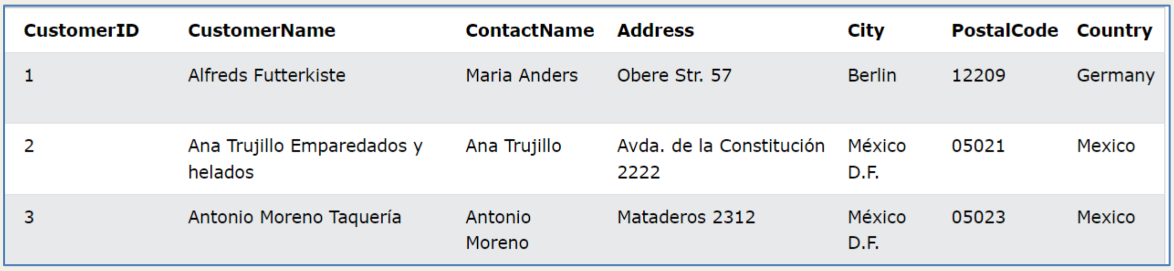
* Note that the UNION operator selects only distinct values by default.

To allow duplicate values, use UNION ALL:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

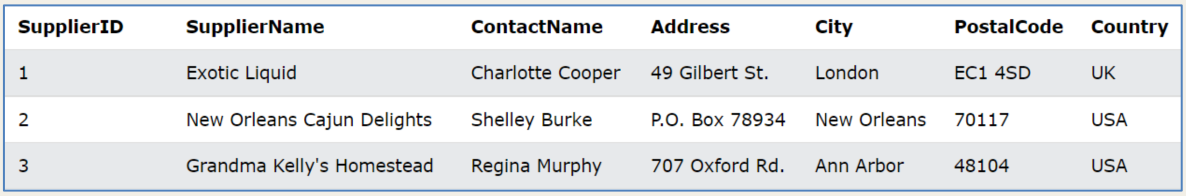
* Note that the column names in the two SELECT statements are usually equal.

Now let’s see some examples of UNION, UNION ALL, and UNION with where statements to understand a little bit better how we can use it.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

*Table 32 - Customers Table in UNION example (**Source:** https://www.w3schools.com/sql/sql_union.asp)*



SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

*Table 33 - Suppliers Table in UNION example (**Source:** https://www.w3schools.com/sql/sql_union.asp)*

The first example is used to return distinct cities from both tables shown above:

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Since we are using UNION, the suppliers from the same city will only be listed once. If you want to see the duplicated values, use UNION ALL.

The following example will do precisely that and return any duplicate values from both tables:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

The following example will return the distinct German cities from both the “Customers” and “Suppliers” tables with the use of WHERE:

```
SELECT City, Country FROM Customers
WHERE Country = ‘Germany’
UNION
SELECT City, Country FROM Customers
WHERE Country = ‘Germany’
ORDER BY City;
```

This example is similar to the previous one, but we will be returning possible duplicate values:

```
SELECT City, Country FROM Customers
WHERE Country = ‘Germany’
UNION ALL
SELECT City, Country FROM Customers
WHERE Country = ‘Germany’
ORDER BY City;
```

Another example will list all customers and suppliers:

```
SELECT 'Customer' AS Type, ContactName, City, Country
FROM Customers
UNION
SELECT 'Supplier', ContactName, City, Country
FROM Suppliers;
```

See that we used AS here to create an alias for the given query that will disappear after completing it.

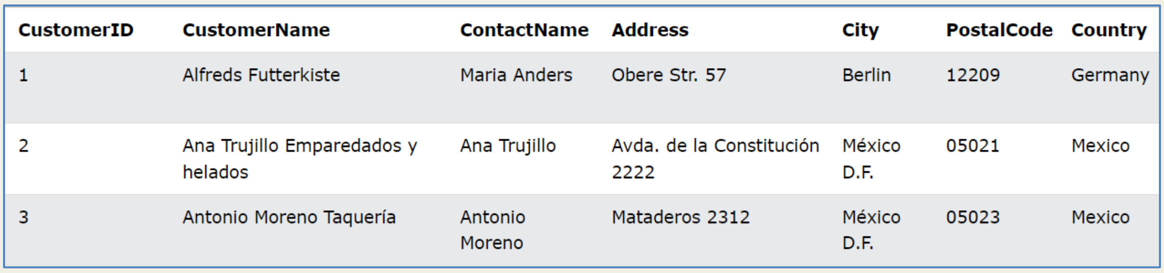
SQL Group By

The GROUP BY statement groups rows with the same values into summary rows. As an example, consider that you want to find the number of customers in each country. Also, the GROUP BY statement is often used with aggregate functions such as COUNT(), MAX(), MIN(), SUM(), AVG() to group the result by one or more columns.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

We will be using the Customers table, shown below, in our examples.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Table 34 *- Customers Table in GROUP BY example (**Source:** https://www.w3schools.com/sql/sql_groupby.asp)*

As a first example, let’s list the number of customers found in each county:

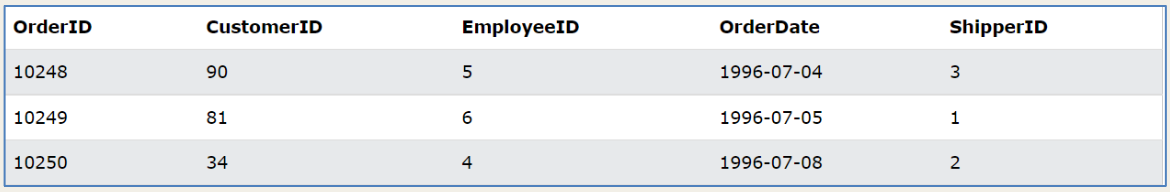
```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

As a second example, we will again list the number of customers in each country, but in descending order:

```
SELECT COUNT(CustomerID), Country
FROM Customers
```

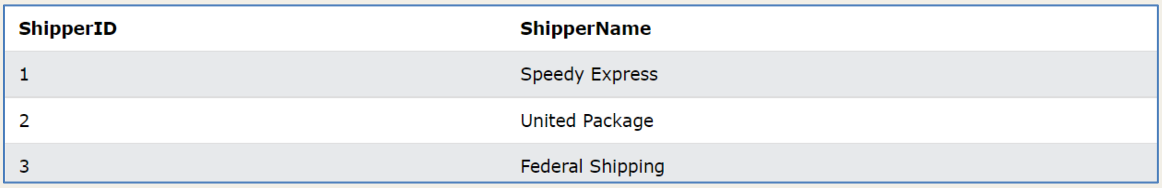
```
GROUP BY Country;
ORDER BY COUNT(CustomerID) DESC;
```

In the following example, we will use GROUP BY with JOIN by using the Orders and Shippers tables.



OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

__Table 35__ - Orders Table in GROUP BY example (__Source__: https://www.w3schools.com/sql/sql_groupby.asp)



ShipperID	ShipperName
1	Speedy Express
2	United Package
3	Federal Shipping

__Table 36__ - Shippers Table in GROUP BY example (__Source:__ https://www.w3schools.com/sql/sql_groupby.asp)

In this example, we will list the number of orders sent by each shipper:

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

See here we started by selecting the Shippers' names in the Shippers table and counted the orders based on their OrderID saved as an alias.

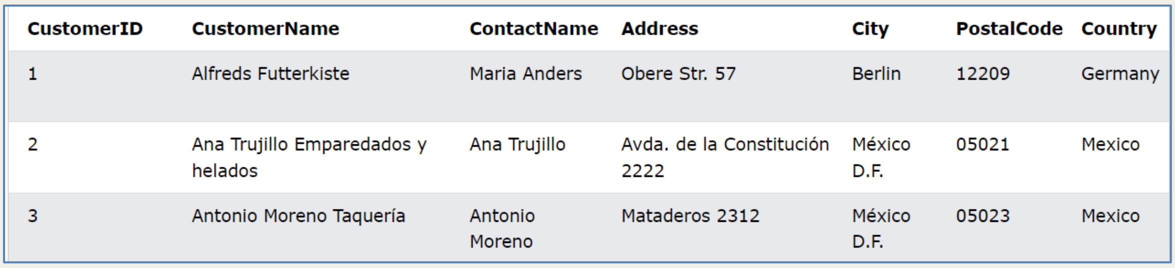
Then we performed a LEFT JOIN to join the Shippers table (table 2) on the Orders table (table 1) and group them by the Shipper's name.

SQL Having

The HAVING clause was added to SQL because WHERE cannot be used with aggregate functions.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

***Table 37** - Customers Table in HAVING example (**Source:** https://www.w3schools.com/sql/sql_having.asp)*

In this example, we will be using the Customers table once again. Here, we want to list the number of customers found in each country, but we also want to include countries that have more than five customers.

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

In this example, we want to list the number of customers per country again and include countries with more than five customers in descending order.

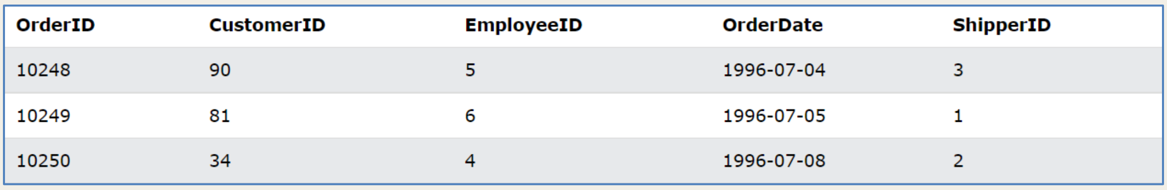
```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
```

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

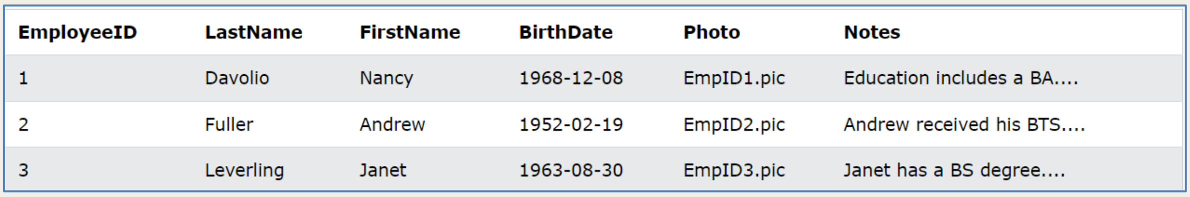
Let’s try a few other examples by combining what we have learnt so far.

We will use the Orders and Employees tables in the following two examples.



OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

Table 38 *- Orders Table in HAVING example (**Source:** https://www.w3schools.com/sql/sql_having.asp)*



EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	1968-12-08	EmpID1.pic	Education includes a BA....
2	Fuller	Andrew	1952-02-19	EmpID2.pic	Andrew received his BTS....
3	Leverling	Janet	1963-08-30	EmpID3.pic	Janet has a BS degree....

Table 39 *- Orders Table in HAVING example (**Source:** https://www.w3schools.com/sql/sql_having.asp)*

The following example will list the employees that have registered more than ten orders:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

In this example, we will list the employees “Davolio” or “Fuller” if they have registered orders more than 25 times:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
```

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
WHERE LastName = ‘Davolio’ OR LastName = ‘Fuller’
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

SQL Select Into

The SELECT INTO statement copies data from one table into a new table.

Syntax to copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

Syntax to copy only some columns into a new table:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

The new table will keep the column names and types the same as the old table. You can create new columns with the AS clause.

Example of creating a backup copy of Customers:

```
SELECT * INTO CustomersBackup2017
FROM Customers;
```

Example of using IN clause to copy the table into a new table in another database:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

Example to copy only a few columns into a new table:

```
SELECT CustomerName, ContactName INTO CustomersBackup2017
FROM Customers;
```

Example to copy only the German customers into a new table:

```
SELECT * INTO CustomersGermany
FROM Customers
WHERE Country = 'Germany';
```

Example to copy data from multiple tables into a new table:

```
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomersOrderBackup2017
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

SELECT INTO can also be used to create a new, empty table using the schema of another.

To do that, add a WHERE clause that returns no data:

```
SELECT * INTO newtable
FROM oldtable
WHERE 1 = 0;
```

SQL Insert Into Select

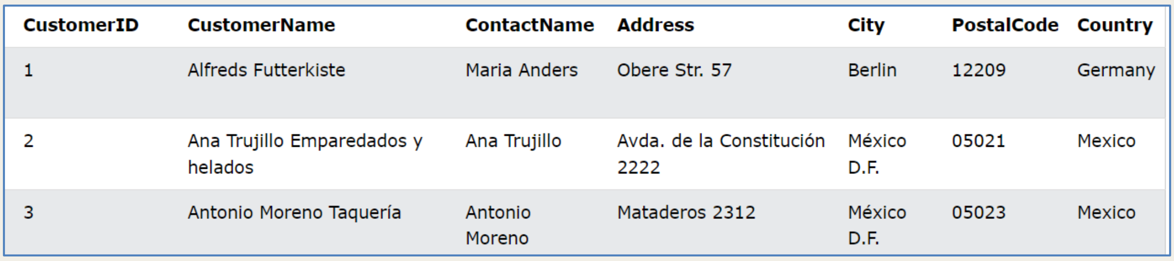
The INSERT INTO SELECT statement copies data from one table and inserts it into another. It requires the data types in the source and target table to match.

Syntax to copy all columns from one table to another:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

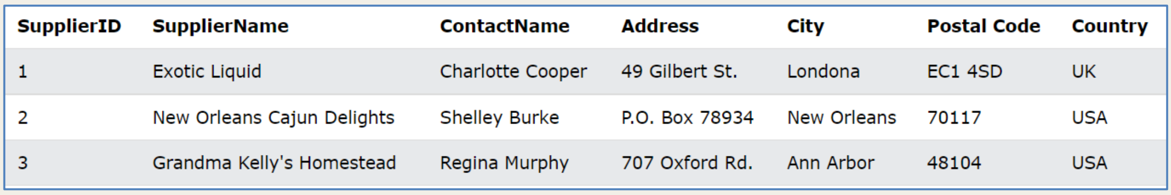
Syntax to copy only some columns from one table to another:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

***Table 40** - Customers Table in INSERT INTO SELECT example (**Source:** https://www.w3schools.com/sql/sql_insert_into_select.asp)*



SupplierID	SupplierName	ContactName	Address	City	Postal Code	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	Londona	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

***Table 41** - Suppliers Table in INSERT INTO SELECT example (**Source:** https://www.w3schools.com/sql/sql_insert_into_select.asp)*

We will be using the Customers and Suppliers tables, shown above, in the following examples.

The first example copies Suppliers into Customers (note that the columns that are not filled with data will contain null values):

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;
```

This example copies Suppliers into Customers to fill all columns:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
SELECT SupplierName, ContactName, Address, City, PostalCode, Country FROM Suppliers;
```

The third example copies only the German suppliers into Customers:

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers
WHERE Country='Germany';
```

SQL Case

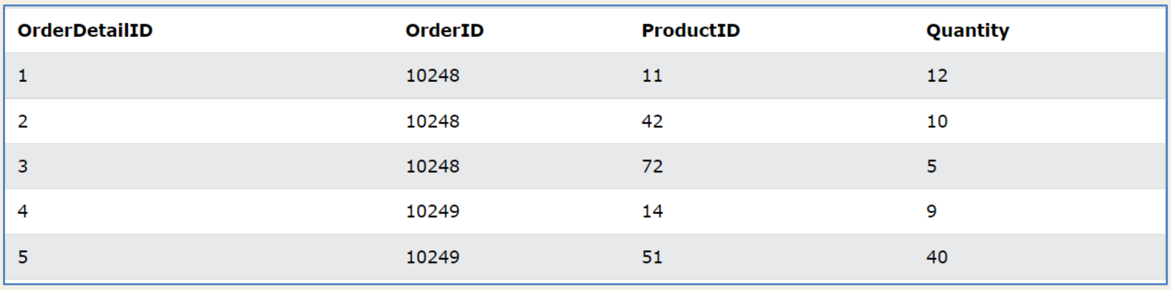
The CASE statement goes through a series of conditions and returns a value when the first condition is met. Think of it as an if, then, else statement.

When one condition is found true, it will stop going through the loop. If no conditions are found true, it will return the value in the ELSE clause.

Note that if there isn’t an ELSE clause and no conditions are found true, it will return NULL.

Syntax:

```
CASE
        WHEN condition1 THEN result1
        WHEN condition2 THEN result2
        WHEN conditionN THEN resultN
        ELSE result
END;
```



OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

***Table 42** - Orders Table in CASE example (**Source:** https://www.w3schools.com/sql/sql_case.asp)*

In the following examples, we will use the Orders table.

The first example will go through a series of conditions and return a value when the first condition is met:

```
SELECT OrderID, Quantity,
CASE
        WHEN Quantity > 30 THEN 'The quantity is greater than 30'
        WHEN Quantity = 30 THEN 'The quantity is 30’
        ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

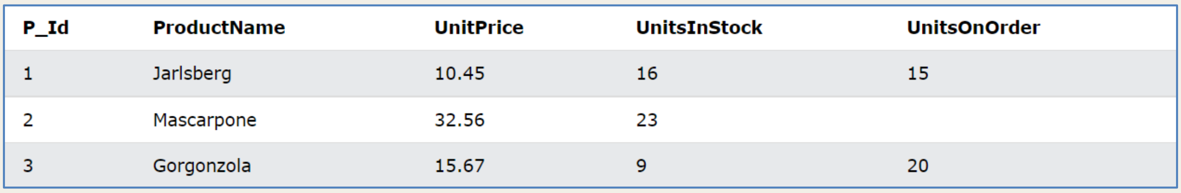
In this second example, we will order the customers by City. Note that if City is NULL, it will be ordered by Country.

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
     (CASE
           WHEN City IS NULL THEN Country
           ELSE City
END);
```

SQL Null Functions

The NULL functions include the following: IFNULL(), ISNULL(), COALESCE(), and NVL().

Here, we will use the “Products” table:



P_Id	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder
1	Jarlsberg	10.45	16	15
2	Mascarpone	32.56	23	
3	Gorgonzola	15.67	9	20

Table 43 *- Products Table in NULL functions example (**Source:** https://www.w3schools.com/sql/sql_isnull.asp)*

Let’s say that the “UnitsOnOrder” column is optional and may contain NULL values.

Example:

```
SELECT ProductName, UnitPrice * (UnitsInStock + UnitsOnOrder)
FROM Products;
```

Here we can see that if any of the UnitsOnOrder values are null, the result will also be null.

Let’s see how we can overcome this issue.

In MySQL, you can use the ISNULL() function that lets you return an alternative value if an expression is null:

```
SELECT ProductName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
FROM Products;
```

Or we can use the COALESCE() function:

```
SELECT ProductName, UnitPrice * (UnitsInStock + COALESCE(UnitsOnOrder, 0))
FROM Products;
```

In SQL Server, the ISNULL() function does the same thing as in MySQL:

```
SELECT ProductName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
FROM Products;
```

In MS Access IsNull() function returns TRUE(-1) if the expression is a null value, otherwise FALSE (0):

```
SELECT ProductName, UnitPrice * (UnitsInStock + IIF(IsNull(UnitsOnOrder), 0,
UnitsOnOrder))
FROM Products;
```

In Oracle, the NVL() function does the same thing:

```
SELECT ProductName, UnitPrice * (UnitsInStock + NVL(UnitsOnOrder, 0))
FROM Products;
```

SQL Comments

SQL Comments explain SQL statement sections or prevent their execution.

Note that the examples in this section are not supported in Firefox and Microsoft Edge, which are Microsoft Access databases. Comments are generally not supported in Microsoft Access databases.

Single line comments in SQL start with - - (two dashes):

```
--Select all:
SELECT * FROM Customers;
```

Or it can be used like this to ignore the end of the line:

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

Or to ignore a statement:

```
--SELECT * FROM Customers;
SELECT * FROM Products;
```

Multiple-line comments start with /* and end with */. Any text written between these two will be ignored.

Example:

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

To ignore part of a statement, you can also use /**/.

Example 1:

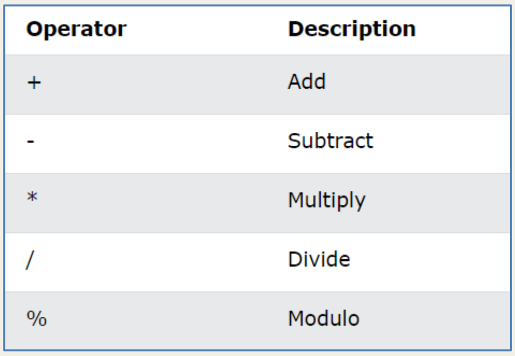
```
SELECT CustomerName, /*City,*/ Country FROM Customers;
```

Example 2:

```
SELECT * FROM Customers WHERE (CustomerName LIKE 'L%'
OR CustomerName LIKE 'R%' /*OR CustomerName LIKE 'S%'
OR CustomerName LIKE 'T%'*/ OR CustomerName LIKE 'W%')
AND Country='USA'
ORDER BY CustomerName;
```

SQL Operators

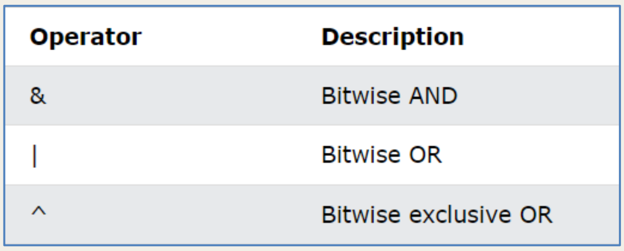
Arithmetic Operators used in SQL:



Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

***Table 44** - Arithmetic Operators (**Source:** https://www.w3schools.com/sql/sql_operators.asp)*

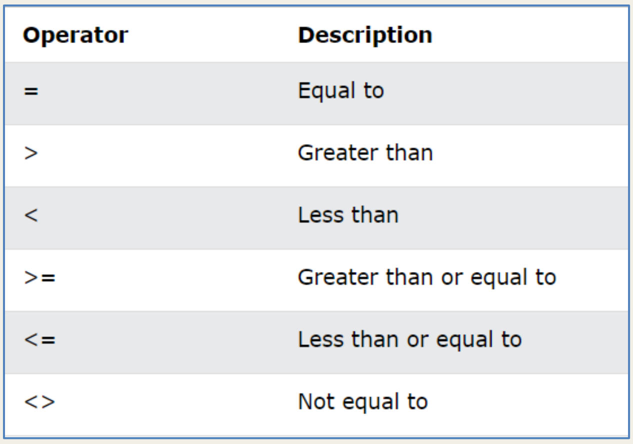
Bitwise operators used in SQL:



Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

***Table 45** – Bitwise operators (**Source**: https://www.w3schools.com/sql/sql_operators.asp)*

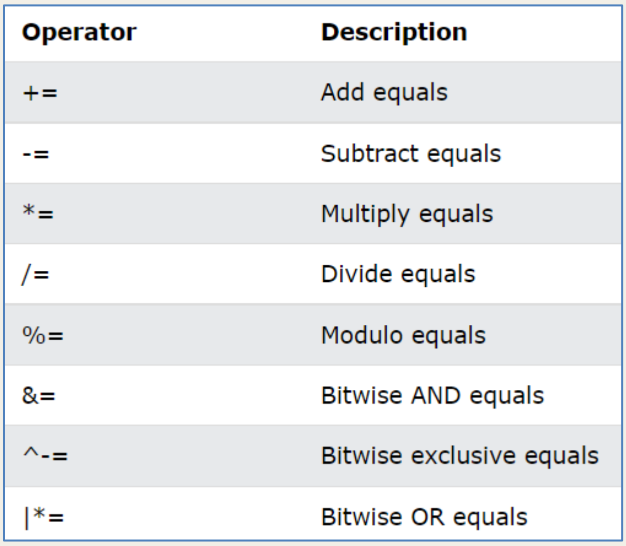
Comparison operators used in SQL:



Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

***Table 46** – Comparison operators (**Source**: https://www.w3schools.com/sql/sql_operators.asp)*

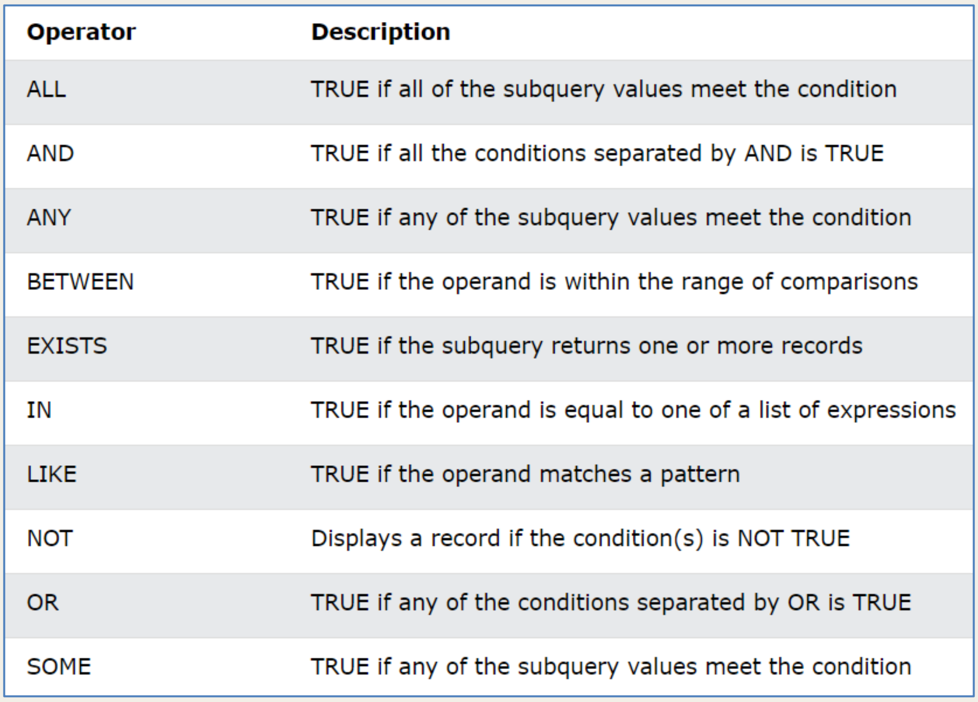
Compound operators used in SQL:



Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

***Table 47** - Compound operators (**Source:** https://www.w3schools.com/sql/sql_operators.asp)*

Logical operators used in SQL:



Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

***Table 48** - Logical operators (**Source**: https://www.w3schools.com/sql/sql_operators.asp)*

4.2. SQL Databases

As we have mentioned in the previous section that was dedicated to the basic statements used in SQL, this programming language is mainly used for relational databases. Therefore, in this section, we will learn how to create a database, modify it, and manipulate it with SQL.

Let’s start simple, and we will build into slightly more complicated statements.

SQL Create DB

The CREATE DATABASE statement creates a new SQL database.

Syntax:
CREATE DATABASE DatabaseName;

Note: Always remember that the name of the database should be unique within the Relational Database Management System (RDMS) that you are using, and make sure that you have admin privileges before creating any database.

Let’s say you want to create a test database. You would use the following statement:

CREATE DATABASE testDB;

SQL Drop DB

The DROP DATABASE statement deletes an existing SQL database.

DROP DATABASE DatabaseName;

Before you delete the database, make sure that you don’t need any of the information that it contains because it completely deletes it.

Remember the database that we just created called “testDB”? Now we are going to delete it.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Example:

```
DROP DATABASE testDB;
```

SQL Backup DB

The BACKUP DATABASE statement does a complete backup on an existing SQL database.

To use this statement, you need to provide two things: the name of the database and the file path.

```
BACKUP DATABASE DatabaseName
TO DISK = ‘filepath’
```

Example:

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak';
```

Note: To avoid technical problems, it is better to back up the database to a different drive than the one the existing database is on.

There is also another option where you perform a differential backup based on changes that have been made since the last complete database backup. This type of backup also reduces the backup time.

To do this, you follow this syntax:

```
BACKUP DATABASE DatabaseName
TO DISK = 'filepath'
WITH DIFFERENTIAL;
```

Example:

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak'
WITH DIFFERENTIAL;
```

SQL Create Table

The CREATE TABLE statement creates a new table in a database.

Syntax:

```
CREATE TABLE table_name (
                column1 datatype,
                column2 datatype,
                column3 datatype,
                …….
);
```

In this statement, you need to specify the **names of the columns** and the **type of data** that the column will contain.

There are many data types such as integer, date or varchar. Depending on the type of data that you want to store, you choose the most suitable option. For instance, if you have a column named “Date of Birth”, then you would probably choose the Date as the data type.

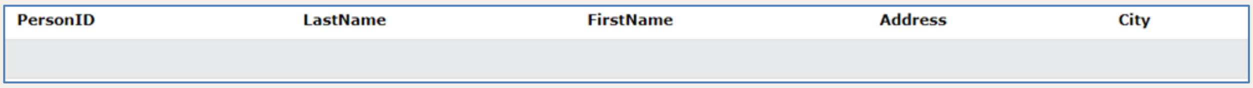
Example:

```
CREATE TABLE Persons (
                PersonID int,
                LastName varchar(255),
                FirstName varchar(255),
                Address varchar(255),
                City varchar(255)
);
```

This example will create a table with the name Persons and will contain 5 columns.

The PersonID will contain an integer (int); the columns LastName, FirstName, Address, and City will contain characters with a maximum length of 255.

The table will look something like this without any data:



PersonID	LastName	FirstName	Address	City

***Table 49** - Empty table in CREATE TABLE Example (**Source:** https://www.w3schools.com/sql/sql_create_table.asp)*

You can also create a table by using another table and choosing which columns you want in the new table. Keep in mind that the data of the existing table will fill the entries of the new table.

The syntax is as follows:

```
CREATE TABLE new_table_name AS
SELECT column1, column2,...
FROM existing_table_name
WHERE ....;
```

As you have learnt in the previous section:

- SELECT specifies the columns from the existing table,
- FROM specifies the name of the existing table, and
- WHERE can be used if you want a set of records that fulfil a specified condition.

Example:

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

SQL Drop Table

Similar to the DROP DATABASE statement that we saw earlier, the DROP TABLE statement deletes an existing table in a database.

Remember that you need to be sure that you do not need any of the information contained in a table before deleting it.

Syntax:

```
DROP TABLE TableName;
```

Example:

```
DROP TABLE Persons;
```

You can also choose to delete the data contained in a table, but not the table itself.

Maybe you created a new table from an existing table that has the structure that you want, but you want to add completely new entries. That is where TRUNCATE TABLE is useful.

Syntax:

```
TRUNCATE TABLE TableName;
```

Example:

```
TRUNCATE TABLE Persons;
```

SQL Alter Table

The ALTER TABLE statement can add, delete or modify columns in an existing table. Also, it can be used to add and drop constraints on an existing table.

Let’s see the syntax of adding a column first:

```
ALTER TABLE TableName
ADD column_name datatype;
```

This is familiar to how we created a table by specifying the name of the column and the type of data to be contained in that column.

Example:

```
ALTER TABLE Customers
ADD Email varchar(255);
```

To delete a column in a table, as we have seen before, you use the DROP statement.

Keep in mind that some database systems do not allow for users to delete a column.

Syntax:

```
ALTER TABLE TableName
DROP COLUMN ColumnName;
```

As an example, let’s delete the column that we created:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

To change the data type of a column, you can use the following statements depending on the RDBMS that you are using:

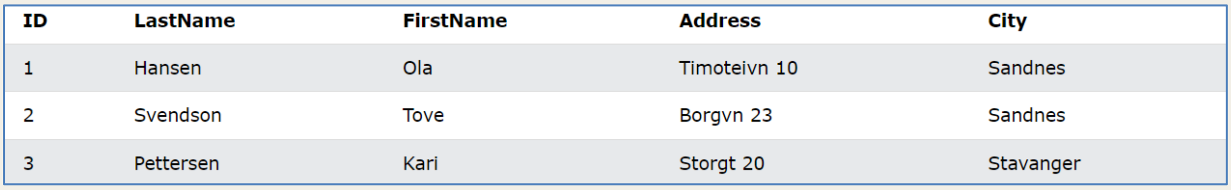
- ALTER COLUMN (for SQL Server/MS Access);
- MODIFY COLUMN (for My SQL/ Oracle prior to version 10G);
- MODIFY (for Oracle version 10G and later).

Syntax:

```
ALTER TABLE TableName
ALTER COLUMN ColumnName datatype;
```

Note that the second statement is the one that changes depending on the RDBMS that you are using from ALTER COLUMN to MODIFY COLUMN or MODIFY. The rest stays the same.

Let’s see an example to understand this statement a bit better. The table underneath is the “Persons” table and contains information about different people.



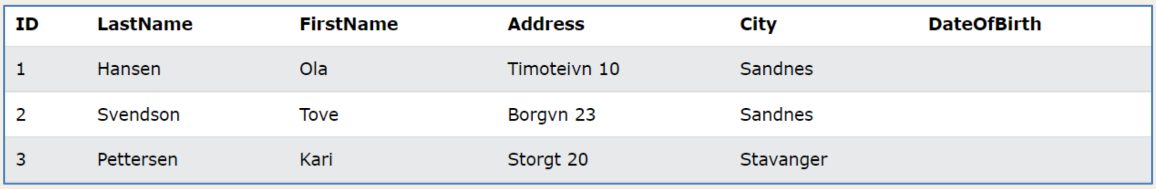
ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Table 50 *- ALTER TABLE Example (**Source:** https://www.w3schools.com/sql/sql_alter.asp)*

As an example, let’s say that we wanted to add a column named “DateofBirth” in this table. We will use the following statement:

```
ALTER TABLE Persons
ADD DateofBirth date;
```

The new column that we added to the table has the data type of date, which means that it stores data in a date format. Underneath, you can see the table with the new column added.



ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Table 51 *- ALTER TABLE Example (**Source**: https://www.w3schools.com/sql/sql_alter.asp)*

However, what if you changed your mind and wanted to change the data type of the new column, then you can use the ALTER COLUMN statement. For example, we can change the newly added column’s type from date to year. To do this, use the following statement:

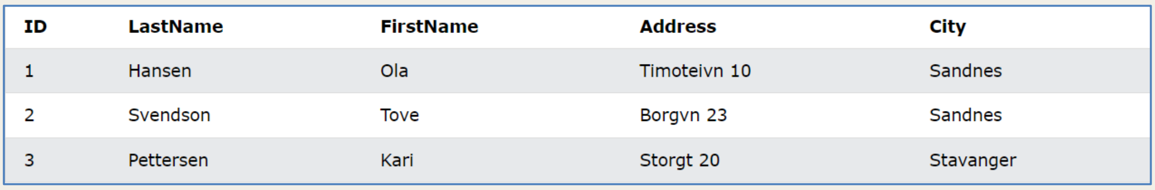
```
ALTER TABLE Persons
ALTER COLUMN DateofBirth year;
```

The year data type holds a year in two- or four-digits format.

To delete the column that we just altered, we use the DROP COLUMN statement.

```
ALTER TABLE Persons
DROP COLUMN DateofBirth;
```

Our table will go back looking the way it did in the beginning.



ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

***Table 52** - ALTER TABLE Example (**Source**: https://www.w3schools.com/sql/sql_alter.asp)*

SQL Constraints

SQL Constraints are used when the table is created with the statement CREATE TABLE or after the table is created with the statement ALTER TABLE.

Syntax:

```
CREATE TABLE table_name (
        column1 datatype constraint,
        column2 datatype constraint,
        column3 datatype constraint,
        ....
);
```

Constraints are used to specify a set of rules and restrictions that apply to a column or a table. They are used to ensure the integrity, accuracy, and reliability of the data. If the constraints are applied to a table, then all columns need to adhere to these constraints.

The following constraints are the ones that are most commonly used:

- NOT NULL

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT
- CREATE INDEX

We will go through each of these constraints to explain their usage and syntax with examples.

SQL Not Null

In SQL, columns can hold null values by default. The NOT NULL constraint is used to avoid null values in columns. This is particularly important to ensure that when a new entry is added to a table all the necessary fields are filled.

As an example, let’s say that we want to create a table named “Persons” and we want to ensure that the columns “ID”, “LastName”, and “FirstName” do not hold any null values:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255) NOT NULL,
        Age int
);
```

If, for some reason, you want to alter an already existing table to add constraints, you can use the following statement:

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

SQL Unique

The UNIQUE constraint is used to ensure that all values stored in a column are unique among the rows in a table. To make this clearer, think of the variable ID. You wouldn't want two people to have the same ID, therefore you would use the constraint UNIQUE on this occasion.

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
        ID int NOT NULL UNIQUE,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

My SQL:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        UNIQUE (ID)
);
```

As you can see, depending on the RDBMS that you are using, there are a few adjustments on where the UNIQUE constraint is put in the code.

If you want to name or define a UNIQUE constraint on multiple columns, use the following:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
       CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

You can also add a UNIQUE constraint after the table has been created by using the ALTER TABLE statement that we learnt earlier.

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD UNIQUE (ID);
```

If you also want to name and define a UNIQUE constraint on multiple already existing columns, you use the following statement:

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Persons UNIQUE (ID, LastName);
```

To delete the UNIQUE constraint, you can use the following statement:

My SQL:

```
ALTER TABLE Persons
DROP INDEX UC_Persons;
```

SQL Server/Oracle/ MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT UC_Persons;
```

SQL Primary Key

The PRIMARY KEY constraint is used to uniquely identify each row or record in a table. Note that primary keys must contain unique values, but cannot contain null values.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

A table can only have **ONE** primary key and that primary key can consist of one or multiple columns.

SQL Server/Oracle/MS Access:

```
CREATE TABLE Persons (
        ID int NOT NULL PRIMARY KEY,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

MySQL:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        PRIMARY KEY (ID)
);
```

The following example allows you to name and define a PRIMARY KEY constraint on multiple columns:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Note that the PRIMARY KEY is still one, but the value of the primary key encompasses two columns.

You can also create a PRIMARY KEY constraint on an existing table by using the following statement:

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

To add and define a PRIMARY KEY constraint on an existing table, use the following statement:

```
ALTER TABLE Persons
ADD CONSTRAINT PK_Persons PRIMARY KEY (ID, LastName);
```

To drop a PRIMARY KEY constraint, use the following statements according to your RDBMS.

MySQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

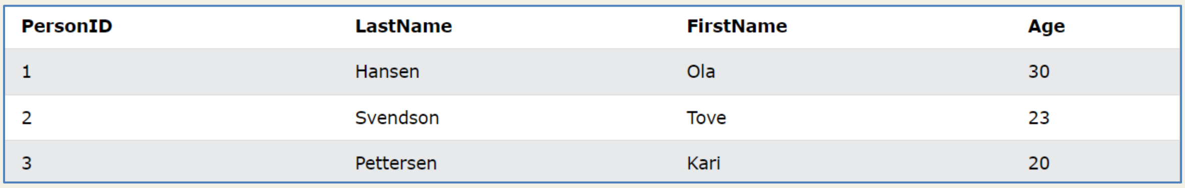
SQL Foreign Key

The FOREING KEY represents the columns of a table that are linked to a primary key in another table. The table that has a foreign key is called the child table, whereas the table that has the primary key is called the referenced or parent table.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

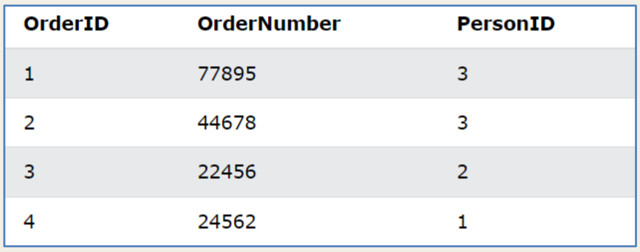
This type of constraint is used to prevent any actions that would destroy links between parent and child tables.

Let’s consider the following two tables:



PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Table 53 *- Persons table in FOREING KEY Example (**Source**: https://www.w3schools.com/sql/sql_foreignkey.asp)*



OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Table 54 *- Orders table in FOREING KEY Example (**Source**: https://www.w3schools.com/sql/sql_foreignkey.asp)*

These two tables are linked by the column “PersonID” that is found in both tables. Now, the primary key is located in the Persons table and the foreign key is the “PersonID” in the Orders table.

The FOREIGN KEY constraint works by preventing the input of invalid data in the foreign key column, because it is linked with the parent table and its values need to be identical.

To use the FOREIGN KEY constraint when creating a table, you can use the following statement according to your RDBMS.

SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
        OrderID int NOT NULL PRIMARY KEY,
        OrderNumber int NOT NULL,
        PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

My SQL:

```
CREATE TABLE Orders (
        OrderID int NOT NULL,
        OrderNumber int NOT NULL,
        PersonID int,
        PRIMARY KEY (OrderID),
        FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

This statement linked the Orders table to the Persons table with the FOREIGN KEY constraint based on PersonID column.

SQL Check

The CHECK constraint is used to specify the values allowed in a column or in certain columns of a table based on values found in other columns of the same row.

Example of CHECK constraint on CREATE TABLE

The following example is used to ensure that a person is not under the age of 18, so the CHECK constraint is added to the "Age" column.

MySQL:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
    Age int,
     CHECK (Age>=18)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int CHECK (Age>=18)
);
```

If you want to name a CHECK constraint and use the constraint on multiple columns, you can use the following statement.

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
   City varchar(255),
   CONSTRAINT CHK_Person CHECK (Age>=18 AND City= 'Sandnes')
);
```

Example of CHECK constraint on ALTER TABLE

To create a constraint on an already existing table, use the following statement.

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

To name a constraint and create it on multiple columns, you can use:

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_Person CHECK (Age>=18 AND City= ‘Sandnes’);
```

Example of DROP a CHECK constraint

To eliminate a CHECK constraint, you can use the following according to the RDMBS.

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;
```

MySQL:

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

SQL Default

The DEFAULT constraint is used to specify a default value for a column. If there are no other values specified, the default value will be added to all new records.

Example of DEFAULT constraint on CREATE TABLE

The following example adds a default value to the City column when the Persons table is created:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        City varchar(255) DEFAULT 'Sandnes'
);
```

This constraint can also be used to insert system values with functions such as GETDATE():

```
CREATE TABLE Orders (
            ID int NOT NULL,
            OrderNumber int NOT NULL,
            OrderDate date DEFAULT GETDATE()
);
```

Example of DEFAULT constraint on ALTER TABLE

In this example, the column “City” is used to create a DEFAULT constraint when we are altering an already existing table.

MySQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

SQL Server:

```
ALTER TABLE Persons
ADD CONSTRAINT df_City
DEFAULT 'Sandnes' FOR City;
```

MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City SET DEFAULT ‘Sandnes’;
```

Oracle:

```
ALTER TABLE Persons
MODIFY City DEFAULT 'Sandnes';
```

Example of DROP a DEFAULT constraint

MySQL:

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT;
```

SQL Index

The CREATE INDEX statement creates an index on a table. Indexes are useful when you want to retrieve data more quickly.

Please note that tables with indexes take more time to update in comparison to tables without. Therefore, it is suggested to only create indexes on columns that are frequently searched.

To CREATE INDEX on a table where duplicate values are allowed, use the following syntax:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

To CREATE UNIQUE INDEX on a table where duplicate values are not allowed, use the following syntax:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

Keep in mind that creating indexes varies from database to database, so always check the syntax to create one in your database.

Examples of CREATE INDEX

In this example, we are creating an index on the LastName column by specifying the name idx_lastname:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

To create an index on a combination of columns, use the following statement:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

If you want, you can add more columns in the parenthesis.

Examples of DROP INDEX

If you want to delete an index, use the following statement according to your RDBMS.

MS Access:

```
DROP INDEX index_name ON table_name;
```

SQL Server:

```
DROP INDEX table_name.index_name;
```

DB2/Oracle:

```
DROP INDEX index_name;
```

MySQL:

```
ALTER TABLE table_name
DROP INDEX index_name;
```

SQL Auto Increment

Auto-increment is used to generate unique numbers automatically when a new record is entered into a table. This is usually used on the primary key field in order to ensure that no one person has the same ID.

This feature uses different syntax in MySQL, SQL Server, Access and Oracle. Therefore, we will be going through each of these to explain how to use Auto-Increment.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

MySQL:

```
CREATE TABLE Persons (
        Personid int NOT NULL AUTO_INCREMENT,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
       PRIMARY KEY (Personid)
);
```

In MySQL, AUTO_INCREMENT adds the auto-increment feature and by default, the value set is 1 and it goes up by 1 each time.

If you would like the sequence to start from a different value, use the following statement:

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

If you enter a new record into the Persons table, you will not have to specify a value for the “PersonID” column since it will be generated automatically:

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen');
```

SQL Server

We are following the same example as above, where we use the “Personsid” column as the primary key in the Persons table:

```
CREATE TABLE Persons (
        Personid int IDENTITY(1,1) PRIMARY KEY,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

In SQL Server, the auto-increment feature uses the keyword IDENTIFY to be activated. The two values in the parenthesis indicate (starting value, adding value for each new record). It will start at 1 and go up by 1 each time a new record is entered.

If you wanted to change the starting value to 10 and to add 5 each time a new record is added, you would write it like this IDENTIFY (10,5).

When entering new records, you do not need to specify the Personsid. It will be automatically generated as in the example above.

MS Access

```
CREATE TABLE Persons (
        Personid AUTOINCREMENT PRIMARY KEY,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

MS Access uses AUTOINCREMENT keyword to activate the auto-increment feature. Similar to the other two, the starting value is one and it adds up by one each time a record is added.

You can specify different values such as 10 for starting value and 5 for each addition with AUTOINCREMENT(10,5).

Again, note that each time we add a new record, we do not need to specify the Personid value. It is generated automatically.

Oracle

In Oracle, the code is a bit trickier. To create an auto-increment field, you need to create a sequence of numbers:

```
CREATE SEQUENCE seq_person
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10;
```

This sequence creates a sequence object named “seq_person”, sets the minimum value to start from (which is 1 in this instance), then specifies the increment by 1. The cache specifies how many sequence values should be stored in memory for faster access.

Unlike the previous examples, to enter a new record into the Persons table, you need to use the nextval function. This function is used to retrieve the next value from the sequence object that we created.

```
INSERT INTO Persons (Personid,FirstName,LastName)
VALUES (seq_person.nextval,'Lars','Monsen');
```

Here, we can see that the Personid column is selected to be assigned the next number from the sequence object that we created called “seq_person”.

SQL Dates

One of the most challenging parts when working with dates is to ensure that the format of the date you are trying to enter is the same with the format of the date column in the database.

It is important to note that data that contains only date portions will work as expected in queries. However, if there is a time portion, things get a bit more complicated.

Date Data types found in MySQL:

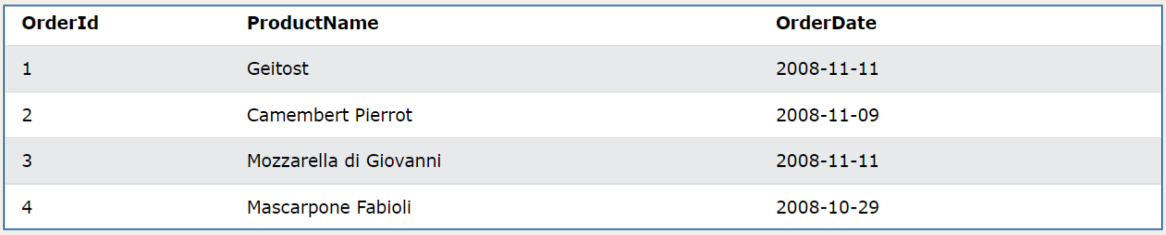
- DATE - format YYYY-MM-DD

- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

Data types found in SQL Server:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: a unique number

Keep in mind that the data types are chosen when you are creating a new table in your database.



OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
2	Camembert Pierrot	2008-11-09
3	Mozzarella di Giovanni	2008-11-11
4	Mascarpone Fabioli	2008-10-29

***Table 55** - Orders table in Dates Example (**Source**: https://www.w3schools.com/sql/sql_dates.asp)*

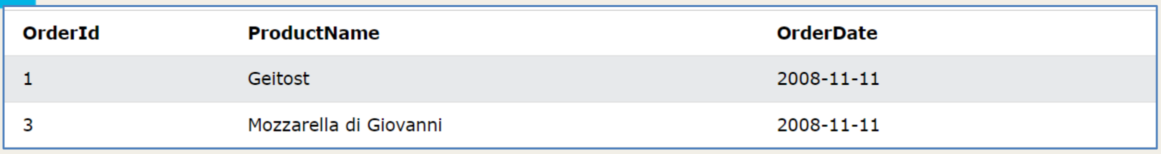
We will use the Orders table in our example to select the records with an OrderDate of “2008-11-11”.

Example:

```
SELECT *
FROM Orders
WHERE OrderDate='2008-11-11';
```

The expected result will look something like this:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

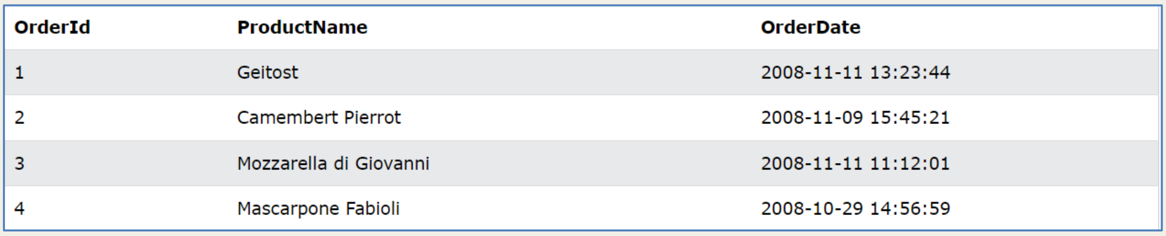


OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
3	Mozzarella di Giovanni	2008-11-11

***Table 56** - Result of OrderDate query in Dates Example (**Source**: https://www.w3schools.com/sql/sql_dates.asp)*

Note that two dates can be easily compared when there is no time stamp involved.

Suppose that you have the Orders table, but with a timestamp in the OrderDate column.



OrderId	ProductName	OrderDate
1	Geitost	2008-11-11 13:23:44
2	Camembert Pierrot	2008-11-09 15:45:21
3	Mozzarella di Giovanni	2008-11-11 11:12:01
4	Mascarpone Fabioli	2008-10-29 14:56:59

***Table 57** - Orders table with timestamp in Dates Example (**Source**: https://www.w3schools.com/sql/sql_dates.asp)*

Here, if you attempted to use the same query as we used above:

```
SELECT *
FROM Orders
WHERE OrderDate='2008-11-11';
```

You would get no result, because the query is not taking into account the time stamp. It is recommended to not use time stamps unless you absolutely have to.

SQL Views

In SQL, a view is a virtual table of a result-set created from a specific query. A view is useful when you want to view and present data through a combination of tables.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
```

```
FROM table_name
WHERE condition;
```

Note that a view always shows up-to-date data since the database recreates the virtual table, every time users query it.

Example to query all customers from Brazil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

To query the view:

```
SELECT * FROM [Brazil Customers];
```

Another example is to create a view that selects every product in the Products table with a price that is higher than the average price:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

To query the view above, use the following statement:

```
SELECT * FROM [Products Above Average Price];
```

To update a view, use the CREATE OR REPLACE VIEW statement:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

The following example adds the “City” column to the Brazil Customer view that we created earlier:

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = 'Brazil';
```

To delete a view, use the DROP VIEW statement:

```
DROP VIEW view_name;
```

For example, suppose we want to delete the “Brazil customers” view:

```
DROP VIEW [Brazil Customers];
```

SQL Data Types

Generally, each column in a table requires a name and a data type.

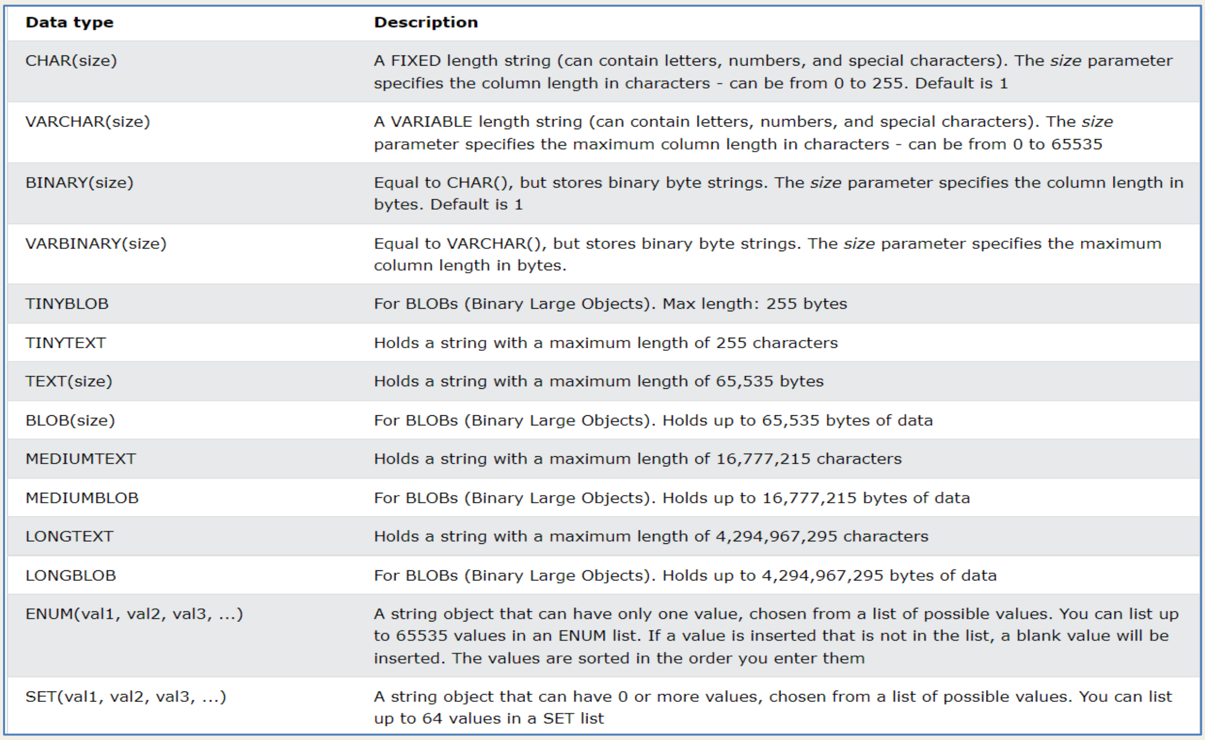
An SQL developer will need to decide the type of data that will be stored inside each column when creating a table. The data type is used for SQL to understand the data that will be contained in each column and also how it will interact with the data.

Please keep in mind that data types might have different names in different databases. Always check the documentation even if the name is the same because other details might be different like the size.

Data types in MySQL (Version 8.0)

MySQL has three main data types: string, numeric, and date/time.

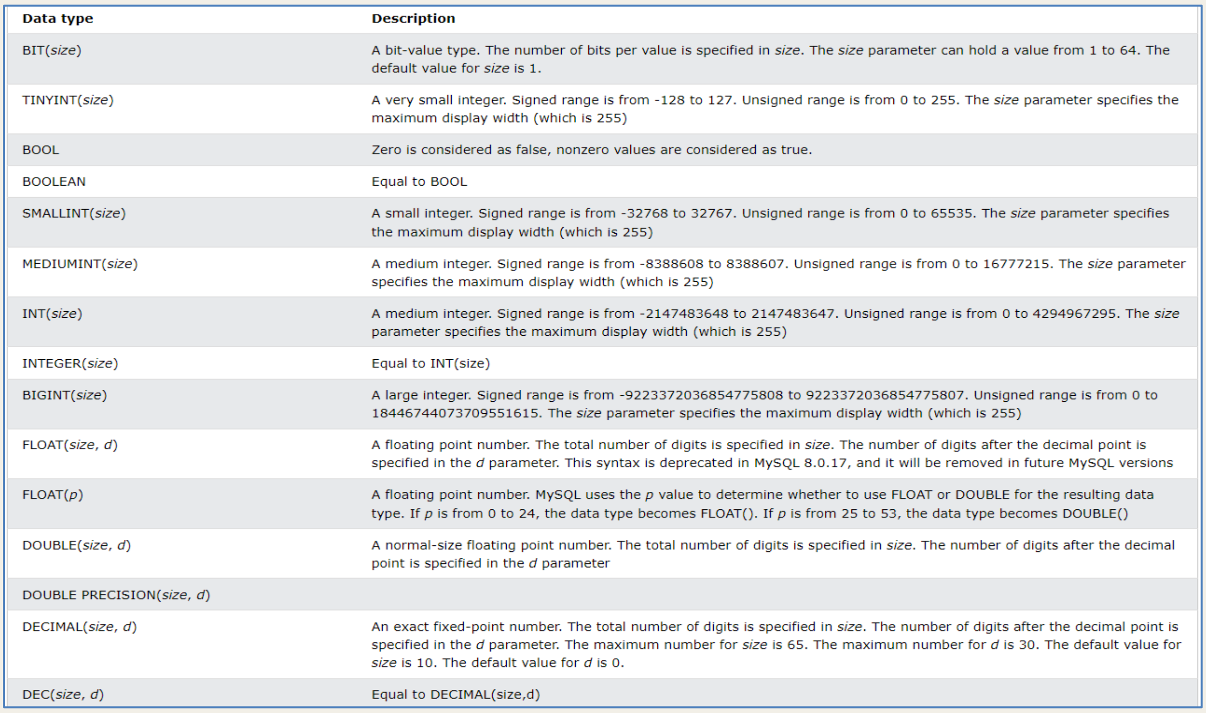
String Data Types



Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

***Table 58** – String Data types (MySQL) (**Source**: https://www.w3schools.com/sql/sql_datatypes.asp)*

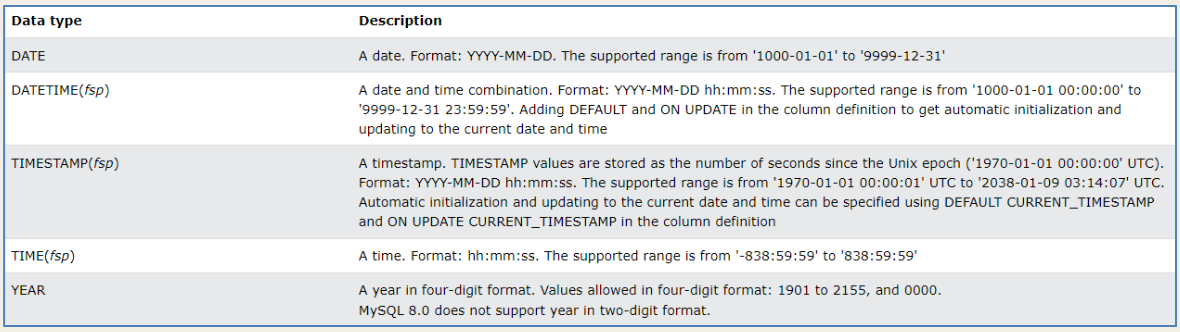
Numeric Data Types



Data type	Description
BIT(*size*)	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.
TINYINT(*size*)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(*size*)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255)
MEDIUMINT(*size*)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The *size* parameter specifies the maximum display width (which is 255)
INT(*size*)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)
INTEGER(*size*)	Equal to INT(size)
BIGINT(*size*)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The *size* parameter specifies the maximum display width (which is 255)
FLOAT(*size*, *d*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(*p*)	A floating point number. MySQL uses the *p* value to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(*size*, *d*)	A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter
DOUBLE PRECISION(*size*, *d*)	
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.
DEC(*size*, *d*)	Equal to DECIMAL(size,d)

***Table 59** – Numeric Data types (MySQL) (**Source**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Date/Time Data Types

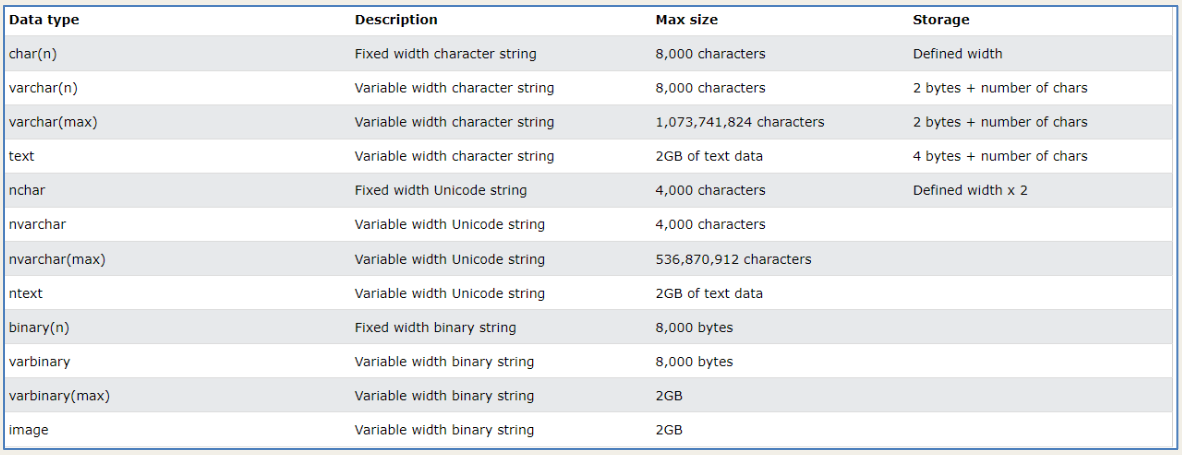


Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(*fsp*)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(*fsp*)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

***Table 60** – Date/Time Data types (MySQL) (**Source**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Data Types in SQL Server

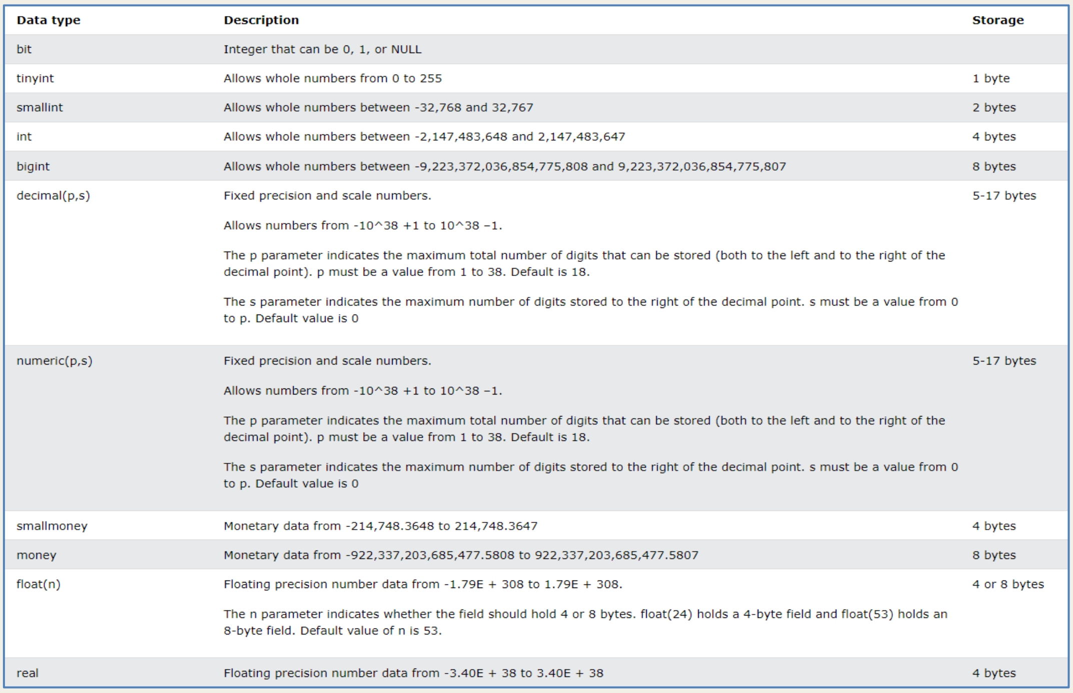
String Data Types



Data type	Description	Max size	Storage
char(n)	Fixed width character string	8,000 characters	Defined width
varchar(n)	Variable width character string	8,000 characters	2 bytes + number of chars
varchar(max)	Variable width character string	1,073,741,824 characters	2 bytes + number of chars
text	Variable width character string	2GB of text data	4 bytes + number of chars
nchar	Fixed width Unicode string	4,000 characters	Defined width x 2
nvarchar	Variable width Unicode string	4,000 characters	
nvarchar(max)	Variable width Unicode string	536,870,912 characters	
ntext	Variable width Unicode string	2GB of text data	
binary(n)	Fixed width binary string	8,000 bytes	
varbinary	Variable width binary string	8,000 bytes	
varbinary(max)	Variable width binary string	2GB	
image	Variable width binary string	2GB	

Table 61 *– String Data types (SQL Server) (**Source**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Numeric Data Types

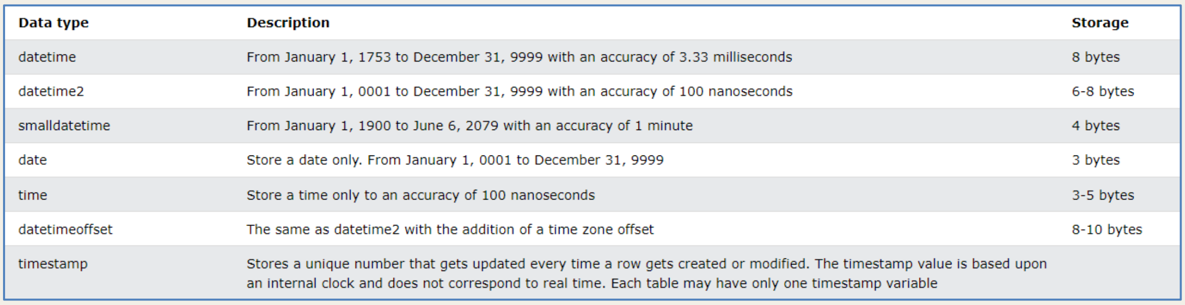


Data type	Description	Storage
bit	Integer that can be 0, 1, or NULL	
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
smallmoney	Monetary data from -214,748.3648 to 214,748.3647	4 bytes
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.	4 or 8 bytes
real	Floating precision number data from -3.40E + 38 to 3.40E + 38	4 bytes

Table 62 *– Numeric Data types (SQL Server) (**Source**: https://www.w3schools.com/sql/sql_datatypes.asp)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

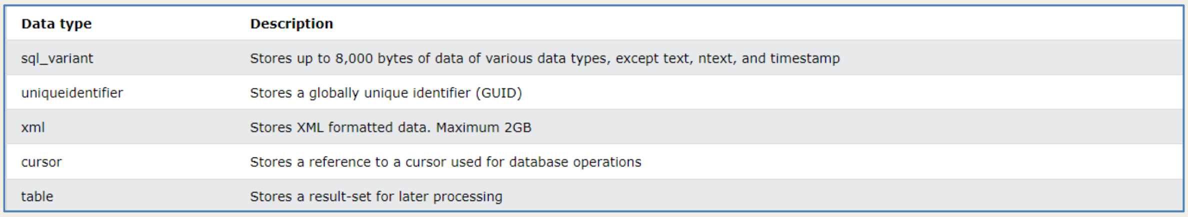
Date/Time Data Types



Data type	Description	Storage
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds	8 bytes
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds	6-8 bytes
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute	4 bytes
date	Store a date only. From January 1, 0001 to December 31, 9999	3 bytes
time	Store a time only to an accuracy of 100 nanoseconds	3-5 bytes
datetimeoffset	The same as datetime2 with the addition of a time zone offset	8-10 bytes
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable	

***Table 63** – Date/Time Data types (SQL Server) (**Source**: https://www.w3schools.com/sql/sql_datatypes.asp)*

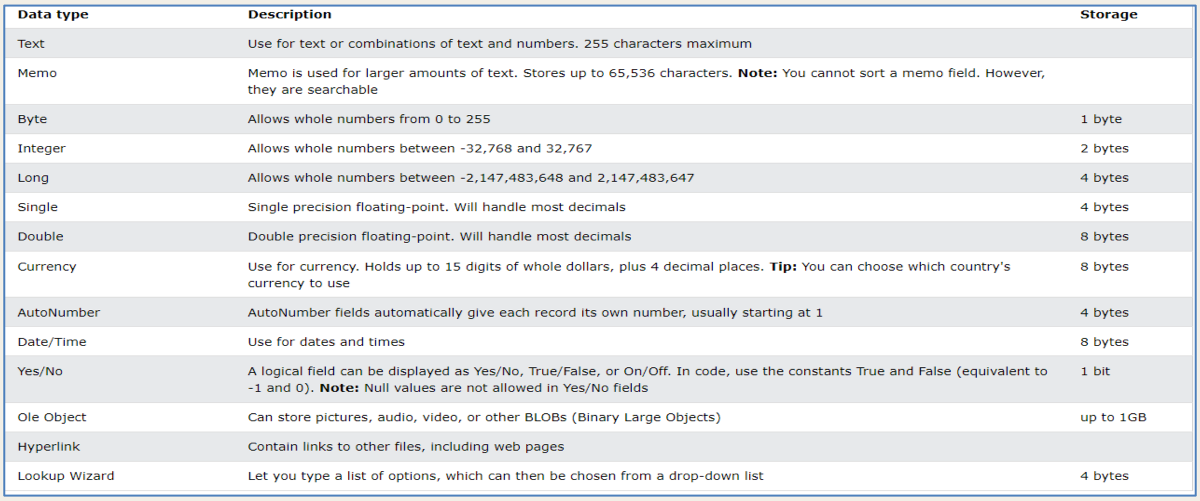
Other Data Types



Data type	Description
sql_variant	Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML formatted data. Maximum 2GB
cursor	Stores a reference to a cursor used for database operations
table	Stores a result-set for later processing

***Table 64** – Other Data types (SQL Server) (**Source**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Data Types in MS Access



Data type	Description	Storage
Text	Use for text or combinations of text and numbers. 255 characters maximum	
Memo	Memo is used for larger amounts of text. Stores up to 65,536 characters. **Note:** You cannot sort a memo field. However, they are searchable	
Byte	Allows whole numbers from 0 to 255	1 byte
Integer	Allows whole numbers between -32,768 and 32,767	2 bytes
Long	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
Single	Single precision floating-point. Will handle most decimals	4 bytes
Double	Double precision floating-point. Will handle most decimals	8 bytes
Currency	Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places. **Tip:** You can choose which country's currency to use	8 bytes
AutoNumber	AutoNumber fields automatically give each record its own number, usually starting at 1	4 bytes
Date/Time	Use for dates and times	8 bytes
Yes/No	A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to -1 and 0). **Note:** Null values are not allowed in Yes/No fields	1 bit
Ole Object	Can store pictures, audio, video, or other BLOBs (Binary Large Objects)	up to 1GB
Hyperlink	Contain links to other files, including web pages	
Lookup Wizard	Let you type a list of options, which can then be chosen from a drop-down list	4 bytes

***Table 65** – Data types in Access (**Source:** https://www.w3schools.com/sql/sql_datatypes.asp)*

4.3. SQL References

SQL Keywords

Keyword	Description
ADD	Adds a column in an existing table
ADD CONSTRAINT	Adds a constraint after a table is already created
ALL	Returns true if all of the subquery values meet the condition
ALTER	Adds, deletes, or modifies columns in a table, or changes the data type of a column in a table
ALTER COLUMN	Changes the data type of a column in a table
ALTER TABLE	Adds, deletes, or modifies columns in a table
AND	Only includes rows where both conditions is true
ANY	Returns true if any of the subquery values meet the condition

AS	Renames a column or table with an alias
ASC	Sorts the result set in ascending order
BACKUP DATABASE	Creates a backup of an existing database
BETWEEN	Selects values within a given range
CASE	Creates different outputs based on conditions
CHECK	A constraint that limits the value that can be placed in a column
COLUMN	Changes the data type of a column or deletes a column in a table
CONSTRAINT	Adds or deletes a constraint
CREATE	Creates a database, index, view, table, or procedure
CREATE DATABASE	Creates a new SQL database
CREATE INDEX	Creates an index on a table (allows duplicate values)

CREATE OR REPLACE VIEW	Updates a view
CREATE TABLE	Creates a new table in the database
CREATE PROCEDURE	Creates a stored procedure
CREATE UNIQUE INDEX	Creates a unique index on a table (no duplicate values)
CREATE VIEW	Creates a view based on the result set of a SELECT statement
DATABASE	Creates or deletes an SQL database
DEFAULT	A constraint that provides a default value for a column
DELETE	Deletes rows from a table
DESC	Sorts the result set in descending order
DISTINCT	Selects only distinct (different) values

DROP	Deletes a column, constraint, database, index, table, or view
DROP COLUMN	Deletes a column in a table
DROP CONSTRAINT	Deletes a UNIQUE, PRIMARY KEY, FOREIGN KEY, or CHECK constraint
DROP DATABASE	Deletes an existing SQL database
DROP DEFAULT	Deletes a DEFAULT constraint
DROP INDEX	Deletes an index in a table
DROP TABLE	Deletes an existing table in the database
DROP VIEW	Deletes a view
EXEC	Executes a stored procedure
EXISTS	Tests for the existence of any record in a subquery
FOREIGN KEY	A constraint that is a key used to link two tables together
FROM	Specifies which table to select or delete data from

FULL OUTER JOIN	Returns all rows when there is a match in either left table or right table
GROUP BY	Groups the result set (used with aggregate functions: COUNT, MAX, MIN, SUM, AVG)
HAVING	Used instead of WHERE with aggregate functions
IN	Allows you to specify multiple values in a WHERE clause
INDEX	Creates or deletes an index in a table
INNER JOIN	Returns rows that have matching values in both tables
INSERT INTO	Inserts new rows in a table
INSERT INTO SELECT	Copies data from one table into another table
IS NULL	Tests for empty values
IS NOT NULL	Tests for non-empty values
JOIN	Joins tables

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

LEFT JOIN	Returns all rows from the left table, and the matching rows from the right table
LIKE	Searches for a specified pattern in a column
LIMIT	Specifies the number of records to return in the result set
NOT	Only includes rows where a condition is not true
NOT NULL	A constraint that enforces a column to not accept NULL values
OR	Includes rows where either condition is true
ORDER BY	Sorts the result set in ascending or descending order
OUTER JOIN	Returns all rows when there is a match in either left table or right table
PRIMARY KEY	A constraint that uniquely identifies each record in a database table
PROCEDURE	A stored procedure

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

RIGHT JOIN	Returns all rows from the right table, and the matching rows from the left table
ROWNUM	Specifies the number of records to return in the result set
SELECT	Selects data from a database
SELECT DISTINCT	Selects only distinct (different) values
SELECT INTO	Copies data from one table into a new table
SELECT TOP	Specifies the number of records to return in the result set
SET	Specifies which columns and values that should be updated in a table
TABLE	Creates a table, or adds, deletes, or modifies columns in a table, or deletes a table or data inside a table
TOP	Specifies the number of records to return in the result set
TRUNCATE TABLE	Deletes the data inside a table, but not the table itself

UNION	Combines the result set of two or more SELECT statements (only distinct values)
UNION ALL	Combines the result set of two or more SELECT statements (allows duplicate values)
UNIQUE	A constraint that ensures that all values in a column are unique
UPDATE	Updates existing rows in a table
VALUES	Specifies the values of an INSERT INTO statement
VIEW	Creates, updates, or deletes a view

***Table 66** – SQL Keywords and Descriptions (**Source:** https://www.w3schools.com/sql/sql_ref_keywords.asp)*

MySQL Functions

For more details and a comprehensive list on specific functions used in MySQL, learners can refer to this link.

SQL Server Functions

For more details and a comprehensive list on specific functions used in SQL Server, learners can refer to this link.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

MS Access Functions

For more details and a comprehensive list on specific functions used in SQL Server, learners can refer to this link.

SQL Quick Ref

For a comprehensive list of SQL statements and their corresponding syntax, learners can refer to this link.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

4.4. SQL Examples

SQL Examples

There is comprehensive list of examples in the W3Schools website that learners can use to self-study and practice their SQL skills further.

SQL Quiz

For learners that want to assess their knowledge and skills on SQL, please refer to one of the following websites:

- W3Schools
- Tutorialspoint

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

5. JavaScript

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Topic Information

Topic:

5. JavaScript

Prerequisites:

To learn JavaScript, learners must know the basics of HTML and CSS. For a working knowledge of JavaScript and most web-based projects, this knowledge will be sufficient. For more advanced projects and skills, it is recommended to know basic OOP concepts and an OOP based programming language (like Java).

Workload:

15 hours.

Description:

JavaScript is a programming language that allows the developer to perform changes on the content of a web page in a dynamic way. JavaScript is among the most powerful and flexible programming languages of the web.

This JavaScript topic covers all the fundamental programming concepts, including data types, operators, creating and using variables, generating outputs, structuring the code to make decisions in programs or to loop over the same block of code multiple times, creating and manipulating strings and arrays, defining and calling functions, etc.

Once learners are comfortable with the basics, they will move on to next level that explains the idea of objects, the Document Object Model (DOM) and Browser Object Model (BOM), as well as how to make use of the native JavaScript objects like Date, Math, etc., and perform type conversions.

Moreover, some advanced concepts like event listeners, event propagation, borrowing methods from other objects, hoisting behavior of JavaScript, encoding and decoding JSON data, as well as detailed overview of new features introduced in ECMAScript 6 (or ES6) will be explored.

Learning outcomes:

Learners will understand how JavaScript allows the developer to change content dynamically and modify the HTML or CSS information on the client side, when the webpage is being shown to the user. Therefore, learners will study the structures of JavaScript code; will get to know how to change HTML/CSS code on the loading of the page and also will learn how to create a function and attach it to an event.

Material required:

- Computer or laptop
- Internet connection
- Text editor (online or offline): Sublime Text/Brackets/W3Schools online editor

Lesson Scenario:

The total time for this topic is 15 hours, and it will be up to the trainer/coach to decide how much time to dedicate to teaching each subtopic. In order to make the most of all the time available, we propose the use of the training materials produced by the project (PPT presentations), which were designed with an effective use of time in mind. These presentations are composed of the following elements:

- Development of the subtopic and main ideas to retain;
- Proposed Activities/Exercises.

That said, if the trainer/coach follows the logical sequence of the PPTs, he/she will certainly be able to complete the session within the stipulated time limit. These presentations can also be made available to learners for individual study.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Subtopics:

- 5.1. JavaScript Basic
- 5.2. JavaScript & DOM
- 5.3. JavaScript & BOM
- 5.4. JavaScript Advanced

Additional resources:

- JavaScript tutorial: w3schools
- Online Course on JavaScript: CodeAcademy

5.1. JavaScript Basic

JavaScript (JS) is the most widespread *client-side* scripting language (client-side scripting is associated to scripts running within a web browser). JS is intended to add interactivity and dynamic effects to the web pages by manipulating the content returned from a web server.

JavaScript was first developed as LiveScript by Netscape's computer programmer Brendan Eich in 1995. It was later renamed to JavaScript, and became an ECMA (European Computer Manufacturers Association) standard in 1997. Nowadays, JavaScript is the standard client-side scripting language for web-based applications, and it is supported by nearly all web browsers available (Chrome, Firefox, Safari, etc.).

JavaScript is an object-oriented language, and it also has some similarities in syntax to Java programming language, even though it is not related to Java at all.

JavaScript can be used for various purposes:

- Modify the content of a web page by adding or removing elements;
- Change the style and position of the elements on a web page;
- Monitor events like mouse click, hover, etc. and react to it;
- Make and control transitions and animations;
- Produce alert pop-ups to display info or warning messages to the user;
- Complete operations based on user inputs and display the results;
- Validate user inputs before submitting them to the server;
- And many other interesting purposes to be checked later.

Getting started with JavaScript

From this point, learners will understand how simple it can be to add interactivity to a web page by using JavaScript.

Characteristically, there are 3 ways of adding JS to a webpage:

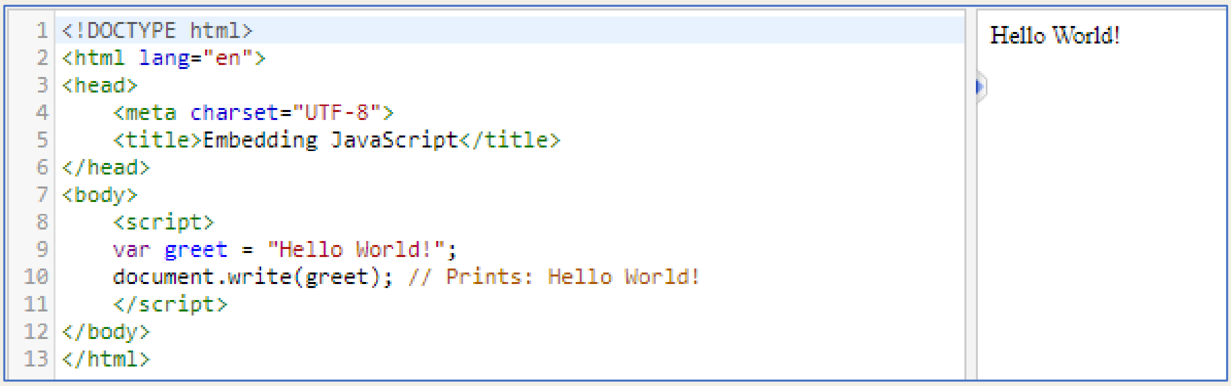
a) Embedding the JavaScript code between a <script> and a </script> tag;

b) Creating an external JavaScript file with the .js extension and then load it within the page through the src attribute of the <script> tag.

c) Placing the JavaScript code directly inside an HTML tag using the special tag attributes such as onclick, onmouseover, onkeypress, onload, etc.

a) Embedding the JavaScript code between a <script> and a </script> tag;

The JavaScript code can be embedded directly within a web page by placing it between the <script> and </script> tags. The <script> tag indicates the browser that the contained statements are to be interpreted as executable script and not HTML, as checked in the following example:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Embedding JavaScript</title>
</head>
<body>
    <script>
    var greet = "Hello World!";
    document.write(greet); // Prints: Hello World!
    </script>
</body>
</html>
```

```
Hello World!
```

Figure 1 – Embedding JS code between a <script> and a </script> tag (Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)

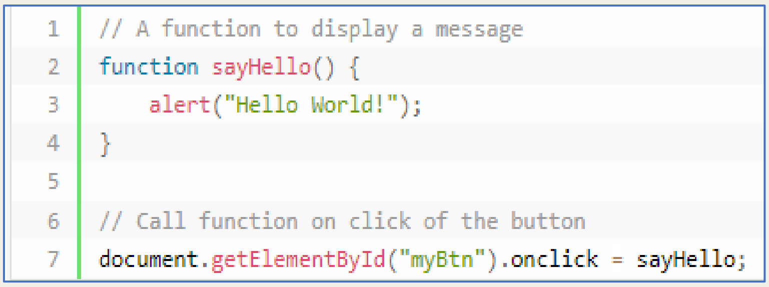
b) Creating an external JavaScript file with the .js extension and then load it within the page through the src attribute of the <script> tag.

A JavaScript code can be placed as well into a separate file with a .js extension, being then called in that file in the same document through the src attribute of the <script> tag, as follows:

```
<script src="js/hello.js"></script>
```

This is especially valuable if the programmer wants the same scripts available to multiple documents. Following this procedure, he/she will avoid repeating the same task over and over again, and it makes his/her website much simpler to maintain.

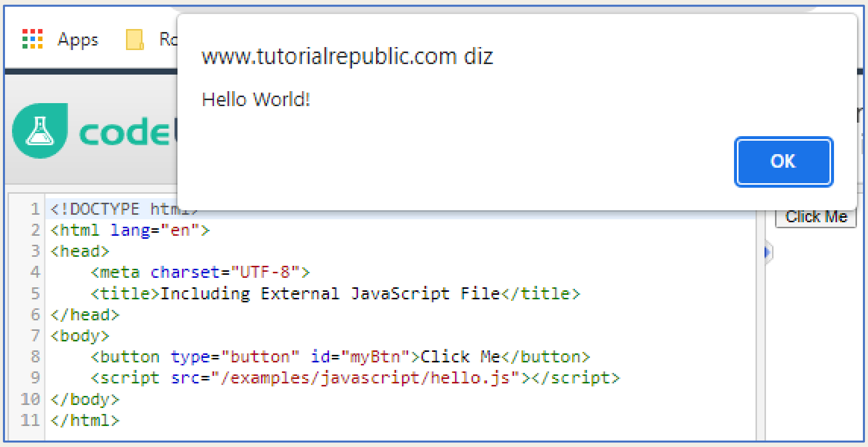
Forward to the statement above, a JavaScript file named "hello.js" will be created and the following code shall be inserted in it:



```
// A function to display a message
function sayHello() {
    alert("Hello World!");
}

// Call function on click of the button
document.getElementById("myBtn").onclick = sayHello;
```

***Figure 2** – Creating a file and calling a function on JS (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

At this point, the programmer may call the above JS file within a webpage by using the <script> tag, as seen in the image below:



www.tutorialrepublic.com diz

Hello World!

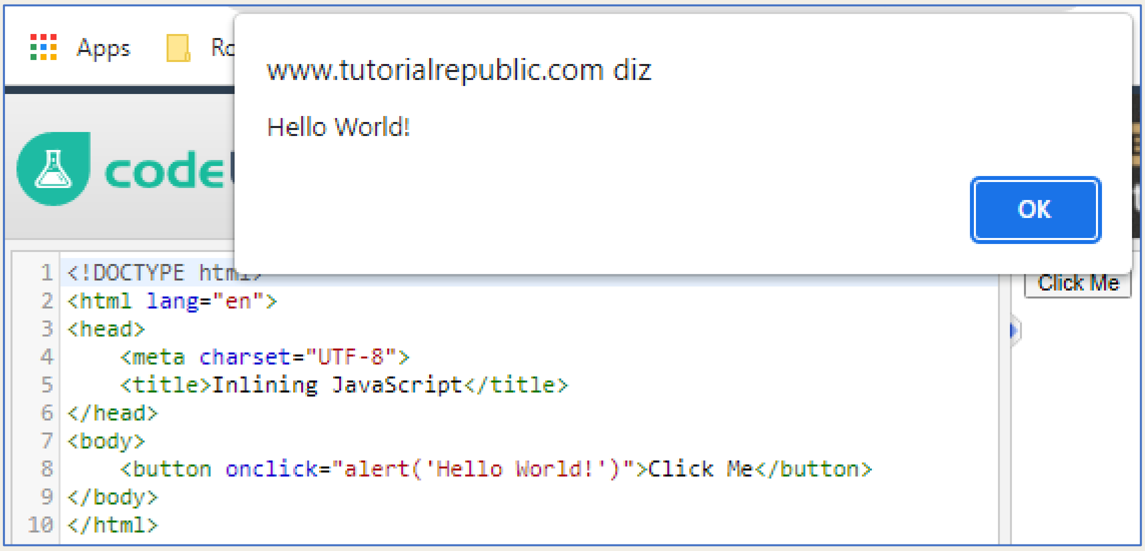
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Including External JavaScript File</title>
</head>
<body>
    <button type="button" id="myBtn">Click Me</button>
    <script src="/examples/javascript/hello.js"></script>
</body>
</html>
```

Figure 3 *– Creating a file and calling a function on JS (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

c) Placing the JavaScript code directly inside an HTML tag using the special tag attributes such as onclick, onmouseover, onkeypress, onload, etc.

JavaScript code may be introduced inline by inserting it directly inside the HTML tag by means of the special tag attributes such as onclick, onmouseover, onkeypress, onload, etc.

Nonetheless, it is not advisable to place large amount of JavaScript code inline as it disorders up HTML with JavaScript and makes JS code hard to maintain. *Figure 4* shows an example (in this case, an alert message is shown upon clicking on the button element):



www.tutorialrepublic.com diz

Hello World!

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Inlining JavaScript</title>
</head>
<body>
    <button onclick="alert('Hello World!')">Click Me</button>
</body>
</html>
```

Figure 4 *– Placing the JS code inline (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

The <script> element can be positioned in the <head> or <body> section of an HTML document. However, scripts should be preferably positioned at the end of the body section, before the closing </body> tag. This procedure will enable web pages to load faster, since it avoids obstruction of initial page rendering. Each <script> tag blocks the page rendering process until it has fully downloaded and executed the JavaScript code, so placing them in the head section (i.e. <head> element) of the document without any valid reason will significantly impact the performance of a website.

There are differences between Client-side and Server-side Scripting. Client-side scripting languages (e.g., JavaScript or VBScript are understood and executed by the web browser, in opposite of server-side scripting languages (e.g., PHP, ASP, Java, Python, Ruby, etc.), which run on the web server and their output is sent back to the web browser in HTML format.

Client-side scripting has many advantages comparing to traditional server-side scripting. For instance, JavaScript can be used to check if the user has entered invalid data in form fields and show notifications for input errors consequently in real-time before submitting the form to the web-server for final data validation and further processing in order to avoid needless network bandwidth usages and the misuse of server system resources.

Likewise, response from a server-side script is slower as compared to a client-side script, as server-side scripts are processed on a remote computer (not on a local one).

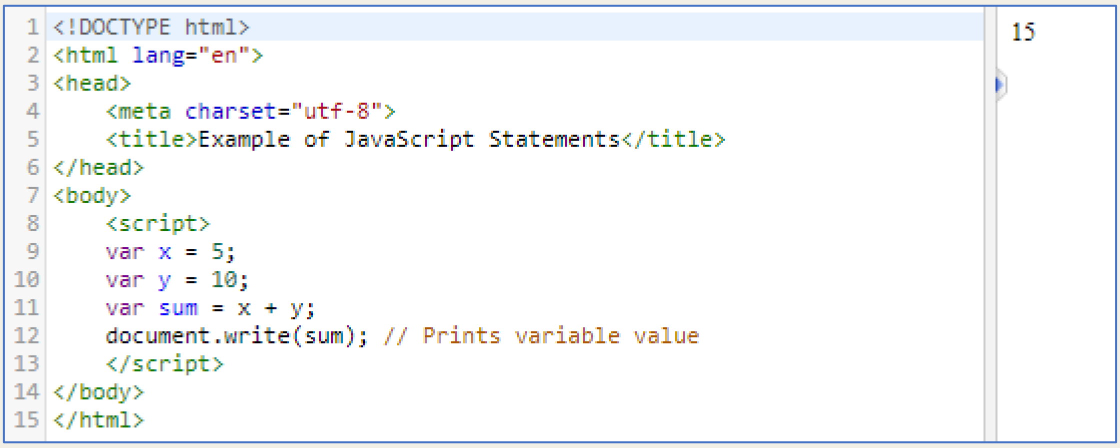
JavaScript Syntax

It is time to learn how to write the JS code.

Firstly, it is important to understand the JavaScript Syntax.

The syntax of JS is the **set of rules** that comprise a well-structured JavaScript programme. JS involves statements that are placed within the <script> </script> HTML tags in a web page, or within the external JavaScript file having .js extension.

Figure 5 below presents an example of a JS statement:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Example of JavaScript Statements</title>
</head>
<body>
    <script>
    var x = 5;
    var y = 10;
    var sum = x + y;
    document.write(sum); // Prints variable value
    </script>
</body>
</html>
```

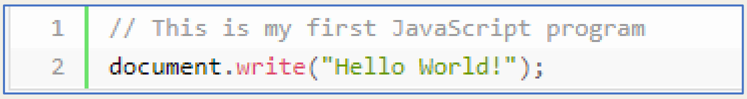
```
15
```

***Figure 5** – Example of a JavaScript statement (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Learners should state that JavaScript is **case-sensitive**. So, variables, language keywords, function names and other identifiers must be typed consistently in terms of letters capitalisation. For instance, the variable myVar must be typed this way (not “MYVAR”, “myvar”, etc.). This applies for **all cases**.

Commonly to the previous topics, JavaScript also provides the possibility of writing comments throughout the coding lines. Comments are inserted mainly because they provide extra information to the source code, but also because it can help programmers on understanding their codes after some time, teamworking, etc.

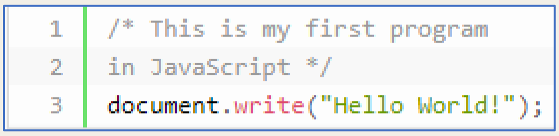
It is possible to add both single-line as well as multi-line comments on JavaScript. Single-line comments start with a double forward slash (//), followed by the comment text:



```
// This is my first JavaScript program
document.write("Hello World!");
```

Figure 6 *– Single-line comment on JavaScript (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

For a multi-line comment, a slash and an asterisk (/*) are the starting point, ending with an asterisk and a slash (*/):



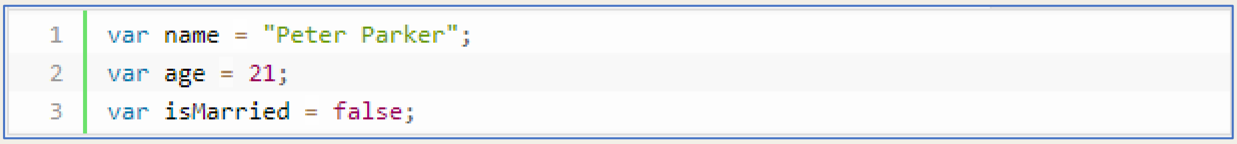
```
/* This is my first program
in JavaScript */
document.write("Hello World!");
```

Figure 7 *– Multi-line comment on JavaScript (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

JavaScript Variables

For storing data in JavaScript, programmers create variables. They are key for all the programming languages, and are used to store data, for instance by string of text, numbers, or other element(s). Whenever the programmer needs, he/she can set, update and retrieve data or value stored in the variables. Variables can be understood as symbolic names for values.

A variable can be created by using the var keyword, in which the assignment operator ("=") is used to allocate value to a variable, as follows: `var varName = value`



```
var name = "Peter Parker";
var age = 21;
var isMarried = false;
```

***Figure 8** – Creating a variable (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

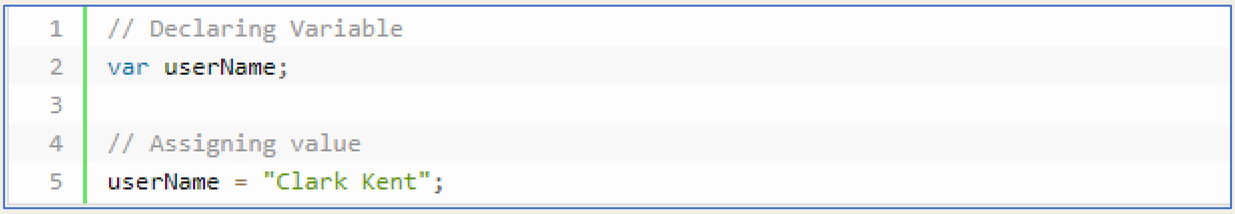
In this example, three variables have been created:

Number 1 – string value

Number 2 – number

Number 3 – Boolean value

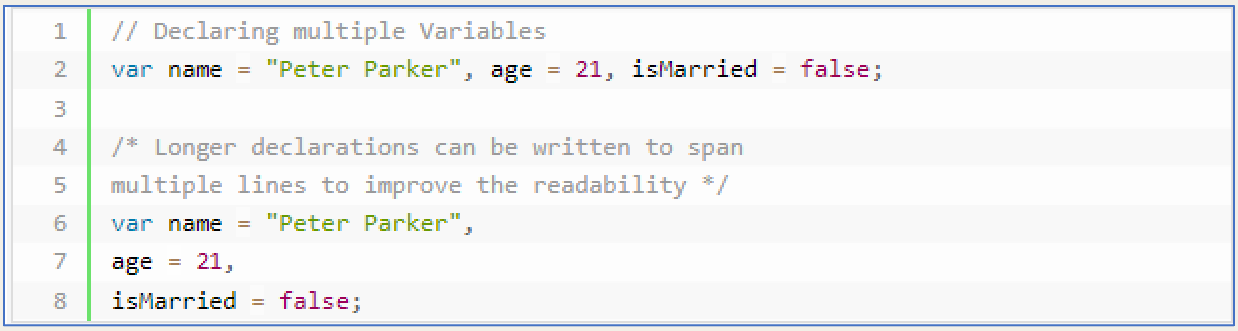
In JS, some variables are created with the purpose of holding values such as user inputs. That being said, they can be declared without having any initial values associated, as follows:



```
// Declaring Variable
var userName;

// Assigning value
userName = "Clark Kent";
```

***Figure 9** – Creating a variable without having any initial values assigned to them (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Moreover, the programmer may also declare multiple variables and, in a single statement, set their initial values. Each variable is detached by commas, as follows:

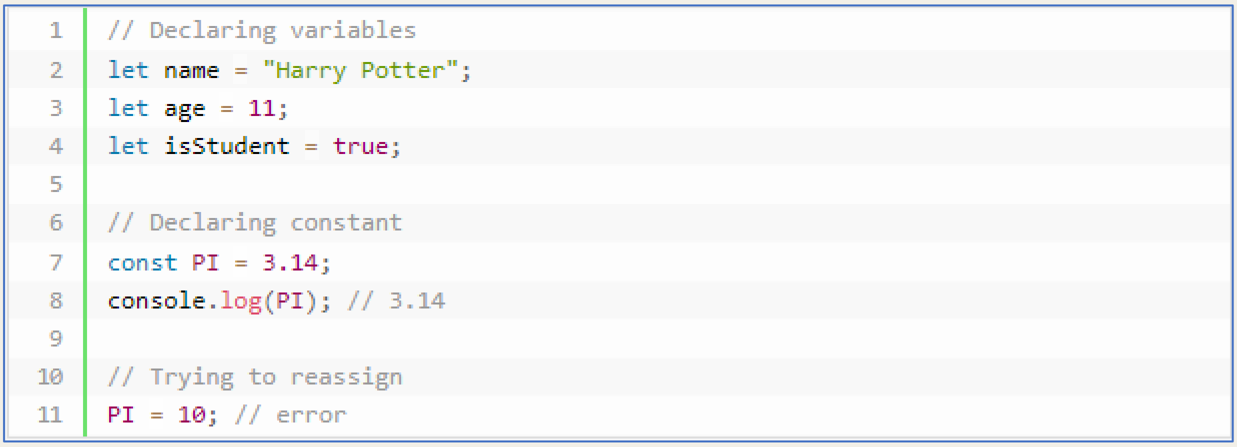


```
// Declaring multiple Variables
var name = "Peter Parker", age = 21, isMarried = false;

/* Longer declarations can be written to span
multiple lines to improve the readability */
var name = "Peter Parker",
age = 21,
isMarried = false;
```

Figure 10 *– Declaring multiple variables (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

The latest revision of JavaScript (ECMAScript 2015 or ES6) introduces two new keywords for declaring variables: let and const.

The const keyword works the same way as let. However, variables declared using const cannot be reassigned later in the code, as follows:



```
// Declaring variables
let name = "Harry Potter";
let age = 11;
let isStudent = true;

// Declaring constant
const PI = 3.14;
console.log(PI); // 3.14

// Trying to reassign
PI = 10; // error
```

Figure 11 *– Declaring multiple variables (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

In opposite to var keyword, which declare function-scoped variables, both let and const keywords declare variables, scoped at block-level ({}). Block scoping means that a new scope is created between a pair of curly brackets.

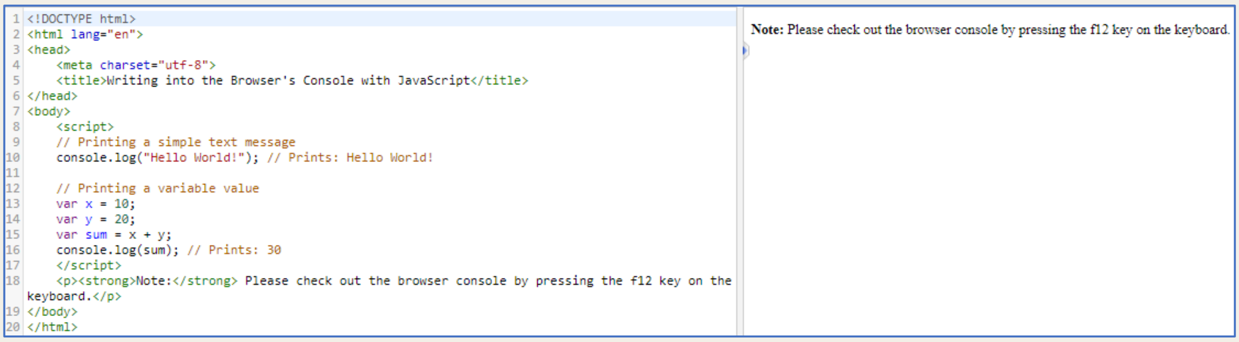
JavaScript variables have **specific rules** for being named:

- A variable name must start with a letter, underscore (_), or dollar sign ($).
- A variable name cannot start with a number.
- A variable name can only comprise alpha-numeric characters (A-Z, 0-9) and underscores.
- A variable name cannot comprehend spaces.
- A variable name cannot be a JavaScript keyword or a JavaScript reserved word.

JavaScript Generating Output

There are certain situations in which programmers may need to create outputs from the JS code, e.g., see the variable's value, write a message to browser console, etc. In JavaScript, there are some different ways of creating output including writing output to the browser window or browser console, displaying output in dialog boxes, writing output into an HTML element, etc.

It is not difficult to output a message or to write data to the browser's console (it can be accessed by clicking F12). For that purpose, the console.log() , a powerful yet simple method, should be applied.

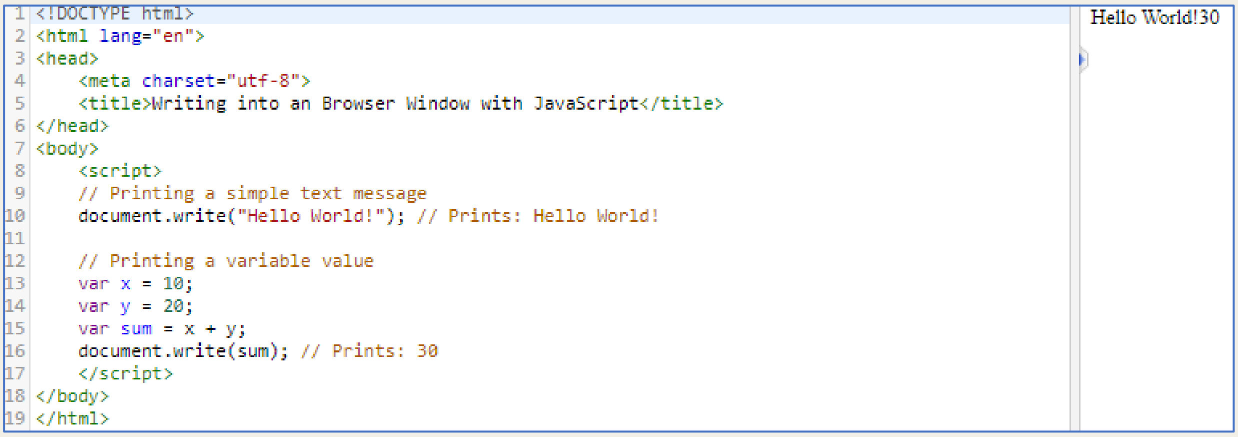


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Writing into the Browser's Console with JavaScript</title>
</head>
<body>
    <script>
    // Printing a simple text message
    console.log("Hello World!"); // Prints: Hello World!

    // Printing a variable value
    var x = 10;
    var y = 20;
    var sum = x + y;
    console.log(sum); // Prints: 30
    </script>
    <p><strong>Note:</strong> Please check out the browser console by pressing the f12 key on the keyboard.</p>
</body>
</html>
```

Note: Please check out the browser console by pressing the f12 key on the keyboard.

***Figure 12** – Generating ouputs by using console.log (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

To write the content to the current document only while that document is being deconstructed, the document.write() method can be used.



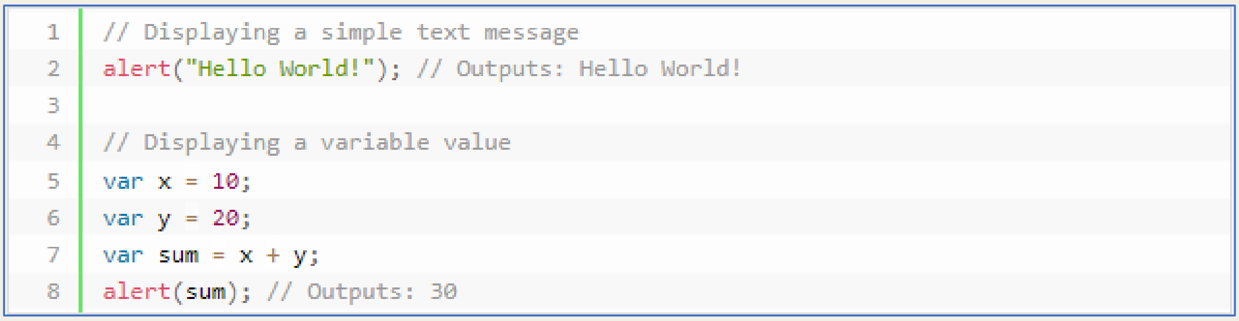
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Writing into an Browser Window with JavaScript</title>
</head>
<body>
    <script>
    // Printing a simple text message
    document.write("Hello World!"); // Prints: Hello World!

    // Printing a variable value
    var x = 10;
    var y = 20;
    var sum = x + y;
    document.write(sum); // Prints: 30
    </script>
</body>
</html>
```

```
Hello World!30
```

Figure 13 *– Generating ouputs by using console.log (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

If the document.write() method is used after the page has been loaded, it will overwrite all the existing content in that document, as follows in this link.

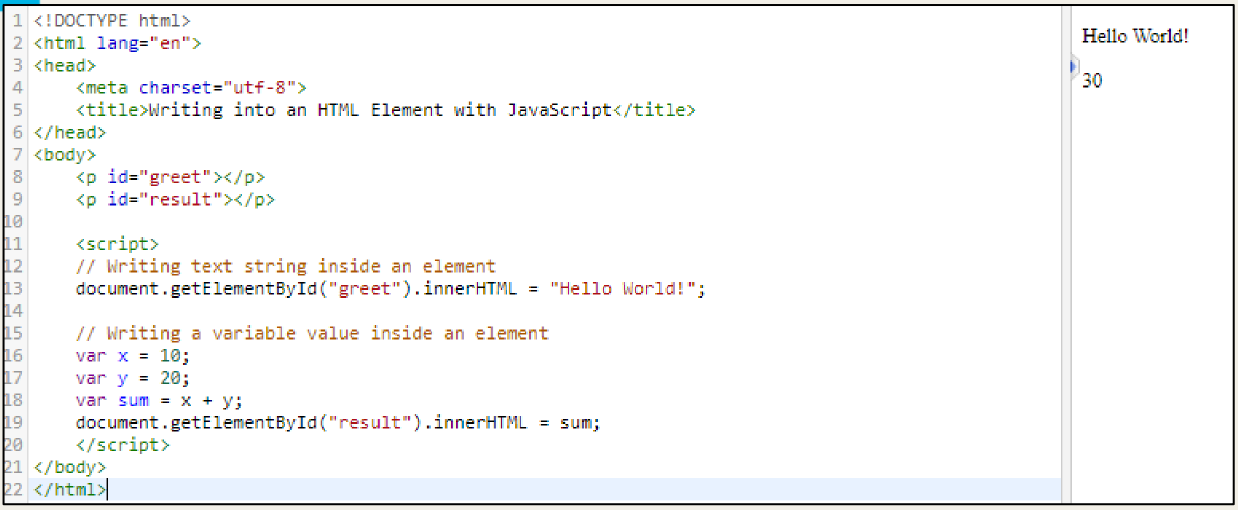
Alert dialog boxes can also be added to display the message or output data to the user. To create an alert dialog, the alert() method is used, as follows:



```
// Displaying a simple text message
alert("Hello World!"); // Outputs: Hello World!

// Displaying a variable value
var x = 10;
var y = 20;
var sum = x + y;
alert(sum); // Outputs: 30
```

Figure 14 *– Generating alert dialog boxes (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Outputs can be inserted or written inside an HTML element using the innerHTML property. Nonetheless, the programmer should select the element before writing the output, using the getElementById() method.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Writing into an HTML Element with JavaScript</title>
</head>
<body>
    <p id="greet"></p>
    <p id="result"></p>

    <script>
    // Writing text string inside an element
    document.getElementById("greet").innerHTML = "Hello World!";

    // Writing a variable value inside an element
    var x = 10;
    var y = 20;
    var sum = x + y;
    document.getElementById("result").innerHTML = sum;
    </script>
</body>
</html>
```

```
Hello World!

30
```

Figure 15 – Using the getElementByld () method (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)

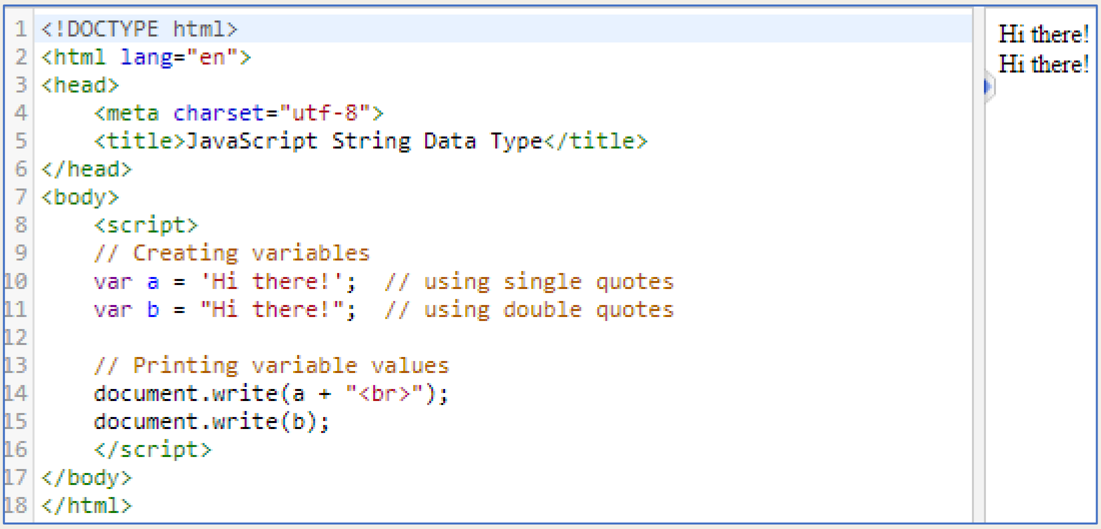
JavaScript Data Types

Data types essentially stipulate what kind of data can be stored and manipulated within a program. There are six basic data types in JS, which can be divided into three main categories:

- Primitive (or primary) – String, Number, and Boolean are examples of primitive data types, which can hold only one value at a time;
- Composite (or reference) – Object, Array, and Function (which are all types of objects) are composite data types. These can hold collections of values and more complex entities; and
- Special data types – Undefined and Null are special data types.

The String data type

It is used to embody textual data (for instance, sequences of characters). Strings are created using single or double quotes surrounding one or more characters:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript String Data Type</title>
</head>
<body>
    <script>
    // Creating variables
    var a = 'Hi there!';  // using single quotes
    var b = "Hi there!";  // using double quotes

    // Printing variable values
    document.write(a + "<br>");
    document.write(b);
    </script>
</body>
</html>
```

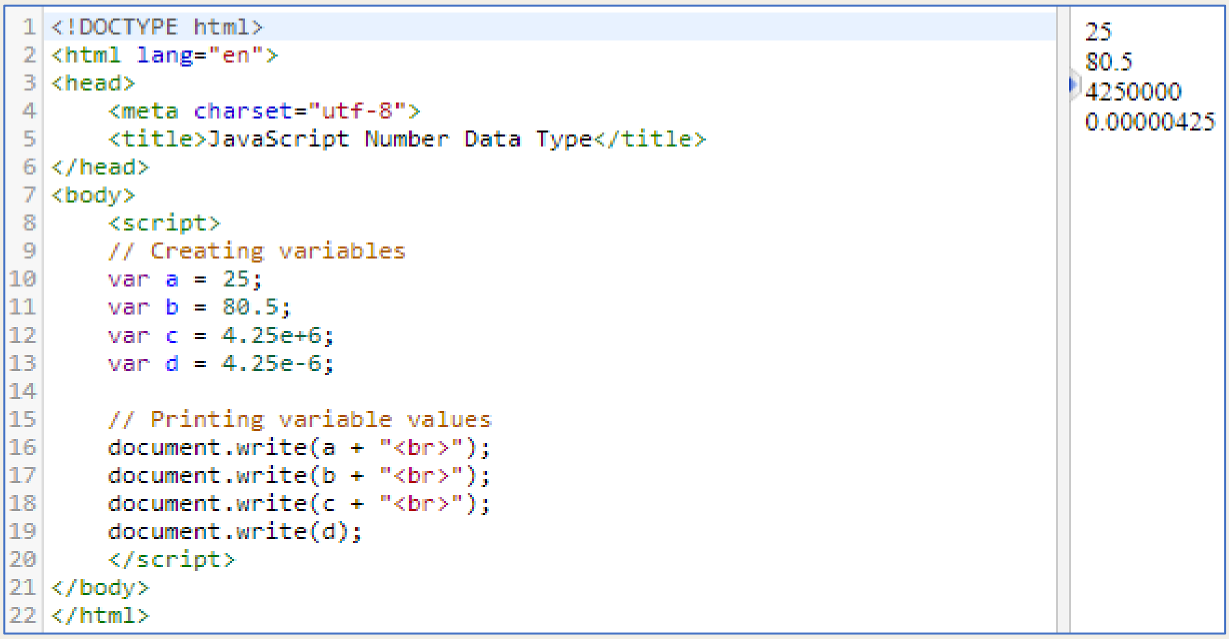
```
Hi there!
Hi there!
```

***Figure 16** – The String data type (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

It should be noted that quotes can be included inside the string, but they should not match the enclosing quotes, as follows on this example.

The Number Data Type

The number data type is useful for exhibiting positive or negative numbers with or without decimal place, or numbers written using exponential notation, for instance: 1.5e-4 (equivalent to 1.5x10^{-4}).

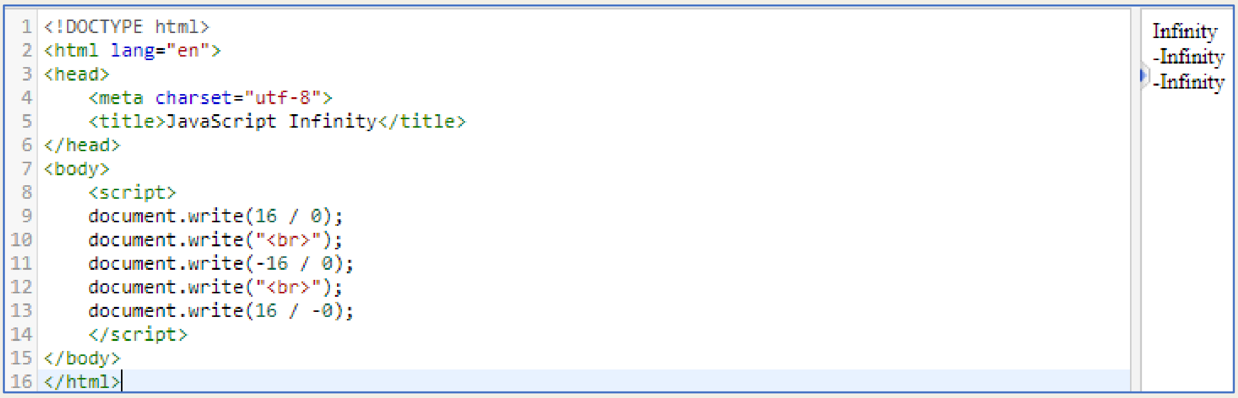


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Number Data Type</title>
</head>
<body>
    <script>
    // Creating variables
    var a = 25;
    var b = 80.5;
    var c = 4.25e+6;
    var d = 4.25e-6;

    // Printing variable values
    document.write(a + "<br>");
    document.write(b + "<br>");
    document.write(c + "<br>");
    document.write(d);
    </script>
</body>
</html>
```

```
25
80.5
4250000
0.00000425
```

Figure 17 *– The String data type (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

The Number Data Type also comprises some special values which are: Infinity, -Infinity and NaN. Infinity represents the mathematical infinity (∞), which is greater than any number. Infinity is the result of dividing a nonzero number by 0, as could be checked on *Figure 18*:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Infinity</title>
</head>
<body>
    <script>
    document.write(16 / 0);
    document.write("<br>");
    document.write(-16 / 0);
    document.write("<br>");
    document.write(16 / -0);
    </script>
</body>
</html>
```

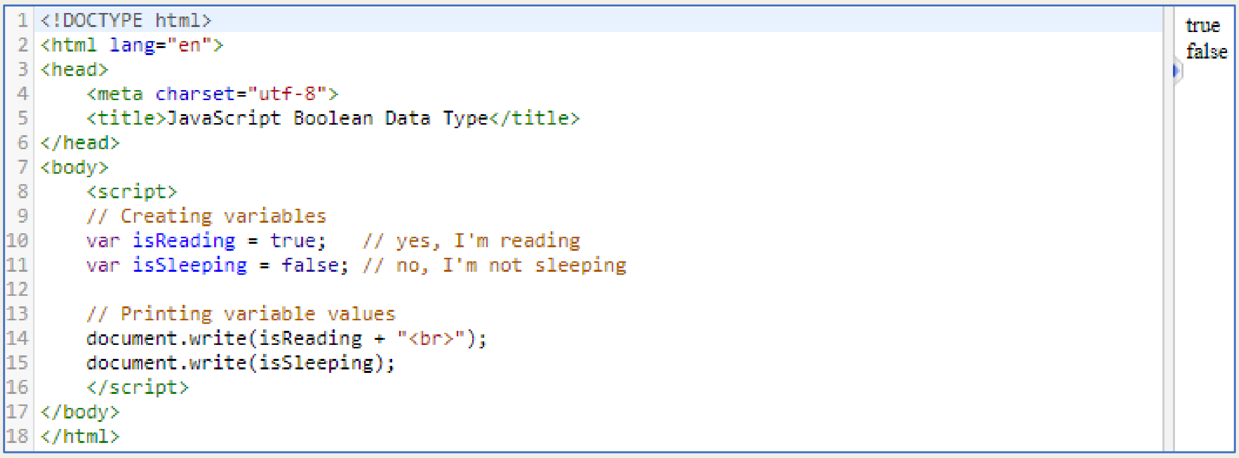
```
Infinity
-Infinity
-Infinity
```

Figure 18 *– The Number Data Type (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

However, NaN represents a special *Not-a-Number* value. It is a result of an invalid or an undefined mathematical operation, for example, taking the square root of -1 or dividing 0 by 0, etc.

The Boolean Data Type

Two values can be hold in this data type: true or false. It is classically used to stock values like yes (true) or no (false), on (true) or off (false), etc. as follows:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Boolean Data Type</title>
</head>
<body>
    <script>
    // Creating variables
    var isReading = true;   // yes, I'm reading
    var isSleeping = false; // no, I'm not sleeping

    // Printing variable values
    document.write(isReading + "<br>");
    document.write(isSleeping);
    </script>
</body>
</html>
```

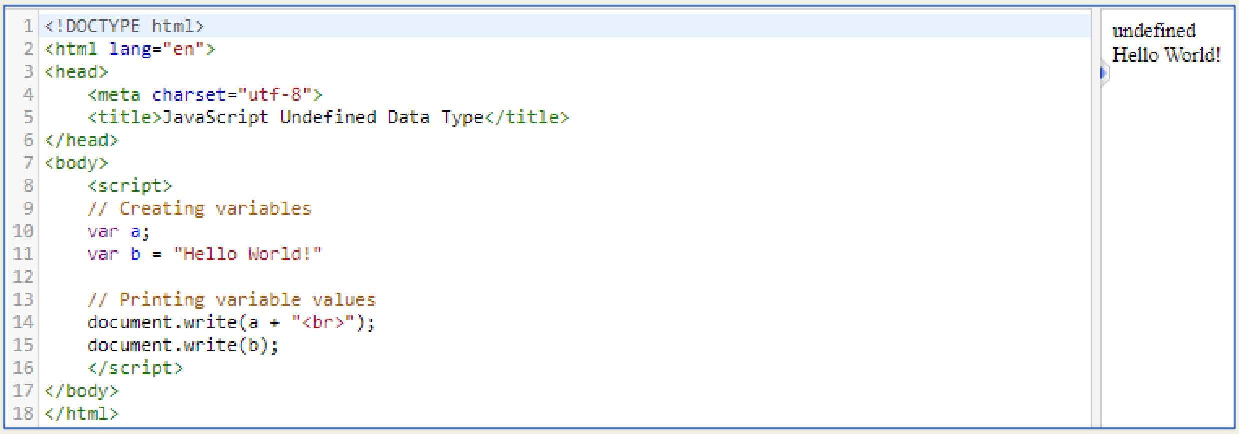
```
true
false
```

***Figure 19** – The Boolean Data Type (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Boolean values likewise come as an outcome of comparisons in a program. This example relates two variables and displays the result in an alert dialog box.

The Undefined Data Type

The undefined data type can only take one value – the special value undefined. If a variable has been declared, but has not been assigned a value, the value shall be declared undefined.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Undefined Data Type</title>
</head>
<body>
    <script>
    // Creating variables
    var a;
    var b = "Hello World!"

    // Printing variable values
    document.write(a + "<br>");
    document.write(b);
    </script>
</body>
</html>
```

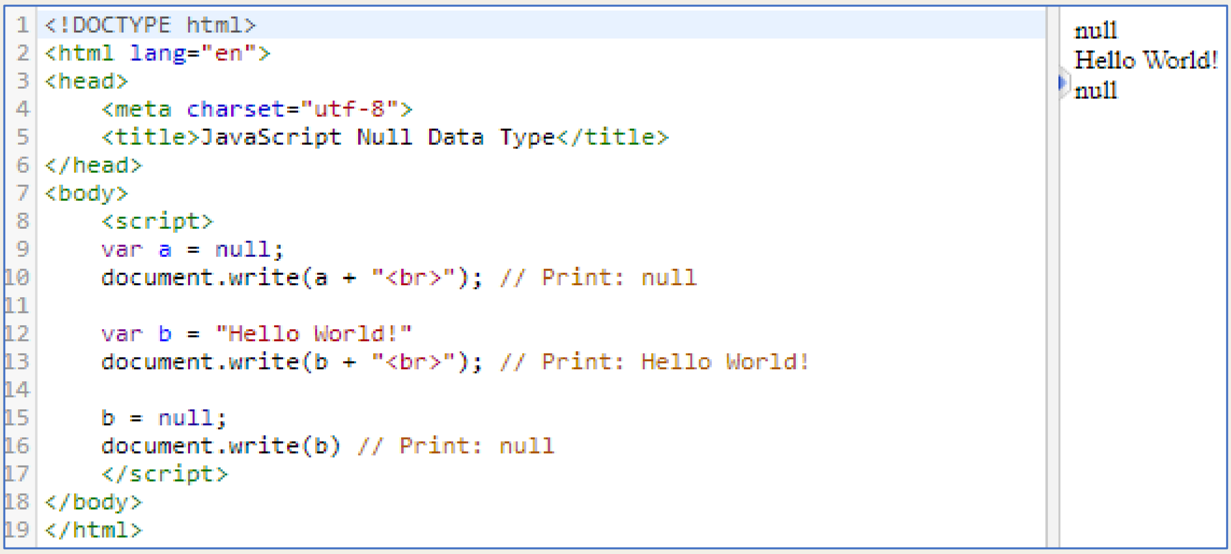
```
undefined
Hello World!
```

Figure 20 *– The Undefined Data Type (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

The Null Data Type

This is one more special data type that can have only one value – the null value. A null value means that simply there is no value. It is not equivalent to an empty string ("") or zero, it is purely nothing.

A variable can be clearly emptied of its current contents by assigning it the **null value**.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Null Data Type</title>
</head>
<body>
    <script>
    var a = null;
    document.write(a + "<br>"); // Print: null

    var b = "Hello World!"
    document.write(b + "<br>"); // Print: Hello World!

    b = null;
    document.write(b) // Print: null
    </script>
</body>
</html>
```

```
null
Hello World!
null
```

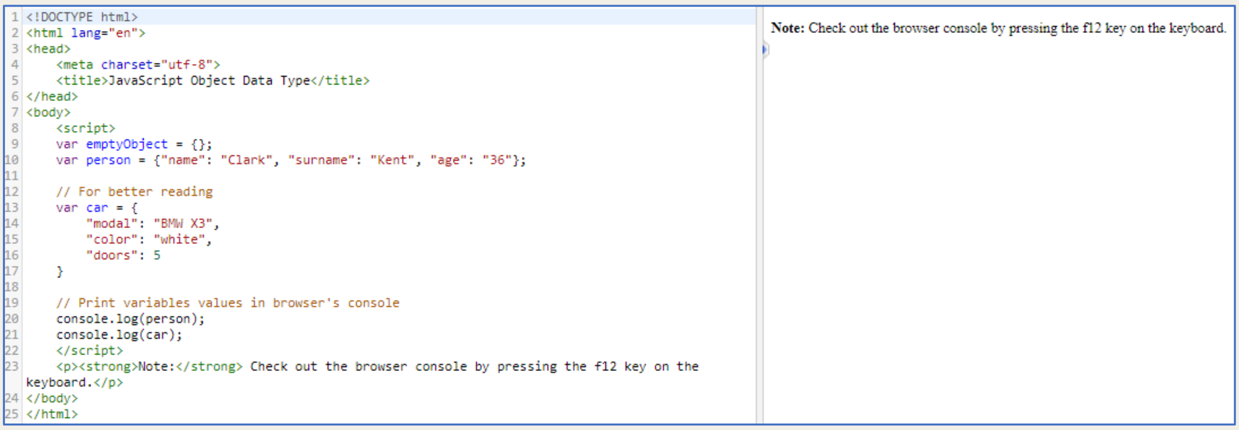
Figure 21 *– The Null Data Type (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

The Object Data Type

The object is a multifaceted data type that permits to store data collections.

An object contains properties, defined as a key-value pair. A property key (name) is always a string, but the value can be any data type (strings, numbers, booleans, or complex data types like arrays, function and other objects). The simplest way to create an object in JavaScript is shown below:

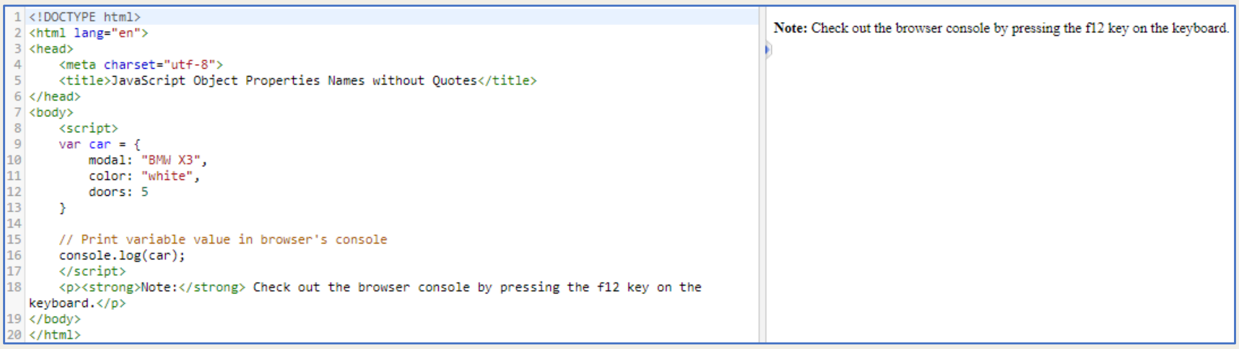


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Object Data Type</title>
</head>
<body>
    <script>
    var emptyObject = {};
    var person = {"name": "Clark", "surname": "Kent", "age": "36"};

    // For better reading
    var car = {
        "modal": "BMW X3",
        "color": "white",
        "doors": 5
    }

    // Print variables values in browser's console
    console.log(person);
    console.log(car);
    </script>
    <p><strong>Note:</strong> Check out the browser console by pressing the f12 key on the
keyboard.</p>
</body>
</html>
```

Note: Check out the browser console by pressing the f12 key on the keyboard.

__Figure 22__ – The simplest way for creating an object in JS (__Source__: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)

The quotes around property name may be omitted if the name is a valid JS name. This means quotes are required around "first-name" but are non-compulsory around firstname. Thus, the car object in *Figure 22* can also be written as follows:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Object Properties Names without Quotes</title>
</head>
<body>
    <script>
    var car = {
        modal: "BMW X3",
        color: "white",
        doors: 5
    }

    // Print variable value in browser's console
    console.log(car);
    </script>
    <p><strong>Note:</strong> Check out the browser console by pressing the f12 key on the
keyboard.</p>
</body>
</html>
```

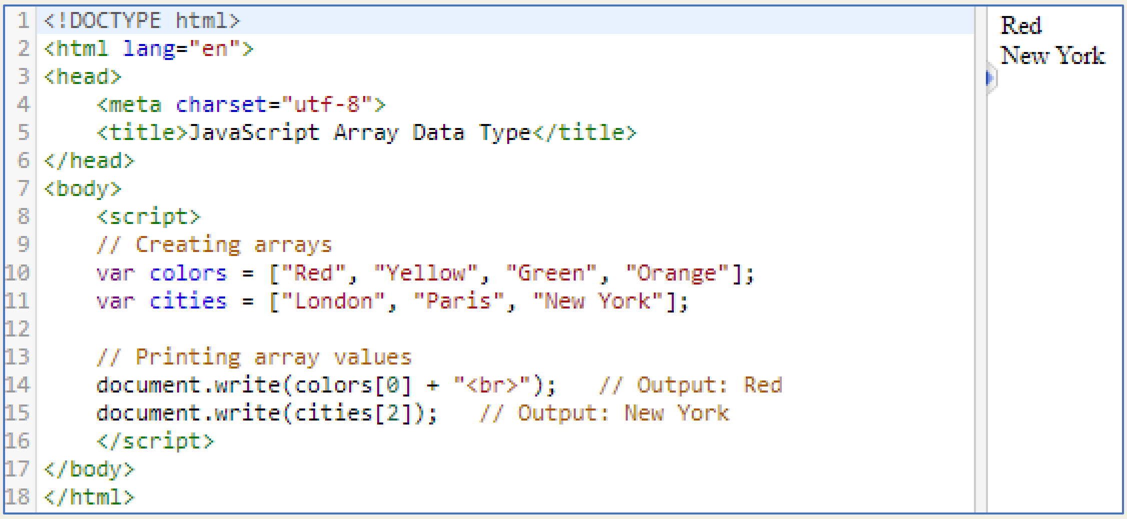
Note: Check out the browser console by pressing the f12 key on the keyboard.

__Figure 23__ – Rewriting the car object from Fig. 22 (__Source__: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)

The Array Data Type

An array is a type of object used for packing multiple values in single variable. Each value (also known as ‘element’) in an array has a numeric position, known as its index, and it may comprise data of any data type-numbers, strings, booleans, functions, objects, and even other arrays. The array index starts from 0 (zero), so that the first array element is arr[0] not arr[1].

The simplest way to make an array is by stipulating the array elements as a comma-separated list enclosed by square brackets, as follows:



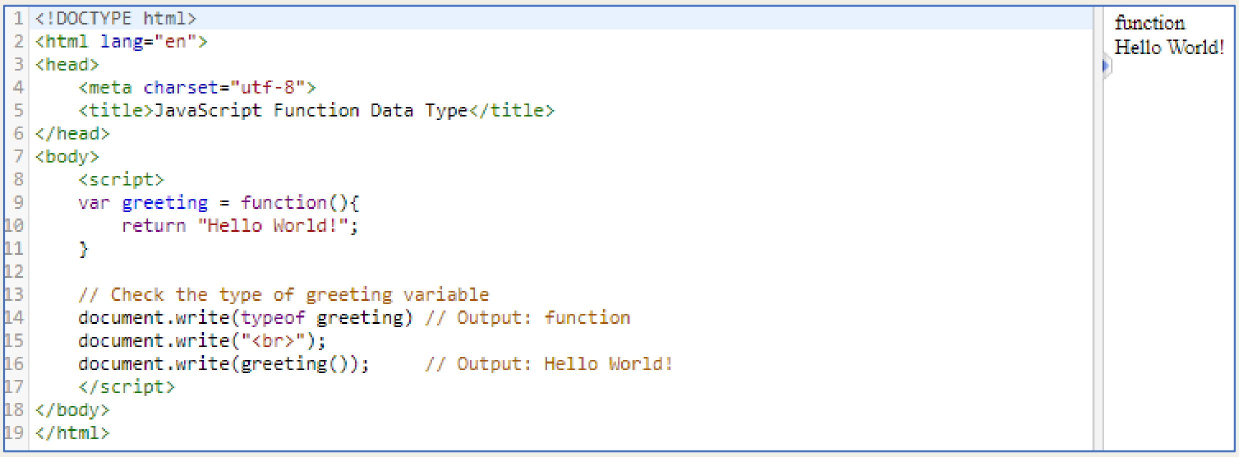
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Array Data Type</title>
</head>
<body>
    <script>
    // Creating arrays
    var colors = ["Red", "Yellow", "Green", "Orange"];
    var cities = ["London", "Paris", "New York"];

    // Printing array values
    document.write(colors[0] + "<br>");   // Output: Red
    document.write(cities[2]);   // Output: New York
    </script>
</body>
</html>
```

```
Red
New York
```

__Figure 24__ – Rewriting the car object from Fig. 22 (__Source__: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)

The Function Data Type

The function is a callable object that makes a block of code. Functions are objects, thus it is possible to assign them to variables, as follows:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Function Data Type</title>
</head>
<body>
    <script>
    var greeting = function(){
        return "Hello World!";
    }

    // Check the type of greeting variable
    document.write(typeof greeting) // Output: function
    document.write("<br>");
    document.write(greeting());     // Output: Hello World!
    </script>
</body>
</html>
```

```
function
Hello World!
```

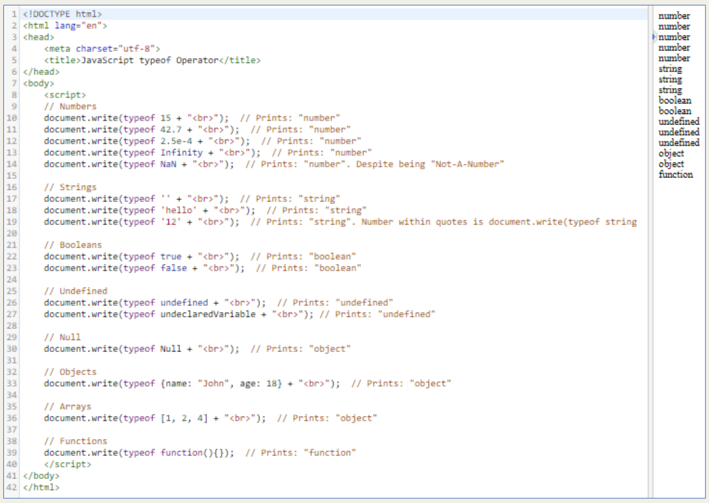
***Figure 25** – The Function Data Type (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Actually, functions can be used anywhere, and other values can be used. They can be stored in variables, objects, and arrays. Functions can be approved as arguments to other functions, and they can be returned from functions.

The typeof Operator

The typeof operator can be used to realize what type of data a variable covers. It can be used with or without parentheses (typeof(x) or typeof x).

The typeof operator is mostly beneficial to process the values of different types in a different way. However, the programmer should be cautious, as it may produce unforeseen results in some cases:



```
 1 <!DOCTYPE html>
 2 <html lang="en">
 3 <head>
 4     <meta charset="utf-8">
 5     <title>JavaScript typeof Operator</title>
 6 </head>
 7 <body>
 8     <script>
 9     // Numbers
10     document.write(typeof 15 + "<br>");  // Prints: "number"
11     document.write(typeof 42.7 + "<br>");  // Prints: "number"
12     document.write(typeof 2.5e-4 + "<br>");  // Prints: "number"
13     document.write(typeof Infinity + "<br>");  // Prints: "number"
14     document.write(typeof NaN + "<br>");  // Prints: "number". Despite being "Not-A-Number"
15 
16     // Strings
17     document.write(typeof '' + "<br>");  // Prints: "string"
18     document.write(typeof 'hello' + "<br>");  // Prints: "string"
19     document.write(typeof '12' + "<br>");  // Prints: "string". Number within quotes is document.write(typeof string
20 
21     // Booleans
22     document.write(typeof true + "<br>");  // Prints: "boolean"
23     document.write(typeof false + "<br>");  // Prints: "boolean"
24 
25     // Undefined
26     document.write(typeof undefined + "<br>");  // Prints: "undefined"
27     document.write(typeof undeclaredVariable + "<br>"); // Prints: "undefined"
28 
29     // Null
30     document.write(typeof Null + "<br>");  // Prints: "object"
31 
32     // Objects
33     document.write(typeof {name: "John", age: 18} + "<br>");  // Prints: "object"
34 
35     // Arrays
36     document.write(typeof [1, 2, 4] + "<br>");  // Prints: "object"
37 
38     // Functions
39     document.write(typeof function(){});  // Prints: "function"
40     </script>
41 </body>
42 </html>
```

```
number
number
number
number
number
string
string
string
boolean
boolean
undefined
undefined
undefined
object
object
function
```

***Figure 26** – The typeof Operator (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

As it is quite noticeable in the example from *Figure 26*, when the null value is tested using the typeof operator (line 22), "object" is returned instead of "null".

This is an enduring bug in JavaScript, but since lots of codes on the web are written around this behaviour, and since fixing it would generate many more problems, this issue was excluded by the committee that design and maintains JavaScript.

JavaScript Operators

Operators are symbols or keywords that inform the JavaScript engine to make a given action. For instance, the addition (+) symbol is an operator that tells JavaScript engine to add two variables or values, whereas the equal-to (==), greater-than (>) or less-than (<) symbols are the operators that tells JavaScript engine to compare two variables or values, etc.

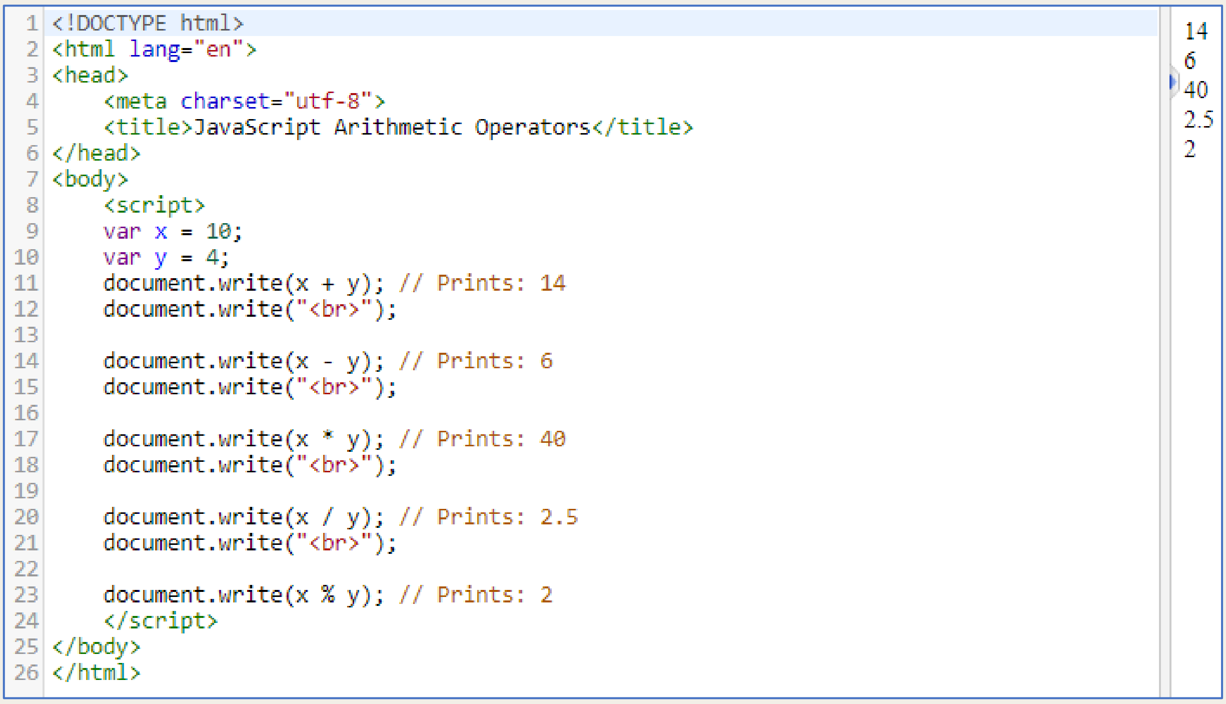
Among the different operators used in JavaScript, the first ones to be described are the JavaScript **Arithmetic Operators**. These are put in action in order to perform common arithmetical operations (additions, subtraction, multiplication, and so on). The complete list follows below:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Operator	Description	Example	Result
+	Addition	x + y	Sum of x and y
-	Subtraction	x - y	Difference of x and y.
*	Multiplication	x * y	Product of x and y.
/	Division	x / y	Quotient of x and y
%	Modulus	x % y	Remainder of x divided by y

***Table 1** – Arithmetic Operators (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Here is a practical example on how the aforementioned operators can be used:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Arithmetic Operators</title>
</head>
<body>
    <script>
    var x = 10;
    var y = 4;
    document.write(x + y); // Prints: 14
    document.write("<br>");

    document.write(x - y); // Prints: 6
    document.write("<br>");

    document.write(x * y); // Prints: 40
    document.write("<br>");

    document.write(x / y); // Prints: 2.5
    document.write("<br>");

    document.write(x % y); // Prints: 2
    </script>
</body>
</html>
```

```
14
6
40
2.5
2
```

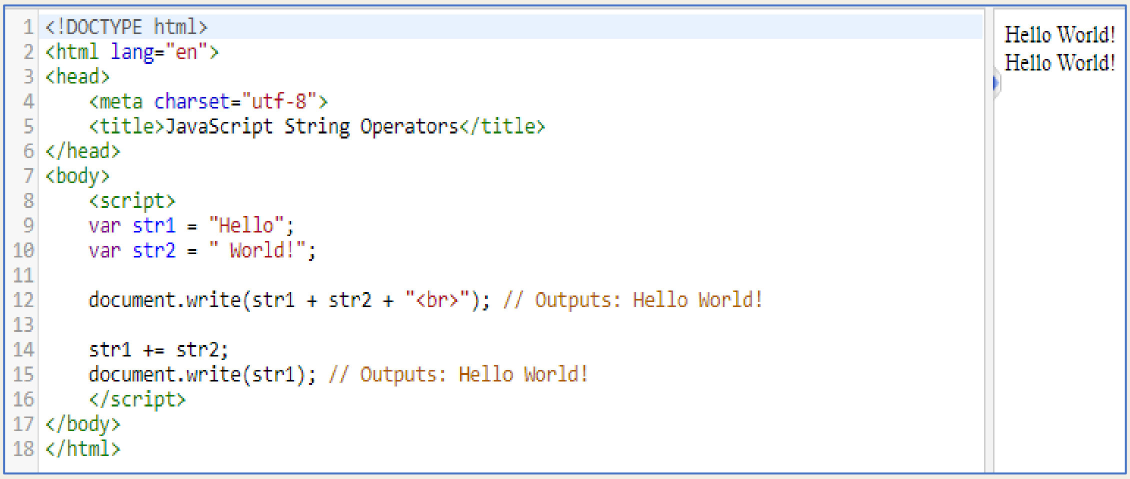
***Figure 27** – The JS Arithmetic Operators (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Then, there are as well **JS String Operators** (two of them):

Operator	Description	Example	Result
+	Concatenation	str1 + str2	Concatenation of str1 and str2
+=	Concatenation assignment	str1 += str2	Appends the str2 to the str1

***Table 2** – String Operators (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

A practical example:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript String Operators</title>
</head>
<body>
    <script>
    var str1 = "Hello";
    var str2 = " World!";

    document.write(str1 + str2 + "<br>"); // Outputs: Hello World!

    str1 += str2;
    document.write(str1); // Outputs: Hello World!
    </script>
</body>
</html>
```

```
Hello World!
Hello World!
```

***Figure 28** – The JS Arithmetic Operators (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

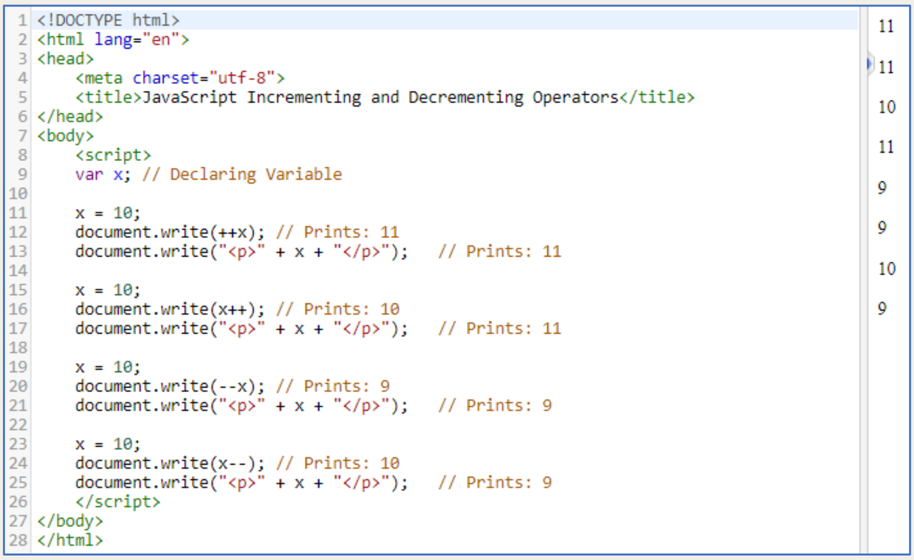
Concerning the **JS Incrementing and Decrementing Operators**, they are used to increment/decrement a variable's value.

Operator	Name	Effect
++x	Pre-increment	Increments x by one, then returns x
x++	Post-increment	Returns x, then increments x by one
--x	Pre-decrement	Decrements x by one, then returns x

x--	Post-decrement	Returns x, then decrements x by one

Table 3 – *Incrementing and Decrementing Operators (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

A real-world example:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Incrementing and Decrementing Operators</title>
</head>
<body>
    <script>
    var x; // Declaring Variable

    x = 10;
    document.write(++x); // Prints: 11
    document.write("<p>" + x + "</p>");   // Prints: 11

    x = 10;
    document.write(x++); // Prints: 10
    document.write("<p>" + x + "</p>");   // Prints: 11

    x = 10;
    document.write(--x); // Prints: 9
    document.write("<p>" + x + "</p>");   // Prints: 9

    x = 10;
    document.write(x--); // Prints: 10
    document.write("<p>" + x + "</p>");   // Prints: 9
    </script>
</body>
</html>
```

```
11

11

10

11

9

9

10

9
```

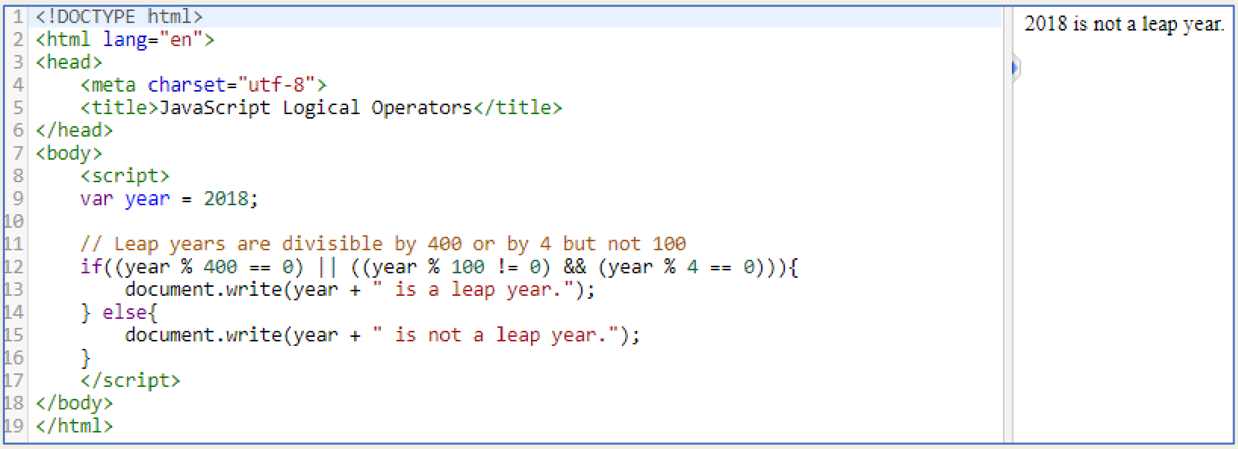
Figure 29 – *JS Incrementing and Decrementing Operators (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

To combine conditional statements, **JS Logical Operators** are used.

Operator	Name	Example	Result
&&	And	x && y	True if both x and y are true
||	Or	x || y	True if either x or y is true
!	Not	!x	True if x is not true

Table 4 – *Logical Operators (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Here is a practical example:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Logical Operators</title>
</head>
<body>
    <script>
    var year = 2018;

    // Leap years are divisible by 400 or by 4 but not 100
    if((year % 400 == 0) || ((year % 100 != 0) && (year % 4 == 0))){
        document.write(year + " is a leap year.");
    } else{
        document.write(year + " is not a leap year.");
    }
    </script>
</body>
</html>
```

```
2018 is not a leap year.
```

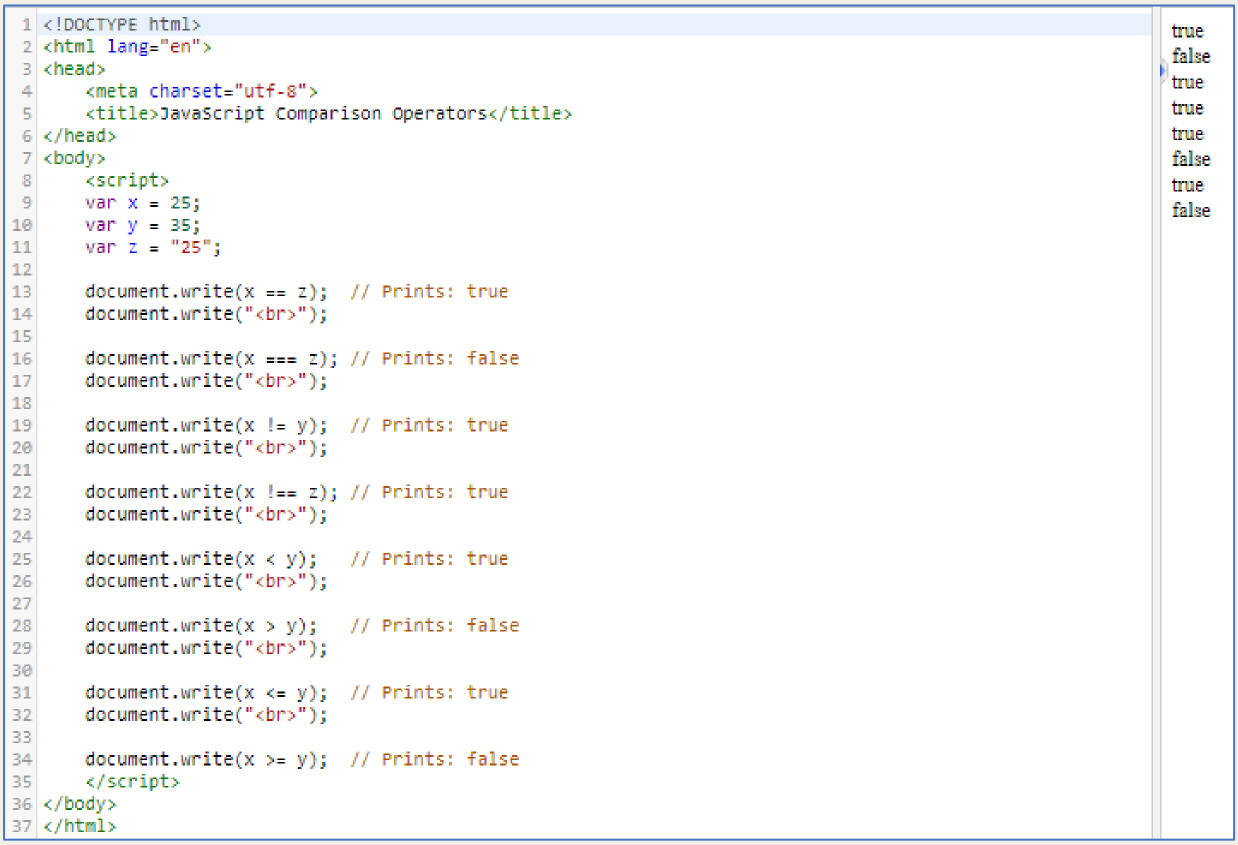
***Figure 30** – JS Incrementing and Decrementing Operators (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Regarding **JS Comparison Operators**, programmers use them to compare two values in a Boolean Fashion.

Operator	Name	Example	Result
==	Equal	x == y	True if x is equal to y
===	Identical	x === y	True if x is equal to y, and they are of the same type
!=	Not equal	x != y	True if x is not equal to y
!==	Not identical	x !== y	True if x is not equal to y, or they are not of the same type
<	Less than	x < y	True if x is less than y
>	Greater than	x > y	True if x is greater than y
>=	Greater than or equal to	x >= y	True if x is greater than or equal to y
<=	Less than or equal to	x <= y	True if x is less than or equal to y

__Table 5__ – Comparison Operators (__Source__: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)

How they work:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Comparison Operators</title>
</head>
<body>
    <script>
    var x = 25;
    var y = 35;
    var z = "25";

    document.write(x == z);  // Prints: true
    document.write("<br>");

    document.write(x === z); // Prints: false
    document.write("<br>");

    document.write(x != y);  // Prints: true
    document.write("<br>");

    document.write(x !== z); // Prints: true
    document.write("<br>");

    document.write(x < y);   // Prints: true
    document.write("<br>");

    document.write(x > y);   // Prints: false
    document.write("<br>");

    document.write(x <= y);  // Prints: true
    document.write("<br>");

    document.write(x >= y);  // Prints: false
    </script>
</body>
</html>
```

```
true
false
true
true
true
false
true
false
```

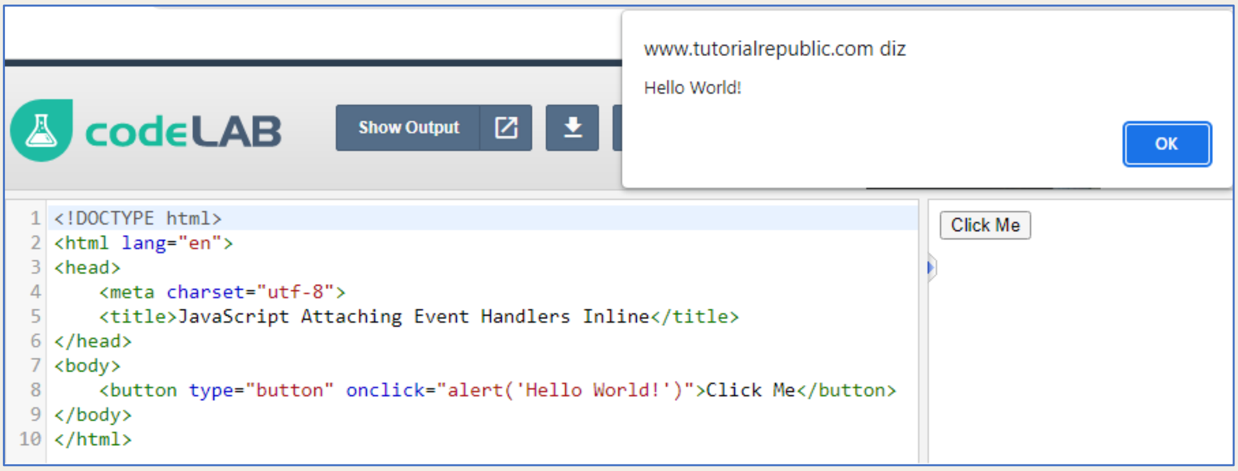
__Figure 31__ – JS Incrementing and Decrementing Operators (__Source__: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)

JavaScript Events

Before going into deep on this section, it is important to acknowledge what an event is in this context. An event is something that occurs each time users interact with the web page, such as when a link or a button is clicked, text is entered into an input box or textarea, a selection is made in a select box, key is pressed on the keyboard, the mouse pointer is moved, a form is submitted, and so on. Every so often, the browser is able to trigger the events itself, for example when loading a page.

When an event happens, programmers can use a JavaScript event handler (or listener) to spot them and do specific task/s. By convention, the names for event handlers always start with the word "on", so an event handler for the click event is named onclick, likewise an event handler for the load event is named onload, event handler for the blur event is named onblur, etc.

There are many ways of assigning an event handler. The simplest way is to add them directly to the start tag of the HTML elements, by means of the special event-handler attributes. E.g., to assign a click handler for a button element, the onclick attribute may be used, as follows:



www.tutorialrepublic.com diz

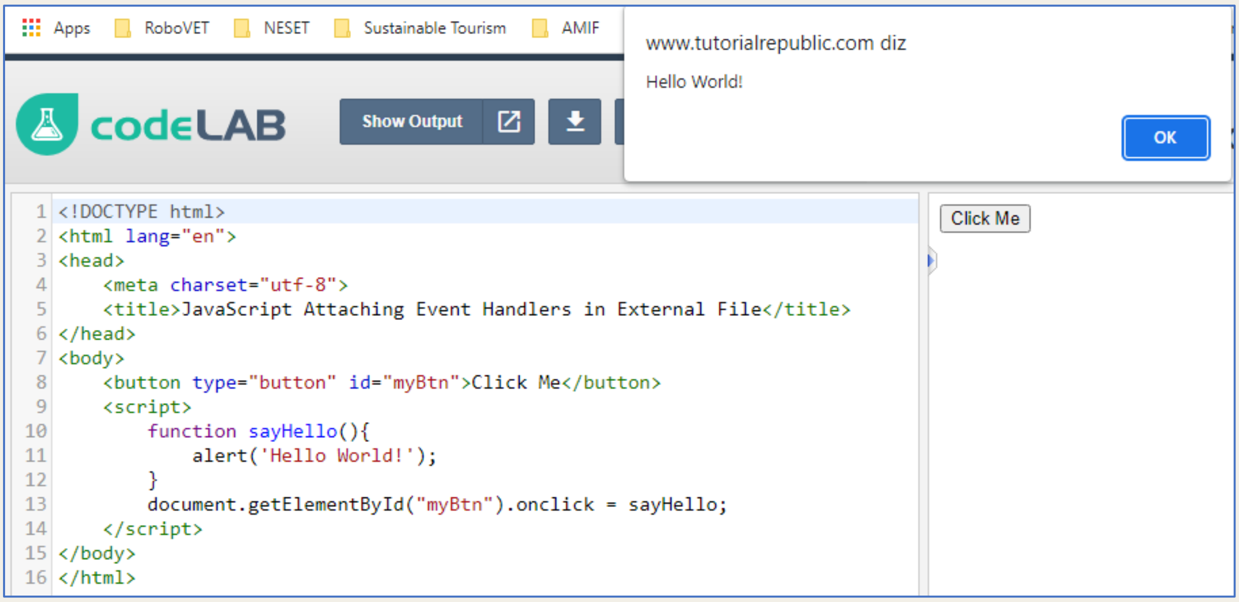
Hello World!

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Attaching Event Handlers Inline</title>
</head>
<body>
    <button type="button" onclick="alert('Hello World!')">Click Me</button>
</body>
</html>
```

Figure 32 *– JS Incrementing and Decrementing Operators (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Nonetheless, to keep the JavaScript detached from HTML, programmers can set up the event handler in an external JavaScript file or within the <script> and </script> tags, as follows:



www.tutorialrepublic.com diz

Hello World!

OK

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Attaching Event Handlers in External File</title>
</head>
<body>
    <button type="button" id="myBtn">Click Me</button>
    <script>
        function sayHello(){
            alert('Hello World!');
        }
        document.getElementById("myBtn").onclick = sayHello;
    </script>
</body>
</html>
```

***Figure 33** – Assigning an event handler (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Generally, events can be categorised into four main groups — **mouse events, keyboard events, form events and document/window events**. There are many other events, which will be exploited more furtherly. The most useful events will be overviewed below:

- **Mouse Events**

A mouse event is activated when the user clicks some element, moves the mouse pointer over an element, and so on. Some important mouse events and their event handlers are as follows:

  - **The Click Event (onclick):** The click event happens when a user clicks on an element on a web page. Habitually, these are form elements and links. A click event can be handled with an onclick event handler.

The following example presents the case of an alert message popping up when the user clicks on the elements.

- **The Contextmenu Event (oncontextmenu):** it occurs when users click the mouse's right button on an element, opening a context menu. Oncontextmenu event handler handles a contextmenu event. This example pops an alert message when users right-click the elements.
- **The Mouseover Event (onmouseover):** it happens when users move the mouse pointer over an element. It can be handled with the onmouseover event handler. The following example shows an alert message when the mouse is placed over the elements.
- **The Mouseout Event (onmouseout):** it takes place when users move the mouse pointer outside of an element. It can be handled by using the onmouseout event handler. The following example will show you an alert message when the mouseout event occurs.

Keyboard Events

A keyboard event takes place when the user presses or releases a key on the keyboard. Some of the most important keyboard events and their event handlers are as follows:

- **The Keydown Event (onkeydown):** it happens when users press down a key on the keyboard. It can be handled by using the onkeydown event handler. This example shows an alert message each time a keydown event occurs.
- **The Keyup Event (onkeyup):** it occurs when users release a key on the keyboard. Handled with onkeyup event handler. This example shows an alert message when it occurs.
- **The Keypress Event (onkeypress):** it happens when a user presses down a key on the keyboard that has a character value linked to it. E.g., Ctrl, Shift, Alt, Esc, Arrow keys, etc. will not generate a keypress event, but will generate a keydown and keyup event. The onkeypress event handler handles the keypress event. It can be checked here through an alert message.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- **Form Events**

A form event is triggered as soon as a form control receives or loses focus or when the user modifies a form control value (e.g., by typing text in a text input), select an option in a select box, etc.

- **The Focus Event (onfocus):** it happens when users give focus to an element on a webpage. It can be handled with onfocus event handler. This example will highlight the background of text input in yellow colour when it obtains the focus.
- **The Blur Event (onblur):** it happens when the user takes the focus away from a window or form element. It can be handled with the onblur event handler. The following example will exhibit an alert message as soon as the text input element misses focus.
- **The Change Event (onchange):** it occurs as soon as a user modificates the value of a form element. Event handler: onchange. This example shows an alert message when the option in the select box is changed.
- **The Submit Event (onsubmit):** it only happens when the user submits a form on a webpage. Event handler: onsubmit. This example shows an alert message while submitting the form.

- **Document/Window Events**

Situations in which the page has loaded or resized the browser window can as well trigger events.

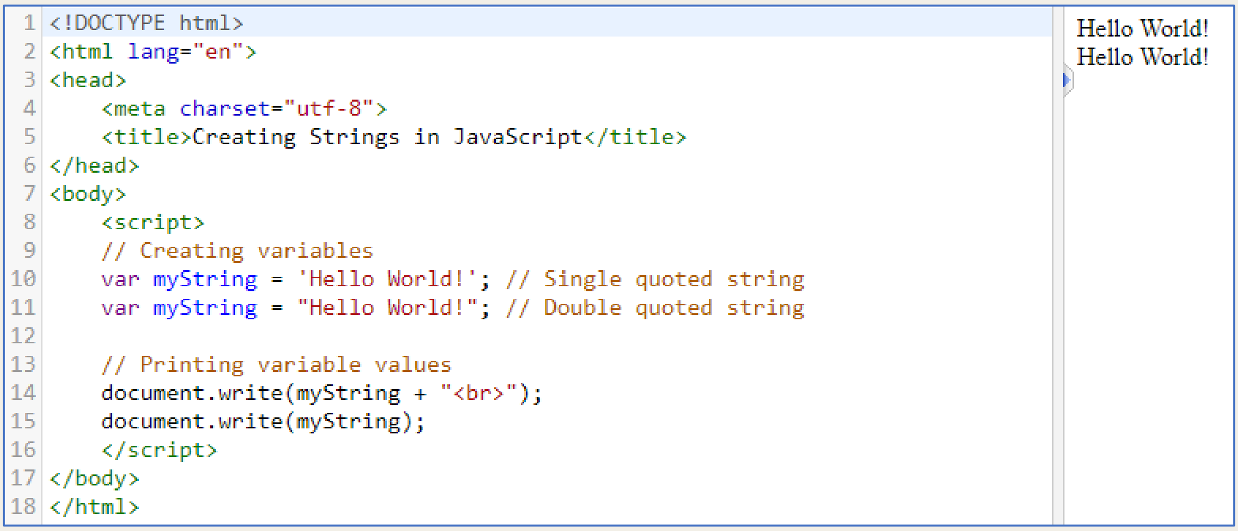
- **The Load Event (onload):** it happens when a webpage fully loads in a web browser. Event handler: onload. The following example will emit an alert message as soon as page loads.
- **The Unload Event (onunload):** when the user leaves the present webpage, this event takes place. Event handler: onunload. This example shows an alert pop-up when the user tries to leave the page.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- **The Resize Event (onresize):** it happens when the Internet user resizes, minimises or maximises the browser window. Event handler: onresize. This example shows an alert message right after the user resizes the browser's window.

JavaScript Strings

In JavaScript, strings play a key role on the overall structure of a webpage, since they are a sequence of letters, numbers, special characters and arithmetic values or even a combination of all. They can be created by enfolding the string literal (i.e. string characters) either within single quotes (') or double quotes ("), as follows:

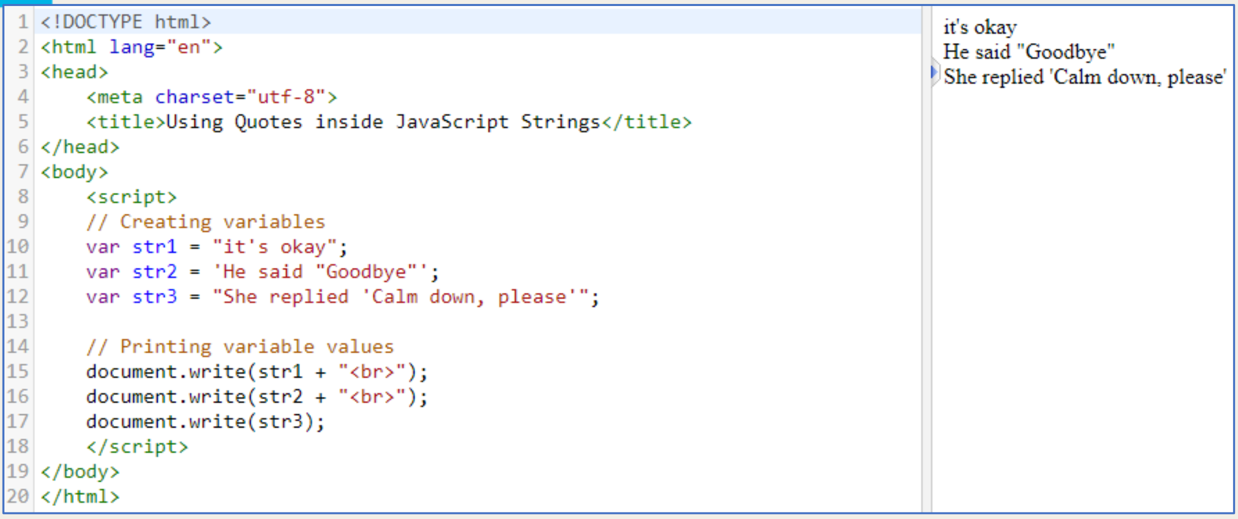


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Creating Strings in JavaScript</title>
</head>
<body>
    <script>
    // Creating variables
    var myString = 'Hello World!'; // Single quoted string
    var myString = "Hello World!"; // Double quoted string

    // Printing variable values
    document.write(myString + "<br>");
    document.write(myString);
    </script>
</body>
</html>
```

```
Hello World!
Hello World!
```

Figure 34 *– JavaScript strings example (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Quotes can be used inside a string, but they should not match the quotes surrounding the string:

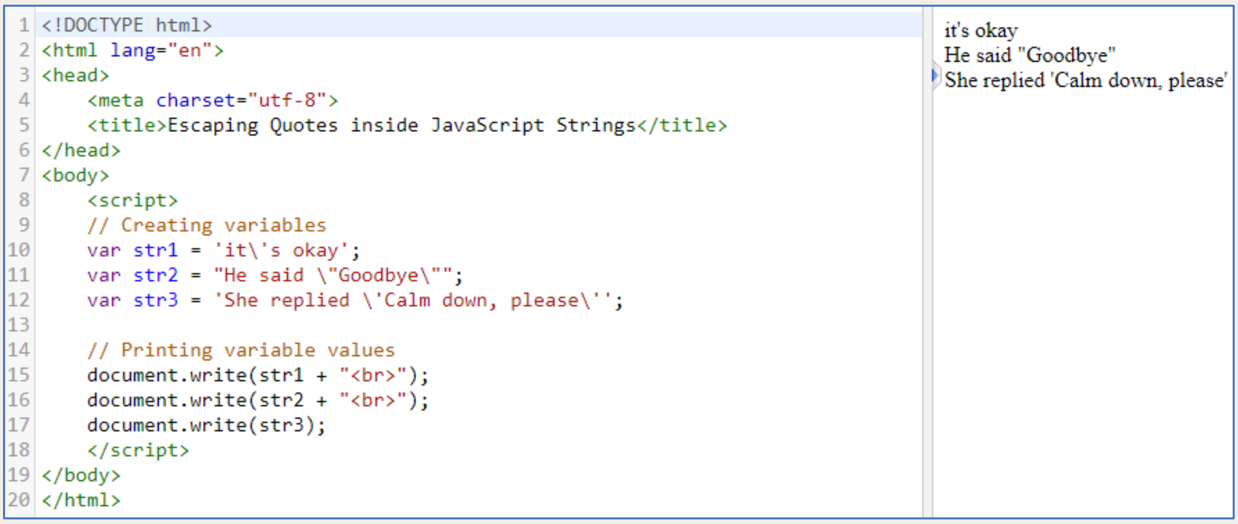


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Using Quotes inside JavaScript Strings</title>
</head>
<body>
    <script>
    // Creating variables
    var str1 = "it's okay";
    var str2 = 'He said "Goodbye"';
    var str3 = "She replied 'Calm down, please'";

    // Printing variable values
    document.write(str1 + "<br>");
    document.write(str2 + "<br>");
    document.write(str3);
    </script>
</body>
</html>
```

```
it's okay
He said "Goodbye"
She replied 'Calm down, please'
```

Figure 35 *– JavaScript strings example II (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Nonetheless, single quotes can still be put inside a single quoted strings or double quotes inside double quoted strings, by separating the quotes with a backslash character (\ ), as *Figure 36* shows. The backslash is termed an **escape character**, and the sequences \’ and \” are **escape sequences**.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Escaping Quotes inside JavaScript Strings</title>
</head>
<body>
    <script>
    // Creating variables
    var str1 = 'it\'s okay';
    var str2 = "He said \"Goodbye\"";
    var str3 = 'She replied \'Calm down, please\'';

    // Printing variable values
    document.write(str1 + "<br>");
    document.write(str2 + "<br>");
    document.write(str3);
    </script>
</body>
</html>
```

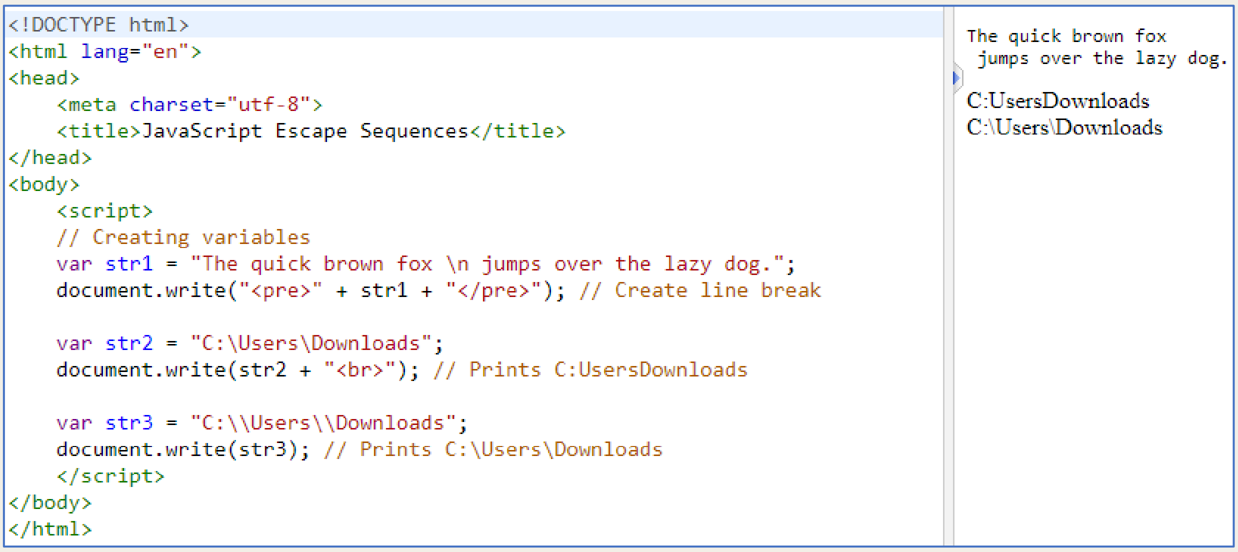
```
it's okay
He said "Goodbye"
She replied 'Calm down, please'
```

Figure 36 *– The backslash (\) escape character (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Escape sequences are likewise valuable for adding characters that cannot be inserted by means of a keyboard. Some other most frequently used escape sequences are:

- \n is replaced by the newline character
- \t is replaced by the tab character
- \r is replaced by the carriage-return character
- \b is replaced by the backspace character
- \\ is replaced by a single backslash (\)

Figure 37 elucidates how escape sequences work:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Escape Sequences</title>
</head>
<body>
    <script>
    // Creating variables
    var str1 = "The quick brown fox \n jumps over the lazy dog.";
    document.write("<pre>" + str1 + "</pre>"); // Create line break

    var str2 = "C:\Users\Downloads";
    document.write(str2 + "<br>"); // Prints C:UsersDownloads

    var str3 = "C:\\Users\\Downloads";
    document.write(str3); // Prints C:\Users\Downloads
    </script>
</body>
</html>
```

```
The quick brown fox
 jumps over the lazy dog.

C:UsersDownloads
C:\Users\Downloads
```

***Figure 37** – How escape sequences work (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

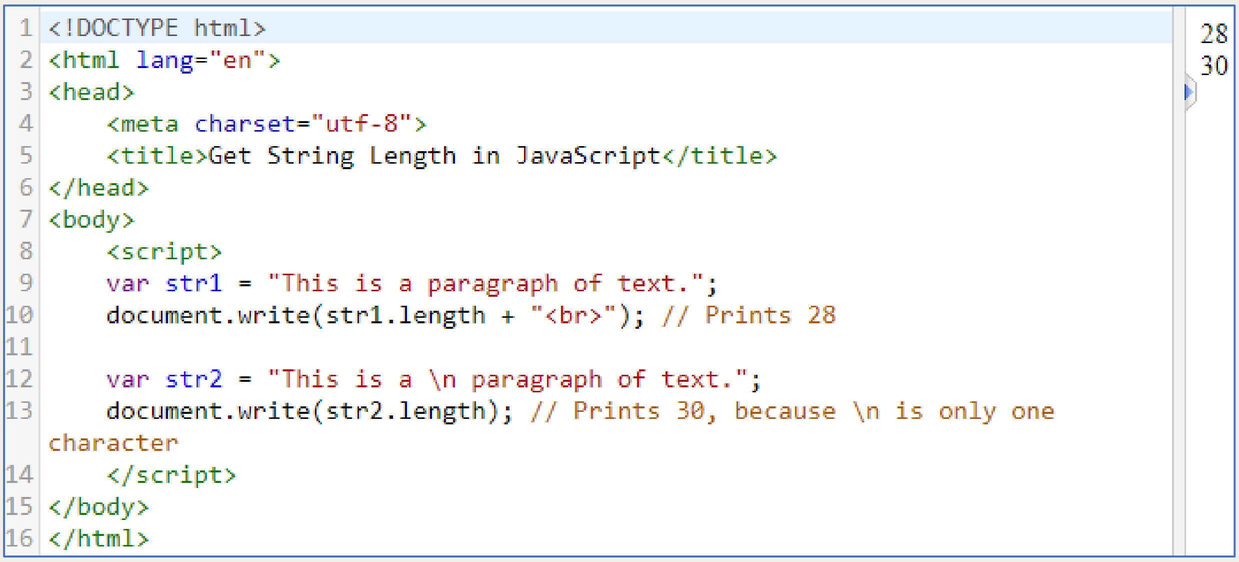
Performing Operations on Strings

JavaScript makes available several properties and methods to make operations on string values. Precisely, only objects can have properties and methods. However, in JavaScript, primitive data types can perform like objects when the programmer refers to them with the property access notation. JavaScript offers this possibility through the creation of a provisional wrapper object for primitive data types. This procedure is done automatically by the JS interpreter in the background.

Getting the Length of a String

The length property returns the length of the string, which is the number of characters delimited in the string, including the number of special characters as well, such as \t or

\n. Programmers should be careful on using parentheses after length (e.g. str.length()), as the correct way is str.length (or else, it will generate an error).



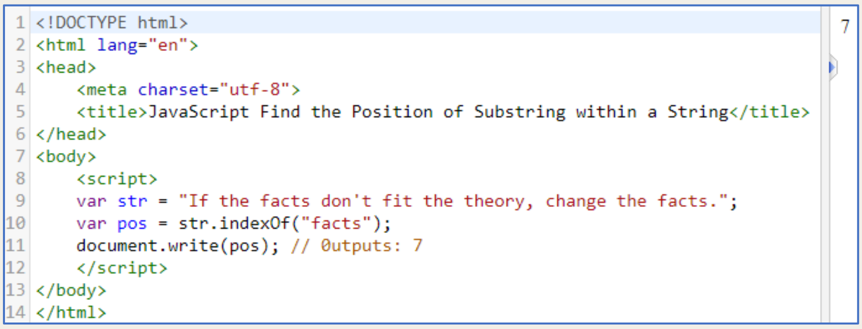
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Get String Length in JavaScript</title>
</head>
<body>
    <script>
    var str1 = "This is a paragraph of text.";
    document.write(str1.length + "<br>"); // Prints 28

    var str2 = "This is a \n paragraph of text.";
    document.write(str2.length); // Prints 30, because \n is only one character
    </script>
</body>
</html>
```

```
28
30
```

Figure 38 *– How to get the length of a string (**Source**: [https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)](https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Finding a String Inside Another String

indexOf() method can be used to find a substring or string within another string. This technique returns the index or position of the first incidence of a specified string within a string.

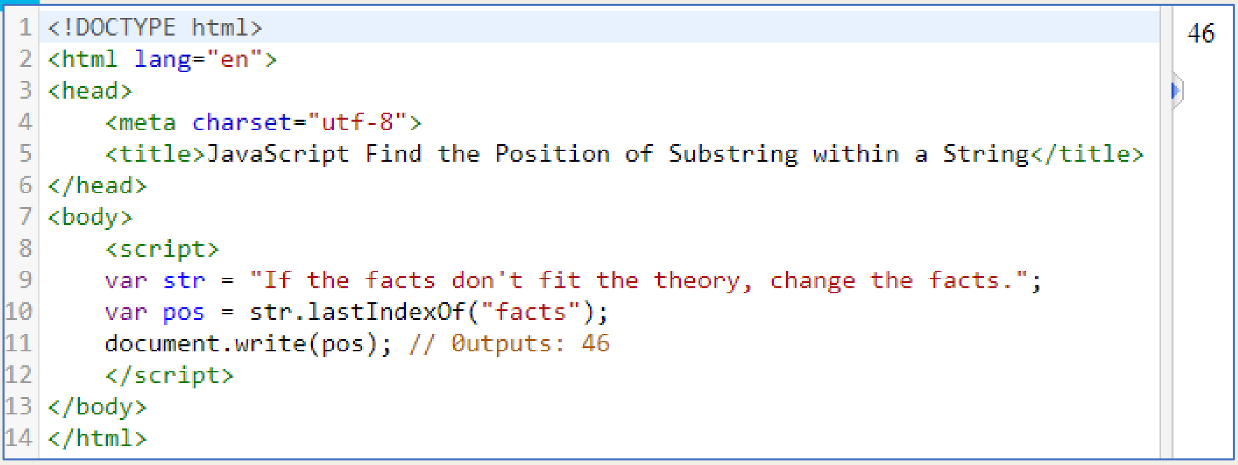


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Find the Position of Substring within a String</title>
</head>
<body>
    <script>
    var str = "If the facts don't fit the theory, change the facts.";
    var pos = str.indexOf("facts");
    document.write(pos); // Outputs: 7
    </script>
</body>
</html>
```

```
7
```

Figure 39 *– Finding a string inside another string (**Source**: [https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)](https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Likewise, the lastIndexOf() technique can be used to get the index or position of the last occurrence of the specified string within a string, as follows:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Find the Position of Substring within a String</title>
</head>
<body>
    <script>
    var str = "If the facts don't fit the theory, change the facts.";
    var pos = str.lastIndexOf("facts");
    document.write(pos); // Outputs: 46
    </script>
</body>
</html>
```

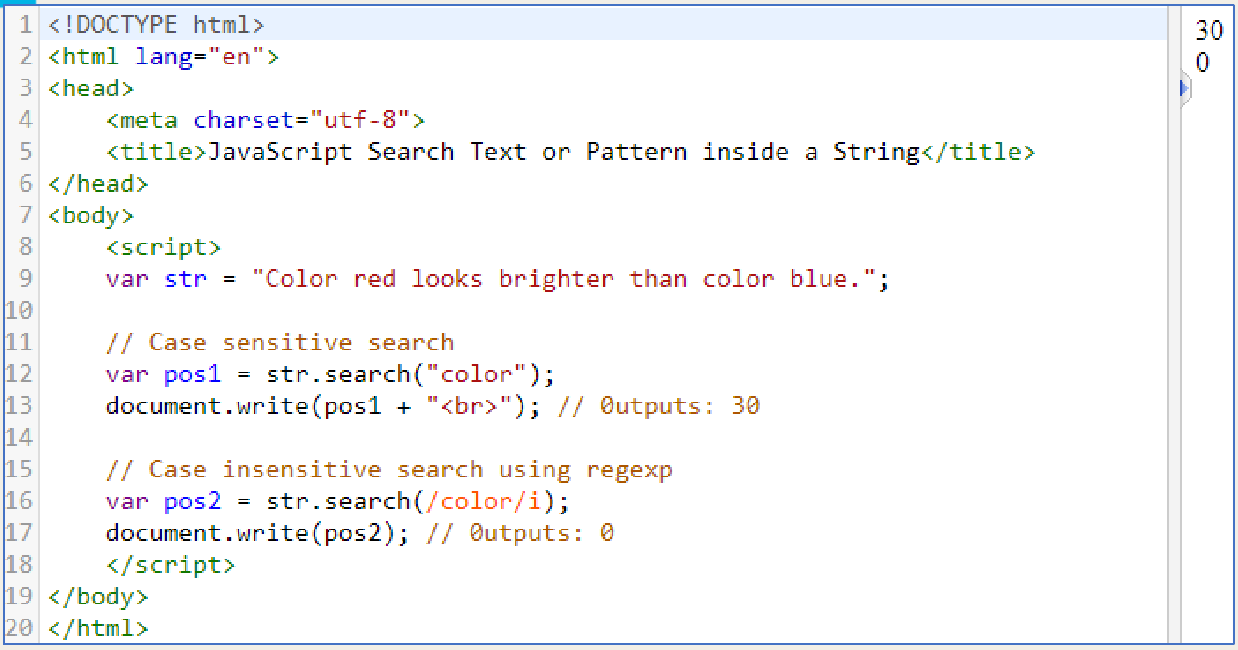
```
46
```

***Figure 40** – Finding a string inside another string using the lastIndexOf() method (**Source**: [https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)](https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Both the indexOf(), and the lastIndexOf() approaches return -1 if the substring is not found. Both methods as well accept an optional integer parameter which stipulates the position within the string at which to start the search.

Searching for a Pattern Inside a String

The search() method can be used to search a particular piece of text or pattern inside a string. As the indexOf() approach, search() also returns the index of the first match, and returns -1 if no matches were found, but unlike indexOf(), search () can as well take a *regular expression* as its argument to deliver advanced search aptitudes. It should be stated that the search() approach does not support global searches, as it disregards the g flag or modifier of its regular expression argument.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Search Text or Pattern inside a String</title>
</head>
<body>
    <script>
    var str = "Color red looks brighter than color blue.";

    // Case sensitive search
    var pos1 = str.search("color");
    document.write(pos1 + "<br>"); // Outputs: 30

    // Case insensitive search using regexp
    var pos2 = str.search(/color/i);
    document.write(pos2); // Outputs: 0
    </script>
</body>
</html>
```

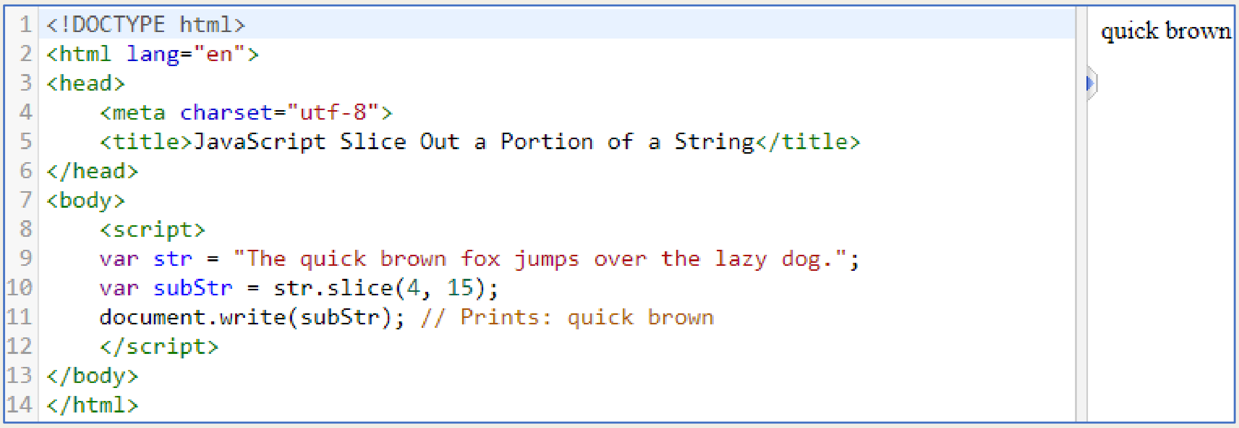
```
30
0
```

Figure 41 – *Searching for a pattern inside a string (**Source**: [https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)](https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Extracting a Substring from a String

For take out a part or substring from a string, the slice() method can be used. This takes two parameters: *start index* (index where extraction begins), and an optional *end index* (index before which to end extraction), like str.slice(startIndex, endIndex).

The example below slices out a portion of a string from position 4 to position 15:



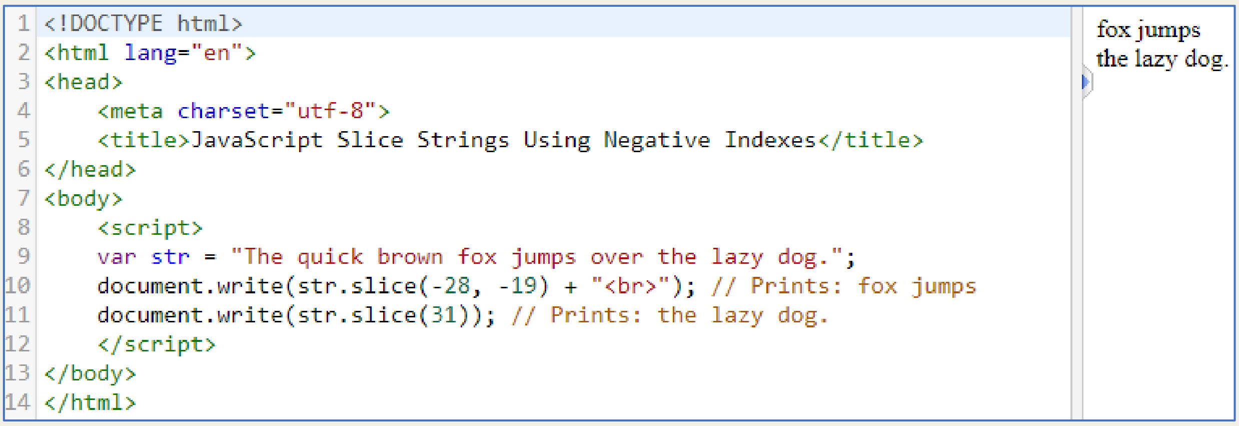
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Slice Out a Portion of a String</title>
</head>
<body>
    <script>
    var str = "The quick brown fox jumps over the lazy dog.";
    var subStr = str.slice(4, 15);
    document.write(subStr); // Prints: quick brown
    </script>
</body>
</html>
```

```
quick brown
```

Figure 42 – *Extracting a Substring from a String using the slice() method (**Source**: [https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)](https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Negative values can be specified as well. These values are treated as strLength + startIndex, where strLength is the length of the string (for instance, str.length), for example, if startIndex is -5 it is treated as strLength - 5. If startIndex is greater than or equal to the length of the string, slice() method returns an empty string. Correspondingly, if optional endIndex is not specified or omitted, the slice() method extracts to the end of the string.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Slice Strings Using Negative Indexes</title>
</head>
<body>
    <script>
    var str = "The quick brown fox jumps over the lazy dog.";
    document.write(str.slice(-28, -19) + "<br>"); // Prints: fox jumps
    document.write(str.slice(31)); // Prints: the lazy dog.
    </script>
</body>
</html>
```

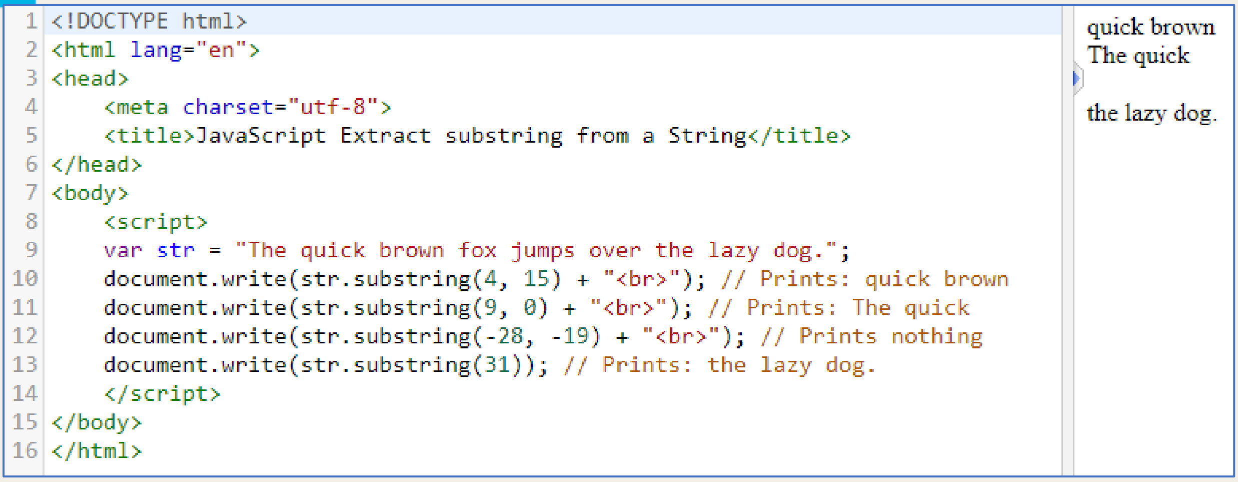
```
fox jumps
the lazy dog.
```

***Figure 43** – Specifying negative values (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

The substring() method to extract a section of the given string based on start and end indexes, as str.substring(startIndex, endIndex). The substring() method is very comparable to the slice(), except some differences:

- If either argument is less than 0 or is NaN, it is treated as 0.
- If either argument is greater than str.length, it is treated as if it were str.length.
- If startIndex is greater than endIndex, then substring() will switch those two arguments; i.e., str.substring(5, 0) == str.substring(0, 5).

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



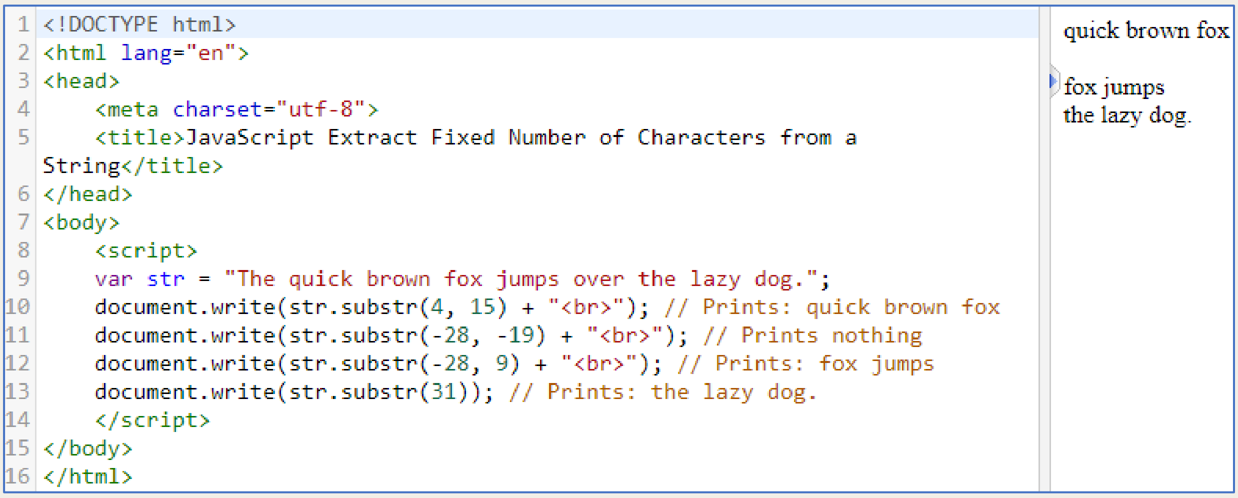
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Extract substring from a String</title>
</head>
<body>
    <script>
    var str = "The quick brown fox jumps over the lazy dog.";
    document.write(str.substring(4, 15) + "<br>"); // Prints: quick brown
    document.write(str.substring(9, 0) + "<br>"); // Prints: The quick
    document.write(str.substring(-28, -19) + "<br>"); // Prints nothing
    document.write(str.substring(31)); // Prints: the lazy dog.
    </script>
</body>
</html>
```

```
quick brown
The quick

the lazy dog.
```

__Figure 44__ – The substring() method to extract a section of the given string based on start and end indexes (**Source**: *https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Extracting a Fixed Number of Characters from a String

JavaScript also delivers the substr() technique, which is similar to slice() with a minor difference: the second parameter stipulates the number of characters to extract instead of ending index, as str.substr(startIndex, length). If length is 0 or a negative number, an empty string is returned.



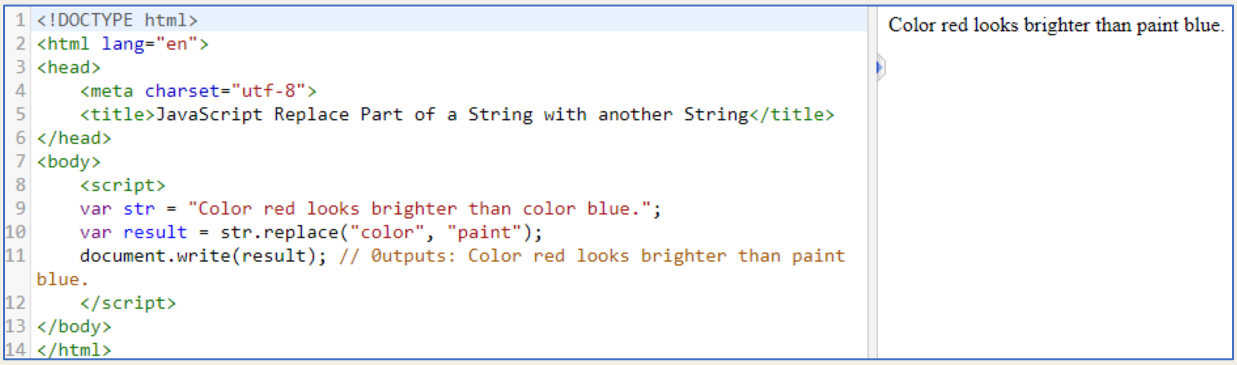
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Extract Fixed Number of Characters from a String</title>
</head>
<body>
    <script>
    var str = "The quick brown fox jumps over the lazy dog.";
    document.write(str.substr(4, 15) + "<br>"); // Prints: quick brown fox
    document.write(str.substr(-28, -19) + "<br>"); // Prints nothing
    document.write(str.substr(-28, 9) + "<br>"); // Prints: fox jumps
    document.write(str.substr(31)); // Prints: the lazy dog.
    </script>
</body>
</html>
```

```
quick brown fox

fox jumps
the lazy dog.
```

__Figure 45__ – Extracting a Fixed Number of Characters from a String (**Source**: *https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Replacing the Contents of a String

The replace() technique is used to replace part of a string with another string. This approach takes a regular expression to match or substring with two parameters and a replacement string, i.e. str.replace(regexp|substr, newSubstr). This replace() method returns a new string, it does not distress the original string, which will remain unaffected.

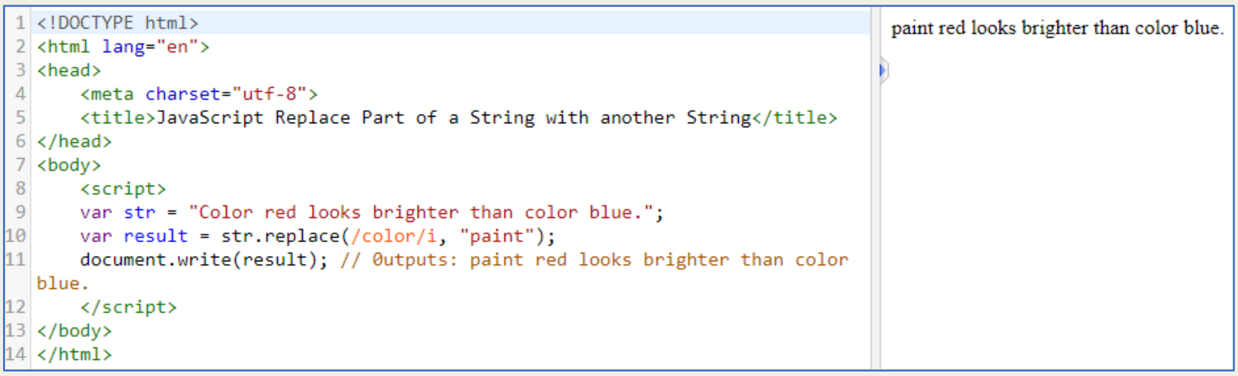


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Replace Part of a String with another String</title>
</head>
<body>
    <script>
    var str = "Color red looks brighter than color blue.";
    var result = str.replace("color", "paint");
    document.write(result); // Outputs: Color red looks brighter than paint blue.
    </script>
</body>
</html>
```

```
Color red looks brighter than paint blue.
```

*Figure 46 – Replacing the Contents of a String using the replace() technique (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

By default, the replace() technique substitutes only the first match, and it is case-sensitive. To replace the substring within a string in a case-insensitive way, a *regular expression (regexp)* with an i modifier can be used, as follows:

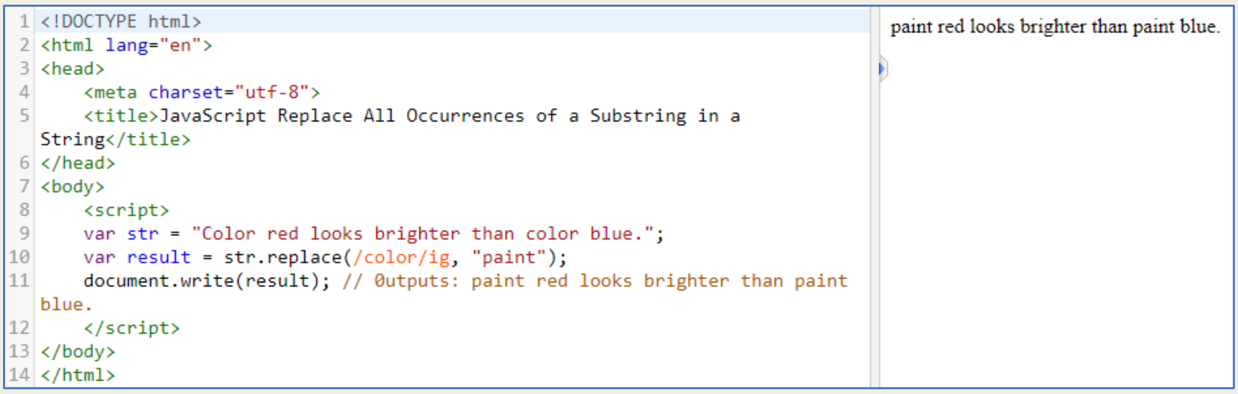


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Replace Part of a String with another String</title>
</head>
<body>
    <script>
    var str = "Color red looks brighter than color blue.";
    var result = str.replace(/color/i, "paint");
    document.write(result); // Outputs: paint red looks brighter than color blue.
    </script>
</body>
</html>
```

```
paint red looks brighter than color blue.
```

*Figure 47 – Replacing the Contents of a String using the replace() technique (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Also, to replace all the incidences of a substring within a string in a case-insensitive manner, the g modifier along with the i modifier can be used, as follows below:



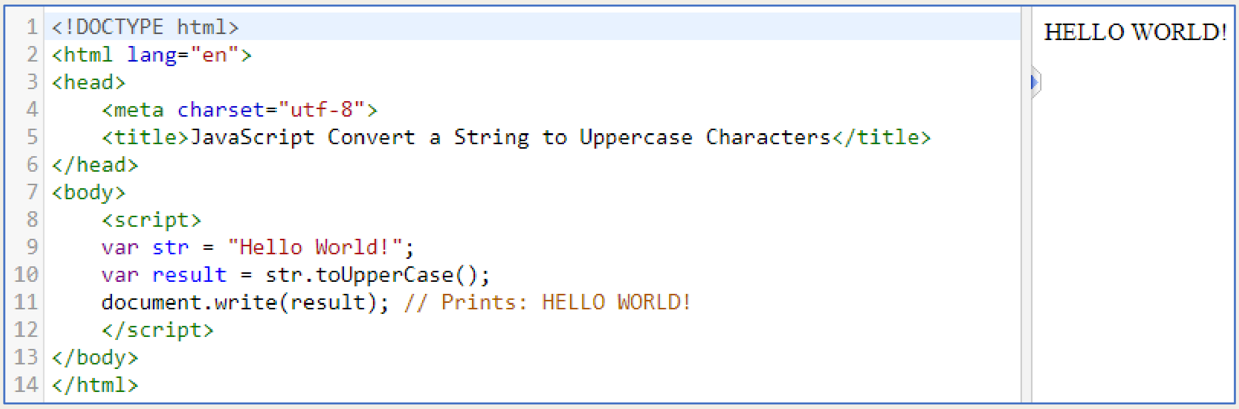
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Replace All Occurrences of a Substring in a String</title>
</head>
<body>
    <script>
    var str = "Color red looks brighter than color blue.";
    var result = str.replace(/color/ig, "paint");
    document.write(result); // 0utputs: paint red looks brighter than paint blue.
    </script>
</body>
</html>
```

```
paint red looks brighter than paint blue.
```

Figure 48 – Replacing all the incidences of a substring within a string in a case-insensitive manner (Source: [*https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*](https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)

Converting a String to Uppercase or Lowercase

The toUpperCase() method is used to convert a string to uppercase, as follows:

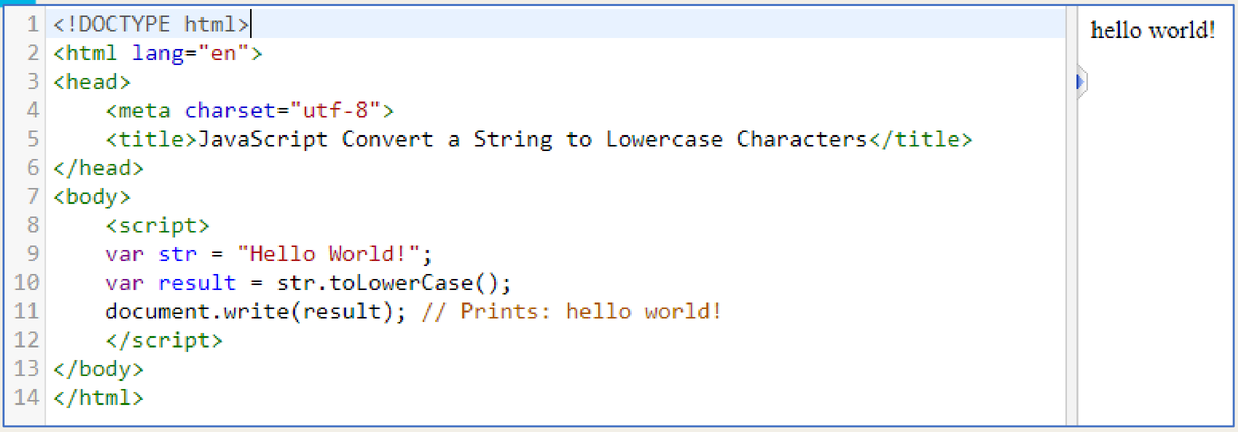


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Convert a String to Uppercase Characters</title>
</head>
<body>
    <script>
    var str = "Hello World!";
    var result = str.toUpperCase();
    document.write(result); // Prints: HELLO WORLD!
    </script>
</body>
</html>
```

```
HELLO WORLD!
```

Figure 49 – Converting a String to Uppercase using toUpperCase() method (Source: [*https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*](https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)

Likewise, the toLowerCase() method is used to convert a string to lowercase:



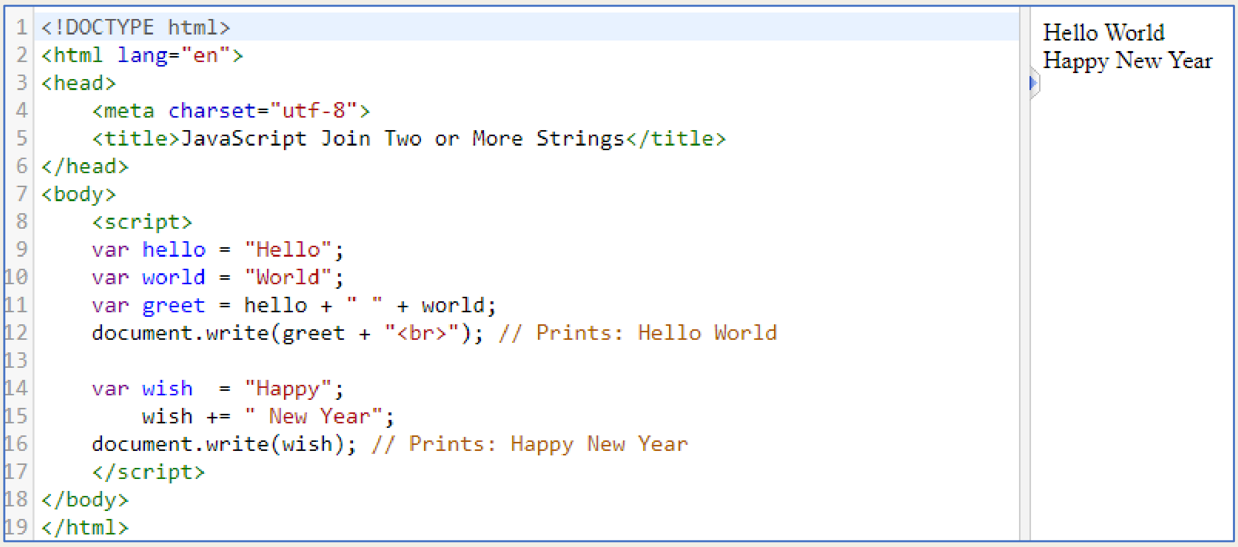
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Convert a String to Lowercase Characters</title>
</head>
<body>
    <script>
    var str = "Hello World!";
    var result = str.toLowerCase();
    document.write(result); // Prints: hello world!
    </script>
</body>
</html>
```

```
hello world!
```

***Figure 50** – Converting a String to Lowercase using toLowerCase() method (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Concatenating Two or More Strings

Two or more strings can be concatenated or combined by using the + and += assignment operators.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Join Two or More Strings</title>
</head>
<body>
    <script>
    var hello = "Hello";
    var world = "World";
    var greet = hello + " " + world;
    document.write(greet + "<br>"); // Prints: Hello World

    var wish  = "Happy";
        wish += " New Year";
    document.write(wish); // Prints: Happy New Year
    </script>
</body>
</html>
```

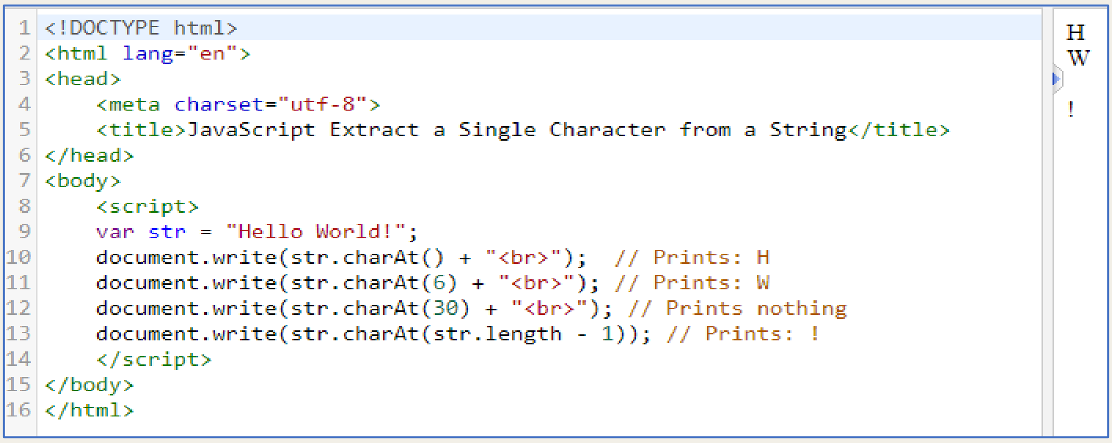
```
Hello World
Happy New Year
```

***Figure 51** – Concatenating Two or More Strings (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Accessing Individual Characters from a String

The charAt() method can be used to access individual character from a string, as str.charAt(index). The index specified should be a number between 0 and str.length - 1. If no index is given, the first character in the string is returned, since the default is 0.

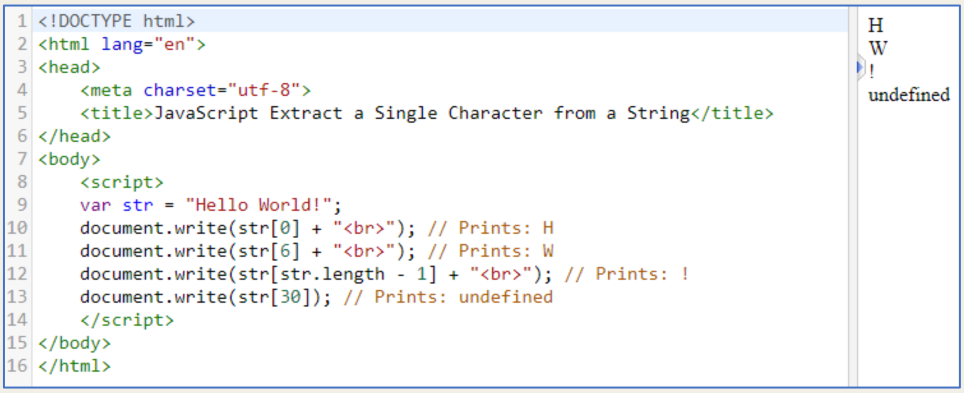


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Extract a Single Character from a String</title>
</head>
<body>
    <script>
    var str = "Hello World!";
    document.write(str.charAt() + "<br>");  // Prints: H
    document.write(str.charAt(6) + "<br>"); // Prints: W
    document.write(str.charAt(30) + "<br>"); // Prints nothing
    document.write(str.charAt(str.length - 1)); // Prints: !
    </script>
</body>
</html>
```

```
H
W

!
```

*Figure 52 – Accessing Individual Characters from a String (**Source**: [https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)](https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

However, there is a good alternative to this procedure. Since ECMAScript 5, strings can be treated like read-only arrays, and individual characters can be seen from a string using square brackets ([]) instead of the charAt() approach, as follows below:



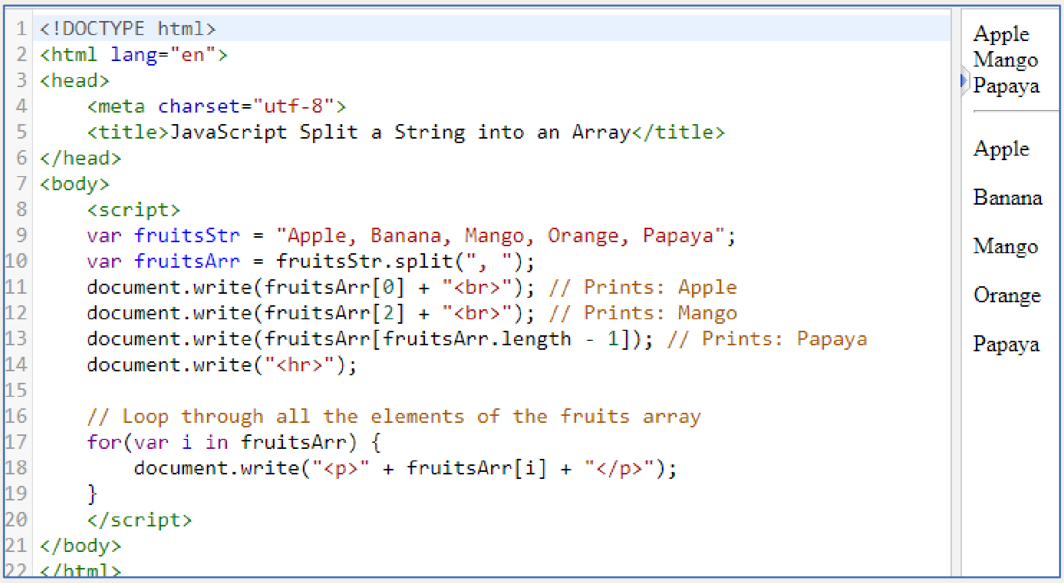
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Extract a Single Character from a String</title>
</head>
<body>
    <script>
    var str = "Hello World!";
    document.write(str[0] + "<br>"); // Prints: H
    document.write(str[6] + "<br>"); // Prints: W
    document.write(str[str.length - 1] + "<br>"); // Prints: !
    document.write(str[30]); // Prints: undefined
    </script>
</body>
</html>
```

```
H
W
!
undefined
```

*Figure 53 – Accessing Individual Characters from a String using square brackets ([]) instead of the charAt() approach (**Source**: [https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)](https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

Splitting a String into an Array

The split() method can be used to fragment a string into an array of strings, using the syntax str.split(separator, limit). The separator argument stipulates the string at which each split should happen, while the limit arguments specifies the maximum length of the array. If separator argument is omitted or not found in the specified string, the entire string is allocated to the first element of the array.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Split a String into an Array</title>
</head>
<body>
    <script>
    var fruitsStr = "Apple, Banana, Mango, Orange, Papaya";
    var fruitsArr = fruitsStr.split(", ");
    document.write(fruitsArr[0] + "<br>"); // Prints: Apple
    document.write(fruitsArr[2] + "<br>"); // Prints: Mango
    document.write(fruitsArr[fruitsArr.length - 1]); // Prints: Papaya
    document.write("<hr>");

    // Loop through all the elements of the fruits array
    for(var i in fruitsArr) {
        document.write("<p>" + fruitsArr[i] + "</p>");
    }
    </script>
</body>
</html>
```

```
Apple
Mango
Papaya
------

Apple

Banana

Mango

Orange

Papaya
```

***Figure 54** – Splitting a String into an Array (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php)*

To split a string into an array of characters, an empty string ("") should be stipulated as a separator, as this example shows.

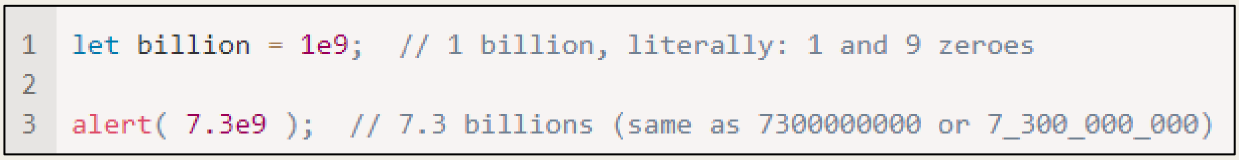
JavaScript Numbers

There are two types of numbers in JavaScript:

- **Regular numbers** in JavaScript are stored in 64-bit format IEEE-754, otherwise known as “*double precision floating point numbers*”. These are the most commonly used type of numbers.
- **BigInt numbers**, to represent integers of random length. They are sometimes needed, because a regular number cannot securely exceed **2**^{53} or be less than **-2**^{53}.

There are more ways to write a number. For instance, the obvious way to write 1 billion would be ‘1000000000’ or ‘1_000_000_000’, using the underscore as a separator. In this case, the underscore is the “syntactic sugar”, meaning that it makes the number clearer to read. The JS engine ignores underscores between digits, so it is the exact same one billion from the first case.

However, in the real world, everyone most certainly will try to avoid writing long sequences of zeros. Something like “1bn” for a billion or “2.5 bn” for 2 billion 500 million seems more reasonable. The same principle applies for most large numbers. That being said, it is possible to shorten a number in JS, by adding the letter “e” to it and specifying the quantity of zeroes:



```
let billion = 1e9;  // 1 billion, literally: 1 and 9 zeroes

alert( 7.3e9 );  // 7.3 billions (same as 7300000000 or 7_300_000_000)
```

***Figure 55** – Specifying the quantity of zeroes (**Source**: https://javascript.info/number)*

So, e multiplies the number by 1 with the given zeroes count.

```
1e3 === 1 * 1000; // e3 means *1000
1.23e6 === 1.23 * 1000000; // e6 means *1000000
```

The same principle aplies for small numbers. For instance, what should be written for 1 microsecond (one millionth of a second)?

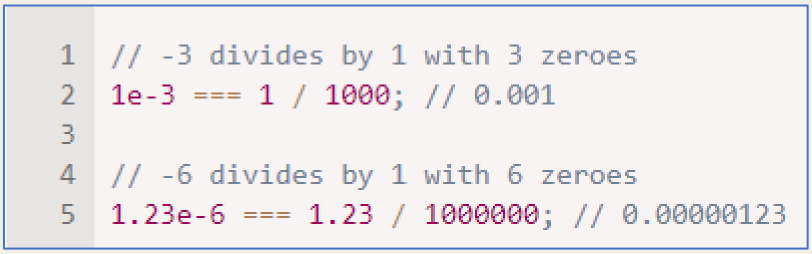
```
let mcs = 0.000001;
```

Just like the aforementioned case with the big numbers, using "e" can be helpful. To avoid writing the zeroes explicitly, the following can be written:

```
let mcs = 1e-6; // six zeroes to the left from 1
```

There are 6 zeroes in 0.000001. Then, it is not hard to conclude 1e-6.

Therefore, a negative number after "e" implies a division by 1 with the given number of zeroes, as follows:

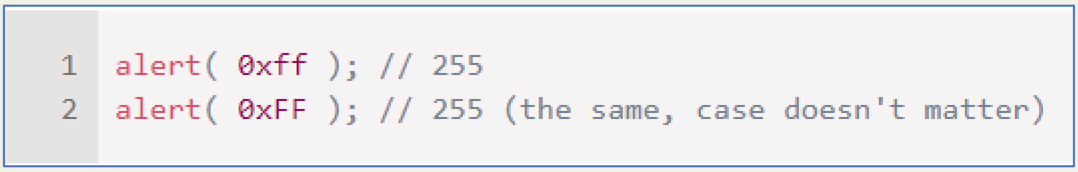


```
// -3 divides by 1 with 3 zeroes
1e-3 === 1 / 1000; // 0.001

// -6 divides by 1 with 6 zeroes
1.23e-6 === 1.23 / 1000000; // 0.00000123
```

Figure 56 – The division of 1 by the given number of zeroes (Source: https://javascript.info/number)

Hex, binary, and octal numbers

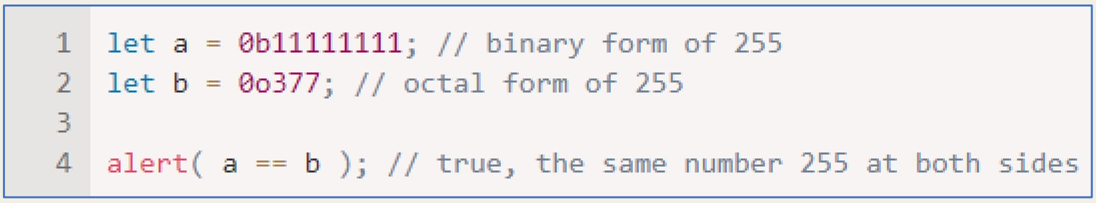
Hexadecimal numbers are commonly utilized in JavaScript to embody colours, encode characters, and for many other purposes. So obviously, there is a quicker way to write them: 0x and then the number, as follows:



```
alert( 0xff ); // 255
alert( 0xFF ); // 255 (the same, case doesn't matter)
```

Figure 57 – The 0x way to speed up JS processes (Source: https://javascript.info/number)

Binary and octal numeral systems are not often utilized, but as well supported using the 0b and 0o (*zero*, *o*) prefixes:



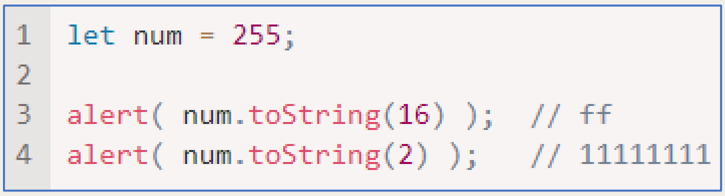
```
let a = 0b11111111; // binary form of 255
let b = 0o377; // octal form of 255

alert( a == b ); // true, the same number 255 at both sides
```

***Figure 58** – The 0b and 0o prefixes for binary and octal numeral systems (**Source**: https://javascript.info/number)*

There are merely 3 numeral systems with this support. For further numeral systems, the function parseInt should be used.

toString(base) method

The num.toString(base) method returns a string representation of num in the numeral system with the provided base, as follows:



```
let num = 255;

alert( num.toString(16) );  // ff
alert( num.toString(2) );   // 11111111
```

***Figure 59** – The num.toString(base) method (**Source**: https://javascript.info/number)*

The base can vary from 2 to 36. By default, it's 10.

There are general use cases for this:

- **base=16** is applied for hex colours, character encodings etc, digits can be 0..9 or A..F.
- **base=2** is mainly for correcting bitwise operations, digits can be 0 or 1.
- **base=36** is the maximum, digits can be 0..9 or A..Z. The entire Latin alphabet is used to represent a number. A curious, but useful case for 36 is when it is needed to turn a long numeric identifier into something shorter, for example to make a short url.

There is a specific case to be noted. When two dots are spotted in 123456..toString(36), this is not a typo. When the programmer wants to call a method directly on a number, it is needed to place two dots ('..') after it.

If a single dot is placed as follows ('123456.toString(36)'), it would be an error, as JavaScript syntax entails the decimal part after the first dot. And if the programmer places one more dot, then JavaScript knows that the decimal part is empty and now goes the method.

Rounding

One of the most applied operations when operating with numbers is **rounding**.

There are some built-in functions for rounding:

- **Math.floor**

Rounds down: 3.1 turn into 3, and -1.1 converts -2.

- **Math.ceil**

Rounds up: 3.1 becomes 4, and -1.1 converts -1.

- **Math.round**

Rounds to the nearest integer: 3.1 becomes 3, 3.6 becomes 4, the middle case: 3.5 rounds up to 4 too.

- **Math.trunc (not supported by Internet Explorer)**

Eliminates anything after the decimal point without rounding: 3.1 turn out to be 3, -1.1 becomes -1.

These functions cover all the potential ways to cope with the decimal part of a number. But what about rounding the number to n-th digit after the decimal?

For example, to round 1.2345 to two digits (1.23).

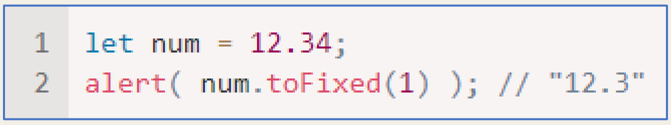
There are two ways to do so:

1. **Multiply-and-divide.**

E.g., to round the number to the 2nd digit after the decimal, one can multiply the number by 100 (or a bigger power of 10), call up the rounding function and then divide it back.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

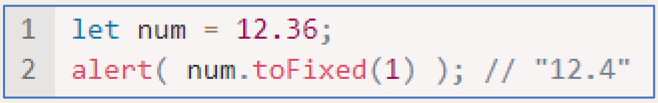
2. The **toFixed(n)** method rounds the number to n digits after the point and returns a string representation of the result.



```
let num = 12.34;
alert( num.toFixed(1) ); // "12.3"
```

Figure 60 *– The toFixed(n) method (**Source**: https://javascript.info/number)*

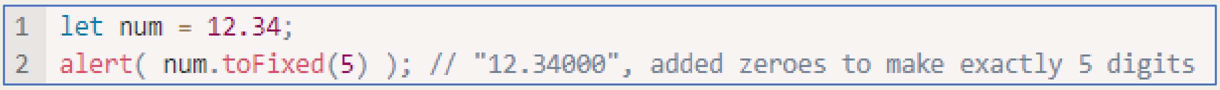
This rounds up or down to the nearest value, like Math.round:



```
let num = 12.36;
alert( num.toFixed(1) ); // "12.4"
```

Figure 61 *– The toFixed(n) method – continuation (**Source**: https://javascript.info/number)*

It should be stated that result of toFixed is a string. If the decimal part is shorter than required, zeroes are attached to the end:



```
let num = 12.34;
alert( num.toFixed(5) ); // "12.34000", added zeroes to make exactly 5 digits
```

Figure 62 *– The toFixed(n) method – final exposition (**Source**: https://javascript.info/number)*

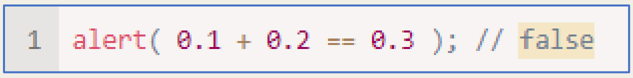
Imprecise calculations

On the inside, a number is represented in 64-bit format **IEEE-754**, so there are precisely 64 bits to store a number: 52 to store the digits; 11 to store the position of the decimal point (zero for integer numbers), and 1 bit is for the sign.

If a number is too big, it would pour out the 64-bit storage, possibly giving an infinity.

Something that ends to happen quite often is the loss of precision.

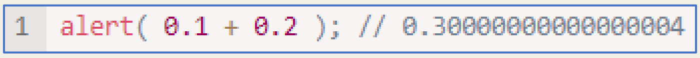
Considering the following test:



```
alert( 0.1 + 0.2 == 0.3 ); // false
```

Figure 63 *– First test for imprecise calculations (**Source**: https://javascript.info/number)*

As it looks, strangely the result from the 0.1+0.2 sum is a false 0.3. What could it be, then?



```
alert( 0.1 + 0.2 ); // 0.30000000000000004
```

***Figure 64** – Second test for imprecise calculations (**Source**: https://javascript.info/number)*

As it could be imagined, there are more consequences than an incorrect comparison in this case. Pretending that a purchase is being made on an online shop and the visitor puts $0.10 and $0.20 goods into his/her cart… the total amount of the order would be $0.30000000000000004. That would be quite surprising.

Why does a situation like this happen?

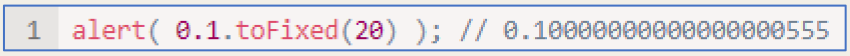
A number is stored in memory in its binary form, a sequence of bits – ones and zeroes. However, fractions like 0.1, 0.2 that seem simple in the decimal numeric system are in fact unending fractions in their binary form.

That is to say, what is 0.1? It is one divided by ten 1/10, one-tenth. In decimal numeral system such numbers are easily representable. Comparing it to one-third: 1/3. It becomes an endless fraction 0.33333(3).

So, division by powers 10 is guaranteed to work well in the decimal system, but division by 3 is not. For the same reason, in the binary numeral system, the division by powers of 2 is guaranteed to work, but 1/10 becomes an endless binary fraction.

There is just no way to store exactly 0.1 or exactly 0.2 using the binary system, just like there is no way to store one-third as a decimal fraction.

The numeric format IEEE-754 resolves this by rounding to the closest possible number. These rounding rules normally do not allow one to sees that “tiny precision loss”, but it exists.

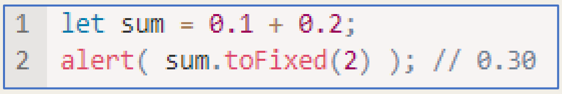


```
alert( 0.1.toFixed(20) ); // 0.10000000000000000555
```

Figure 65 – Final test for imprecise calculations (Source: https://javascript.info/number)

When two numbers are summed, their ‘’precision losses’’ come together. Therefore 0.1 + 0.2 is not 0.3 precisely.

The most reliable method to cope with this situation is **toFixed(n)**:



```
let sum = 0.1 + 0.2;
alert( sum.toFixed(2) ); // 0.30
```

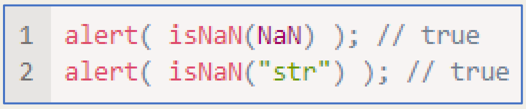
Figure 66 – The toFixed(in) method (Source: https://javascript.info/number)

It should be stated that toFixed(n) returns a string every time. It ensures that it has 2 digits after the decimal point. That is quite useful if there is need to show $0.30 for e-shopping purposes.

Tests: isFinite and isNaN

Previously, those two special numeric values have been described. **Infinity** (and **-Infinity**) is a special numeric value that is greater (less) than anything. **NaN** represents an error. Both belong to the type **number**, but are not “normal” numbers, so there are special functions to check for them:

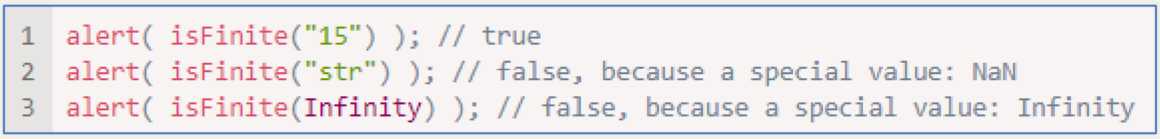
- **isNaN(value)** converts its argument to a number and then tests it for being **NaN**:



```
alert( isNaN(NaN) ); // true
alert( isNaN("str") ); // true
```

Figure 67 – The isNaN(value) method (Source: https://javascript.info/number)

- **isFinite(value)** converts its argument to a number and returns **true** if it’s a regular number, not **NaN/Infinity/-Infinity;**
-



```
alert( isFinite("15") ); // true
alert( isFinite("str") ); // false, because a special value: NaN
alert( isFinite(Infinity) ); // false, because a special value: Infinity
```

***Figure 68** – The isFinite(value) method (**Source**: https://javascript.info/number)*

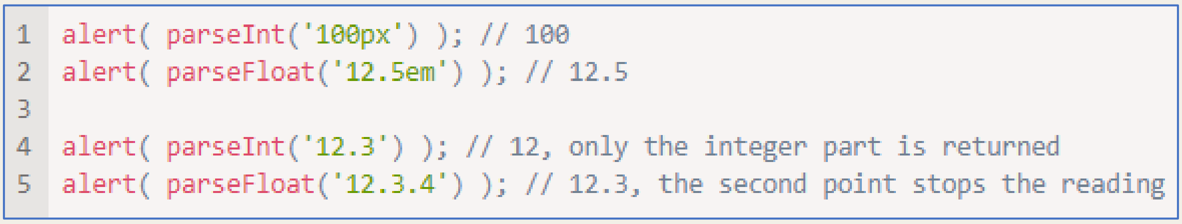
parseInt and parseFloat

Numeric conversion using a plus **+** or **Number()** is rigorous. If a value is not exactly a number, it fails. The only exception is spaces at the beginning or at the end of the string, as they are ignored.

However, in real life there are often values in units, as "100px" or "12pt" in CSS. Likewise, in many countries the currency symbol goes after the amount, so having the example of “19€", the programmer would like to extract a numeric value out of that.

That’s what **parseInt** and **parseFloat** are for.

They “*read*” a number from a string until they cannot. In case of an error, the collected number is returned. The function **parseInt** returns an integer, whereas **parseFloat** will return a floating-point number:



```
alert( parseInt('100px') ); // 100
alert( parseFloat('12.5em') ); // 12.5

alert( parseInt('12.3') ); // 12, only the integer part is returned
alert( parseFloat('12.3.4') ); // 12.3, the second point stops the reading
```

***Figure 69** – parseInt and parseFloat (**Source**: https://javascript.info/number)*

Other math functions

JavaScript owns a built-in Math object which includes a small library of mathematical functions and constants:

- **Math.random()**

Returns a random number from 0 to 1 (not including 1).

- **Math.max(a, b, c...) / Math.min(a, b, c...)**

Returns the greatest/smallest from the arbitrary number of arguments.

- **Math.pow(n, power)**

Returns n raised to the given power.

JavaScript If…Else Statements

As many other programming languages, JavaScript also allows to write code that execute different actions based on the results of a logical or comparative test conditions at run time. Thus, test conditions can be created as expressions that evaluates to either true or false and, based on these results, certain actions can be performed.

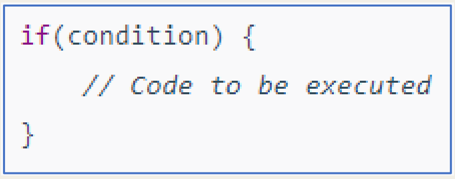
There are several conditional statements in JavaScript that can be used to make decisions:

- The **if** statement;
- The **if...else** statement;
- The **if...else if....else** statement;
- The **switch...case** statement.

The if statement

The if statement is used to execute a block of code only if the specified condition is evaluated as being true. This is the simplest JS's conditional statements and can be written as:

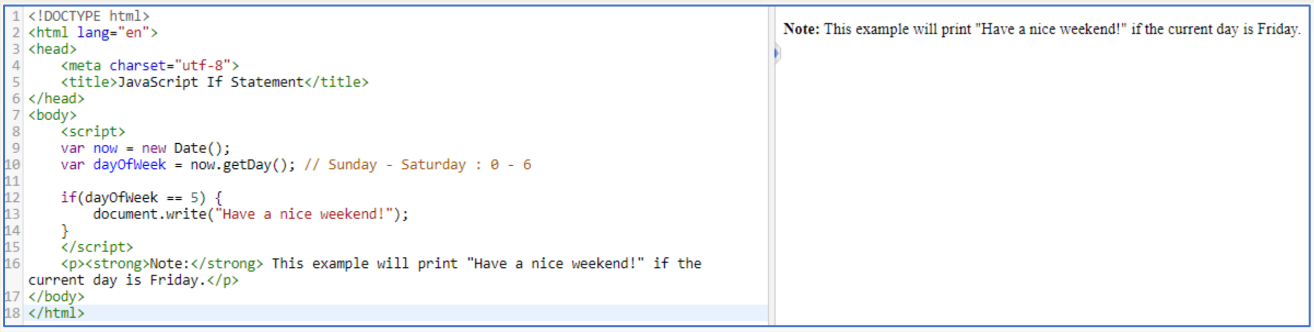
The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
if(condition) {
    // Code to be executed
}
```

***Figure 70** – Code to create an if condition (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-if-else-statements.php)*

A practical example can be quite demonstrative of this feature's usefulness. If the current day is Friday, the coding shown on *Figure 68* will display a message saying "Have a nice weekend":



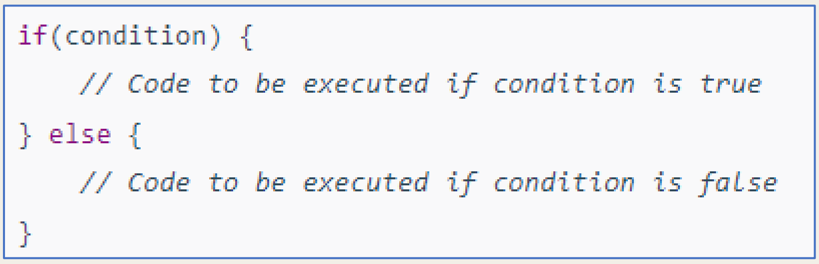
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript If Statement</title>
</head>
<body>
    <script>
    var now = new Date();
    var dayOfWeek = now.getDay(); // Sunday - Saturday : 0 - 6

    if(dayOfWeek == 5) {
        document.write("Have a nice weekend!");
    }
    </script>
    <p><strong>Note:</strong> This example will print "Have a nice weekend!" if the
current day is Friday.</p>
</body>
</html>
```

Note: This example will print "Have a nice weekend!" if the current day is Friday.

***Figure 71** – If...statement created to display a "Have a nice weekend" message in case is Friday (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-if-else-statements.php)*

The if…else statement

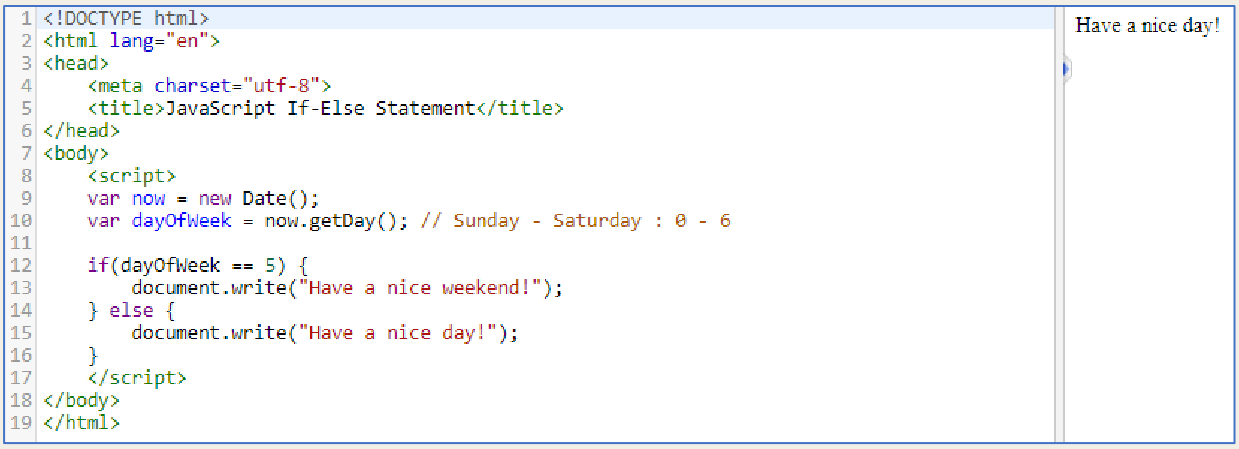
The decision-making process of JS can be enhanced by providing an alternative choice through adding an **else statement** to the **if statement**. The if…else statement allows to execute one block of code if the specified condition evaluates as *true* and another block of code if it evaluates as *false*.



```
if(condition) {
    // Code to be executed if condition is true
} else {
    // Code to be executed if condition is false
}
```

***Figure 72** – Code to create an if…else condition (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-if-else-statements.php)*

The example from *Figure 68* will be applied to test the if…else statement. This time, the “Have a nice weekend!” message will be displayed in case the current day is Friday (as the previous case), however, it will emit the “Have a nice day” message if it is not Friday.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript If-Else Statement</title>
</head>
<body>
    <script>
    var now = new Date();
    var dayOfWeek = now.getDay(); // Sunday - Saturday : 0 - 6

    if(dayOfWeek == 5) {
        document.write("Have a nice weekend!");
    } else {
        document.write("Have a nice day!");
    }
    </script>
</body>
</html>
```

```
Have a nice day!
```

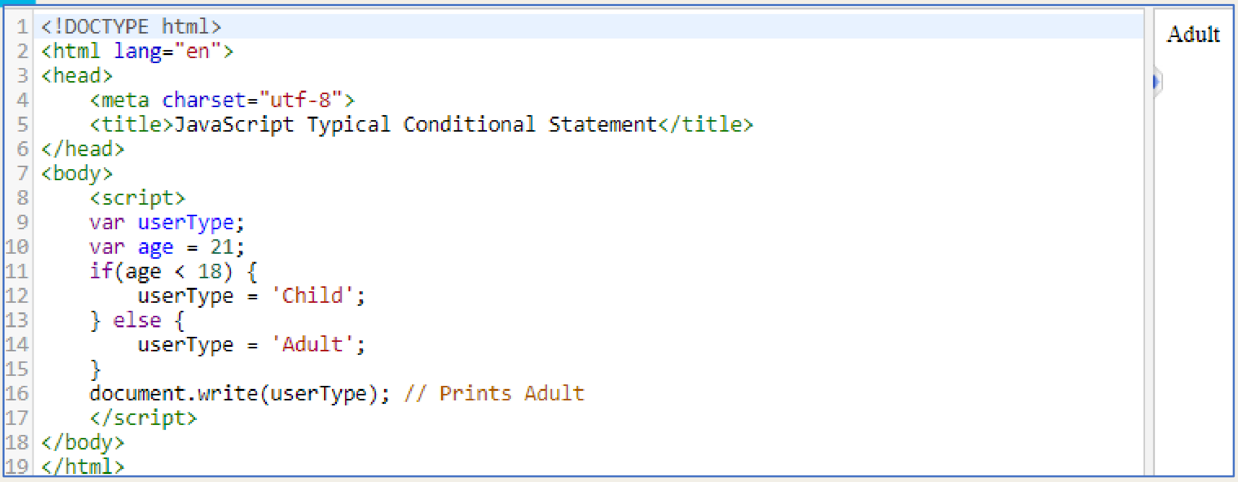
Figure 73 – *Code to create an if…else condition, displaying a “Have a nice day” or “Have a nice weekend” message depending on the case (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-if-else-statements.php)*

The Ternary Operator

The ternary operator gives a **shorthand way of writing the if...else statements**. It is characterized by the question mark (‘?’) symbol and it takes three operands: a condition to check, a result for true, and a result for false. Its basic syntax is as follows:

```
var result = (condition) ? value1 : value2
```

If the condition is evaluated as *true,* the value1 will be returned, if not value2 will be returned.



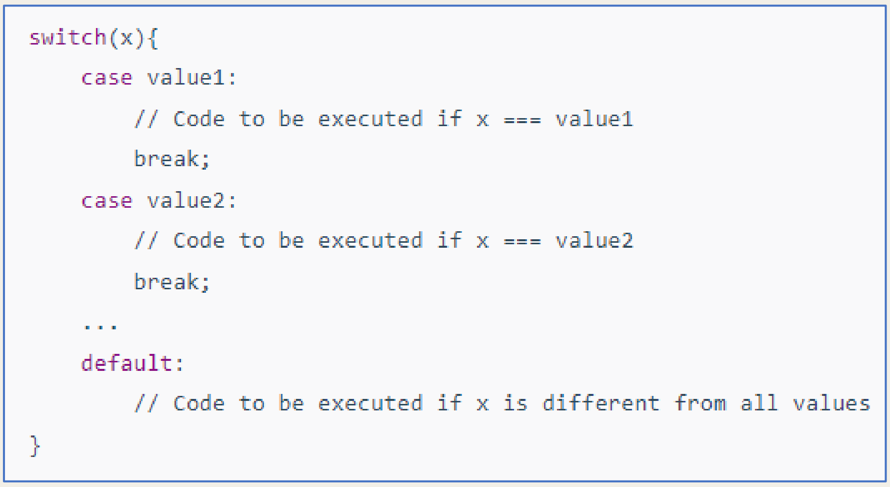
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Typical Conditional Statement</title>
</head>
<body>
    <script>
    var userType;
    var age = 21;
    if(age < 18) {
        userType = 'Child';
    } else {
        userType = 'Adult';
    }
    document.write(userType); // Prints Adult
    </script>
</body>
</html>
```

```
Adult
```

*Figure 74 – How to apply the ternary operator (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-if-else-statements.php)*

JavaScript Switch…Case Statements

The **switch..case** statement is an alternate scenario to the **if...else if...else** statement, which does nearly the same thing. The **switch...case** statement analyses a variable or expression against a series of values until it finds a match, and then executes the block of code corresponding to that match. Its syntax is as follows:



```
switch(x){
    case value1:
        // Code to be executed if x === value1
        break;
    case value2:
        // Code to be executed if x === value2
        break;
    ...
    default:
        // Code to be executed if x is different from all values
}
```

***Figure 75 –** Syntax for switch…case statements (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-if-else-statements.php)*

This example displays the name of the day of the week the reader is in.

The **switch...case** statement diverges from the **if...else** statement in one crucial way. The switch statement executes line by line and when JavaScript finds a case clause that evaluates to *true*, it does not only performs the code corresponding to that case clause, but also executes all the successive case clauses till the end of the switch block automatically.

To prevent this, a break statement must be included after each case (as can be checked on *Figure 73*). The break statement informs the JS interpreter to break out of the **switch...case** statement block once it executes the code linked to the first *true* case.

The break statement is yet not required for the case or default clause when it appears at last in a switch statement. However, it is a good programming practice to dismiss the last case, or default clause in a switch statement with a break. It prevents a possible programming error later on if another case statement is included in the switch statement.

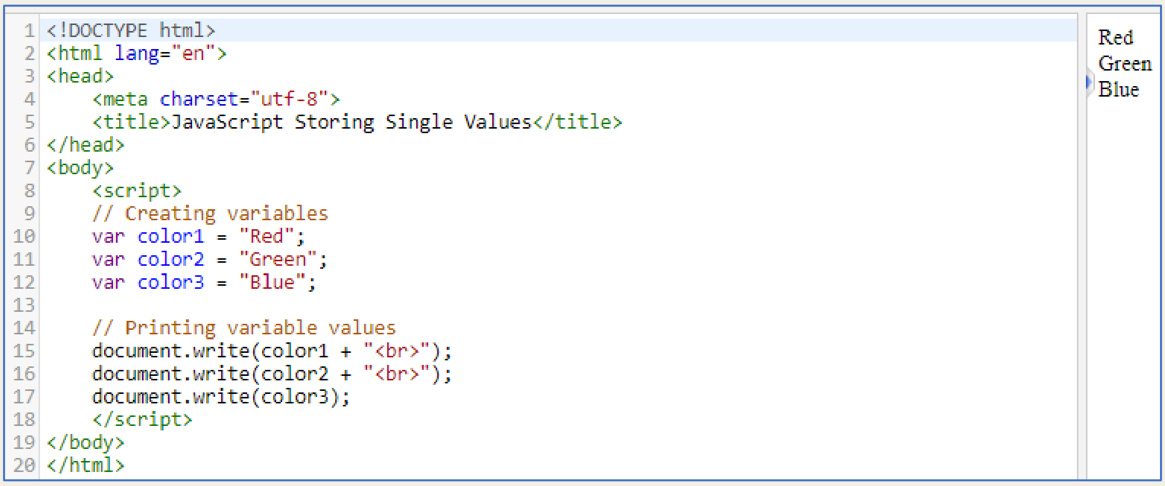
The default clause is voluntary, which stipulates the actions to be made if no case matches the switch expression. The default clause does not have to be the last clause to be found in a switch statement.

Each case value must be exclusive within a switch statement. Still, different cases do not need to have a unique action. Several cases can share the same action.

JavaScript Arrays

Arrays are complex variables that allow to store more than one value or a group of values under a single variable name. JavaScript arrays can store any valid value, including strings, numbers, objects, functions, and at the same time other arrays, therefore enabling to create more complex data structures such as an array of objects or an array of arrays.

In the following example, the name of colours will be stored in a JavaScript code:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Storing Single Values</title>
</head>
<body>
    <script>
    // Creating variables
    var color1 = "Red";
    var color2 = "Green";
    var color3 = "Blue";

    // Printing variable values
    document.write(color1 + "<br>");
    document.write(color2 + "<br>");
    document.write(color3);
    </script>
</body>
</html>
```

```
Red
Green
Blue
```

***Figure 76 –** Storing the name of colours in a JS code (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

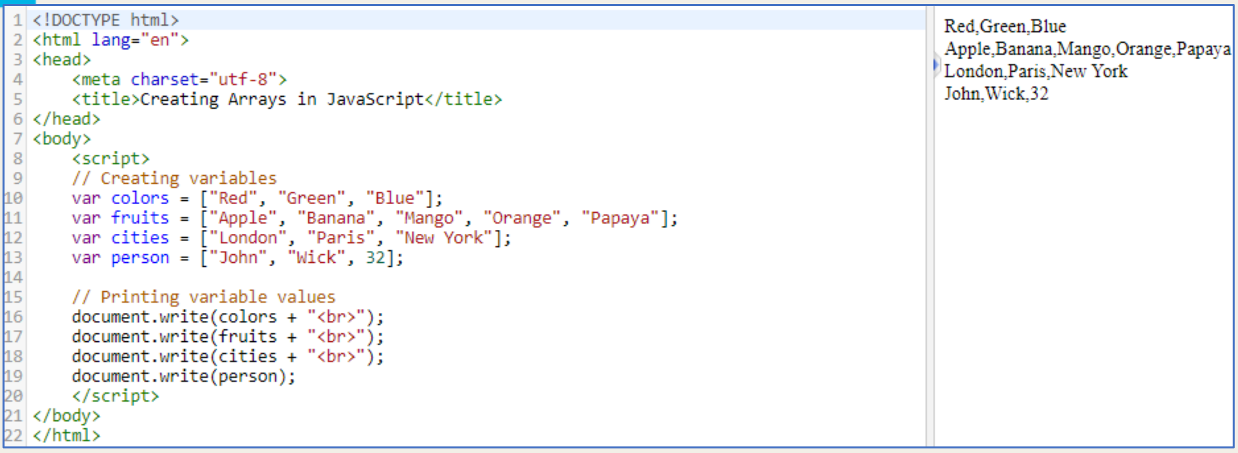
However, it may be quite difficult and tiring to store multiple elements in variables, opposite to only three in the case above. Additionally, using so many variables at the same time and keeping track of them all will be a very tricky task. And here array comes into play. Arrays solve this problem by providing an ordered structure for storing multiple values or a group of values.

Creating an Array

The easiest way to create an array in JavaScript is enclosing a comma-separated list of values in square brackets ([]), as shown in the following syntax:

var myArray = [*element0*, *element1*, ..., *elementN*];

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Creating Arrays in JavaScript</title>
</head>
<body>
    <script>
    // Creating variables
    var colors = ["Red", "Green", "Blue"];
    var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
    var cities = ["London", "Paris", "New York"];
    var person = ["John", "Wick", 32];

    // Printing variable values
    document.write(colors + "<br>");
    document.write(fruits + "<br>");
    document.write(cities + "<br>");
    document.write(person);
    </script>
</body>
</html>
```

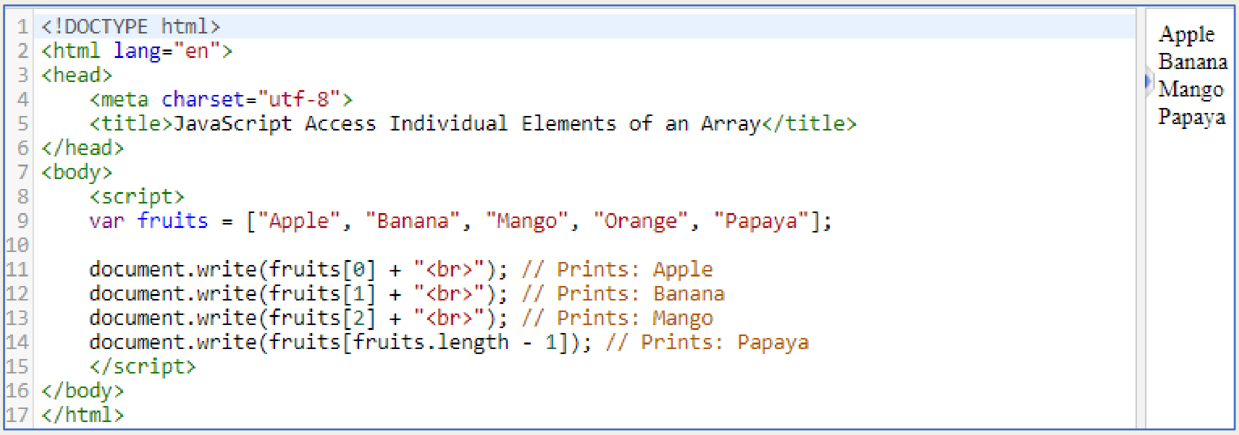
```
Red,Green,Blue
Apple,Banana,Mango,Orange,Papaya
London,Paris,New York
John,Wick,32
```

***Figure 77 –** Creating an Array (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

Acessing the elements of an Array

Array elements can be accessed by their index utilizing the square bracket notation. An index is a number that represents an element's position in an array.

Array indexes are zero-based. This means that the first item of an array is stored at index 0, not 1, the second item is stored at index 1, and so on. Array indexes start at 0 and go up to the number of elements minus 1. So, array of five elements would have indexes from 0 to 4.



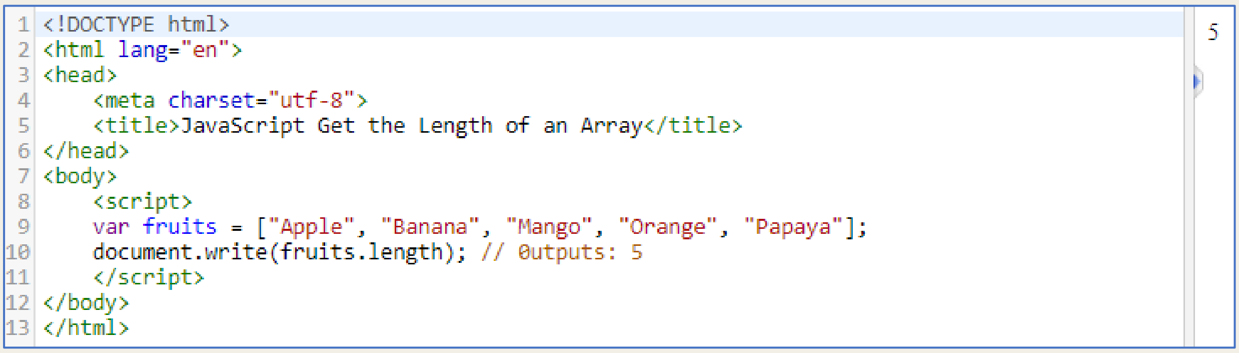
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Access Individual Elements of an Array</title>
</head>
<body>
    <script>
    var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];

    document.write(fruits[0] + "<br>"); // Prints: Apple
    document.write(fruits[1] + "<br>"); // Prints: Banana
    document.write(fruits[2] + "<br>"); // Prints: Mango
    document.write(fruits[fruits.length - 1]); // Prints: Papaya
    </script>
</body>
</html>
```

```
Apple
Banana
Mango
Papaya
```

***Figure 78 –** How to get individual array element by their index (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

Getting the Length of an Array

The length property returns the length of an array, which is the total number of elements included in the array. Array length is always greater than the index of any of its element.



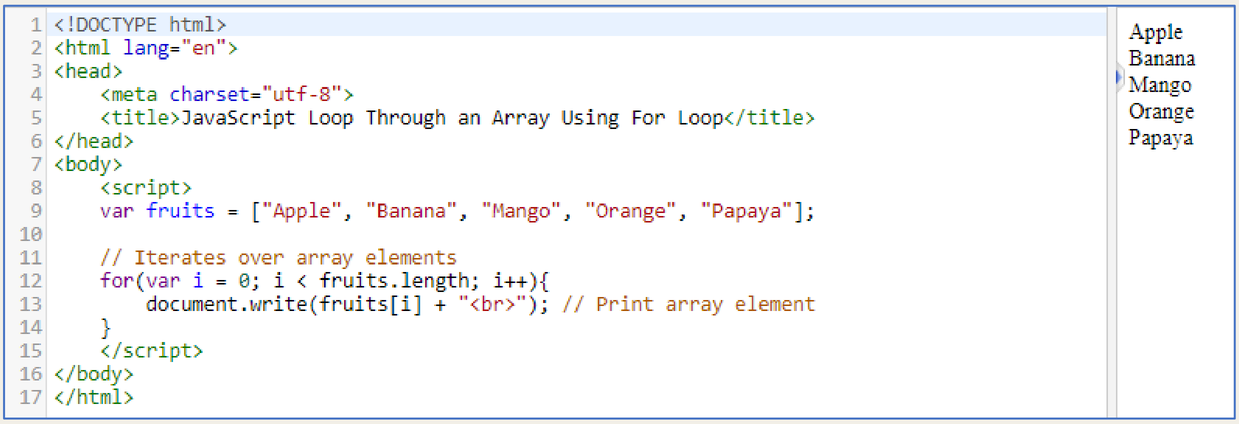
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Get the Length of an Array</title>
</head>
<body>
    <script>
    var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
    document.write(fruits.length); // Outputs: 5
    </script>
</body>
</html>
```

```
5
```

*Figure 79 – Getting the Length of an Array (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

Looping Through Array Elements

for loop can be used to gain access to each element of an array in sequential order, as follows:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Loop Through an Array Using For Loop</title>
</head>
<body>
    <script>
    var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];

    // Iterates over array elements
    for(var i = 0; i < fruits.length; i++){
        document.write(fruits[i] + "<br>"); // Print array element
    }
    </script>
</body>
</html>
```

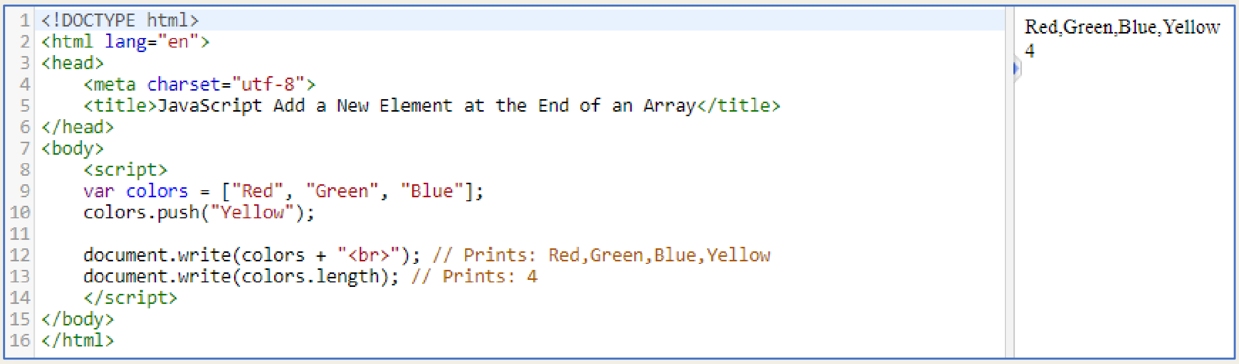
```
Apple
Banana
Mango
Orange
Papaya
```

*Figure 80 – Looping through array elements (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

ECMAScript 6 has introduced a simpler way to iterate over array element, which is for-of loop. In this loop there is no need to initialise and keep track of the loop counter variable (i).

Adding New Elements to an Array

To add a new element at the end of an array, simply use the push() method, as follows:

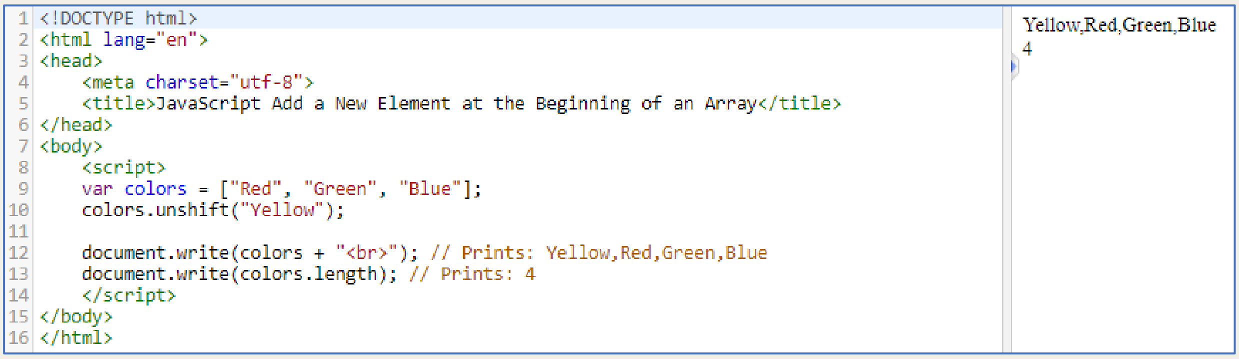


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Add a New Element at the End of an Array</title>
</head>
<body>
    <script>
    var colors = ["Red", "Green", "Blue"];
    colors.push("Yellow");

    document.write(colors + "<br>"); // Prints: Red,Green,Blue,Yellow
    document.write(colors.length); // Prints: 4
    </script>
</body>
</html>
```

```
Red,Green,Blue,Yellow
4
```

***Figure 81 –** Adding New Elements to an Array (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

Likewise, to add a new element at the beginning of an array the unshift() method should be used, as follows:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Add a New Element at the Beginning of an Array</title>
</head>
<body>
    <script>
    var colors = ["Red", "Green", "Blue"];
    colors.unshift("Yellow");

    document.write(colors + "<br>"); // Prints: Yellow,Red,Green,Blue
    document.write(colors.length); // Prints: 4
    </script>
</body>
</html>
```

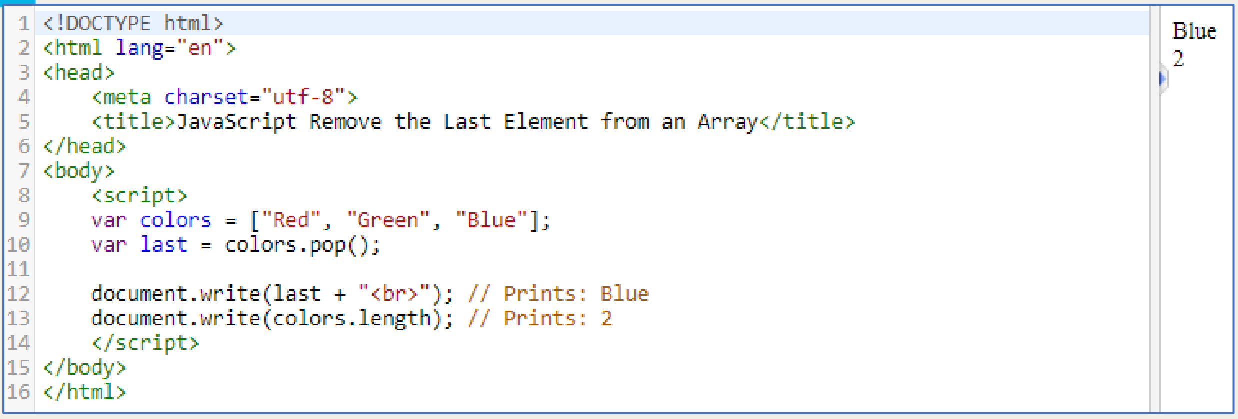
```
Yellow,Red,Green,Blue
4
```

***Figure 82 –** Add a new element at the beginning of an array using the unshift() method (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

Removing Elements from an Array

To eliminate the last element from an array you can use the pop() method. This method returns the value that was popped out.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Remove the Last Element from an Array</title>
</head>
<body>
    <script>
    var colors = ["Red", "Green", "Blue"];
    var last = colors.pop();

    document.write(last + "<br>"); // Prints: Blue
    document.write(colors.length); // Prints: 2
    </script>
</body>
</html>
```

```
Blue
2
```

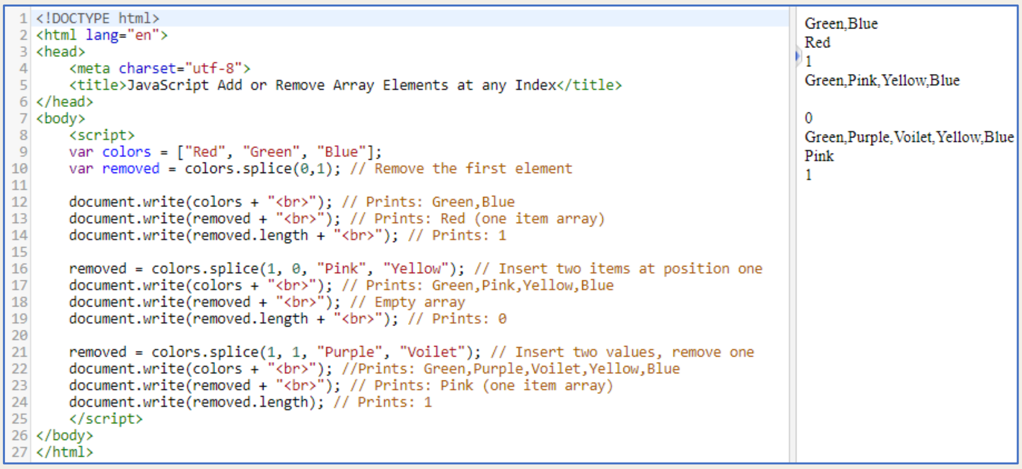
Figure 83 – *Removing Elements from an Array (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

Adding or Removing Elements at any Position

The splice() method is a very flexible array method that allows to add or remove elements from any index, using the syntax `arr.splice(startIndex, deleteCount, elem1, ..., elemN)`

This method takes three parameters: the first is the index at which to start splicing the array, it is required; the second is the number of elements to remove (0 should be used in case the programmer does not want to remove any elements), it is optional; and the third parameter is a set of replacement elements, it is also optional.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Add or Remove Array Elements at any Index</title>
</head>
<body>
    <script>
    var colors = ["Red", "Green", "Blue"];
    var removed = colors.splice(0,1); // Remove the first element

    document.write(colors + "<br>"); // Prints: Green,Blue
    document.write(removed + "<br>"); // Prints: Red (one item array)
    document.write(removed.length + "<br>"); // Prints: 1

    removed = colors.splice(1, 0, "Pink", "Yellow"); // Insert two items at position one
    document.write(colors + "<br>"); // Prints: Green,Pink,Yellow,Blue
    document.write(removed + "<br>"); // Empty array
    document.write(removed.length + "<br>"); // Prints: 0

    removed = colors.splice(1, 1, "Purple", "Voilet"); // Insert two values, remove one
    document.write(colors + "<br>"); //Prints: Green,Purple,Voilet,Yellow,Blue
    document.write(removed + "<br>"); // Prints: Pink (one item array)
    document.write(removed.length); // Prints: 1
    </script>
</body>
</html>
```

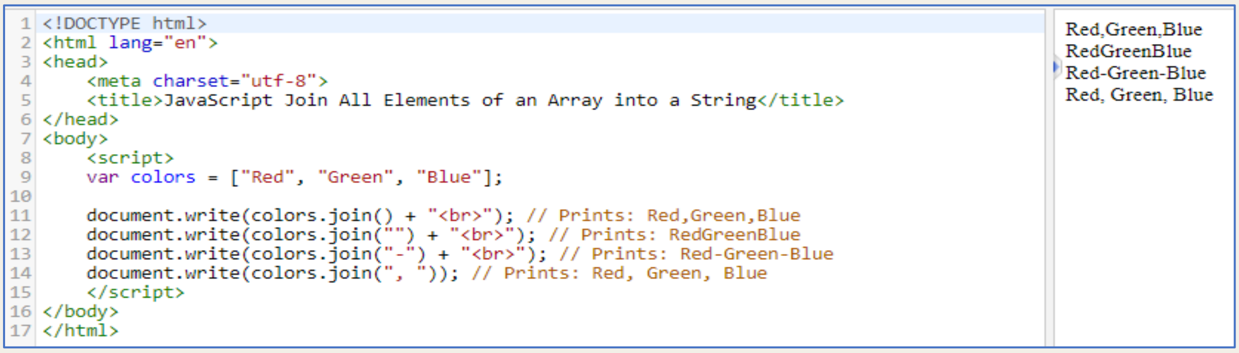
```
Green,Blue
Red
1
Green,Pink,Yellow,Blue

0
Green,Purple,Voilet,Yellow,Blue
Pink
1
```

***Figure 84 –** Adding or Removing Elements at any Position (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

The splice() method returns an array of the deleted elements, or an empty array if no elements were deleted, as it can be seen in *Figure 81*. If the second argument is absent, all elements from the start to the end of the array are removed. Unlike slice() and concat() methods, the splice() method modifies the array on which it is called on.

Creating a String from an Array

There may be situations in which a programmer simply intends to create a string by joining the elements of an array. To do so, he/she can use the join() method. This method takes an optional parameter which is a separator string that is added in between each element. If you omit the separator, then JavaScript will use comma (,) by default.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Join All Elements of an Array into a String</title>
</head>
<body>
    <script>
    var colors = ["Red", "Green", "Blue"];

    document.write(colors.join() + "<br>"); // Prints: Red,Green,Blue
    document.write(colors.join("") + "<br>"); // Prints: RedGreenBlue
    document.write(colors.join("-") + "<br>"); // Prints: Red-Green-Blue
    document.write(colors.join(", ")); // Prints: Red, Green, Blue
    </script>
</body>
</html>
```

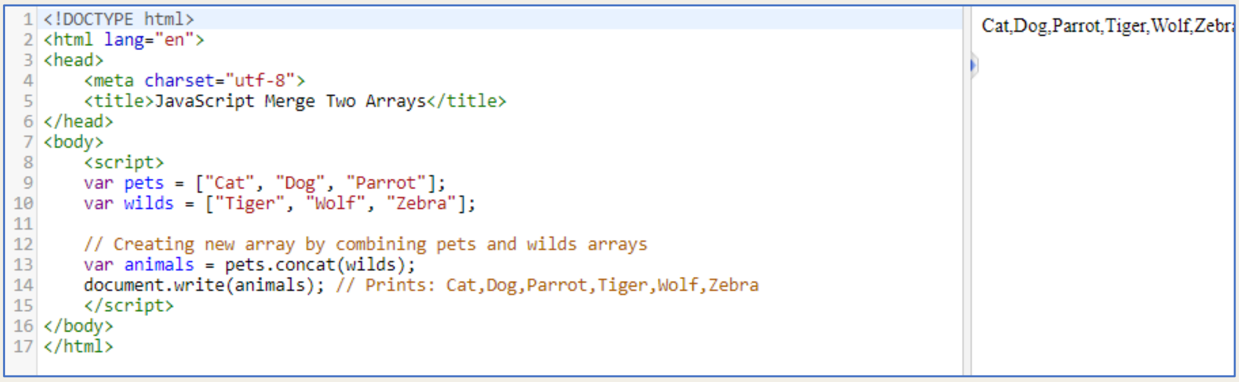
```
Red,Green,Blue
RedGreenBlue
Red-Green-Blue
Red, Green, Blue
```

***Figure 85 –** Creating a String from an Array (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

An array can also be converted to a comma-separated string using the toString(). This method does not allow the separator parameter as join().

Merging Two or More Arrays

The concat() method can be used to combine two or more arrays. This method does not change the prevailing arrays, instead it returns a new array.

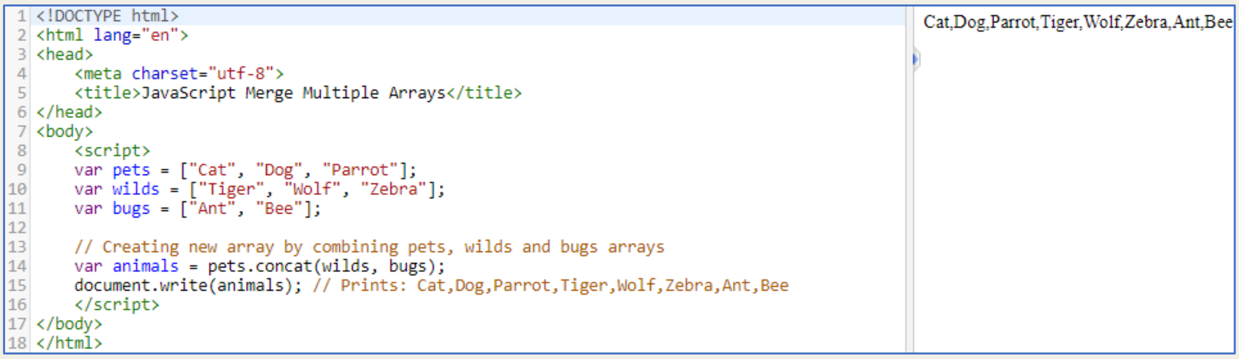


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Merge Two Arrays</title>
</head>
<body>
    <script>
    var pets = ["Cat", "Dog", "Parrot"];
    var wilds = ["Tiger", "Wolf", "Zebra"];

    // Creating new array by combining pets and wilds arrays
    var animals = pets.concat(wilds);
    document.write(animals); // Prints: Cat,Dog,Parrot,Tiger,Wolf,Zebra
    </script>
</body>
</html>
```

```
Cat,Dog,Parrot,Tiger,Wolf,Zebr
```

Figure 86 – *Merging Two or More Arrays (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

The concat() method can take any number of array arguments, so an array can be created from any number of other arrays, as shown in the following example:



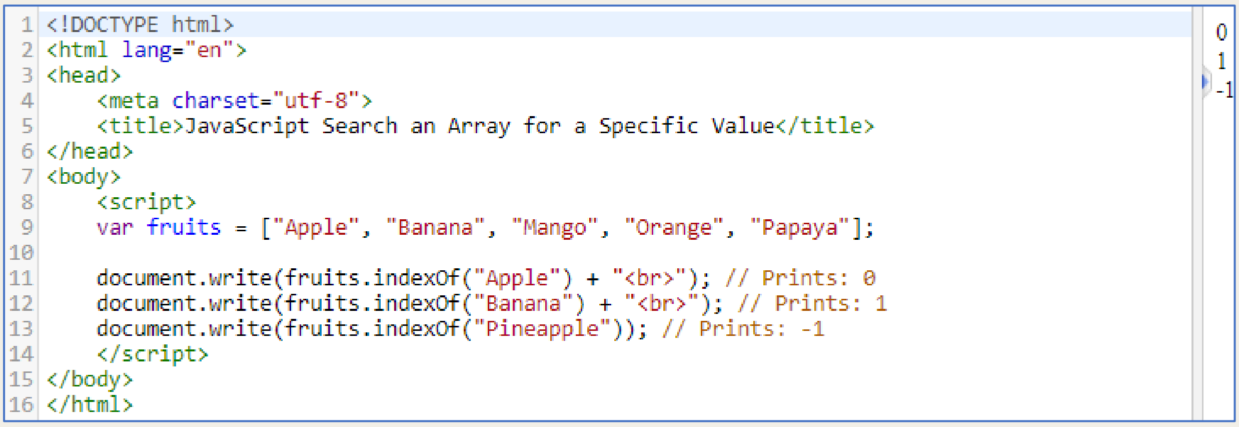
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Merge Multiple Arrays</title>
</head>
<body>
    <script>
    var pets = ["Cat", "Dog", "Parrot"];
    var wilds = ["Tiger", "Wolf", "Zebra"];
    var bugs = ["Ant", "Bee"];

    // Creating new array by combining pets, wilds and bugs arrays
    var animals = pets.concat(wilds, bugs);
    document.write(animals); // Prints: Cat,Dog,Parrot,Tiger,Wolf,Zebra,Ant,Bee
    </script>
</body>
</html>
```

```
Cat,Dog,Parrot,Tiger,Wolf,Zebra,Ant,Bee
```

Figure 87 – *The concat() method (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

Searching Through an Array

The indexOf() and lastIndexOf() methods can be used for searching an array for a specific value. If the value is found, both methods return an index representing the array element. If the value is not found, -1 is returned. The indexOf() method returns the first one found, although the lastIndexOf() returns the last one found. Both methods also recognise an optional integer parameter from index which specifies the index within the array at which to start the search.

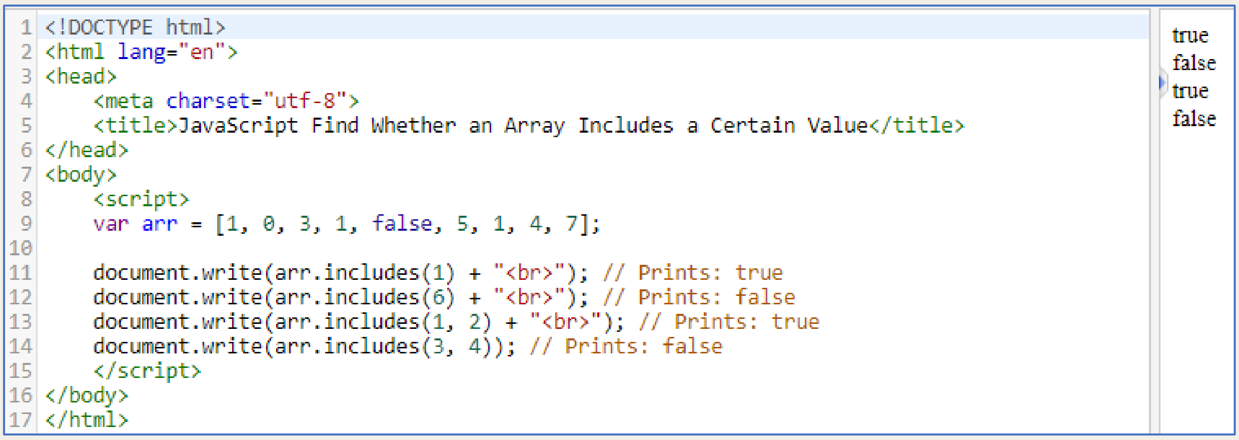


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Search an Array for a Specific Value</title>
</head>
<body>
    <script>
    var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];

    document.write(fruits.indexOf("Apple") + "<br>"); // Prints: 0
    document.write(fruits.indexOf("Banana") + "<br>"); // Prints: 1
    document.write(fruits.indexOf("Pineapple")); // Prints: -1
    </script>
</body>
</html>
```

```
0
1
-1
```

Figure 88 – *Searching through an array (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

includes() method can also be used to find out whether an array involves a certain element or not. This method takes the same parameters as indexOf() and lastIndexOf() methods, but it returns true or false instead of index number.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Find Whether an Array Includes a Certain Value</title>
</head>
<body>
    <script>
    var arr = [1, 0, 3, 1, false, 5, 1, 4, 7];

    document.write(arr.includes(1) + "<br>"); // Prints: true
    document.write(arr.includes(6) + "<br>"); // Prints: false
    document.write(arr.includes(1, 2) + "<br>"); // Prints: true
    document.write(arr.includes(3, 4)); // Prints: false
    </script>
</body>
</html>
```

```
true
false
true
false
```

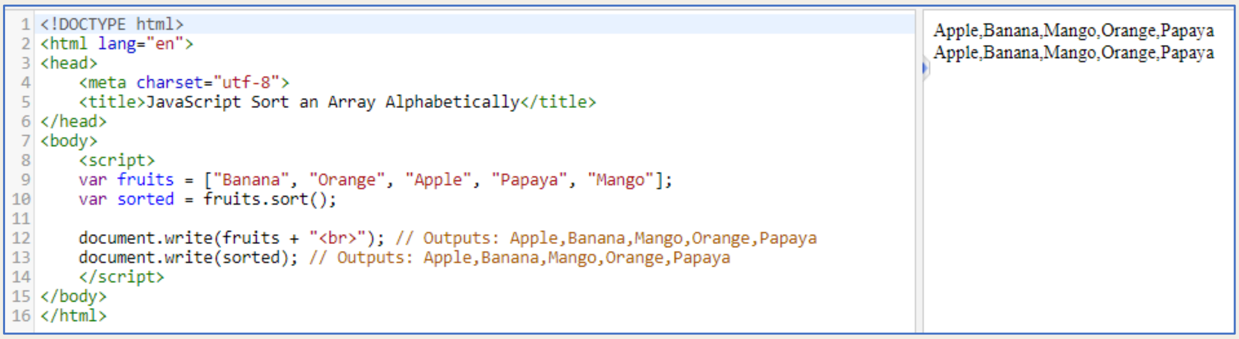
Figure 89 – *Searching through an array using the includes () method (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

To search an array based on certain condition the JavaScript find() method can be used, which has been recently introduced in ES6. This method returns the value of the first element in the array that fulfils the provided testing function. If not, it returns undefined. The find() method simply searches the first element that meets the provided testing function. Still, if the programmer intends to find out all the matched elements, the filter() method can be used. It creates a new array with all the elements that successfully passes the given test, as can be checked in this example.

JavaScript Sorting Arrays

Sorting is a popular task when working with arrays. It can be used, for example, to display the city or country names in alphabetical order. The JavaScript Array object has a built-in method sort() for sorting array elements in alphabetical order.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Sort an Array Alphabetically</title>
</head>
<body>
    <script>
    var fruits = ["Banana", "Orange", "Apple", "Papaya", "Mango"];
    var sorted = fruits.sort();

    document.write(fruits + "<br>"); // Outputs: Apple,Banana,Mango,Orange,Papaya
    document.write(sorted); // Outputs: Apple,Banana,Mango,Orange,Papaya
    </script>
</body>
</html>
```

```
Apple,Banana,Mango,Orange,Papaya
Apple,Banana,Mango,Orange,Papaya
```

Figure 90 – *The built-in method sort() for sorting array elements in alphabetical order (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php)*

To **reverse the order of the elements** of an array, the reverse () method can be used. It reverses an array in such a way that the first array element becomes the last, and vice-versa. The sort() and reverse() method changes the initial array and return a reference to the same array, as it can be checked here.

For **sorting numeric arrays**, it is not advisable to use the sort () method, as it can produce unexpected results. For this purpose, programmers should pass a compare

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

function, as when a compare function is specified, array elements are sorted according to the return value of the compare function.

For example, when comparing a and b:

- If the compare function returns a value less than 0, then *a* comes first.
- If the compare function returns a value greater than 0, then *b* comes first.
- If the compare function returns 0, *a* and *b* remain unchanged regarding each other, but grouped with respect to all other elements.

Therefore, since 5 - 20 = -15 which is less than 0, therefore 5 comes first, likewise 20 - 10 = 10 which is greater than 0, therefore 10 comes before 20, likewise 20 - 75 = -55 which is less than 0, so 20 comes before 75, similarly 50 comes before 75, and so on. This can be checked in this example.

To **find the maximum and minimum value in an Array**, the apply() method in combination with the Math.max() and Math.min() can be used, as follows in this example. The apply() method offers an accessible way to pass array values as arguments to a function that accepts multiple arguments in an array-like manner, but not an array. Then, the resulting statement Math.max.apply(null, numbers) in the example above is equivalent to the Math.max (3, -7, 10, 8, 15, 2).

Finally, to **sort an array of objects** the sort() method can be used. In this example, it is shown how to sort an array of objects by property values.

JavaScript Loops

Loops are applied to perform the same block of code many times if a certain condition is encountered. The key idea behind a loop is to mechanize the repetitive tasks within a program to save the time and effort. JavaScript now supports five different types of loops:

- **while** — loops through a block of code if the condition specified evaluates to *true*.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- **do…while** — loops through a block of code one time; then the condition is evaluated. If the condition is *true*, the statement is repeated if the specified condition is *true*.
- **for** — loops through a block of code until the counter reaches a specified number.
- **for…in** — loops through the properties of an object.
- **for…of** — loops over iterable objects such as arrays, strings, etc.

The while loop

This is the easiest looping statement provided by JS. It loops through a block of code if the specified condition evaluates to *true*. Once the condition fails, the loop is stopped. The generic syntax of the while loop is:

```
while(condition) {
  // Code to be executed
}
```

In this example, a loop that keeps on running as long as the variable *i ≤ 5.* As it can be checked, i will increase by 1 each time the loop runs. Programmers must ensure that the condition specified in the loop can eventually go false. If not, the loop will never stop iterating (infinite loop).

The do…while loop

The do-while loop is a variant of the while loop, which assesses the condition at the end of each loop iteration. With a do…while loop, the block of code is executed once, and then the condition is evaluated, if the condition is *true*, the statement is repeated if the specified condition evaluated to is *true*. Its common syntax is:

```
do {
   // Code to be executed
}
while(condition);
```

The JavaScript code in this example defines a loop that starts with *i=1.* It will then produce the output and increase the value of variable i by 1. After that the condition is evaluated, and the loop will continue to run if *i ≤ 5.*

The for loop

It repeats a block of code if a certain condition is met. It is normally used to implement a block of code for certain number of times. Its syntax is:

```
for(initialization; condition; increment) {
   // Code to be executed
}
```

The parameters of the **for loop** statement have the following implications:

- **initialisation** — it is used to set the counter variables and evaluated once entirely before the first execution of the body of the loop.
- **condition** — it is evaluated at the beginning of each iteration. If it evaluates to *true*, the loop statements achieve. If it evaluates to *false*, the execution of the loop ends.
- **increment** — it updates the loop counter with a new value each time the loop runs.

This example defines a loop that begins with i=1. The loop will go on until the value of *i ≤ 5*. The variable i will increase by 1 each time the loop runs.

The for...in Loop

It is a special type of a loop that repeats over the properties of an object, or the elements of an array. Its syntax is:

```
for(variable in object) {
   // Code to be executed
}
```

The loop counter i.e., variable in the for-in loop is a string, not a number. It comprises the name of present property or the index of the current array element.

This example demonstrates how to loop through all properties of a JS object.

The for...of Loop

ES6 presents a new for-of loop which allows to iterate over arrays or other iterable objects (e.g., strings) very simply. Additionally, the code inside the loop is executed for each element of the iterable object. This example shows how to loop trough arrays and strings using this loop.

JavaScript Functions

A function is a set of statements that accomplish specific tasks and can be kept and maintained independently. Functions give a way to create reusable code packages which are more manageable and easier to debug. Some advantages of using functions are as follows:

- **Functions decreases the repetition of code within a program** — Function allows to extract frequently used block of code into a single component. This way it is possible to perform the same task by calling this function wherever needed within a script without having to copy and paste the same block of code over and over.

- **Functions makes the code much easier to maintain** — Since a function created once can be applied many times, any changes made inside a function are automatically implemented at all the places without affecting the respective files.
- **Functions makes it easier to get rid of the errors** — When the program is split into functions, if any error arise, the programmer knows precisely what function is causing the error and where to spot it. Consequently, fixing errors becomes much simpler.

Defining and Calling a Function

The declaration of a function starts with the function keyword, followed by the name of the function to be created, then followed by parentheses, and lastly by the function's code between curly brackets {}. To declare a function, the following syntax applies:

```
function functionName() {
   // Code to be executed
}
```

This simple example displays a “Hello” message.

Once a function is defined, it can be called from anywhere in the document, by typing its name followed by a set of parentheses, like sayHello() in the aforementioned example.

Adding Parameters to Functions

Parameters can be specified when defining a function to accept input values at run time. The parameters act like placeholder variables within a function; they are replaced at run time by the values (known as *argument*) offered to the function at the time of request.

Parameters are set on the first line of the function inside the set of parentheses, as follows:

```
function functionName(parameter1, parameter2, parameter3) {
    // Code to be executed
}
```

The displaySum() function in this example gets two numbers as arguments, simply add them as one and then display the result in the browser. There is no limit for defining parameters. Nevertheless, for each parameter specified, a corresponding argument requires to be passed to the function when it is called, if not its value becomes undefined.

Default Values for Function Parameters

With ES6, it is possible to specify default values to the function parameters. This implies that if no arguments are provided to function when it is called, these default parameters values will be used. This example is quite explanative regarding how valuable this feature is, since to achieve the same result, the prior procedure was this one.

Returning Values from a Function

A function can return a value back to the script that called the function as a result, by utilizing the return statement. The value may be of any type (i.e., arrays and objects). The return statement typically placed as the last line of the function before the closing curly bracket and ends it with a semicolon, as displayed here.

A function can does not return multiple values. Still, similar results can be obtained by returning an array of values, as exhibited here.

Working with Function Expressions

The syntax used before to create functions is called **function declaration**. There is another syntax for building a function – function expression. Once stored in a variable, the variable can be utilized as a function. The syntax of the function declaration and function expression seems very similar, but they vary in the way they are evaluated. As observed in the previous example, the function expression gave an exception when it was invoked before it is defined, but the function declaration executed effectively. JS analyses declaration function before the program executes. Hence, it does not make a difference if the program invokes the function before it is defined, as JavaScript has elevated the function to the top of the current scope in the background. The function expression is not assessed until it is assigned to a variable; consequently, it is still undefined when invoked.

Understanding the Variable Scope

Variables can be declared anywhere in JS. However, the location of the declaration will establish the extent of a variable's availability within the JavaScript program – this process is also called **variable scope**. By default, variables declared within a function have local scope, which means they cannot be viewed or controlled from outside of that function, as shown here. Though, any variables declared in a program outside of a function has global scope, wherever that script is located concerning the function, as it can be checked here.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

JavaScript Objects

JavaScript is an object-based language and there nearly everything is an object or acts like one. Thus, to work with JS successfully and efficiently, programmers need to understand how objects work, as well as how to create objects and use them.

A JavaScript object is simply a collection of named values. They are typically referred to as properties of the object. Being an array a collection of values, in which each value has an index (a numeric key) starting from zero and incrementing by one for each value. An object is like an array, but the difference is that the programmer defines the keys, (name, age, gender, etc.).

Creating an object

Programmers can create objects with curly brackets, including a voluntary list of properties as well. A property can be "key:value" pair, in which the key (or *property name*) is always a string, and value (or *property value*) can be any data type (strings, numbers, Booleans, arrays, functions, etc.). Moreover, properties with functions as their values are frequently called methods to distinguish them from other features. A JS object may be as this. This example creates an object called person, which has 3 properties (name, age and gender) and one method displayName(). This method shows the value of this.name, which agrees to person.name. This is the easiest and ideal way to create a new object in JF, which is known as **object literals syntax**. The property names generally do not need to be quoted except if they are reserved words, or if they contain spaces or special characters (anything other than letters, numbers, and the _ and $ characters), or if they start with a number, as shown here.

Accessing Object's Properties

In order to access or get the value of a property, the dot (.) can be applied, as well as the square bracket ([]) notation, as this example shows. The dot notation is simpler to read and write, but it cannot always be used. If the name of the property is not valid (for

spel CITIZENS IN POWER Center for Social Innovation CDDR ZAUG gGmbH 4Action social hackers academy EPE

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

example, if it contains special characters or spaces), the dot notation cannot be used, but the bracket notation instead. It presents much more flexibility than dot notation and additionally allows to identify property names as variables as a replacement for string literals.

Looping Through Object's Properties

Programmers can iterate through the key-value pairs of an object using the for...in loop. It is specifically enhanced for iterating over object's properties, as seen here.

Setting Object's Properties

Likewise, new properties can be set, or the existing one can be updated by using the dot (.) or bracket ([]) notation, as demonstrated here.

Deleting Object's Properties

The delete operator can be applied to entirely delete properties from an object. Removing is the only way to really get rid of a property. Adjusting the property to undefined or null simply changes the value of the property, it does not remove property from the object. So, it has no effect on variables or declared functions. Nevertheless, programmers should avoid delete operator for the purpose of deleting an array element, as it does not modify the array's length, it just drops a hole in the array.

Calling Object's Methods

An object's method can be accessed the same way as one would access properties—using the dot notation or applying the square bracket notation, as can be checked here.

Manipulating by Value vs. Reference

JS objects are reference types, which means that when the programmer makes copies of them, they are simply copying the references to that object, while primitive values like strings and numbers are assigned or copied as a whole value. This example is quite demonstrative of this idea. As it could be observed, a copy of a variable message has

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

been done and it changed the value of that same copy. Both variables remain distinct and separate. However, if the same principle is applied to an object, a different result will be gathered. So, any changes done to the variable user also interfere with the person variable, as both variables refer to the same object. Therefore, merely copying the object does not really clone it but copies the reference to that object.

5.2. JavaScript & DOM

What is the Document Object Model (DOM)?

DOM is created by the browser when a web page is loaded in HTML or XML documents. It is used to define the logical structure of these documents and to access and alter their elements.

The HTML DOM refers to the Document Object Model of the HTML documents, whereas the XML DOM refers to the Document Object Model of the XML documents. In this subchapter, we will focus on the HTML DOM which can be used to access and manipulate HTML documents via JavaScript.

The DOM is constructed as a hierarchical tree of objects that include all parts of an HTML document such as elements, attributes, text, etc. These individual objects of the tree are also known as nodes, and they are comprised of parent and child nodes. In graphical form, it will look something like the diagram provided below.

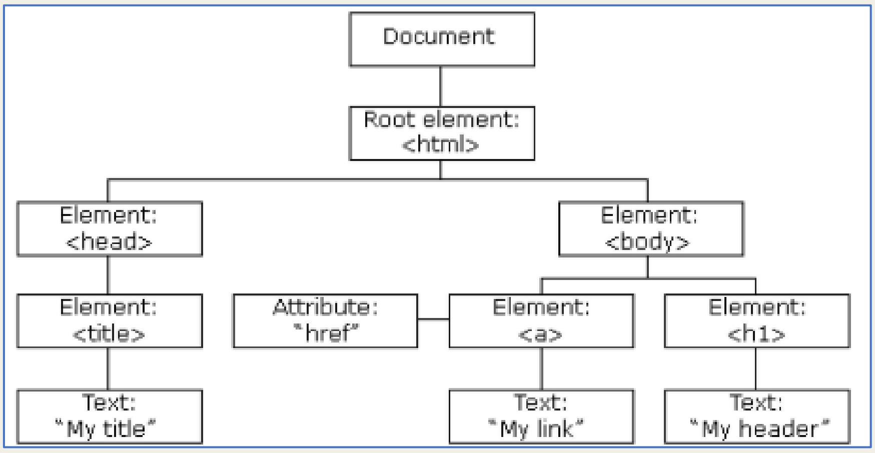


*Figure 91 – HTML DOM Tree of Objects (**Source**: https://www.w3schools.com/js/js_htmldom.asp)*

Within the HTML DOM, JavaScript can do the following:

- change all the HTML elements in the page

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- change all the HTML attributes in the page
- change all the CSS styles in the page
- remove existing HTML elements and attributes
- add new HTML elements and attributes
- react to all existing HTML events in the page
- create new HTML events in the page

JavaScript DOM Selectors

Selecting DOM Elements in JavaScript

JavaScript is used to get or modify the content or value of the HTML elements of the web page and apply some special effects such as animations or hide. To be able to perform any action, you need to find or select the target HTML element.

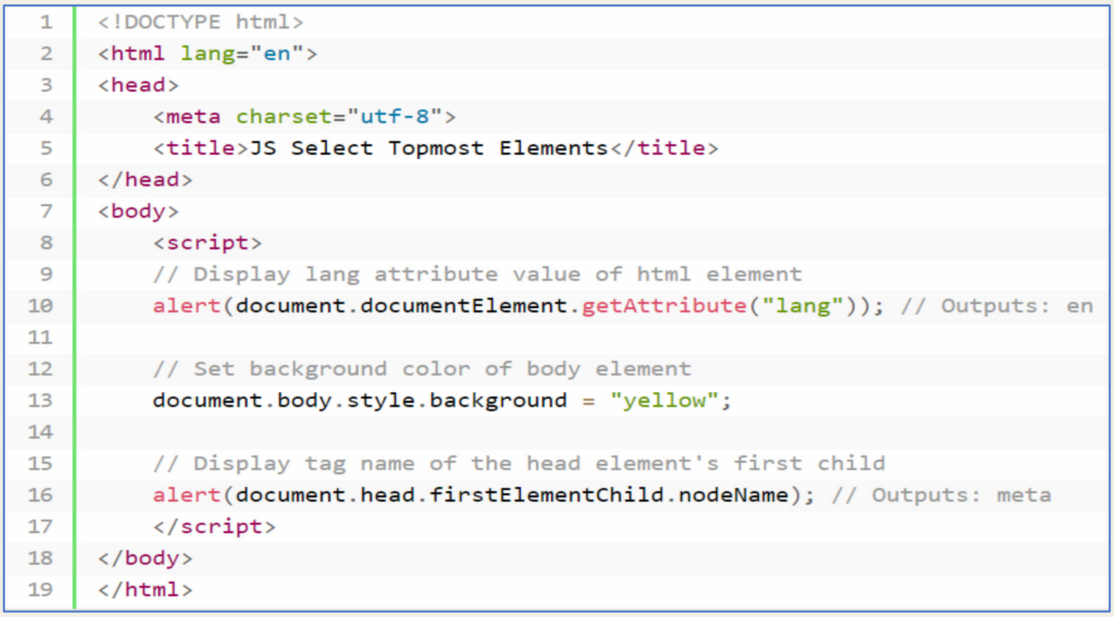
We will go through some of the most common ways of selecting elements on a page and manipulating them with JavaScript.

Selecting the Topmost Elements

The topmost elements can be accessed directly as document properties.

For instance, to access the <html> element, use the document.documentElement property. For the <head> element, you can use the document.head property and for the <body> element, the document.body property.

Let’s see an example to understand all of this a bit better.

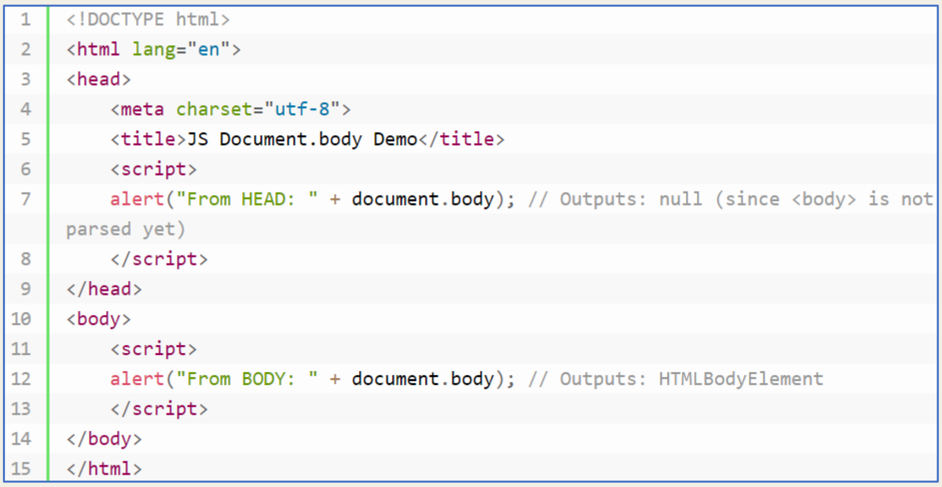


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Topmost Elements</title>
</head>
<body>
    <script>
    // Display lang attribute value of html element
    alert(document.documentElement.getAttribute("lang")); // Outputs: en

    // Set background color of body element
    document.body.style.background = "yellow";

    // Display tag name of the head element's first child
    alert(document.head.firstElementChild.nodeName); // Outputs: meta
    </script>
</body>
</html>
```

*Figure 92 - Topmost Elements Example (**Source**:* https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-selectors.php)

* It is important to note that the document.body should not be used before the <body> element since it will return null. The program needs to go through the <body> element first to access the document.body property.

Let's look at this example to understand why the <body> will be null:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Document.body Demo</title>
    <script>
    alert("From HEAD: " + document.body); // Outputs: null (since <body> is not parsed yet)
    </script>
</head>
<body>
    <script>
    alert("From BODY: " + document.body); // Outputs: HTMLBodyElement
    </script>
</body>
</html>
```

***Figure 93** - Topmost Elements Example (**Source**:* https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-selectors.php)

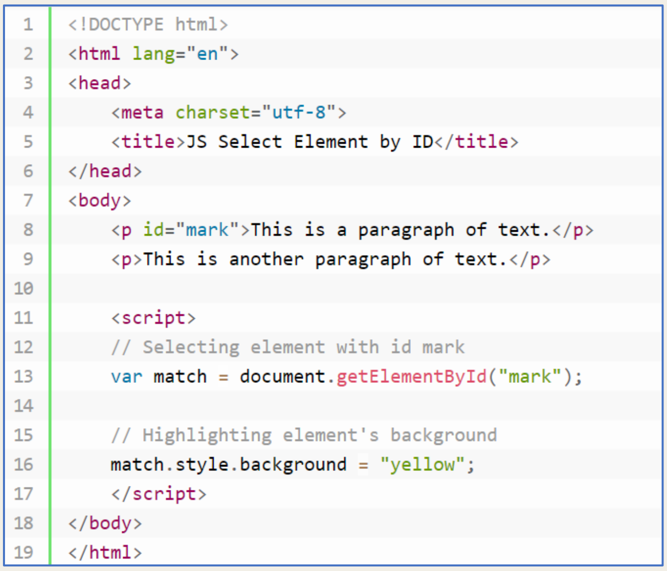
The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

This example demonstrates what we saw in the beginning about the hierarchical relationships that exist between nodes. You need to be mindful that in order to access the document.body property, you will have to start from the <body> element to avoid null values.

Selecting Elements by ID

If you want to find or select an HTML element, the easiest way is to select it is based on its unique ID. You can do this with the getElementById() method.

In the following example, the element that has an ID attribute id="mark" is selected and highlighted:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Element by ID</title>
</head>
<body>
    <p id="mark">This is a paragraph of text.</p>
    <p>This is another paragraph of text.</p>

    <script>
    // Selecting element with id mark
    var match = document.getElementById("mark");

    // Highlighting element's background
    match.style.background = "yellow";
    </script>
</body>
</html>
```

Figure 94** - Selecting Elements by ID Example (**Source: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-selectors.php)

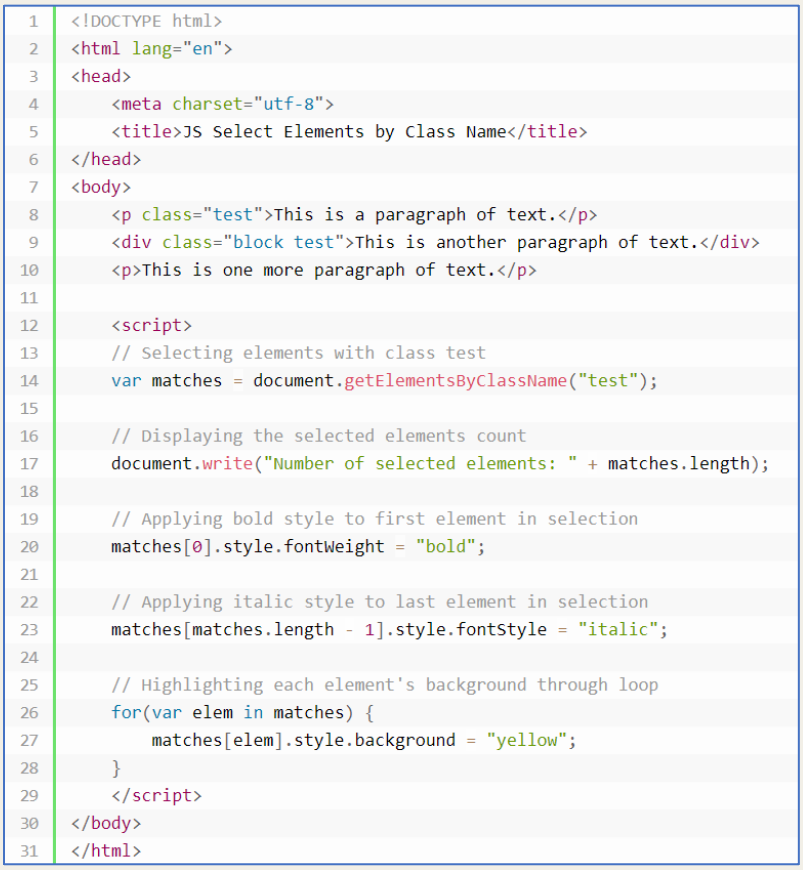
The getElementById() method is used to return the element as an object if a matching element is found. Otherwise, it will return null.

* Keep in mind that any HTML element can have an id attribute, which must be a **unique value within a page**. This essentially means that no two elements can have the same id.

Selecting Elements by Class Name

If you want to select all the elements with specific class names, use the getElementsByClassName() method. It will return an array-like object of all child elements which have all the given class names.

Let’s look at an example:



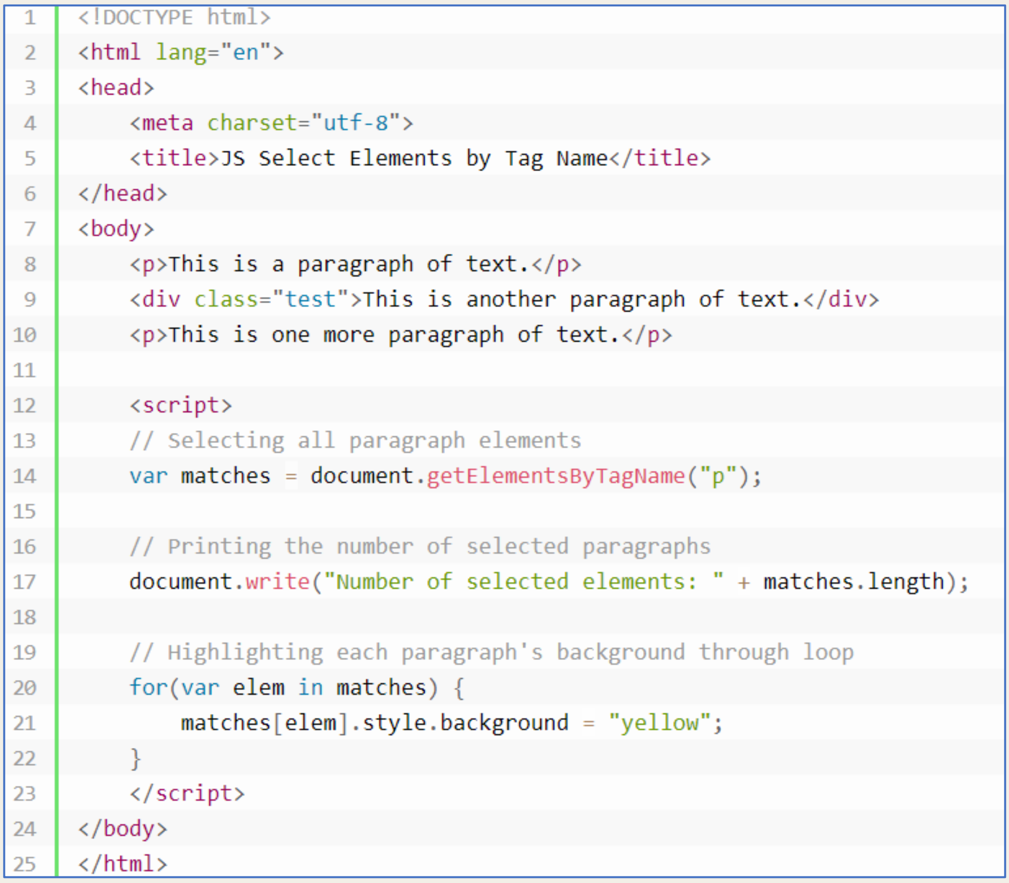
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Elements by Class Name</title>
</head>
<body>
    <p class="test">This is a paragraph of text.</p>
    <div class="block test">This is another paragraph of text.</div>
    <p>This is one more paragraph of text.</p>

    <script>
    // Selecting elements with class test
    var matches = document.getElementsByClassName("test");

    // Displaying the selected elements count
    document.write("Number of selected elements: " + matches.length);

    // Applying bold style to first element in selection
    matches[0].style.fontWeight = "bold";

    // Applying italic style to last element in selection
    matches[matches.length - 1].style.fontStyle = "italic";

    // Highlighting each element's background through loop
    for(var elem in matches) {
        matches[elem].style.background = "yellow";
    }
    </script>
</body>
</html>
```

Figure 95 – *Selecting Elements by Class Name Example (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-selectors.php)*

Selecting Elements by Tag Name

If you want to select elements by their tag name, use the getElementsByTagName() method. This method will also return an array-like object of all child elements which have the given tag name.

Let’s look at an example to understand this a bit better:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Elements by Tag Name</title>
</head>
<body>
    <p>This is a paragraph of text.</p>
    <div class="test">This is another paragraph of text.</div>
    <p>This is one more paragraph of text.</p>

    <script>
    // Selecting all paragraph elements
    var matches = document.getElementsByTagName("p");

    // Printing the number of selected paragraphs
    document.write("Number of selected elements: " + matches.length);

    // Highlighting each paragraph's background through loop
    for(var elem in matches) {
        matches[elem].style.background = "yellow";
    }
    </script>
</body>
</html>
```

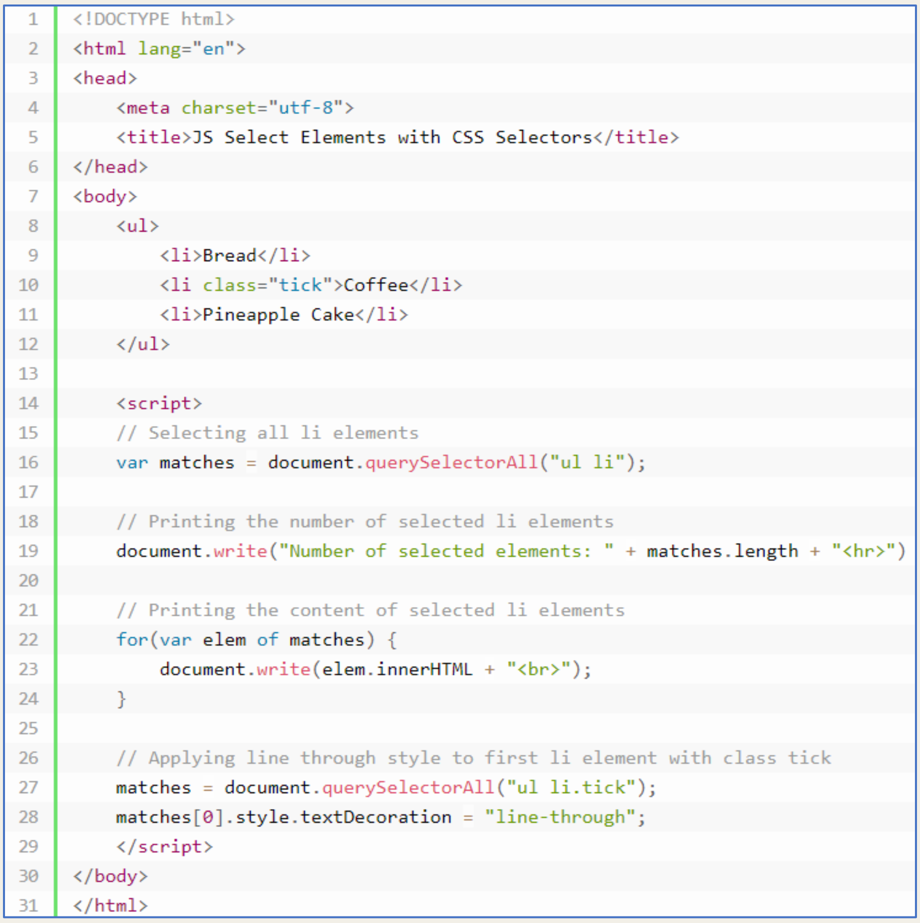
***Figure 96** - Selecting Elements by Tag Name Example (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-selectors.php)*

Selecting Elements with CSS Selectors

CSS Selectors offer a very powerful and efficient way to select HTML elements in a document. To select elements that match the specified CSS Selector, you can use the querySelectorAll() method.

This method will return a list of all the elements that match the specified selectors.

Follow the example below to see how you can examine this list like an array:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Elements with CSS Selectors</title>
</head>
<body>
    <ul>
        <li>Bread</li>
        <li class="tick">Coffee</li>
        <li>Pineapple Cake</li>
    </ul>

    <script>
    // Selecting all li elements
    var matches = document.querySelectorAll("ul li");

    // Printing the number of selected li elements
    document.write("Number of selected elements: " + matches.length + "<hr>")

    // Printing the content of selected li elements
    for(var elem of matches) {
        document.write(elem.innerHTML + "<br>");
    }

    // Applying line through style to first li element with class tick
    matches = document.querySelectorAll("ul li.tick");
    matches[0].style.textDecoration = "line-through";
    </script>
</body>
</html>
```

Figure 97 *- Selecting Elements with CSS Selectors Example (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-selectors.php)*

* Please note that the querySelectorAll() method supports CSS pseudo-classes such as :first-child, :last-child, :hover, etc. However, this method will always return an empty list for CSS pseudo-elements like ::before, ::after, ::first-line, etc.

JavaScript DOM Styling

Styling DOM Elements in JavaScript

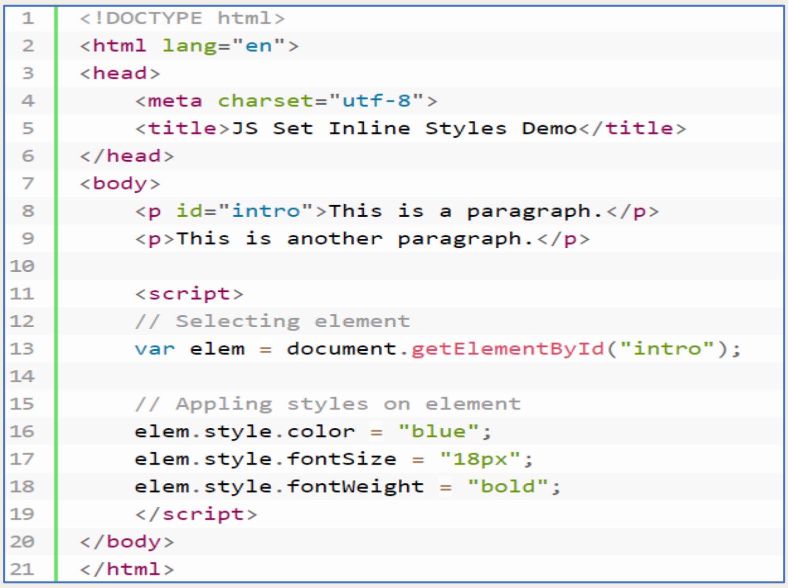
You can also change the visual presentation of HTML documents in a dynamic way by using JavaScript to apply different styles to HTML elements. Almost all element styles can be set such as fonts, colours, margins, borders, background images, text alignment, width and height, position, and so on.

Here, we will go through various methods that can be used to set styles in JavaScript.

Setting Inline Styles on Elements

The style attribute is used to apply inline styles directly to the specific HTML element. The style property is used in JavaScript to get or set the inline style of an element.

In the following example, colour and font properties will be set for an element with id="intro":



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Set Inline Styles Demo</title>
</head>
<body>
    <p id="intro">This is a paragraph.</p>
    <p>This is another paragraph.</p>

    <script>
    // Selecting element
    var elem = document.getElementById("intro");

    // Appling styles on element
    elem.style.color = "blue";
    elem.style.fontSize = "18px";
    elem.style.fontWeight = "bold";
    </script>
</body>
</html>
```

Figure 98 *- Inline Styles on Elements Example (****Source:*** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-styling.php)

Naming Conventions of CSS Properties in JavaScript

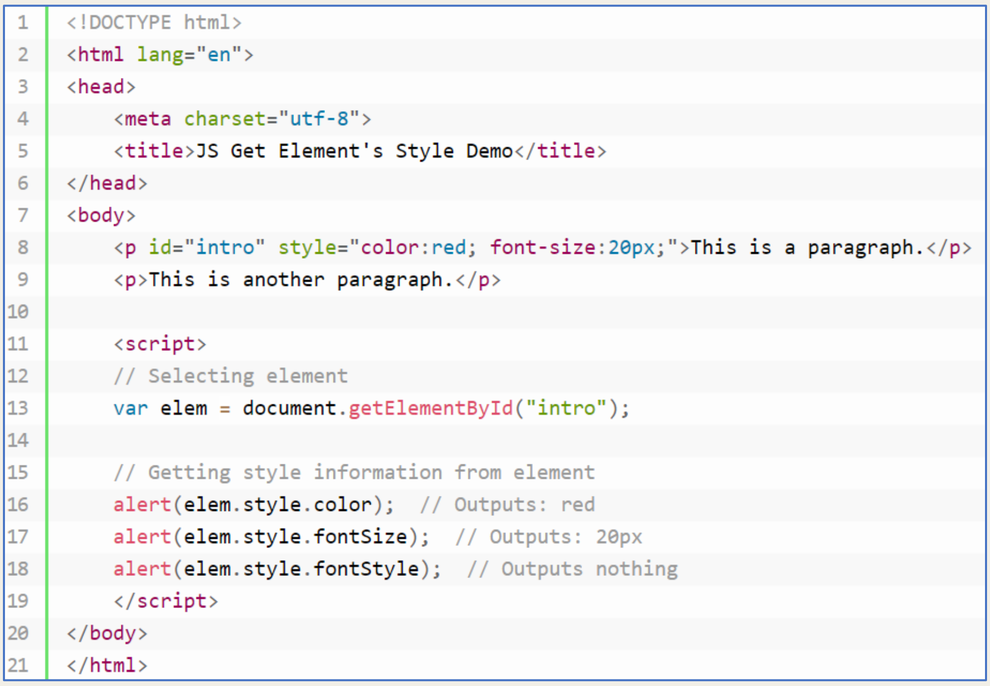
It is important to mention that many of CSS properties contain hyphens (-) in their names such as font-size, background-image, text-decoration, etc. However, in JavaScript, the hyphen is a reserved operator that signifies a minus sign. Therefore, it is not possible to write an expression in this way: elem.style.font-size.

To overcome this issue, CSS property names in JavaScript that contain one or more hyphens are converted to intercapatisalised style words. This essentially means that the hyphens are removed and the first letter after the hyphen is capitalised. For instance, the CSS property font-size become fontSize in DOM property.

Getting Style Information from Elements

The style property is also used to get the styles applied to HTML elements.

The example below will get style information from the element with id="intro":

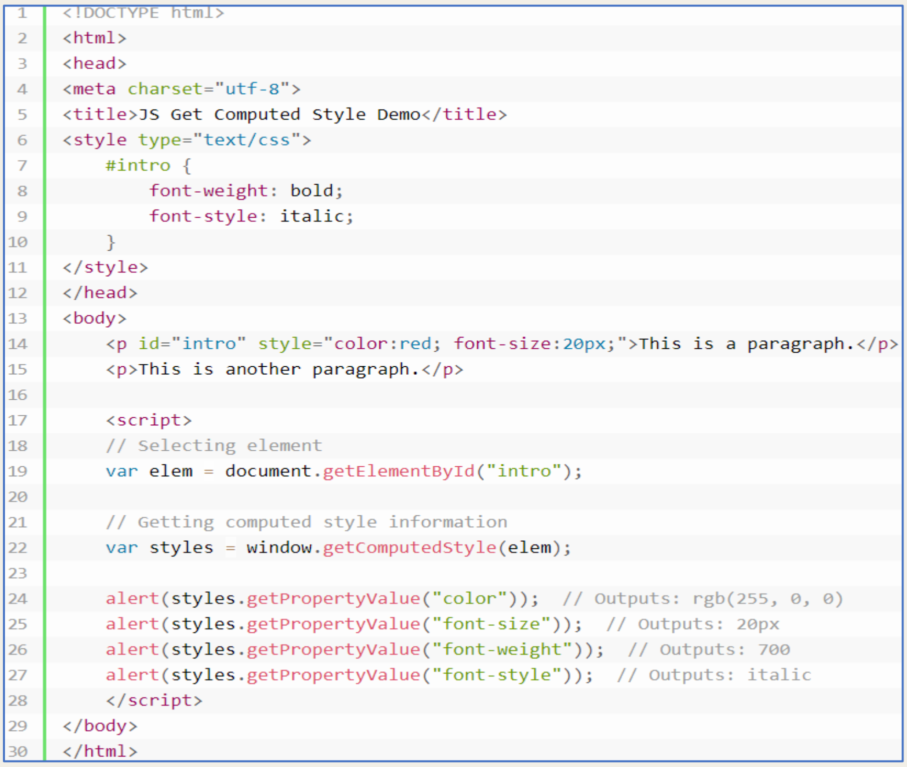


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Get Element's Style Demo</title>
</head>
<body>
    <p id="intro" style="color:red; font-size:20px;">This is a paragraph.</p>
    <p>This is another paragraph.</p>

    <script>
    // Selecting element
    var elem = document.getElementById("intro");

    // Getting style information from element
    alert(elem.style.color);  // Outputs: red
    alert(elem.style.fontSize);  // Outputs: 20px
    alert(elem.style.fontStyle);  // Outputs nothing
    </script>
</body>
</html>
```

Figure 99 *- Style property – Example (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-styling.php)*

The style property isn't the most useful when it comes to getting style information from the elements since it only returns the style rules that are set in the element's style attribute and not those that come from elsewhere such as style rules in the embedded style sheets, or external style sheets.

If you want to get the values of all CSS properties that are used to render an element you can use the window.getComputedStyle() method, as shown in the following example:



```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>JS Get Computed Style Demo</title>
<style type="text/css">
    #intro {
        font-weight: bold;
        font-style: italic;
    }
</style>
</head>
<body>
    <p id="intro" style="color:red; font-size:20px;">This is a paragraph.</p>
    <p>This is another paragraph.</p>

    <script>
    // Selecting element
    var elem = document.getElementById("intro");

    // Getting computed style information
    var styles = window.getComputedStyle(elem);

    alert(styles.getPropertyValue("color"));  // Outputs: rgb(255, 0, 0)
    alert(styles.getPropertyValue("font-size"));  // Outputs: 20px
    alert(styles.getPropertyValue("font-weight"));  // Outputs: 700
    alert(styles.getPropertyValue("font-style"));  // Outputs: italic
    </script>
</body>
</html>
```

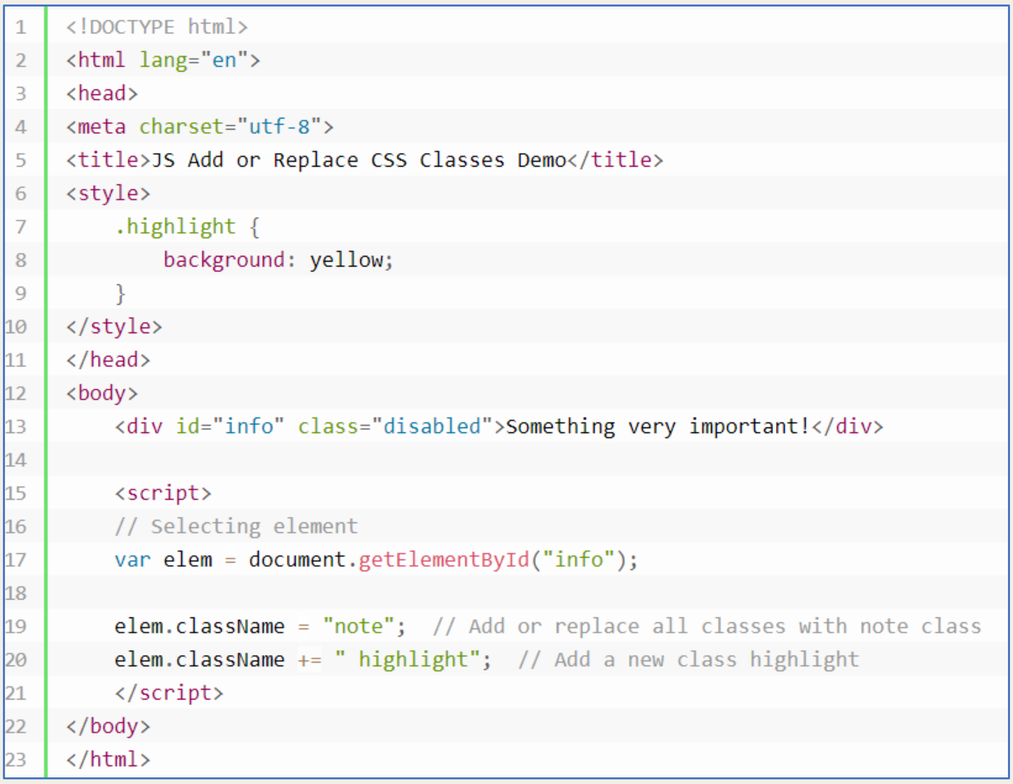
Figure 100 - *window.getComputedStyle() – Example* (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-styling.php)

* Keep in mind that the value 700 for the CSS property font-weight is the same as the keyword bold. The colour keyword red is the same as rgb(255,0,0), which is the rgb notation of a colour.

Adding CSS Classes to Elements

Another way to get or set CSS classes to HTML elements is by using the className property. Class is a reserved word in JavaScript; thus, JavaScript uses the className property to refer to the value of the HTML class attribute.

Let's look at the following example to learn how to add a new class, or replace all existing classes to a <div> element with id="info":

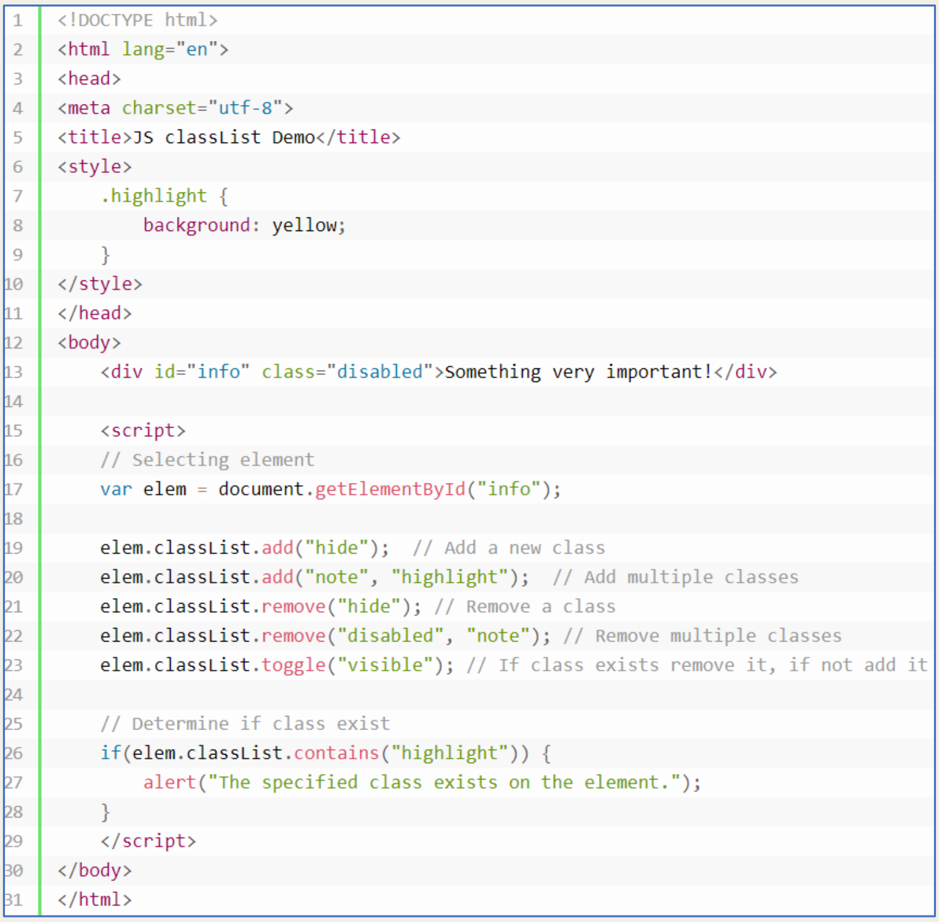


```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS Add or Replace CSS Classes Demo</title>
<style>
    .highlight {
        background: yellow;
    }
</style>
</head>
<body>
    <div id="info" class="disabled">Something very important!</div>

    <script>
    // Selecting element
    var elem = document.getElementById("info");

    elem.className = "note";  // Add or replace all classes with note class
    elem.className += " highlight";  // Add a new class highlight
    </script>
</body>
</html>
```

***Figure 101** - Adding Classes to Elements Example (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-styling.php)*

An even better way to work with CSS classes is by using the classList property to get, set or remove CSS classes easily from an element. This property is supported in all major browsers except Internet Explorer before version 10.

Let’s see an example of this property:



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS classList Demo</title>
<style>
    .highlight {
        background: yellow;
    }
</style>
</head>
<body>
    <div id="info" class="disabled">Something very important!</div>

    <script>
    // Selecting element
    var elem = document.getElementById("info");

    elem.classList.add("hide");  // Add a new class
    elem.classList.add("note", "highlight");  // Add multiple classes
    elem.classList.remove("hide"); // Remove a class
    elem.classList.remove("disabled", "note"); // Remove multiple classes
    elem.classList.toggle("visible"); // If class exists remove it, if not add it

    // Determine if class exist
    if(elem.classList.contains("highlight")) {
        alert("The specified class exists on the element.");
    }
    </script>
</body>
</html>
```

__Figure 102__ - classList property - Adding Classes to Elements Example (__Source:__ https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-styling.php)

JavaScript DOM Get Set Attributes

Working with Attributes

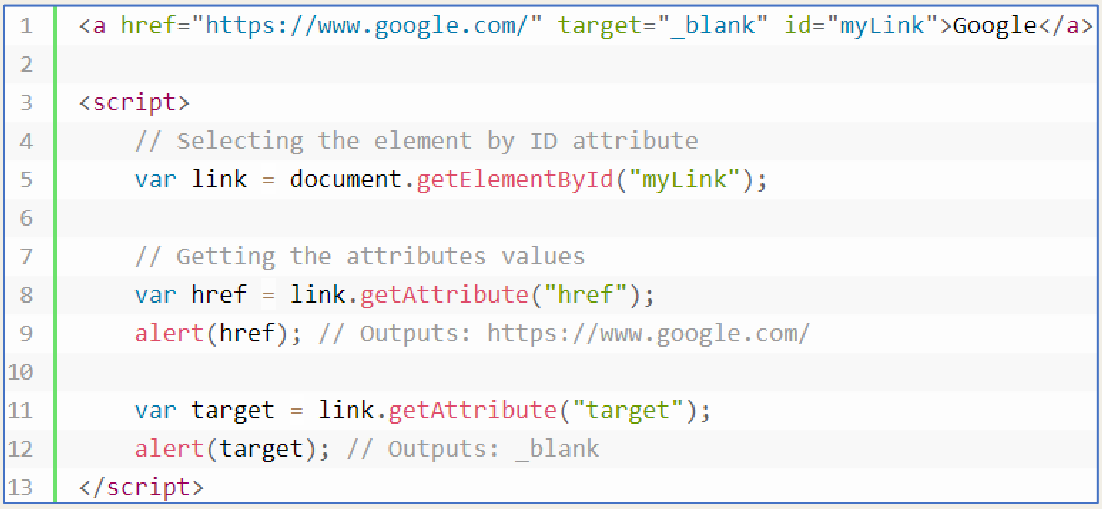
Attributes are special words used inside the start tag of an HTML element to control the tag’s behaviour or provide more information about the tag.

In this section, we will go through several methods of adding, removing or changing an HTML element’s attribute.

Getting Element's Attribute Value

To get the current value of an element’s attribute, you can use the getAttribute() method. If that particular attribute is not found on the element, it will return null.

Let’s see the following example below:



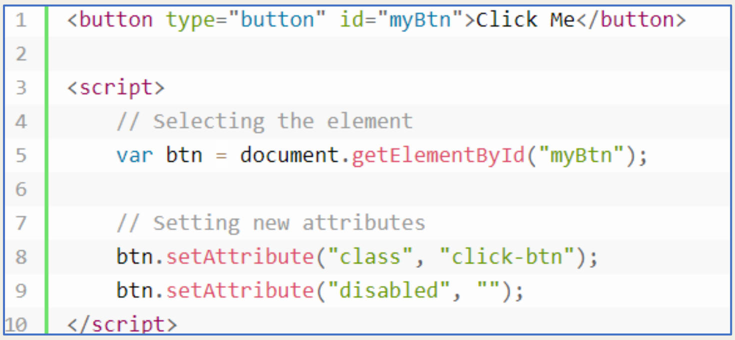
```
<a href="https://www.google.com/" target="_blank" id="myLink">Google</a>

<script>
    // Selecting the element by ID attribute
    var link = document.getElementById("myLink");

    // Getting the attributes values
    var href = link.getAttribute("href");
    alert(href); // Outputs: https://www.google.com/

    var target = link.getAttribute("target");
    alert(target); // Outputs: _blank
</script>
```

Figure 103 *- getAttribute() method – Getting Element’s Attribute Value Example (**Source**:* https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-get-set-attributes.php)

Setting Attributes on Elements

If you want to set an attribute on a specified element, you can use the setAttribute() method. If the attribute already exists on the element, the value will be updated. If not, a new attribute will be added with specified name and value.

In the following example, we will add a class and a disabled attribute to the <button> element:

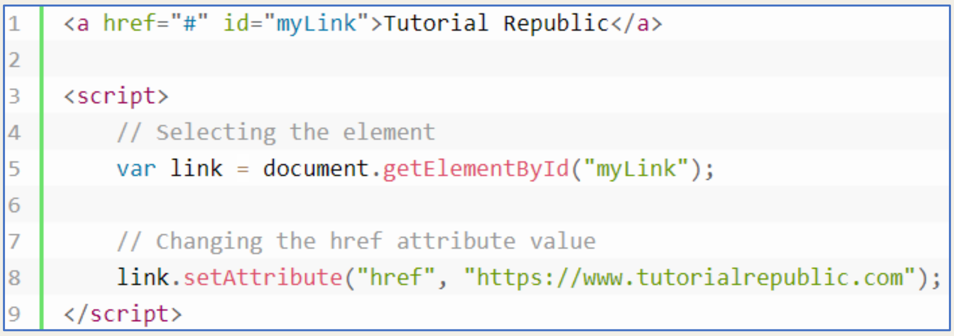


```
<button type="button" id="myBtn">Click Me</button>

<script>
    // Selecting the element
    var btn = document.getElementById("myBtn");

    // Setting new attributes
    btn.setAttribute("class", "click-btn");
    btn.setAttribute("disabled", "");
</script>
```

Figure 104 *- setAttribute() method – Setting Attributes on Elements Example (**Source:*** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-get-set-attributes.php)

If you want to update or change the value of an existing attribute on an element, you can also use the setAttribute() method.

Let’s see an example that will update the value of the existing href attribute of an anchor (<a>) element:



```
<a href="#" id="myLink">Tutorial Republic</a>

<script>
    // Selecting the element
    var link = document.getElementById("myLink");

    // Changing the href attribute value
    link.setAttribute("href", "https://www.tutorialrepublic.com");
</script>
```

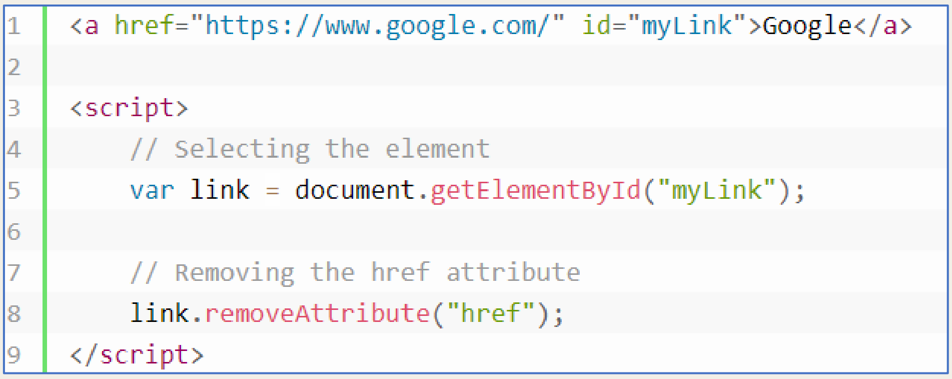
Figure 105 *- setAttribute() method – Updating or Changing Attributes on Elements Example (**Source:*** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-get-set-attributes.php)

Removing Attributes from Elements

To remove an attribute from a specific element, you can use the removeAttribute() method.

Remember the href attribute that we changed from the anchor element; we are now going to remove it in the following example:

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<a href="https://www.google.com/" id="myLink">Google</a>

<script>
    // Selecting the element
    var link = document.getElementById("myLink");

    // Removing the href attribute
    link.removeAttribute("href");
</script>
```

***Figure 106** - removeAttribute() method – Removing Attributes from Elements Example*

(**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-get-set-attributes.php)

JavaScript DOM Manipulation

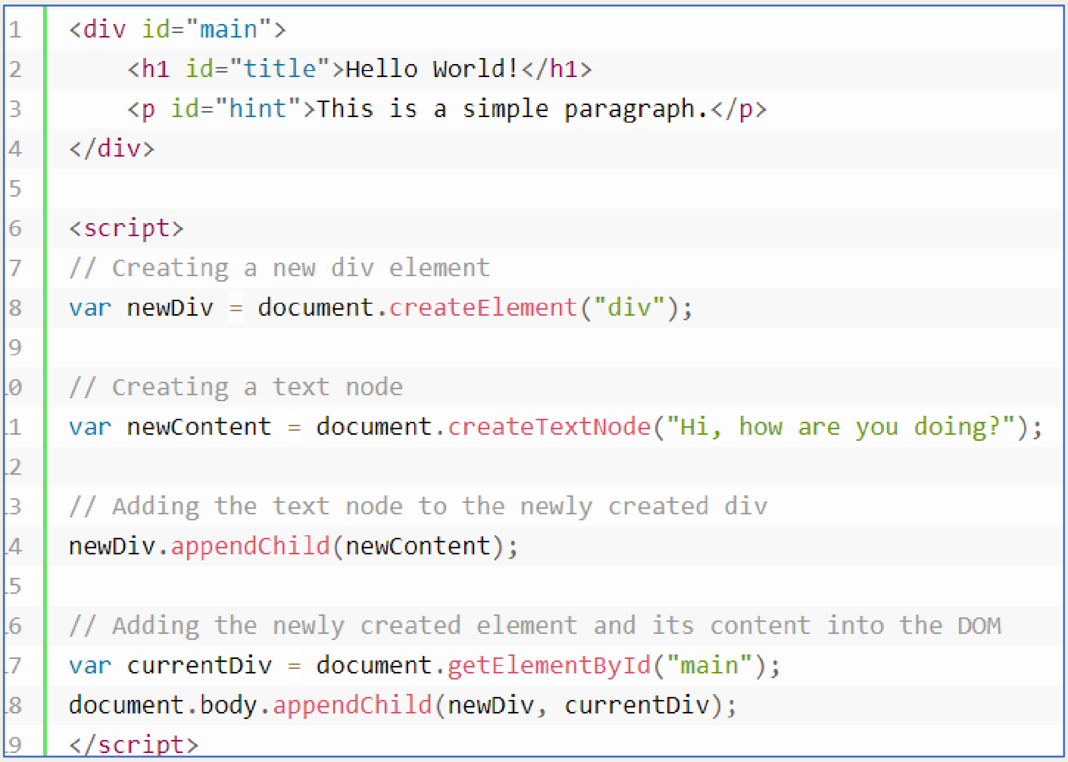
Manipulating DOM Elements in JavaScript

So far, we have learnt how to select and style HTML DOM elements. Now, we will learn how to add or remove DOM elements in a dynamic way, how to get their contents and many more.

Adding New Elements to DOM

The document.createElement() method is used to create a new element in an HTML document. It creates a new element; however, it does not add it to the DOM.

A separate step is needed to add it to the DOM, as shown in the example below:

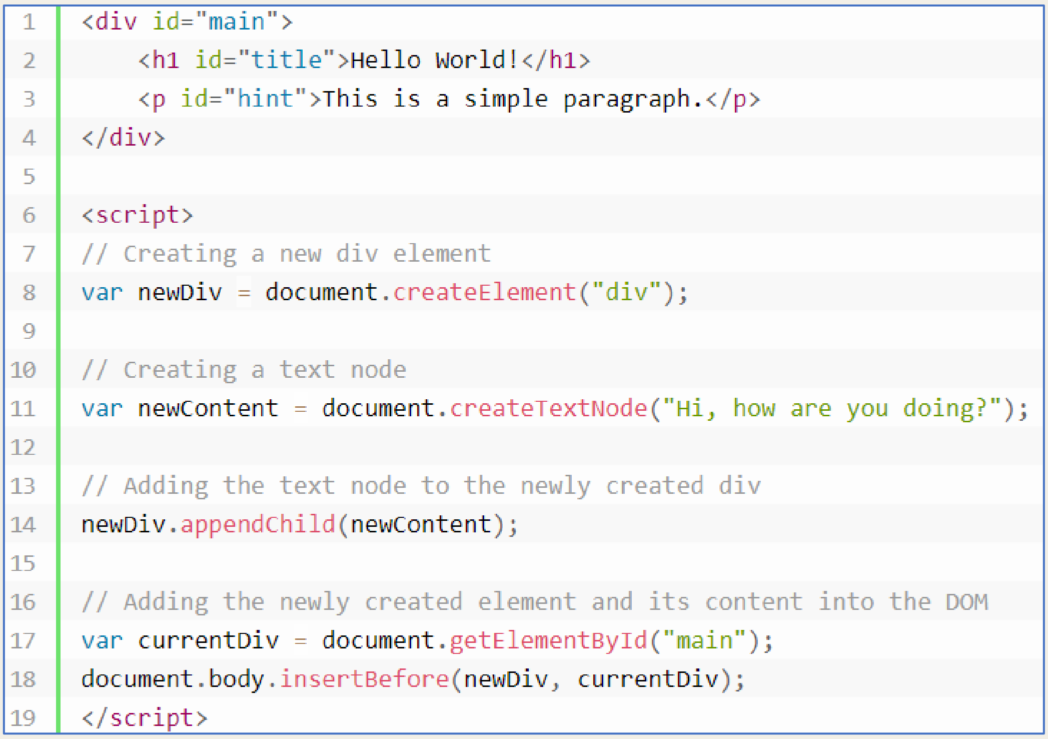


```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
// Creating a new div element
var newDiv = document.createElement("div");

// Creating a text node
var newContent = document.createTextNode("Hi, how are you doing?");

// Adding the text node to the newly created div
newDiv.appendChild(newContent);

// Adding the newly created element and its content into the DOM
var currentDiv = document.getElementById("main");
document.body.appendChild(newDiv, currentDiv);
</script>
```

Figure 107 *- Adding New Elements Example (****Source****:* https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)

In the example we just saw, the appendChild() is used to add the new element at the end of any other children under the specified parent node.

You also have the option to add the new element before any other children, as shown in the example below:



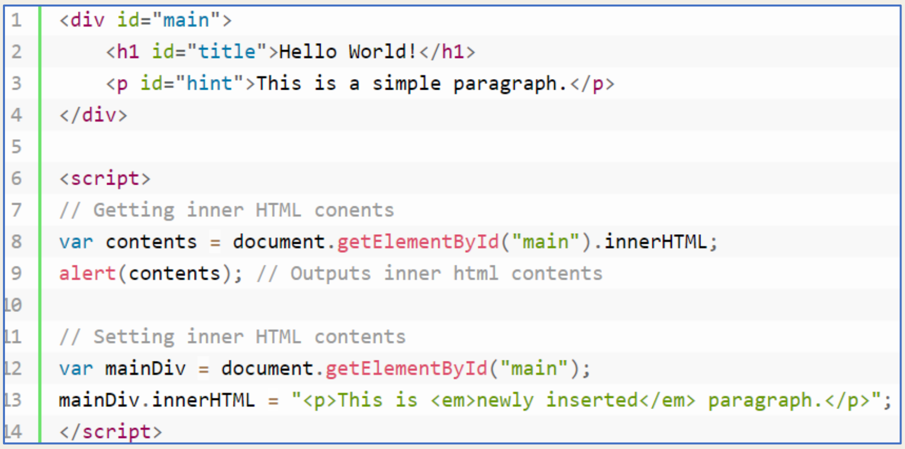
```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
// Creating a new div element
var newDiv = document.createElement("div");

// Creating a text node
var newContent = document.createTextNode("Hi, how are you doing?");

// Adding the text node to the newly created div
newDiv.appendChild(newContent);

// Adding the newly created element and its content into the DOM
var currentDiv = document.getElementById("main");
document.body.insertBefore(newDiv, currentDiv);
</script>
```

***Figure 108** - Adding New Elements Example (**Source**:* https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)

Getting or Setting HTML Contents to DOM

If you want to get or set the contents of HTML elements, you can use the innerHTML property. This property is used to set or get the HTML markup inside the element, which contains content between its opening and closing tags.

Let’s look at an example to get a better understanding of this:



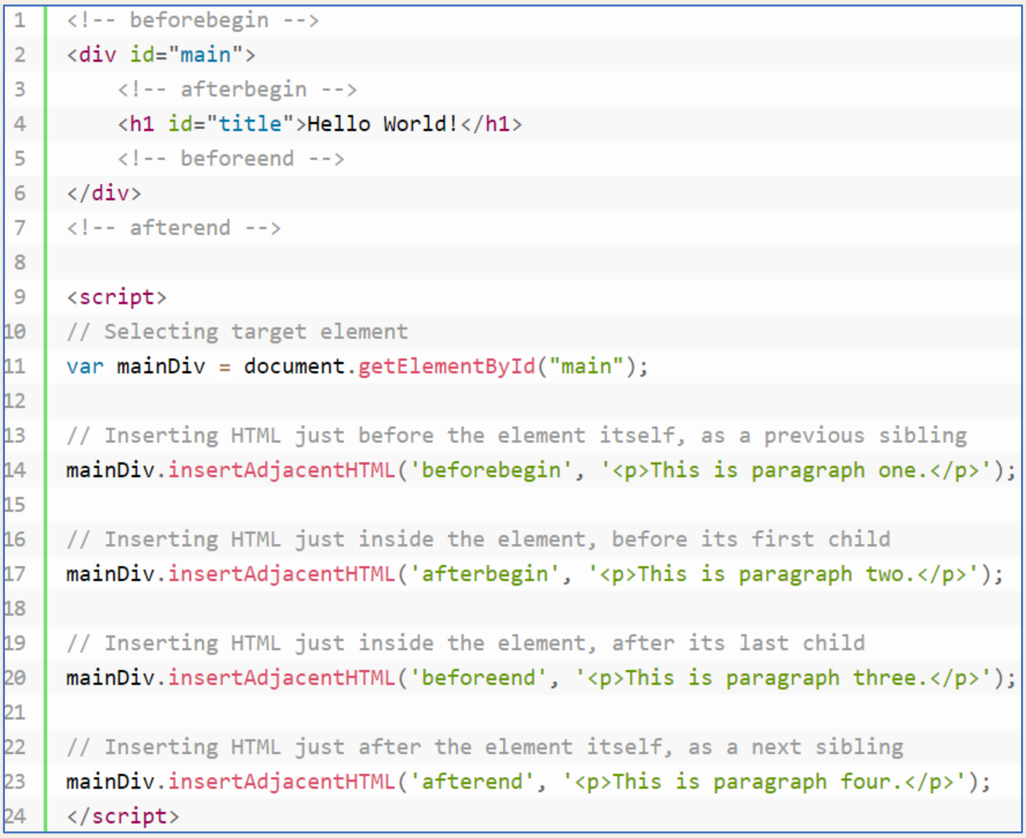
```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
// Getting inner HTML conents
var contents = document.getElementById("main").innerHTML;
alert(contents); // Outputs inner html contents

// Setting inner HTML contents
var mainDiv = document.getElementById("main");
mainDiv.innerHTML = "<p>This is <em>newly inserted</em> paragraph.</p>";
</script>
```

***Figure 109** - Getting or Setting HTML Contents to DOM Example (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)*

As you can from the example, new elements are inserted quite easily into the DOM with the innerHTML property. But this property replaces all the existing content of an element.

Therefore, if you do not want to replace the existing contents of an element, you can use the insertAdjacentHTML() method. This method takes two parameters: the HTML to be inserted and its position. The position must be one of the following: "beforebegin", "afterbegin", "beforeend", and "afterend". It is also significant to note that this method is supported in all major browsers.

In the following example, you can see how the positioning works:



```
<!-- beforebegin -->
<div id="main">
    <!-- afterbegin -->
    <h1 id="title">Hello World!</h1>
    <!-- beforeend -->
</div>
<!-- afterend -->

<script>
// Selecting target element
var mainDiv = document.getElementById("main");

// Inserting HTML just before the element itself, as a previous sibling
mainDiv.insertAdjacentHTML('beforebegin', '<p>This is paragraph one.</p>');

// Inserting HTML just inside the element, before its first child
mainDiv.insertAdjacentHTML('afterbegin', '<p>This is paragraph two.</p>');

// Inserting HTML just inside the element, after its last child
mainDiv.insertAdjacentHTML('beforeend', '<p>This is paragraph three.</p>');

// Inserting HTML just after the element itself, as a next sibling
mainDiv.insertAdjacentHTML('afterend', '<p>This is paragraph four.</p>');
</script>
```

***Figure 110** - insertAdjacentHTML() method - Getting or Setting HTML Contents to DOM Example (**Source**:* https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)

* Keep in mind that for the beforebegin and afterend positions to work the node has to be in the DOM tree and have a parent element.

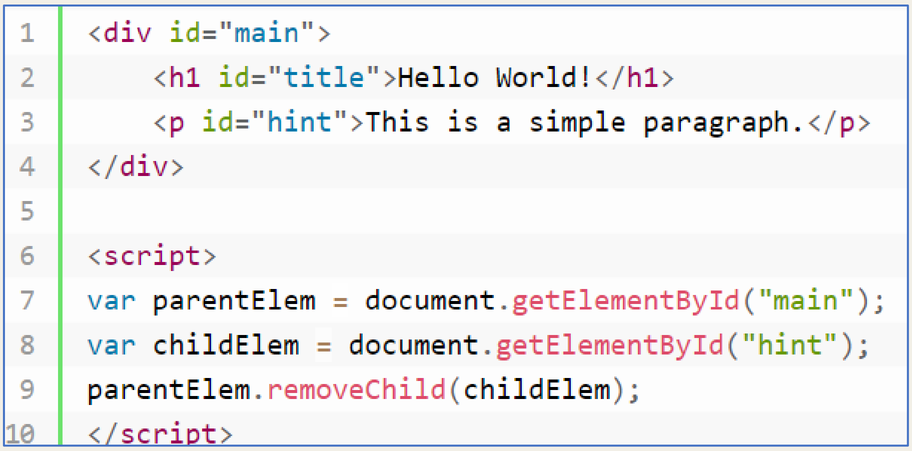
* Also, when inserting an HTML into a page, be careful not to use user input that hasn't been escaped/sanitised, to prevent XSS attacks.

Removing Existing Elements from DOM

To remove a child node from the DOM, you can use the removeChild() method. This method will also return the removed node.

Let’s see the example below:

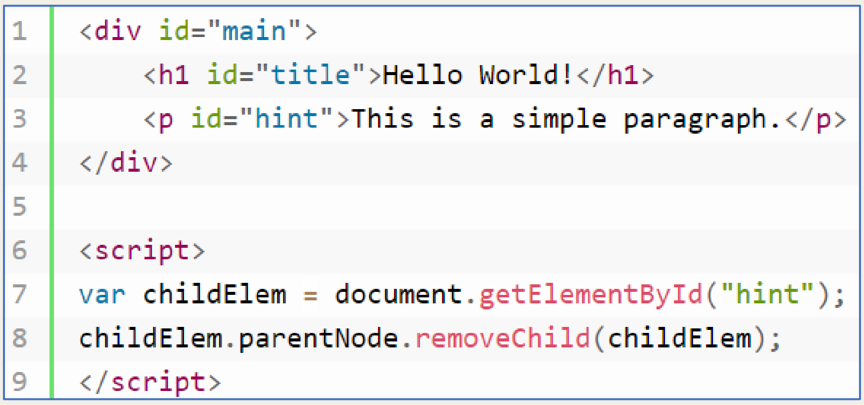
The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
var parentElem = document.getElementById("main");
var childElem = document.getElementById("hint");
parentElem.removeChild(childElem);
</script>
```

Figure 111 *- Removing Existing Elements from DOM Example (**Source**:* https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)

You can also remove the child element without knowing the parent element. You can find the child element and use the parentNode property to find its parent. It will return the parent of the given node in the DOM tree.

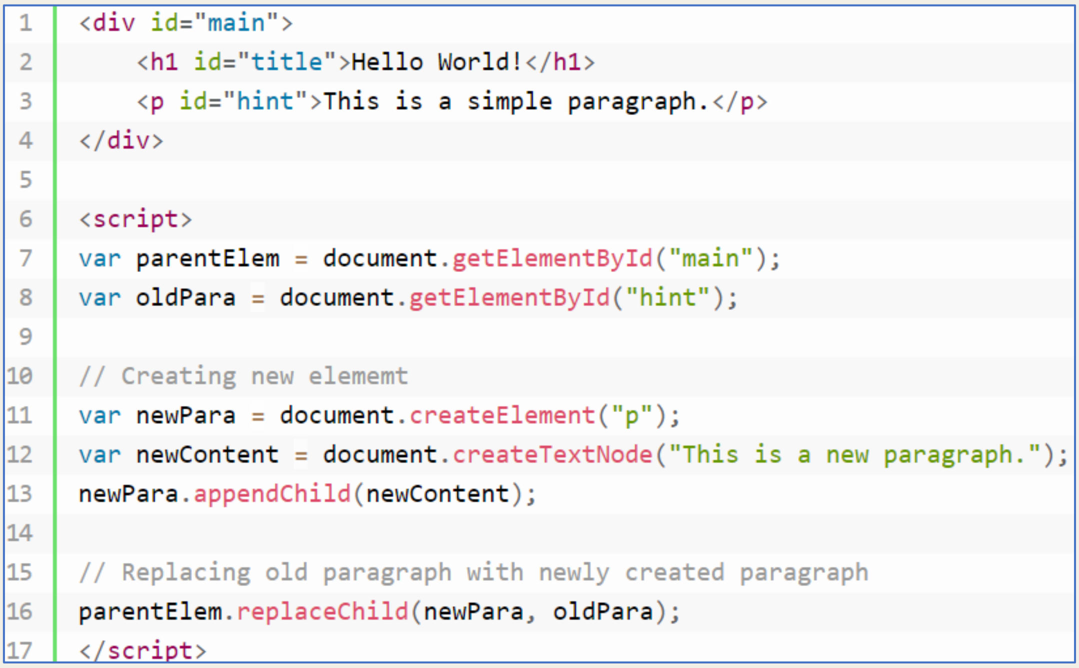


```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
var childElem = document.getElementById("hint");
childElem.parentNode.removeChild(childElem);
</script>
```

Figure 112 *- Removing Existing Elements from DOM Example (**Source:*** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php*)*

Replacing Existing Elements in DOM

You also have the option of replacing an element in HTML DOM with another by using the replaceChild() method. This method takes on two parameters: the node to be inserted and the node to be replaced. The syntax is used is as follows: parentNode.replaceChild(newChild, oldChild);



```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
var parentElem = document.getElementById("main");
var oldPara = document.getElementById("hint");

// Creating new elememt
var newPara = document.createElement("p");
var newContent = document.createTextNode("This is a new paragraph.");
newPara.appendChild(newContent);

// Replacing old paragraph with newly created paragraph
parentElem.replaceChild(newPara, oldPara);
</script>
```

***Figure 113** - Replacing Existing Elements in DOM Example (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-manipulation.php)*

JavaScript DOM Navigation

Navigating Between DOM Nodes

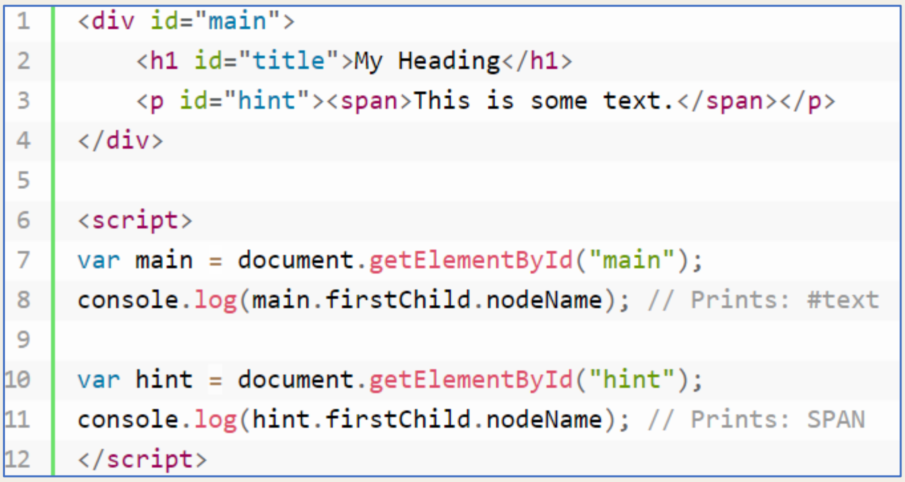
By now, you should have a better idea of how to select individual elements on a web page. There are many occasions where you would need to access child, parent or ancestor element. We have talked about nodes in the beginning of this subchapter and now we will see how we can access the different types of nodes.

DOM nodes have several properties and methods that let you navigate or traverse through the tree DOM structure and make necessary changes quite easily.

Accessing the Child Nodes

The firstChild and lastChild properties allow you to access the first and last direct child node of a node respectively. If a node does not have any child element, it will return null.

Let's check out the example below:



```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var main = document.getElementById("main");
console.log(main.firstChild.nodeName); // Prints: #text

var hint = document.getElementById("hint");
console.log(hint.firstChild.nodeName); // Prints: SPAN
</script>
```

Figure 114 *- Accessing Child Nodes Example (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)*

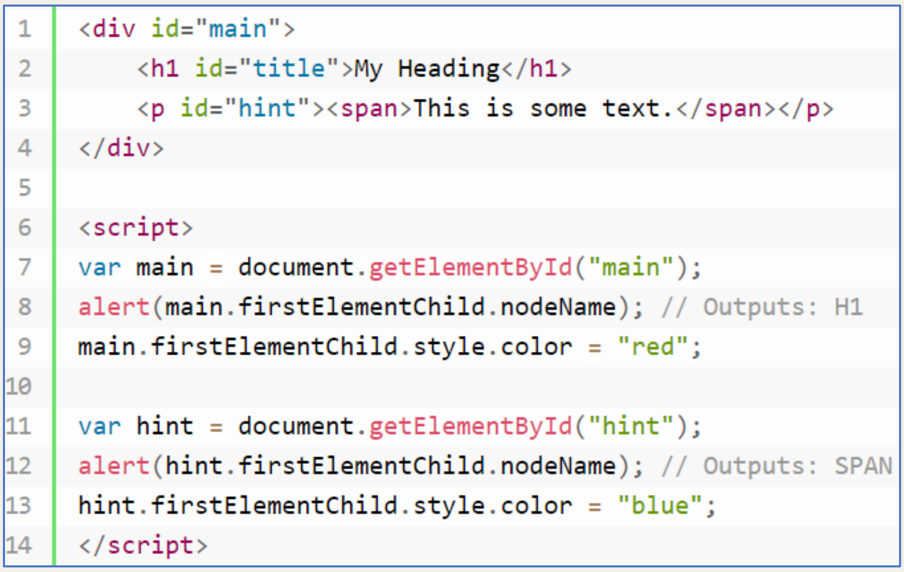
* Please note that the nodeName is a read-only property, which returns the name of the current node as a string. For example, it will return the tag name of an element node, #text for text node, #comment for comment node, #document for document node, and so on.

In the example that we just saw, the nodeName of the first child node of the main DIV elements returned #text instead of H1. This happens because white space, i.e., spaces, tabs, newlines, and so on, are considered valid characters and they become part of the DOM tree in the form of #text nodes. Then, the <div> tag that contains a newline before the <h1> will create #text node.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

In order to prevent this issue with the firstChild and lastChild returning #text or #comment nodes, you can use the firstElementChild and lastElementChild properties as an alternative. These properties will return only the first and last element of the node respectively. However, this will not work in Internet Explores prior to Version 9.

The example below will give you a better understanding of this:

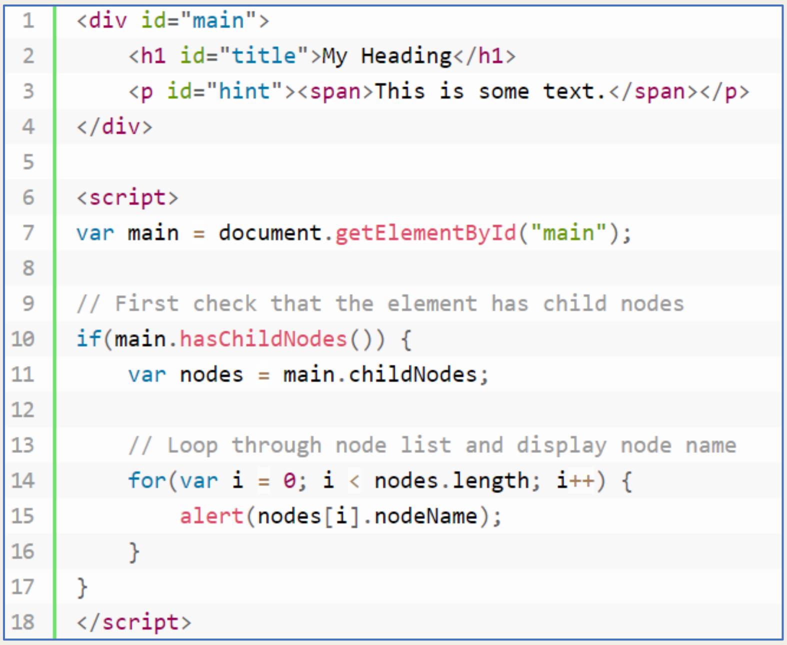


```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var main = document.getElementById("main");
alert(main.firstElementChild.nodeName); // Outputs: H1
main.firstElementChild.style.color = "red";

var hint = document.getElementById("hint");
alert(hint.firstElementChild.nodeName); // Outputs: SPAN
hint.firstElementChild.style.color = "blue";
</script>
```

Figure 115 *- Accessing Child Nodes Example (**Source**: https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)*

To access all child nodes of a given element, you can also use the childNodes property. Keep in mind that the first child node is assigned index 0.

See the example provided below:



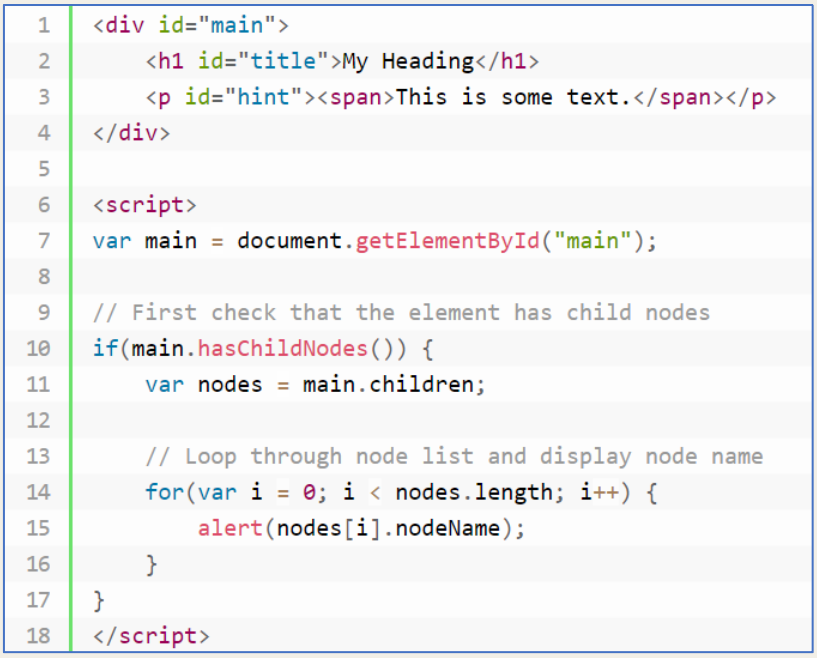
```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var main = document.getElementById("main");

// First check that the element has child nodes
if(main.hasChildNodes()) {
    var nodes = main.childNodes;

    // Loop through node list and display node name
    for(var i = 0; i < nodes.length; i++) {
        alert(nodes[i].nodeName);
    }
}
</script>
```

***Figure 116** - Accessing Child Nodes Example (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)*

Here, the childNodes returns *all child nodes*, including non-element nodes like text and comment nodes.

If you want to get *a collection of only elements*, you should use the children property instead.

Let's look at an example to understand how to use this property:



```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var main = document.getElementById("main");

// First check that the element has child nodes
if(main.hasChildNodes()) {
    var nodes = main.children;

    // Loop through node list and display node name
    for(var i = 0; i < nodes.length; i++) {
        alert(nodes[i].nodeName);
    }
}
</script>
```

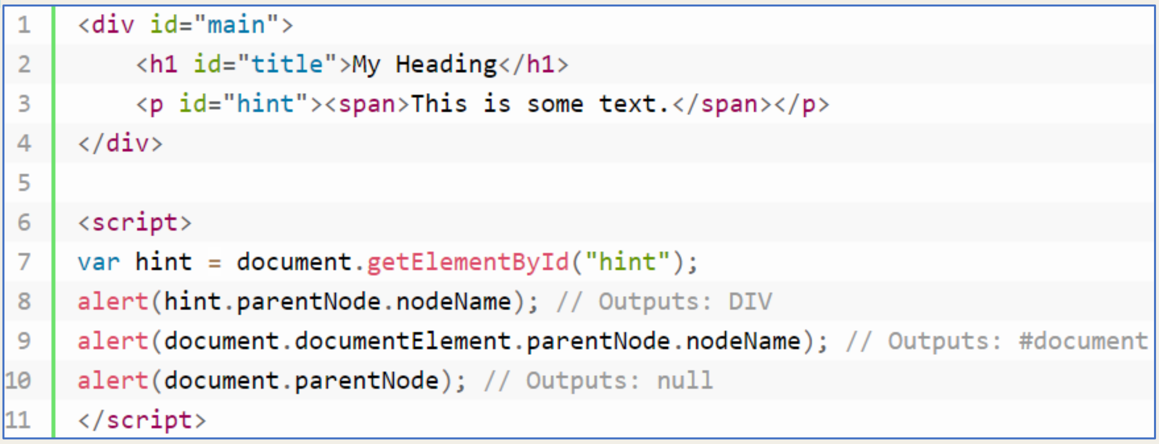
Figure 117** - Accessing Child Nodes Example* (Source:*** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)

Accessing the Parent Nodes

To access the parent node of a specific node in the DOM tree, you can use the parentNode property.

* Note that the parentNode property will always return null values for document nodes because they do not have parents.

In the following example, you can see how the parentNode property is used:

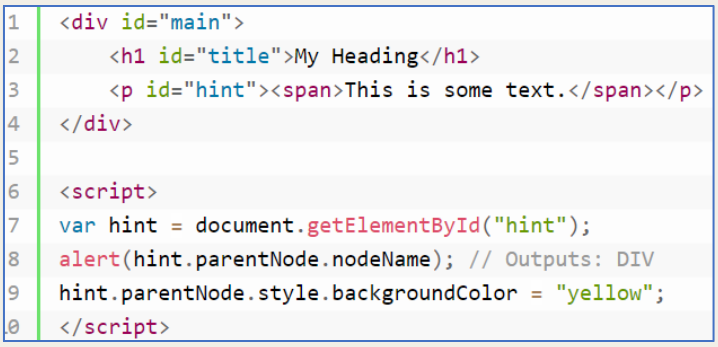


```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var hint = document.getElementById("hint");
alert(hint.parentNode.nodeName); // Outputs: DIV
alert(document.documentElement.parentNode.nodeName); // Outputs: #document
alert(document.parentNode); // Outputs: null
</script>
```

Figure 118 *- Accessing Parent Nodes Example (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)*

It is good to know that the topmost DOM tree nodes can be accessed directly as document properties. We saw some examples of the topmost DOM tree nodes in earlier sections such as the <html> element, which can be accessed with document.documentElement property. Also, the <head> element can be accessed with document.head property, and the <body> element can be accessed with document.body property.

There is also an option to get only element nodes with the parentElement, as shown in the example below:



```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var hint = document.getElementById("hint");
alert(hint.parentNode.nodeName); // Outputs: DIV
hint.parentNode.style.backgroundColor = "yellow";
</script>
```

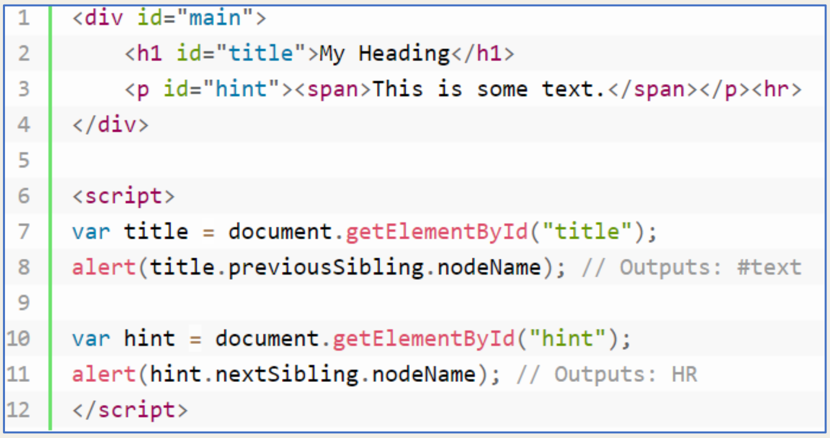
Figure 119 *- Accessing Parent Nodes Example (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Accessing the Sibling Nodes

To access the previous and next node in the DOM tree, you can use the previousSibling and nextSibling properties respectively.

Let’s look at an example:

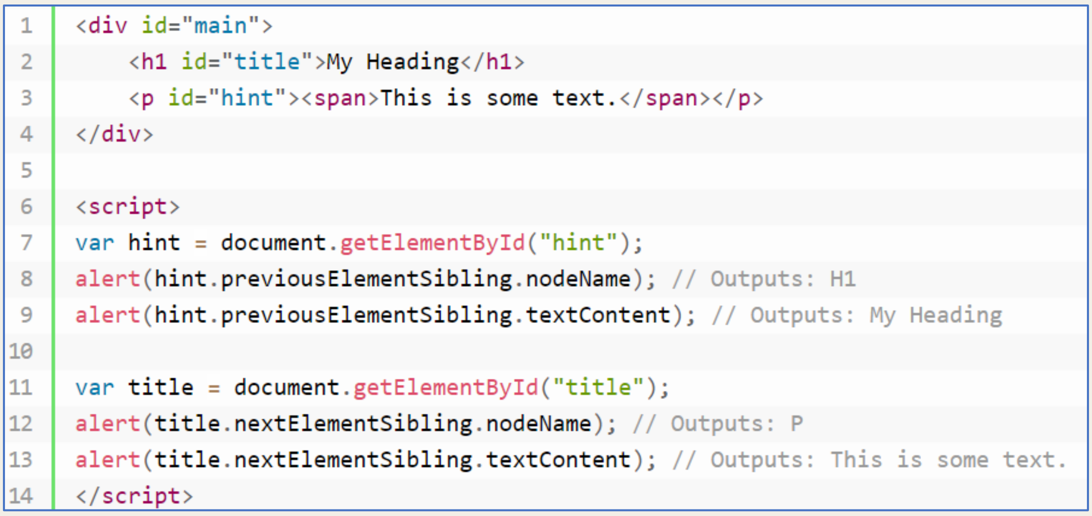


```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p><hr>
</div>

<script>
var title = document.getElementById("title");
alert(title.previousSibling.nodeName); // Outputs: #text

var hint = document.getElementById("hint");
alert(hint.nextSibling.nodeName); // Outputs: HR
</script>
```

***Figure 120** - Accessing Sibling Nodes Example (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)*

To skip any whitespace text nodes, you can use the previousElementSibling and nextElementSibling as alternatives to get the previous and next sibling elements. If no such sibling is found, these properties will return null values.

Let’s see an example below:



```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var hint = document.getElementById("hint");
alert(hint.previousElementSibling.nodeName); // Outputs: H1
alert(hint.previousElementSibling.textContent); // Outputs: My Heading

var title = document.getElementById("title");
alert(title.nextElementSibling.nodeName); // Outputs: P
alert(title.nextElementSibling.textContent); // Outputs: This is some text.
</script>
```

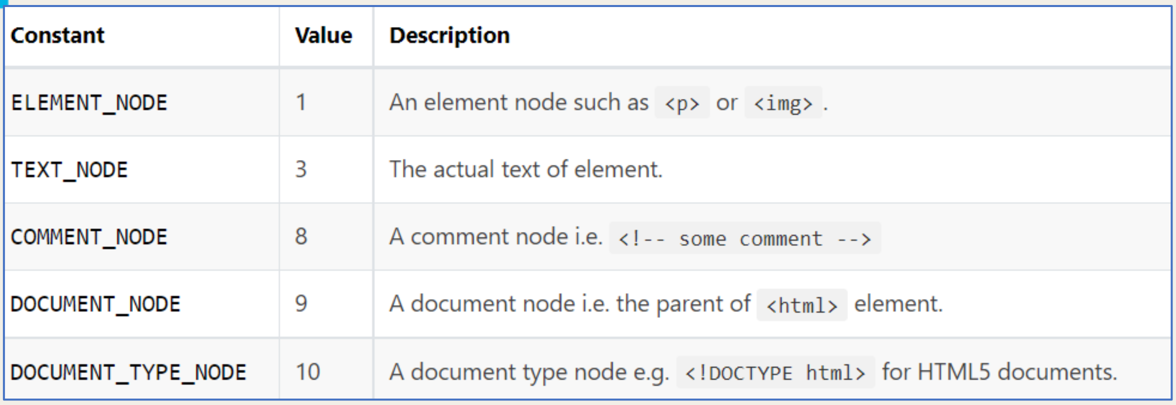
***Figure 121** - Accessing Sibling Nodes Example (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)*

The textContent property used here signifies the text content of a node and all of its descendants.

Types of DOM Nodes

The DOM tree is comprised of different types of nodes that includes elements, text, comments and many more.

Every node has a nodeType property that can help you understand how you can access and manipulate said node. The following table provides a list with the most important and used node types that one needs to know:



Constant	Value	Description
`ELEMENT_NODE`	1	An element node such as `<p>` or `<img>`.
`TEXT_NODE`	3	The actual text of element.
`COMMENT_NODE`	8	A comment node i.e. `<!-- some comment -->`
`DOCUMENT_NODE`	9	A document node i.e. the parent of `<html>` element.
`DOCUMENT_TYPE_NODE`	10	A document type node e.g. `<!DOCTYPE html>` for HTML5 documents.

***Figure 122** - Table of Most Common Types of DOM Nodes (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dom-navigation.php)*

5.3. JavaScript & BOM

JavaScript Window - The Browser Object Model

The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.

The Browser Object Model (BOM)

There are no official standards for the Browser Object Model (BOM).

Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to, as methods and properties of the BOM.

The Window Object

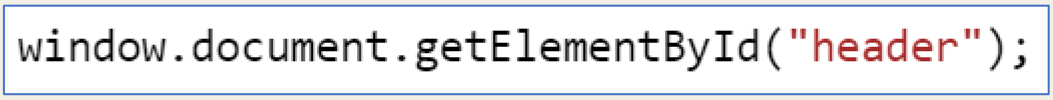
The window object is supported by all browsers. It represents the browser's window.

All global JavaScript objects, functions, and variables automatically become members of the window object.

Global variables are properties of the window object.

Global functions are methods of the window object.

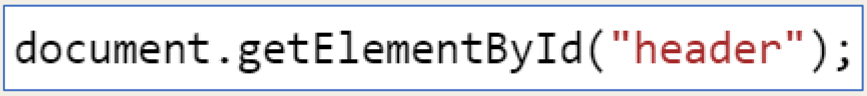
Even the document object (of the HTML DOM) is a property of the window object:



```
window.document.getElementById("header");
```

***Figure 123** – The Window Object on JS BOM (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window.php)*

Is the same as:



```
document.getElementById("header");
```

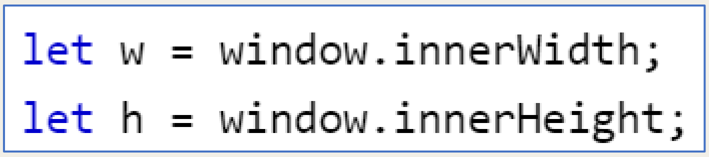
***Figure 124** – The Window Object on JS BOM – alternative version (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window.php)*

Window Size

Two properties can be used to determine the size of the browser window.

Both properties return the sizes in pixels:

- window.innerHeight - the inner height of the browser window (in pixels)
- window.innerWidth - the inner width of the browser window (in pixels)



```
let w = window.innerWidth;
let h = window.innerHeight;
```

***Figure 125** – The Window size alternatives (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window.php)*

Other Window Methods

Some other methods:

- window.open() - open a new window
- window.close() - close the current window
- window.moveTo() - move the current window
- window.resizeTo() - resize the current window

JavaScript Window Screen

The window.screen object contains information about the user's screen.

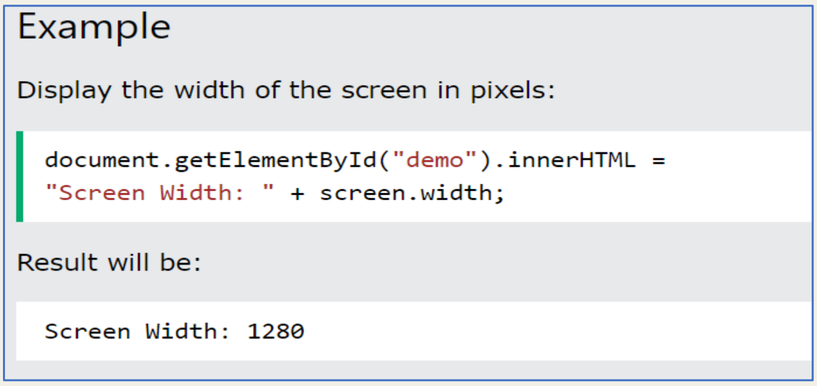
The window.screen object can be written without the window prefix.

Properties:

- screen.width
- screen.height
- screen.availWidth
- screen.availHeight
- screen.colorDepth
- screen.pixelDepth

Window Screen Width

The screen.width property returns the width of the visitor's screen in pixels.



Example

Display the width of the screen in pixels:

```
document.getElementById("demo").innerHTML =
"Screen Width: " + screen.width;
```

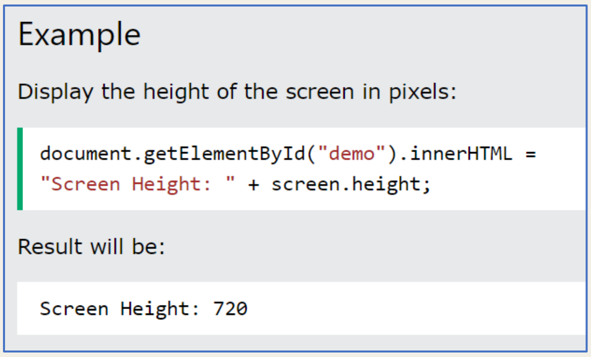
Result will be:

```
Screen Width: 1280
```

***Figure 126** – The screen.width property (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window.php)*

Window Screen Height

The screen.height property returns the height of the visitor's screen in pixels.



Example

Display the height of the screen in pixels:

```
document.getElementById("demo").innerHTML =
"Screen Height: " + screen.height;
```

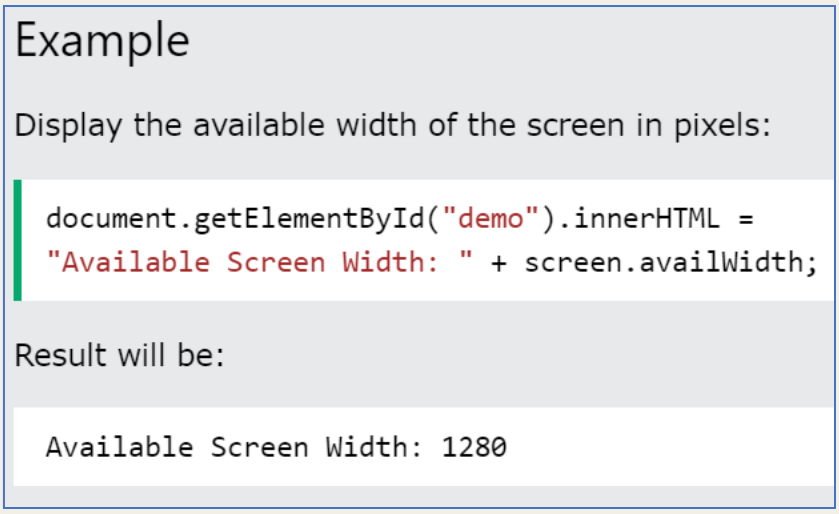
Result will be:

```
Screen Height: 720
```

***Figure 127** – The Window screen.height property (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window.php)*

Window Screen Available Width

The screen.availWidth property returns the width of the visitor's screen, in pixels, minus interface features like the Windows Taskbar.



Example

Display the available width of the screen in pixels:

```
document.getElementById("demo").innerHTML =
"Available Screen Width: " + screen.availWidth;
```

Result will be:

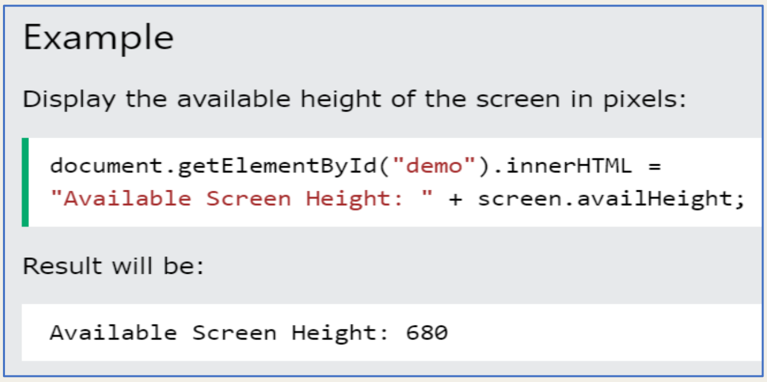
```
Available Screen Width: 1280
```

***Figure 128** – The Window screen.availWidth property (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window.php)*

Window Screen Available Height

The screen.availHeight property returns the height of the visitor's screen, in pixels, minus interface features like the Windows Taskbar.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



Example

Display the available height of the screen in pixels:

```
document.getElementById("demo").innerHTML =
"Available Screen Height: " + screen.availHeight;
```

Result will be:

```
Available Screen Height: 680
```

***Figure 129** – The Window screen.availWidth property (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window.php)*

Window Screen Colour Depth

The screen.colorDepth property returns the number of bits used to display one colour.

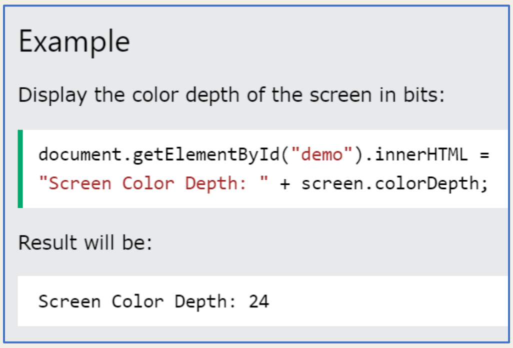
All modern computers use 24 bit or 32-bit hardware for colour resolution:

24 bits = 16,777,216 different "True Colours"

32 bits = 4,294,967,296 different "Deep Colours"

Older computers used 16 bits: 65,536 different "High Colours" resolution.

Very old computers, and old cell phones used 8 bits: 256 different "VGA colours".



Example

Display the color depth of the screen in bits:

```
document.getElementById("demo").innerHTML =
"Screen Color Depth: " + screen.colorDepth;
```

Result will be:

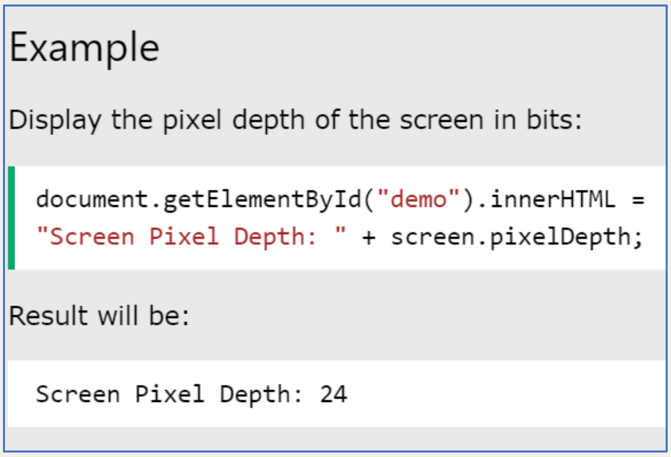
```
Screen Color Depth: 24
```

***Figure 130** – The Window screen.availWidth property (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window.php)*

Window Screen Pixel Depth

The screen.pixelDepth property returns the pixel depth of the screen.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



Example

Display the pixel depth of the screen in bits:

```
document.getElementById("demo").innerHTML =
"Screen Pixel Depth: " + screen.pixelDepth;
```

Result will be:

```
Screen Pixel Depth: 24
```

***Figure 131**– The Window screen.pixelDepth property (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window.php)*

JavaScript Window Location

The window.location object can be used to get the current page address (URL) and to redirect the browser to a new page.

Window Location

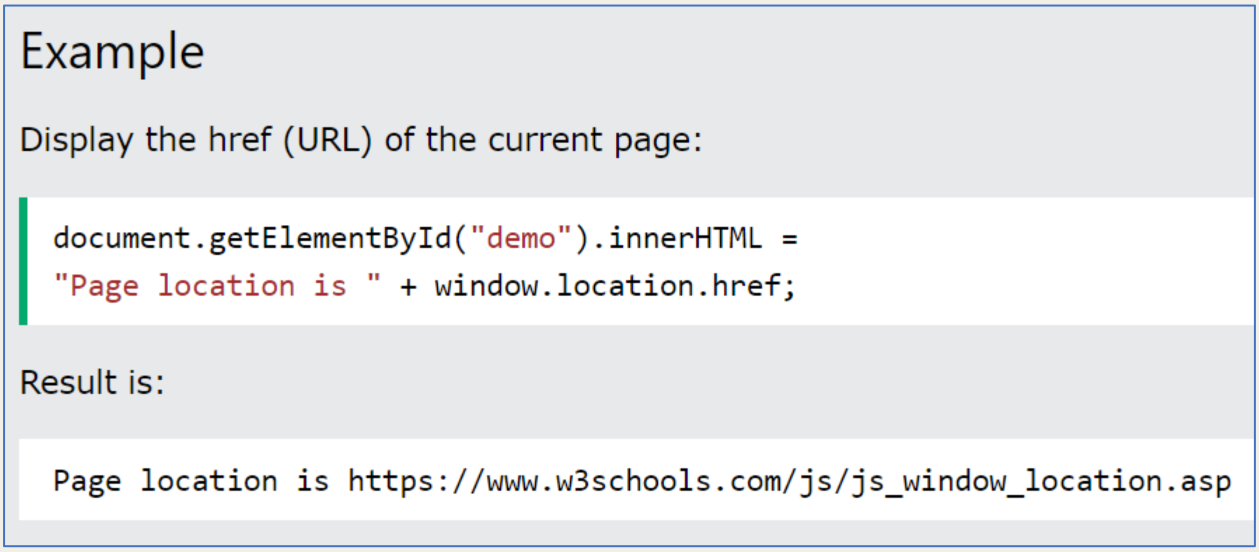
The window.location object can be written without the window prefix.

Some examples:

- window.location.href returns the href (URL) of the current page
- window.location.hostname returns the domain name of the web host
- window.location.pathname returns the path and filename of the current page
- window.location.protocol returns the web protocol used (http: or https:)
- window.location.assign() loads a new document

Window Location Href

The window.location.href property returns the URL of the current page.



Example

Display the href (URL) of the current page:

```
document.getElementById("demo").innerHTML =
"Page location is " + window.location.href;
```

Result is:

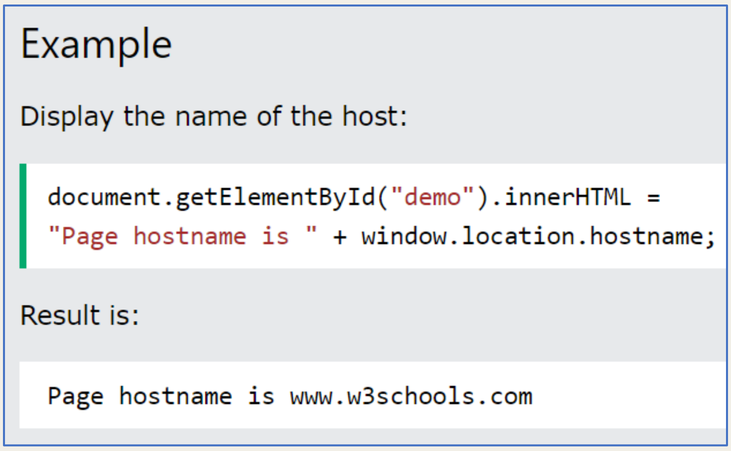
```
Page location is https://www.w3schools.com/js/js_window_location.asp
```

Figure 132 *– The Window window.location.href property (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window-location.php)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Window Location Hostname

The window.location.hostname property returns the name of the internet host (of the current page).



Example

Display the name of the host:

```
document.getElementById("demo").innerHTML =
"Page hostname is " + window.location.hostname;
```

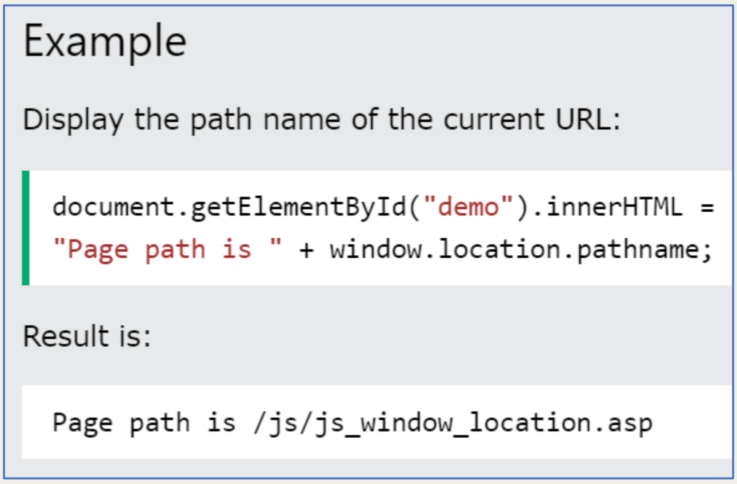
Result is:

```
Page hostname is www.w3schools.com
```

***Figure 133** – The Window window.location.hostname property (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window-location.php)*

Window Location Pathname

The window.location.pathname property returns the pathname of the current page.



Example

Display the path name of the current URL:

```
document.getElementById("demo").innerHTML =
"Page path is " + window.location.pathname;
```

Result is:

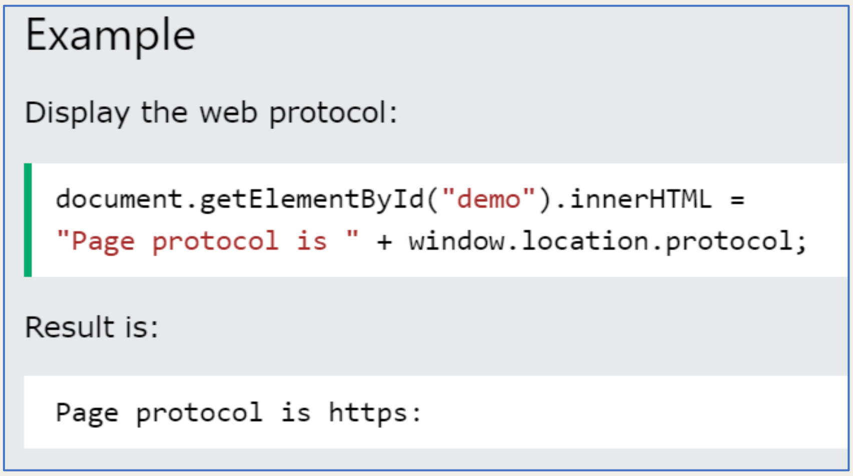
```
Page path is /js/js_window_location.asp
```

***Figure 134** – The Window window.location.pathname property (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window-location.php)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Window Location Protocol

The window.location.protocol property returns the web protocol of the page.



Example

Display the web protocol:

```
document.getElementById("demo").innerHTML =
"Page protocol is " + window.location.protocol;
```

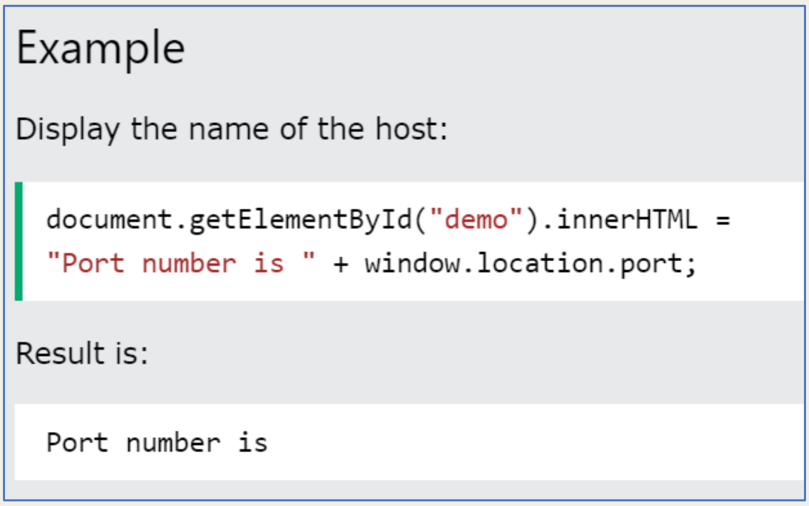
Result is:

```
Page protocol is https:
```

***Figure 135** – The Window window.location.protocol property (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window-location.php)*

Window Location Port

The window.location.port property returns the number of the internet host port (of the current page).



Example

Display the name of the host:

```
document.getElementById("demo").innerHTML =
"Port number is " + window.location.port;
```

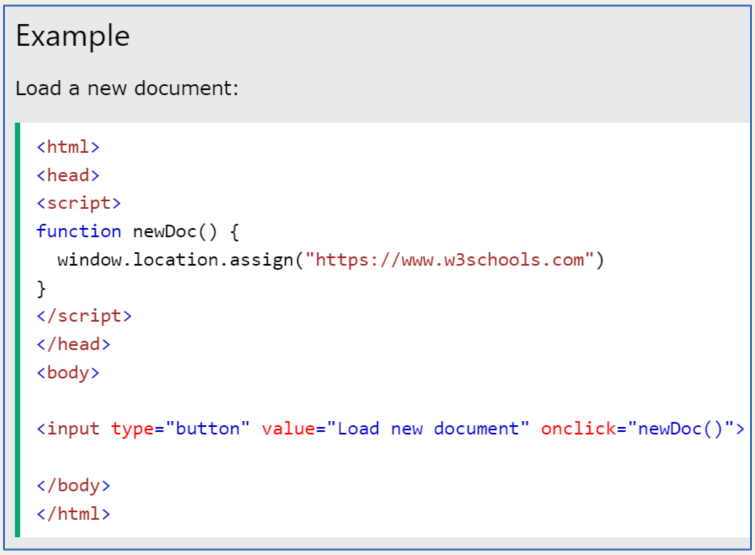
Result is:

```
Port number is
```

***Figure 136** – The Window window.location.port property (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window-location.php)*

Window Location Assign

The window.location.assign() method loads a new document.



Example

Load a new document:

```
<html>
<head>
<script>
function newDoc() {
  window.location.assign("https://www.w3schools.com")
}
</script>
</head>
<body>

<input type="button" value="Load new document" onclick="newDoc()">

</body>
</html>
```

Figure 137** – The Window window.location.assign() property (**Source:
https://www.tutorialrepublic.com/javascript-tutorial/javascript-window-location.php)

JavaScript Window History

The window.history object contains the browsers history.

Window History

The window.history object can be written without the window prefix.

To protect the privacy of the users, there are limitations to how JavaScript can access this object.

Some methods:

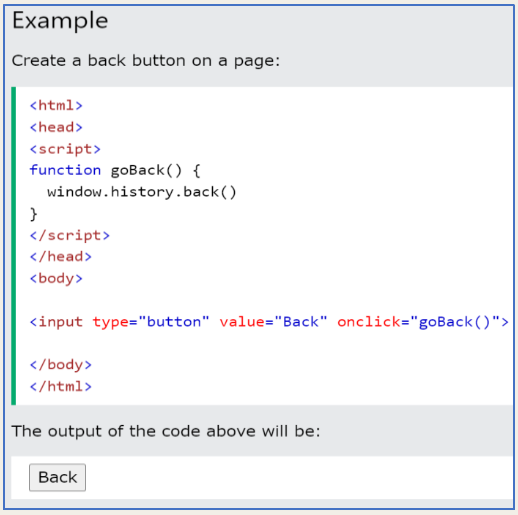
- history.back() - same as clicking back in the browser
- history.forward() - same as clicking forward in the browser

Window History Back

The history.back() method loads the previous URL in the history list.

This is the same as clicking the Back button in the browser.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



Example

Create a back button on a page:

```
<html>
<head>
<script>
function goBack() {
  window.history.back()
}
</script>
</head>
<body>

<input type="button" value="Back" onclick="goBack()">

</body>
</html>
```

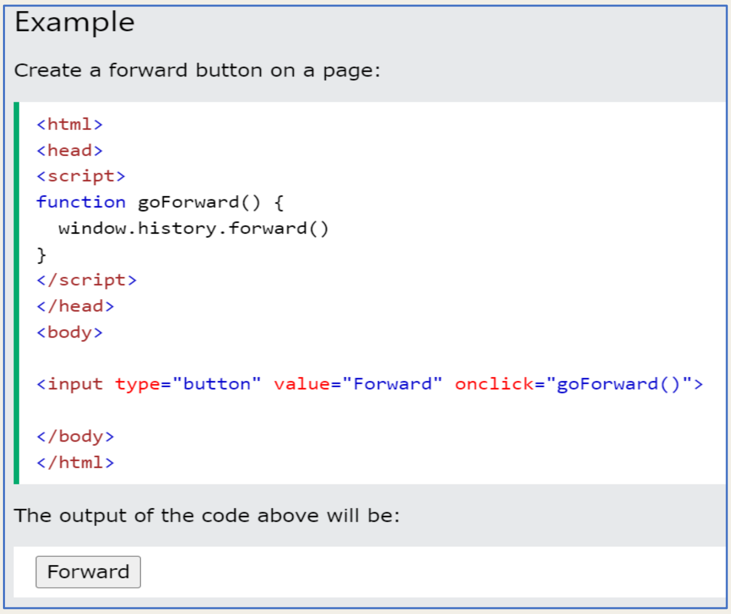
The output of the code above will be:

Figure 138 *– The Window history.back() property (**Source:*** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window-location.php*)*

Window History Forward

The history.forward() method loads the next URL in the history list.

This is the same as clicking the Forward button in the browser.



Example

Create a forward button on a page:

```
<html>
<head>
<script>
function goForward() {
  window.history.forward()
}
</script>
</head>
<body>

<input type="button" value="Forward" onclick="goForward()">

</body>
</html>
```

The output of the code above will be:

Figure 139 *– The Window history.forward() property (**Source:*** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window-location.php*)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

JavaScript Window Navigator

The window.navigator object contains information about the visitor's browser.

Window Navigator

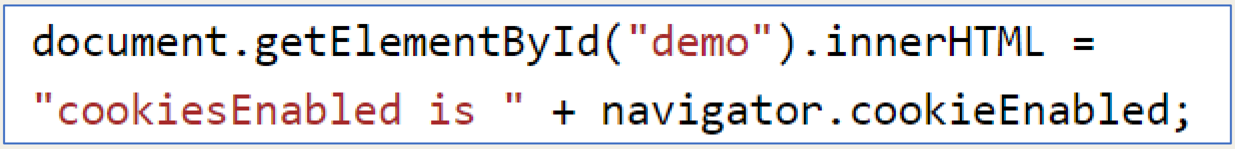
The window.navigator object can be written without the window prefix.

Some examples:

- navigator.appName
- navigator.appCodeName
- navigator.platform

Browser Cookies

The cookieEnabled property returns true if cookies are enabled, otherwise false:



```
document.getElementById("demo").innerHTML =
"cookiesEnabled is " + navigator.cookieEnabled;
```

***Figure 140** – The Window history.forward() property (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-window-location.php)*

Browser Application Name

The appName property returns the application name of the browser.

Browser Application Code Name

The appCodeName property returns the application code name of the browser.

The Browser Engine

The product property returns the product name of the browser engine.

The Browser Version

The appVersion property returns version information about the browser.

The Browser Agent

The userAgent property returns the user-agent header sent by the browser to the server.

Warning !!!

The information from the navigator object can often be misleading, and should not be used to detect browser versions because:

- Different browsers can use the same name
- The navigator data can be changed by the browser owner
- Some browsers misidentify themselves to bypass site tests
- Browsers cannot report new operating systems, released later than the browser

The Browser Platform

The platform property returns the browser platform (operating system).

The Browser Language

The language property returns the browser's language.

Is The Browser Online?

The onLine property returns true if the browser is online.

Is Java Enabled?

The javaEnabled() method returns true if Java is enabled.

JavaScript Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

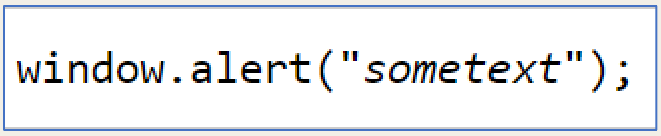
The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

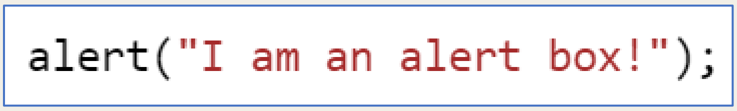
Syntax:



```
window.alert("sometext");
```

***Figure 141**– Alert box syntax (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dialog-boxes.php)*

The window.alert() method can be written without the window prefix.



```
alert("I am an alert box!");
```

***Figure 142** – The window.alert() method (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dialog-boxes.php)*

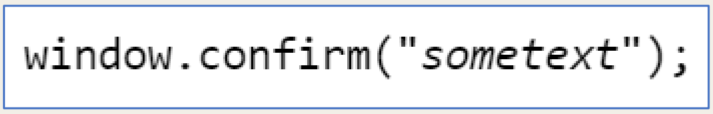
Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax:

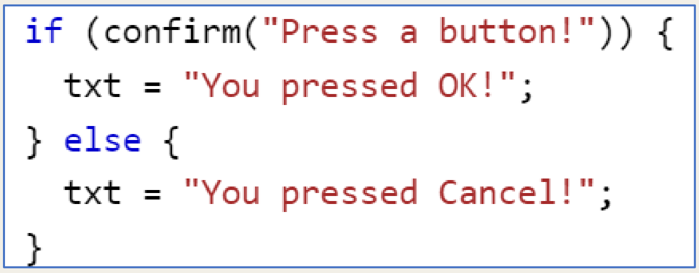


```
window.confirm("sometext");
```

***Figure 143** – The window.alert() method (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dialog-boxes.php)*

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

The window.confirm() method can be written without the window prefix.



```
if (confirm("Press a button!")) {
  txt = "You pressed OK!";
} else {
  txt = "You pressed Cancel!";
}
```

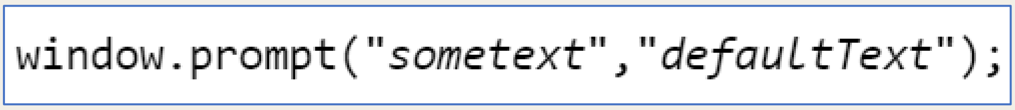
Figure 144 *– The window.confirm() method (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dialog-boxes.php)*

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page. When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax:

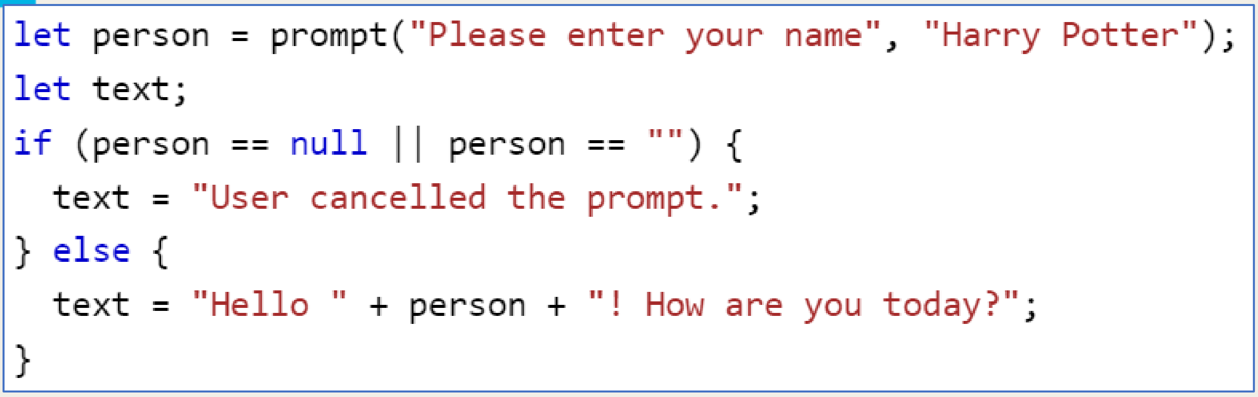


```
window.prompt("sometext","defaultText");
```

Figure 145 *– The prompt box syntax (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dialog-boxes.php)*

The window.prompt() method can be written without the window prefix.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

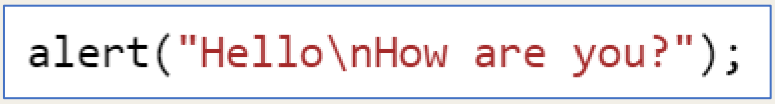


```
let person = prompt("Please enter your name", "Harry Potter");
let text;
if (person == null || person == "") {
  text = "User cancelled the prompt.";
} else {
  text = "Hello " + person + "! How are you today?";
}
```

***Figure 146** – The window.prompt() method (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dialog-boxes.php)*

Line Breaks

To display line breaks inside a popup box, use a back-slash followed by the character n.



```
alert("Hello\nHow are you?");
```

***Figure 147** – Displaying line breaks inside a popup box (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-dialog-boxes.php)*

JavaScript Timing Events

Timing Events

The window object allows execution of code at specified time intervals.

These time intervals are called timing events.

The two key methods to use with JavaScript are:

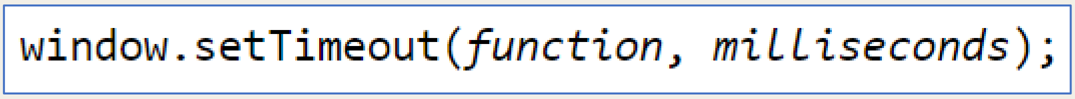
- setTimeout(function,milliseconds)

  Executes a function, after waiting a specified number of milliseconds.
- setInterval(function,milliseconds)

  Same as setTimeout(), but repeats the execution of the function continuously.

The setTimeout() and setInterval() are both methods of the HTML DOM Window object.

The setTimeout() Method



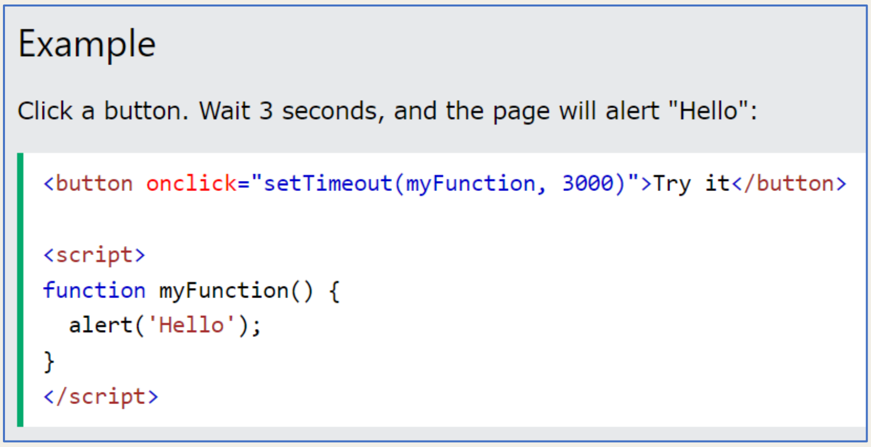
```
window.setTimeout(function, milliseconds);
```

***Figure 148** – The setTimeout() method (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-timers.php)*

The window.setTimeout() method can be written without the window prefix.

The first parameter is a function to be executed.

The second parameter indicates the number of milliseconds before execution.



Example

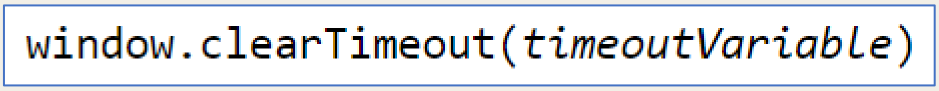
Click a button. Wait 3 seconds, and the page will alert "Hello":

```
<button onclick="setTimeout(myFunction, 3000)">Try it</button>

<script>
function myFunction() {
  alert('Hello');
}
</script>
```

***Figure 149** – The window.setTimeout() method (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-timers.php)*

How to Stop the Execution?

The clearTimeout() method stops the execution of the function specified in setTimeout().

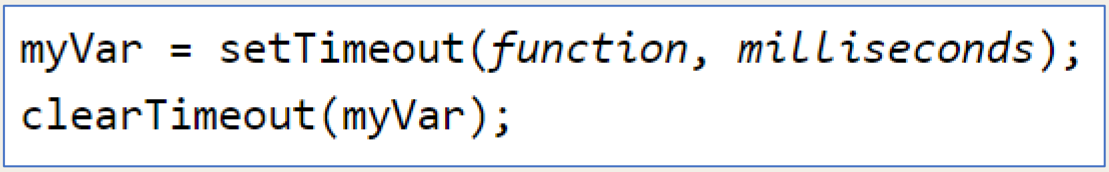


```
window.clearTimeout(timeoutVariable)
```

***Figure 150** – The clearTimeOut() method (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-timers.php)*

The window.clearTimeout() method can be written without the window prefix.

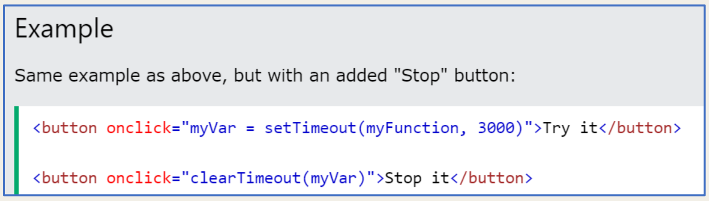
The clearTimeout() method uses the variable returned from setTimeout():



```
myVar = setTimeout(function, milliseconds);
clearTimeout(myVar);
```

***Figure 151** – The window.clearTimeout() method (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-timers.php)*

If the function has not already been executed, you can stop the execution by calling the clearTimeout() method:



Example

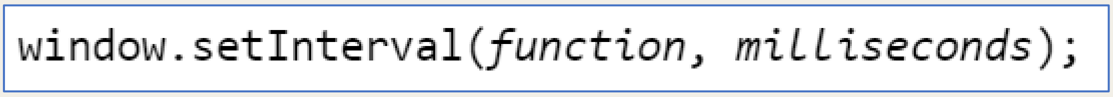
Same example as above, but with an added "Stop" button:

```
<button onclick="myVar = setTimeout(myFunction, 3000)">Try it</button>

<button onclick="clearTimeout(myVar)">Stop it</button>
```

***Figure 152** – The clearTimeout() method (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-timers.php)*

The setInterval() Method

The setInterval() method repeats a given function at every given time-interval.



```
window.setInterval(function, milliseconds);
```

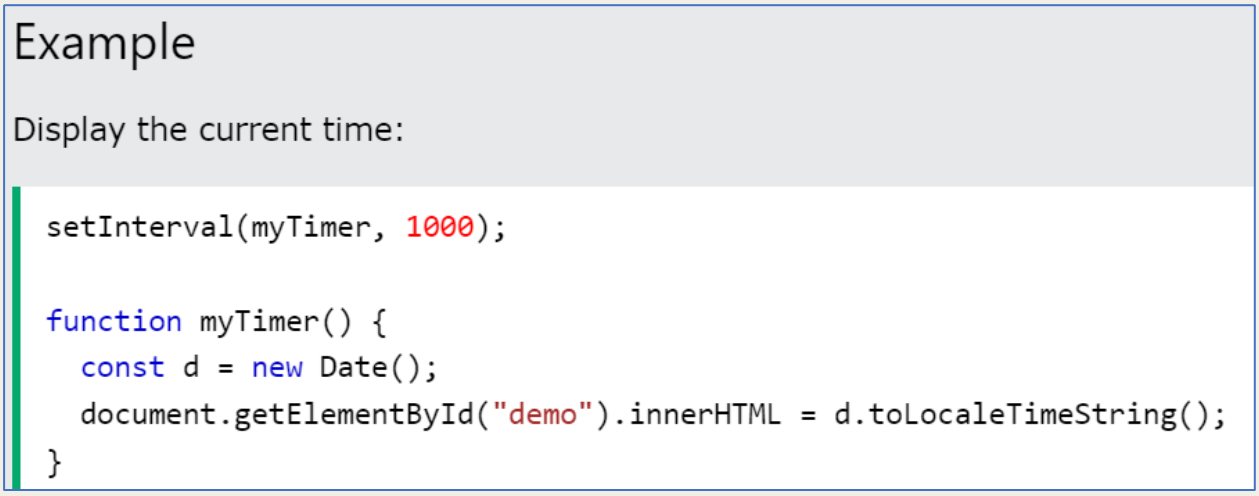
***Figure 153** – The setInterval() method (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-timers.php)*

The window.setInterval() method can be written without the window prefix.

The first parameter is the function to be executed.

The second parameter indicates the length of the time-interval between each execution.

This example executes a function called "myTimer" once every second (like a digital watch).



Example

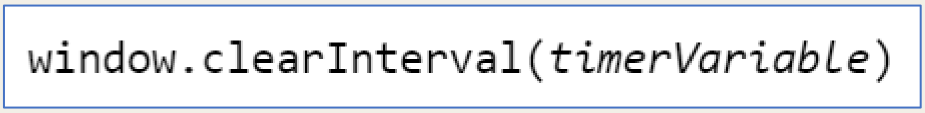
Display the current time:

```
setInterval(myTimer, 1000);

function myTimer() {
  const d = new Date();
  document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
```

***Figure 154** – The window.setInterval() method (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-timers.php)*

How to Stop the Execution?

The clearInterval() method stops the executions of the function specified in the setInterval() method.

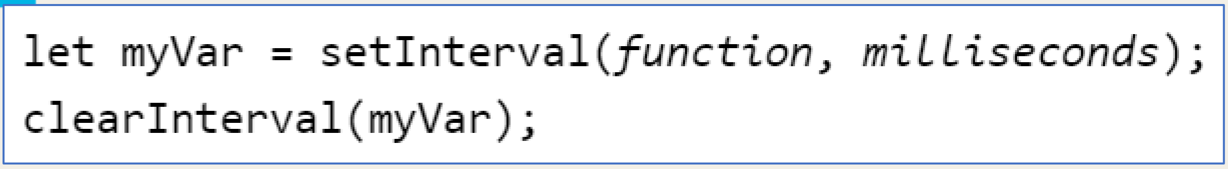


```
window.clearInterval(timerVariable)
```

***Figure 155** – The clearInterval() method (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-timers.php)*

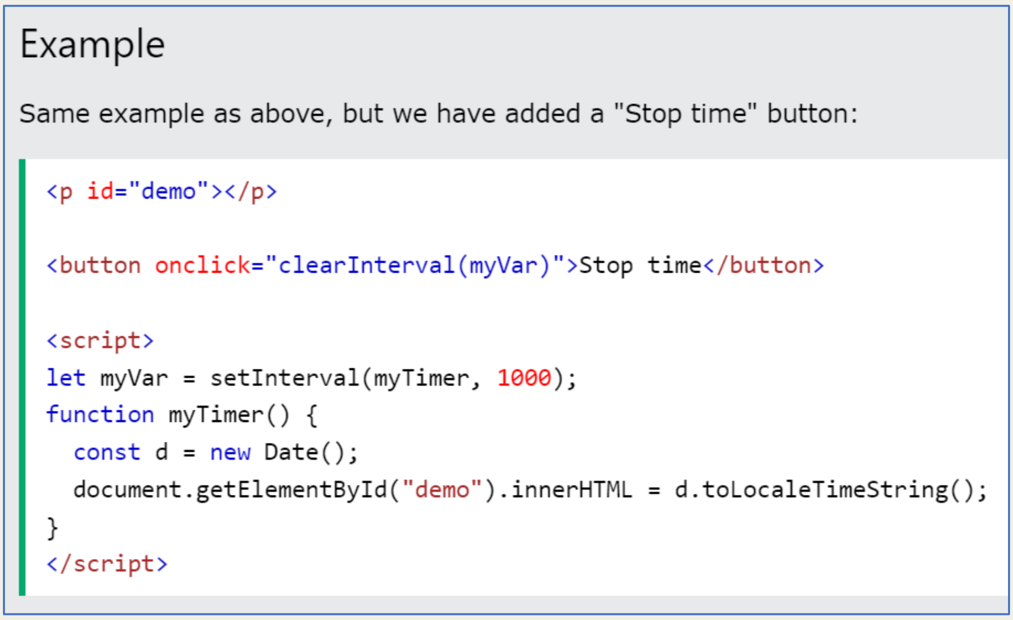
The window.clearInterval() method can be written without the window prefix.

The clearInterval() method uses the variable returned from setInterval():



```
let myVar = setInterval(function, milliseconds);
clearInterval(myVar);
```

***Figure 156** – The variable returned from setInterval() (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-timers.php)*



Example

Same example as above, but we have added a "Stop time" button:

```
<p id="demo"></p>

<button onclick="clearInterval(myVar)">Stop time</button>

<script>
let myVar = setInterval(myTimer, 1000);
function myTimer() {
  const d = new Date();
  document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
</script>
```

***Figure 157**– The variable returned from setInterval() (**Source:** https://www.tutorialrepublic.com/javascript-tutorial/javascript-timers.php)*

5.4. JavaScript Advanced

JavaScript Date and Time

Note that the JavaScript Date object is provided by default as a global object in all JavaScript environments, thus you don't have to import it.

Date objects contain several methods for setting, getting, and manipulating dates and times.

Creating Date Objects

You can create a new Date object by passing the string representation of a date to the Date() constructor, or by passing the individual components of a date as parameters to the Date() constructor (year, month, day, hour, minute, second, millisecond).

The following examples create date objects:

- // Create a date object for today's date: const now = new Date(); //
- Outputs today's date and time in standard format console.log(now); //
- Outputs the day, date, month and year console.log(now.toDateString()); //
- Outputs the hour, minute and second console.log(now.toTimeString());
- Outputs the hour, minute, second and milliseconds console.log(now.toLocaleString());
- Outputs the year console.log(now.getFullYear());
- Outputs the month (from 0-11) console.log(now.getMonth());
- Outputs the day (from 1-31) console.log(now.getDate());
- Outputs the day of the week (from 0-6) console.log(now.getDay());
- Outputs the hour (from 0-23) console.log(now.getHours());
- Outputs the minute (from 0-59) console.log(now.getMinutes());
- Outputs the second (from 0-59) console.log(now.getSeconds());
- Outputs the millisecond (from 0-999) console.log(now.getMilliseconds());

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- Outputs the number of milliseconds since the beginning of the day console.log(now.getTime());
- Outputs the number of seconds since 01.01.1970 console.log(now.getTime() / 1000);

Examples of outputs:

```
Thu, 22 Aug 2019 11:37:45 GMT
Thu Aug 22 2019 11:37:45 GMT+0000 (Coordinated Universal Time)
11:37:45 GMT+0000 (Coordinated Universal Time)
11:37:45 GMT+0000 (Coordinated Universal Time) 2019 8 22 4 11 37 45
Today is a Thursday, the 22nd of August, 2019.
```

You can check a practical example here.

The Date() constructor can also take an integer value representing the number of milliseconds since 01.01.1970, or the string representation of a date.

If you pass an integer value representing the number of milliseconds since 01.01.1970, the constructor will create a date object representing this date and time.

If you pass a string representation of a date, the constructor will try to parse this date and create a date object representing this date and time. If the string cannot be parsed, the constructor will create an invalid date object.

Creating Date Objects With Moment.js

Moment.js is a JavaScript library which makes working with dates and times easy.

To use Moment.js in your project, you have to install it first with a package manager such as npm or yarn, as follows:

```
npm install moment yarn add moment
```

After the installation, you can import Moment.js into your JavaScript file with the import statement:

```
import moment from 'moment';
```

In the following example we use Moment.js to create a date object for today's date and time, and then we use the format() method to output the date and time in different formats:

```
import moment from 'moment'; // Create a date object for today's date and time const now = moment(); // Outputs today's date and time in standard format console.log(now.format()); // Outputs the day, date, month and year console.log(now.format('dddd, MMMM D, YYYY')); // Outputs the hour, minute and second console.log(now.format('h:mm:ss a')); // Outputs the day, date, month, year, hour, minute and // second console.log(now.format('ddd, hA'));
```

Outputs:

```
2019-08-22T15:25:33+04:00 Thursday, August 22, 2019 3:25:33 pm Thu, 3PM
```

Date Arithmetic

Date objects can be manipulated to change dates and times. Date objects have a set() method which can be used to set the year, month, day, hour, minute, second, and millisecond of a date.

When using set() , the object is updated in-place, thus you don't have to create a new Date object.

The following example uses the set() method to set the hour and minute of a date:

```
const now = new Date(); // Outputs the hour, minute and second console.log(now.toLocaleString()); // Changes the hour to 12 and the
```

```
minute to 45 now.setHours(12); now.setMinutes(45);
console.log(now.toLocaleString());
```

Outputs:

```
11:56:09 AM 12:45:09 PM
```

The setHours() method can also be used to set the hour and minute, and the setMinutes() method can also be used to set the minute and second.

The Date object also has getters to get the year, month, day, hour, minute, second and millisecond of a date.

The following example uses the get() method to get the hour and minute of a date:

```
const now = new Date(); // Outputs the hour, minute and second
console.log(now.toLocaleString()); // Outputs the hour and minute
console.log(now.getHours() + ':' + now.getMinutes());
```

Outputs:

```
11:56:09 AM 11:56
```

When using the set() method, the object is updated in-place, thus you don't have to create a new Date object.

The setDate() method can be used to set the day of a date. The setMonth() method can be used to set the month of a date. The setFullYear() method can be used to set the year of a date.

The getDate() method can be used to get the day of a date. The getMonth() method can be used to get the month of a date. The getFullYear() can be used to get the year of a date.

The following example uses the set() , setDate() , setMonth() , setFullYear() , get() , getDate() , getMonth() and getFullYear() methods to set and get the year, month and day of a date:

```
const now = new Date(); now.setDate(1); // Sets the day to the first
day of the month now.setMonth(0); // Sets the month to January
now.setFullYear(2019); // Sets the year to 2019
console.log(now.toLocaleString()); now.setDate(now.getDate() + 10); //
Adds 10
```

JavaScript Math Operations

You can perform numerical operations using JavaScript. This can be done with the help of the + (addition), - (subtraction), * (multiplication), / (division) and % (modulus) operators.

In the example below, we use the addition operator to add two numbers:

```
var x = 10; var y = 5; var z = x + y;
```

This will give the value of z as 15.

Similarly, we can use the - (subtraction), * (multiplication), / (division) and % (modulus) operators to perform their respective operations.

The modulus operator returns the remainder of a division operation. For example, if we divide 10 by 3, the remainder will be 1. Therefore, 10 % 3 will give the output 1.

Apart from these basic arithmetic operations, JavaScript also provides some built-in mathematical functions. These functions can be used to perform more complex operations.

Some of the most commonly used mathematical functions are:

- Math.abs(x) : This function returns the absolute value of a number. For example, Math.abs(-10) will return 10.
- Math.ceil(x) : This function rounds a number up to the nearest integer value. For example, Math.ceil(4.7) will return 5.
- Math.floor(x) : This function rounds a number down to the nearest integer value. For example, Math.floor(4.7) will return 4.
- Math.max(x, y, ...) : This function returns the maximum of the arguments passed to it. For example, Math.max(10, 5, 20) will return 20.
- Math.min(x, y, ...) : This function returns the minimum of the arguments passed to it. For example, Math.min(10, 5, 20) will return 5.
- Math.pow(x, y) : This function returns the value of x raised to the power of y. For example, Math.pow(2, 3) will return 8.
- Math.random() : This function returns a random number between 0 and 1.
- Math.sqrt(x) : This function returns the square root of x. For example, Math.sqrt(16) will return 4.

You can learn more about the Math object and its properties and methods from the JavaScript Math reference.

JavaScript Type Conversions

You can convert a value to a specific data type using built-in methods, such as parseInt() for converting a value to an integer, or parseFloat() for converting a value to a float.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

You can also use the Number() method to convert a value to a number, or the Boolean() method to convert a value to a boolean.

In the following example we will convert a string to a number using the Number() method:

```
var x = "100"; var y = Number(x); console.log(y); // 100
```

In this example we have a string with the value "100". We convert this string to a number using the Number() method, and then we print the value of the number to the console.

You can also use the unary + operator to convert a value to a number. For example:

```
var x = "100"; var y = +x; console.log(y); // 100
```

In this example we have a string with the value "100". We use the unary + operator to convert the string to a number, and then we print the value of the number to the console.

You can use the parseInt() method to convert a string to an integer, or the parseFloat() method to convert a string to a float.

For example:

```
var x = "100.50"; var y = parseInt(x); console.log(y); // 100 var z =
parseFloat(x); console.log(z); // 100.5
```

In this example we have a string with the value "100.50". We use the parseInt() method to convert the string to an integer, and then we print the value of the integer to the console.

We also use the parseFloat() method to convert the string to a float, and then we print the value of the float to the console.

You can use the Boolean() method to convert a value to a boolean.

For example:

```
var x = "100"; var y = Boolean(x); console.log(y); // true
```

In this example we have a string with the value "100". We convert the string to a boolean using the Boolean() method, and then we print the value of the boolean to the console.

You can use the !! operator to convert a value to a boolean. For example:

```
var x = "100"; var y = !!x; console.log(y); // true
```

In this example we have a string with the value "100". We use the !! operator to convert the string to a boolean, and then we print the value of the boolean to the console.

If you want to convert a value to a string, you can use the toString() method.

For example:

```
var x = 100; var y = x.toString(); console.log(y); // "100"
```

In this example we have a number with the value 100. We convert the number to a string using the toString() method, and then we print the string to the console.

If you want to convert a value to an array, you can use the Array.from() method.

For example:

```
var x = "100"; var y = Array.from(x); console.log(y); // [ "1", "0", "0" ]
```

In this example we have a string with the value "100". We convert the string to an array using the Array.from() method, and then we print the array to the console.

If you want to convert a value to an object, you can use the Object() method.

For example:

```
var x = "100"; var y = Object(x); console.log(y); // { "0": "1", "1": "0", "2": "0" }
```

In this example we have a string with the value "100". We convert the string to an object using the Object() method, and then we print the object to the console.

JavaScript Event Listeners

DOM Event Listeners

JavaScript Event Listeners allow you to define functions that will run when a specific event occurs on an element in the DOM.

There are a number of different events that can occur on a DOM element, such as when the element is clicked, hovered over, or even when the element's contents are changed.

To add an event listener to an element, you first need to select the element using one of the DOM selection methods, such as document.querySelector() or document.getElementById() .

Once the element is selected, you can use the addEventListener() method to attach an event listener to the element.

The addEventListener() method takes two arguments: the name of the event to listen for, and a function to run when the event occurs.

For example, to add a click event listener to a button element, you would use the following code:

```
button . addEventListener ( "click" , function ( ) { console . log (
"The button was clicked!" ) ; } ) ;
```

In the code above, when the button element is clicked, the function passed to the addEventListener() method will be executed.

You can also pass a named function to the addEventListener() method, instead of an anonymous function:

```
function handleClick ( ) { console . log ( "The button was clicked!" )
; } button . addEventListener ( "click" , handleClick ) ;
```

DOM Event Types

There are a number of different events that can occur on a DOM element.

The most common events are:

- click
- mouseover
- mouseout
- keypress
- change
- submit

You can find a complete list of DOM events on the Mozilla Developer Network.

DOM Event Object

When an event listener function is called, it is passed an event object as an argument.

The event object contains information about the event that occurred, such as the type of event, the element that the event occurred on, and any event-specific data.

For example, the click event object contains information about the click event, such as the x and y coordinates of the click.

For example, to log the x and y coordinates of a click event, you would use the following code:

```
button . addEventListener ( "click" , function ( ) { console . log (
"x: " + this . clientX + ", y: " + this . clientY ) ; } ) ;
```

In the code above, the this keyword refers to the event object, and the clientX and clientY properties are used to access the x and y coordinates of the click event.

DOM Event Delegation

DOM Event Delegation is a technique for attaching event listeners to elements that do not yet exist in the DOM.

This is useful when you have a large number of elements that you want to add event listeners to, but do not want to add an event listener to each element individually.

For example, if you have a list of 100 items, and you want to add a click event listener to each item, you could use event delegation to add a single click event listener to the parent element of the list, and have that listener function handle the click events for all of the child elements.

To use event delegation, you first need to select the parent element of the elements you want to add event listeners to.

You can then use the addEventListener() method to attach an event listener to the parent element.

The addEventListener() method takes two arguments: the name of the event to listen for, and a function to run when the event occurs.

The function passed to the addEventListener() method will be executed whenever the event occurs on any of the child elements of the parent element.

For example, to add a click event listener to all of the list items in a list, you would use the following code:

```
var listItems = document . querySelectorAll ( "li" ) ; listItems . forEach ( function ( listItem ) { listItem . addEventListener ( "click" , function ( ) { console . log ( "The list item was clicked!" ) ; } ) ; } ) ;
```

In the code above, a click event listener is added to each list item individually.

This can be inefficient if there are a large number of list items.

Instead, you could use event delegation to add a single click event listener to the parent element of the list items:

```
var list = document . querySelector ( "ul" ) ; list . addEventListener ( "click" , function (
event ) { if ( event . target . tagName === "LI" ) { console . log ( "The list item was clicked!"
) ; } } ) ;
```

In the code above, a click event listener is added to the parent element of the list items.

The function passed to the addEventListener() method checks the tagName property of the element that was clicked to see if it is an <li> element.

If it is an <li> element, then the function logs a message to the console.

This technique can be used to add event listeners to any type of element, not just list items.

For example, you could use event delegation to add a click event listener to all of the buttons in a document:

```
var buttons = document . querySelectorAll ( "button" ) ; buttons . forEach ( function (
button ) { button . addEventListener ( "click" , function ( ) { console . log ( "The button
was clicked!" ) ; } ) ; } ) ;
```

Or you could use event delegation to add a click event listener to all of the links in a document:

```
var links = document . querySelectorAll ( "a" ) ; links . forEach ( function ( link ) { link .
addEventListener ( "click" , function ( ) { console . log ( "The link was clicked!" ) ; } ) ; } )
;
```

Event Delegation Summary

In this tutorial you learned about DOM event listeners in JavaScript.

You learned how to use the addEventListener() method to attach event listeners to elements in the DOM.

You also learned about the different events that can occur on a DOM element, and how to access the event object from within an event listener function.

Finally, you learned about DOM event delegation, and how to use it to attach event listeners to elements that do not yet exist in the DOM.

JavaScript Event Propagation

When an event occurs on an element, that event can be recognized by JavaScript and acted upon.

For example, if you click on a button on a web page, you can program JavaScript to recognize that event and take some action.

The event that occurred is a click event.

When you click on an element, the click event is fired on the element that was clicked.

The event then propagates up the DOM tree.

This means that if there is an event handler on a parent element, that event handler will be executed.

If there is an event handler on a grandparent element, that event handler will be executed.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

This propagation up the DOM tree continues until the event reaches the root element of the DOM tree or until the event is stopped.

When the event reaches the root element, the event has bubbled all the way up the DOM tree.

The event will then propagate back down the DOM tree in the same way, from the root element to the element that was clicked.

This is called event **bubbling**. The following diagram shows event bubbling.

You can handle events on any element in the DOM tree.

You can also stop the propagation of an event.

For example, you might have an event handler on a parent element, and you do not want that event handler to be executed when the event occurs on a child element.

In that case, you can stop the event from propagating up the DOM tree.

This is called event propagation.

The event does not reach the root element, and it does not propagate back down the DOM tree.

You can also stop the propagation of an event at any point in the DOM tree.

For example, you might have an event handler on a grandparent element, and you do not want that event handler to be executed when the event occurs on a grandchild element.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

In that case, you can stop the event from propagating down the DOM tree. This is called event capture.

In summary, when an event occurs on an element, that event can be recognized by JavaScript and acted upon.

The event propagates up the DOM tree, and then back down the DOM tree.

You can handle events on any element in the DOM tree.

You can also stop the propagation of an event.

JavaScript Borrowing Methods

We will use an example of an object that contains a property we need to borrow.

```
var myObject = {

someProperty: “foo”,

};
```

We can borrow the someProperty property from myObject like this:

```
var myProperty = myObject.someProperty;

myProperty will now contain the value “foo”.
```

We can also use the same syntax to borrow methods from objects.

For example, if we have an object with a method we want to borrow:

```
var myObject = {

someMethod: function() {

// do something

}

};
```

We can borrow the someMethod method like this:

```
var myMethod = myObject.someMethod;

We can now call myMethod like a normal function.

 myMethod();
```

JavaScript Hoisting

Hoisting is a behaviour that is unique to JavaScript. It is a concept that is often misunderstood. Many developers believe that JavaScript hoists variables and function declarations to the top of the scope.

This is not entirely accurate. What JavaScript actually does is move declarations to the top of the scope, but not initialization. This can lead to some unexpected behaviour.

Consider the following code:

```
var foo = 1; function bar() { if (!foo) { var foo = 10; } console.log(foo); } bar();
```

What do you think the output of this code will be?

If you guessed 10, you would be wrong. The output of this code is 1.

This is because the declaration of foo is hoisted to the top of the scope, but the initialization is not. When the if statement is executed, foo is undefined and is thus set to 10.

This can be a bit confusing, but it is important to understand how hoisting works in JavaScript. It can help you to avoid some common mistakes.

JavaScript Closures

JavaScript closure is an inner function that has access to the variables in the outer (enclosing) function's scope chain. The closure has three scope chains: it has access to its own scope (variables defined between its curly brackets), it has access to the outer function's variables, and it has access to the global variables.

JavaScript closures are created when the inner function is made within the outer function. Closures are used extensively in JavaScript libraries such as jQuery.

Here is a simple example of a closure in JavaScript:

```
function outerFunction(x) {
  var innerFunction = function(y) {
    return x + y;
  }
  return innerFunction;
}
```

```
var add5 = outerFunction(5);
var add10 = outerFunction(10);

console.log(add5(2)); // 7
console.log(add10(2)); // 12
```

In the example above, we have a function outerFunction which has a single parameter x. This function contains a function innerFunction which has a single parameter y. The innerFunction returns the sum of x and y.

When we call the outerFunction with a value, it returns the innerFunction. When we call the function returned by the outerFunction with a value, it returns the sum of the value we passed to the outerFunction and the value we passed to the innerFunction.

In the example above, we have two closures: add5 and add10. When we call add5(2), it returns 7 because 5 is passed to the outerFunction as the value of x, and 2 is passed to the innerFunction as the value of y. When we call add10(2), it returns 12 because 10 is passed to the outerFunction as the value of x, and 2 is passed to the innerFunction as the value of y.

A closure is a function that makes use of variables defined in outer functions that have previously returned. In the following example, the inner function plus() is making use of the variable num that was defined in the outer function returnNotification():

```
function returnNotification() {
  var num = 42;

  function plus() {
    return num + 1;
  }

  return plus;
}

var result = returnNotification();

console.log(result()); // 43
```

In the example above, we have a function returnNotification(). This function has a local variable num and a function plus(). The plus() function returns the value of num plus 1. The returnNotification() function returns the plus() function.

We assign the return value of the returnNotification() function to the variable result. When we call result(), it calls the plus() function and returns the value of num plus 1.

A closure is a function that makes use of variables defined in outer functions that have previously returned. In the following example, the inner function plus() is making use of the variable num that was defined in the outer function returnNotification():

```
function returnNotification() {
  var num = 42;

  function plus() {
    return num + 1;
  }

  return plus;
}
```

```
var result = returnNotification();

console.log(result()); // 43
```

In the example above, we have a function returnNotification(). This function has a local variable num and a function plus(). The plus() function returns the value of num plus 1. The returnNotification() function returns the plus() function.

We assign the return value of the returnNotification() function to the variable result. When we call result(), it calls the plus() function and returns the value of num plus 1.

Closures are often used in JavaScript libraries such as jQuery. In the following example, we are using the jQuery library to create a button. When the button is clicked, an alert box is displayed with the text "Hello world!":

```
$(function() {
  var button = $('button');
  button.click(function() {
    alert('Hello world!');
  });
});
```

In the example above, we have a button element with an id of "button". We are using the jQuery library to select the button element. We are then using the click() method to register a function that will be executed when the button is clicked.

The function that is executed when the button is clicked is a closure. It has access to the variables in the outer function, which in this case is the button element. When the function is executed, it displays an alert box with the text "Hello world!".

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

JavaScript Strict Mode

What is Strict Mode in JavaScript?

The strict mode is a way to opt in to a restricted variant of JavaScript. Strict mode isn't just a subset: it intentionally has different semantics from normal code. Browsers aren't required to run JS code in strict mode and many don't.

Strict mode makes it easier to write "secure" JavaScript. Strict mode changes previously accepted "bad syntax" into real errors.

SyntaxError: Unexpected eval or arguments in strict mode.

Strict mode also solves some mistakes that silently failed in JavaScript, throwing exceptions.

- Deleting a variable or a function.
- Using a variable that isn't declared.
- Deleting an object's property.
- Attempting to change a read-only property.
- Adding a property to an object that is not extensible.
- Using an object that has been frozen or sealed.
- Writing to a read-only property.
- Attempting to change the DontDelete attribute on a property.
- Assigning a value to a read-only global variable.
- Incrementing or decrementing a read-only property.
- Attempting to change the DontDelete attribute on a function.
- Assigning a value to a read-only global function.
- Attempting to change the prototype of a function.

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

The strict mode also prevents, or throws errors, when "unsafe" actions are taken.

- Using a variable without declaring it.
- Creating a global variable (without using the var statement).
- Deleting a variable or a function.
- Deleting an object's property.
- Using an object that has been frozen or sealed.
- Attempting to change the DontDelete attribute on a property.
- Assigning a value to a read-only global function.
- Incrementing or decrementing a read-only property.
- Attempting to change the prototype of a function.

In order to execute your code in strict mode, you need to add the "use strict"; directive to the beginning of your code. The directive is not a string, but rather a literal expression that appears at the top of a file, script, or function body.

```
"use strict";
```

The directive can be placed at either the beginning of a script or at the beginning of a function.

Example 1: Placing the directive at the beginning of a script.

```
"use strict"; x = 3.14; // This will cause an error because x is not declared.
```

Example 2: Placing the directive at the beginning of a function.

```
function myFunction() { "use strict"; y = 3.14; // This will cause an error because y is not declared. }
```

If the strict mode is used inside a function, the strict mode will only be applied to the function. If the strict mode is used at the top level of a script, the strict mode will be applied to the whole script.

If the strict mode is used at the top level of a script, the strict mode will be applied to the whole script.

If the strict mode is used inside a function, the strict mode will only be applied to the function.

In strict mode, any reference to an undeclared variable will cause a reference error.

```
function myFunction() { "use strict"; x = 3.14; // This will cause an error because x is not declared. }
```

In strict mode, any assignment to a non-writable global variable, a non-writable element of an arguments object, or a non-writable property of this will cause an error.

JavaScript JSON Parsing

What is JSON?

JSON stands for JavaScript Object Notation. It’s a lightweight data-interchange format that allows you to exchange and store the data in an organized, easy-to-access manner. The best thing about it is that its syntax resembles plain English so even if you don’t know anything about programming or coding, there are still chances of understanding what this code means!

Parsing JSON Data in JavaScript

The following example shows how to parse a string and create an object from it. You should always prefer parsing data as shown below:

```
var jsonString = '{"name":"John","age":30,"city":"New York"}'; var obj =
JSON.parse(jsonString); console.log('Name: ',obj['name'],' Age:',obj['age'], ' City:' , obj
['city'] ); // Outputs Name : John, age : 30 and city as New york
```

Parsing Nested JSON Data in JavaScript

The following example shows how to parse nested data from a string into an object using the eval() function (which is not recommended). You can also use the same method for parsing nested objects but you need to be careful while doing so because if there are any errors then it will throw exceptions which might crash your program!

Encoding Data as JSON in JavaScript

The following example shows how to encode data into a string using the JSON.stringify() function.

```
var obj = { name : 'John', age: 30, city:'New York' }; var jsonString = JSON.stringify(obj);
console.log('JSON String is ',jsonString ); // Outputs the stringified object as a JSON
```

JavaScript Error Handling

JavaScript is a very flexible language, which means that there are many ways to write code. This flexibility can lead to errors in your program if you're not careful about how you structure it.

One of the most common mistakes developers make is assuming that all code will execute correctly. This assumption can cause problems because JavaScript is an interpreted language, meaning that each line of code is executed as it's read by the interpreter. If there's an error in your code, the interpreter will stop executing at that point and throw an error message.

This tutorial will show you how to deal with errors gracefully so that your program can continue running even if there are some problems with the input data or other parts of the code.

The first thing you need to do is identify where the potential errors might occur in your code. For example, if you're reading data from a file, there's a possibility that the file doesn't exist or that it's been corrupted. If you're working with user input, there's a chance that the users will enter invalid data. Once you've identified these potential sources of error, you can start writing code to deal with them gracefully.

One common way to deal with errors is to use try/catch blocks. These blocks allow you to "try" some code and "catch" any errors that occur while it's executing. The syntax for a try/catch block looks like this:

```
try { // Code to try goes here } catch (error) { // Code to handle errors goes here }
```

The code inside the try block will be executed first. If no errors occur, the code in the catch block will never be executed. However, if an error does occur, execution will jump directly to the catch block and any subsequent code in the try block will be skipped.

Let's look at an example of how this works. Suppose you have a function that reads data from a file and prints it to the console. However, there's a possibility that the file doesn't exist:

```
function readFile(filename) { try { // Code to read and print the contents of "filename" goes here } catch (error) { console.log("Error reading file: " + error); } }
```

In this example, the readFile function will try to read and print the contents of a file. If an error occurs while reading the file, it will be caught by the catch block and printed to the console.

It's important to note that only errors that occur inside the try block will be caught by the catch block. If there's an error in the catch block itself (for example, if you forget to close a curly brace), it won't be caught and your program will crash.

Another way to deal with errors is to use JavaScript's built-in Error object. The Error object can be used to create custom error messages which can then be passed into a throw statement:

```
throw new Error("This is a custom error message");
```

When this code is executed, an error will be thrown and execution of the current function will stop. Any subsequent code in the function will not be executed.

You can also use the Error object to deal with errors that occur inside asynchronous code. For example, suppose you have a function that makes an HTTP request and returns the response data:

```
function makeRequest(url) { return new Promise((resolve, reject) => { http.get(url, (response) => { // Code to handle the response goes here }); }); }
```

This function returns a promise, which means that the data returned by the HTTP request will be available at some point in the future. If an error occurs while making the request (for example, if the URL is invalid), it will be caught by JavaScript's built-in error handling and rejected with an error message.

This rejection will cause the promise to be returned with an error, which can then be handled by the code that called makeRequest :

```
makeRequest("http://www.example.com") .then((data) => { // Code to handle the data goes here }) .catch((error) => { console.log("Error making request: " + error); });
```

In this example, we've registered a callback function for the "then" event which will be called if the promise is resolved successfully. We've also registered a callback function for the "catch" event which will be called if an error occurs. This allows us to deal with errors gracefully and continue executing our code even if there are some problems with the input data or other parts of the code.

JavaScript Regular Expressions

Regular expressions are a powerful tool used to perform pattern matching and "search-and-replace" functions on text.

A regular expression is simply a sequence of characters that defines a particular search pattern. For example, the following regex would match any string containing an uppercase letter A: /A/.

Regular expressions can be used to perform a wide variety of tasks, such as:

- Extracting information from a given string (e.g., an email address);
- Validating user input (e.g., ensuring that a password is strong enough);
- Searching for and replacing text in a given string (e.g., changing all occurrences of "a" to "b"),
- Formatting text (e.g., adding commas between words),
- Generating random strings (e.g., creating unique IDs or passwords).
- And much more!

In JavaScript, regular expressions are also objects. These objects contain a set of methods that can be used to perform various operations on strings, such as searching for patterns or replacing text.

Creating Regular Expressions in JavaScript

There are two ways to create regular expressions in JavaScript: using a regular expression literal or the RegExp() constructor function.

A regular expression literal is simply a string that contains the pattern you want to match, enclosed within two forward slashes (/). For example:

```
const regex = /abc/ ; console .log(regex); // => /abc/
```

The above code creates a new regular expression object containing the pattern abc and assigns it to the variable regex . The value of this variable can then be used like any other JavaScript object. In particular, we can use its methods for performing various operations on strings.

The RegExp() constructor function can also be used to create regular expressions. This function takes two arguments: the first is a string containing the pattern you want to match, and the second is an optional "flags" argument that specifies certain settings for how the regex should work. For example:

```
const regex = new RegExp ( 'abc' ); console .log(regex); // => /abc/ const flagsRegex = new RegExp ( '123' , 'i' ); console .log(flagsRegex); // => /123/i; i stands for case-insensitive matching
```

The above code creates two new regular expression objects. The first contains the pattern abc and is case-sensitive (the default setting). The second object also has a pattern of 123 but its matching will be case-insensitive, thanks to the i flag that was passed as the second argument to RegExp() .

Note: In JavaScript, regexes created using literal notation are immutable; they cannot have their properties or methods changed after they've been defined. On the other hand, regexes created with RegExp() can be modified at any time because it's just another constructor function like Date , Array , etc.

JavaScript Form Validation

JavaScript is a client-side scripting language, which means that the script runs on your visitor's computer. This allows you to do things like check if an email address has been entered correctly before it gets sent off to the server. It also reduces strain on your server because form data doesn't have to be submitted until after it has been checked for errors by JavaScript.

The first thing you need to do is create a form. You can use the following code:

```
<form action="yourpage.php" method="post"> <p>Name:</p><input type="text"
name="name"><br /> <p>Email Address:</p><input type="text" name= "email"><br />
</form>
```

This will create a basic form with two fields, one for the name and one for an email address. The action attribute tells the browser where to send the data when it is submitted, in this case yourpage.php . You can change this to whatever page you want to process the form data on. The method attribute specifies how the data should be sent, either GET or POST. In most cases you will want to use POST because it is more secure than GET, but there are some situations where using GET may be preferable (for example if you wanted people to be able bookmarked search results). For more information about these methods see our HTML Forms Tutorials: Introduction article which covers them in detail.

The next thing we need do is add a submit button so that visitors can actually send us their details:

```
<input type="submit" value="Submit">
```

You can put this button anywhere inside the form tags. Now if you save your page and try it out, when you click on the submit button a new window will open with just yourpage.php in it (assuming that is what you set as the action). This happens because we haven't told our browser to do anything else yet - at present all submitting the form does is send us to another page which isn't very useful!

We need JavaScript to check for errors before sending off any data so let's add some code:

```
<script type="text/javascript"> function checkForm() { var errorMsg = ""; // Check each
field to make sure it has a value. if (document.form1.name == "") { errorMsg += "- Please
enter your name

"; } else if (document.form1 .email == "" || document .form 1 .email_confirm != document
form 1 email)  {errorMsg += "- You must provide an e-mail address and confirm it by
entering the same address again."; } return true; // This line submits the form only when
there are no errors in any of the fields, otherwise we display our message telling them
what they need to do: alert(errormsg); return false ; </script> <body
onload="checkForm();" >

 ... rest of page here...

</body>
```

JavaScript Cookies

Creating a cookie in JavaScript is very simple. You just have to use the document object and its method createCookie(). The syntax of this function looks like:

```
   document.cookie = "name=value; expires=date";
```

The name parameter represents the name of your cookie, while value stands for its content (string). If you want to set an expiration date on it, then add another attribute called expires with a valid Date() string as value or simply pass null if you don't need one.

```
   document.cookie="username=John Doe;expires"+Date(30); //creates username
cookies that will expire after 30 days from now
   document.cookie="username=John Doe;expires"+Date(30*24); //creates username
cookies that will expire after 30 days from now
```

Reading a cookie in JavaScript is very simple as well, you just have to use the document object and its method getCookie(). The syntax of this function looks like:

```
   var name = getCookie("name");
```

The only parameter represents the name of your cookie (string). This function returns null if no such cookie exists or it has already expired. You can also read all existing cookies using another built-in property called document.cookies which contains an array with all available ones on your page separated by semicolons ";" . For example:

```
   alert(document.cookies)        ;//will        display        something        like        :
userName1=value1;userName2=value2 etc...    /*this code displays nothing because
there are no any active cookiess*/
```

Updating a cookie in JavaScript is very simple as well. You just have to create it again with the same name and value, but this time you also need to set an expiration date that's greater than its current one (if any). For example:

```
   document.cookie = "name=newValue; expires=" + Date(Date().getTime()+1);   /*this code will update your cookies*/
   document.cookie = "username=John Doe;expires"+Date(30*24); //creates username cookies that will expire after 30 days from now   /*this code creates new cookies if there are no active ones or updates existing ones */
```

JavaScript Ajax Requests

Ajax is a web development technique for creating interactive web applications. The goal of Ajax is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire page does not have to be reloaded each time the user makes a change.

In order to use Ajax, you will need to use a JavaScript library such as jQuery or Prototype. These libraries provide an easy way to send and receive data from a server without having to refresh the page.

Once you have included the Ajax library in your web page, you can start making requests to the server. For example, if you wanted to load some data from a file on the server, you would use the following code:

```
$.ajax({ url: 'data.json', success: function(data) { // do something with the data } });
```

This code will make an Ajax request to the URL 'data.json'. If the request is successful, the function in the 'success' callback will be executed with the data from the server as its argument.

If you want to send data to the server, you can use the '$.ajax()' method's 'data' option:

```
$.ajax({ url: 'saveData.php', type: 'POST', data: { name: 'John', age: 20 } });
```

This code will make a POST request to the URL 'saveData.php'. The data that is being sent to the server is specified in the 'data' option as an object. In this example, we are sending two pieces of data: 'name' and 'age'.

The following example will show you how to make an Ajax GET request in JavaScript:

```
$.ajax({ url: 'getData.php', type: 'GET', success: function(data) { // do something with the data } });
```