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

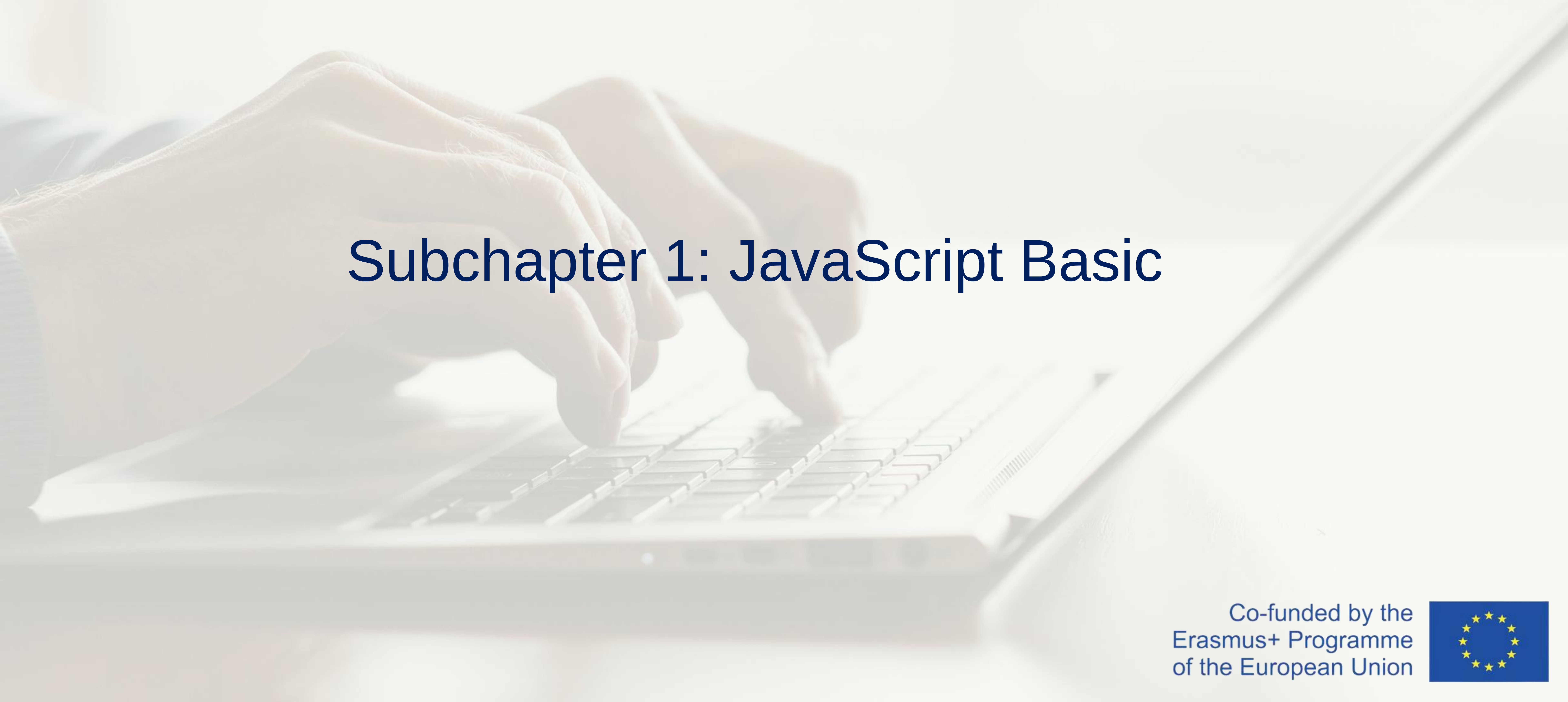
code4sp
coding for social promotion

JavaScript Trainer Materials
Subchapter 1 – JavaScript Basic

WP3: Code4SP Training Materials

Prepared by:

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. (Project Number: 621417-EPP-1-2020-1-PT-EPPKA3-IPI-SOC-IN)



Subchapter 1: JavaScript Basic

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

What is JavaScript?

- 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 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.

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

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:

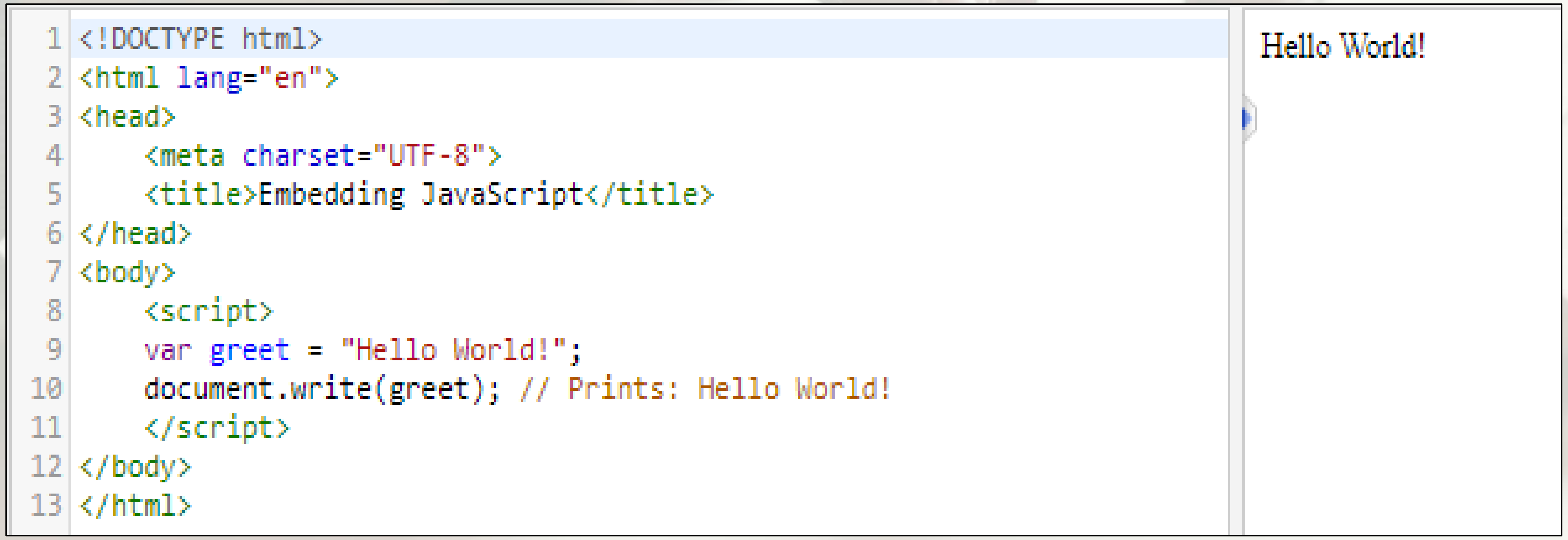
- **Embedding the JavaScript code between a <script> and a </script> tag;**
- **Creating an external JavaScript file with the .js extension and then load it within the page through the src attribute of the <script> tag.**
- **Placing the JavaScript code directly inside an HTML tag using the special tag attributes such as onclick, onmouseover, onkeypress, onload, etc.**

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

Getting started with JavaScript

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!
```

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

Getting started with JavaScript

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:

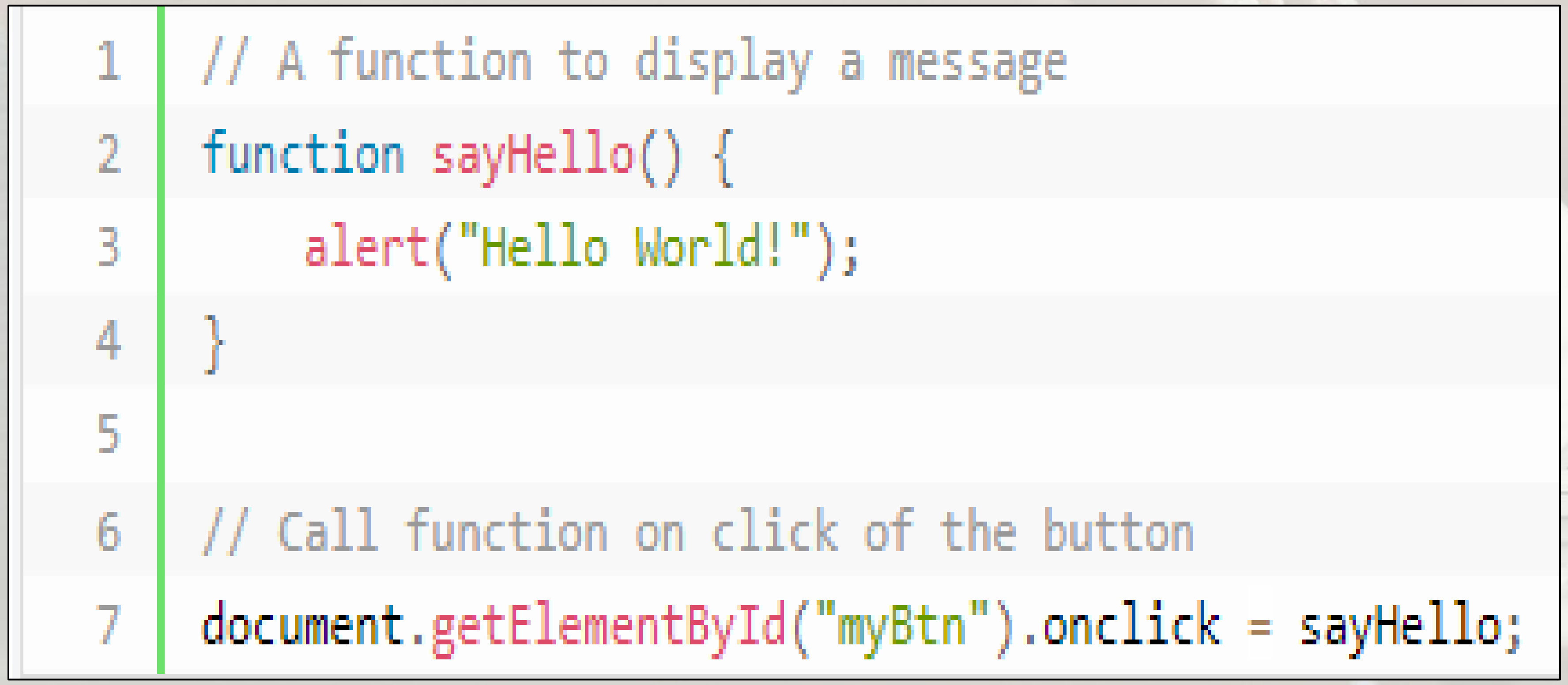
<script src="js/hello.js"></script>

- 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.

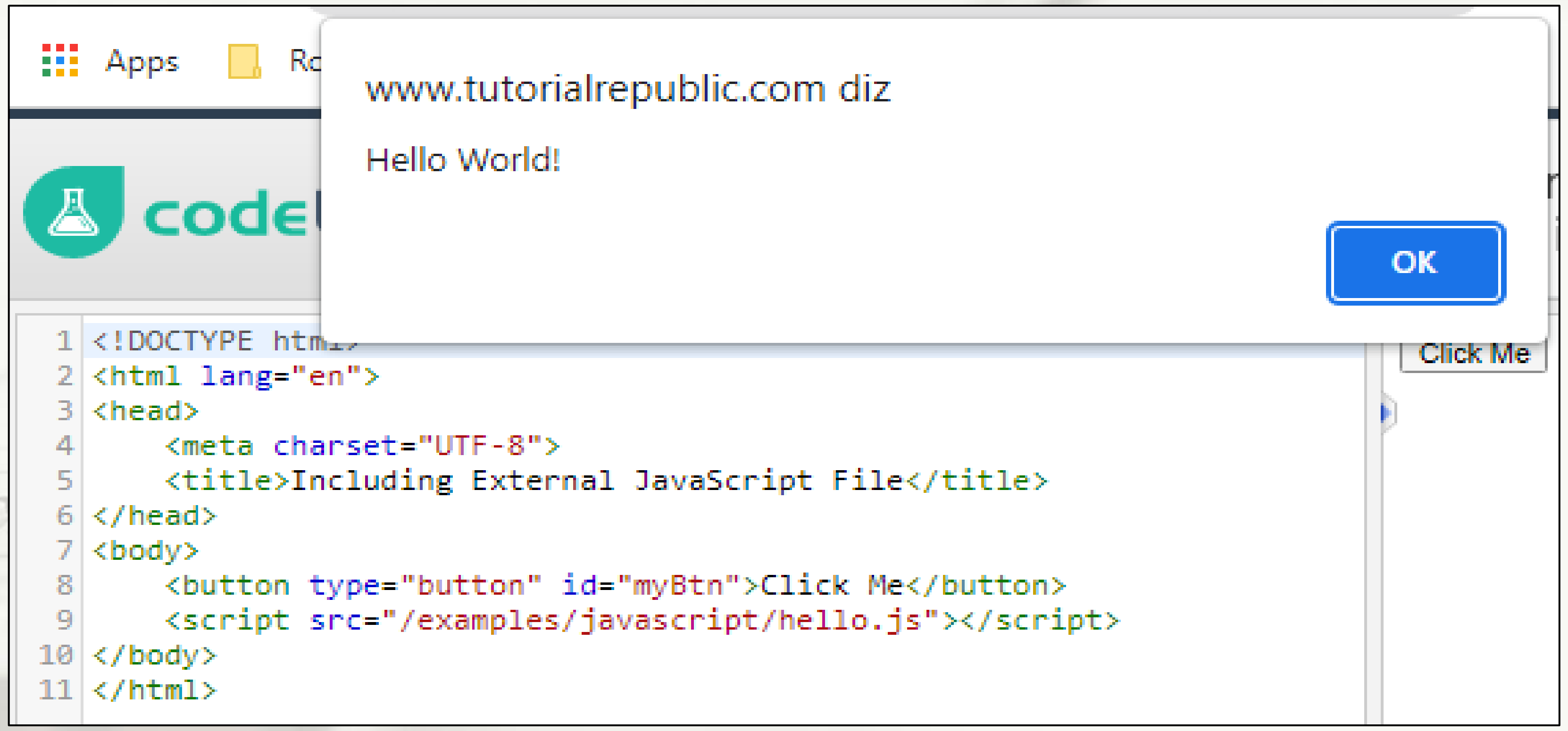
Co-funded by the Erasmus+ Programme of the European Union

Getting started with JavaScript

Forward to the previous statement, a JavaScript file named "**hello.js**" will be created and the following code shall be inserted in it. The result is available in the right picture.



```
// A function to display a message
function sayHello() {
    alert("Hello World!");
}

// Call function on click of the button
document.getElementById("myBtn").onclick = sayHello;
```



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>
```

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

Getting started with JavaScript

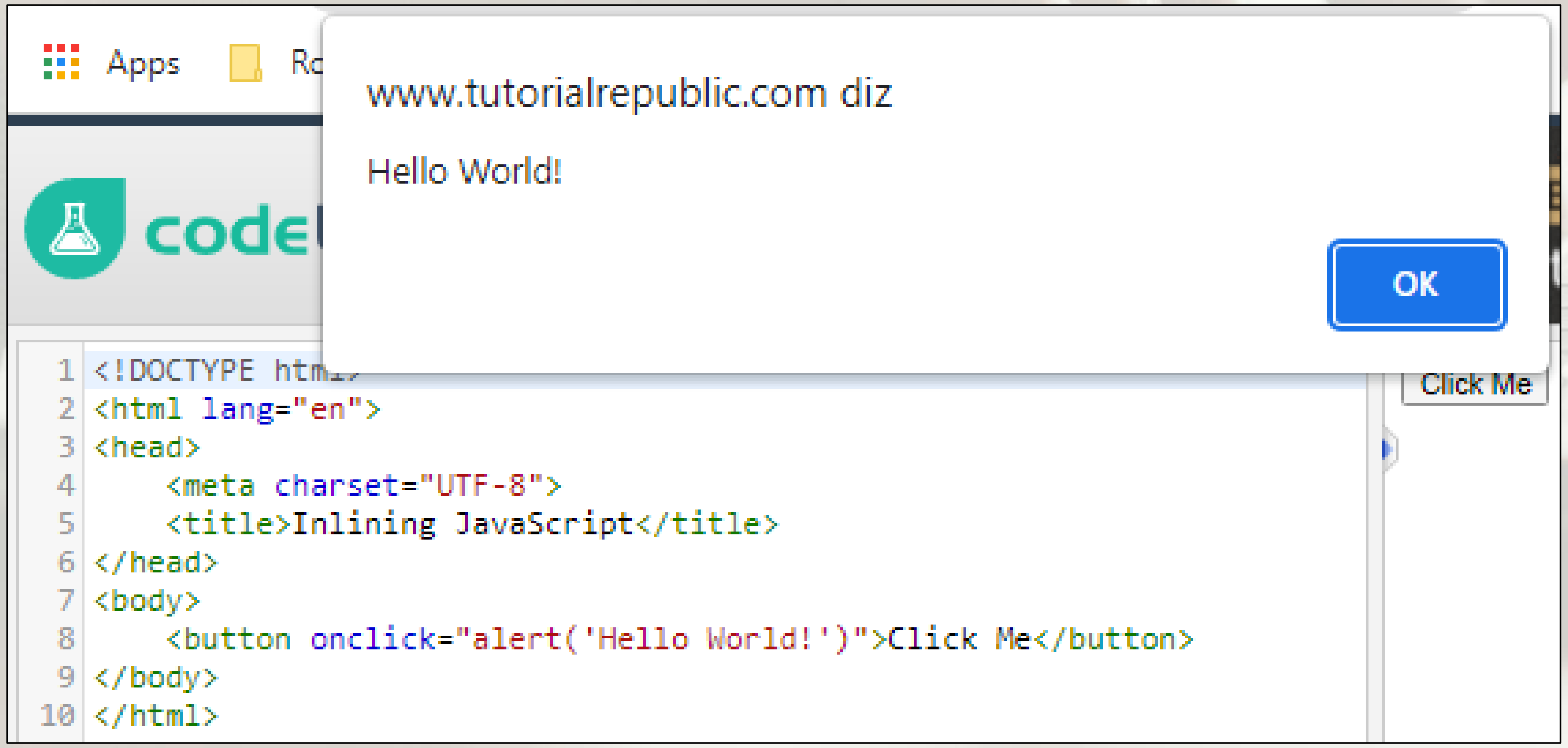
Placing the JavaScript code directly inside an HTML tag using the special tag attributes such as onclick, onmouseover, onkeypress, onload, etc.

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.

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

Getting started with JavaScript

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>
```

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

Getting started with JavaScript

- 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.

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

Getting started with JavaScript

- **There are differences between Client-side and Server-side Scripting.**

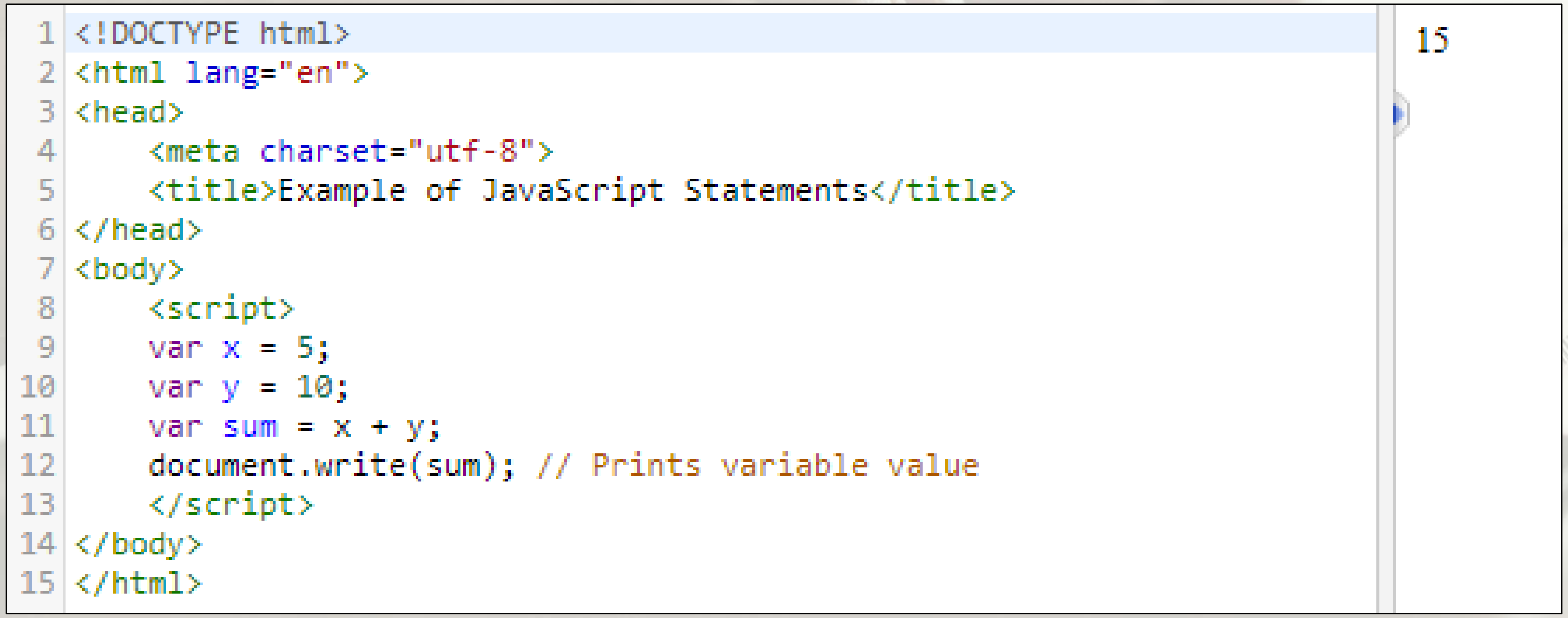
- Client-side scripting languages (e.g., JavaScript or VBScript) are understood and executed by the web browser, in opposite of server-side scripting languages (e.g., PHP, ASP, Java, Python, Ruby, etc.), which run on the web server and their output is sent back to the web browser in HTML format.

- 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.

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

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.



```
<!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
```

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

JavaScript Syntax

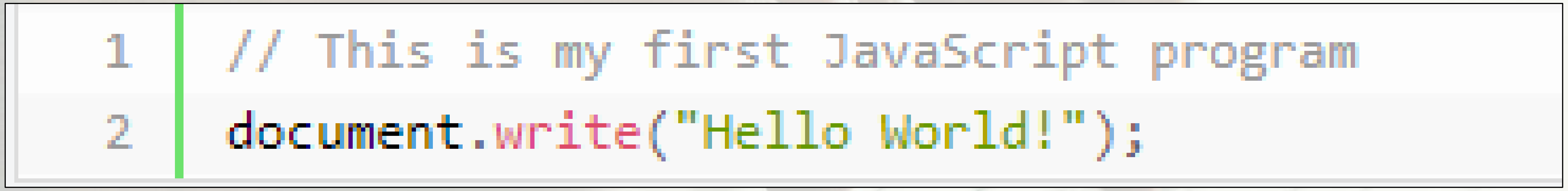
- Learners should state that JavaScript is case-sensitive. So, variables, language keywords, function names and other identifiers must be typed consistently in terms of letters capitalisation. For instance, the variable myVar must be typed this way (not “MYVAR”, “myvar”, etc.). This applies for all cases.

- 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.

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

JavaScript Syntax

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:

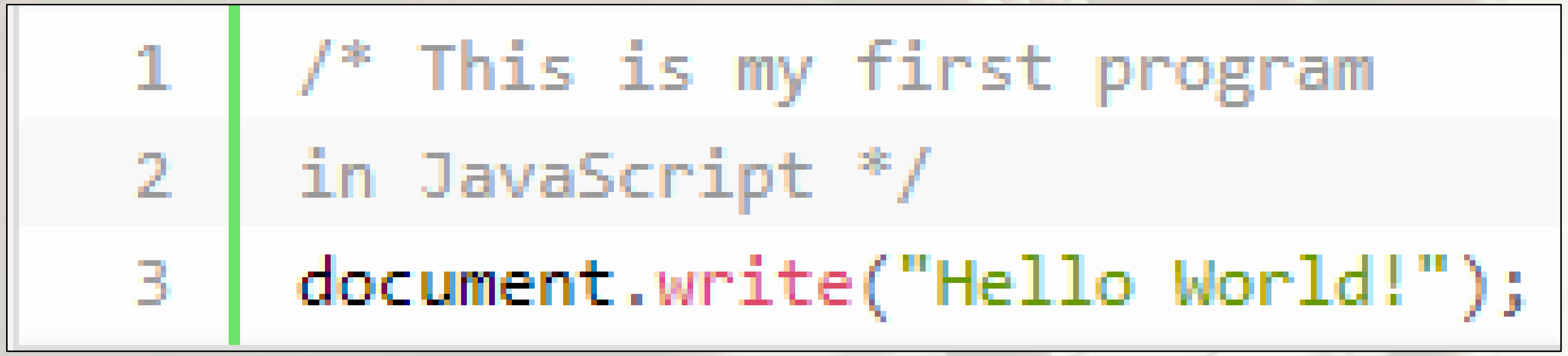


```
// This is my first JavaScript program
document.write("Hello World!");
```

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

JavaScript Syntax

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!");
```

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

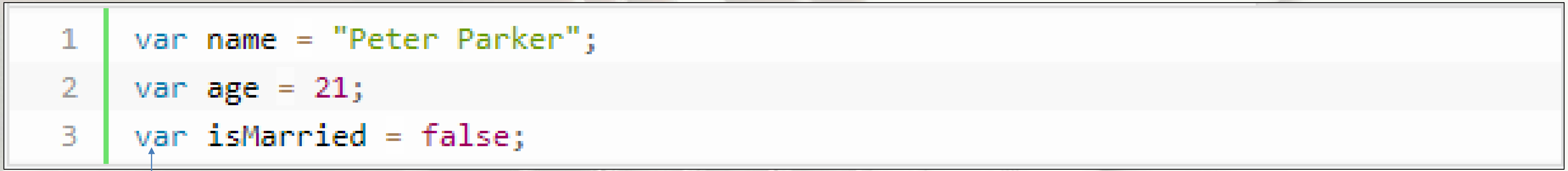
JavaScript Variables

- 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.

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

JavaScript Variables

- 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*



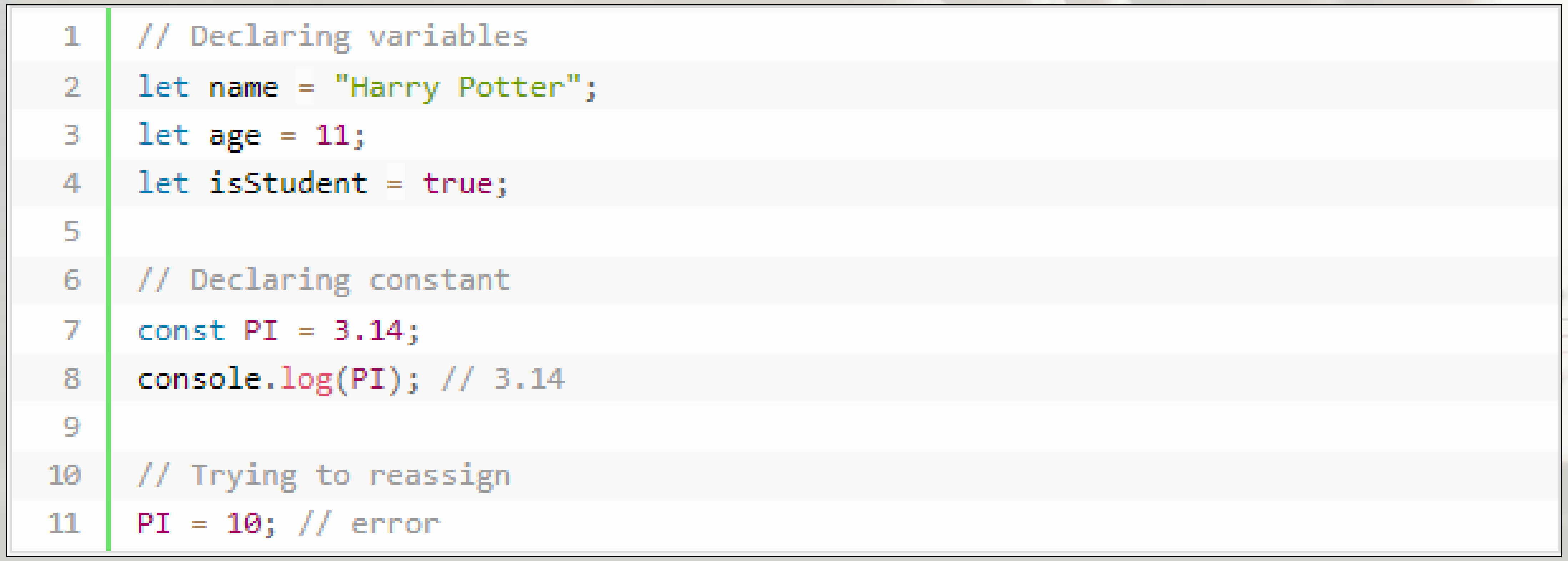
```
var name = "Peter Parker";
var age = 21;
var isMarried = false;
```

3 variables have been created

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

JavaScript Variables

- 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
```

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.

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

JavaScript Variables

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.

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

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.

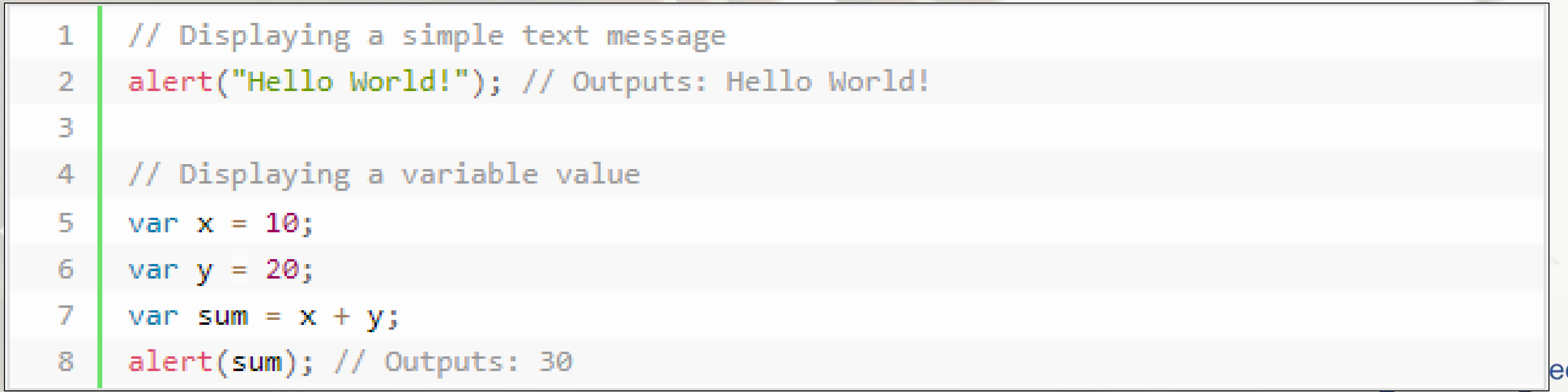
- To write the content to the current document only while that document is being deconstructed, the document.write() method can be used.

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

JavaScript Generating Output

- 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:

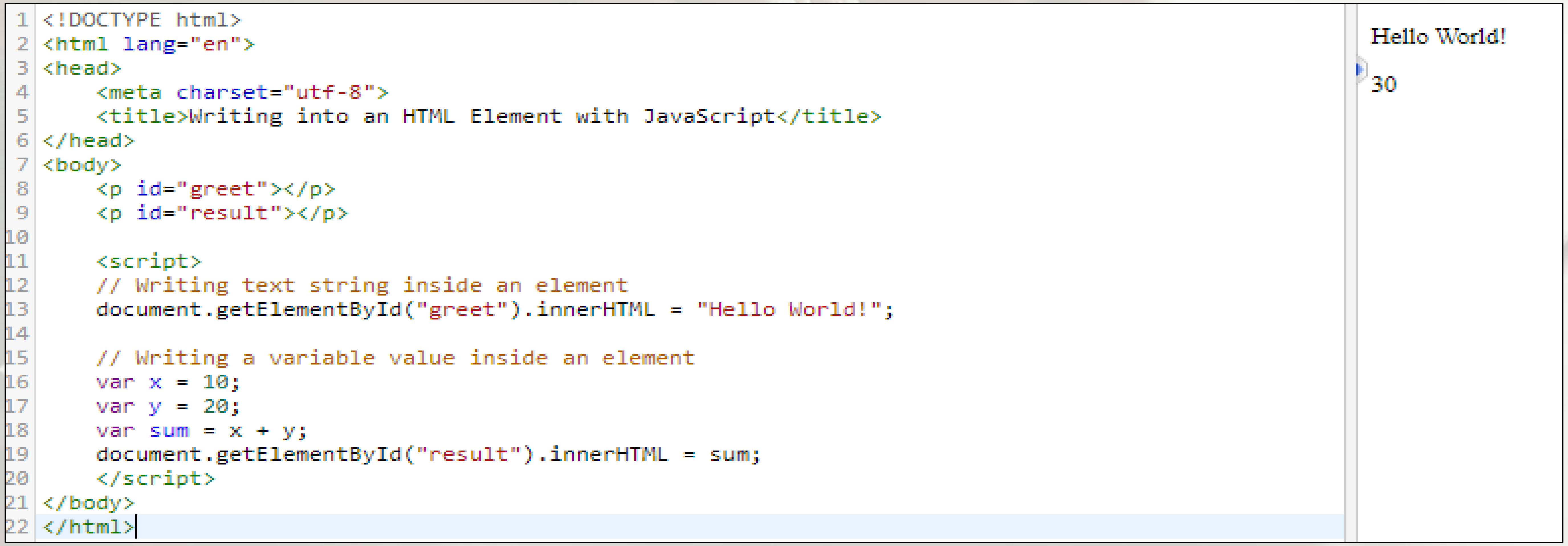


```
// Displaying a simple text message
alert("Hello World!"); // Outputs: Hello World!

// Displaying a variable value
var x = 10;
var y = 20;
var sum = x + y;
alert(sum); // Outputs: 30
```

ed by the
Erasmus+ Programme
of the European Union

JavaScript Generating Output

- 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
```

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

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:

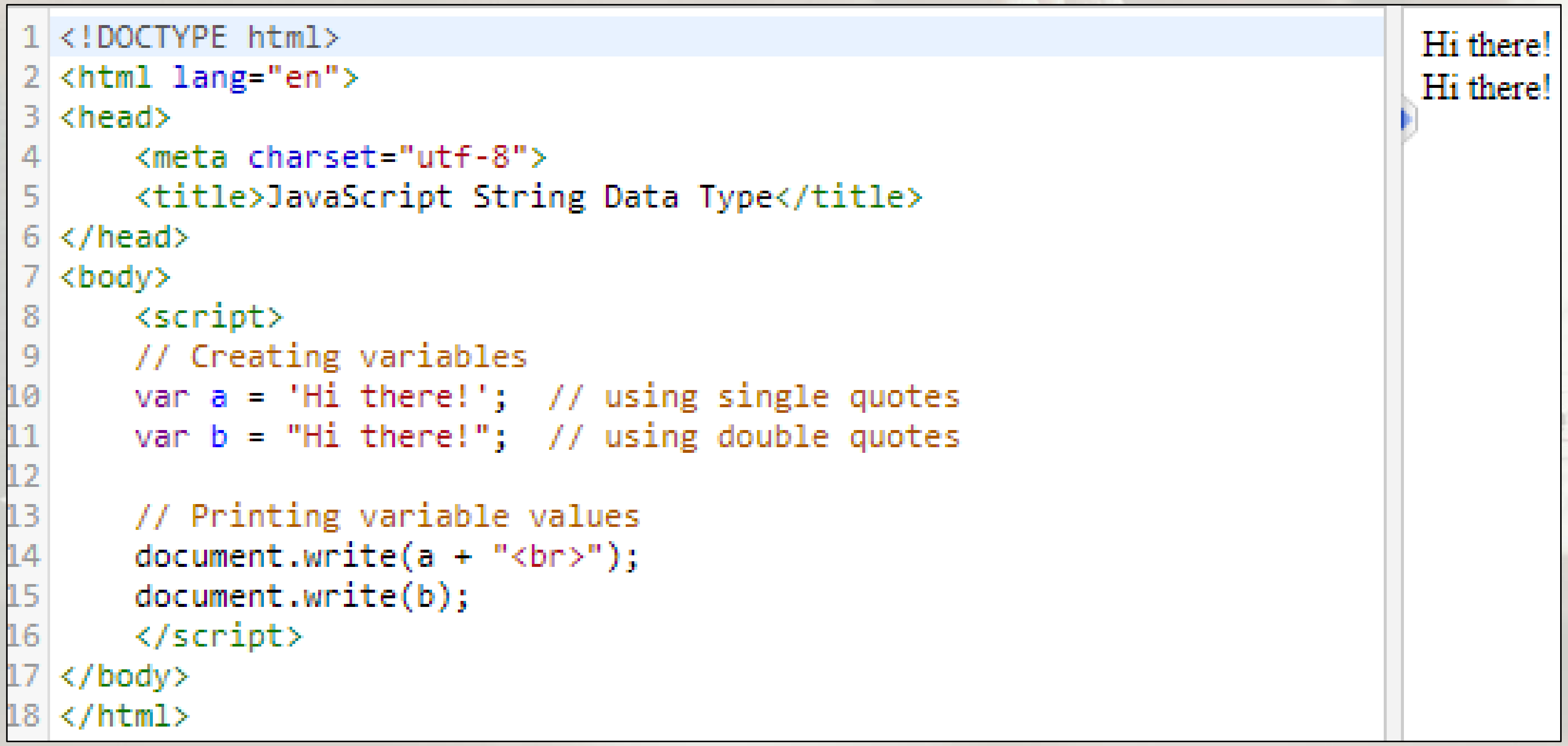
- **Primitive (or primary)** – String, Number, and Boolean are examples of primitive data types, which can hold only one value at a time;
- **Composite (or reference)** – Object, Array, and Function (which are all types of objects) are composite data types. These can hold collections of values and more complex entities; and
- **Special data types** – Undefined and Null are special data types.

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

JavaScript 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>
```

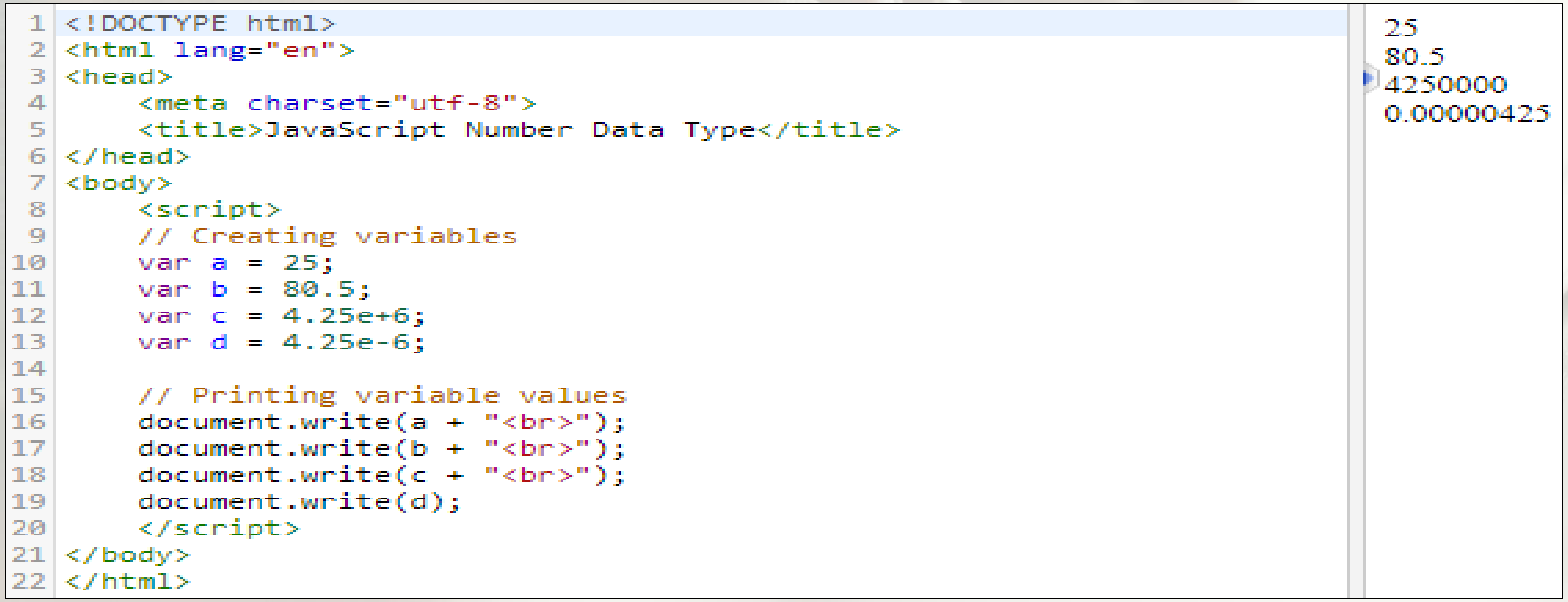
```
Hi there!
Hi there!
```

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

JavaScript Data Types

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 1.5×10^{-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>
```

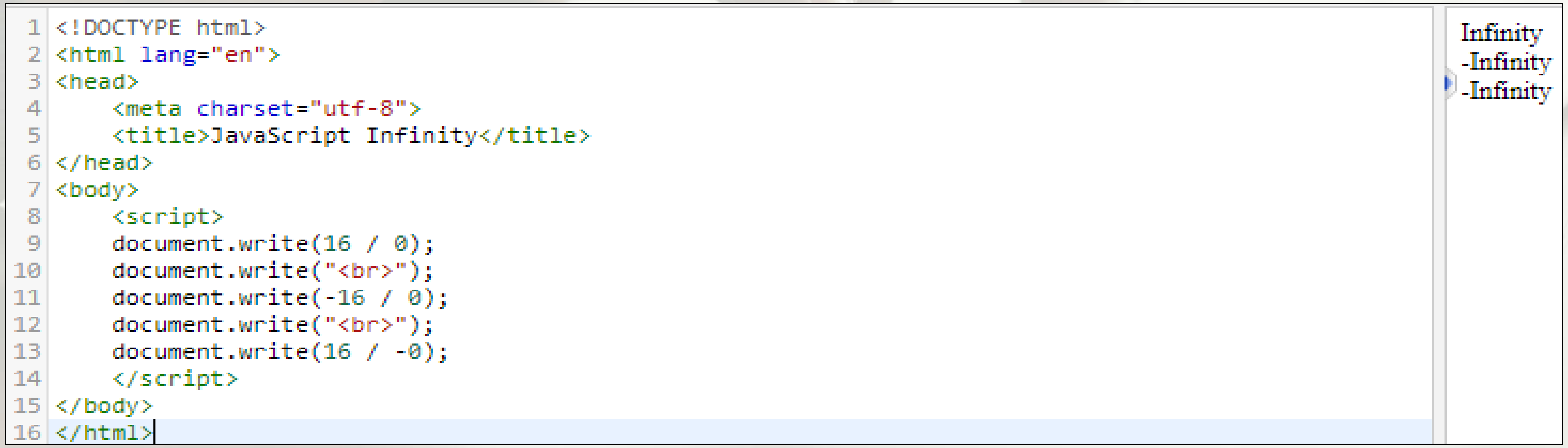
```
25
80.5
4250000
0.00000425
```

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

JavaScript Data Types

The Number Data Type

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 the image below:



```
<!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>
```

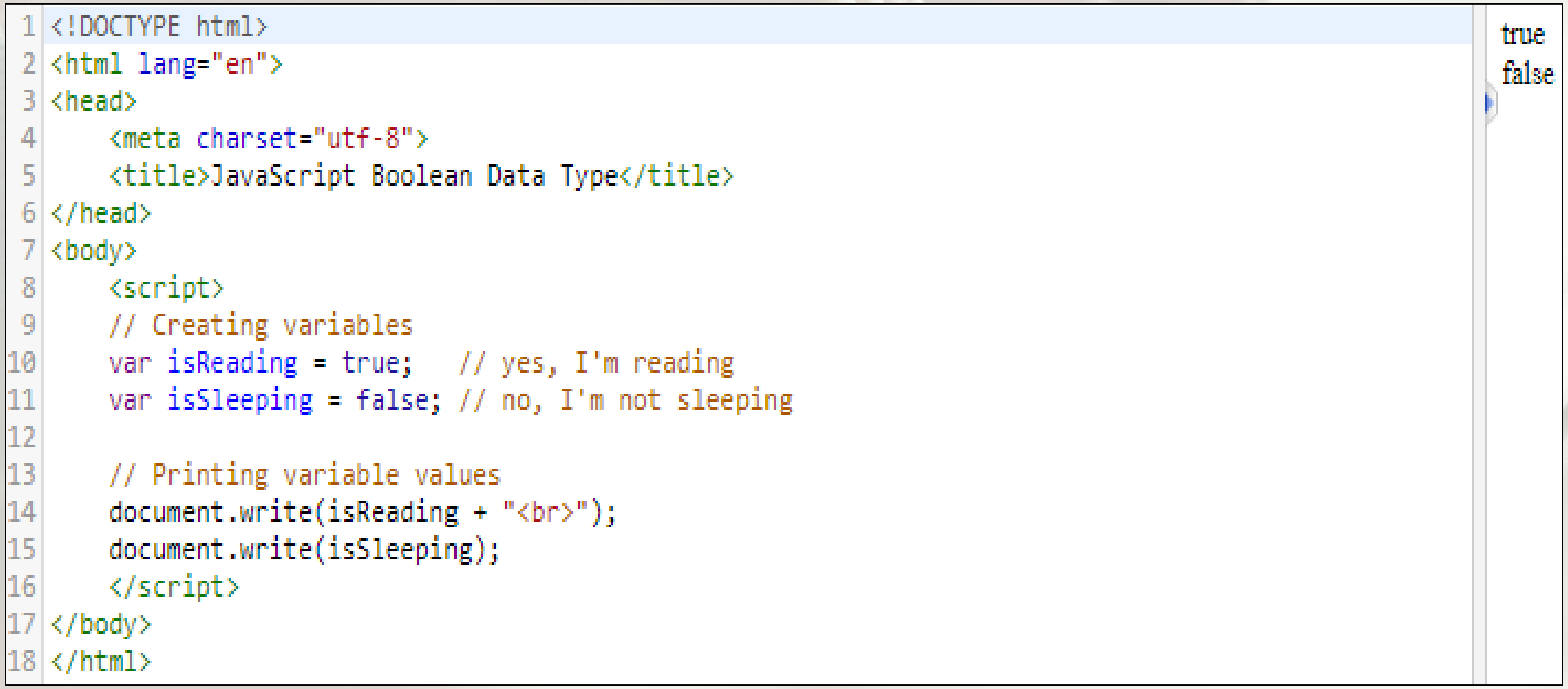
```
Infinity
-Infinity
-Infinity
```

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

JavaScript Data Types

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>
```

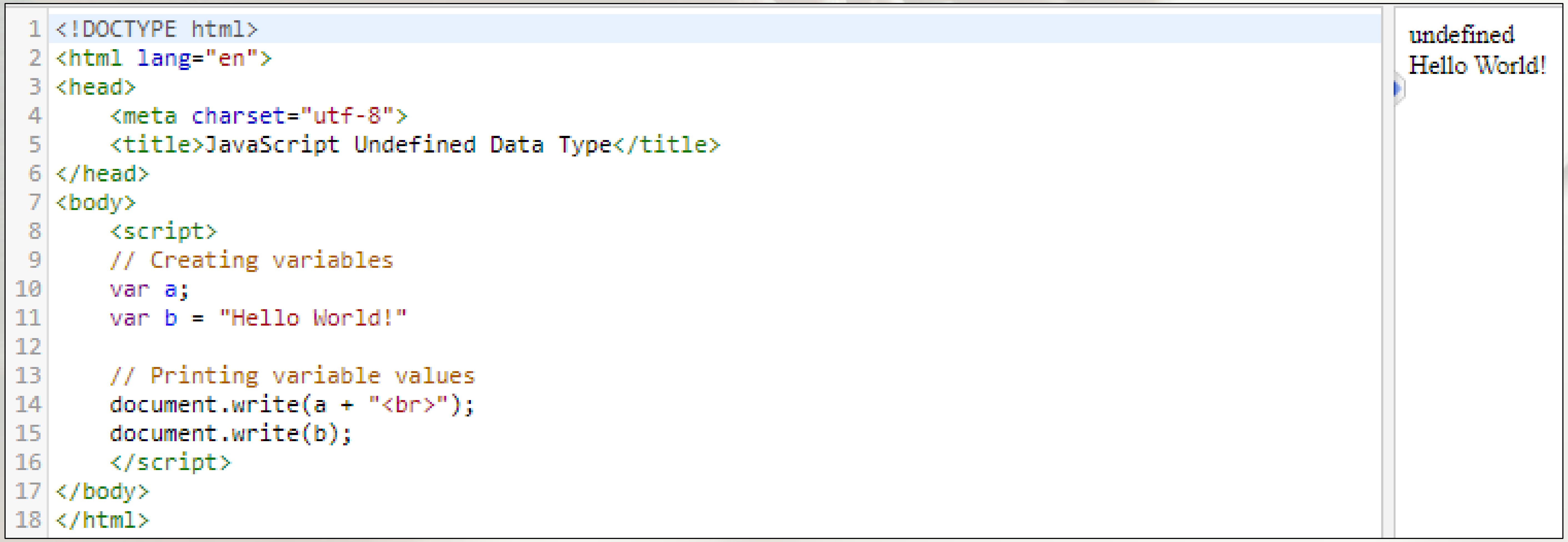
```
true
false
```

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

JavaScript Data Types

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>
```

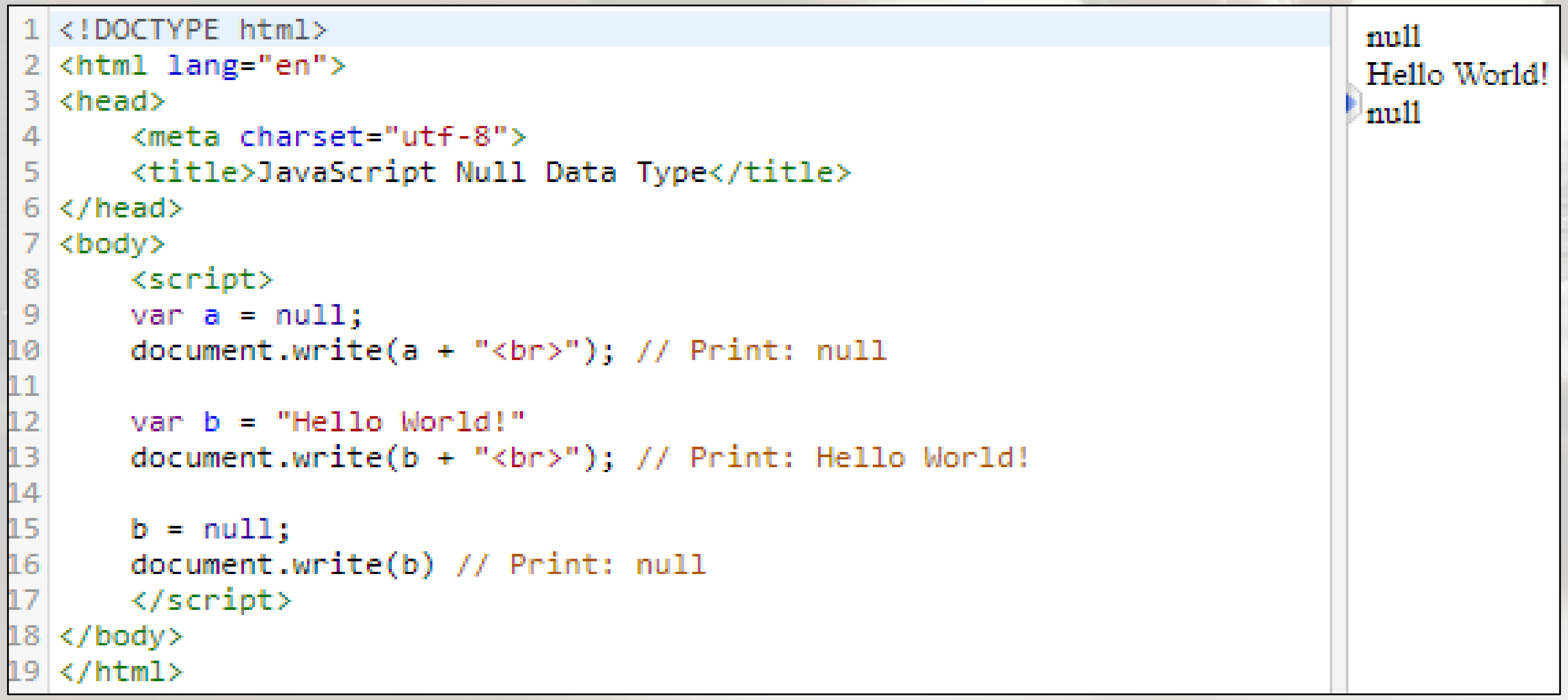
```
undefined
Hello World!
```

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

JavaScript Data Types

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
```

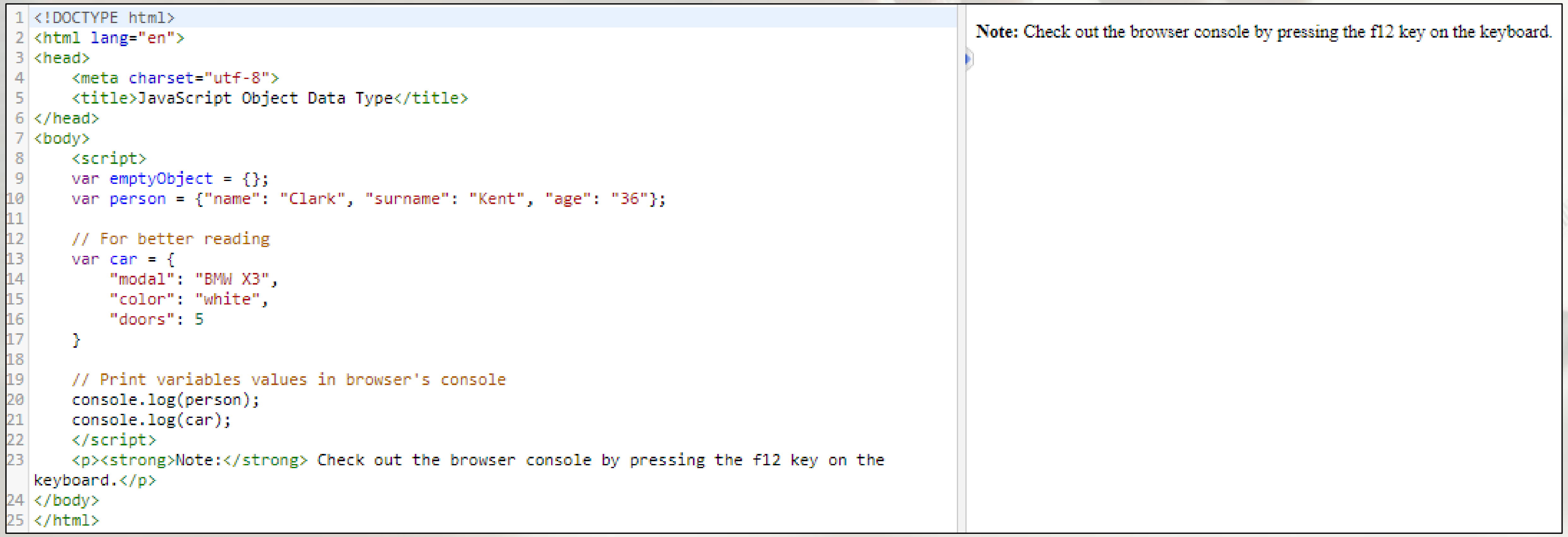
Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Data Types

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>
```

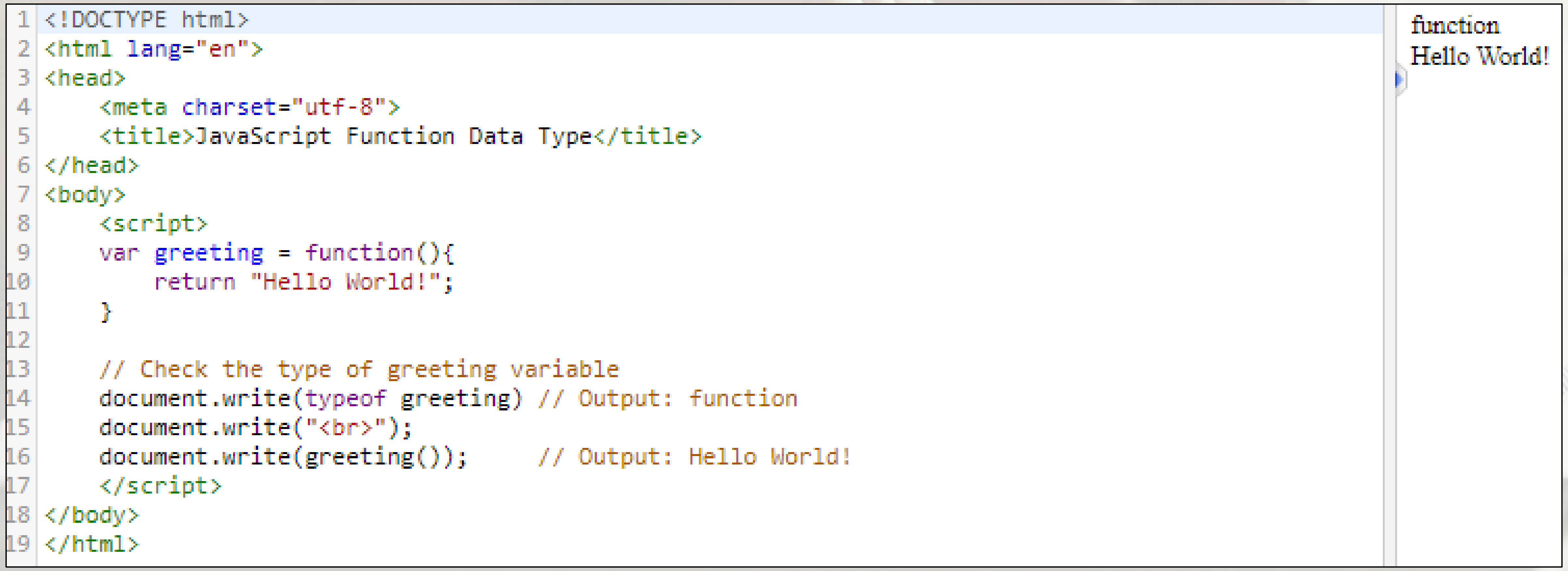
Note: Check out the browser console by pressing the f12 key on the keyboard.

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

JavaScript Data Types

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>
```

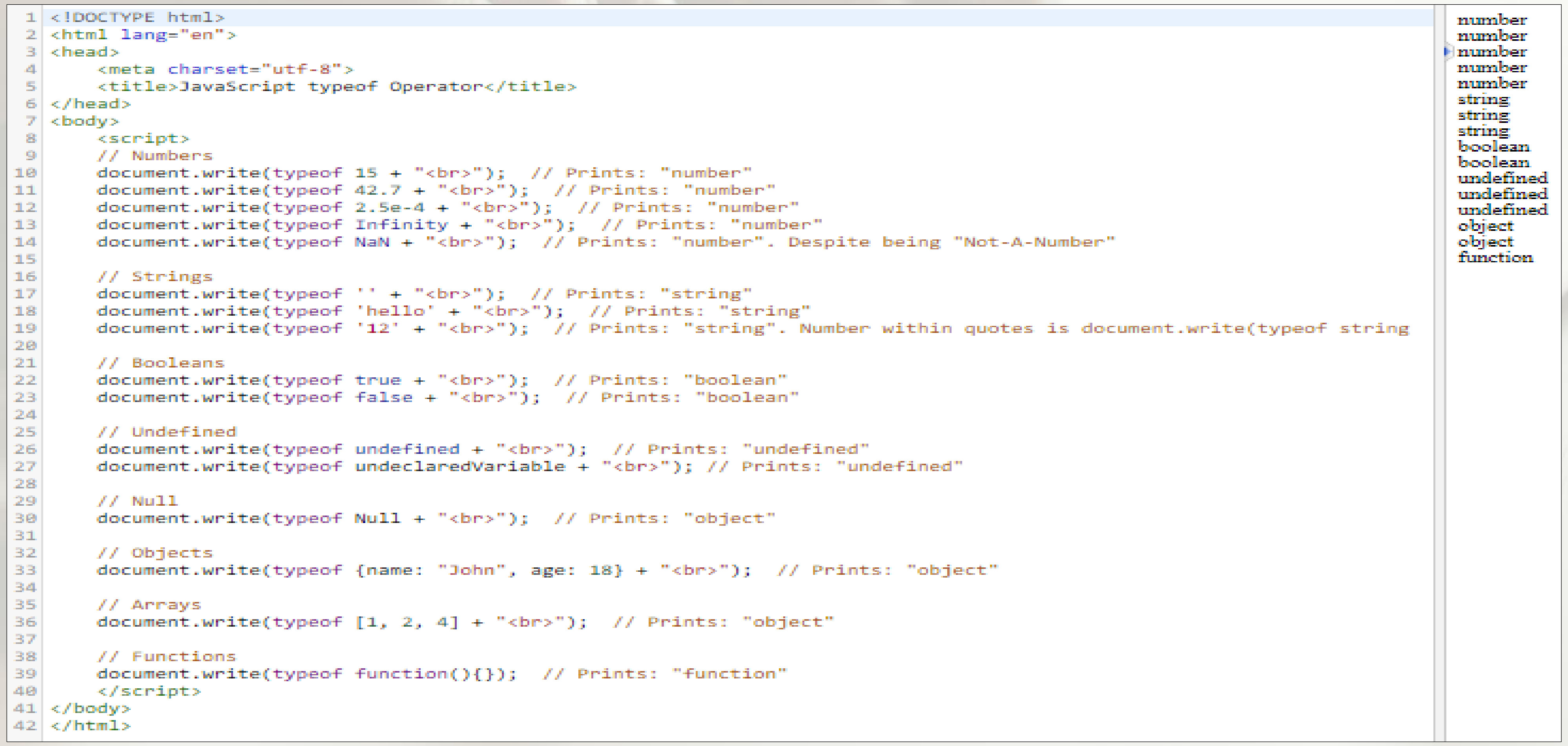
```
function
Hello World!
```

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

JavaScript Data Types

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:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript typeof Operator</title>
</head>
<body>
    <script>
    // Numbers
    document.write(typeof 15 + "<br>");  // Prints: "number"
    document.write(typeof 42.7 + "<br>");  // Prints: "number"
    document.write(typeof 2.5e-4 + "<br>");  // Prints: "number"
    document.write(typeof Infinity + "<br>");  // Prints: "number"
    document.write(typeof NaN + "<br>");  // Prints: "number". Despite being "Not-A-Number"

    // Strings
    document.write(typeof '' + "<br>");  // Prints: "string"
    document.write(typeof 'hello' + "<br>");  // Prints: "string"
    document.write(typeof '12' + "<br>");  // Prints: "string". Number within quotes is document.write(typeof string

    // Booleans
    document.write(typeof true + "<br>");  // Prints: "boolean"
    document.write(typeof false + "<br>");  // Prints: "boolean"

    // Undefined
    document.write(typeof undefined + "<br>");  // Prints: "undefined"
    document.write(typeof undeclaredVariable + "<br>"); // Prints: "undefined"

    // Null
    document.write(typeof Null + "<br>");  // Prints: "object"

    // Objects
    document.write(typeof {name: "John", age: 18} + "<br>");  // Prints: "object"

    // Arrays
    document.write(typeof [1, 2, 4] + "<br>");  // Prints: "object"

    // Functions
    document.write(typeof function(){});  // Prints: "function"
    </script>
</body>
</html>
```

```
number
number
number
number
number
string
string
string
boolean
boolean
undefined
undefined
undefined
object
object
function
```

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

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).

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

JavaScript Operators

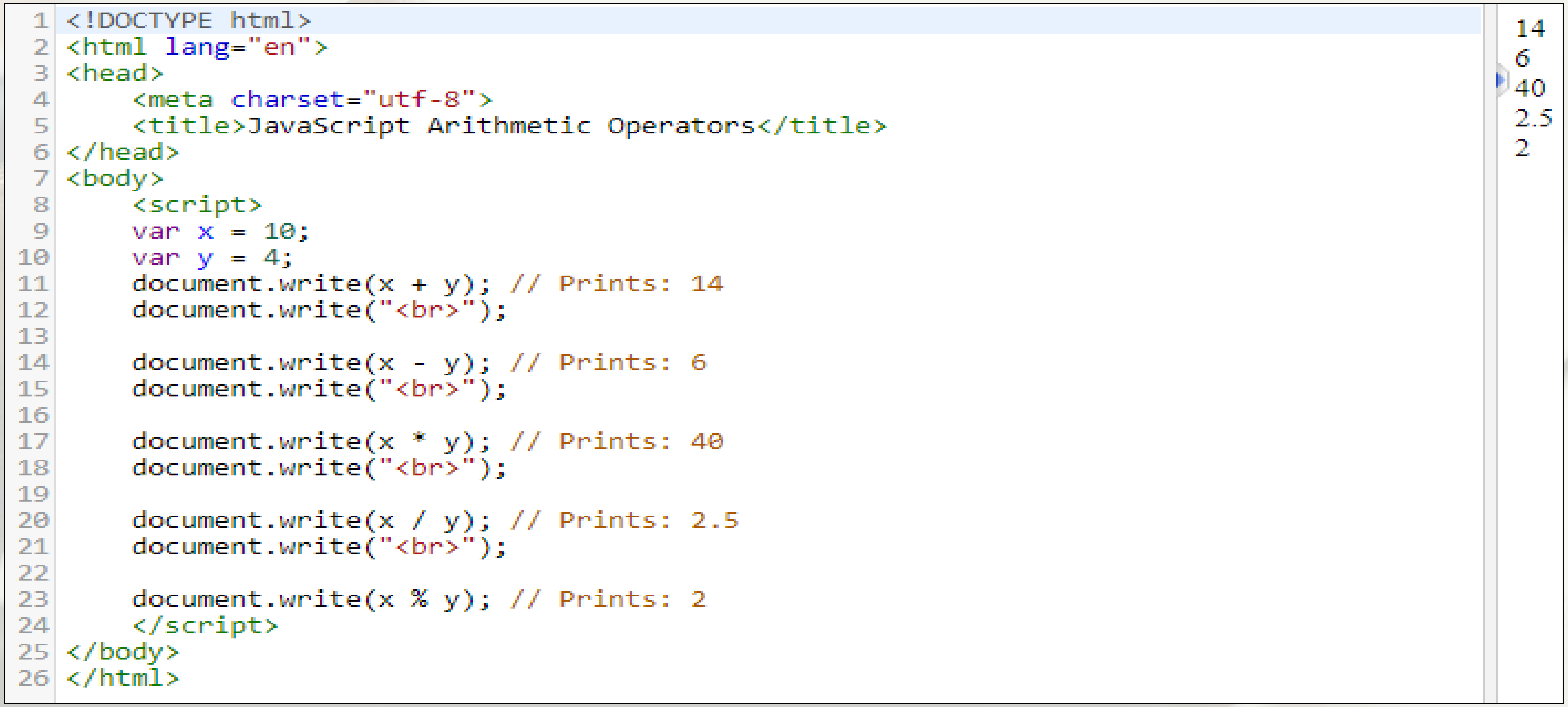
JS Arithmetic Operators

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

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

JavaScript Operators

JS Arithmetic Operators



```
<!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
```

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

JavaScript Operators

JS String Operators

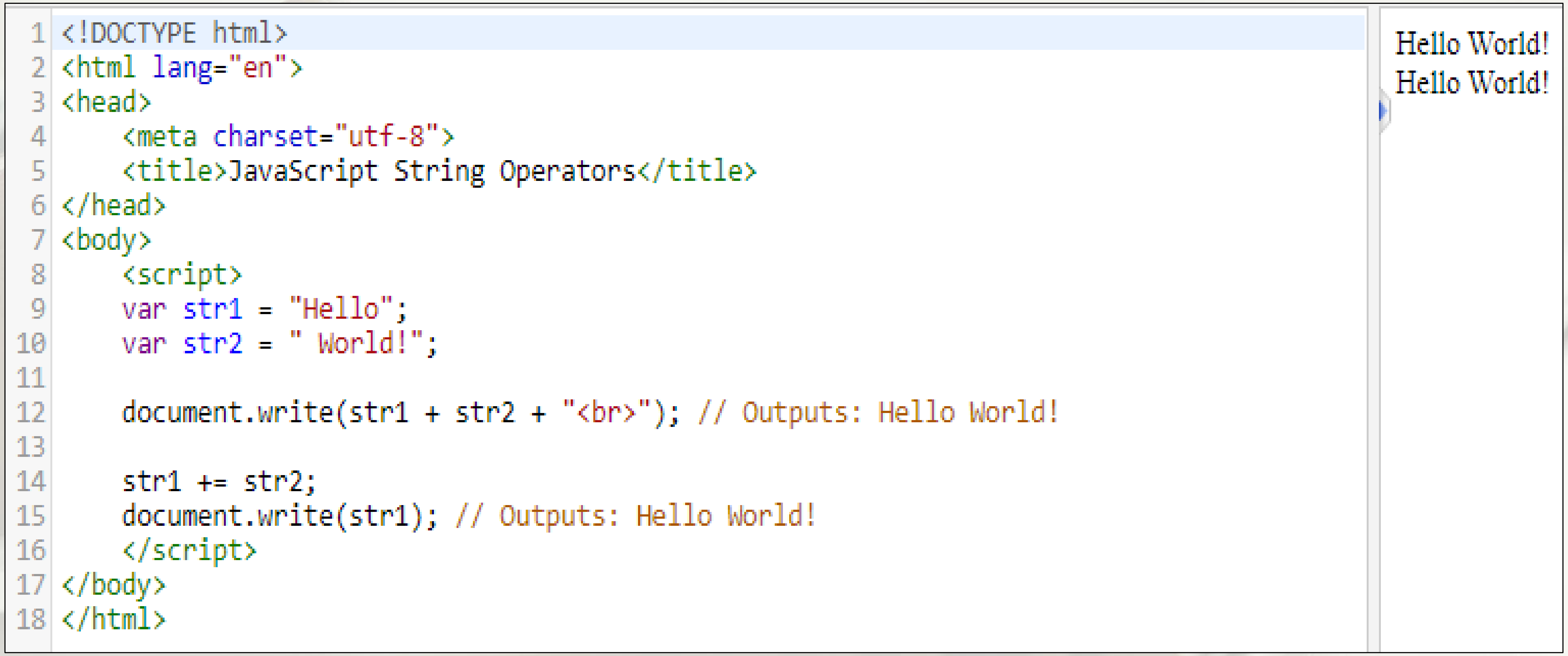
Operator	Description	Example	Result
+	Concatenation	str1 + str2	Concatenation of str1 and str2
+=	Concatenation assignment	str1 += str2	Appends the str2 to the str1

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

code4sp
coding for social promotion

JavaScript Operators

JS String Operators



```
<!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!
```

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

JavaScript Operators

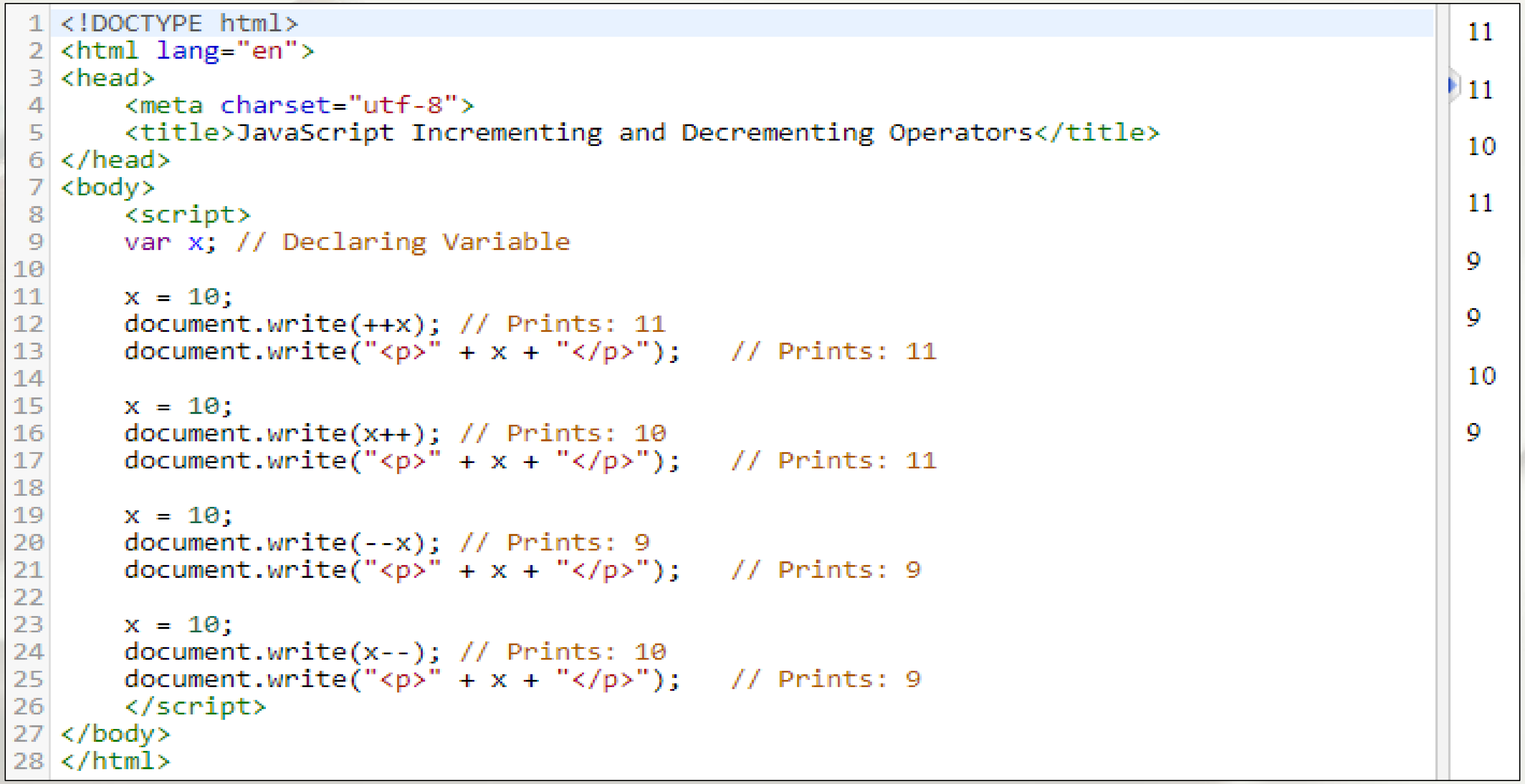
JS Incrementing and Decrementing Operators

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

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

JavaScript Operators

JS Incrementing and Decrementing Operators



```
<!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
```

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

JavaScript Operators

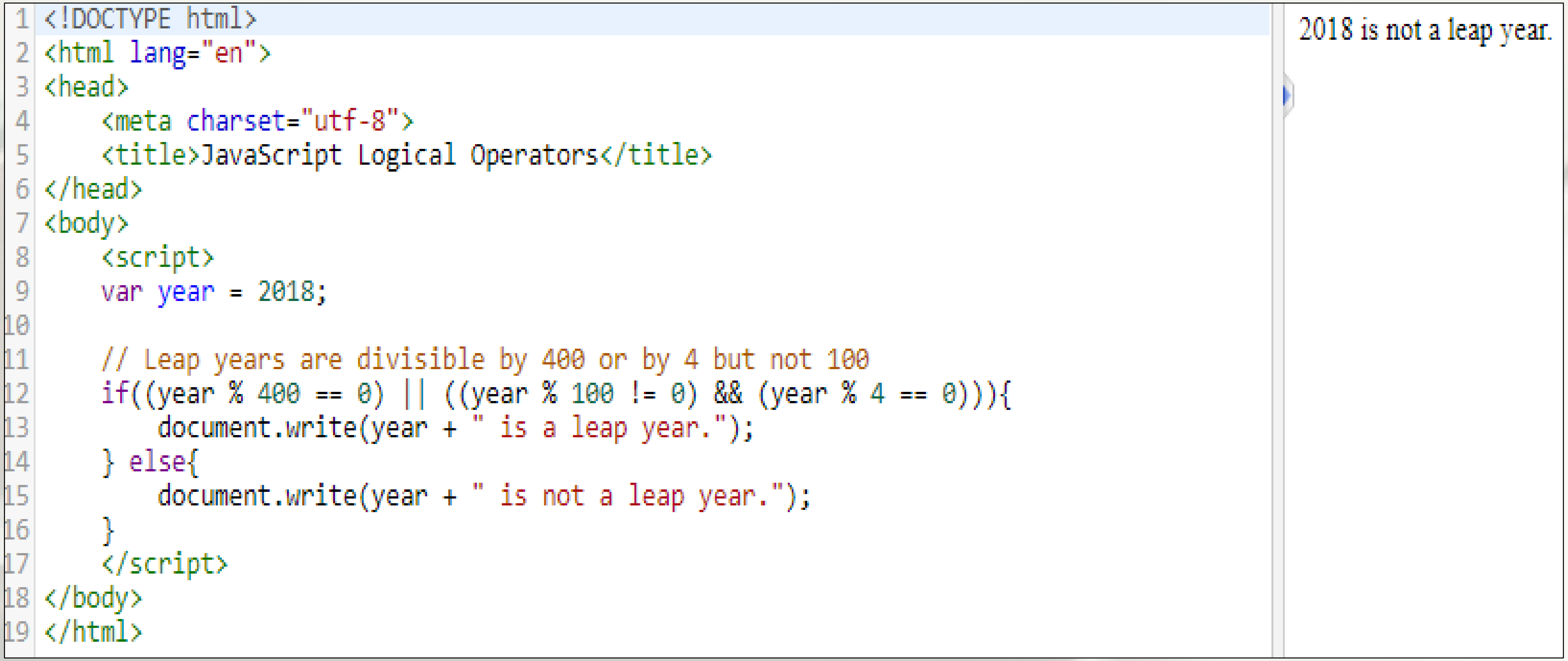
JS Logical Operators

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

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

JavaScript Operators

JS Logical Operators



```
<!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.
```

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

code4sp
coding for social promotion

JavaScript Operators

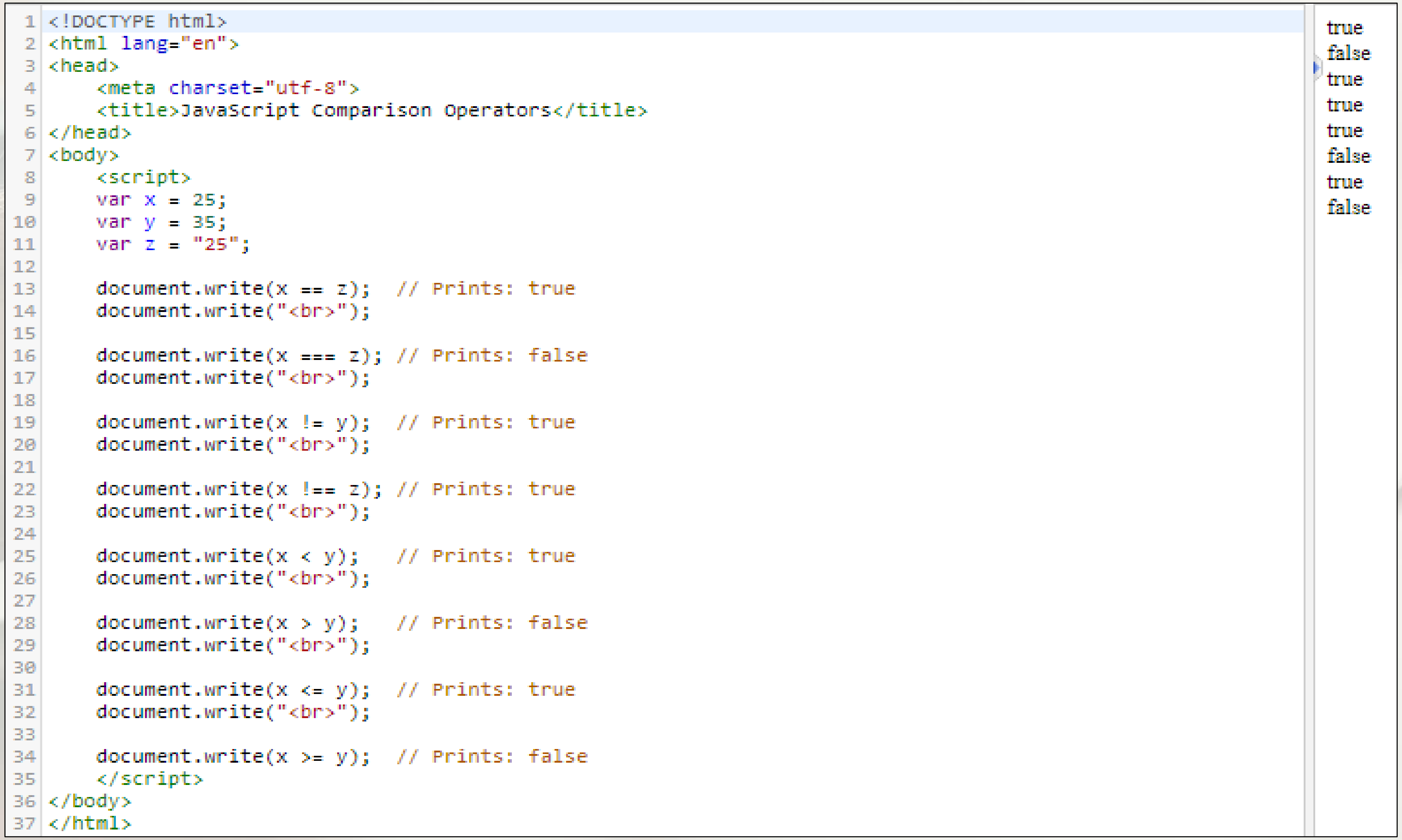
JS Comparison Operators

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

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

JavaScript Operators

JS Comparison Operators



```
<!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
```

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

JavaScript Events

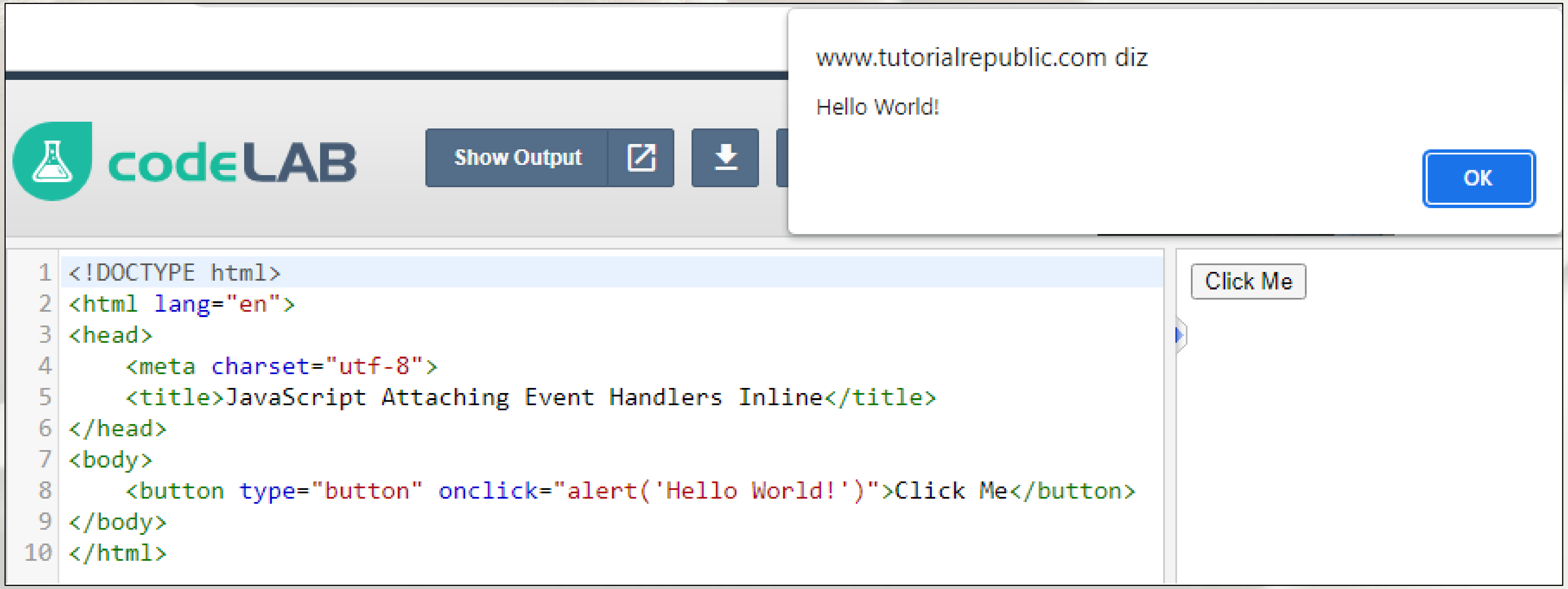
- Before going into deep on this section, it is important to acknowledge what an event is in this context. An event is something that occurs each time users interact with the web page, such as when a link or a button is clicked, text is entered into an input box or textarea, a selection is made in a 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.

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

JavaScript Events

- 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

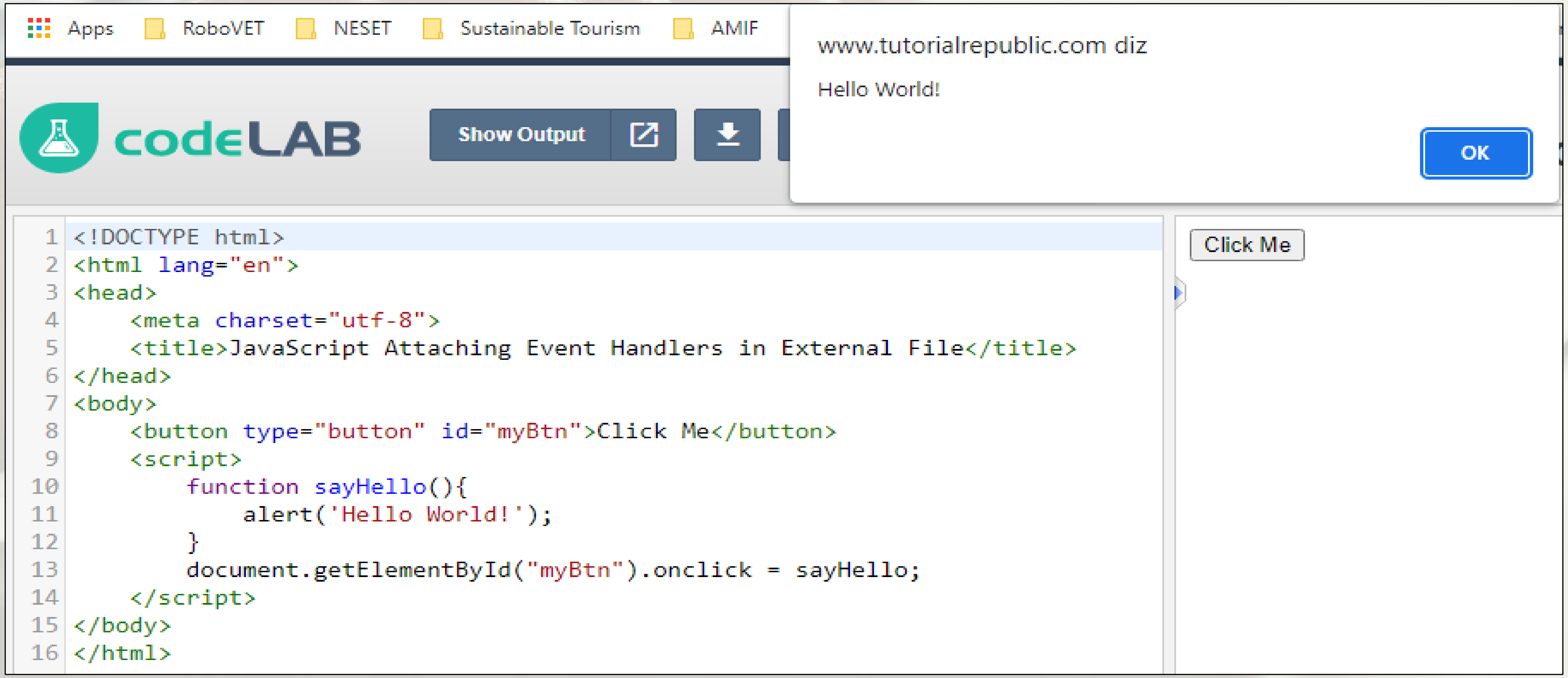
Hello World!

```
<!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>
```

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

JavaScript Events

- 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:

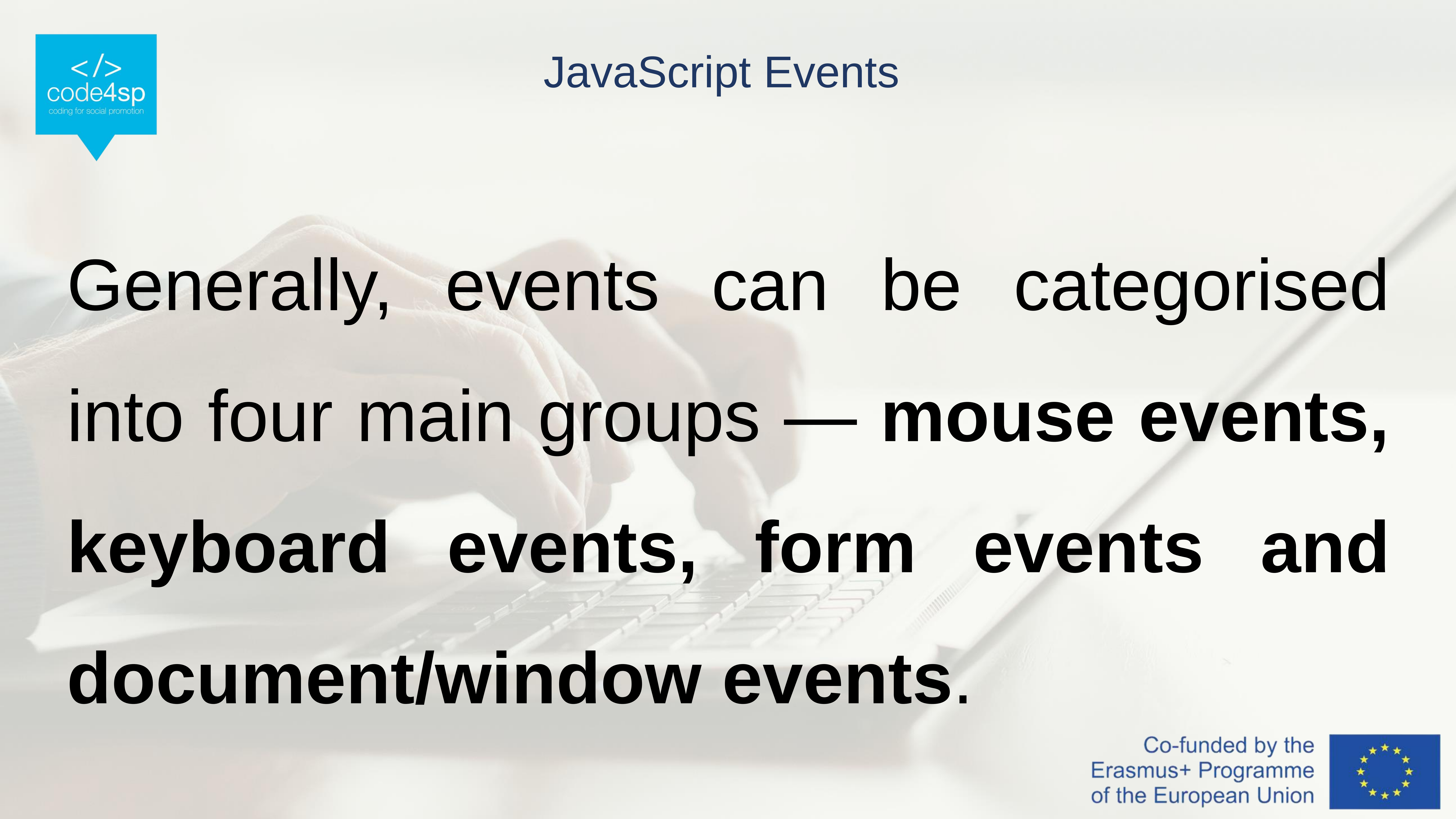


www.tutorialrepublic.com diz

Hello World!

```
<!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>
```

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



JavaScript Events

Generally, events can be categorised into four main groups — **mouse events, keyboard events, form events and document/window events**.

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

JavaScript Events

- **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 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.
   - **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 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.

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

JavaScript Events

- **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.
- **The Keyup Event (onkeyup):** it occurs when users release a key on the keyboard. Handled with onkeyup event handler.
- **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.

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

JavaScript Events

- **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.
- **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 Change Event (onchange):** it occurs as soon as a user modificates the value of a form element. Event handler: onchange.
- **The Submit Event (onsubmit):** it only happens when the user submits a form on a webpage. Event handler: onsubmit.

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

JavaScript Events

- **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.
- **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 Change Event (onchange):** it occurs as soon as a user modificates the value of a form element. Event handler: onchange.
- **The Submit Event (onsubmit):** it only happens when the user submits a form on a webpage. Event handler: onsubmit.

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

JavaScript Events

- **Document/Window Events**

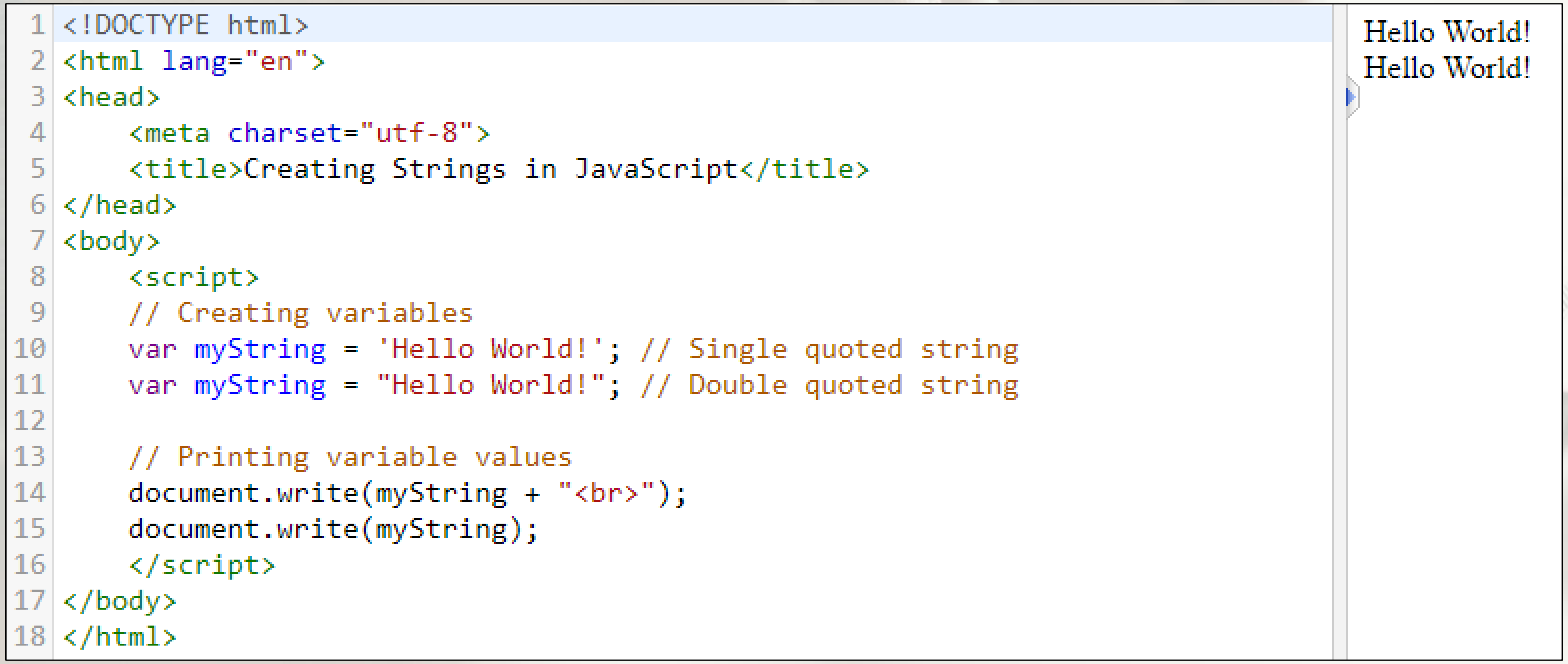
Situations in which the page has loaded or resized the browser window can as well trigger events.

- **The Load Event (onload):** it happens when a webpage fully loads in a web browser. Event handler: onload.
- **The Unload Event (onunload):** when the user leaves the present webpage, this event takes place. Event handler: onunload.
- **The Resize Event (onresize):** it happens when the Internet user resizes, minimises or maximises the browser window. Event handler: onresize.

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

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:



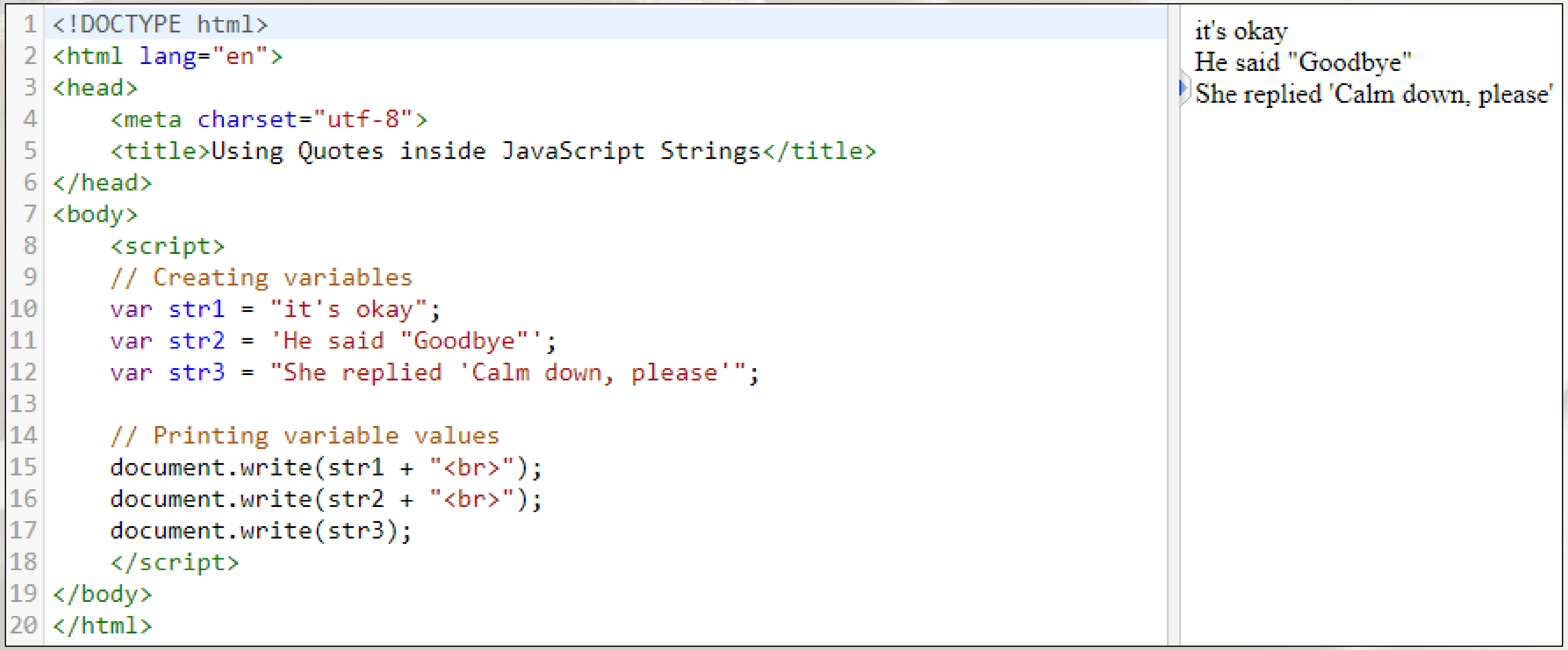
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Creating Strings in JavaScript</title>
</head>
<body>
    <script>
    // Creating variables
    var myString = 'Hello World!'; // Single quoted string
    var myString = "Hello World!"; // Double quoted string

    // Printing variable values
    document.write(myString + "<br>");
    document.write(myString);
    </script>
</body>
</html>
```

```
Hello World!
Hello World!
```

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

JavaScript Strings

Quotes can be used inside a string, but they should not match the quotes surrounding the string:



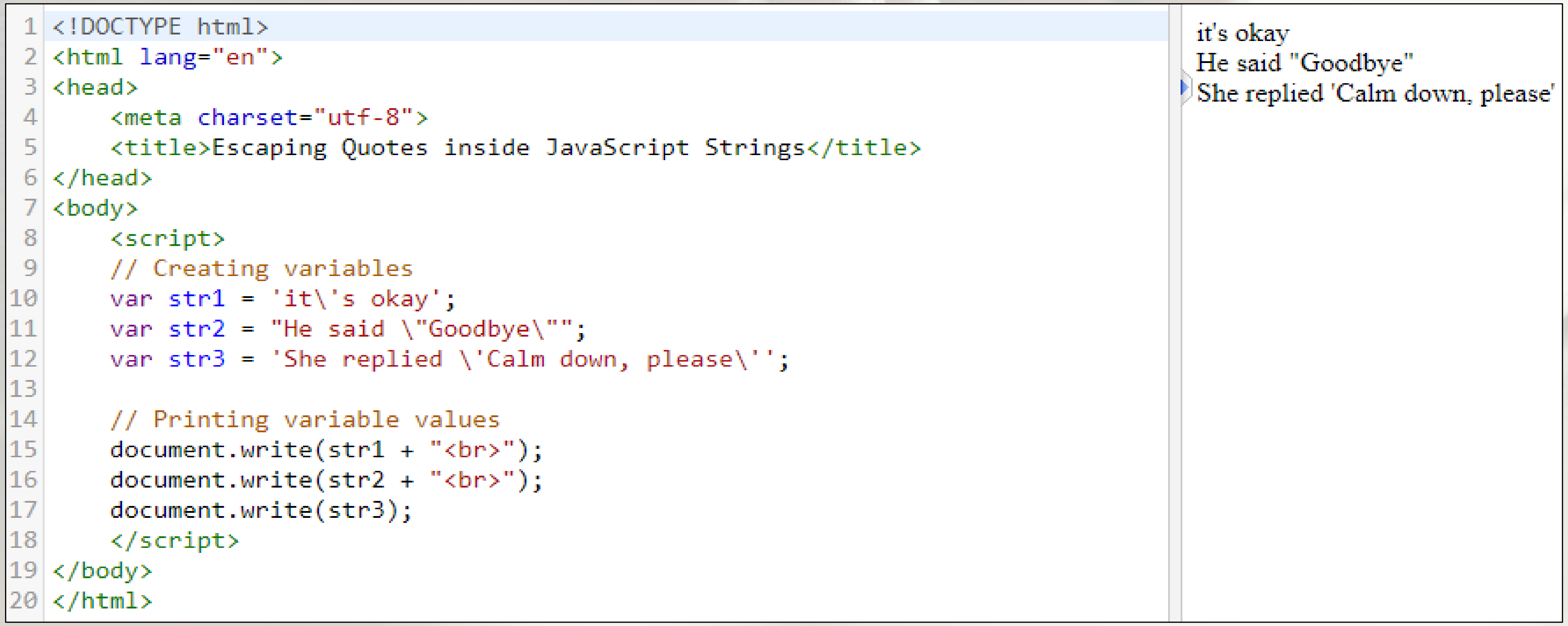
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Using Quotes inside JavaScript Strings</title>
</head>
<body>
    <script>
    // Creating variables
    var str1 = "it's okay";
    var str2 = 'He said "Goodbye"';
    var str3 = "She replied 'Calm down, please'";

    // 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'
```

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

JavaScript Strings

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 the image below 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'
```

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

JavaScript Strings

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 (\)

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

JavaScript Strings

Performing Operations on Strings

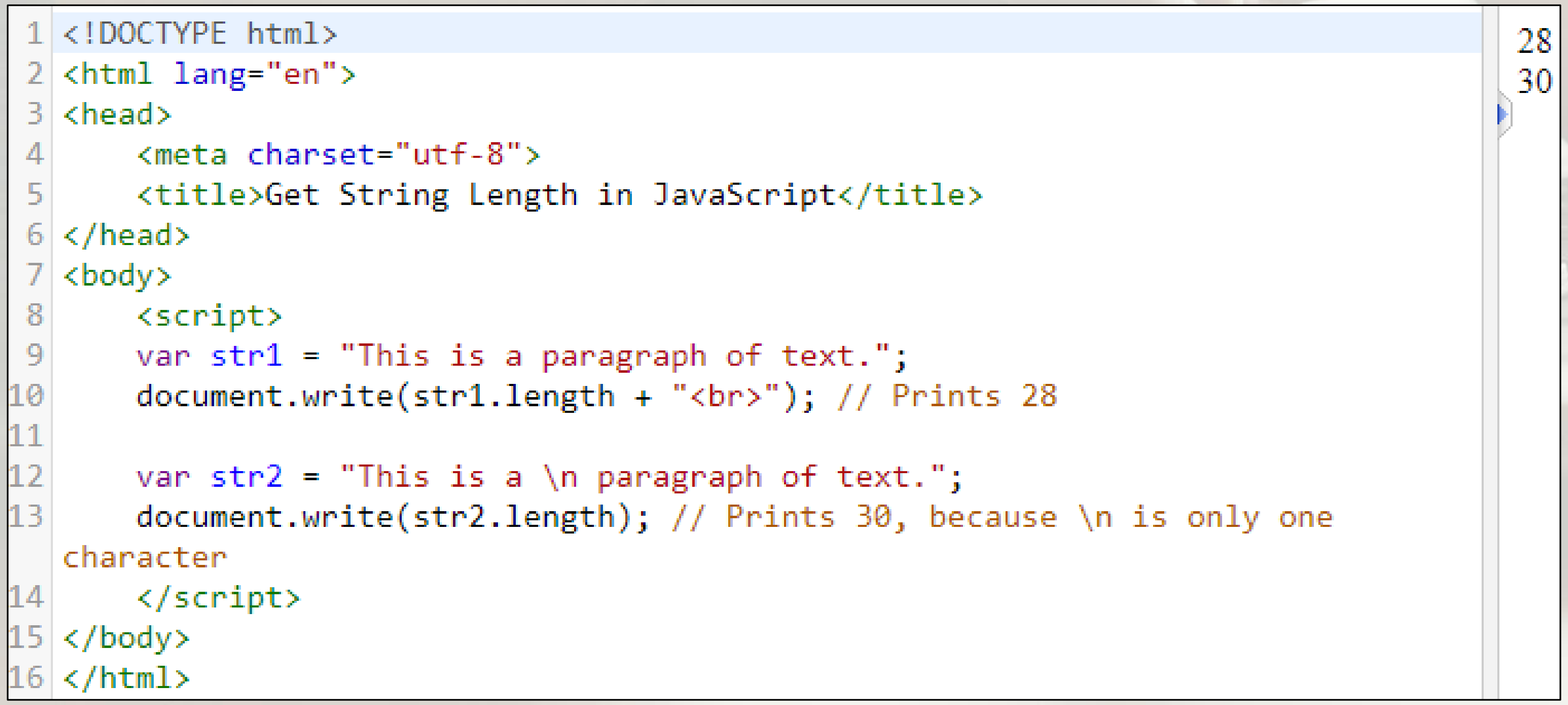
- JavaScript makes available several properties and methods to make operations on string values.
- Precisely, only objects can have properties and methods. However, in JavaScript, primitive data types can perform like objects when the programmer refers to them with the property access notation.
- JavaScript 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.

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

JavaScript Strings

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>
```

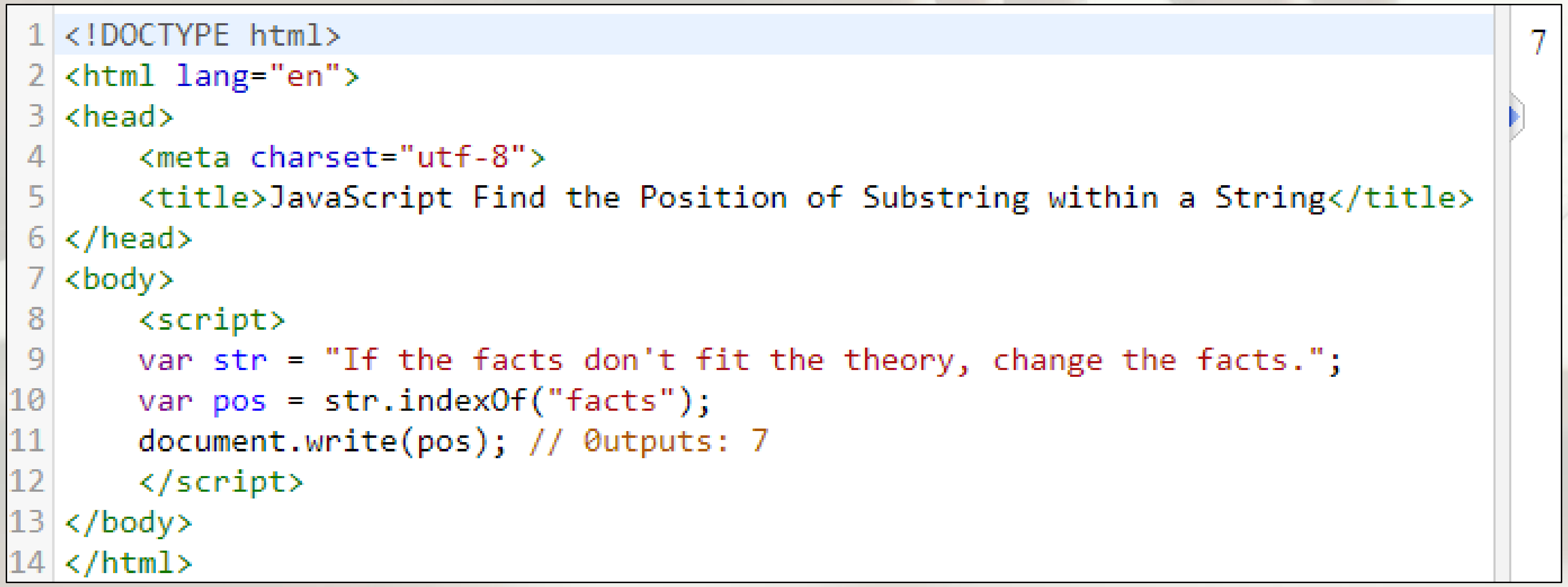
```
28
30
```

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

JavaScript Strings

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>
```

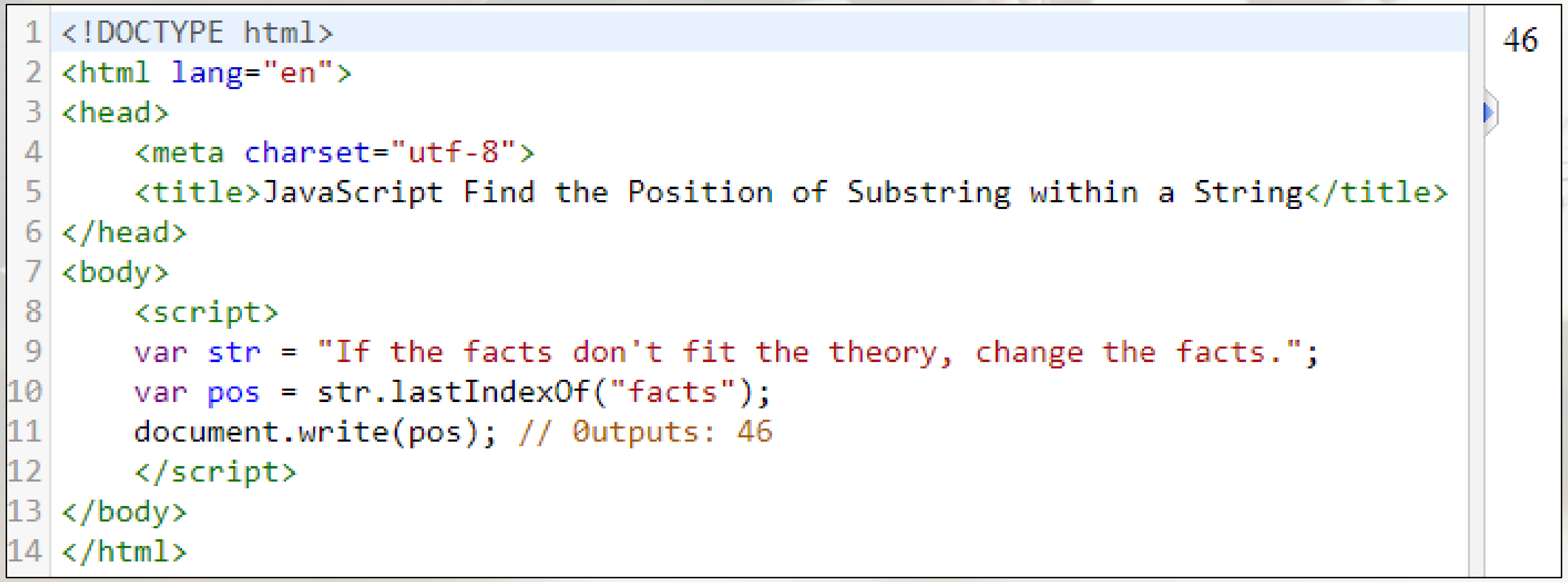
```
7
```

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

JavaScript Strings

Finding a String Inside Another String

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 on the image below. 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.



```
<!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
```

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

JavaScript Strings

Searching for a Pattern Inside a String

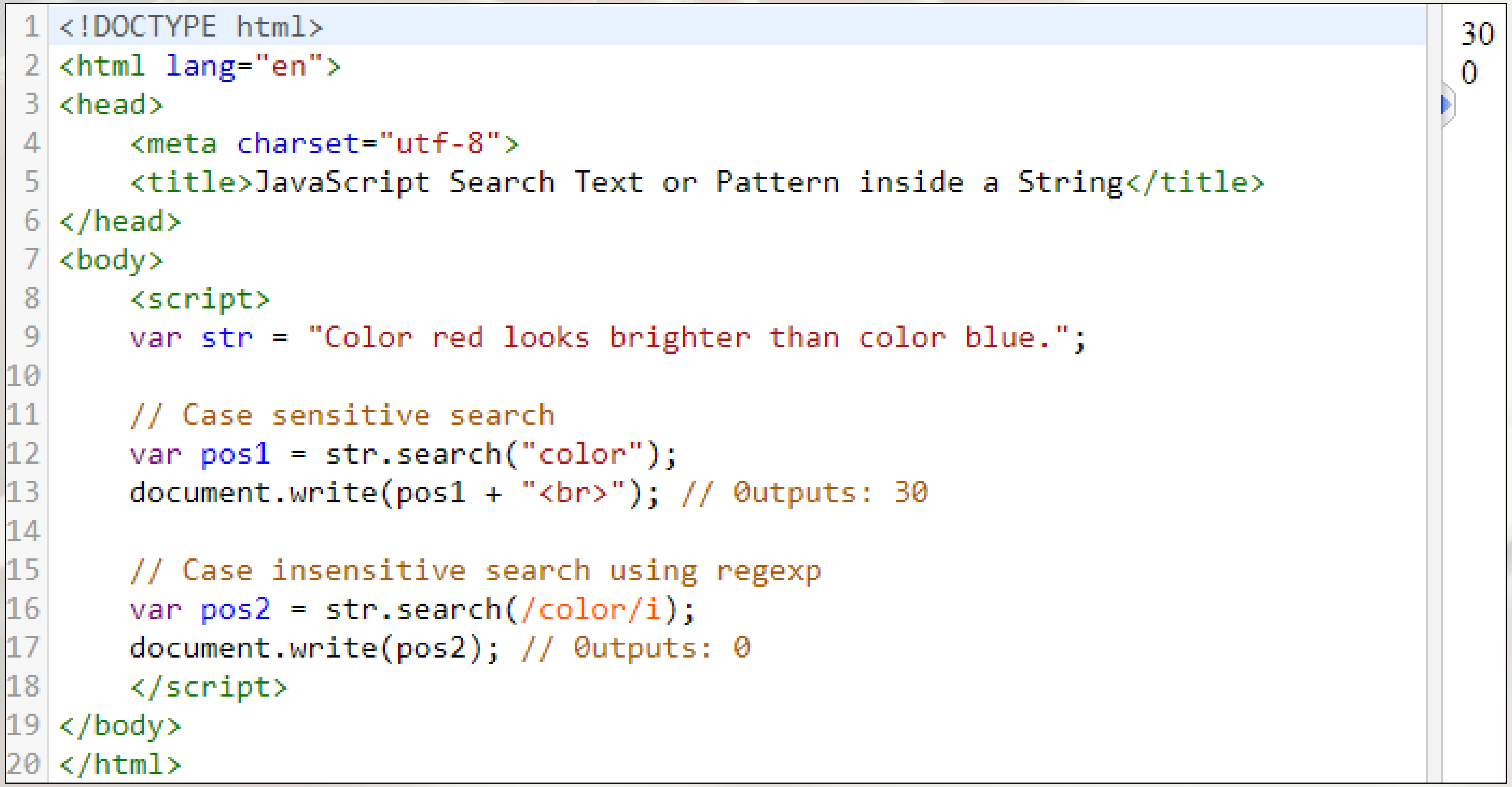
The search() method can be used to search a particular piece of text or pattern inside a string. As the indexOf() approach, search() also returns the index of the first match, and returns -1 if no matches were found, but unlike indexOf(), search () can as well take a *regular expression* as its argument 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.

.

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

JavaScript Strings

Searching for a Pattern Inside a String



```
<!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>
```

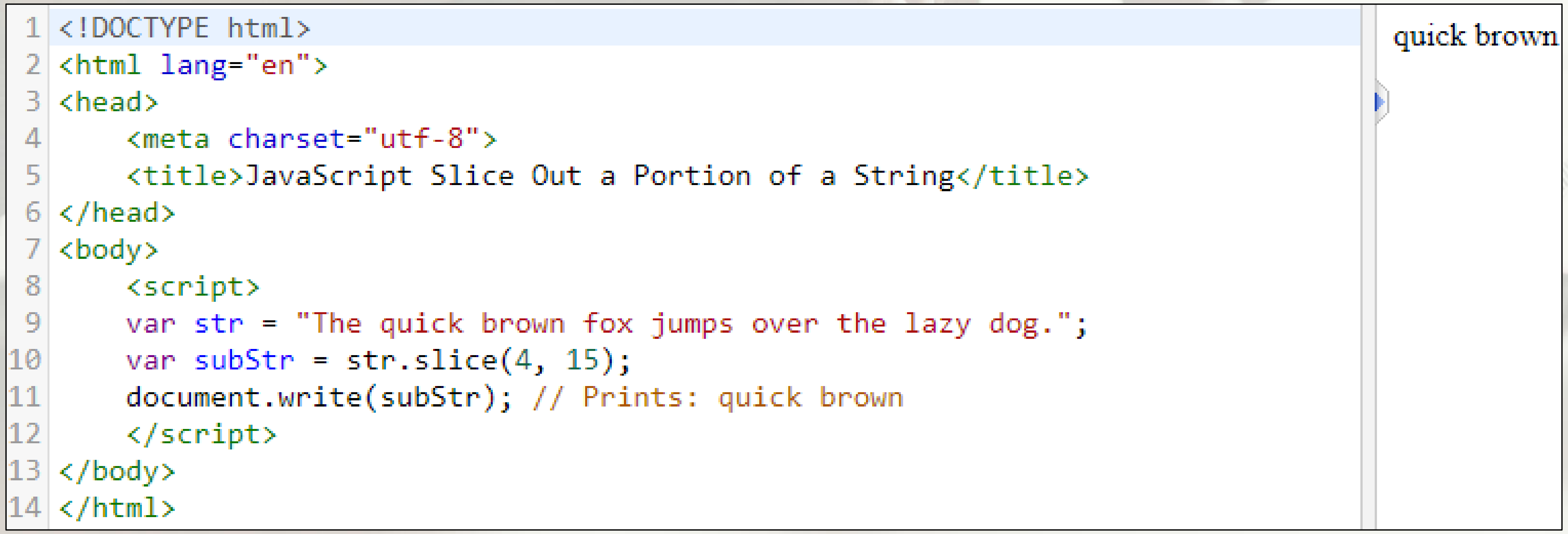
```
30
0
```

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

JavaScript Strings

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
```

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

JavaScript Strings

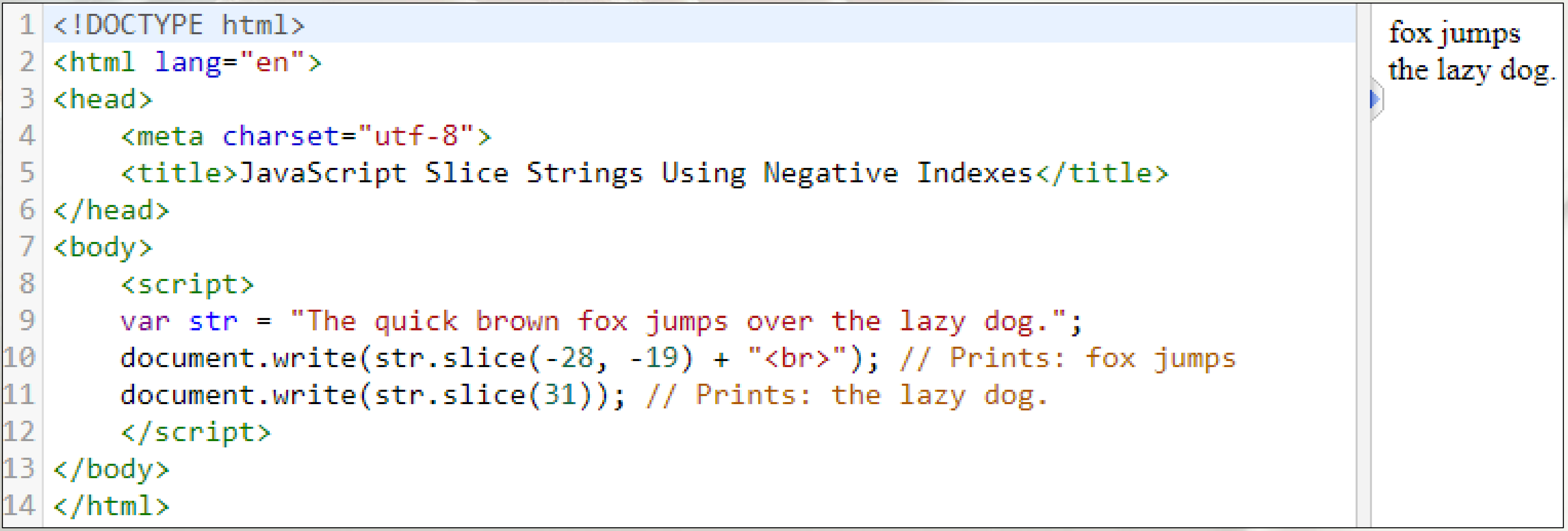
Extracting a Substring from a String

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.

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

JavaScript Strings

Extracting a Substring from a 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.
```

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

JavaScript Strings

Extracting a Substring from a String

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).

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

JavaScript Strings

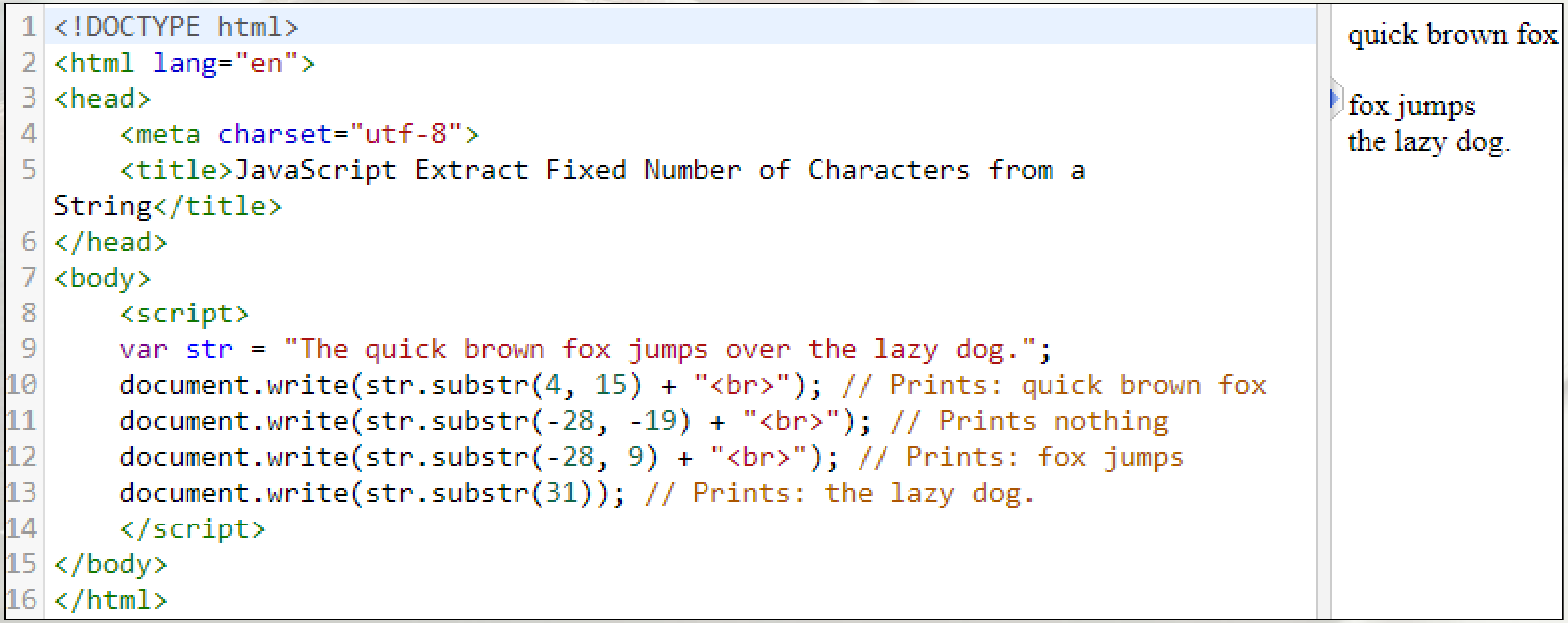
Extracting a Fixed Number of Characters from a String

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.

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

JavaScript Strings

Extracting a Fixed Number of Characters from a String



```html
<!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.
```

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

JavaScript Strings

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.

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

JavaScript Strings

Replacing the Contents of a String

- 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.

- 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.

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

JavaScript Strings

Converting a String to Uppercase or Lowercase

- The toUpperCase() method is used to convert a string to uppercase.
- Likewise, the toLowerCase() method is used to convert a string to lowercase.

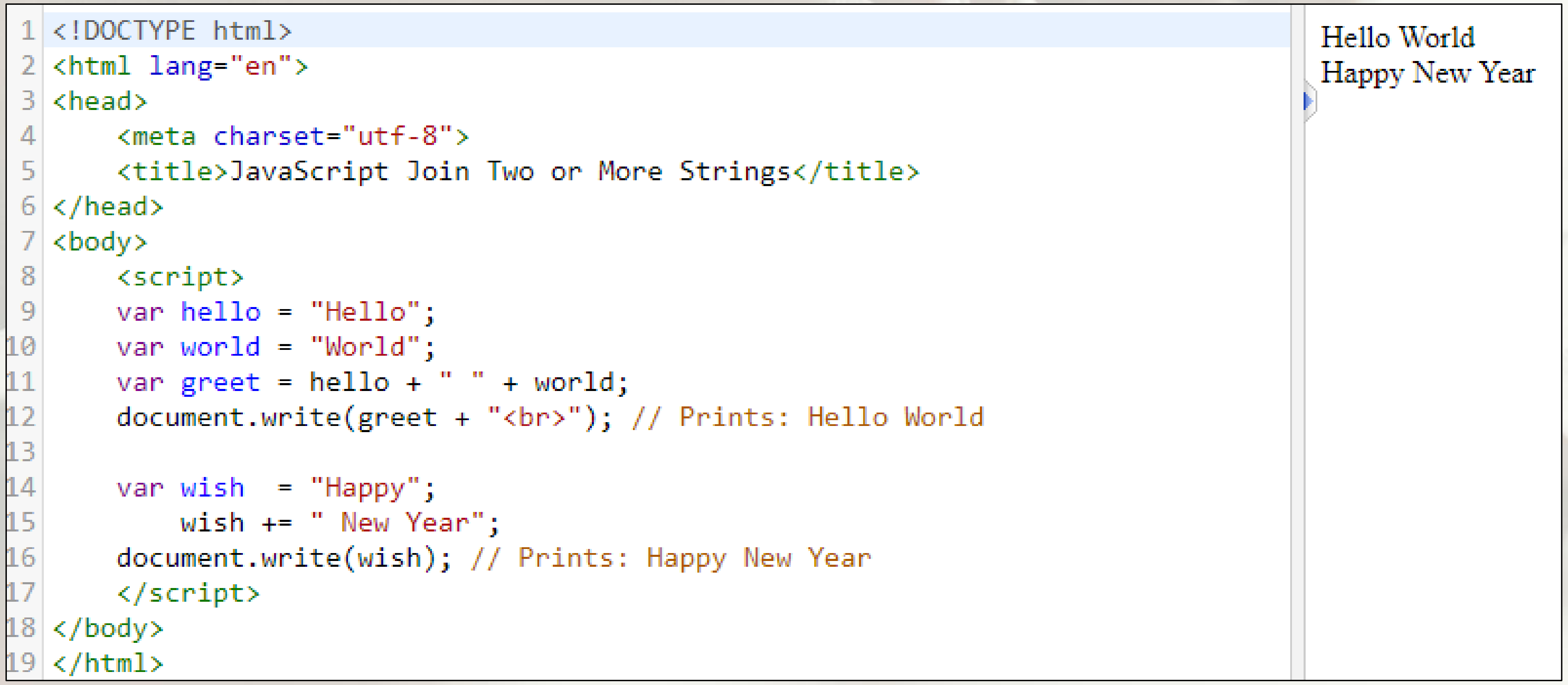
Co-funded by the Erasmus+ Programme of the European Union

code4sp
coding for social promotion

JavaScript Strings

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
```

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

JavaScript Strings

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.

- 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.

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

JavaScript Strings

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.

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

code4sp
coding for social promotion

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 used type of numbers.

- **BigInt numbers**, to represent integers of random length. They are sometimes needed, because a regular number cannot securely exceed $\mathbf{2^{53}}$ or be less than $\mathbf{-2^{53}}$.

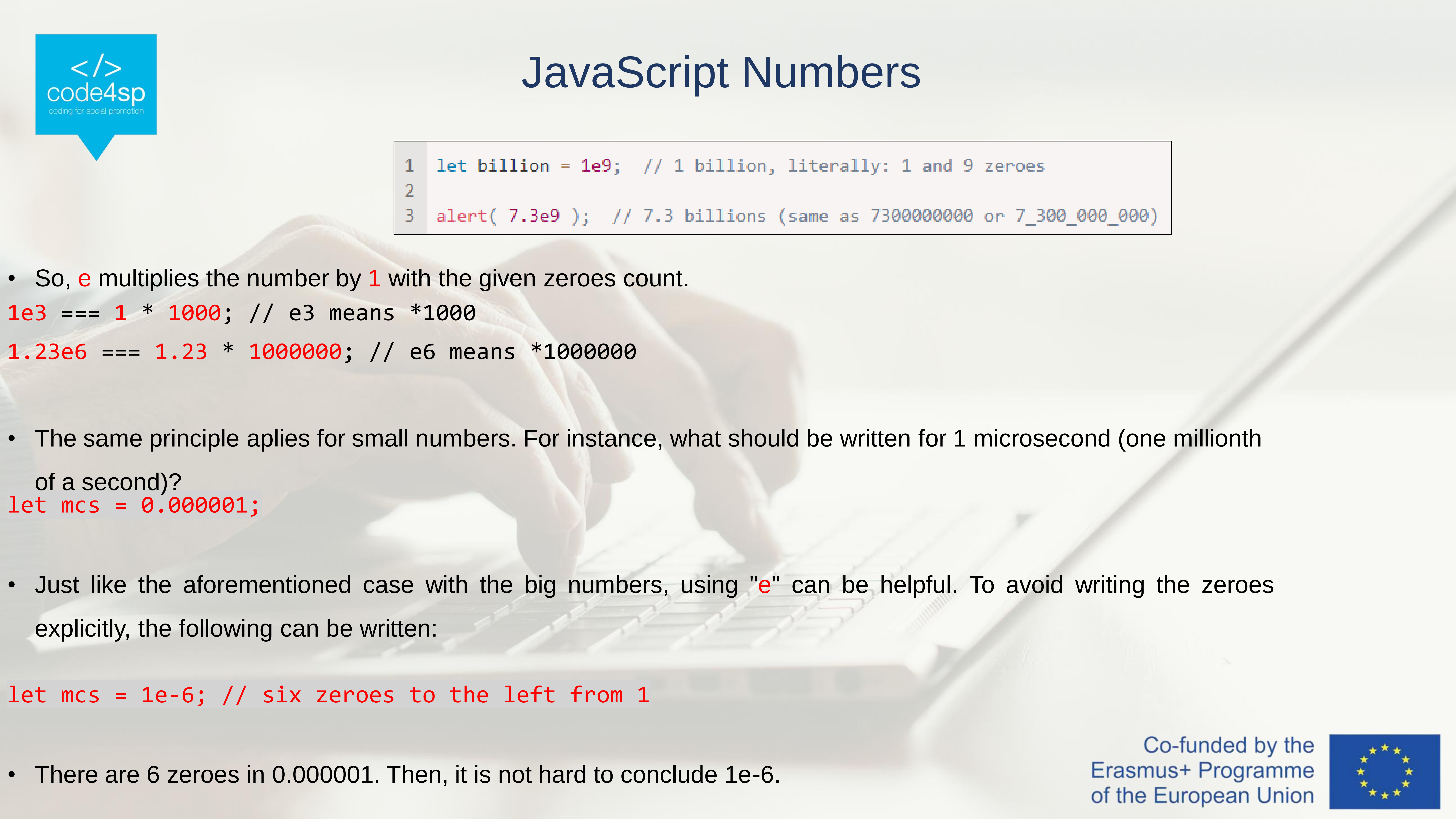
Co-funded by the Erasmus+ Programme of the European Union

JavaScript Numbers

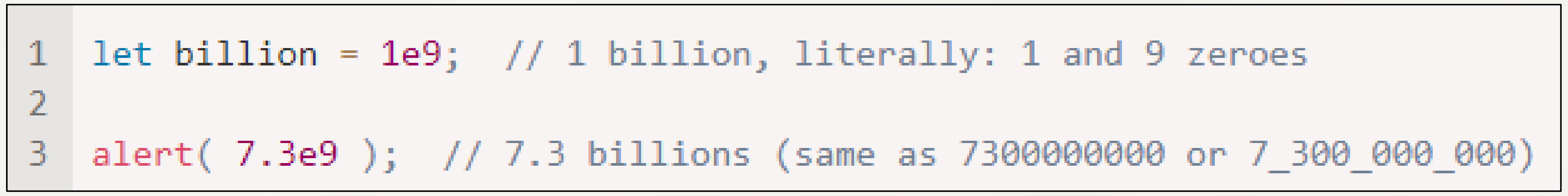
- There are more ways to write a number. For instance, the obvious way to write 1 billion would be ‘1000000000’ or ‘1_000_000_000’, using the underscore as a separator. In this case, the underscore is the “syntactic sugar”, meaning that it makes the number clearer to read. The JS engine ignores underscores 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. It is possible to shorten a number in JS, by adding the letter “e” to it and specifying the quantity of zeroes.

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



JavaScript Numbers



```
let billion = 1e9;  // 1 billion, literally: 1 and 9 zeroes

alert( 7.3e9 );  // 7.3 billions (same as 7300000000 or 7_300_000_000)
```

- 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.

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

JavaScript Numbers

Therefore, a negative number after "e" implies a division by 1 with the given number of zeroes, as follows:

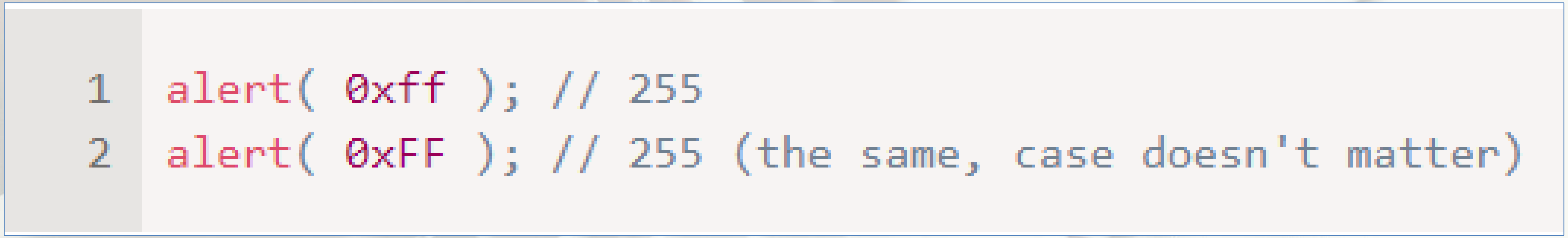
```
// -3 divides by 1 with 3 zeroes
1e-3 === 1 / 1000; // 0.001

// -6 divides by 1 with 6 zeroes
1.23e-6 === 1.23 / 1000000; // 0.00000123
```

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

JavaScript Numbers

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)
```

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

JavaScript Numbers

Hex, binary, and octal numbers

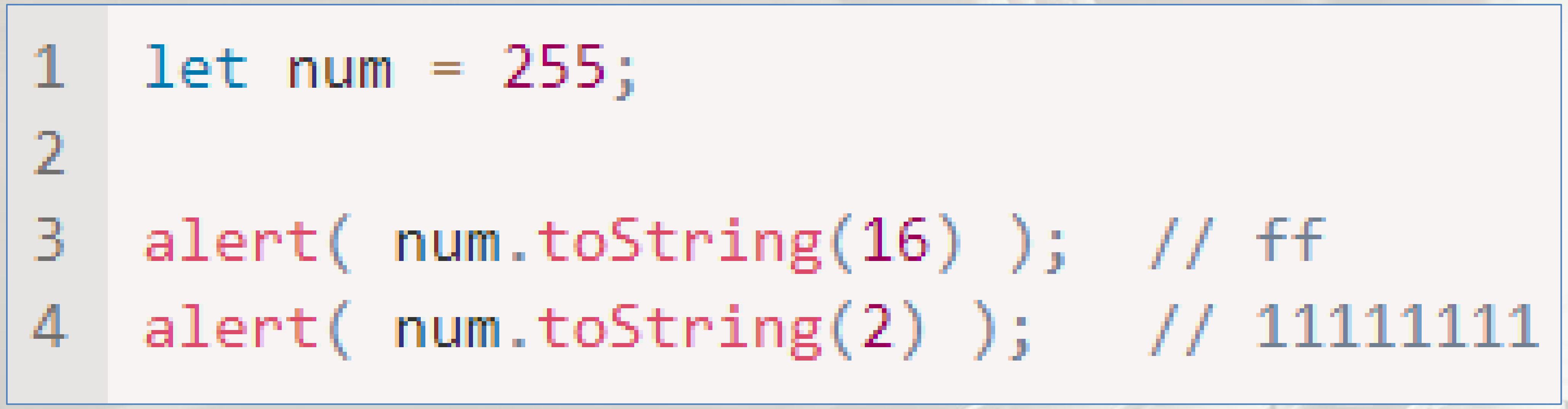
- Binary and octal numeral systems are not often utilized, but as well supported using the 0b and 0o (zero, o) prefixes.
- There are merely 3 numeral systems with this support. For further numeral systems, the function parseInt should be used.

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

JavaScript Numbers

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
```

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

JavaScript Numbers

toString(base) method

- 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.

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

code4sp
coding for social promotion

JavaScript Numbers

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.

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

JavaScript Numbers

Rounding

Functions from last slide 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.

2. The **toFixed(n)** method rounds the number to n digits after the point and returns a string representation of the result.

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

JavaScript Numbers

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.

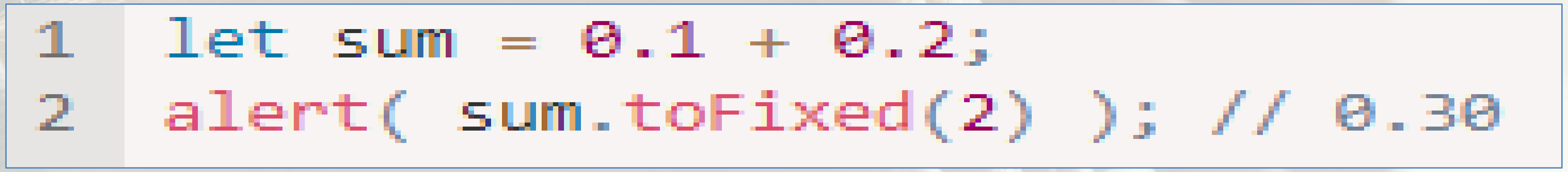
- Something that ends to happen quite often is the loss of precision.

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

JavaScript Numbers

Imprecise calculations

- The most reliable method to cope with this situation is **toFixed(n)**:



```
let sum = 0.1 + 0.2;
alert( sum.toFixed(2) ); // 0.30
```

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

JavaScript Numbers

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.

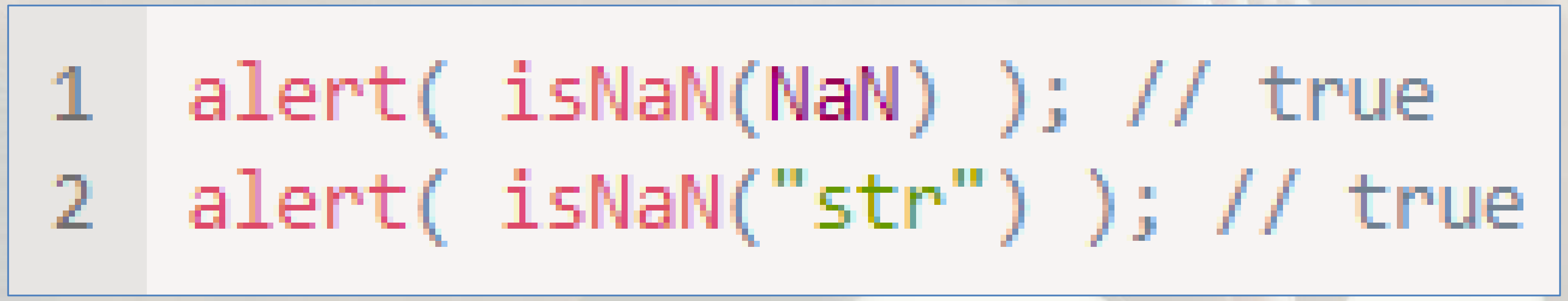
Co-funded by the Erasmus+ Programme of the European Union

JavaScript Numbers

Tests: isFinite and isNaN

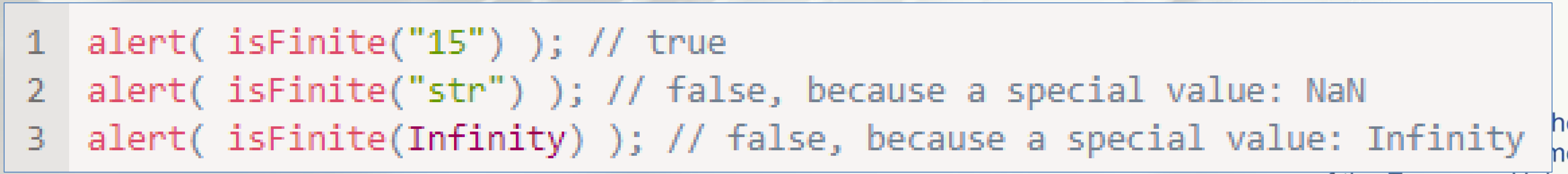
Both belong to the type **number**, but are not “normal” numbers, so there are special functions to check for them:

- **isNaN(value)** converts its argument to a number and then tests it for being **NaN**:



```
alert( isNaN(NaN) ); // true
alert( isNaN("str") ); // true
```

- **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
```

JavaScript Numbers

parseInt and parseFloat

- Numeric conversion using a plus **+** or **Number()** is rigorous. If a value is not exactly a number, it fails.
- **parseInt** and **parseFloat** are meant to extract numeric values out of these unexpected situations (e.g., when symbols come after the number – 18€).
- 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.

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

JavaScript Numbers

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).

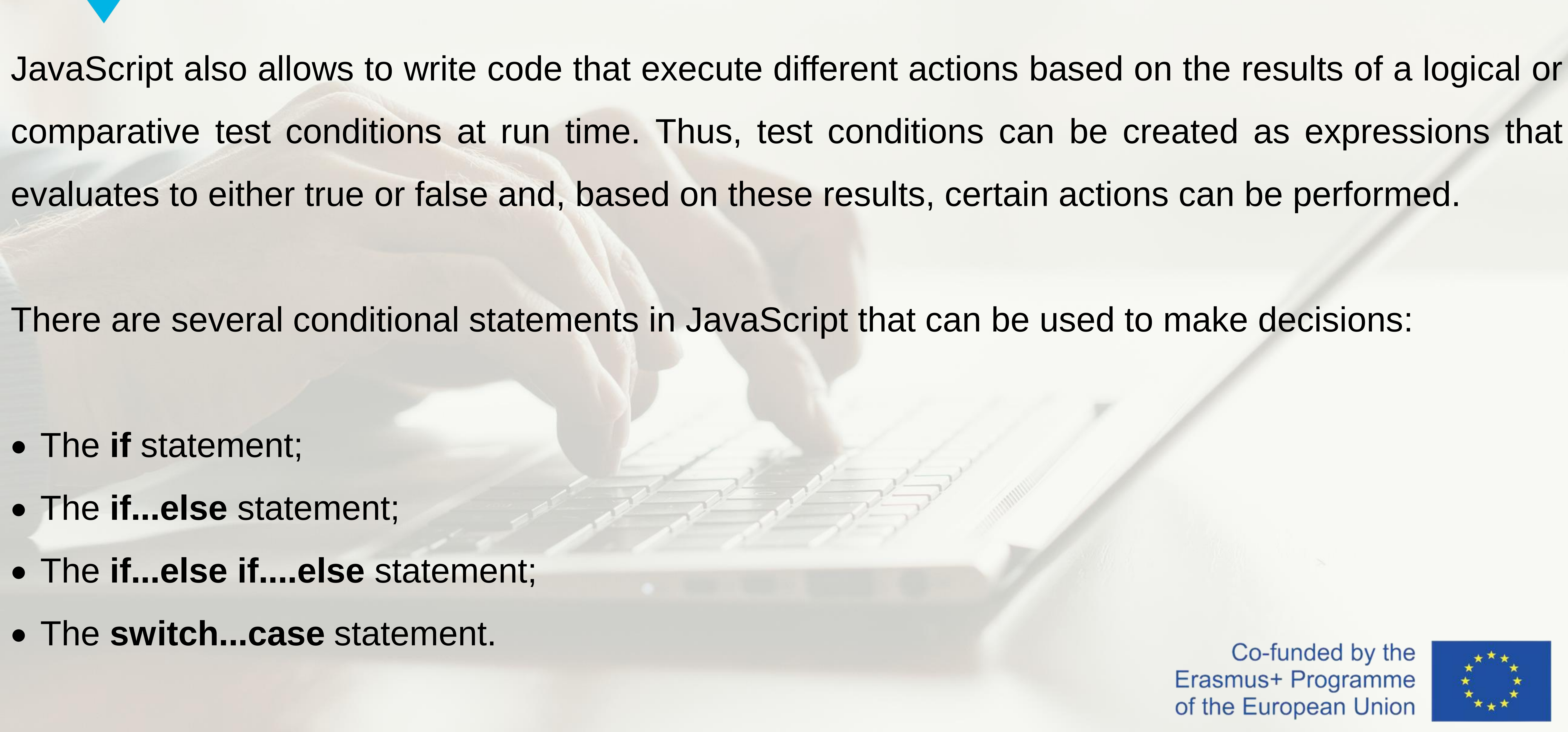
- **Math.max(a, b, c...) / Math.min(a, b, c...)**

  Returns the greatest/smallest from the arbitrary number of arguments.

- **Math.pow(n, power)**

  Returns n raised to the given power.

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



JavaScript If…Else Statements

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:

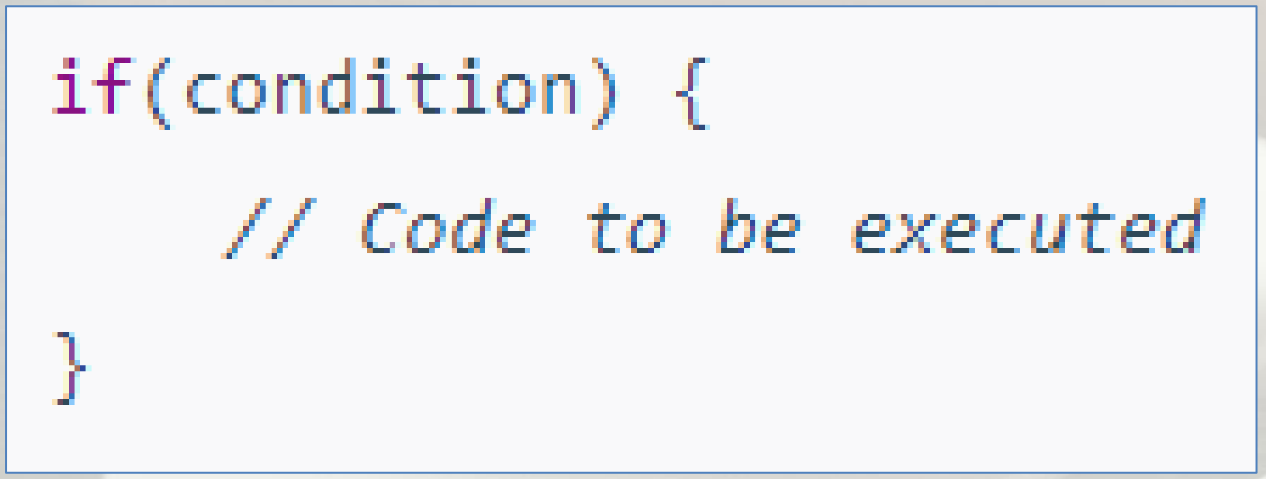
- The **if** statement;
- The **if...else** statement;
- The **if...else if....else** statement;
- The **switch...case** statement.

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

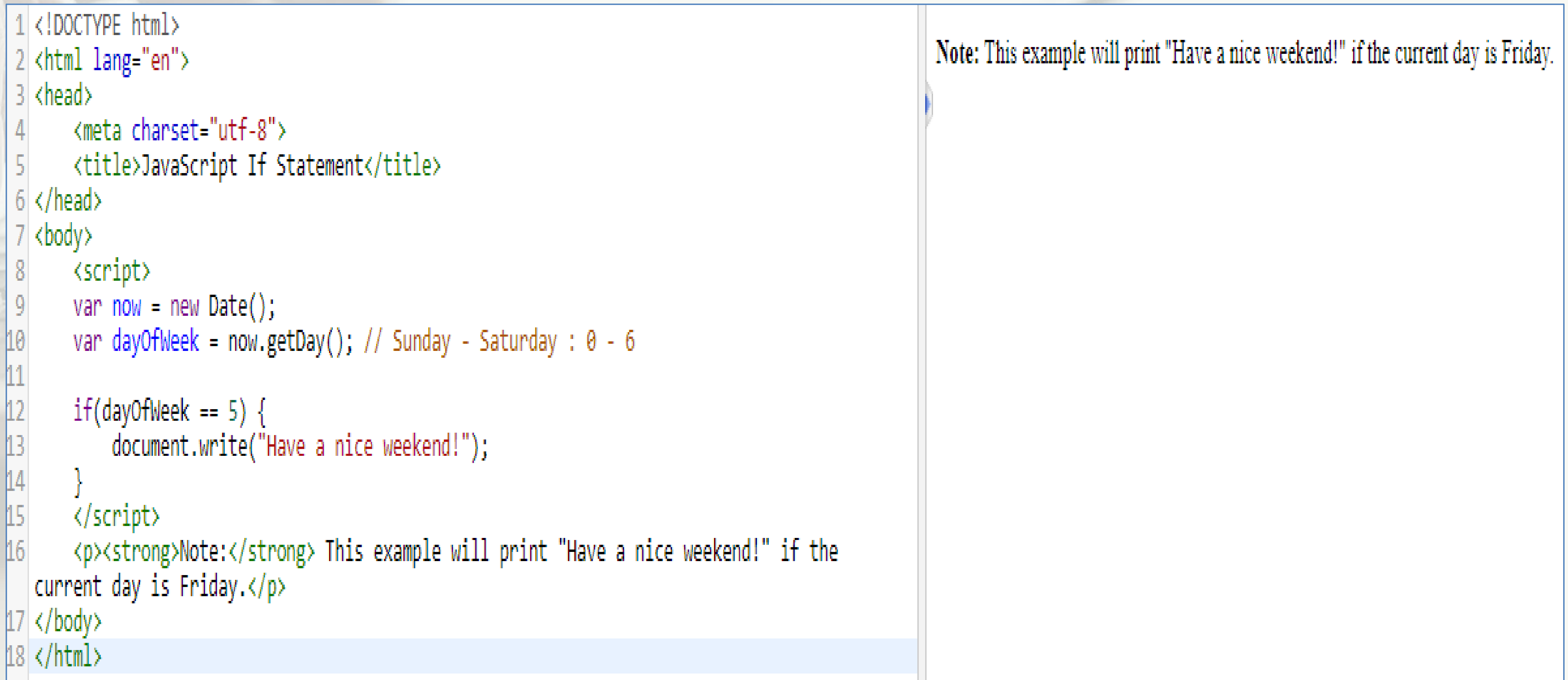
JavaScript If…Else Statements

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:



```
if(condition) {
    // Code to be executed
}
```



```
<!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.

JavaScript If…Else Statements

The if…else statement

The decision-making process of JS can be enhanced by providing an alternative 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
}
```

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

JavaScript If…Else Statements

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.

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

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
}
```

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

JavaScript Switch…Case Statements

- This example displays the name of the day of the week the reader is in.

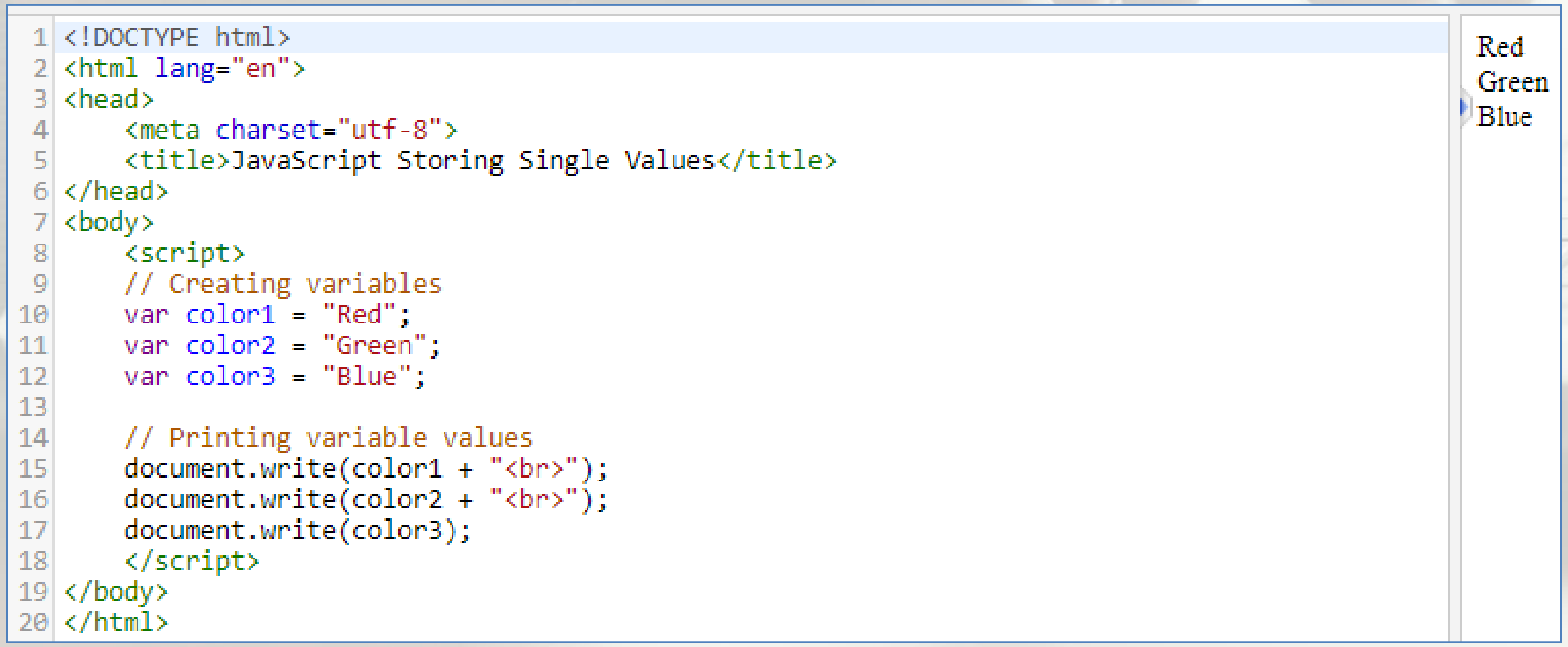
- The **switch...case** statement diverges from the **if...else** statement in one crucial way. The switch statement executes line by line and when JavaScript finds a case clause that evaluates to *true*, it does not only perform the code corresponding to that case clause, but also executes all the successive case clauses till the end of the switch block automatically.

- To prevent this, a break statement must be included after each case. 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.

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

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
```

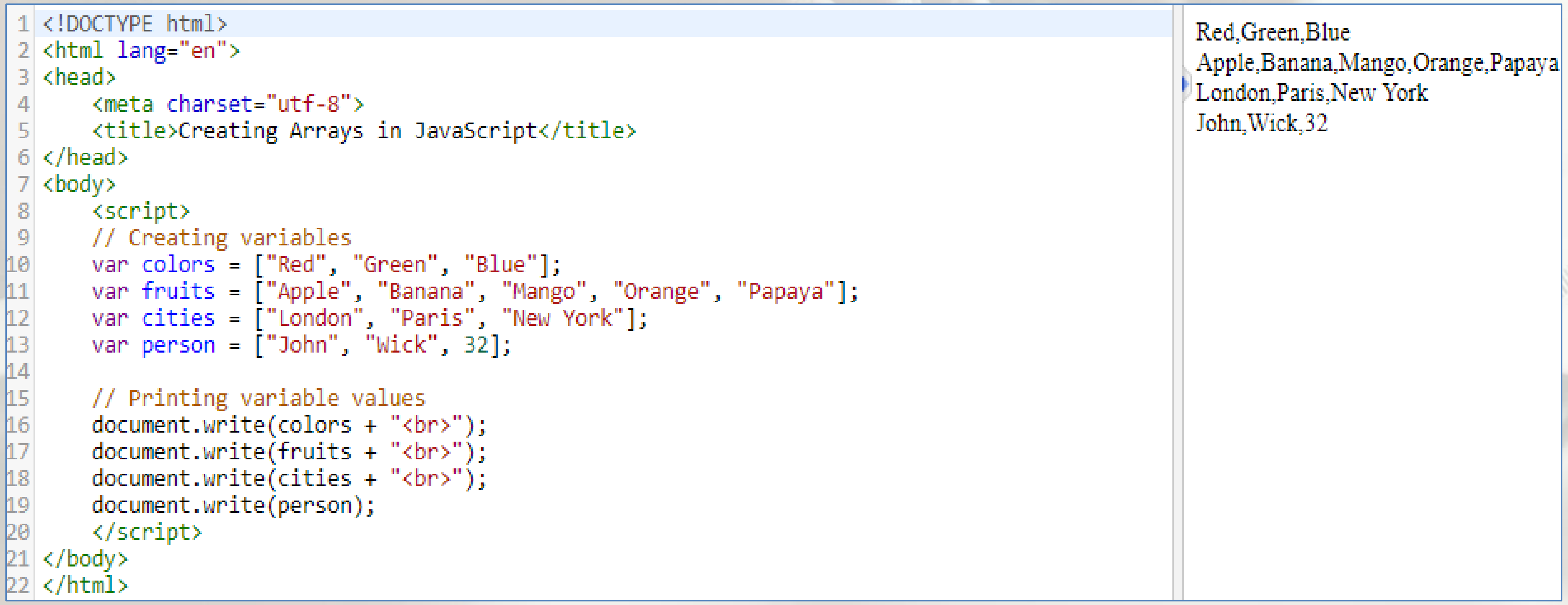
Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Arrays

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];
```



```
<!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>
```

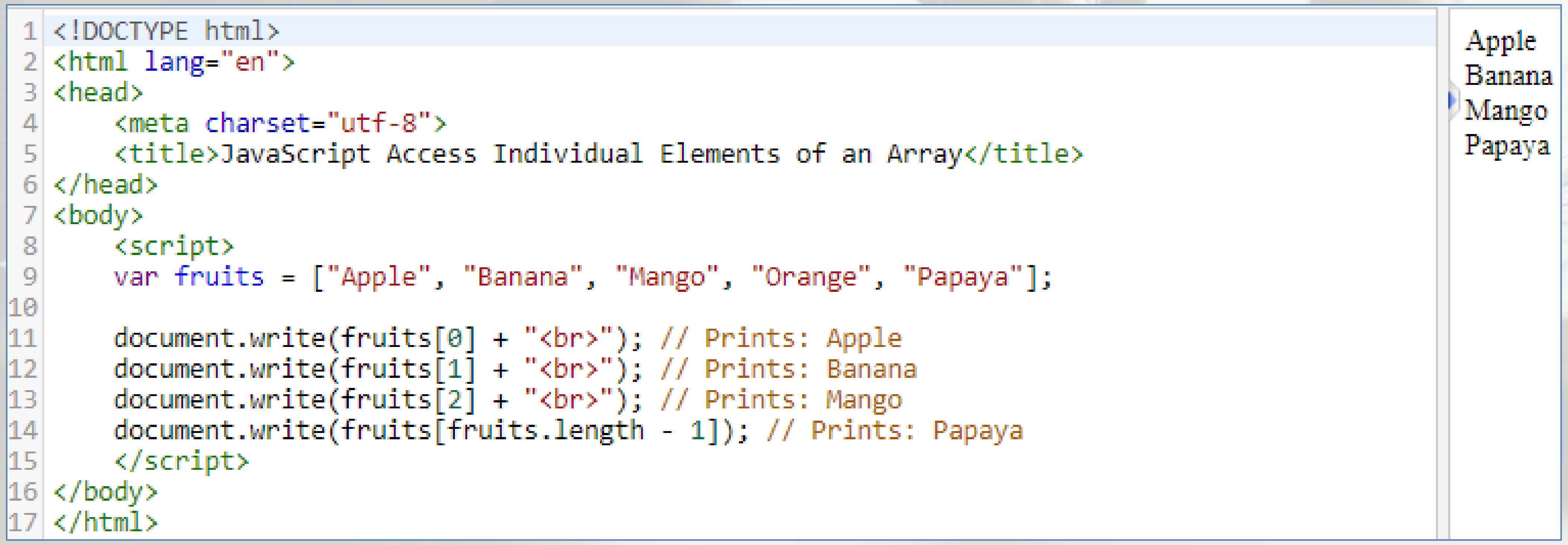
```
Red,Green,Blue
Apple,Banana,Mango,Orange,Papaya
London,Paris,New York
John,Wick,32
```

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

JavaScript Arrays

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
```

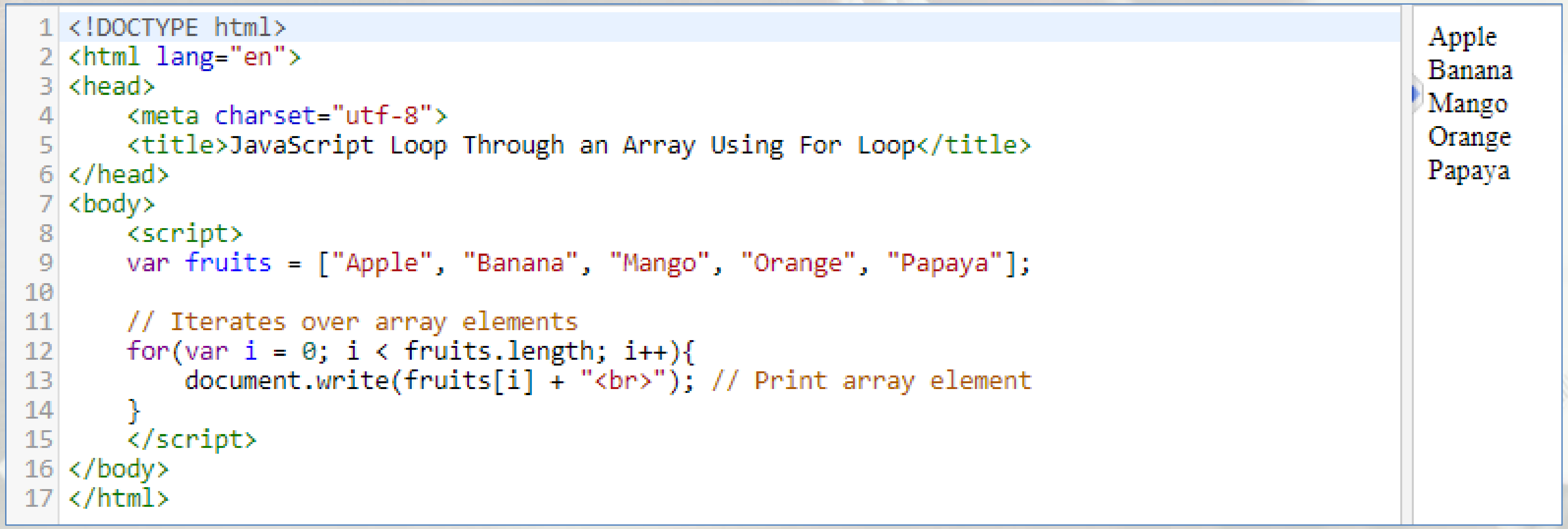
Co-funded by the Erasmus+ Programme of the European Union

code4sp
coding for social promotion

JavaScript Arrays

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>
```

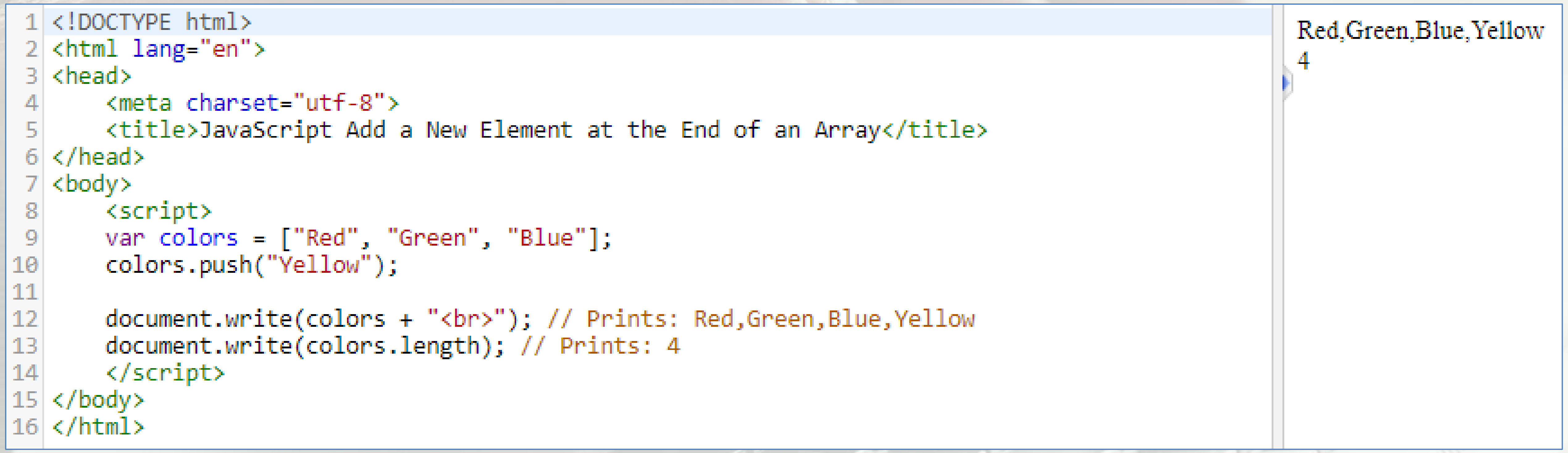
```
Apple
Banana
Mango
Orange
Papaya
```

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

JavaScript Arrays

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>
```

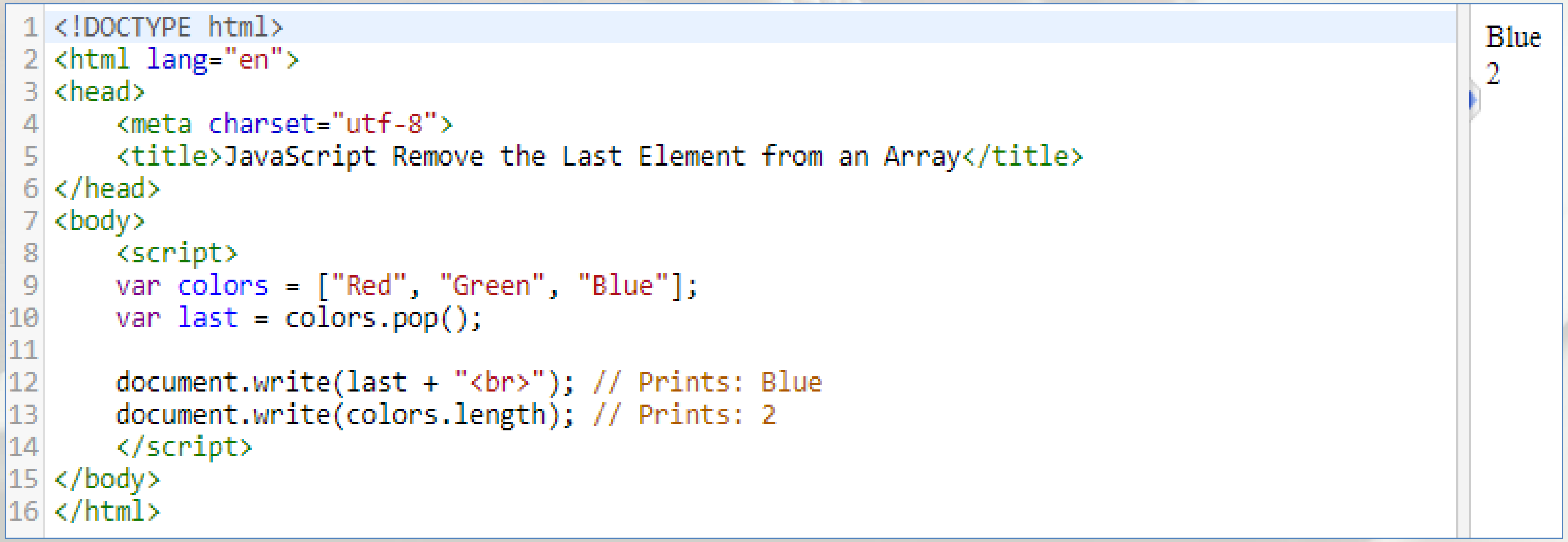
```
Red,Green,Blue,Yellow
4
```

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

JavaScript Arrays

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.



```
<!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
```

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

JavaScript Arrays

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.

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

JavaScript Arrays

Adding or Removing Elements at any Position

- 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.

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

JavaScript Arrays

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.

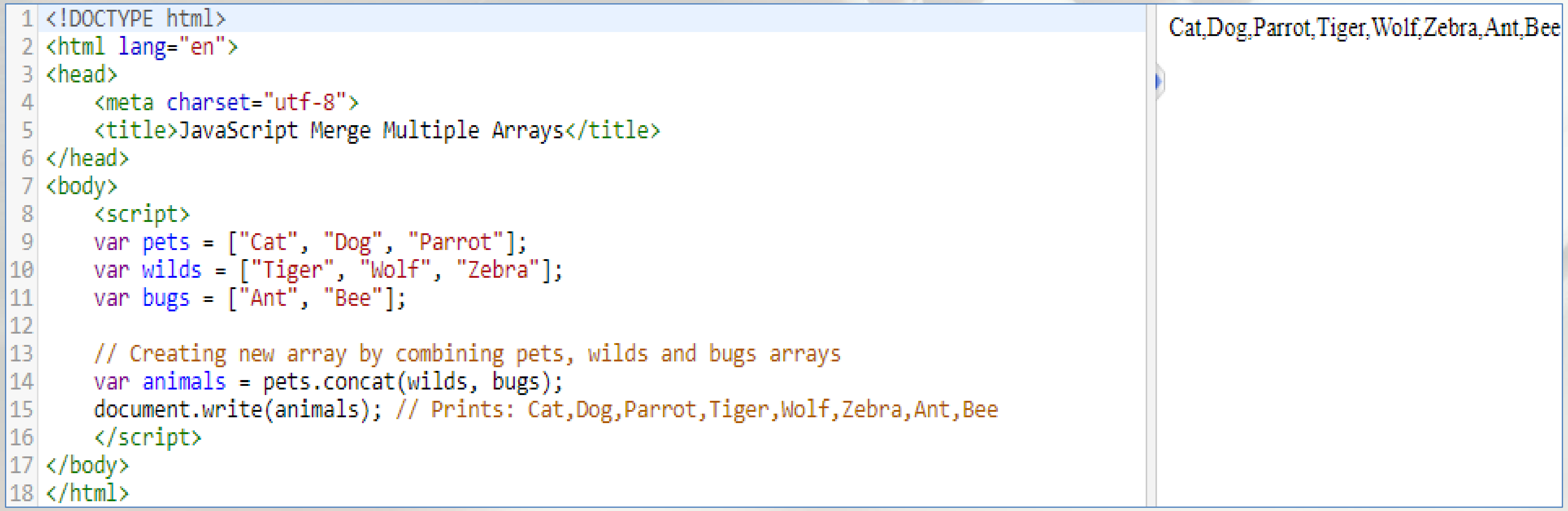
- An array can also be converted to a comma-separated string using the toString(). This method does not allow the separator parameter as join().

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

JavaScript Arrays

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.
- 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
```

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

JavaScript Arrays

Searching Through an Array

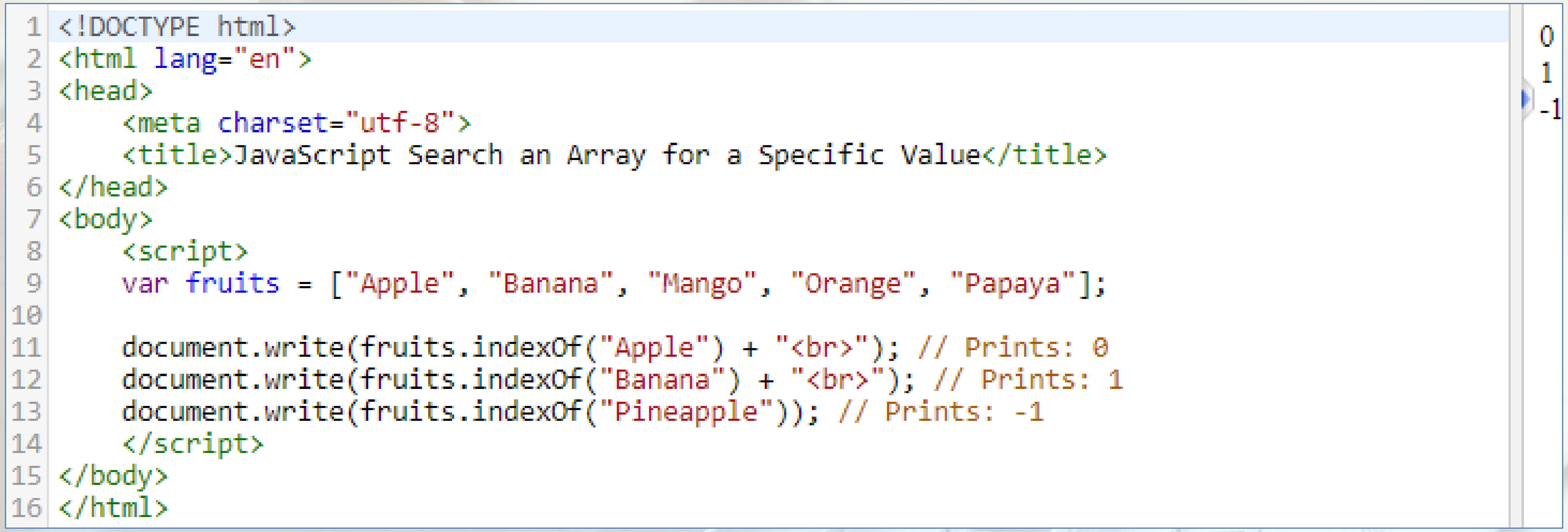
- The indexOf() and lastIndexOf() methods can be used for searching an array for a specific value. If the value is found, both methods return an index representing the array element. If the value is not found, -1 is returned.

- 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.

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

JavaScript Arrays

Searching Through an Array



```
<!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>
```

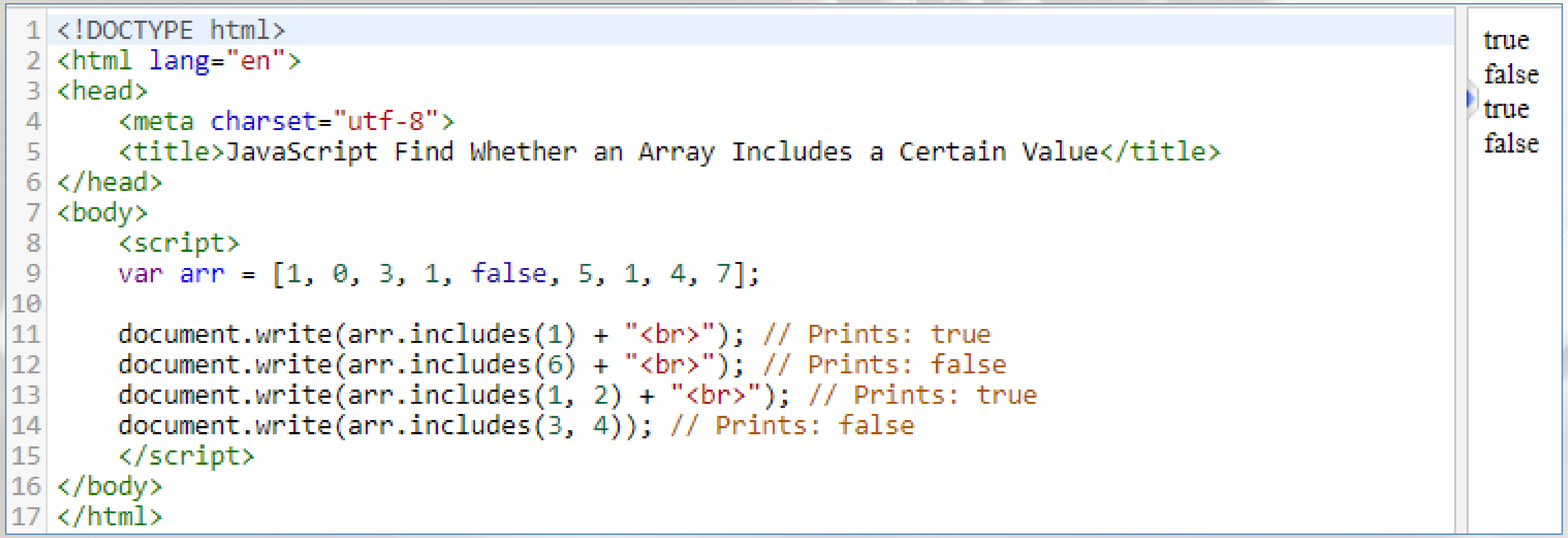
```
0
1
-1
```

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

JavaScript Arrays

Searching Through an Array

- 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
```

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

JavaScript Arrays

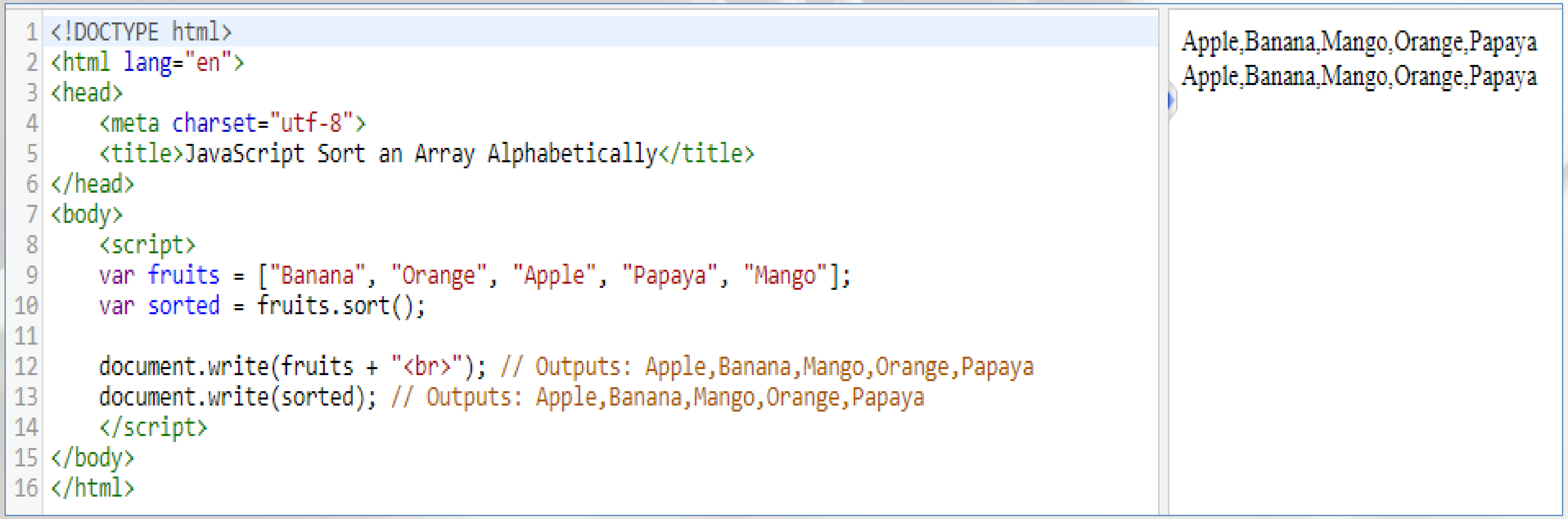
Searching Through an Array

- 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.

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

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 i 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
```

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

JavaScript Sorting Arrays

- 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 function, as when a compare function is specified, array elements are sorted according to the return value of the compare function.

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

JavaScript Sorting Arrays

- 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.

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

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*.

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.

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

JavaScript Loops

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).

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

JavaScript Loops

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);
```

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

JavaScript Loops

The do…while loop

- 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*.

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

JavaScript Loops

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
}
```

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

JavaScript Loops

The for loop

- 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.

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

JavaScript Loops

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.

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

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.

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

JavaScript Functions

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.

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

JavaScript Functions

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
}
```

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

JavaScript Functions

Defining and Calling a Function

- 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 example given previously.

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

JavaScript Functions

Adding Parameters to Functions

- 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.

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

JavaScript Functions

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.

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

JavaScript Functions

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.

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

JavaScript Functions

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.

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

JavaScript Functions

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.

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

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.).

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

code4sp
coding for social promotion

JavaScript Objects

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().

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

JavaScript Objects

Creating an object

- 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.

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

JavaScript Objects

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 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.

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

JavaScript Objects

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.

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

JavaScript Objects

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.

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

JavaScript Objects

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.

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

JavaScript Objects

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.

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

JavaScript Objects

Manipulating by Value vs. Reference

- As it could be observed, a copy of a variable message has 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.

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

LET’S PRACTICE!

Open the following link in order to practice some of the concepts acquired so far:

https://www.w3resource.com/javascript-exercises/javascript-basic-exercises.php

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

THANK YOU!

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