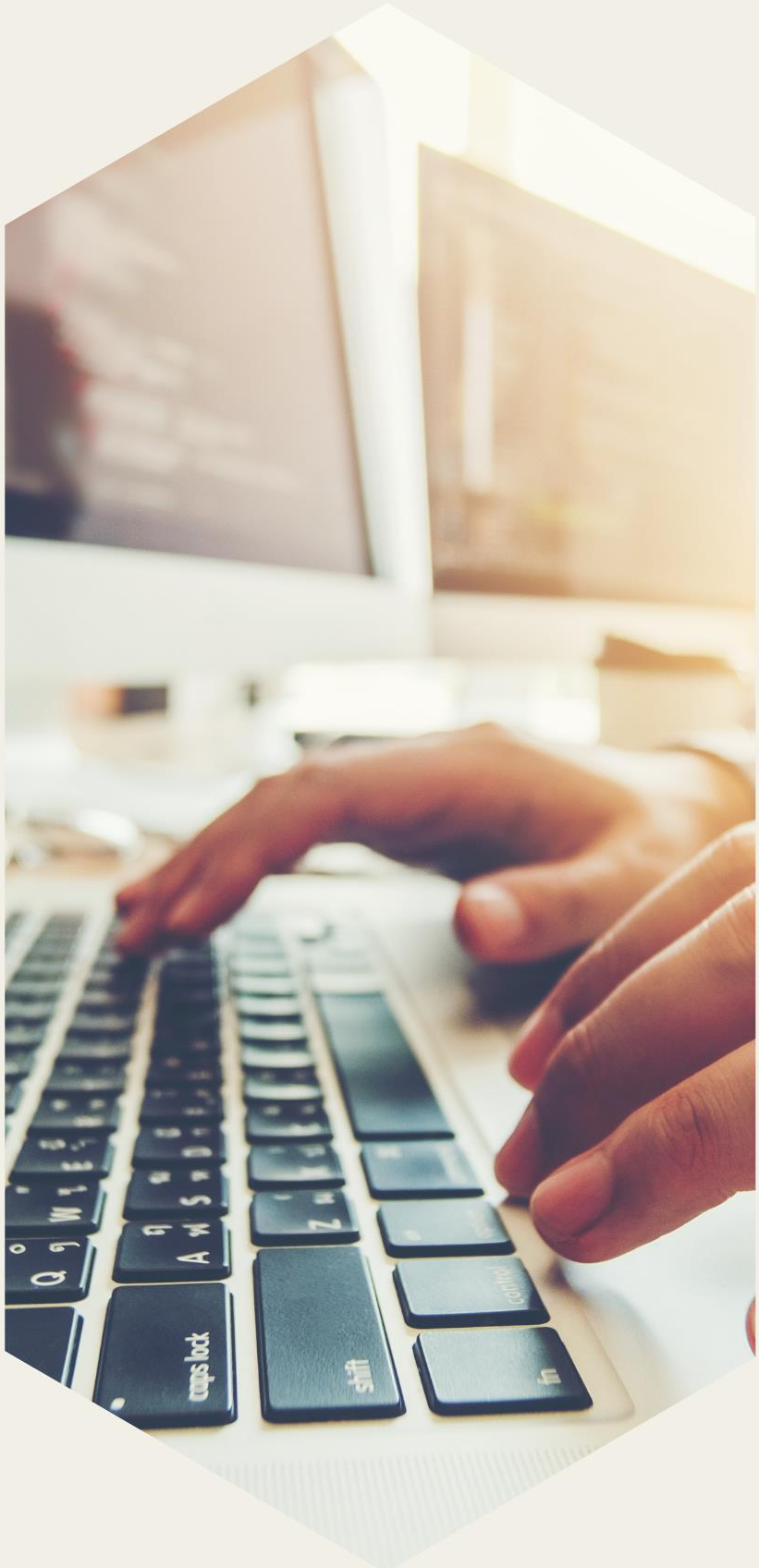
code4sp
coding for social promotion

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

Code4SP E-book



WP3:

Materiais de Formação Code4SP

Elaborado por:

CD DR

GSPE

social hackers academy

CITIZENS IN POWER

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

Informação do Projeto

Acrónimo do Projeto: Code4SP

Título do Projeto: Coding for Social Promotion

Referência do Projeto: 621417-EPP-1-2020-1-PT-EPPKA3-IPI-SOC-IN

Sítio *web* do projeto: www.code4sp.eu

Parceiro(s) autor(es): CodeDoor, CEPROF, SHA and CIP

Versão do Documento: 2

Data de Elaboração: 15/04/2022

Historial do Documento			
Data	Versão	Autor	Descrição
04/04/2022	1	CEPROF	Esboço
15/04/2022	2	Vários Autores	Versão Final

Introdução

Este *e-book* faz parte do projeto europeu Code4SP, Ação-Chave (AC) 3, número de referência 621417-EPP-1-2020-1-PT-EPPKA3-IPI-SOC-IN, cofinanciado pelo Programa Erasmus+, AC3 - Reforma Política, da Comissão Europeia. O mesmo é coordenado pela SPEL e o seu consórcio é composto por mais sete parceiros (CEPROF, CIP - Citizens in Power, CSI - Center for Social Innovation Ltd., CODEDOOR. ORG EV, ZAUG, Action Sinergy SA e Social Hackers Academy), provenientes de quatro países (Chipre, Grécia, Alemanha e Portugal).

O Code4SP tem como objetivos:
Gerar promoção socioeconómica, através de formação orientada para o mercado de trabalho em programação de computadores;
Transferir boas práticas atualmente estabelecidas no âmbito da educação não formal, relativas a programação de computadores, para países do Sul da Europa, considerados mais vulneráveis economicamente e, em simultâneo, propensos a uma exposição sem precedentes a ondas migratórias, caracterizadas por pessoas de baixa condição socioeconómica.

O e-book nasce de uma compilação de materiais de formação para formadores, sendo o objetivo final a plena implementação da metodologia Code4SP nos seus países. Os materiais de formação, no seu todo, são compostos por cenários de aulas, apresentações em formato PowerPoint, tutoriais e vários currículos ajustados ao projeto (emergentes da metodologia original criada pelo parceiro CodeDoor), bem como das melhores práticas de cada país.

Tabela de Conteúdos

Introdução 3
Tabela de Conteúdos 4
1. Programação de Computadores e seus conceitos básicos 5
1.1. Computadores e Programação – introdução 8
1.2. Conceito de *hardware* e *software* 10
1.3 Como os computadores armazenam dados 10
1.4. Como funciona um programa 13
1.5. Programas e linguagens de programação 16
2. HTML –Linguagem de Marcação de HiperTexto 19
2.1. As bases de HTML 22
2.2. Conceitos avançados de HTML 69
2.3.Características HTML5 84
2.4. Referências HTML5 122
3. CSS – Folhas de Estilo em Cascata 123
3.1. CSS - Introdução 126
3.2. CSS – nível avançado 164
3.3. Web Design Reativo do CSS 182
3.4. Grelhas no CSS 192
3.5. CSS SASS 194
4. SQL – Linguagem de Consulta Estruturada 212
4.1. Noções básicas de SQL 215
4.2. Bases de Dados SQL 273
4.3. Referências SQL 301
4.4. Exemplos SQL 315
5. JavaScript 316
5.1. JavaScript num nível básico 320
5.2. JavaScript e DOM 397
5.3. JavaScript e BOM 427
5.4. JavaScript Avançado 447

1. Programação de Computadores e seus conceitos básicos

Informação do Tópico

Tópico:

1. Programação de Computadores e seus conceitos básicos

Pré-requisitos:

Literacia básica de computadores, software básico instalado e conhecimento básico de trabalhar com ficheiros.

Carga horária:

5 horas.

Descrição:

Neste tópico, faremos uma breve introdução à programação de computadores e aos seus conceitos básicos. Trataremos de conceitos de hardware e software, bem como da forma como os computadores interpretam os dados. No final, enfatizaremos a forma como um programa funciona e que tipo de linguagens de programação existem.

Resultados da aprendizagem:

- Reconhecer o conceito de HyperText Markup Language (HTML) na família de linguagens de descrição de documentos.
- Distinguir entre estrutura, conteúdo e estilos de página.
- Usar HTML na construção de páginas para a Web

Material necessário:

- Computador ou portátil
- Ligação à Internet

Cenário de Aula:

O tempo total para este tópico é de 5 horas, e caberá ao formador decidir quanto tempo dedicar ao ensino de cada subtópico. A fim de aproveitar ao máximo todo o tempo disponível, propomos a utilização dos materiais de formação produzidos pelo projeto

(apresentações PPT), que foram concebidos tendo em mente uma utilização eficaz do tempo. Estas apresentações são compostas pelos seguintes elementos:

- Desenvolvimento do subtópico e principais ideias a reter;
- Atividades/Exercícios propostos.

Dito isto, se o/a formador/a seguir a sequência lógica dos PPTs, certamente conseguirá completar a sessão dentro do prazo estipulado. Essas apresentações também podem ser disponibilizadas ao corpo discente para estudo individual.

Subtópicos:

1.1. Introdução aos computadores e à programação

1.2. Conceito de Hardware e Software

1.3. Como os computadores armazenam dados

1.4. Como funciona um programa

1.5. Programação e Linguagens de Programação

Recursos adicionais:

- Khan Academy: recursos úteis em várias línguas
- Marc Andreessen: https://future.a16z.com/software-is-eating-the-world/

1.1. Computadores e Programação – introdução

Os computadores são máquinas que podem ser programadas para executar uma sequência de instruções. A programação é o processo através do qual essas instruções se materializam. Os programas são construídos numa linguagem particular, que fornece uma estrutura para o programador e utiliza instruções específicas para controlar a sequência de operações que o computador executa. Citando Marc Andreesen, "*o software está a comer o mundo*", por isso é importante alertar a sua plateia discente para o facto de que, nos dias de hoje, quase tudo se encontra informatizado e programado (veja também esta página informativa acerca da Internet das Coisas).

Quais algumas formas de utilizar os computadores?

Permita que a turma faça um *brainstorming* ou que crie um mapa mental acerca dos vários propósitos de um computador. São várias as formas de utilizar o computador no nosso dia-a-dia. Algumas delas seguem na lista abaixo:

- Navegar na Internet
- Para verificar o correio eletrónico
- Fazer pesquisas
- Jogar videojogos
- Assistir a filmes ou a programas de TV
- Ouvir música
- Trabalhar num artigo ou folha de cálculo
- Ler as notícias

Que outros dispositivos são também computadores?

Os computadores não são apenas os dispositivos que se encontram nas nossas secretárias ou nos nossos bolsos. São também os dispositivos que fazem funcionar os

nossos carros, aviões e barcos. Estão nos nossos televisores, frigoríficos, e mesmo nos nossos relógios. Em suma, os computadores estão em todo o lado.

"Que software têm utilizado?"

Organize a turma de forma a levar a cabo uma sessão de *brainstorming* ou de desenho de um mapa mental sobre qual(is) o(s) software(s) que os seus integrantes já tenham, porventura, utilizado. Procure informar a turma que cada *software* foi desenvolvido por um determinado programador, seja o *software* que dá vida aos *smartphones*, máquinas de ATM ou até mesmo os televisores que ligam diariamente. Todos estes *softwares* foram desenvolvidos por um/a programador/a ou conjunto de programadores/as.

1.2. Conceito de *hardware* e *software*

Definição de *hardware* e *software*

Hardware refere-se ao conjunto de componentes físicos de um sistema informático, enquanto que o *software* se refere às instruções e dados que fazem o computador funcionar.

Os principais componentes do computador e as suas funções

Um computador tem quatro componentes principais: a unidade central de processamento (CPU), a memória, os dispositivos de entrada, e os dispositivos de saída. O CPU (*Central Processing Unit*) é a parte do computador que efetua os cálculos e controla as outras partes. A memória é o local onde o computador armazena os dados em que está a trabalhar. Os dispositivos de entrada são os dispositivos que o utilizador usa para introduzir dados no computador, tais como o teclado e o rato. Os dispositivos de saída são os dispositivos que o computador utiliza para exibir os resultados dos seus cálculos, como, por exemplo, o monitor e a impressora.

1.3 Como os computadores armazenam dados

O sistema binário – conceito

O sistema binário é uma forma de representar a informação usando dois símbolos: 0 (zero) e 1 (um). É utilizado em sistemas informáticos para armazenar e processar informação, sendo a forma mais simples de o conseguir.

Armazenamento de número e caracteres

ASCII

Os sistemas informáticos armazenam textos e números de várias maneiras, cada um com as suas próprias vantagens e desvantagens. A forma mais comum de armazenar

texto é através dos caracteres ASCII (*American Standard Code for Information Interchange*). No ASCII, cada caractere é representado por um número, de 0 (zero) a 127. Este número é chamado de código ASCII do caractere. Quando um computador armazena texto no modo ASCII, isto quer dizer que armazena os códigos ASCII para cada caractere no texto.

Unicode

Outra forma de armazenar texto é salvando-o como caracteres Unicode. Unicode é uma norma internacional que define um número único para cada carácter em cada língua. Quando um computador armazena texto como caracteres Unicode, armazena o código Unicode para cada caractere do texto.

UTF-8

UTF-8 é uma codificação de caracteres que consegue armazenar texto e números num único carácter Unicode. Esta codificação é amplamente utilizada na Internet porque pode codificar todos os caracteres numa variedade de línguas. O UTF-8 é uma codificação de comprimento variável, o que significa que pode codificar caracteres de diferentes tamanhos. A codificação mais pequena é de 1 *byte*, sendo a maior de 4 bytes. Esta codificação é retrocompatível com ASCII, o que significa que o texto ASCII será codificado em UTF-8 usando 1 byte.

Números

A forma mais comum de armazenar números é como números inteiros binários. Em binário, cada número é representado por uma sequência de 0s e 1s. Por exemplo, o número 12 pode ser representado como: 01001000 O número 12 também pode ser representado em hexadecimal, que é um sistema de numeração de base 16. Em hexadecimal, cada número é representado por uma cadeia de dígitos hexadecimais. Por exemplo, o número 12 pode ser representado como: C Quando um computador armazena um número em binário ou hexadecimal, armazena o valor inteiro do número.

Outros tipos de dados

Os computadores são frequentemente referidos como dispositivos digitais. O termo digital pode ser usado para descrever qualquer coisa que utilize números binários. Dados binários são dados que são armazenados em binário, e um dispositivo digital é qualquer dispositivo que funcione com dados binários. Nesta secção discutimos como os números e caracteres são armazenados em sistema binário, mas os computadores também funcionam com muitos outros tipos de dados digitais. Por exemplo, considerando as fotografias que tiramos com a máquina fotográfica digital, estas imagens são compostas por pequenos pontos de cor conhecidos como *pixels* (o termo *pixel* significa elemento de imagem). Cada *pixel* de uma imagem é convertido para um código numérico que representa a cor do pixel. O código numérico é armazenado na memória como um número binário. A música que toca no seu leitor de CD, iPod ou leitor de MP3 é também digital. Uma canção digital é composta por pequenas peças de música chamadas *samples*. Cada amostra é transformada num número binário, que pode ser armazenado na memória de um computador. Quanto mais amostras uma canção tiver, soará de forma mais parecida em relação à música original quando reproduzida. Uma canção com qualidade de CD tem mais de 44.000 *samples* por segundo!

1.4. Como funciona um programa

Existem muitos tipos diferentes de programas de computador, mas todos eles têm os mesmos componentes básicos: uma interface de utilizador, um processador e memória. A interface de utilizador permite que este introduza informações e instruções no programa, o processador executa as instruções, e a memória armazena o programa e os dados que este último processa. A maioria dos programas de computador são escritos numa linguagem de programação de alto nível, uma linguagem concebida para ser de fácil leitura e escrita pelo ser humano. Contudo, o processador só pode compreender o código da máquina, que é uma série de uns e zeros. Portanto, antes de um programa poder ser executado, deve ser convertido em código de máquina. Ou seja, terá de ser feito por um programa chamado *compilador*. O compilador lê o programa e converte-o em código de máquina. Depois armazena o código da máquina num ficheiro chamado *executável*. Quando o utilizador executa o programa, o executável é carregado na memória e o processador executa as instruções.

O ciclo *fetch-decode-execute*

O ciclo *fetch-decode-execute* é o processo básico que um computador utiliza para executar instruções. O ciclo começa quando o computador vai buscar uma instrução à memória. Em seguida, descodifica a instrução para determinar o que é suposto fazer. Finalmente, executa a instrução. O ciclo repete-se então, recuperando a instrução seguinte da memória.

Da linguagem de máquina à linguagem de montagem

Como a programação em linguagem de máquina, que consiste apenas em código binário, é demasiado complexa para um ser humano, foi criada uma linguagem de montagem. A linguagem de montagem (ou *assembly*) trata-se de uma linguagem de programação de baixo nível para um computador, microprocessador, ou outro dispositivo programável, na qual o programador utiliza as instruções da linguagem

assembly para controlar o funcionamento do dispositivo. A linguagem de montagem é específica para um determinado microprocessador ou família de microprocessadores. Consiste numa série de códigos mnemónicos, nomes simbólicos para as operações que o microprocessador pode realizar, e os operandos (dados) sobre os quais estas operações devem ser realizadas. A linguagem de montagem é convertida em código de máquina, uma forma de código binário específico de um determinado tipo de computador e que pode ser compreendido pelo processador do computador. O código da máquina é a única forma de código que o processador pode executar diretamente.

Mesmo sendo mais fácil do que a programação em linguagem de máquina, a programação em linguagem de montagem não é suficientemente útil para produzir código fonte rápido e fácil de ler. Por conseguinte, as linguagens de programação de alto nível (tais como *C#* ou *python*) foram criadas.

Atualmente, existem muitas linguagens de programação de alto nível. Algumas das mais comuns são Java, Python e Ruby. As linguagens de programação de alto nível são mais fáceis de utilizar do que as linguagens de programação de baixo nível. Permitem que se concentre na tarefa em mãos, em vez de se concentrar nos detalhes do computador. Isto torna-as ideais para a criação de aplicações e programas. As linguagens de programação de alto nível também tendem a ser mais tolerantes do que as linguagens de programação de baixo nível. Se cometer um erro ao escrever código numa linguagem de alto nível, o compilador será normalmente capaz de o corrigir por si. Isto pode poupar-lhe muito tempo e frustração ao "*bater código*".

Palavras-chave, Operadores, e Sintaxe: uma visão geral

Há muitas linguagens de programação de alto nível disponíveis hoje em dia. Cada uma tem o seu conjunto único de palavras-chave, operadores e sintaxe. Para ser eficaz com uma linguagem de programação de alto nível, é importante estar familiarizado com as palavras-chave, operadores, e sintaxe específicos utilizados por essa linguagem. Algumas das palavras-chave mais comuns utilizadas nas linguagens de programação

de alto nível incluem: *if, then, else, while, for, do, break, continue*. Estas palavras-chave são utilizadas para controlar o fluxo da execução do programa. Os operadores são símbolos que representam operações que podem ser realizadas sobre valores. Os operadores mais comuns incluem: + *(addition), - (subtraction), * (multiplication), / (division), e % (modulus)*. Estes operadores podem ser utilizados para calcular os resultados das expressões. A sintaxe de uma linguagem de programação é o conjunto de regras que dirigem a forma como o código deve ser escrito para ser interpretado pelo compilador ou pelo intérprete. A sintaxe de uma linguagem de programação de alto nível é tipicamente mais tolerante do que a sintaxe de uma linguagem de nível inferior. Isto pode facilitar aos principiantes a aprendizagem da programação.

Compiladores e Intérpretes

Os compiladores e intérpretes de computador são ferramentas importantes para os programadores de *software*. Um compilador pega no código escrito numa língua e converte-o em código que pode ser executado numa máquina diferente. Um intérprete pega no código escrito numa língua e executa-o tal como está, sem o compilar primeiro. Os compiladores são tipicamente utilizados para linguagens que têm muita estrutura, como *C* ou *Java*. Os intérpretes são tipicamente utilizados para linguagens que são mais flexíveis, como Python ou Ruby. Os compiladores produzem normalmente códigos mais rápidos do que os intérpretes. No entanto, os intérpretes são tipicamente mais *portáteis*, o que significa que podem funcionar em mais tipos de máquinas. O instrumento a utilizar depende da situação. Se a velocidade é importante, um compilador é uma melhor escolha. Se a portabilidade é importante, um intérprete é uma melhor alternativa.

1.5. Programas e linguagens de programação

As linguagens de programação são usadas para criar programas, que são usados para controlar o comportamento de uma máquina, tipicamente um computador. Uma linguagem de programação fornece uma estrutura para o programador dar instruções à máquina, e uma forma de comunicar essas instruções a outros programadores. Existem muitas linguagens de programação em uso atualmente. As mais populares são C, Java, Python, e JavaScript.

Tipos de linguagens

Existem dezenas de linguagens de programação em uso hoje em dia, podendo ser sumariamente classificadas em cinco categorias:

- **Linguagens de programação de baixo nível:** Estas linguagens de programação são muito próximas do *hardware* e são utilizadas para programar microprocessadores e outros dispositivos de baixo nível. Não são fáceis de aprender e não são populares para a programação de uso geral. Exemplos: Linguagem de montagem, linguagem de programação C, e linguagem de montagem de baixo nível.
- **Linguagens de programação de alto nível:** Estas linguagens de programação foram concebidas para serem fáceis de aprender e utilizar. São populares para a programação de uso geral. Exemplos: Java, C++ e Python.
- **Linguagens de *scripting*:** são concebidos para serem fáceis de usar e são populares para de *scripting*. Exemplos: Python, Ruby e JavaScript.
- **Línguas específicas do domínio:** são concebidas para uma tarefa ou indústria específica. Não são fáceis de aprender e não são populares para programação de uso geral. Exemplos: MATLAB, SQL, and FORTRAN.
- **Linguagens de programação orientadas a objetos:** são baseadas no paradigma da programação orientada para objectos. Exemplos: Java, C++, and Python.

Do programa de alto nível ao ficheiro executável

Quando um programa de computador é escrito numa língua de alto nível, é primeiramente traduzido para uma língua de nível inferior, que é mais facilmente compreendida por máquinas. A língua de nível inferior é então compilada num ficheiro executável, que pode ser executado num computador.

IDEs (*Integrated Development Environments*)

Um Ambiente Integrado de Desenvolvimento (IDE) é uma aplicação de software que proporciona instalações abrangentes aos programadores de computadores para o desenvolvimento de *software*. Um IDE consiste tipicamente num editor de código fonte, ferramentas de automação de construção e um depurador. O editor de código fonte permite ao programador escrever o código, enquanto as ferramentas de automatização da construção automatizam o processo de compilação desse código numa forma que o computador pode executar. O depurador permite ao programador percorrer o código, examinando o estado do programa em cada ponto da sua execução. As IDEs são muitas vezes utilizadas em conjunto com um sistema de controlo de versões, o que permite a diferentes programadores que trabalham no mesmo projeto partilhar e fundir as suas alterações sem problemas.

Os elementos mais comuns em linguagens de programação

As linguagens de programação informática partilham uma série de elementos comuns, apesar das suas diferenças. Todas as linguagens de programação têm uma forma de representar instruções para o computador de uma forma que o computador possa compreender. Isto é normalmente chamado código, ou código fonte. Os programadores utilizam o código para criar programas e aplicações de *software*. Todas as linguagens de programação têm também uma forma de organizar instruções para que possam ser reutilizadas, modificadas ou partilhadas com outros programadores. A isto chamamos normalmente “biblioteca” ou “módulo”. As bibliotecas e os módulos permitem aos

programadores criar programas complexos, baseando-se no trabalho de outros programadores. Por fim, todas as linguagens de programação têm uma forma de transmitir informação ao utilizador sobre o que o programa está a fazer e como está a funcionar. Isto é normalmente chamado de informação de saída ou *debug*. A informação de saída e *debug* ajuda os programadores a compreender e corrigir problemas dos seus programas.

Programação processual e orientada a objetos

Há dois tipos principais de programação: processual e orientada para objetos. A programação processual envolve um processo passo-a-passo, enquanto a programação orientada aos objetos envolve a criação de objetos que interagem uns com os outros. A programação processual é muitas vezes vista como mais simples do que a programação orientada a objetos. É fácil aprender os passos necessários para completar uma tarefa, e é fácil alterar a ordem desses passos sem afetar o resultado. Contudo, a programação de procedimentos pode ser menos eficiente porque pode ser difícil reutilizar o código que foi escrito para uma tarefa específica. A programação orientada a objetos é mais complexa que a programação processual, mas permite maior flexibilidade e reutilização do código. Os objetos podem ser criados para tarefas específicas e depois reutilizados conforme necessário. Além disso, o código orientado para objetos é muitas vezes mais fácil de ler e compreender do que o código processual. Contudo, a programação orientada aos objetos pode ser mais difícil de aprender e pode ser menos eficiente do que a programação processual.

2. HTML – Linguagem de Marcação de HiperTexto

Informação do Tópico:

Tópico:

2. HTML

Pré-requisitos:

Literacia básica de computadores, software básico instalado e conhecimento básico de trabalhar com ficheiros.

Carga horária:

10 horas.

Descrição:

Neste tópico, vamos cobrir os básicos de HTML, para atualizar os alunos no mundo da programação, depois de adquirir alguns conceitos básicos. Definimos os elementos, atributos e todos os outros termos importantes que eles podem ter ouvido e onde esses termos se encaixam na linguagem. Também mostramos como um elemento HTML é estruturado, como uma página HTML típica é estruturada e explicamos outros recursos básicos importantes da linguagem.

Resultados da aprendizagem:

- Reconhecer o conceito de HyperText Markup Language (HTML) na família de linguagens de descrição de documentos.
- Distinguir entre estrutura, conteúdo e estilos de página.
- Usar HTML na construção de páginas para a Web.

Material necessário:

- Computador ou portátil
- Conexão à Internet
- Construtor online de Websites (https://sites.google.com/new)
- Editor online de texto (https://www.w3schools.com/html/default.asp)

Cenário de Aula:

O tempo total para este tópico é de 10 horas e caberá ao formador/coach decidir quanto tempo dedicar ao ensino de cada subtópico. Para aproveitar ao máximo todo o tempo disponível, propomos a utilização dos materiais de formação produzidos pelo projeto (apresentações PPT), que foram concebidos a pensar numa utilização eficaz do tempo. Estas apresentações são compostas pelos seguintes elementos:

- Desenvolvimento do subtópico e principais ideias a reter;
- Atividades/Exercícios propostos.

Dito isto, se o formador/coach seguir a sequência lógica dos PPTs, certamente conseguirá completar a sessão dentro do prazo estipulado. Essas apresentações também podem ser disponibilizadas aos alunos para estudo individual.

Subtópicos:

2.1. As bases de HTML

2.2. Conceitos avançados de HTML

2.3. Características de HTML5

2.4. Referências de HTML5

Recursos adicionais:

- HTML reference guide
- W3Schools - guia para cada elemento HTML e regra CSS, e exemplos para cada um deles
- Khan Academy: recursos e vídeos úteis sobre codificação HTML e CSS, em vários idiomas

2.1. As bases de HTML

HTML (Hypertext Markup Language) **não é** uma linguagem de programação. É uma linguagem de marcação que comunica aos navegadores da web como estruturar as páginas da web visitadas. Pode ser tão complexo ou tão simples quanto o desenvolvedor web deseje que seja. HTML compreende uma série de elementos, usados para incluir, envolver ou elevar diferentes partes do conteúdo, para o fazer aparecer ou agir de uma determinada maneira. As tags anexas podem transformar o conteúdo num hyperlink para conectar a outra página, colocar palavras em itálico e assim por diante. Por exemplo, considerando a seguinte linha de texto:

```
HTML is cool.
```

***Figura 1** – Linha de texto "HTML is cool" (Fonte: Autor)*

Se alguém quisesse que o texto ficasse sozinho, poderia especificar que é um parágrafo, colocando-o num elemento de parágrafo (<p>):

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

***Figura 2** – Codificação para o parágrafo "HTML is cool" (Fonte: Autor)*

Nota: Tags em HTML não diferenciam maiúsculas de minúsculas. Isso significa que eles podem ser escritos em maiúsculas ou minúsculas. Por exemplo, uma tag <title> pode ser escrita como <title>, <TITLE>, <Title>, <TiTlE>, etc., e funcionará. No entanto, é uma prática recomendada escrever todas as tags em letras minúsculas para efeitos de consistência e legibilidade.

Anatomia do elemento HTML

Vamos explorar mais o elemento de parágrafo da secção anterior:

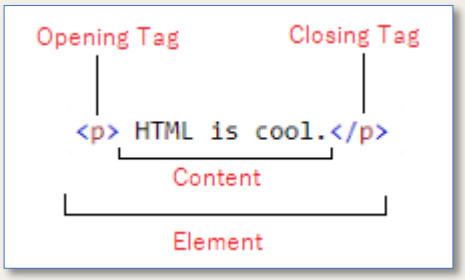


***Figura 3** – Anatomia de um elemento HTML (Fonte: Autor)*

Portanto, a anatomia do elemento é composta por:

A tag de abertura: o nome do elemento (neste exemplo, p para parágrafo), entre sinais angulares de abertura e fecho. Essa tag de abertura marca onde o elemento começa ou começa a ter efeito. Neste exemplo, ele vem primeiro, no início do texto do parágrafo.

O conteúdo: Isto é o conteúdo do elemento. Neste exemplo, é o texto do parágrafo.

A tag de fecho: É o mesmo que a tag de abertura, exceto que inclui uma barra antes do nome do elemento. Isso marca onde o elemento termina. Não incluir uma tag de fecho é um erro comum para iniciantes que pode produzir resultados peculiares.

O **elemento** é a tag de abertura, seguido pelo conteúdo, seguido pela tag de fecho.

Nota: Alguns elementos HTML não possuem conteúdo (como o elemento <br>). Esses elementos são chamados de “elementos vazios”. Eles não têm uma tag final!

Navegadores Web

O objetivo de um navegador da Web (Chrome, Edge, Firefox, Safari) é ler documentos HTML e exibi-los corretamente.

O navegador não mostra as tags HTML. Usa-as para determinar como exibir o documento:

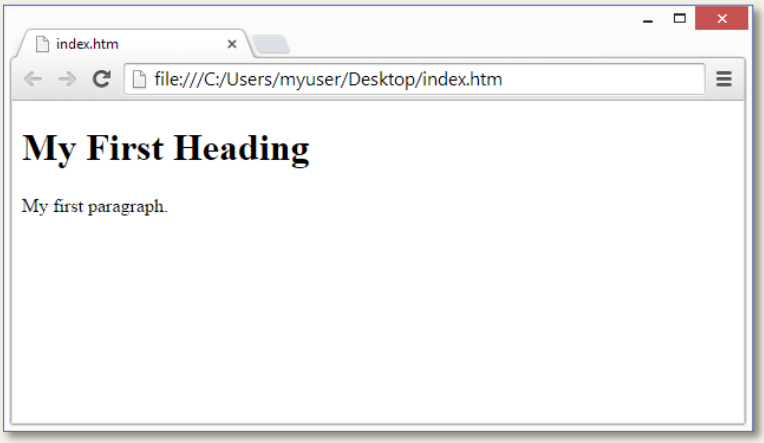


***Figura 4** – Um documento HTML determinado num navegador web (Fonte: https://www.w3schools.com/html/html_intro.asp)*

Estrutura da página HTML

Uma página HTML deve ser estruturada da seguinte forma:

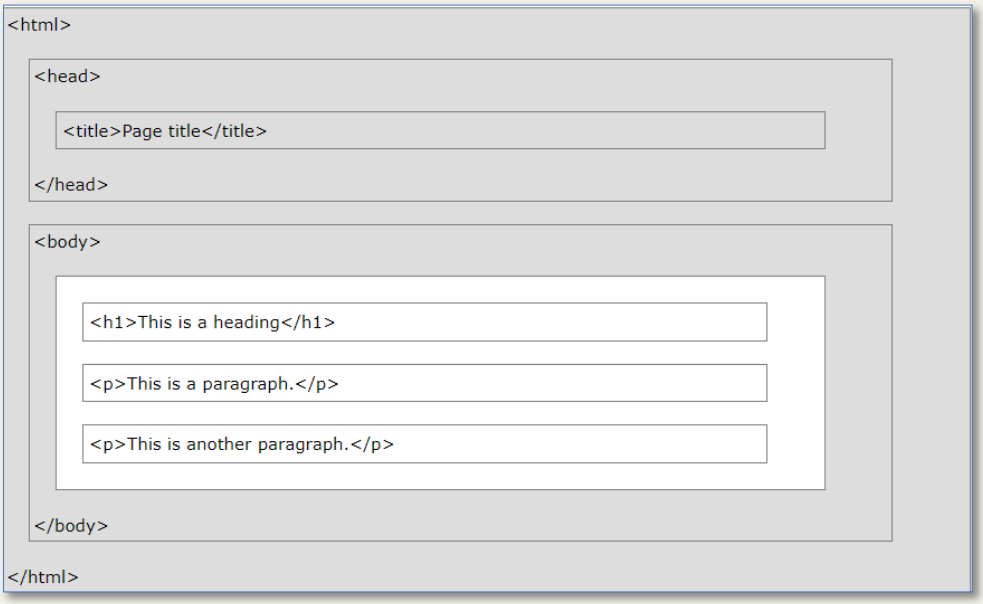


***Figura 5** – Estrutura de uma página HTML (Fonte: https://www.w3schools.com/html/html_intro.asp)*

O conteúdo dentro da secção <body> (a área branca acima) será exibido num navegador. O conteúdo dentro do elemento <title> será mostrado na barra de título do navegador ou na aba da página.

História do HTML

Desde os primeiros dias da World Wide Web, já existiram várias versões de HTML:

Ano	Versão
1989	Tim Berners-Lee inventou www
1991	Tim Berners-Lee inventou HTML
1993	Dave Raggett esboçou HTML+
1995	HTML Working Group definiu HTML 2.0
1997	W3C Recomendação: HTML 3.2
1999	W3C Recomendação: HTML 4.01
2000	W3C Recomendação: XHTML 1.0
2008	WHATWG HTML5 Primeiro esboço público
2012	WHATWG HTML5 Padrão de vida
2014	W3C Recomendação: HTML5
2016	W3C Recomendação de candidatos: HTML 5.1
2017	W3C Recomendação: HTML5.1 2nd Edition
2017 -	W3C Recomendação: HTML5.2

Tabela 1 *– História HTML (Fonte: https://www.w3schools.com/html/html_intro.asp)*

Este manual segue o mais recente padrão HTML5.

Editores HTML

Um simples editor de texto é tudo o que é preciso para aprender HTML.

Aprender HTML Usando Notepad ou TextEdit

Páginas Web pode ser criadas e modificadas usando editores profissionais de HTML.

Contudo, para aprender HTML, um simples editor de texto com o Notepad (PC) ou o TextEdit (Mac) é recomendado. Usar um simples editor de texto pode ser uma boa forma de aprender HTML.

Os passos abaixo devem ser seguidos para criar a primeira página Web do aprendiz, usando o Notepad ou o TextEdit.

Passo 1: Abrir o Notepad (PC)

(Windows 8 ou mais recente: Abrir Iniciar (o símbolo de janela no canto inferior esquerdo do ecrã). Escrever Notepad.

Windows 7 ou mais antigo: Abrir Iniciar > Programas > Accessórios > Notepad)

- Abrir TextEdit (Mac)
- Abrir Procurar > Aplicações > TextEdit
- Fazer com que a aplicação grave os ficheiros da forma correta. Em Preferências > Formato > escolher "Plain Text"
- Depois, por baixo de "Open and Save", carregar na caixa que diz "Display HTML files as HTML code instead of formatted text".
- Depois, **abrir novo documento** para colocar o código.

Passo 2: Escrever HTML

- Escreva ou copie o seguinte Código de HTML para o NotePad:

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

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

Passo 3: Guardar a página HTML

- Guarde o ficheiro no seu computador.
- Selecione File > Save as no menu do NotePad.
- Identifique o ficheiro como "index.htm" e defina a codificação para UTF-8 (a codificação preferencial para ficheiros HTML).

Passo 4: Visualize a página HTML no seu navegador

- Abra o ficheiro HTML guardado no seu navegador preferido (duplo click no ficheiro ou click direito e escolher "Open with").

O resultado será semelhante à seguinte imagem:

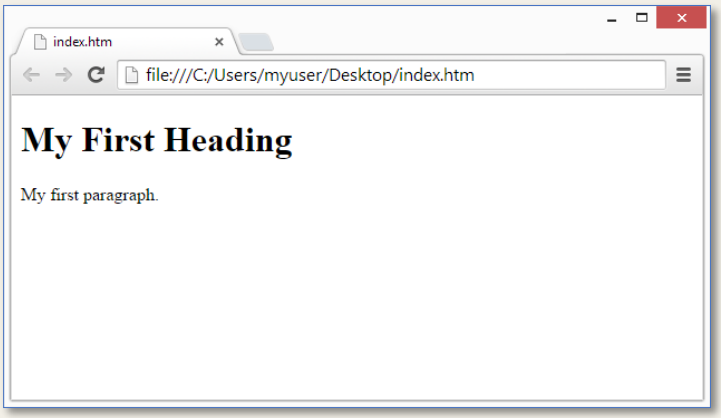


***Figura 6** – Um documento HTML determinado num navegador web (Fonte: https://www.w3schools.com/html/html_intro.asp)*

Exemplo básicos de HTML

Nesta secção, alguns exemplos básicos de HTML serão partilhados.

Documentos HTML

Todos os documentos HTML devem começar com uma declaração tipo documento: <!DOCTYPE html>.

A declaração <!DOCTYPE> representa o tipo de documento e ajuda os navegadores a exibir páginas web de forma correta. Só deve aparecer uma vez, no topo da página (antes de quaisquer tags HTML). Não é sensível ao caso.

A declaração <!DOCTYPE> para HTML5 é: <!DOCTYPE html>

Cabeçalhos HTML

Cabeçalhos HTML são definidos com as tags <h1> até <h6>.

<h1> define o cabeçalho mais importante. <h6> define o cabeçalho menos importante:

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

Figura 7 *– Cabeçalhos HTML (Fonte: https://www.w3schools.com/html)*

Parágrafos HTML

Parágrafos HTML são definidos com a tag <p>:

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

Figura 8 *– Parágrafos HTML (Fonte: https://www.w3schools.com/html)*

Ligações HTML

Ligações HTML são definidas com a tag <a>:

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

***Figura 9** – Ligação HTML(Fonte: https://www.w3schools.com/html)*

O destino da ligação é específico no atributo href.

Atributos são usados para providenciar informação adicional acerca de elementos HTML.

Mais acerca dos atributos será ensinado mais à frente.

Imagens HTML

Imagens HTML são definidas com a tag <img>.

O ficheiro fonte (src), texto alternativo (alt), largura e altura são dados como atributos:

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

***Figura 10** – Imagens HTML (Fonte: https://www.w3schools.com/html)*

Como ver uma fonte HTML?

Ver Código de fonte HTML:

- Click direito numa página HTML e selecione "View Page Source" (no Chrome), "View Source" (no Edge) ou semelhante noutros navegadores. Isto irá abrir uma janela que contém o código fonte da página HTML.

Inspecionar um elemento HTML:

- Clique com o botão direito do rato num elemento (ou numa área em branco) e escolha "Inspecionar" ou "Inspecionar Elemento" para ver de que elementos são compostos (irá ver o HTML e o CSS). Pode também editar o HTML ou CSS dinamicamente no painel Elementos ou Estilos, que é aberto.

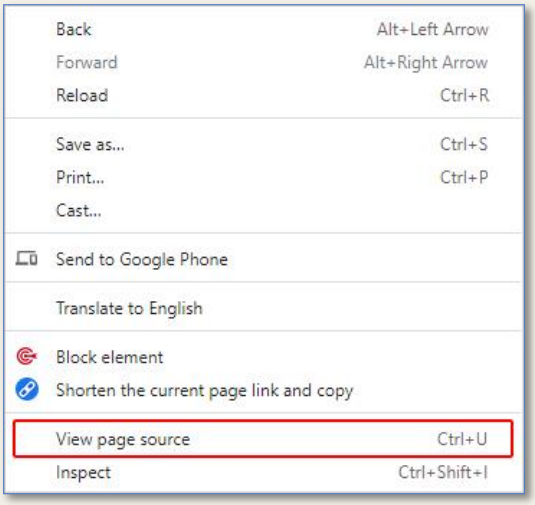


Figura 11 *– Como ver a página fonte (Fonte:Author)*

A estrutura de um documento HTML

O documento HTML começa com <html> e acaba com </html>. A parte visível do documento HTML está entre <body> e </body>.

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

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

</body>
</html>
```

Figura 12 *– Declaração tipo documento, HTML e partes visíveis do documento HTML (Fonte: https://www.w3schools.com/html)*

Atributos HTML

Os atributos HTML fornecem informações adicionais sobre elementos HTML e todos os elementos HTML podem tê-los. Os atributos fornecem informações adicionais sobre

os elementos, sendo sempre especificados na tag inicial. Eles geralmente vêm em pares nome/valor como: name="value".

Lista de atributos comuns:

- **href** – a tag <a> define um hyperlink. O atributo href específica o URL da página para qual a ligação vai, da seguinte forma:

Visit W3Schools

- **src** – a tag <img> é usada como uma imagem embutida numa página HTML. O atributo <src> específica o caminho para a página ser exibida, como visto abaixo:

<img src="img_girl.jpg">

Existem duas formas de especificar o URL no atributo <src>:

1. **URL absoluto** – Liga a uma imagem externa que está situada num outro website. E.g.: *src="https://www.w3schools.com/images/img_girl.jpg".*

Notas: Imagens externas podem estar sob **direitos de autor**. Se alguém não tiver permissão para usar, poderá estar a violar as leis dos direitos de autor. Além disso, as imagens externas não podem ser controladas; elas podem ser removidas ou alteradas abruptamente.

2. **URL relativo** - Liga a uma imagem situada no site. Aqui, o URL não inclui o nome do domínio. Se o URL começar sem uma barra, será relativo à página atual. E.g*: src="img_girl.jpg"*. Se o URL começa com uma barra, será relativo ao domínio. E.g.: *src="/images/img_girl.jpg".*

Dica: É quase sempre melhore usar URLs relativos. Eles não vão partir se o domínio mudar.

- **Atributos de largura e altura** – a tag <img> deve também compreender os atributos de largura e altura, que estipula a largura e altura da imagem (em pixels):

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

- **alt –** o atributo alt necessário para a tag <img> específica um texto alternativo para uma imagem, se a imagem não puder ser exibida, por alguma razão. Isto pode ser devido a uma conexão lenta, um erro no atributo src, ou se o usuário usa um leitor de ecrã.

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

  Se tentarmos exibir uma imagem que não existe, o valor do atributo alt será exibido, como segue:

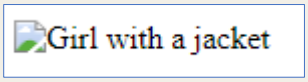


Figura 13 – Valor Alt se a imagem não existe (Fonte: https://www.w3schools.com/html)

- **Estilo** – o atributo estilo é usado para adicionar estilos a um elemento, como cor, fonte, tamanho e mais.

This is a red paragraph.

Resultado:

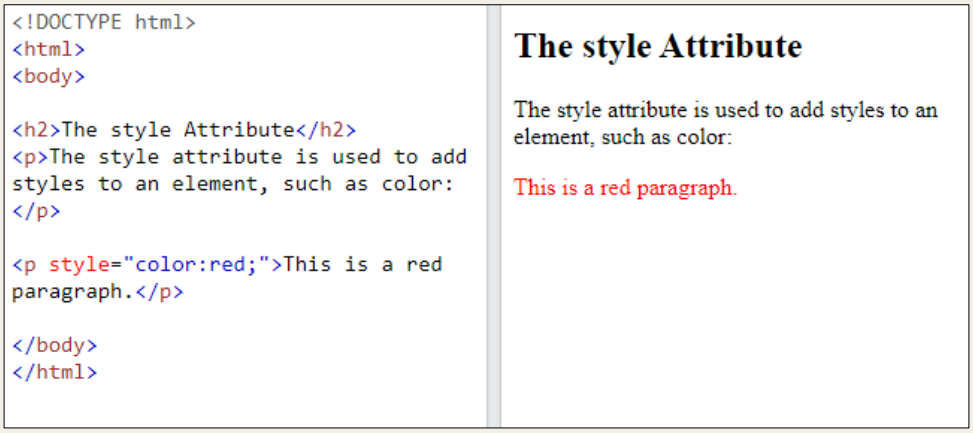


Figura 14 – Código para colorir um parágrafo (Fonte: https://www.w3schools.com/html)

- **lang** – o atributo lang deve sempre estar incluído dentro da tag <html>, para declarar a linguagem da página Web. Isto destina-se a suportar motores de busca e navegadores. O exemplo abaixo estipula o inglês como o idioma em uso:

<!DOCTYPE html>
<html lang="en">
<body>
...

```
</body>
</html>
```

Os códigos de país também podem ser adicionados ao código de idioma no atributo lang. Assim, os dois primeiros caracteres expressam o **idioma** da página HTML e os dois últimos caracteres descrevem o **país**.

O exemplo a seguir especifica português como idioma e Portugal como país:

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

HTML Language Code Reference incluiu os códigos de todas as linguagens.

- **Título –** este atributo descreve algumas informações adicionais sobre um elemento. O seu valor será exibido como uma dica de ferramenta quando o ponteiro do rato passar sobre o elemento, como segue:

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

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

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

</body>
</html>
```

The title Attribute

I'm a header

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

***Figura 15** – Código para exibir uma dica de ferramenta 'título' (Fontee: Autor)*

Recomendação: É altamente recomendável usar sempre atributos em letras minúsculas e citar sempre valores de atributos para tipos de documentos mais rígidos como XHTML.

Cabeçalhos HTML

Cabeçalhos HTML são títulos ou legendas que se deseja exibir numa página da web.

Figura 16 *– Código para adicionar cabeçalhos (Fonte: Autor)*

Os cabeçalhos HTML são delineados com as tags <h1> a <h6>. <h1> identifica o título mais importante. <h6> descreve o título menos importante. Os cabeçalhos <h1> devem ser usados para os cabeçalhos principais, seguidos pelos cabeçalhos <h2>, depois os menos importantes <h3> e assim por diante.

Os títulos são de suma importância, pois os mecanismos de pesquisa os usam para indexar a estrutura e o conteúdo das páginas da web. Os usuários geralmente navegam em uma página por seus títulos. É importante usar títulos para exibir a estrutura do documento.
É importante dizer que, por definição, os navegadores adicionam automaticamente uma margem antes e depois de um título.
Recomendação: É altamente recomendável usar cabeçalhos HTML apenas para cabeçalhos, não para tornar o texto grande ou em negrito.

Além disso, no tópico CSS, será ensinado que o tamanho dos cabeçalhos pode ser especificado usando o atributo style, usando a propriedade CSS font-size, como segue:

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

Parágrafos HTML

Um continua constantemente em uma nova linha e normalmente é um bloco de texto. É definido pelo elemento HTML <p> e, como os cabeçalhos, os navegadores adicionam automaticamente margem antes e depois de algum parágrafo.

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

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

</body>
</html>
```

Code4SP helps me to code.

I love Code4SP.

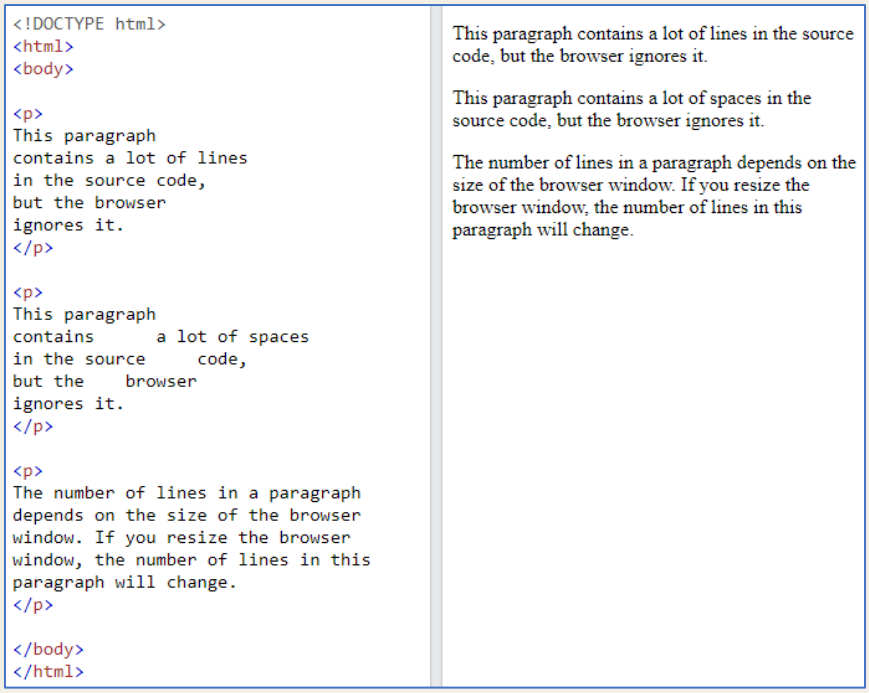
Coding is so great!

Figura 17 *– Código para adicionar parágrafos (Fonte: Autor)*

Exibição HTML

Não se pode ter certeza de como o HTML será apresentado, pois pode variar de ecrã para ecrã. Com HTML, a exibição não pode ser alterada adicionando espaços extras ou linhas extras no código HTML.

O navegador removerá automaticamente quaisquer espaços e linhas extras quando a página for exibida, conforme mostrado no exemplo a seguir:



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

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

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

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

</body>
</html>
```

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

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

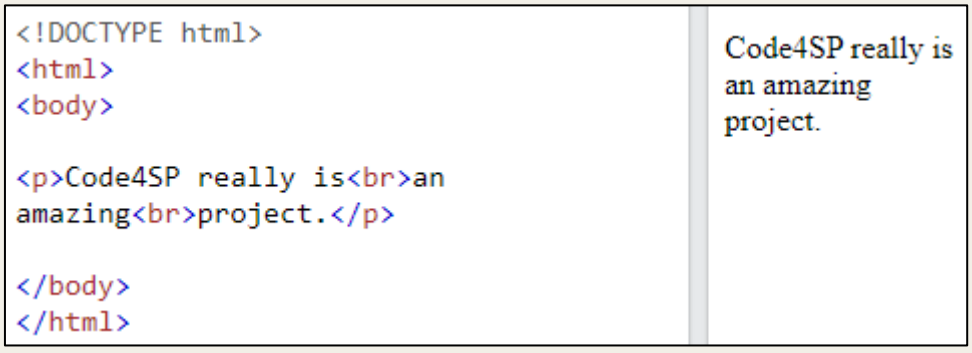
The number of lines in a paragraph depends on the size of the browser window. If you resize the browser window, the number of lines in this paragraph will change.

Figura 18 *– Exemplo de como os navegadores por vezes ignoram os espaços (Fonte: https://www.w3schools.com/html)*

Cortes de linha HTML

O elemento HTML <br> define o corte de linha. É uma tag vazia, o que significa que não tem tag de fecho.

 deve ser usado se alguém quiser uma nova linha sem começar um novo parágrafo:



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

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

</body>
</html>
```

Code4SP really is
an amazing
project.

Figura 19 *– O elemento
 (Fonte: Autor)*

Estilos HTML

O atributo de estilo HTML é usado para adicionar estilos a um elemento, como cor, fonte, tamanho, etc. Para definir o estilo de um elemento HTML, deve-se usar o atributo de estilo. Ele tem a seguinte sintaxe (deve-se notar que propriedade e valor são recursos CSS, a serem aprendidos posteriormente).

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

- **Cor de fundo**

A propriedade CSS *background-color* especifica a cor de fundo para um elemento HTML.

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

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

</body>
</html>
```

Figura 20 *– Definir cor de fundo (Fonte: Autor)*

- **Cor de texto**

A propriedade de cor CSS descreve a cor do texto para um elemento HTML:

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

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

</body>
</html>
```

Figura 21 *– Definir cor de fundo (Fonte: Autor)*

- **Fonte**

A propriedade CSS *font-family* define a fonte a ser usada para um elemento HTML:

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

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

</body>
</html>
```

Code4SP

Coding for Social Promotion.

Figura 22 *– Definir a fonte a ser usada para um elemento HTML (Fonte: Autor)*

- **Tamanho de texto**

A propriedade CSS *font-size* identifica o tamanho do texto para um elemento HTML:

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

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

</body>
</html>
```

CODE4SP

Coding for Social Promotion.

Figura 23 *– Definir o tamanho de texto para um elemento HTML (Fonte: Autor)*

- **Alinhamento de texto**

O recurso de alinhamento de texto CSS define o alinhamento de texto horizontal para um elemento HTML:

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

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

</body>
</html>
```

CODE4SP

Coding for Social Promotion.

Figura 24 *– Característica Alinhamento de texto (Fonte: Autor)*

Formatação de texto HTML

HTML compreende vários elementos para definir o texto com uma implicação especial (negrito, itálico, subscrito, sobrescrito, etc.).

Os seguintes são os **elementos de formatação HTML**:

Propriedade	Resultado	Definição	Exemplo
<b>	Texto em negrito	O elemento HTML <b> especifica texto em negrito, sem nenhuma importância extra.	**Texto em negrito**
	Texto importante	O elemento HTML descreve o texto com grande importância. O conteúdo interno geralmente é exibido em negrito.	**Texto importante**
<i>	Texto itálico	O elemento HTML <i> define uma parte do texto em uma voz ou humor alternativo. O conteúdo	*Texto itálico*

		dentro é normalmente exibido em itálico.	
	Texto enfatizado	O elemento HTML define o texto enfatizado. O conteúdo dentro é normalmente exibido em itálico.	*Texto enfatizado*
<mark>	Texto marcado	O elemento HTML <mark> define o texto que deve ser marcado ou destacado.	Texto marcado
<small>	Texto pequeno	O elemento HTML <small> define um texto menor.	Texto pequeno
<del>	Texto apagado	O elemento HTML <del> define o texto que foi excluído de um documento. Os navegadores geralmente riscam uma linha através do texto excluído.	~~Texto apagado~~
<ins>	Texto inserido	O elemento HTML <ins> define um texto que foi inserido em um documento. Os navegadores geralmente sublinham o texto inserido:	Texto ~~Inserido~~
<sub>	Texto subscrito	O elemento HTML <sub> expressa o texto subscrito. O texto subscrito aparece meio caractere abaixo da linha normal e às vezes é renderizado em uma fonte menor. O texto subscrito	Texto Su$_{b}$scripto

		pode ser usado para Química, como H2O.	
<sup>	Texto sobrescripto	O elemento HTML <sup> especifica o texto sobrescrito. O texto sobrescrito aparece meio caractere acima da linha normal e às vezes é renderizado em uma fonte menor. O texto sobrescrito pode ser usado para notas de rodapé, como WWW [1]:	Texto suᵖerscripto

Formatação de texto HTML

HTML <blockquote> para Cotações

O elemento HTML <blockquote> identifica uma secção que é citada de outra fonte. Os navegadores normalmente recuam elementos <blockquote>, como pode ser visto abaixo:

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

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

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

</body>
</html>
```

Browsers typically indent blockquote elements.

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

Figura 25 *– O elemento Blockquote (Fonte: Autor)*

HTML <q> para pequenas Cotações

A tag HTML <q> especifica uma citação curta. Os navegadores geralmente inserem aspas ao redor da cotação, como segue:

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

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

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

</body>
</html>
```

Browsers usually insert quotation marks around the q element.

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

Figura 26 *– O elemento de pequenas cotações (Fonte: https://www.w3schools.com/html/html_quotation_elements.asp)*

HTML <abbr> para abreviações

A tag HTML <abbr> especifica uma abreviação ou um acrónimo, como "HTML", "CSS", "Mr.", "Dr.", "ASAP", etc. Abreviações de marcação podem fornecer informações valiosas para navegadores, sistemas de tradução e motores de busca, como visto anteriormente. Caso não se conheça o significado de alguma abreviatura, pode-se usar o atributo global title para mostrar a descrição da abreviatura/sigla ao passar o rato sobre o elemento, como visto abaixo:

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

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

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

</body>
</html>
```

The WHO was founded in 1948.

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

Figura 27 *– O elemento <abbr> e a função mouse over (Fonte: Autor)*

HTML <address> para Informação de Contacto

A tag HTML <address> define as informações de contato do autor/proprietário de um documento ou artigo. Pode ser um endereço de e-mail, URL, endereço físico, número de telefone, etc. O texto contido no elemento <address> normalmente é apresentado em itálico e os navegadores adicionarão sempre uma quebra de linha antes e depois dele, como segue:

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

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

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

</body>
</html>
```

Please contact us:.

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

***Figura 28** – O elemento <address> (Fonte: Autor)*

HTML <cite> para Título de Trabalho

A tag HTML <cite> define o título de um livro, um poema, uma música, um filme, uma pintura e todos os trabalhos criativos. Deve-se afirmar que o nome do autor não é o título de uma obra.
Como nas tags acima, o texto no elemento <cite> normalmente é renderizado em itálico:

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

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

</body>
</html>
```

The Scream by Edvard Munch, 1893.

Figura 29 *– O elemento <cite> (Fonte: https://www.w3schools.com/html/html_quotation_elements.asp)*

HTML <bdo> for Substituição Bidirecional

A tag HTML <bdo> significa "substituição bidirecional", que é usada para substituir a direção do texto atual/padrão. Essa tag define a direção do conteúdo dentro dela para executar no navegador da esquerda para a direita ou da direita para a esquerda (rtl – direita para esquerda; ltf – esquerda para direita).

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

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

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

</body>
</html>
```

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

PS4edoC

Figura 30 *– O elemento <bdo> (Fonte: https://www.w3schools.com/html/html_quotation_elements.asp)*

Comentários HTML

Adicionar Comentários

Este elemento é usado para adicionar um comentário a um documento HTML. Um comentário HTML começa com <!— e termina com —>. Os comentários HTML são visíveis para qualquer pessoa que visualize o código-fonte da página, mas não são renderizados quando o documento HTML é renderizado por um navegador. Deve-se notar que há um ponto de exclamação na tag inicial, mas não na tag final. Este recurso é especialmente útil para colocar notificações e lembretes no código HTML:

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

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

</body>
</html>
```

Code4SP project.

Figura 31 *– A funcionalidade dos comentários (Fonte: Autor)*

Esconder conteúdo

Os comentários também podem ser usados para ocultar o conteúdo, e isso pode ser útil se alguém o ocultar no momento. Pode também ocultar mais do que uma linha, tudo entre <!-- e --> será ocultado do ecrã. Os comentários também são ótimos para depurar HTML, porque é possível comentar linhas de código HTML, uma de cada vez, para procurar erros.

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

<p>Code4SP project.</p>

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

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

</body>
</html>
```

Code4SP project.

But this will appear.

Figura 32 *– A funcionalidade de esconder conteúdo (Fonte: Autor)*

Cores HTML

As cores HTML são estipuladas com nomes predefinidos de cores ou com RGB, HEX, HSL, RGBA, ou valores HSLA.

Figura 33 *– Alguns nomes de cores que se podem definir. (Fonte: https://www.w3schools.com/html/html_colors.asp)*

As cores podem ser definidas para **o fundo de página**:

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

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

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

</p>

</body>
</html>
```

Code4SP

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

Figura 34 *– Definir a cor de fundo de uma Webpage. (Fonte: Autor)*

O mesmo princípio pode ser aplicado à cor de texto:

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

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

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

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

</body>
</html>
```

Code4SP

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

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

Figura 35 *– Definir a cor de texto (Fonte: Autor)*

Também para as cores da **borda**:

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

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

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

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

</body>
</html>
```

Code4SP

Code4SP

Code4SP

Figura 36 *– Adicionar bordas e definir a sua cor (Fonte: Autor)*

Valores de cor

Conforme mencionado acima, as cores HTML são estipuladas com nomes de cores predefinidos ou com valores RGB, HEX, HSL, RGBA ou HSLA. Os três elementos a seguir têm sua cor de fundo definida com valores RGB, HEX e HSL:

rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

Figura 37 *– Valores RGB, HEX e HSL para a cor Tomato (Fonte: https://www.w3schools.com/html/html_colors.asp)*

A transparência também é um recurso que pode ser adicionado ao definir uma cor, adicionando-lhe um canal Alfa. O exemplo a seguir como 50% de transparência:

rgba(255, 99, 71, 0.5)

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

Figura 38 *– Definir valores de transparência para a cor Tomato (Fonte: https://www.w3schools.com/html/html_colors.asp)*

O código para configurar os dois recursos facilitará a compreensão dos alunos. Como visto, ele compreende recursos CSS a serem explorados posteriormente:

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

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

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

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

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

</body>
</html>
```

Same as color name "Tomato":

rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

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

rgba(255, 99, 71, 0.5)

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

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

Figura 39 *– Definir cor e transparência da cor Tomato (Fonte: https://www.w3schools.com/html/html_colors.asp)*

Mais informações em relação a valores RGB, HEX, HSL, RGBA ou HSLA pode ser encontrada em: https://www.w3schools.com/html/html_colors.asp

Ligações HTML

As ligações podem ser encontradas em quase todas as páginas da web. Elas permitem que os utilizadores da Internet naveguem de página em página. Não precisa ser texto, pode ser uma imagem ou qualquer outro elemento HTML.

Hyperlinks

Hyperlinks HTML são ligações que podem ser clicadas, direcionando para outro documento.
Quando o rato é movido sobre uma ligação, a seta do rato transforma-se numa pequena mão.

Sintaxe

A tag HTML <a> define um hyperlink. Tem a seguinte sintaxe:

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

O atributo mais significativo do elemento <a> é o atributo href, que indica o destino do link. O texto do link é a parte que ficará visível para o utilizador. Ao clicar no texto do link, o leitor será redirecionado para o endereço URL necessário.

Por padrão, os links serão vistos em todos os navegadores da seguinte forma:
• Um link não visitado é sublinhado e azul
• Um link visitado é sublinhado e roxo
• Um link ativo é sublinhado e vermelho

Nota: As cores dos links podem ser alteradas ao usar funcionalidades do CSS.

O atributo de destino

Por padrão, a página vinculada será mostrada na janela atual do navegador. Para modificar isso, os alunos devem indicar outro destino para o link.

O atributo de destino indica onde abrir o documento vinculado. Pode ter um dos seguintes valores:

• _self - Padrão. Abre o documento na mesma janela/guia em que foi clicado
• _blank - Abre o documento numa nova janela ou guia
• _parent - Abre o documento no parent frame
• _top - Abre o documento em todo o corpo da janela

Usar imagem como ligação

Para usar uma imagem como link, basta colocar a tag <img> dentro da tag <a>, como pode ser verificado no tutorial a seguir:
https://www.w3schools.com/html/tryit.asp?filename=tryhtml_links_image

Ligar um endereço de email

Para criar uma ligação que abra o software de e-mail do utilizador (permitindo que ele envie um novo e-mail), mailto: deve ser adicionado dentro do atributo href, seguindo o próximo exemplo:

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

Criar um marcador em HTML

Ligações HTML podem ser aplicadas para criar marcadores, para que os leitores possam ir para partes específicas de uma página da Web, podendo ser realmente útil se a página Web for muito longa. Este processo é composto por duas etapas muito simples:

• Para criar um marcador - primeiro o marcador deve ser criado e, em seguida, uma ligação deve ser-lhe adicionada. Para criar, deve-se utilizar o atributo id (ex.: <h2 id="C1">Capítulo 1</h2>), em seguida, deve-se adicionar uma ligação para o favorito, dentro da mesma página (ex.: Pular para o Capítulo 4)
• Quando a ligação for clicada, a página irá para o local com o marcador.

Imagens HTML

Inserir imagens em páginas Web

As imagens melhoram a aparência visual das páginas web, tornando-as mais atraentes e coloridas. A tag <img> é usada para adicionar imagens nas páginas HTML. É um elemento vazio e contém apenas atributos.

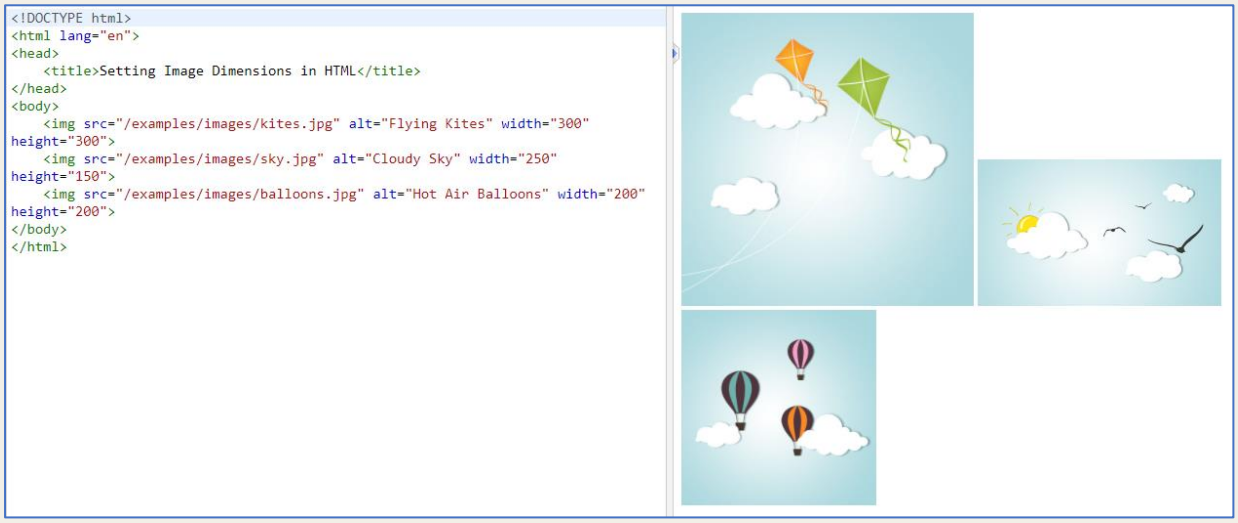
Cada imagem deve ter pelo menos dois atributos: os atributos src e alt. O atributo src informa o navegador onde encontrar a imagem, sendo o seu valor o URL do arquivo de imagem. O atributo alt fornece um texto alternativo para a imagem se ela estiver inacessível ou não puder ser exibida por algum motivo (conexão lenta, imagem não disponível na URL especificada ou se o utilizador usar um leitor de ecrã ou navegador não gráfico). O seu valor deve ser um substituto significativo para a imagem, preferencialmente um texto sugestivo.

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

Figura 40 *– Adicionar uma imagem a um documento HTML, usando os atributos src e alt (Fonte: Autor)*

Definir a Largura e Altura de uma Imagem

Os atributos largura e altura são usados para indicar a largura e a altura de uma imagem. Os valores desses atributos são interpretados em pixels por padrão. É uma boa prática especificar os atributos de largura e altura, para que o navegador possa atribuir o espaço necessário para a imagem, antes de esta ser transferida.



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

Figura 41 *– Definir a altura e a largura de uma imagem (Fonte:: https://www.tutorialrepublic.com/codelab.php?topic=html&file=specify-dimensions-for-images)*

O atributo estilo também pode ser usado para indicar largura e altura. Impede que as folhas de estilo alterem o tamanho da imagem acidentalmente, porque o estilo embutido tem a prioridade mais elevada.

Usar o elemento de imagem HTML5

De vez em quando, dimensionar uma imagem para cima ou para baixo para caber em diferentes dispositivos (ou tamanhos de ecrã) não funciona conforme o esperado. Além disso, reduzir a dimensão da imagem usando o atributo largura e altura não diminui o tamanho inicial do arquivo. Para resolver esses problemas, o HTML5 introduziu a tag <picture>, que permite definir várias versões de uma imagem para atingir diferentes tipos de dispositivos.

O elemento <picture> contém zero ou mais elementos <source>, cada um referindo-se a uma fonte de imagem diferente, e um elemento <img> no final. Da mesma forma, cada elemento <source> tem o atributo media que especifica uma condição de media que é usada pelo navegador para determinar quando uma determinada fonte deve ser usada.

Mapa de Imagem

Um mapa de imagem permite definir pontos de acesso numa imagem que funciona como um hyperlink. A ideia-chave por trás da criação de um mapa de imagem é fornecer uma maneira simples de vincular várias partes de uma imagem sem dividi-la em arquivos de imagem separados. Por exemplo, um mapa de um país pode ter cada cidade vinculada a mais informações sobre essa cidade.

O exemplo a seguir é bastante preciso sobre esses recursos:
https://www.tutorialrepublic.com/codelab.php?topic=html&file=image-maps

O atributo name da tag <map> é usado para referenciar o mapa da tag <img> usando seu atributo usemap. A tag <area> é usada dentro do elemento <map> para definir as áreas clicáveis numa imagem. Qualquer número de áreas clicáveis pode ser definido numa imagem.

Favicon HTML

Um favicon é uma pequena imagem exibida à esquerda do título da página na guia do navegador:

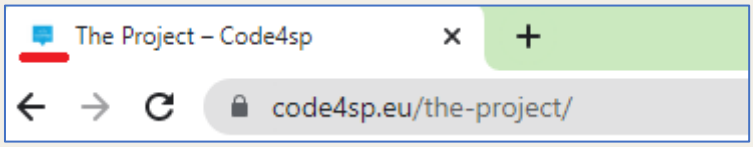


***Figura 42** – Favicon do Code4SP's (Fonte: Autor)*

Para adicionar um favicon a um site, uma imagem favicon deve ser guardada na diretória raiz do servidor web. Outra forma é criar uma pasta na diretória raiz chamada imagens e guardar a imagem do favicon nessa pasta. Um nome comum para uma imagem favicon é "favicon.ico".

Em seguida, um elemento <link> deve ser adicionado ao arquivo "index.html", após o elemento <title>, conforme segue:

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

Tabelas HTML

Criar tabelas em HTML

As tabelas HTML permitem organizar os dados em linhas e colunas. Eles geralmente são usados para exibir dados tabulares, como listagens de produtos, detalhes do cliente, relatórios financeiros, etc.

Uma tabela pode ser criada usando o elemento <table>. Dentro do elemento <table>, os elementos <tr> podem ser utilizados para criar linhas, e para criar colunas dentro de uma linha, os elementos <td> podem ser usados. Uma célula pode ser definida como um cabeçalho para um grupo de células da tabela usando o elemento <th>.

As tabelas não têm bordas por padrão. A propriedade CSS border pode ser usada para adicionar bordas às tabelas. Além disso, as células da tabela são dimensionadas apenas o suficiente para caber o conteúdo por padrão. Para adicionar mais espaço ao redor do conteúdo nas células da tabela, a propriedade de preenchimento CSS pode ser usada.

Por padrão, as bordas ao redor da tabela e suas células são separadas umas das outras. Mas eles podem ser reduzidos numa só, usando a propriedade border-collapse no elemento <table>. Além disso, o texto dentro dos elementos <th> é exibido em negrito, alinhado horizontalmente no centro da célula por padrão. Para alterar o alinhamento padrão, a propriedade CSS text-align pode ser usada. Para isso, sugere-se que os alunos atinjam primeiro o tópico CSS. A maioria dos atributos do elemento <table>, como border, cellpadding, cellspacing, width, align, etc. para estilizar aparências de tabelas em versões anteriores foram descartados no HTML5, portanto, devem ser evitados. CSS deve ser **privilegiado para estilizar tabelas HTML.**

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

Spotify Top Songs of 2021 (USA)

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

Figura 43 *– Uma tabela básica (Fonte: Autor)*

Abranger várias linhas e colunas

A abrangência permite estender linhas e colunas da tabela em várias outras linhas e colunas. Normalmente, uma célula da tabela não pode passar para o espaço abaixo ou acima de outra célula da tabela. No entanto, os atributos rowspan ou colspan podem ser usados para abranger várias linhas ou colunas numa tabela.

Da mesma forma, o atributo rowspan pode ser usado para criar uma célula que abrange mais de uma linha, como segue:

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

Spanning Rows

Name:	John Carter
Phone:	55577854
	55577855

Figura 44 *– O atributo rowspan (Fonte: https://www.tutorialrepublic.com/html-tutorial/html-tables.php)*

Legendas da tabela

Uma legenda (ou título) para tabelas pode ser criada usando o elemento <caption>. Este elemento deve ser colocado diretamente após a tag (de abertura) <table>. Por padrão, a legenda aparece na parte superior da tabela, mas isso pode ser alterado usando a propriedade CSS caption-side.

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

WIKITECHY WEBSITE

Firstname	Lastname
Wiki	techy

Figura 45 *– Adicionar legendas de tabela (Fonte: https://www.wikitechy.com)*

Definir um cabeçalho, corpo e rodapé de tabela

HTML fornece uma série de tags <thead>, <tbody> e <tfoot> que ajudam os alunos a criar tabelas mais coordenadas, definindo regiões de cabeçalho, corpo e rodapé, nessa ordem.

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

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

Figura 46 *– Adicionar legendas de tabela (Fonte: https://www.tutorialrepublic.com/html-tutorial/html-tables.php)*

Listas HTML

As listas HTML são aplicadas para apresentar informações de forma bem formada. Dentro delas, pode-se adicionar texto, imagens, ligações, quebras de linha, etc. Existem três tipos diferentes de listas em HTML e cada uma tem um propósito e significado específico:

• **A) Listas não ordenadas** — Usadas para criar uma lista de itens relacionados, sem ordem específica.
• **B) Listas ordenadas** — Usadas para criar uma lista de itens relacionados, em uma determinada ordem.
• **C) Listas de descrição** — Usadas para criar uma lista de termos e suas descrições.

A) Listas não ordenadas

Uma lista não ordenada é criada usando o elemento <ul>, e cada item da lista começa com o elemento <li>. Está marcado com pontos, como segue:

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

Reasons why you should travel by train

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

***Figura 47** – Listas não ordenadas (Fonte: Autor)*

A) Listas ordenadas

Uma lista ordenada é criada usando o elemento <ol> e cada item da lista começa com o elemento <li>. As listas ordenadas são usadas quando a ordem dos itens da lista é crítica. É marcado com números, como segue:

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

How to cook your own veggie burger

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

__Figura 48__ – Listas ordenadas (Fonte: Autor)

B) Listas de descrição

Uma lista de descrição é uma lista de itens com uma descrição ou definição de cada item.

A lista de descrição é criada usando o elemento <dl>. O elemento <dl> é usado em conjunto com o elemento <dt> que especifica um termo e o elemento <dd>, que especifica a definição do termo.

Os navegadores geralmente renderizam as listas de definições colocando os termos e as definições em linhas separadas, onde as definições dos termos são levemente recuadas.

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

What are bread and coffee?

Bread
: A baked food made of flour.

Coffee
: A drink made from roasted coffee beans.

__Figura 49__ – Listas de descrição (Fonte: Autor)

Formulários HTML

Espera-se que os formulários HTML coletem diferentes tipos de entradas do utilizador, como detalhes de contacto (nome, e-mail, número de telefone, conta bancária, etc.). Os formulários incluem elementos especiais conhecidos como controles, por exemplo caixa de entrada, caixas de seleção, botões de opção, botões de envio, etc. esses dados.

A tag <form> é usada para gerar um formulário HTML. Um exemplo simples de um formulário de login seria o seguinte:


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

***Figura 50** – Exemplo de um formulário de login (Fonte: https://www.tutorialrepublic.com/html-tutorial/html-forms.php)*

Existem diferentes tipos de controles a serem aplicados num formulário HTML, sendo os elementos de entrada os mais utilizados. Eles identificam vários tipos de campos de entrada do utilizador, dependendo do atributo de tipo. Os elementos de entrada podem ser campos de texto, campos de senha, caixas de seleção, botões de envio, botões de reinicialização, caixas de seleção de arquivos, bem como vários novos tipos de entrada introduzidos no HTML5 (eles podem ser verificados aqui).

Campos de texto são áreas que permitem aos utilizadores adicionar texto. Eles são criados usando um elemento <input>, cujo atributo type tem um valor de texto. Deve-se notar que a tag <label> é usada para identificar os rótulos dos elementos <input>. Se o webmaster quiser que os seus utilizadores insiram várias linhas, <textarea> deve ser adicionado.

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

Login:

Figura 51 *– Campo de texto (Fonte: Autor)*

Os campos de senha são como campos de texto, sendo que os únicos caracteres de diferença em um campo de senha são ocultos, portanto, são exibidos como asteriscos ou pontos. Da mesma forma, esse procedimento impede que outra pessoa leia a senha na tela. Este também é um controle de entrada de texto de linha única gerado usando um elemento <input> cujo atributo type tem um valor de senha.

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

Password:

Figura 52 *– Exemplo de campo de password (Fonte: https://www.tutorialrepublic.com/html-tutorial/html-forms.php)*

Os botões de opção são aplicados para permitir que o utilizador selecione exatamente uma opção de um conjunto pré-especificado de opções. Ele é gerado usando um elemento <input> cujo atributo type tem um valor de radio.

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

Single Married Other

Figura 53 *– Código para configurar botões de rádio (Fonte: Autor)*

As **caixas de seleção** disponibilizam ao usuário uma ou mais opções de um conjunto predefinido de opções. Ele é gerado usando um elemento <input> cujo atributo type tem um valor de checkbox. Se preferir gerar uma caixa de seleção (ou botão de rádio) selecionada por padrão, basta adicionar o atributo marcado ao elemento de entrada (<input type="checkbox" checked>).

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

Football Handball Basketball

Figura 54 *– Código para definir checkboxes (Fonte: Autor)*

As caixas de seleção de arquivo permitem que um utilizador procure um arquivo local e o envie como um anexo com os dados do formulário. Por exemplo, o Google Chrome fornece um campo de entrada de seleção de arquivo com um botão "Procurar" que permite ao usuário selecionar um arquivo de seu disco rígido.

As caixas de seleção de arquivo também são criadas usando um elemento <input>, cujo valor do atributo type é definido como file.

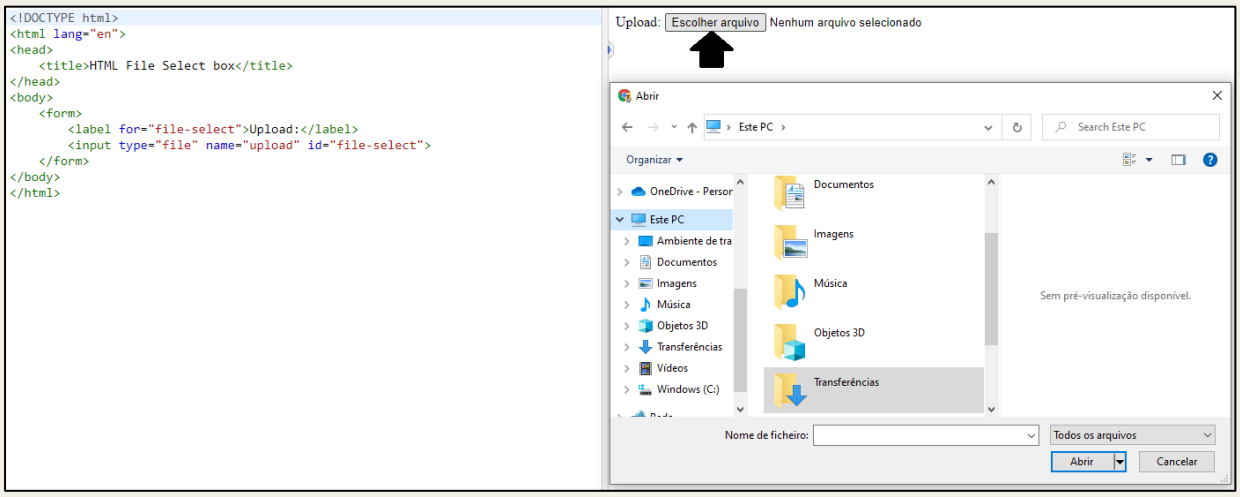


Figura 55 *– Código para configurar caixas de seleção de arquivos (Fonte: Autor)*

Textarea pode ser definida como um controle de entrada de texto de várias linhas, que permite inserir mais de uma linha de texto. Esses controles são criados usando um elemento <textarea>, como segue:

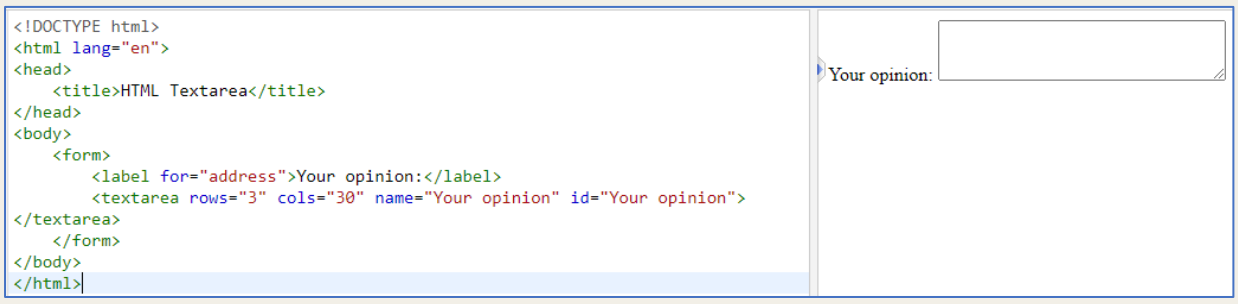


Figura 56 *– Código para configurar um controle de entrada de texto de várias linhas (Fonte: Autor)*

As caixas de seleção são listas suspensas de opções nas quais um usuário pode selecionar uma ou mais opções num menu suspenso. Eles são criados usando o elemento <select> e o elemento <option>. Os elementos <option> dentro do elemento <select> definem cada item da lista.

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

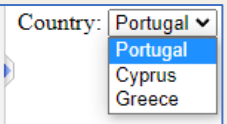


Figura 57 *– Código para configurar caixas de seleção (Fonte: Autor)*

Os botões Enviar e Redefinir são muito comuns na maioria dos sites. Os botões de envio são usados para enviar (formulário) dados para um servidor da Web, enquanto os botões de redefinição são criados para redefinir o formulário para os valores padrão. Quando o usuário clica no botão enviar, os dados do formulário são enviados para o arquivo especificado no atributo de ação do formulário para processar os dados enviados.

Os botões também podem ser criados usando o elemento <button>. Eles têm a mesma finalidade dos botões criados com o elemento input. No entanto, eles oferecem mais possibilidades de renderização, pois permitem a incorporação de outros elementos HTML.

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

Figura 58 *– Código para configurar os botões de envio e redefinição (Fonte: Autor)*

Agrupar controles de formulário é uma ótima ferramenta para os usuários localizarem um controle, tornando o formulário mais acessível. O elemento <legend> é fundamental para criar controles logicamente relacionados, como visto na figura abaixo:

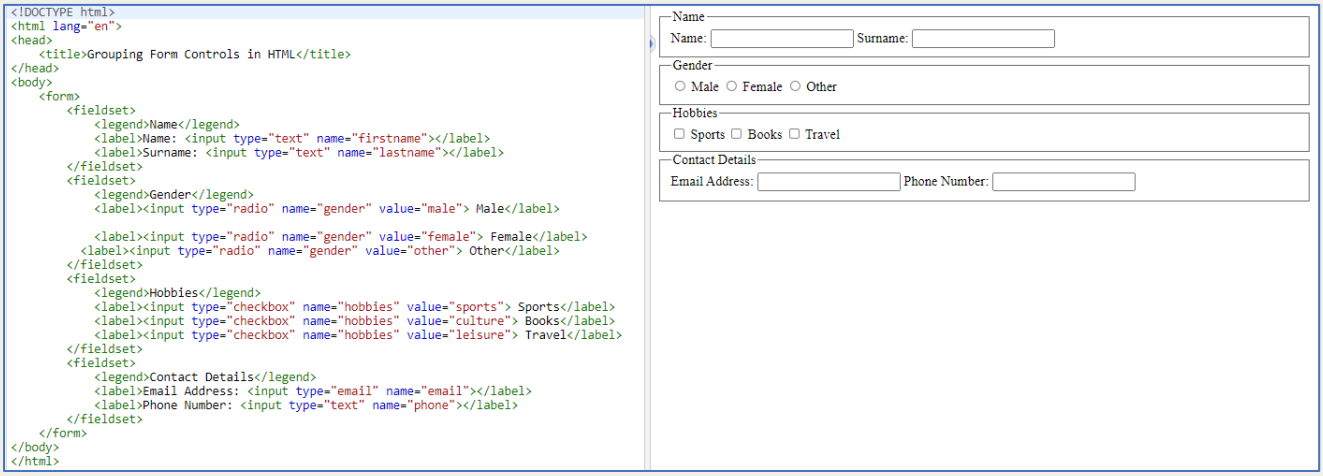


***Figura 59** – Código para configurar controles de formulário de agrupamento (Fonte: Autor, check this for better definition)*

Abaixo tem uma lista de atributos de elementos de formulário usados com frequência:

Atributo	Descrição
`name`	Especifica o nome do formulário.
`action`	Especifica o URL do programa ou script no servidor web que será usado para processar as informações enviadas via formulário.
`method`	Especifica o método HTTP usado para enviar os dados ao servidor web pelo navegador. O valor pode ser get (o padrão) e post.
`target`	Especifica onde exibir a resposta recebida após o envio do formulário. Os valores possíveis são _blank, _self, _parent e _top.
`enctype`	Especifica como os dados do formulário devem ser codificados ao enviar o formulário ao servidor. Aplicável somente quando o valor do atributo method for post.

***Tabela 2** – Lista de atributos de elementos de formulário usados com frequência (Fonte: https://www.tutorialrepublic.com/html-tutorial/html-forms.php)*

Mais informações sobre outros atributos podem ser encontradas aqui.

HTML iFrame

Um iframe (ou *frame embutido*) é usado para exibir objetos externos dentro de uma página da web, incluindo outras páginas da web. Um iframe funciona como um mini navegador da web dentro de um navegador da web. Da mesma forma, o conteúdo dentro de um iframe ocorre separado dos elementos adjacentes.

A sintaxe básica para adicionar este recurso a uma página da web é a seguinte:

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

A URL especificada no atributo src indica a localização de um objeto externo ou de uma página da web.

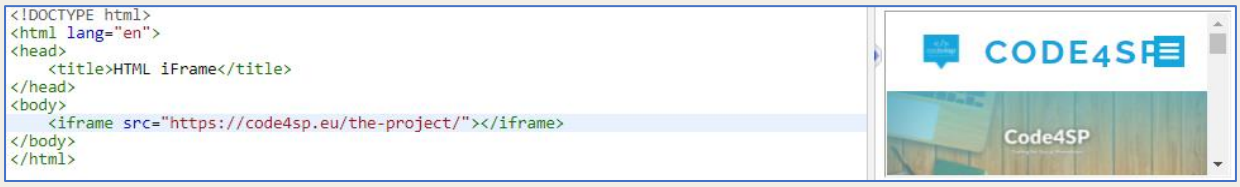


Figura 60 *– Código para configurar um frame embutido (Fonte: Autor)*

A **altura** e a **largura** do iframe podem ser definidas aplicando o código disposto na Figura 61 abaixo. Os valores dos atributos largura e altura são estipulados em pixels por padrão, mas os utilizadores também podem definir esses valores em percentagem, como 50%, 100%, etc. A largura padrão de um iframe é 300 pixels, enquanto a altura padrão é 150 pixels.

Figura 61 *– Código para configurar a altura e a largura de um quadro embutido (Fonte: Autor)*

Como pode ser verificado, o iframe tem uma borda ao contorno dele definida por padrão. Para modificá-lo ou removê-lo, a melhor maneira é usar o recurso de borda CSS (a ser ensinado no tópico CSS).

Um iframe também pode ser usado como destino para os hyperlinks. Ele pode ser nomeado usando o atributo name. Isso significa que quando uma ligação com um atributo de destino (com esse nome definido como valor) é clicada, a fonte vinculada será aberta no mesmo quadro inline, como segue:

Figura 62 *– Usando iframe como destino para um hiperlink (Fonte: Autor)*

2.2. Conceitos avançados de HTML

Doctypes HTML

Uma Declaração de Tipo de Documento (DOCTYPE) é uma instrução para o navegador da Web sobre a versão da linguagem de marcação na qual uma página da Web é criada. Ele aparece no topo de uma página da web antes de todos os outros elementos. De acordo com a especificação HTML, todos os documentos HTML requerem uma declaração de tipo de documento válida para garantir que as páginas da Web sejam exibidas da maneira que devem ser. A declaração doctype é frequentemente a primeira coisa definida num documento HTML (mesmo antes da tag de abertura <html>); no entanto, a declaração de doctype em si não é uma tag HTML.

O DOCTYPE para HTML5 é muito curto, conciso e não diferencia maiúsculas de minúsculas: <!DOCTYPE html>.

A seguinte síntaxe pode ser usada como modelo de criação para um novo documento HTML5:

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

Layouts HTML

Existem diferentes métodos para criar um layout de página web, posicionando os diversos elementos que compõem uma página web de forma bem estruturada e dando uma aparência envolvente ao site. A maioria dos sites geralmente exibe o seu conteúdo em várias linhas e colunas, configuradas como uma revista para fornecer aos utilizadores uma melhor atmosfera de leitura e escrita. Isso pode ser facilmente alcançado utilizando as tags HTML, como <table>, , <header>, <footer>, <section>, etc. e combinando alguns estilos CSS a elas.

A maneira mais simples de criar layouts em HTML é de fato obtida projetando tabelas. Como visto nas seções anteriores, isso geralmente envolve o processo de colocar conteúdo (texto, imagens, etc.) em linhas e colunas.

O layout abaixo contém uma tabela HTML com três linhas e duas colunas. Deve-se observar que a primeira e a última linha abrangem o atributo colspan. Deve-se afirmar que o método utilizado para a criação do layout neste exemplo, embora não esteja errado, não é recomendado. Tabelas e estilos embutidos para criar layouts devem ser evitados. Layouts criados usando tabelas são renderizados muito lentamente. **As tabelas só devem ser usadas para exibir dados tabulares**. Para criar tais layouts, **as técnicas de flutuação CSS** são recomendadas. Os usuários devem aprender sobre esta ferramenta além disso.


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

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

Figura 63 *– Layout básico de HTML (Fonte: Autor, adaptado do website do Tutorial Republic. Clique aqui para obter uma imagem com melhor resolução)*

O HTML5 estabeleceu novos elementos estruturais, por exemplo <header>, <footer>, <nav>, <section>, etc. para identificar as diferentes partes de uma página web de uma forma mais semântica. Este exemplo aplica os novos elementos estruturais HTML5, para criar o mesmo layout criado na *Figura 63*.

Para saber mais sobre as tags recém-introduzidas, os alunos devem visitar esta fonte.

Cabeçalho HTML

O elemento head é o recetáculo para todos os elementos head que fornecem informações extras sobre o documento ou referência a outros recursos necessários para que o documento funcione corretamente. Ele ilustra as propriedades do documento, como título, entrega meta-informações como conjunto de caracteres, informa ao navegador onde encontrar as folhas de estilo ou scripts que permitem expandir o documento HTML de forma interativa.

Os elementos HTML que podem ser usados dentro do elemento <head> são: <title>, <base>, <link>, <style>, <meta>, <script> e <noscript>.

The HTML title Element

O elemento <title> identifica o título do documento e apenas um é necessário em todos os documentos HTML/XHTML para produzir um documento válido. Deve ser colocado dentro do elemento <head>. O elemento title compreende texto simples e entidades e não pode incluir outras tags de marcação. Pode ser usado para diferentes funções:

• Para mostrar um título na barra de título do navegador e na barra de tarefas;

• Para entregar um título para a página quando ela é adicionada aos favoritos ou marcada;

• Para apresentar um título para a página nos resultados do mecanismo de pesquisa (por exemplo, pesquisa do Google).

Deve ser curto e específico para o conteúdo do documento, pois os motores de busca devem prestar atenção especial às palavras presentes no título – idealmente, deve ter 65 caracteres.

Um título em um documento HTML deve ser exibido da seguinte forma:

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

The HTML base Element

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

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

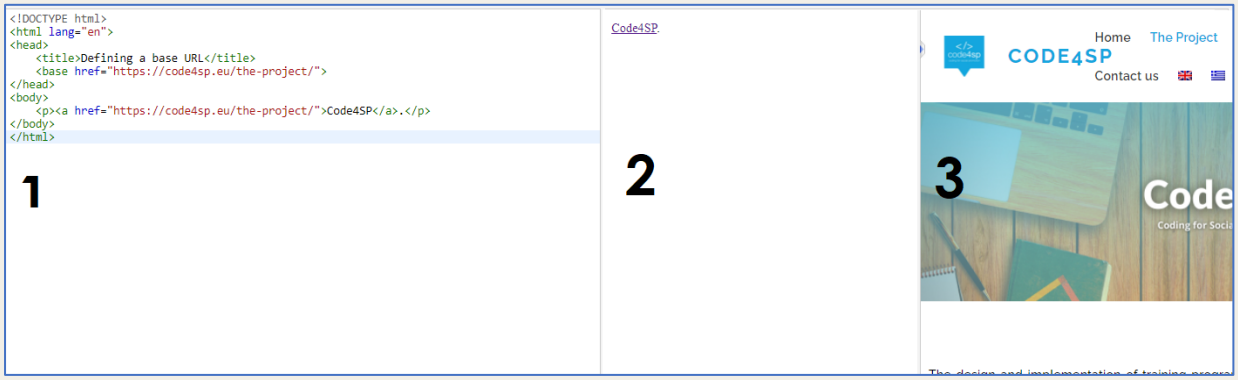
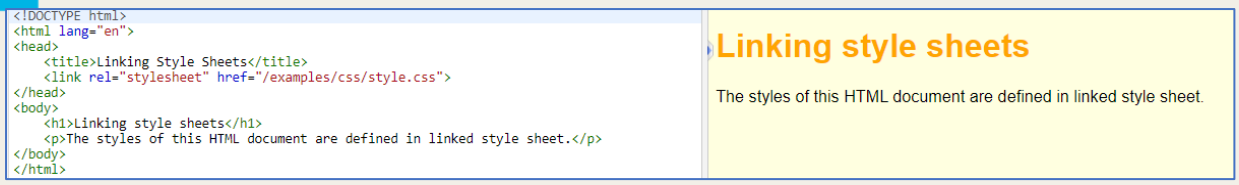


Figura 64 *– Elemento básico HTML (Fonte: Autor)*

O elemento HTML <base> deve ser localizado antes de qualquer elemento que pertença a um recurso externo. HTML permite apenas um elemento base para cada documento.

Elemento de ligação HTML

O elemento <link> descreve a correlação entre o documento atual e um documento ou recurso externo. Um uso comum do elemento link é conectar-se a folhas de estilo externas.




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

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

Deve ser declarado que o elemento <head> de um documento HTML pode incluir qualquer número de elementos <link>. O elemento <link> tem atributos, mas nenhum conteúdo.

O Elemento Estilo de HTML

O elemento <style> é aplicado para descrever informações de estilo incorporadas para um documento HTML. As regras de estilo dentro do elemento <style> indicam como os elementos HTML devem ser renderizados num navegador. Uma folha de estilo incorporada deve ser usada quando um único documento tem um estilo único. Se a mesma folha de estilo for usada em vários documentos, uma folha de estilo externa seria mais adequada.

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

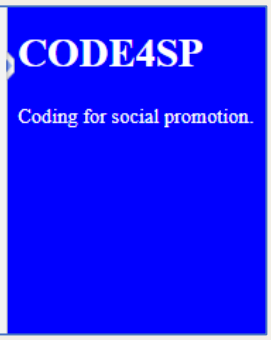


Figura 66 *– Codificar vários elementos de estilo (Fonte: Autor)*

O Elemento Meta de HTML

O elemento <meta> fornece metadados sobre o documento HTML, que é um conjunto de dados que descreve e fornece informações sobre outros dados. As tags <meta>

aparecem sempre dentro do elemento <head> e normalmente são usadas para especificar o conjunto de caracteres, descrição da página, palavras-chave, autor do documento e configurações da janela de visualização.

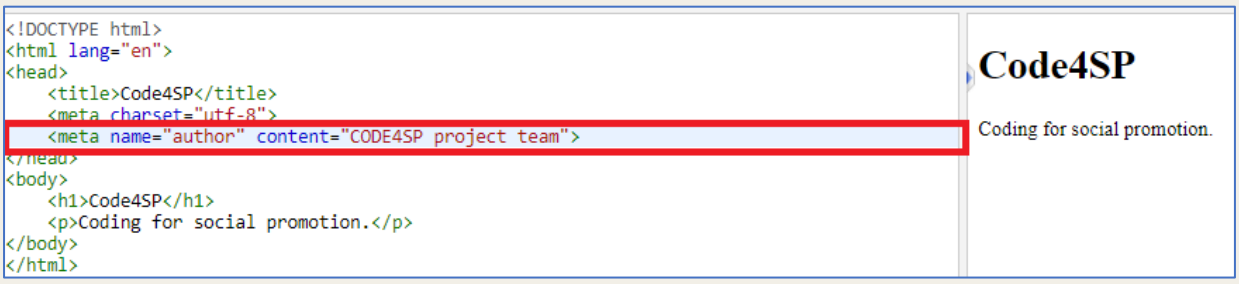


***Figura 67** – O elemento meta (Fonte: Autor)*

Como visto acima, as metatags incluem informações sobre uma página da web. Não é visível no navegador (embora seja analisável por máquina). Os metadados são utilizados por navegadores (como exibir conteúdo ou recarregar a página), mecanismos de pesquisa (palavras-chave) e outros serviços da web. Além disso, existe uma técnica para permitir que os web designers governem a **viewport**, por meio da tag <meta>. A janela de visualização é a área visível do usuário de uma página da web. Ele difere de dispositivo para dispositivo (será maior num ecrã de computador do que num telemóvel).

O elemento <meta> a seguir deve ser incluído em todas as páginas da web, pois dará ao navegador diretrizes sobre como assumir as dimensões e o dimensionamento da página: `<meta name="viewport" content="width=device-width, initial-scale=1.0">`

A parte **width=device-width** define a largura da página para seguir a largura da tela do dispositivo (que varia de acordo com a tela). A parte **initial-scale=1.0** define o nível de zoom inicial quando a página é carregada inicialmente pelo navegador.

A diferença entre páginas da web com ou sem meta tag viewport é bem visível no exemplo a seguir:

Figura 68 *– Exemplos de metatags com ou sem viewport (Fonte: https://www.w3schools.com/tags/tag_meta.asp)*

O elemento script HTML

O elemento <script> é usado para definir o script do lado do cliente, como JavaScript em documentos HTML.

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

Hello CODE4SP learners!

The above heading is inserted in this document by JavaScript.

Figura 69 *– O elemento script (Fonte: Autor)*

Trabalhar com script do lado do cliente

O script do lado do cliente está vinculado ao tipo de programas de computador executados pelo navegador da Web do utilizador. **JavaScript (JS)** é a linguagem de script do lado do cliente mais difundida na web. O elemento <script> é usado para incorporar ou referenciar JavaScript num documento HTML para adicionar interatividade a páginas da Web e criar uma experiência mais amigável. Alguns dos

usos mais comuns do JavaScript são validação de formulários, geração de mensagens de alerta, criação de galeria de imagens, exibição de conteúdo oculto, manipulação de DOM, etc.

JavaScript pode ser incorporado de duas formas:
1. Diretamente dentro da página HTML;
2. Colocado num arquivo de script externo e **referenciado** dentro da página HTML.
Nota: ambas as técnicas usam o elemento <script>.

Para embutir JS num arquivo HTML, o usuário deve adicionar o código como conteúdo do elemento <script>, seguindo o exemplo da Figura 70. De preferência, os elementos script devem ser colocados no final da página, antes da tag de fecho do corpo (</body>), pois quando o navegador encontra um script, ele pausa a renderização do restante da página até desconstruir o script que pode impactar drasticamente o desempenho do site.

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

Figura 70 *– Incorporando JS em um documento HTML (Fonte: Autor)*

Os códigos JavaScript também podem ser colocados num arquivo separado, chamando uma extensão .js, enquanto a corresponde no documento HTML usando o atributo src da tag <script>. Isso pode ser especialmente útil se o web designer quiser

disponibilizar o mesmo script para vários documentos, para que ele não precise executar as mesmas tarefas repetidamente. Quando o atributo src é indicado, o elemento <script> deve estar vazio, para que o web designer não possa usar o mesmo elemento <script> para incorporar o JavaScript e vincular a um arquivo JavaScript externo num documento HTML.

Se um navegador, por algum motivo, não suportar scripts do lado do cliente ou os utilizadores tiverem desabilitado o JS em seus navegadores, o elemento <no script> poderá ser usado para fornecer um conteúdo alternativo. Este elemento pode incluir qualquer elemento HTML, exceto <script>, pois pode ser incluído no elemento <body> de uma página HTML.

Entidades HTML

A maioria dos alunos certamente ficará curiosa sobre como exibir caracteres e símbolos especiais em seus processos de programação. Assim, este subcapítulo pretende explicar como eles podem ser bem sucedidos em fazer tal coisa.

Primeiro, é importante entender o que é uma entidade HTML. Conforme percebido nos capítulos anteriores, alguns caracteres são bastante reservados em HTML. Por exemplo, não se pode usar sinais como '<' ou '>', pois o navegador pode confundi-los com uma marcação. Além disso, alguns caracteres simplesmente não estão disponíveis no teclado (por exemplo, o símbolo de copyright).

Esses caracteres especiais podem ser exibidos simplesmente sendo substituídos pelas entidades de caracteres (ou apenas entidades), resolvendo os problemas mencionados acima. Abaixo está uma lista das entidades mais usadas em HTML:

Resultado	Descrição	Nome da entidade	Referência numérica
	Espaço sem quebra		
<	Menos do que	&lt;	&#60;
>	Mais do que	&gt;	&#62;
&	ampersand	&amp;	&#38;
"	marca citação	&quot;	&#34;
'	apóstrofe	&apos;	&#39;
¢	cêntimo	&cent;	&#162;

£	pound	&pound;	&#163;
¥	yen	&yen;	&#165;
€	euro	&euro;	&#8364;
©	copyright	&copy;	&#169;
®	marca registada	&reg;	&#174;
™	trademark	&trade;	&#8482;

***Table 3** – As entidades mais frequentes em HTML (Fonte: https://www.tutorialrepublic.com/html-tutorial/html-entities.php)*

Referências de caracteres numéricos também podem ser usadas como alternativa para nomes de entidades, especialmente porque têm suporte mais forte do navegador e podem ser usadas para especificar qualquer caracter Unicode, porém as entidades são limitadas a um subgrupo deste.

URL HTML

URL: quantas vezes essa abreviação apareceu em ambientes virtuais? Significa Uniform Resource Locator e funciona como o endereço global de documentos e outros recursos na World Wide Web. O seu objetivo é identificar a localização de um documento e outros recursos disponíveis online, especificando o mecanismo para lhe aceder usando um navegador da web. Por exemplo, https://code4sp.eu/ é um URL.

A sintaxe geral de URLs pode ser descrita da seguinte forma: `scheme://host:port/path?query-string#fragment-id`

Os URLs têm uma estrutura linear e são compostos pelos seguintes componentes:

- **Nome do esquema** — O esquema reconhece o protocolo a ser usado para aceder a um recurso na Internet. Os nomes dos esquemas são seguidos pelos três caracteres :// (dois pontos e duas barras). Os protocolos mais usados são http://, https://, ftp:// e mailto://.
- **Nome de anfitrião** — identifica o host onde o recurso está situado. Um nome de host é um nome de domínio dado a um computador host. Isso geralmente é uma combinação do nome local do host com o nome do parente domain. Por exemplo, www.code4sp.eu/ consiste no nome da máquina do host www e no nome de domínio code4sp.eu.
- **Número do post** — Os servidores geralmente entregam mais do que um tipo de serviço, portanto, eles devem ser informados sobre qual serviço está sendo solicitado. Essas solicitações são feitas pelo número da porta. Normalmente, os números de porta conhecidos de um serviço são omitidos da URL. Por exemplo, HTTP de serviço da web é executado por padrão na porta 80, HTTPS é executado por padrão na porta 443.
- **Caminho** — identifica o recurso específico dentro do host que o utilizador deseja aceder. Por exemplo, /html/html-url.php, /news/technology/, etc.
- **String de consulta** — contém dados a serem passados para scripts do lado do servidor, executados no servidor web. Por exemplo, parâmetros para uma pesquisa. A string de consulta precedida por um ponto de interrogação (?), geralmente é uma string de pares de nome e valor separados por e comercial (&), por exemplo, ?first_name=John&last_name=Corner, q=mobile+phone, e por aí fora.
- **Identificador de fragmentos** — especifica um local dentro da página. O navegador move-se para exibir essa parte da página. O identificador de fragmento introduzido por um caracter hash (#) é a última parte opcional de um URL para um documento.

Codificação do URL HTML

Sabe-se que às vezes os dados não são transmitidos com segurança pela Internet. Isso acontece principalmente porque os URLs não são codificados de forma completa ou precisa e causa alguns mal-entendidos entre os utilizadores da Internet.

De acordo com a RFC 3986, os caracteres numa URL restringem-se apenas a um conjunto definido de caracteres US-ASCII reservados e não reservados. Quaisquer outros caracteres não são permitidos num URL (é por isso que alguns caracteres latinos e cirílicos não são vistos no URL). Mas o URL geralmente inclui caracteres fora do conjunto de caracteres US-ASCII, portanto, eles devem ser convertidos num formato US-ASCII válido para interoperabilidade mundial. A codificação de URL é um processo de conversão de informações de URL para que possam ser transmitidas com segurança pela Internet.

Para mapear a grande variedade de caracteres usados globalmente, um método de duas etapas é seguido:

- Em primeiro lugar, os dados são codificados em relação à codificação de caracteres UTF-8.
- Então, apenas os bytes que não correspondem aos caracteres no conjunto não reservado devem ser codificados por percentagem como %HH, onde HH é o valor hexadecimal do byte.

Por exemplo, veja esta popular frase portuguesa:

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

Esta frase seria codificada como:

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

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

Certos caracteres são restritos ao uso numa URL, pois podem (ou não) ser definidos como delimitadores pela sintaxe genérica num determinado esquema URL (por exemplo, barra / caracteres são usados para separar diferentes partes de um URL).

Caracteres reservados num URL são os seguintes:

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

***Figura 71** – Caracteres reservados num URL (Fonte: https://www.tutorialrepublic.com/html-tutorial/html-url-encode.php)*

No entanto, também existem caracteres que, apesar de permitidos num URL, não possuem uma finalidade reservada – por isso são chamados de "caracteres não reservados".
Isso inclui letras maiúsculas e minúsculas, dígitos decimais, hífen, ponto, sublinhado e til. A tabela a seguir lista todos os caracteres não reservados num URL:

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

***Figura 72** – Caracteres não reservados em um URL (Fonte: https://www.tutorialrepublic.com/html-tutorial/html-url-encode.php)*

Para codificar carateres, os utilizadores podem usar este conversor.

Validação HTML

Para garantir que um código HTML segue os padrões atuais da Web, livre de erros, é importante descobrir como validar um código HTML. Os iniciantes geralmente cometem erros ao escrever códigos HTML, e códigos incorretos ou fora do padrão certamente causarão resultados inesperados na forma como uma página da Web é exibida num navegador.

Para evitar que isso aconteça, os utilizadores podem testar seus códigos dentro das diretrizes e padrões formais estabelecidos pela Wide Web Consortium (W3C) para páginas da web HTML/XHTML. Existe uma ferramenta online que verifica automaticamente os códigos HTML e aponta quaisquer problemas/erros, como tags de fecho ausentes ou aspas ausentes em torno de atributos.

O processo de validação de uma página web garante o respeito às normas/padrões definidos pelo W3C, por isso é muito importante. Algumas razões para validar uma página da web são:

- Ajuda a criar páginas da Web compatíveis com vários navegadores e plataformas. Muito provavelmente eles serão compatíveis com a versão futura dos navegadores e padrões da web.
- As páginas da web em conformidade com os padrões aumentam a visibilidade dos spiders e rastreadores do mecanismo de pesquisa, como resultado, as páginas da web provavelmente aparecerão nos resultados da pesquisa.
- Isso reduzirá erros inesperados e tornará suas páginas da web mais acessíveis ao visitante.

Características HTML5

Neste subcapítulo, serão explicados os recursos da quinta (e última) versão HTML principal recomendada pelo W3C.

Novos tipos de entrada HTML5

O HTML5 apresenta vários novos tipos de <input> como email, data, hora, cor, intervalo e assim por diante, com o objetivo de melhorar a experiência do usuário e tornar os formulários mais interativos. No entanto, se um navegador não reconhecer esses novos tipos de entrada, ele vai considerá-los uma caixa de texto normal.

Os seguintes são alguns novos tipos de entrada:

- cor
- data
- datetime-local
- email
- mês
- número
- range
- pesquisa
- tel
- hora
- url
- semana

Com relação à entrada de cores, permite selecionar uma cor de um seletor de cores e fornece informações sobre o valor da cor em formato hexadecimal (por exemplo, #000000, que é preto, a cor padrão se o usuário não especificar um valor), conforme verificado na *Figura 73* abaixo.

Deve-se notar que a entrada de cores é suportada em todos os principais navegadores modernos (Firefox, Chrome, Opera, Safari (12.1+), Edge (14+)), mas não é suportado pelo Microsoft Internet Explorer e versões mais antigas do Navegadores Apple Safari.

Figura 73 *– Escolher uma cor usando o color input (Fonte: https://www.tutorialrepublic.com/codelab.php?topic=html5&file=color-input-type)*

Quanto ao tipo de entrada de data, permite ao usuário selecionar uma data de um calendário suspenso, no qual pode escolher o ano, mês e dia (mas não a hora). Esse recurso também é compatível com a maioria dos navegadores, exceto Internet Explorer e Safari.

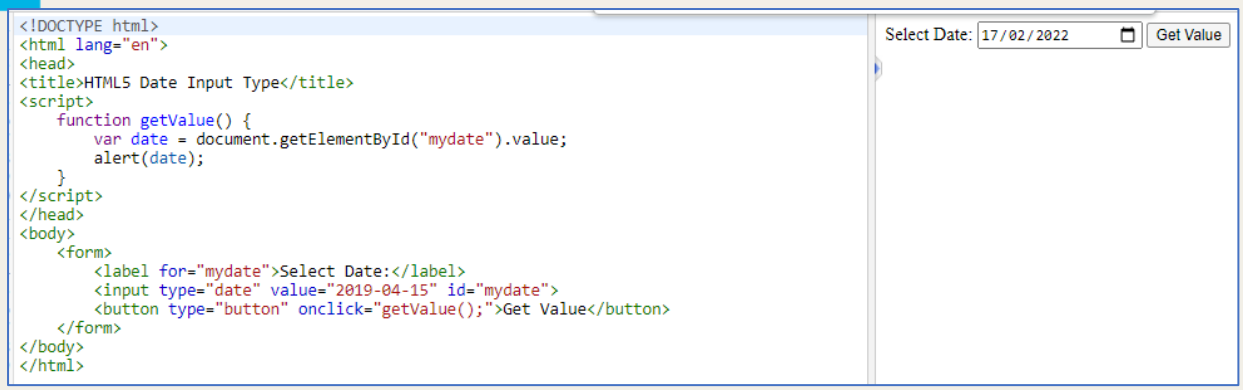


Figura 74 *– Data input type (Fonte: https://www.tutorialrepublic.com/codelab.php?topic=html5&file=color-input-type)*

O tipo de entrada datetime-local disponibiliza ao utilizador a seleção de data e hora local, incluindo ano, mês e dia, e incluindo a hora em horas e minutos. Esta entrada é suportada pelo Safari, Firefox e IE, mas não pelo Chrome, Edge e Opera.

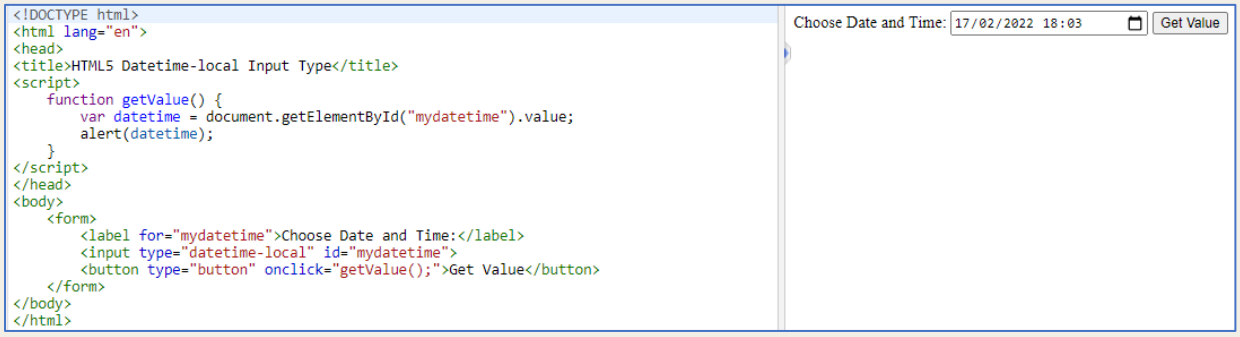
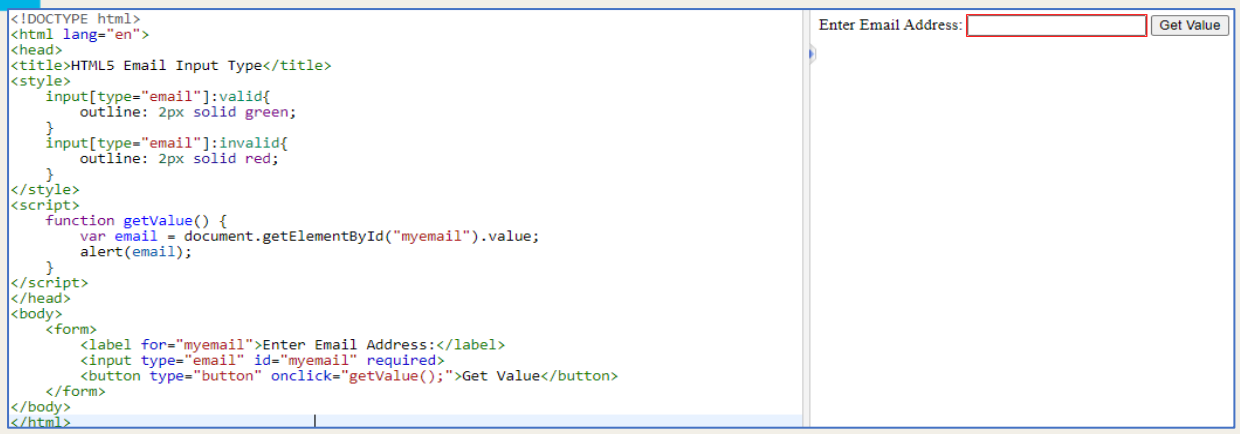


Figura 75 *– A entrada datetime-local (Fonte: https://www.tutorialrepublic.com/codelab.php?topic=html5&file=color-input-type)*

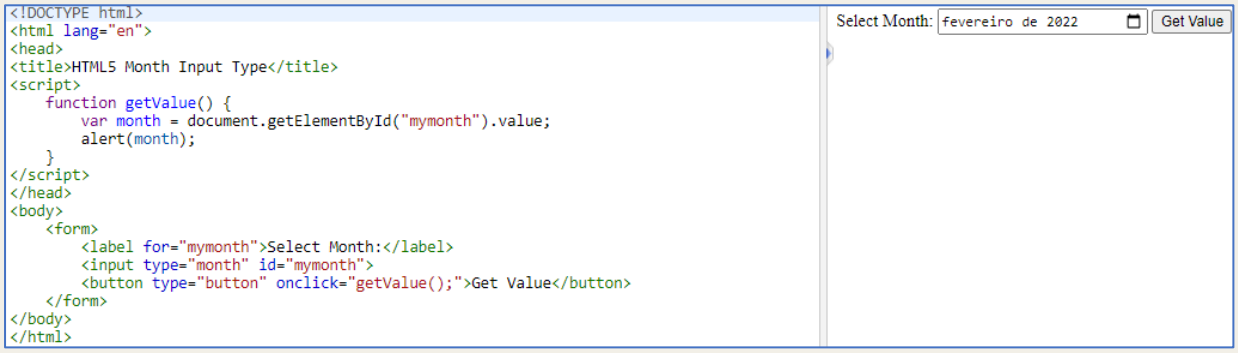
Para permitir que o utilizador insira o seu endereço de e-mail, e-mail é o tipo de entrada preferencial. É como um tipo de entrada de texto padrão, mas se for aplicado em combinação com o atributo obrigatório, os navegadores podem procurar os padrões para garantir que um endereço de e-mail formatado corretamente seja inserido. O campo de entrada de email pode ser estilizado para diferentes estados de validação, quando um valor é inserido usando as pseudoclasses :valid, :invalid ou :required. Este tipo de entrada é suportado por todos os principais navegadores.




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

Figura 76 *– A entrada de email (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

No que diz respeito ao tipo de entrada de mês, esta é uma funcionalidade muito semelhante às anteriores, pois permite selecionar um mês e ano de um calendário suspenso (sendo 'YYYY' o ano e "MM" o mês). Deve-se notar que isso não é suportado pelo Firefox, Safari e Internet Explorer. Apenas os navegadores Chrome, Edge e Opera o suportam.




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

Figura 77 *– A entrada de mês (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

No que diz respeito ao tipo de entrada numérica, é utilizado para inserir um valor numérico. O web designer também pode limitar o utilizador a inserir apenas valores aceitáveis. Para que isso aconteça, os atributos adicionais min, max e step devem ser usados. Este recurso é suportado por todos os principais navegadores da web.

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

Enter a Number: 1,5 Get Value

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

Figura 78 *– A entrada numérica (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

O tipo de entrada de intervalo pode ser aplicado para registrar um valor numérico dentro de um intervalo específico. Funciona de maneira muito semelhante à entrada numérica acima, embora apresente uma maneira mais fácil de inserir um número. Este tipo de entrada é suportado por todos os principais navegadores da web.

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

Select a Number: Get Value

Figura 79 *– A entrada de range (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

O tipo de entrada de pesquisa é adequado para gerar campos de entrada de pesquisa. Deve-se dizer que, em alguns navegadores (isto é, Chrome e Safari), assim que o utilizador começa a escrever na caixa de pesquisa, surge uma minúscula cruz do lado direito do campo, que pode ser útil para limpar todo o campo de pesquisa. É suportado por todos os principais navegadores.

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

Figura 80 *– A entrada de pesquisa (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Quanto ao tipo de entrada tel, é especialmente útil para inserir um número de telefone. Como os navegadores não suportam a validação de entrada tel por padrão, o atributo placeholder pode ser usado para ajudar os utilizadores a inserir o formato correto para seu número de telefone ou indicar uma expressão regular para validar a entrada do usuário aplicando o atributo pattern. Este recurso não é suportado por nenhum navegador, pois os números de telefone variam muito em todo o mundo.

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

Figura 81 *– A entrada tel (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Em relação ao tipo de entrada de hora, ele pode ser usado para inserir qualquer hora (horas e minutos), e o navegador pode usar os dois formatos de hora (12 ou 24 horas) para inserir horas, dependendo da região. Esta entrada não é suportada pelos navegadores IE e Safari.

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

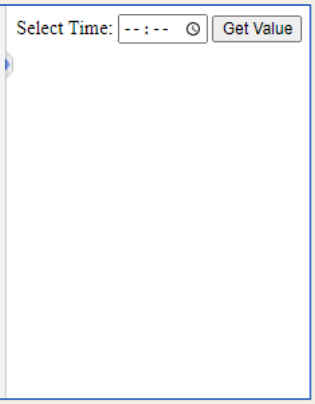


Figura 82 *– A entrada de hora (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Em relação ao tipo de entrada de url, ele pode ser usado para inserir URLs ou endereços da web. O atributo multiple pode ser usado para inserir mais do que uma URL. Além disso, se o atributo obrigatório for restrito, o navegador executará automaticamente a validação para garantir que apenas o texto que atenda ao formato padrão para URLs entre na caixa de entrada. Todos os principais navegadores suportam este tipo de entrada.

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

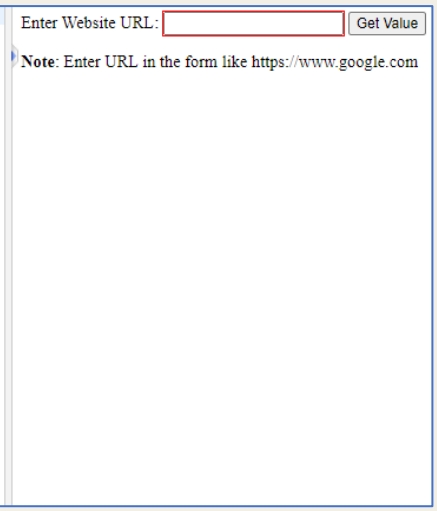
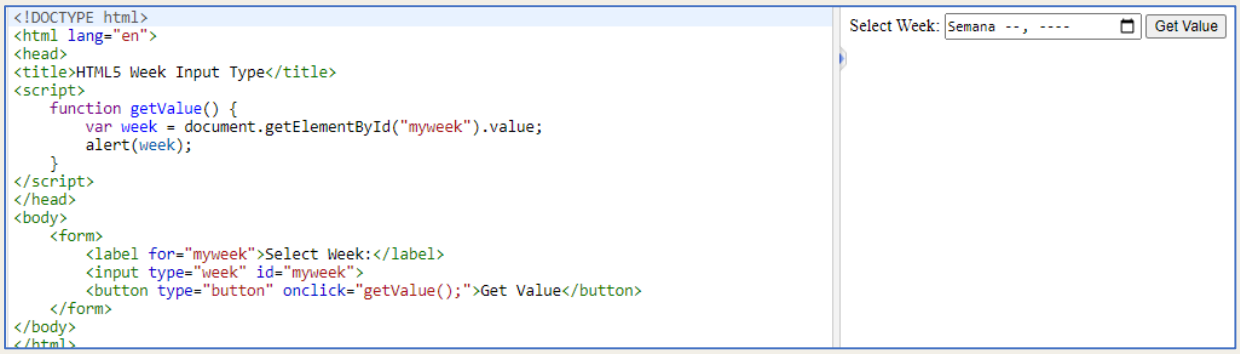


Figura 83 *– A entrada de URL (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Por fim, o tipo de entrada de semana permite que o usuário escolha uma semana e um ano em um calendário suspenso. Este recurso não é suportado pelo Firefox, Safari e IE, mas atualmente é suportado pelos navegadores Chrome, Edge e Opera.




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

Figura 84 *– A entrada de semana (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Canvas HTML5

Este subcapítulo será essencialmente útil para aprender a desenhar gráficos usando o elemento canvas do HTML5. Ele foi originalmente criado pela Apple para os widgets do painel do Mac OS e para habilitar gráficos no Safari. Além disso, foi adotado por outros navegadores, como Firefox, Google Chrome e Opera.

Por padrão, o elemento <canvas> tem 300px de largura e 150px de altura sem nenhuma borda e conteúdo. No entanto, largura e altura personalizadas podem ser especificadas usando o recurso CSS altura e largura.

O ecrã é uma área bidimensional de quatro lados. As coordenadas do canto superior esquerdo do ecrã são (0, 0), identificadas como origem, e as coordenadas do canto inferior direito são (largura do ecrã, altura do ecrã), como pode ser visto usando a ferramenta interativa disponível aqui.

Para desenhar caminhos e formas básicas utilizando o elemento canvas HTML5 recentemente introduzido e JavaScript, será útil dar uma olhada a vários modelos.

Em primeiro lugar, o modelo base para desenhar caminhos e formas na tela 2D HTML5:

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

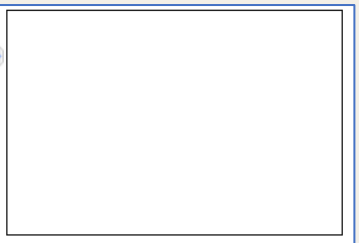


Figura 85 *– O modelo básico para desenhar caminhos e formas no ecrã 2D HTML5 (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Todas as linhas, exceto as de 7 a 11, são bastante diretas e fáceis de interpretar. A função anónima afixada ao evento window.onload será executada quando a página for carregada. Assim que a página é carregada, pode-se aceder ao elemento canvas com o método document.getElementById(). Mais tarde, um contexto de ecrã 2D é definido passando 2d para o método getContext() do objeto do ecrã.

O passo inicial para desenhar no ecrã é uma linha reta. Os procedimentos mais importantes usados para este propósito são moveTo(), lineTo() e o stroke(). O método moveTo() identifica a posição do cursor de desenho na tela, enquanto o método lineTo() é usado para definir as coordenadas do ponto final da linha e, finalmente, o método stroke() é usado para tornar a linha visível:

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

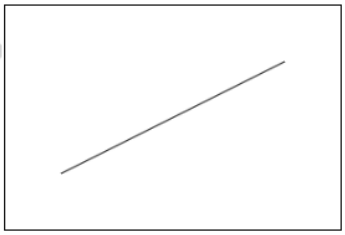


Figura 86 *– Os procedimentos moveTo(), lineTo() e stroke() (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Que tal **desenhar um arco**? Ele pode ser obtido simplesmente usando o método arc(), cuja sintaxe é:

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

No exemplo acima, um arco foi desenhado na tela inserindo um código JavaScript:

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

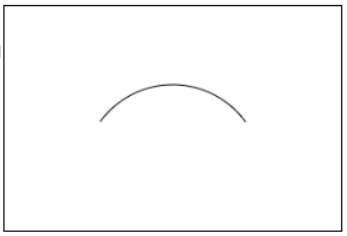


Figura 87 *– Desenhar um arco usando o código JS (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Para desenhar um retângulo e formas quadradas, o método rect() é o caminho a seguir. Ela envolve quatro parâmetros: posições x, y do retângulo e sua largura e altura. A sintaxe básica do método rect() é a seguinte:

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

Para desenhar usando um código JS:

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

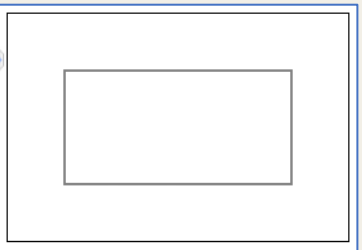


Figura 88 *– Desenhar um retângulo usando um código JS (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Ao contrário do método rect(), não existe um procedimento específico para desenhar um círculo. No entanto, o resultado pode ser obtido criando um arco totalmente fechado, usando o método arc(). A sintaxe para desenhar um círculo completo usando o método arc() é a seguinte:

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

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

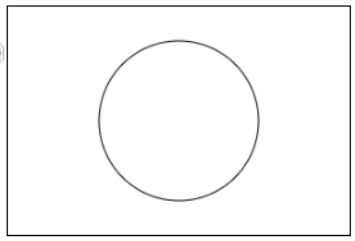


Figura 89 *– Desenhando um círculo na canvas HTML5 (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Em relação aos **estilos e cores do traço**, a cor padrão do traço é preto, sendo sua espessura de 1 pixel. No entanto, esses atributos podem ser alterados usando as propriedades strokeStyle e lineWidth, conforme segue na *Figura 90*.

Na *Figura 91*, é possível definir o estilo de cap para as linhas usando a propriedade lineCap, com três estilos disponíveis: butt, round e square.

Também é possível preencher a cor dentro das formas da canvas usando a abordagem fillStyle(). A *Figura 92* mostra como preencher uma cor sólida dentro de uma forma retangular. Ao projetar as formas na tela, sugere-se usar o método fill() antes do método stroke() para renderizar o traçado adequadamente.

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

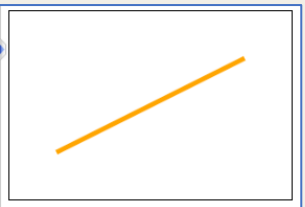


Figura 90 *– Definir os estilos e cores no traçado usando as propriedades strokeStyle e lineWidth (Source: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

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

Figura 91 *– Configurar o estilo de cap para as linhas usando a propriedade lineCap (Source: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

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

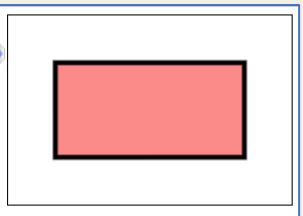


Figura 92 *– Arquivar a cor dentro das formas da tela usando a abordagem fillStyle() (Source: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

Também é possível preencher gradiente de cores (uma transição visual suave de uma cor para outra) nas formas da tela. Existem dois tipos de gradientes aqui: linear e radial.

A sintaxe básica para criar um **gradiente linear** é:

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

A sintaxe básica para criar um **gradiente radial** é:

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

A *Figura 93* mostra como preencher uma cor **gradiente linear** dentro de um retângulo usando o método createLinearGradient():

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

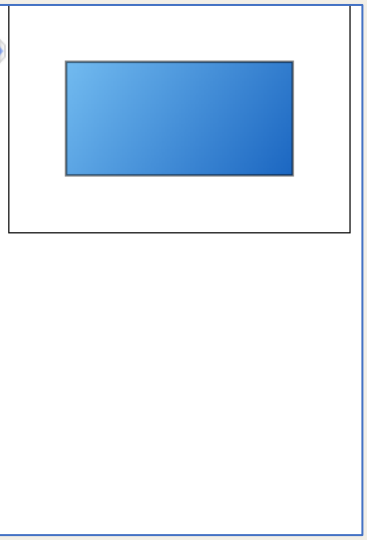


Figura 93 *– Arquivar uma cor gradiente linear dentro de um retângulo usando o método createLinearGradient()*

(Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)

A *Figura 94* demonstra como preencher uma cor de gradiente radial dentro de um círculo por meio do método createRadialGradient():

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

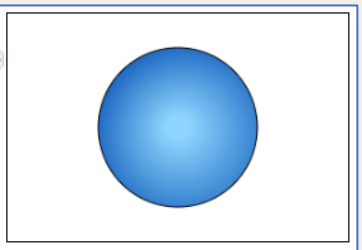


Figura 94 *– Preencher uma cor gradiente radial dentro de um círculo através do método createRadialGradient()*

(Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)

Também é possível desenhar texto em tela (contendo apenas caracteres Unicode), além de **adicionar cor e alinhamento**, e até aplicar traço no texto usando o método strokeText(), que irá colorir o perímetro do texto em vez de preenchê-lo . No entanto, se o web designer pretende definir o preenchimento e o traço no elemento de texto, ele pode usar os métodos fillText() e strokeText() juntos. sugere-se usar o método fillText() antes do método strokeText() para renderizar o traço com precisão.

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

Figura 95 *– Desenhar texto no canvas (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-new-input-types.php)*

SVG HTML5

Espera-se que este subcapítulo seja claro sobre como usar elementos svg HTML5 para desenhar gráficos vetoriais numa página da web. Para isso, em primeiro lugar é importante definir SVG.

SVG significa Scalable Vector Graphics e é um formato de imagem baseado em XML que é usado para definir gráficos vetoriais bidimensionais para a web. Diferente da imagem rasterizada (por exemplo, .jpg, .gif, .png e outros formatos bidimensionais), uma imagem vetorial pode ser ampliada ou reduzida em qualquer medida sem perder a qualidade da imagem. As imagens vetoriais são compostas por uma série de formas definidas pela matemática, enquanto as imagens rasterizadas são compostas por um conjunto fixo de pontos (pixels). Como outros tópicos discutidos ao longo deste capítulo, o SVG também é uma recomendação do W3C.

Uma imagem SVG é construída usando uma sequência de instruções que acompanham o esquema XML, portanto, as imagens SVG podem ser criadas e editadas com um editor de texto, como o Bloco de Notas. Existem inúmeras vantagens para usar imagens SVG em vez de outros formatos de imagem, como segue:

- Elas podem ser pesquisadas, indexadas, roteirizadas e compactadas.
- Elas podem ser criadas e modificadas usando JavaScript em tempo real.
- Podem ser impressas com alta qualidade em qualquer resolução.
- Elas podem ser animadas usando os elementos de animação integrados.
- Podem conter hiperlinks para outros documentos.

Gráficos SVG podem ser incorporados diretamente num documento usando o elemento HTML5 <svg> (veja a *Figura 96* abaixo).
Todos os principais navegadores modernos (Chrome, Firefox, Safari e Opera), bem como o Internet Explorer 9 e superior, são compatíveis com renderização SVG inline.

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

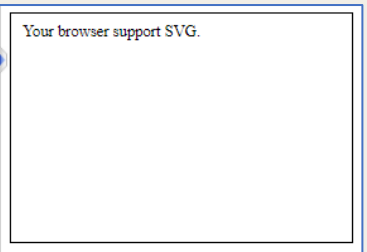


Figura 96 *– Incorporando gráficos SVG diretamente usando o elemento svg (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)*

Caminhos e formas básicas baseadas em vetor nas páginas da Web podem ser desenhados utilizando o elemento HTML5 <svg>.

O caminho mais básico para trabalhar com SVG é **desenhar uma linha reta**. Para que isso aconteça, o elemento SVG <line> deve ser usado. Como pode ser visto na Figura 97, os atributos x1, x2, y1 e y2 do elemento <line> desenham uma linha de (x1,y1) to (x2,y2).

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

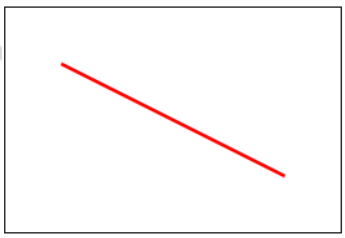


Figura 97 *– Desenhar uma linha reta com o elemento SVG <line> (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)*

Para desenhar um **retângulo e quadrados**, o elemento SVG <rect> é a forma mais adequada. Os atributos x e y do elemento <rect> especificam as coordenadas do canto superior esquerdo do retângulo. Os atributos largura e altura especificam a largura e a altura da forma.

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

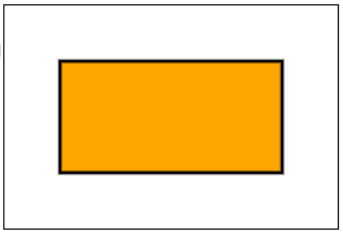


Figura 98 *– Desenhar um retângulo com o elemento SVG <rect> (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)*

Se um **círculo** for a forma escolhida, o elemento SVG <circle> é o mais adequado. Os atributos cx e cy do elemento <circle> especificam as coordenadas do centro do círculo e o atributo r identifica o raio do círculo. Ainda assim, se os atributos cx e cy estiverem ausentes ou não especificados, o centro do círculo é definido como (0,0).

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

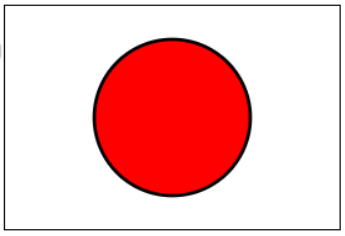


Figura 99 *– Desenhar um círculo com o elemento SVG <circle> (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)*

Também é possível **desenhar texto** com SVG. O texto em SVG é renderizado como um gráfico para que o web designer possa usar toda a transformação gráfica nele, mas ainda funciona como texto, para que possa ser selecionado e copiado como texto pelo utilizador. Os atributos x e y do elemento <text> identificam a localização do canto superior esquerdo em termos absolutos, embora os atributos dx e dy indiquem a localização relativa.

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

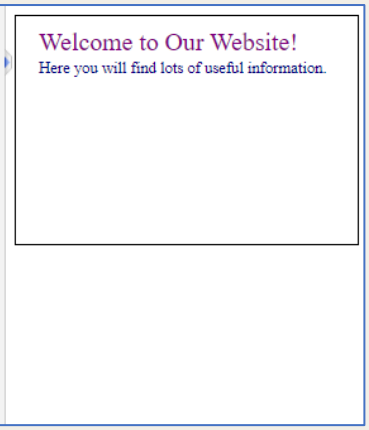


Figura 100 *– Texto de desenho com elemento SVG <text> (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)*

Em alternativa, os web designers podem usar o elemento <tspan> para redimensionar ou realocar o intervalo de texto incluído num elemento <text>. O texto é incluído em tspans separados, mas dentro do mesmo elemento de texto podem ser selecionados todos ao mesmo tempo — ao clicar e arrastar para selecionar o texto. No entanto, o texto em elementos de texto separados não pode ser escolhido ao mesmo tempo.

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

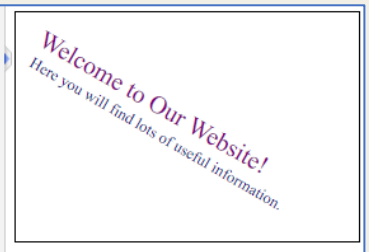


Figura 101 *– Desenho de texto com elemento SVG <tspan> (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)*

Embora os novos elementos gráficos <canvas> e <svg> tenham sido introduzidos pelo HTML 5 para criar gráficos de alta qualidade na web, eles diferem bastante.
Na tabela abaixo, as diferenças entre ambos são resumidas, e isso ajudará os alunos a usá-los de maneira apropriada e eficaz:

SVG	Canvas
Baseado em vetor (composto de formas)	Baseado em rasterização (composto por pixels)
Vários elementos gráficos, que se tornam parte da árvore DOM da página	Elemento único semelhante a <img> no comportamento. O diagrama de canva pode ser guardado no formato PNG ou JPG
Modificado através de script e CSS	Modificado apenas por script
Bons recursos de renderização de texto	Más capacidades de renderização de texto
Dê melhor desempenho com menor número de objetos ou superfície maior, ou ambos	Dá melhor desempenho com grande número de objetos ou superfície menor, ou ambos
Melhor escalabilidade. Pode ser impresso com alta qualidade em qualquer resolução. A pixelização não ocorre	Má escalabilidade. Não é adequado para impressão em resolução mais alta. Pixelização pode ocorrer

***Figura 102** – As diferenças entre SVG e canvas (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-svg.php)*

Áudio HTML 5

Este subcapítulo destina-se a explicar como incorporar áudio num documento HTML. Como os navegadores da web não tinham um padrão uniforme para incorporar arquivos de media como áudio, não era uma tarefa fácil de executar no passado. No entanto, existem muitas maneiras de incorporar som numa página da Web, desde simplesmente usar um link simples até usar o elemento HTML5 <audio>. Este elemento fornece uma maneira padrão de inserir áudio em páginas da web. Como o elemento de áudio é recente, ele é executado na maioria dos navegadores modernos.

Há muitas maneiras de inserir um áudio num documento HTML5. Uma delas é usar o grupo de controlos padrão do navegador, com uma fonte definida pelo atributo src, como pode ser verificado neste código, em que é possível ouvir um grupo de pássaros a cantar.

Outra maneira pode ser alcançada usando o elemento <object>, que é usado para incorporar diferentes tipos de arquivos de media. Este exemplo inclui um arquivo de áudio numa página da web seguindo o método mencionado acima. Deve-se notar que o elemento <object> não é amplamente suportado e depende do tipo de objeto que está a ser implantado. Outros métodos, como o elemento HTML5 <audio> ou players de áudio HTML5 de terceiros podem ser uma opção melhor em muitos casos.

Finalmente, o elemento <embed> também pode ser outra forma de inserir media num documento HTML, seguindo este exemplo. Embora o elemento <embed> seja muito bem suportado nos navegadores atuais e definido como padrão em HTML5, o áudio inserido pode não ser reproduzido devido à ausência de suporte do navegador para esse formato de arquivo ou inacessibilidade de plugins.

Vídeo HTML5

Agora é hora de aprender a incorporar conteúdo de vídeo num documento HTML.

Assim como o som, o conteúdo de vídeo também era difícil de inserir numa página da web, e pelo mesmo motivo (navegadores da web não tinham um padrão uniforme para definir arquivos de media incorporados como vídeo). Nos próximos parágrafos, muitas maneiras de inserir esses conteúdos serão explicadas.

O elemento <video> recém-introduzido funciona na maioria dos navegadores modernos. Este exemplo explica como simplesmente inserir um vídeo no documento HTML, usando o conjunto de controlos padrão do navegador, com uma fonte definida pelo atributo src. O elemento <object> também é usado para incorporar diferentes tipos de arquivos de media. Seguindo este exemplo, pode-se entender como incorporar um vídeo Flash numa página da web (somente navegadores/aplicativos que suportem Flash poderão reproduzi-lo). Deve-se notar que o elemento <object> não é amplamente suportado e depende muito do tipo de objeto incorporado. Outros métodos podem ser uma escolha melhor em muitos casos, como, por exemplo, os dispositivos iPad e iPhone não podem exibir vídeos em Flash.

E quanto à **incorporação de vídeos do YouTube**? Essa é realmente a maneira mais simples e comum de incorporar arquivos de vídeo em páginas da web, hoje em dia. O web designer deve simplesmente fazer o upload do vídeo no YouTube e inserir o código HTML para exibir o vídeo na sua página da web. Aqui está um miniguia passo a passo:

Passo 1 – Carregar vídeo no YouTube.

Passo 2 – Após o upload de um vídeo no YouTube, o web designer deve procurar o botão 'partilhar', localizado abaixo de um vídeo em execução no player de vídeo da plataforma, como segue :

Figura 103 *– Botão partilhar no YouTube (Fonte: Autor)*

Ao clicar no botão 'partilhar', um painel para partilhar será aberto, exibindo mais algumas opções. Agora, o botão '**Incorporar**' deve ser clicado, pois ele irá gerar o código HTML para incorporar diretamente o vídeo na página da web. Para que isso aconteça, o web designer deve copiar e colar esse código no documento HTML.

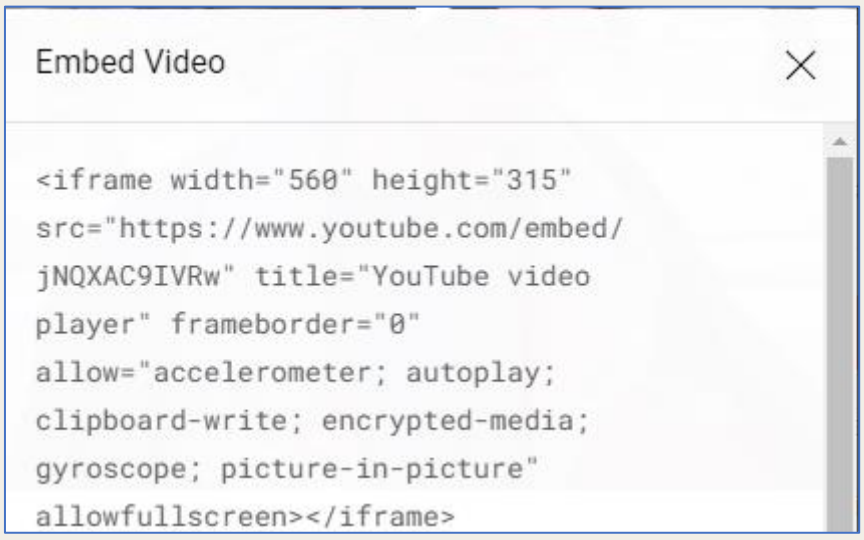


Figura 104 *– Opção "incorporar" no YouTube (Fonte: Autor)*

Este código pode ser customizado ainda mais e para isso o web designer só precisa selecionar a opção de customização dada logo abaixo da caixa de entrada do código de incorporação.

A inserção de um vídeo do YouTube numa página da web é explicada neste exemplo.

Armazenamento Web HTML5

Já se perguntou como usar o recurso de armazenamento na Web HTML5 para armazenar dados no navegador do utilizador? Os parágrafos a seguir serão úteis para entender isso completamente.

Em primeiro lugar, é importante entender as implicações do “armazenamento na web”.

Com o armazenamento na Web, os aplicativos da Web podem armazenar dados localmente, no navegador do utilizador. Antes do HTML5, os dados do aplicativo precisavam ser armazenados em cookies, incorporados em todas as solicitações do servidor. O armazenamento na Web é mais seguro e quantidades substanciais de dados podem ser armazenadas localmente, sem afetar o desempenho do site. As informações mantidas no armazenamento web não são enviadas ao servidor web, ao contrário dos cookies onde os dados são entregues ao servidor a cada solicitação. Além disso, os cookies permitem armazenar apenas uma pequena quantidade de dados (quase 4 KB), enquanto o armazenamento na web permite armazenar até 5 MB.

Existem dois tipos de armazenamento na web:

- **Armazenamento local** — usa o objeto localStorage para armazenar dados de todo o site de forma permanente. Dito isto, os dados locais armazenados estarão disponíveis no dia seguinte, na próxima semana ou no próximo ano, a menos que sejam removidos.
- **Armazenamento de sessão** — usa o objeto sessionStorage para armazenar dados temporariamente, para uma única janela ou guia do navegador. Os dados desaparecem quando a sessão termina, por exemplo, quando o utilizador fecha a janela ou guia do navegador.

No que diz respeito ao armazenamento local, cada dado é coletado num par chave/valor. A chave identifica o nome da informação (ou seja, 'first_name') e o valor é o valor relacionado à mesma chave (ou seja, 'Peter'). O seguinte código JS expressa o seguinte:

- localStorage.setItem(chave, valor) armazena o valor associado a uma chave.
- localStorage.getItem(key) salva o valor associado à chave.

Também é possível remover um item específico do armazenamento, passando o nome da chave para o método removeItem(), ou seja, localStorage.removeItem("first_name").

No entanto, se o web designer pretende remover o armazenamento completo, ele deve usar o método clear(), ou seja, localStorage.clear(). O método clear() simplesmente limpa todos os pares de chave/valor do localStorage de uma só vez, portanto, **deve ser usado com cautela**. Os dados de armazenamento na web não estarão acessíveis entre diferentes navegadores.

Por fim, o objeto sessionStorage funciona de maneira semelhante ao localStorage, exceto que armazena os dados apenas para uma sessão. O seguinte exemplo é bastante explicativo em relação a como funciona.

Cache do Aplicativo HTML5

Durante o subcapítulo, os alunos podem aprender mais sobre como criar aplicativos offline usando o **recurso de cache HTML5**.

Sabe-se que a maioria dos aplicativos baseados na web não funcionará se o web designer estiver offline. No entanto, o HTML5 traz um mecanismo de cache de aplicativos que permite que o navegador guarde automaticamente o arquivo HTML e todos os outros recursos necessários para exibi-lo corretamente na máquina local, dessa forma o navegador ainda pode aceder à página da web e os seus recursos sem estabelecer uma conexão com a internet. Isso é suportado em todos os principais navegadores da Web modernos (Firefox, Chrome, Opera, Safari e Internet Explorer 10 e superior).

Existem várias vantagens em usar este recurso:

• **Navegação offline** — Os visitantes podem usar o aplicativo mesmo quando não estão online ou há interrupções inesperadas na conexão de rede.

• **Melhore o desempenho** — Os recursos armazenados em cache são carregados diretamente da máquina do utilizador, em vez do servidor remoto, para que as páginas da Web sejam carregadas mais rapidamente e tenham um desempenho melhor.

• **Reduzir a solicitação HTTP e a carga do servidor** — O navegador terá apenas que baixar os recursos atualizados/alterados do servidor remoto que reduzem as solicitações HTTP e economizam largura de banda valiosa, além de diminuir a carga no servidor web.

Existem algumas etapas a serem seguidas para armazenar em cache os arquivos para uso offline:

PASSO 1 – Crie um arquivo de manifesto de cache. Este é um arquivo de texto especial que informa aos navegadores quais os arquivos que eles devem armazenar (e não) e quais os arquivos que devem ser substituídos. Começa sempre com as palavras CACHE MANIFEST (sempre em maiúsculas).

A *Figura 105* abaixo é um exemplo de um arquivo de manifesto:

Example ⬇ Download

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

CACHE:
# pages
index.html

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

# images
/favicon.ico
images/logo.png

NETWORK:
login.php

FALLBACK:
/ /offline.html
```

Figura 105 *– Exemplo de arquivo de manifesto (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-application-cache.php)*

Agora é hora de explicar a codificação da Figura 105.

• Em primeiro lugar, é importante entender que os arquivos de manifesto podem ter três seções diferentes: **CACHE, NETWORK** e **FALLBACK**.

• Os arquivos listados sob o cabeçalho da secção **CACHE**: (ou logo após a linha CACHE MANIFEST) são claramente armazenados em cache após serem guardados pela primeira vez;

• Os arquivos sob o cabeçalho da seção **NETWORK**: são recursos da lista de permissões que nunca são armazenados em cache e estão disponíveis apenas online. Isso significa que os usuários nunca podem aceder a página login.php quando estiverem offline;

• A secção **FALLBACK**: especifica páginas alternativas que o navegador deve usar caso a conexão com o servidor não possa ser feita. Cada entrada nesta secção lista dois URLs — o primeiro é o recurso primário, o segundo é o fallback. Por exemplo, no caso da *Figura 105*, a página offline.html será exibida se o

utilizador estiver offline. Além disso, ambos os URLs devem ser da mesma origem que o arquivo de manifesto.

• Deve-se notar que as linhas que começam com '#' (símbolo de hash) são linhas de comentário.

Portanto, se houver um cache de aplicativo, o navegador carrega o documento e seus recursos associados diretamente do cache, sem aceder à rede. Em seguida, o navegador verifica se o arquivo de manifesto foi atualizado no servidor. Caso tenha sido atualizado, o navegador baixa a nova versão do manifesto e os recursos listados nele.

É importante observar que o próprio arquivo de manifesto não deve ser especificado no arquivo de manifesto de cache. Nesse caso, será muito difícil notificar o navegador de que um novo manifesto está disponível.

PASSO 2 – Use o arquivo de manifesto de cache. Depois de criá-lo, o web designer deve fazer upload do arquivo de manifesto de cache no servidor da web — certificando-se de que o servidor da web esteja configurado para servir os arquivos de manifesto com o tipo MIME text/cache-manifest.

Para fazer o cache manifest funcionar, o web designer precisará habilitá-lo nas páginas da web, adicionando o atributo manifest ao elemento raiz <html>, conforme mostrado na Figura 106 abaixo:

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

Figura 106 *– Fazer o cache manifest funcionar (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-application-cache.php)*

Se o utilizador está online, o resultado para este Código será o seguinte:

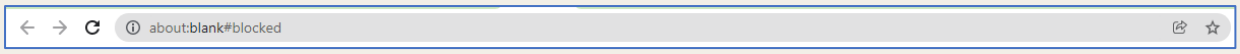


Figura 107 *– about_blank#blocked (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-application-cache.php)*

Web workers do HTML5

Este subtópico será essencialmente útil para aprender mais sobre tópicos JS, pois ensinará como usar o web worker HTML5 para executar o código JS em segundo plano. Assim, os alunos devem voltar se tiverem alguma dificuldade.

Se alguém tentar realizar cálculos intensivos, demorados e pesados, exigindo tarefas com JavaScript, provavelmente irá congelar as páginas da Web e impedir que os utilizadores façam qualquer coisa até que o trabalho seja concluído. Por quê? Bem, porque o código JS é sempre executado em primeiro plano. No entanto, o HTML5 possui uma nova tecnologia (‘web worker’) criada para realizar o trabalho em segundo plano, além de outros scripts de interface do usuário, sem afetar o desempenho da página. Diferente das operações normais de JS, o web worker não interrompe o utilizador e a página da web permanece responsiva, porque estão as tarefas estão a ser executadas em segundo plano.

Os web workers são especialmente úteis para executar uma tarefa demorada. Assim, no primeiro exemplo, será criada uma tarefa JS simples que conta de zero a 100.000 (o nome do arquivo deve ser worker.js), conforme segue na *Figura 108:*

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

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

Figura 108 *– Criar uma tarefa JS contando de 0 a 100.000 (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-web-workers.php)*

Nota: Para uma melhor compreensão, é aconselhável transferir o código da Fig. 101 e seguir cada passo deste capítulo.

Desta forma, agora que o arquivo do web worker foi criado, é hora de iniciar o web worker a partir de um documento HTML. Esse documento executa o código dentro do arquivo chamado "worker.js", em segundo plano, e mostra gradualmente o resultado na página da web. Deve-se notar que o número à direita estará sempre a crescer, até atingir 100 000.

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

26

Figura 109 *– Iniciar o web worker (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-web-workers.php)*

Agora explicando o que está acontecendo no exemplo acima, a instrução **var worker = new Worker**("worker.js"); cria um novo objeto no trabalhador da web, que é usado para comunicar com o trabalhador da web. Quando o trabalhador publica uma mensagem, ele aciona o manipulador de eventos **onmessage** (linha 14), que permite que o código receba mensagens do web worker. O elemento **event.data** compreende a mensagem enviada do **web worker**. Para o registo, o código que um trabalhador executa é sempre armazenado num arquivo JavaScript, separado para evitar que o desenvolvedor da Web escreva o código do trabalhador da Web, que tenta usar variáveis globais ou abrir diretamente os elementos na página da Web.

Também é possível **encerrar um worker em execução** no meio da operação, seguindo o exemplo abaixo:

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

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

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

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

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

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

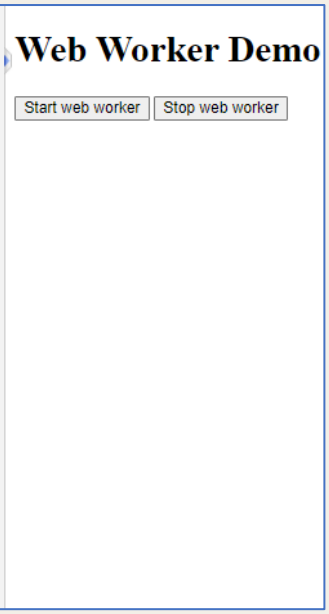


Figura 110 *– Parar o running worker (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-web-workers.php)*

O exemplo acima mostra como iniciar e parar o trabalhador de uma página da web simplesmente clicando nos botões HTML.

Eventos enviados pelo servidor HTML5

Este subcapítulo será útil para entender como usar o recurso de eventos enviados pelo servidor HTML5, para fazer uma conexão unidirecional e permanente entre uma página da Web e um servidor.

O evento enviado pelo servidor HTML5 é uma maneira inovadora de as páginas da Web comunicarem com um servidor da Web. No entanto, existem algumas circunstâncias em que as páginas da Web precisam de uma conexão de longo prazo com o servidor da Web, por exemplo, em cotações de ações em sites de finanças onde o preço é atualizado automaticamente ou tickers de jogos executados em vários sites de desportos. Tais coisas são viáveis com os eventos enviados pelo servidor HTML5, pois disponibiliza para uma página web manter uma conexão aberta com um servidor web, de forma que o servidor web possa enviar uma nova resposta mecanicamente a qualquer momento. Neste ponto, não há necessidade de reconectar e executar repetidamente o mesmo script de servidor desde o início.

Para uma melhor compreensão dos conceitos mencionados acima, use um arquivo PHP(1) chamado "server_time.php" e digite o seguinte script nele. Este arquivo irá apenas relatar a hora atual do relógio interno do servidor web em intervalos regulares:

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

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

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

Figura 111 *– exemplo server_time.php (Fonte https://www.tutorialrepublic.com/html-tutorial/html5-server-sent-events.php)*

(1) (1) Um arquivo PHP é um arquivo de texto simples que contém código escrito na linguagem de programação PHP. Como o PHP é uma linguagem de script do lado do servidor (back-end), o código escrito nele é executado no servidor. Na verdade, um arquivo PHP pode conter texto simples, tags HTML ou código de acordo com a sintaxe PHP. O PHP é frequentemente usado para desenvolver aplicativos da Web que são processados por um mecanismo PHP no servidor da Web.

As duas primeiras linhas do script PHP (Fig. 111) definem dois cabeçalhos cruciais. Em primeiro lugar, define o tipo MIME para text/event-stream, que é necessário para o padrão de evento do lado do servidor. A segunda linha informa ao servidor web para desligar o cache ou então a saída do script pode ser armazenada em cache.

Cada mensagem enviada por meio de eventos enviados pelo servidor HTML5 deve começar com os dados de texto: seguido pelo texto da mensagem real e a nova sequência de caracteres de linha (\n\n).

E por último, a função PHP flush() foi usada para garantir que os dados são enviados imediatamente, em vez de armazenados em buffer até que o código PHP esteja completo.

Sobre como **processar mensagens em uma página web**, o objeto EventSource é utilizado para receber mensagens de eventos enviadas pelo servidor. No exemplo abaixo, os alunos verão como um documento HTML simplesmente recebe a hora atual informada pelo servidor da web e a exibe para os visitantes da página da web. Para um

melhor entendimento, um documento HTML chamado “demo_sse.html” será criado e posteriormente colocado no mesmo diretório do projeto onde está localizado “server_time.php”. O download do código a seguir pode ser feito no link da fonte disponível abaixo.

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

Figura 112 *– Como processar mensagens numa página web (Fonte https://www.tutorialrepublic.com/html-tutorial/html5-server-sent-events.php)*

Geolocalização HTML 5

Por meio deste subtópico, os alunos obterão algumas informações sobre como usar o recurso de geolocalização HTML5 para detetar a localização do utilizador. Esse recurso permite que o programador descubra as coordenadas geográficas (latitude e longitude) da localização atual do visitante do site. É especialmente útil para proporcionar a melhor experiência de navegação ao visitante, pois, por exemplo, esta ferramenta pode mostrar resultados de pesquisa fisicamente próximos da localização do utilizador.

Receber as informações de posição do visitante do site usando a API de geolocalização HTML5 não é difícil. Ele explora os três métodos incluídos no objeto navigator.geolocation — getCurrentPosition(), watchPosition() e clearWatch().

Depois do utilizador concordar em permitir que o navegador informe ao servidor da Web sobre sua posição (os navegadores da Web não partilharão a localização do visitante com uma página da Web, a menos que o usuário concorde em fazê-lo), o processo de geolocalização deve ocorrer da seguinte forma:

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

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

Figura 113 *– O processo de Geolocalização (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-geolocation.php)*

Caso um utilizador não pretenda partilhar os seus dados de localização com o site, o programador pode fornecer duas funções, ao chamar a função getCurrentLocation(). A primeira função é chamada caso a tentativa de geolocalização seja bem-sucedida, enquanto a segunda é chamada se a tentativa de geolocalização falhar.

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

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

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

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

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

Figura 114 *– Aplicando duas funções à função getCurrentLocation() (Fonte: https://www.tutorialrepublic.com/html-tutorial/html5-geolocation.php)*

Existem muitas funções interessantes para explorar com dados de geolocalização, como mostrar a localização do usuário no Google Maps. Com base nos dados de latitude e longitude recuperados por meio do recurso de geolocalização HTML5, este exemplo mostra a localização atual do leitor. Isso simplesmente mostra uma imagem estática mostrando a localização do usuário, embora um Google Maps interativo com arrastar, aumentar/diminuir o zoom e outros recursos, como este exemplo mostra.

Todos os exemplos mencionados acima foram baseados no método getCurrentPosition(). No entanto, a função de geolocalização possui outra técnica – watchPosition() que permite rastrear o movimento do visitante retornando a posição atualizada à medida que a localização muda. watchPosition() tem os mesmos parâmetros de entrada que getCurrentPosition(). No entanto, watchPosition() pode

ativar a função de sucesso várias vezes - quando obtém a localização pela primeira vez e, novamente, toda vez que identifica uma nova posição, como este exemplo sugere.

Arrastar e soltar HTML5

É um procedimento comum na rotina diária online, arrastar e soltar um elemento para outro local em um site. O recurso HTML5 Drag and Drop permite fazer isso, e qualquer elemento pode ser arrastado e solto. Este exemplo w3schools define um exemplo simples de arrastar e soltar, os alunos podem tentar se familiarizar mais com esse conceito. Embora o código pareça difícil de entender, é bastante simples e lógico:

- Em primeiro lugar, para tornar um elemento arrastável, o atributo arrastável deve ser verdadeiro:

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

- Em seguida, deve-se especificar o que deve acontecer quando o elemento for arrastado. No exemplo w3schools dado acima, o atributo ondragstart chama uma função (arrastar (evento)) que especifica quais dados serão arrastados. O processo dataTransfer.setData() define o tipo de dados e o valor dos dados arrastados:

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

Neste exemplo, o tipo de dado é “texto”, sendo o valor o id do elemento arrastável (“drag1”).

- O evento ondragover estipula onde os dados arrastados podem ser soltos. Por padrão, os dados/elementos não podem ser descartados noutros elementos. Para permitir uma queda, o web designer deve evitar a manipulação padrão do elemento, chamando a técnica event.preventDefault() para o evento ondragover:

```
event.preventDefault()
```

- Assim que os dados arrastados são soltos, ocorre um evento de soltar. No exemplo dado anteriormente, o atributo ondrop chama uma função, drop(event):

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

- ✓ Chame preventDefault() para evitar que o navegador manipule os dados por padrão (o padrão é aberto como um link ao soltar);
- ✓ O programador obtém os dados arrastados com a técnica dataTransfer.getData(). Essa técnica retornará todos os dados que foram configurados para o mesmo tipo na técnica setData();
- ✓ Os dados arrastados são o id do elemento arrastado ("drag1")
- ✓ O programador também deve anexar o elemento arrastado no elemento soltar.

Referências HTML5

Para obter uma lista detalhada e abrangente de elementos relacionados a **tags/elementos HTML5, atributos globais HTML5, atributos de evento HTML5, códigos de idioma HTML5, entidades de caracteres HTML5, códigos de status HTTP, seletor de cores HTML5** e outras referências úteis, os alunos devem consultar o seguinte link (HTML5 References section).

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

3. CSS – Folhas de Estilo em Cascata

Informação do Tópico

Tópico:

3. CSS

Pré-requisitos:

Literacia informática básica, software básico instalado, conhecimentos básicos de gestão de ficheiros, e noções básicas de HTML (Introdução ao HTML).

Carga horária:

10 horas.

Descrição:

O CSS é uma linguagem de folhas de estilo utilizada para descrever a apresentação de um documento escrito em HTML ou XML (incluindo dialetos XML, tais como o SVG, MathML ou XHTML). O CSS descreve como os elementos devem ser apresentados no ecrã, em papel, ou noutros meios.

Resultados de aprendizagem:

Os alunos obterão conhecimentos sobre como definir CSS, utilizar sintaxe CSS básica, configurar páginas web com CSS, utilizar CSS para estilizar texto, fonte, e propriedades, utilizar CSS para estilizar fundos de páginas, listas de estilo em CSS, utilizar classes CSS e IDs, utilizar bordas e propriedades CSS de altura e largura, utilizar pseudo elementos CSS, posicionar elementos com CSS e validar CSS e HTML.

Material necessário:

- Computador ou portátil
- Ligação à internet

- Editor de texto (online ou offline): Sublime Text/Brackets/W3Schools online editor

Cenário de Aula:

O tempo total para este tópico é de 10 horas, e caberá ao *coach*/formador decidir quanto tempo dedicar ao ensino de cada subtópico. Para aproveitar ao máximo todo o tempo disponível, propomos a utilização dos materiais de formação produzidos pelo projeto (apresentações PPT), que foram concebidos tendo em mente uma utilização eficaz do tempo. Estas apresentações são compostas pelos seguintes elementos:

- Desenvolvimento do subtópico e principais ideias a reter;
- Atividades/Exercícios propostos.

Dito isto, se o *coach*/formador seguir a sequência lógica dos PPTs, poderá certamente completar a sessão dentro do tempo limite estipulado. Estas apresentações podem também ser disponibilizadas aos formandos para estudo individual.

Subtópicos:

- 3.1. CSS - Introdução
- 3.2. CSS - Avançado
- 3.3. CSS - Modelos de Design Reativo
- 3.4. CSS - Grelhas
- 3.5. CSS SASS

Recursos adicionais:

- Tutorial de CSS: w3schools
- Curso online de HTML e CSS: CodeAcademy

3.1. CSS - Introdução

O que é o CSS?

Cascading Style Sheets (Folhas de Estilo em Cascata), habitualmente designado por CSS, é uma linguagem de design simples, destinada a simplificar o processo de apresentar uma página web.

O CSS trata do aspeto e da vibração que uma página web liberta. Ao utilizar o CSS é possível controlar a cor do texto, o estilo das fontes, o espaçamento entre parágrafos, como as colunas são dimensionadas e dispostas, que imagens ou cores de fundo são usadas, bem como uma variedade de outros efeitos.

CSS é fácil de aprender e compreender, mas proporciona um controlo poderoso sobre a apresentação de um documento HTML. Mais comummente, o CSS é combinado com as linguagens de marcação HTML ou XHTML

Vantagens do CSS

- **O CSS poupa tempo** - Poderás digitar CSS uma vez e depois reutilizar a mesma folha em múltiplas páginas HTML. Podes definir um estilo para cada elemento HTML e aplicá-lo a quantas páginas web entenderes.
- **As páginas carregam mais rápido** - Se estiveres a usar o CSS, não precisarás de escrever atributos de *tag* HTML todas as vezes. Basta escrever uma regra CSS de uma *tag* e aplicá-la a todas as ocorrências dessa *tag*. Portanto, menos código significa tempos de *download* mais rápidos.
- **Manutenção fácil** - Para fazer uma alteração global, basta alterar o estilo e todos os elementos em todas as páginas da web serão atualizados automaticamente.

- **Estilos superiores ao HTML** - O CSS tem uma gama muito mais ampla de atributos em relação ao HTML, por isso pode dar uma aparência muito melhor à tua página HTML, em comparação com os atributos HTML.
- **Compatibilidade de múltiplos dispositivos** - As folhas de estilo permitem a optimização do conteúdo para mais de um tipo de dispositivo. Utilizando o mesmo documento HTML, diferentes versões de um website podem ser apresentadas para dispositivos portáteis, tais como PDAs e telemóveis ou para impressão.
- **Padrões globais da web** – Agora os atributos HTML estão a tornar-se obsoletos, recomendando-se o uso de CSS. Portanto, é uma boa ideia começar a usar o CSS em todas as páginas HTML para torná-las compatíveis com futuros navegadores.

Quem cria e mantém o CSS?

O CSS é criado e mantido através de um grupo de pessoas dentro do W3C chamado Grupo de Trabalho do CSS. O Grupo de Trabalho do CSS cria documentos, designados por *especificações*. Quando uma especificação é discutida e ratificada oficialmente pelos membros do W3C, torna-se uma recomendação.
Estas especificações ratificadas são chamadas recomendações porque o W3C não tem qualquer controlo sobre a implementação efetiva da língua. Empresas e organizações independentes criam esse *software*.

NOTA: O World Wide Web Consortium ou W3C é um grupo que faz recomendações sobre como funciona a Internet e como esta deve evoluir.

Versões do CSS

O CSS foi originalmente lançado em 1996 e consiste em propriedades para adicionar propriedades da fonte, tais como tipo de letra e cor de ênfase do texto, fundos, e outros elementos. O CSS2 foi lançado em 1998 com estilos adicionados para outros tipos de suportes, de modo a poder ser utilizado para a conceção do *layout* da

página. O CSS3 foi lançado em 1999 e nele foram adicionadas propriedades de estilo de apresentação que lhe permitem construir uma apresentação a partir de documentos.

Sintaxe do CSS

Um CSS inclui regras de estilo que são interpretadas pelo navegador e depois aplicadas aos elementos correspondentes no seu documento. Uma regra de estilo é feita de três partes:

- **Selecionador**: Um selecionador é uma etiqueta HTML na qual será aplicado um estilo. Esta poderá ser qualquer tag, como por exemplo <h1>, <table>, etc.
- **Propriedade:** Uma propriedade é um tipo de atributo da etiqueta HTML. Em termos simples, todos os atributos HTML são convertidos em propriedades CSS, que podem ser cor, contorno, etc.
- **Valor**: Os valores são atribuídos às propriedades. Por exemplo, a propriedade de cor pode ter o valor ou vermelho ou #F1F1F1F1 etc.

Selecionadores do CSS

Os selecionadores CSS são usados para *encontrar* (ou *selecionar*) os elementos HTML nos quais desejas aplicar um estilo (ou *estilizar*). Podemos dividir os selecionadores de CSS em cinco categorias:

- **Selecionadores simples** – selecionam elementos com base no nome, id, classe;
- **Selecionadores combinados** – selecionam elementos com base num relacionamento específico entre eles;
- **Selecionadores de pseudo-classe** – selecionam elementos com base num determinado estado;
- **Selecionadores de pseudo-elementos** – selecionam e estilizam uma parte de um elemento;

- **Selecionadores de atributo** – selecionam elementos com base num atributo ou valor de atributo.

O selecionador de elementos CSS

O selecionador de elementos seleciona elementos HTML com base no nome do elemento.

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

***Figura 1** – O selecionador de elementos CSS (**Fonte**: https://www.w3schools.com/css/css_selectors.asp)*

O selecionador de ID do CSS

O selector de id utiliza o atributo id de um elemento HTML para seleccionar um elemento específico. O id de um elemento é único dentro de uma página, por isso o selector de id é usado para selecionar um elemento único. Para selecionar um elemento com um id específico, digita-se um caractere *hash* (#), seguido do *ID* do elemento.

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

***Figura 2** – O Selector de ID CSS (**Fonte:** https://www.w3schools.com/css/css_selectors.asp)*

O selecionador de classe do CSS

O selector de classes seleciona elementos HTML com um atributo de classe específico. Para selecionar elementos com uma classe específica, digita-se um caractere ponto ('.'), seguido do nome da classe.

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

***Figura 3** – O selecionador de classe do CSS (**Fonte:** https://www.w3schools.com/css/css_selectors.asp)*

O selecionador universal do CSS

O selecionador universal (*) seleciona todos os elementos HTML da página.

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

***Figura 4** – O selecionador universal do CSS (**Fonte:** https://www.w3schools.com/css/css_selectors.asp)*

Selecionadores de agrupamento

Se desejável, poderá aplicar-se um estilo a vários selecionadores. Basta separar os selecionadores com uma vírgula, como indicado no exemplo seguinte:

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

***Figura 5** – Selecionadores de agrupamento (**Fonte:** https://www.w3schools.com/css/css_selectors.asp)*

Comentários do CSS

- Os comentários são usados para explicar o código e podem ajudar quando se edita o código-fonte *a posteriori*;
- Os comentários são ignorados pelos navegadores;
- Um comentário do CSS começa com /* e termina com */.

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

Figura 6 *– Comentários do CSS (**Fonte:** https://www.w3schools.com/css/css_comments.asp)*

Cores do CSS

As cores no CSS podem ser especificadas através dos seguintes métodos:

- Cores hexadecimais;
- Cores hexadecimais com transparência;
- Cores RGB;
- Cores RGBA;
- Cores HSL;
- Cores HSLA;
- Nomes de cores pré-definidos/cruzados;
- Com a palavra-chave de cor atual.

Cores hexadecimais

É especificada uma cor hexadecimal com: #RRGGBB, onde os números inteiros hexadecimais RR (vermelho), GG (verde) e BB (azul) especificam os componentes da cor. Todos os valores devem estar compreendidos entre 00 e FF.

Por exemplo, o valor #0000ff é apresentado como azul, porque o componente azul é definido para o seu valor mais alto (ff) e os outros são definidos para 00.

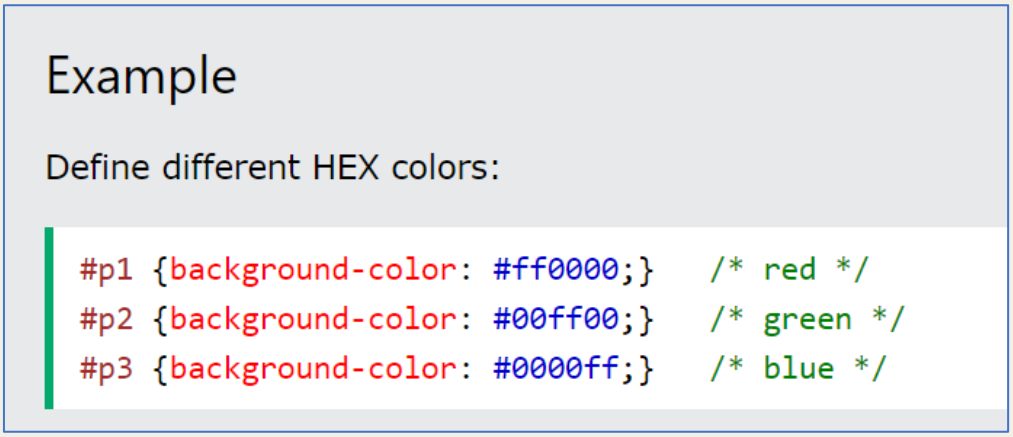


Figura 7 *– Cores hexadecimais (**Fonte:** https://www.w3schools.com/css/css_colors.asp)*

Cores hexadecimais com transparência

É especificada uma cor hexadecimal com: #RRGGBB. Para acrescentar transparência, acrescentar dois dígitos adicionais entre 00 e FF.

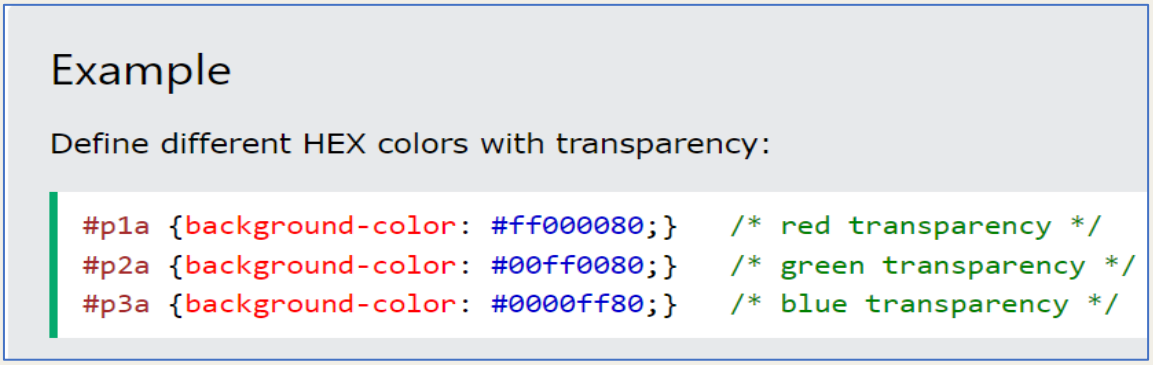


Figura 8 *– Cores hexadecimais com transparência (**Fonte:** https://www.w3schools.com/css/css_colors.asp)*

Cores RGB

Um valor de cor RGB é especificado com a função rgb()#, que tem a seguinte sintaxe:

```
rgb(vermelho, verde, azul)
```

Cada parâmetro (vermelho, verde e azul) define a intensidade da cor e pode ser um número inteiro entre 0 e 255 ou um valor percentual (de 0% a 100%).

Por exemplo, o valor rgb (0,0,255) é apresentado como azul, porque o parâmetro azul é fixado no seu valor mais alto (255) e os outros são fixados em 0.

Além disso, os seguintes valores definem cores iguais: rgb (0,0,255) e rgb (0%,0%,100%).

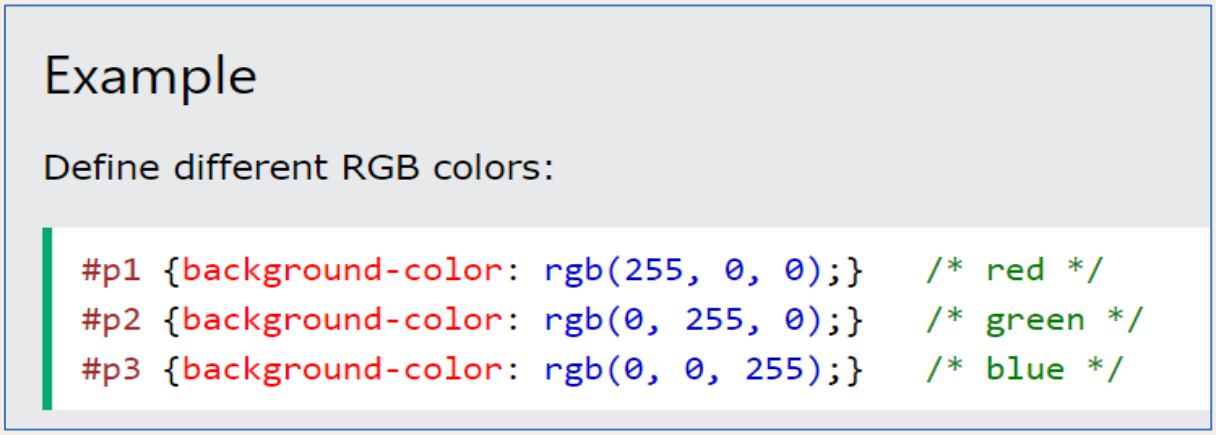


***Figura 9** – Cores RGB (**Fonte:** https://www.w3schools.com/css/css_colors.asp)*

Os valores RGBA das cores

Os valores RGBA das cores são uma extensão dos valores de cor RGB com um canal *alfa* - que especifica a opacidade do objeto.

Os RGBA de uma cor são especificados com a função rgba()#, que tem a seguinte sintaxe:

rgba (vermelho, verde, azul, alfa)

O parâmetro alfa é um número entre 0,0 (totalmente transparente) e 1,0 (totalmente opaco).

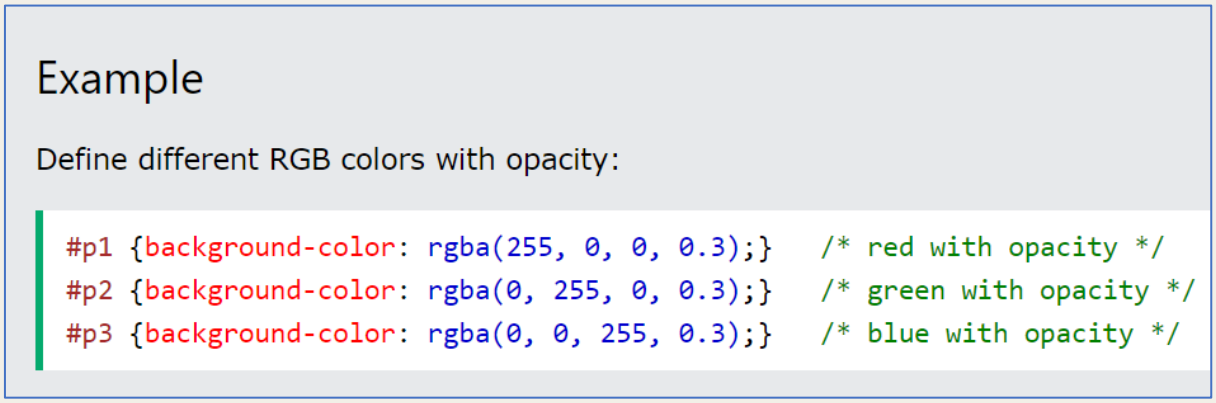


Figura 10 *– RGBA das cores com opacidade (**Fonte:** https://www.w3schools.com/css/css_colors.asp)*

Valores HSL das cores

A sigla HSL (do inglês *hue, saturation and lightness*) remete-nos para tonalidade, saturação e leveza - e transmite uma representação cilíndrico-coordenada de cores.

Um valor de cor HSL é especificado com a função #hsl()##, que possui a seguinte sintaxe:

hsl(hue, saturation, lightness)

A tonalidade é um grau na roda de cor (de 0 a 360) - 0 (ou 360) é vermelho, 120 é verde, 240 é azul. A saturação é um valor percentual; 0% significa uma tonalidade de cinzento e 100% é a cor completa. A leveza é também uma percentagem; 0% é o preto, 100% é o branco.

Example

Define different HSL colors:

```
#p1 {background-color: hsl(120, 100%, 50%);}   /* green */
#p2 {background-color: hsl(120, 100%, 75%);}   /* light green */
#p3 {background-color: hsl(120, 100%, 25%);}   /* dark green */
#p4 {background-color: hsl(120, 60%, 70%);}    /* pastel green */
```

***Figura 11** – HSL das cores (**Fonte:** https://www.w3schools.com/css/css_colors.asp)*

HSLA das cores

Os valores de cor HSLA são uma extensão dos valores de cor HSL com um canal alfa - que especifica a opacidade do objeto.

Um valor de cor HSLA é especificado com a #função hsla()##, #que tem a seguinte sintaxe:

```
hsla(hue, saturation, lightness, alpha)
```

O parâmetro alfa é um número entre 0,0 (totalmente transparente) e 1,0 (totalmente opaco).

Example

Define different HSL colors with opacity:

```
#p1 {background-color: hsla(120, 100%, 50%, 0.3);}   /* green with opacity */
#p2 {background-color: hsla(120, 100%, 75%, 0.3);}   /* light green with opacity */
#p3 {background-color: hsla(120, 100%, 25%, 0.3);}   /* dark green with opacity */
#p4 {background-color: hsla(120, 60%, 70%, 0.3);}    /* pastel green with opacity */
```

Figura 12 *– HSL das cores com opacidade (**Fonte:** https://www.w3schools.com/css/css_colors.asp)*

Nomes de cores pré-definidos/cruzados

140 nomes de cores são predefinidos na especificação de cores HTML e CSS.

Por exemplo: azul, vermelho, coral, castanho, etc.

Example

Define different color names:

```
#p1 {background-color: blue;}
#p2 {background-color: red;}
#p3 {background-color: coral;}
#p4 {background-color: brown;}
```

Figura 13 *– Nomes de cores cruzados (**Fonte:** https://www.w3schools.com/css/css_colors.asp)*

A palavra-chave currentcolor

A palavra-chave *currentcolor* refere-se ao valor da propriedade de cor de um elemento.

Example

The border color of the following element will be blue, because the text color of the element is blue:

```
#myDIV {
  color: blue; /* Blue text color */
  border: 10px solid currentcolor; /* Blue border color */
}
```

Figura 14 *– A palavra-chave currentcolor (**Fonte:** https://www.w3schools.com/colors/colors_currentcolor.asp)*

Fundos do CSS

As propriedades de plano de fundo do CSS são usadas para adicionar efeitos de plano de fundo para elementos. Algumas propriedades de fundo são:

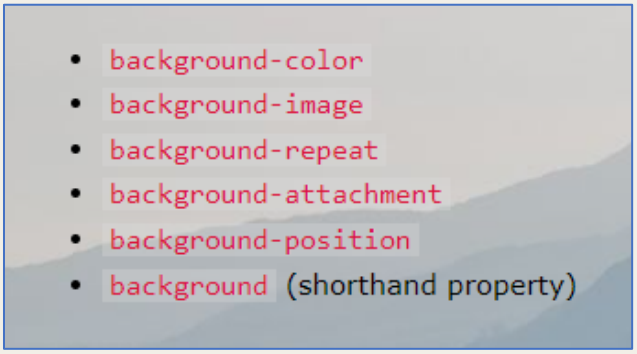


Figura 15 *– Fundo do CSS (**Fonte:** https://www.w3schools.com/css/css_background.asp)*

A propriedade *background-color*

A propriedade background-color especifica a cor de fundo de um elemento.

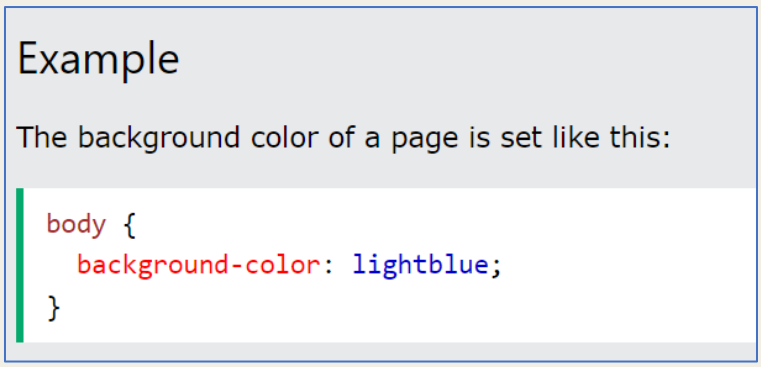


***Figura 16** – Definição da cor de um fundo (**Fonte:**https://www.w3schools.com/css/css_background.asp)*

A propriedade *background-image*

A propriedade *background-image* especifica uma imagem a ser usada como plano de fundo de um elemento.

Por defeito, a imagem é repetida de modo a cobrir todo o elemento.

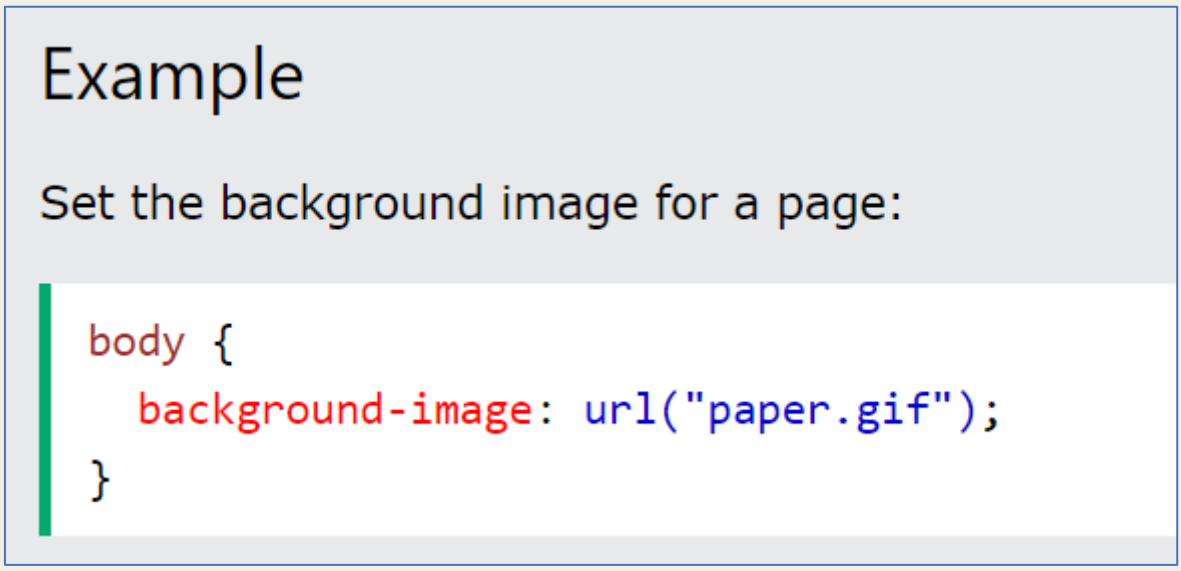


***Figura 17** – A propriedade background-image (**Fonte:** https://www.w3schools.com/css/css_background.asp)*

A propriedade *background-repeat*

Por defeito, a propriedade *background-repeat* repete uma imagem na horizontal e na vertical. Algumas imagens devem ser repetidas apenas horizontalmente ou verticalmente, caso contrário, terão um aspeto estranho.

A propriedade *background-attachment*

A propriedade *background-attachment* especifica se a imagem de fundo deve rolar ao longo da página (efeito "*scroll*") ou ser fixa.

Figura 18 *– A propriedade background-attachment (**Fonte:** https://www.w3schools.com/css/css_background_attachment.asp)*

Example

Specify that the background image should scroll with the rest of the page:

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
  background-attachment: scroll;
}
```

***Figura 19** – A propriedade background-attachment, ex. 2(**Fonte:** https://www.w3schools.com/css/css_background_attachment.asp)*

A propriedade *shorthand*

Para encurtar o código, é também possível especificar todas as propriedades de fundo numa única propriedade. A isto chama-se uma propriedade *shorthand* ("*abreviada*").

Em vez de digitar o seguinte:

```
body {
  background-color: #ffffff;
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
}
```

***Figura 20** – A propriedade shorthand*
*(**Fonte:**https://www.w3schools.com/css/css_background_shorthand.asp)*

Pode ser usada a propriedade *shorthand*:

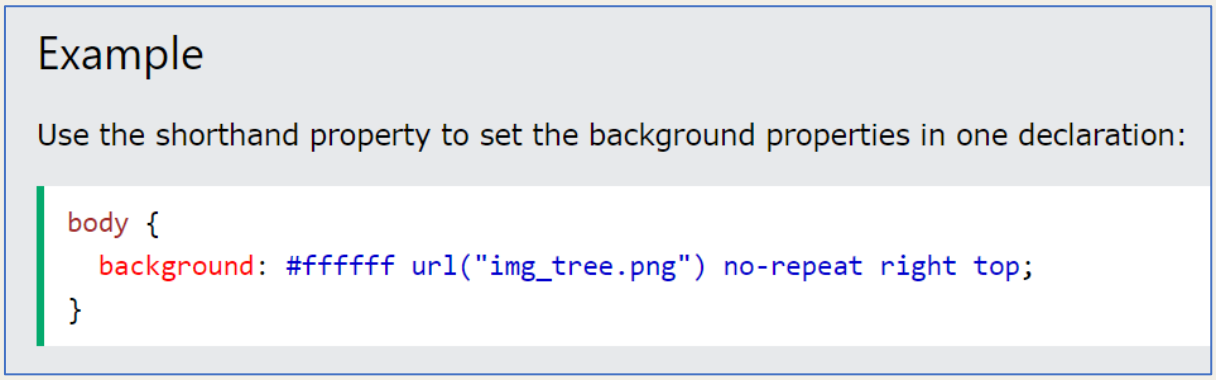


Figura 21 *– A propriedade shorthand*

*(**Fonte:**https://www.w3schools.com/css/css_background_shorthand.asp)*

Limites de margem no CSS

As propriedades de margens no CSS permitem especificar o estilo, largura e cor dos limites de margem de um elemento.

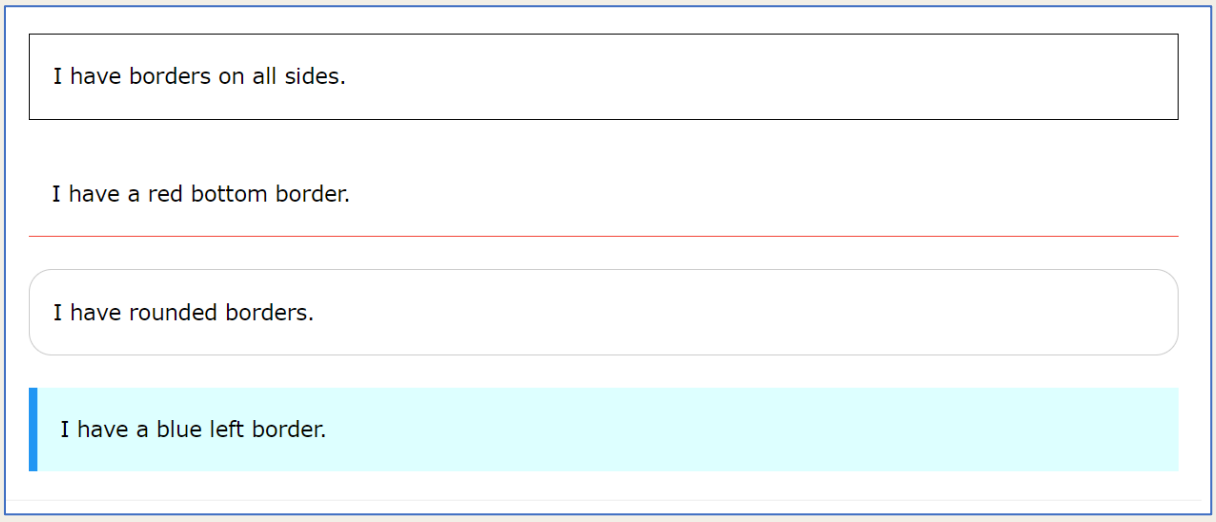


Figura 22 *– Limites de margem no CSS (**Fonte:** https://www.w3schools.com/css/css_border.asp)*

As margens no CSS

As margens são utilizadas para criar espaço em torno de elementos, fora de quaisquer fronteiras definidas.

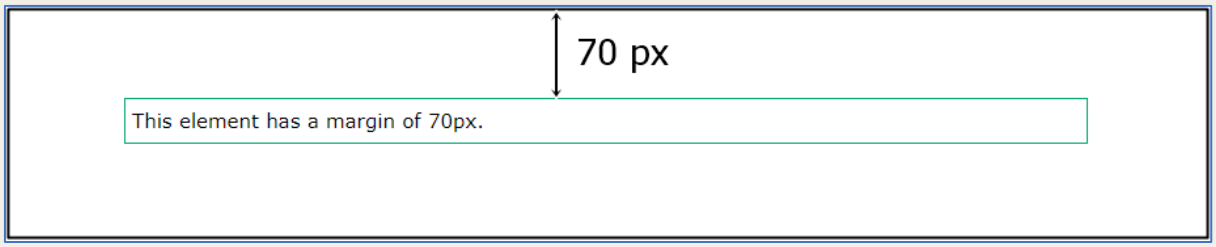


Figura 23 *– Margens no CSS (**Fonte:**https://www.w3schools.com/css/css_margin.asp)*

Com o CSS, há total controlo sobre as margens. Existem propriedades para definir a margem para cada lado de um elemento (superior, direita, inferior, e esquerda).

Example

Set different margins for all four sides of a <p> element:

```
p {
  margin-top: 100px;
  margin-bottom: 100px;
  margin-right: 150px;
  margin-left: 80px;
}
```

Figura 24 *– Margens no CSS (**Fonte:**https://www.w3schools.com/css/css_margin.asp)*

Preenchimento no CSS

O preenchimento é usado para criar espaço ao redor do conteúdo de um elemento, dentro de qualquer limite de margem definido.

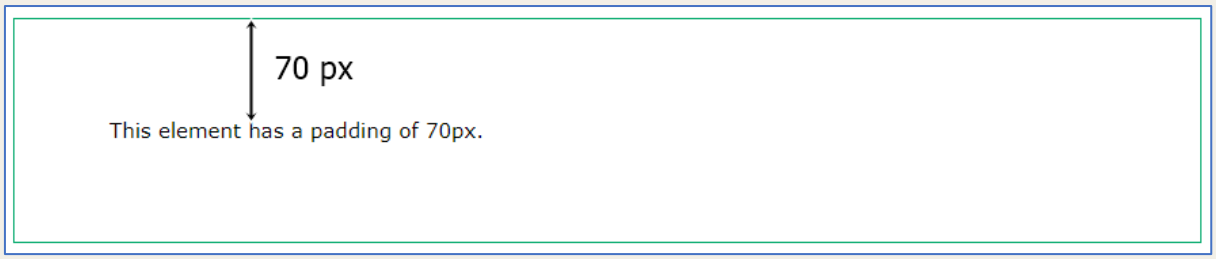


Figura 25 *– Preenchimento no CSS (**Fonte:**https://www.w3schools.com/css/css_padding.asp)*

Com o CSS, há total controlo sobre as margens. Existem propriedades para definir a margem para cada lado de um elemento (superior, direita, inferior, e esquerda).

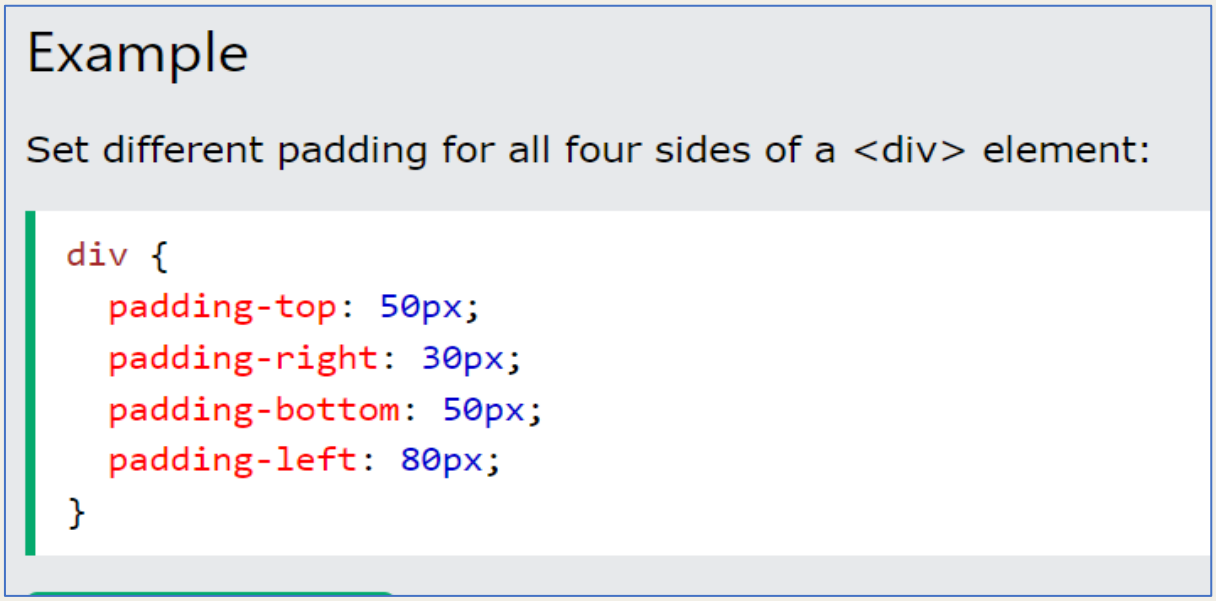


Figura 26 *– Preenchimento no CSS (**Fonte:**https://www.w3schools.com/css/css_padding.asp)*

Altura e largura do CSS

As propriedades de altura e largura são utilizadas para definir a altura e largura de um elemento.
As propriedades de altura e largura não incluem preenchimento, limites de margem ou margens. Ele define a altura/largura da área dentro do preenchimento, limites de margem e margem do elemento.

As propriedades de altura e largura podem ter os seguintes valores:

- *auto* – o padrão. O navegador calcula a altura e a largura;
- comprimento – define a altura/largura em px, cm, etc.;
- % - define a altura/largura em percentagem do bloco que contém;
- inicial – define a altura/largura para o seu valor por defeito;
- *inherit* - altura/largura será *herdada* do seu valor-pai.

Example

Set the height and width of a element:

```
div {
  height: 200px;
  width: 50%;
  background-color: powderblue;
}
```

Figura 27 *– Altura e largura (**Fonte:** https://www.w3schools.com/css/css_dimension.asp)*

O modelo de caixa CSS

Em CSS, o termo "modelo de caixa" é usado quando se fala de design e *layout*.
O modelo de caixa CSS é essencialmente uma caixa que envolve cada elemento HTML. Consiste em: margens, bordas, acolchoamento, e o conteúdo real. A imagem abaixo ilustra o modelo de caixa:

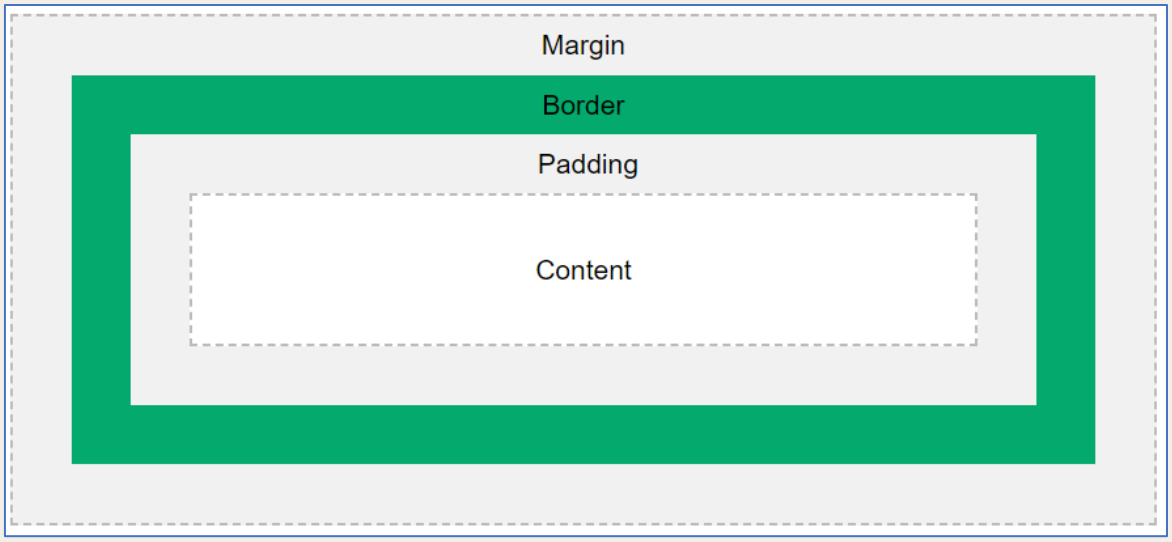


Figura 28 *– Modelo de caixa do CSS (**Fonte:** https://www.w3schools.com/css/css_boxmodel.asp)*

Explicação das diferentes partes:

- Conteúdo – o conteúdo da caixa, onde o texto e as imagens aparecem;
- Preenchimento – limpa uma área ao redor do conteúdo. O forro é transparente;
- Limite da margem/borda – borda que contorna o preenchimento e o conteúdo;
- Margem - área transparente, fora da fronteira.
  O modelo de caixa permite-nos adicionar uma fronteira em torno de elementos, e definir o espaço entre elementos.

Contorno do CSS

Um contorno é uma linha que é traçada em torno de elementos, fora das fronteiras para fazer o elemento "*sobressair*".

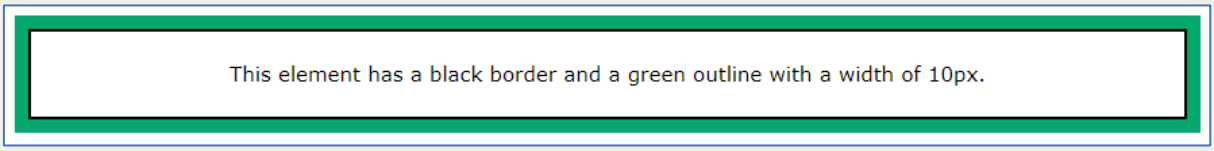


Figura 29 *– Contorno do CSS (**Fonte:** https://www.w3schools.com/css/css_outline.asp)*

Texto em CSS

O CSS tem muitas propriedades de formatação de texto.

Figura 30 *– Texto em CSS (**Fonte:** https://www.w3schools.com/css/css_text.asp)*

- Cor do texto
- Alinhamento do texto
- Decoração do texto
- Transformação do texto
- Espaçamento do texto
- Sombra do texto

Tipos de letra no CSS

- O tipo de letra certo pode ajudar a criar uma identidade forte para um website.
- Usar uma fonte que seja fácil de ler é importante. A fonte agrega valor ao texto. Também é importante escolher a cor e o tamanho do texto corretos para a fonte.

Generic Font Family	Examples of Font Names
Serif	Times New Roman Georgia Garamond
Sans-serif	Arial Verdana Helvetica
Monospace	Courier New Lucida Console Monaco
Cursive	Brush Script MT Lucida Handwriting
Fantasy	Copperplate Papyrus

Figura 31 *– CSS Fonts (**Fonte:** https://www.w3schools.com/css/css_font.asp)*

Ícones do CSS

- A forma mais simples de adicionar um ícone à página HTML, é com uma biblioteca de ícones, tal como o tipo de letra “Awesome”.
- Adicionar o nome da classe do ícone especificado a qualquer elemento HTML em linha (como <i> ou).
- Todos os ícones nas bibliotecas de ícones abaixo são vetores escaláveis que podem ser personalizados com CSS.

***Figura 32** – Ícones do CSS (**Fonte:** https://www.w3schools.com/css/css_icons.asp)*

Hiperligações no CSS

As hiperligações (ou "links") podem ser estilizadas com qualquer propriedade CSS (por exemplo, cor, font-family, fundo, etc,).

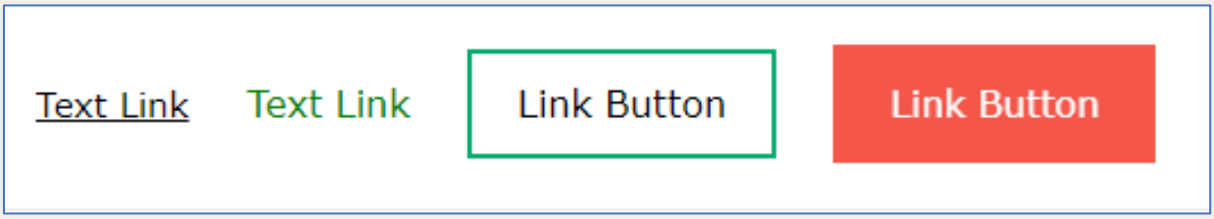


***Figura 33** – Links ou hiperligações no CSS (**Fonte:** https://www.w3schools.com/css/css_link.asp)*

Além disso, as hiperligações podem ser estilizadas de forma diferente dependendo do estado em que se encontram.

Os quatro estados dos *links*/hiperligações são:

- a:link - um link normal, não visitado;
- a:visited - um link que o utilizador visitou;
- a:hover - um link que surge quando o utilizador passa o cursor por cima dele;
- a:active - um link no momento em que é clicado.

Listas do CSS

As propriedades delista CSS permitem:

- Definir marcadores de itens de lista diferentes para listas ordenadas;
- Definir marcadores de itens de lista diferentes para listas não ordenadas;
- Definir uma imagem como marcador de item da lista;

- Adicionar cores de fundo a listas e itens de lista.

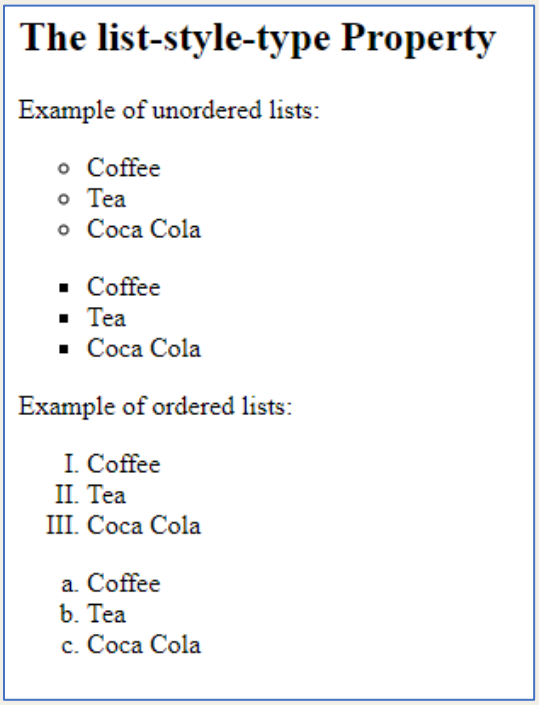


***Figura 34** – Listas do CSS (**Fonte:** https://www.w3schools.com/css/css_list.asp)*

Propriedades de disposição no CSS

- A propriedade de exibição especifica se/como é exibido um elemento.
- Cada elemento HTML tem um valor de exibição padrão dependendo do tipo de elemento que é. O valor de exibição padrão para a maioria dos elementos é *block* ou *inline*.

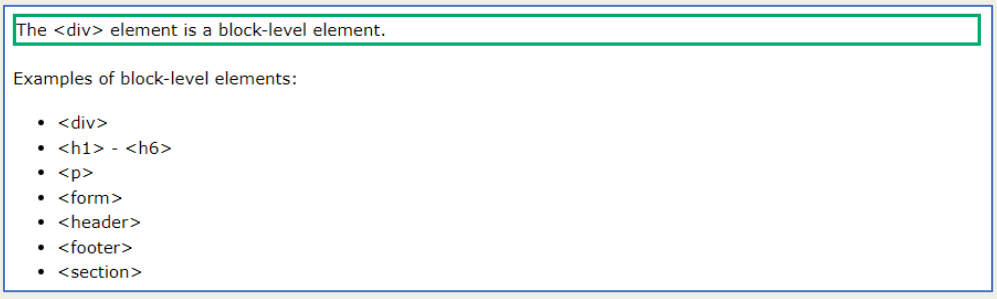


***Figura 35** – Disposição no CSS (**Fonte:** https://www.w3schools.com/css/css_display_visibility.asp)*

Tabelas no CSS

O aspeto de uma tabela HTML pode ser fortemente melhorado com o CSS:

Company	Contact	Country
Alfreds Futterkiste	Maria Anders	Germany
Berglunds snabbköp	Christina Berglund	Sweden
Centro comercial Moctezuma	Francisco Chang	Mexico
Ernst Handel	Roland Mendel	Austria
Island Trading	Helen Bennett	UK
Königlich Essen	Philip Cramer	Germany
Laughing Bacchus Winecellars	Yoshi Tannamuri	Canada
Magazzini Alimentari Riuniti	Giovanni Rovelli	Italy

Figura 36 *– Tabelas no CSS (**Fonte:** https://www.w3schools.com/css/css_table.asp)*

A propriedade *max-width*

- O problema com a função , mencionado anteriormente, ocorre quando a janela do navegador é menor do que a largura do elemento. O navegador adiciona então uma barra de deslocamento horizontal à página. O navegador adiciona então uma barra de deslocamento horizontal à página.
- Usar *max-width* em vez disso, nessa situação, melhorará o manuseio de janelas pequenas pelo navegador. Isto é importante ao tornar um site utilizável em dispositivos de tamanho reduzido.

Figura 37 *– A propriedade maxwidth (**Fonte:** https://www.w3schools.com/css/css_max-width.asp)*

O posicionamento no CSS

A propriedade de posição especifica o tipo de método de posicionamento utilizado para um elemento.

Existem cinco valores de posição diferentes:

- estático
- relativo
- fixo
- absoluto
- "*sticky*"

A propriedade *z-index*

A propriedade CSS Z-index especifica a ordem de empilhamento de um elemento (que elemento deve ser colocado à frente, ou atrás dos outros). Um elemento pode ter uma ordem de empilhamento positiva ou negativa, como se pode verificar abaixo:

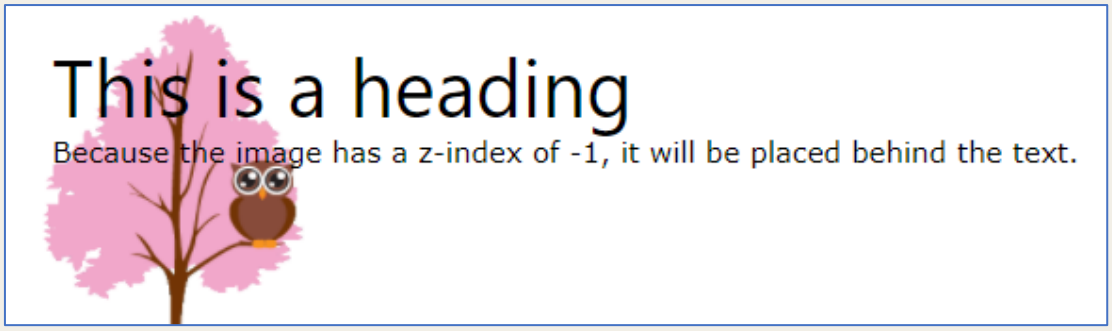


Figura 38 *– A propriedade Z-index (**Fonte:** https://www.w3schools.com/css/css_z-index.asp)*

Transbordamento (ou *overflow*) no CSS

A propriedade de transbordo ("overflow") especifica se se deve cortar o conteúdo ou adicionar barras de rolagem ("scrollbars") quando o conteúdo de um elemento é demasiado grande para caber na área determinada.

A propriedade de *overflow* tem os seguintes valores:

- visível

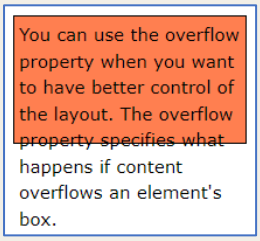


Figura 39 *– Valor visível na propriedade overflow (**Fonte:** https://www.w3schools.com/css/css_overflow.asp)*

- escondido

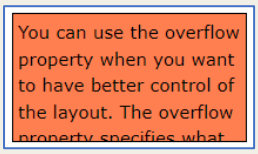


Figura 40 *– Valor escondido na propriedade overflow (**Fonte:** https://www.w3schools.com/css/css_overflow.asp)*

- “scroll”

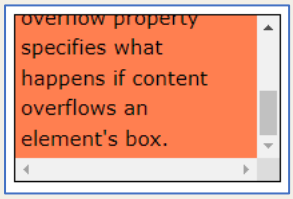


***Figura 41** – Valor “scroll” na propriedade overflow (**Fonte:** https://www.w3schools.com/css/css_overflow.asp)*

- auto

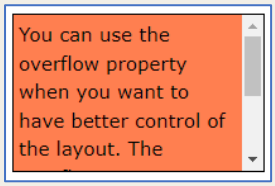


***Figura 42** – Valor “auto” na propriedade overflow (**Fonte:** https://www.w3schools.com/css/css_overflow.asp)*

A propriedade *float*

A propriedade float é utilizada para posicionamento e formatação do conteúdo.

Exemplo: **float: right**

O exemplo seguinte especifica que uma imagem deve flutuar para a **direita**.

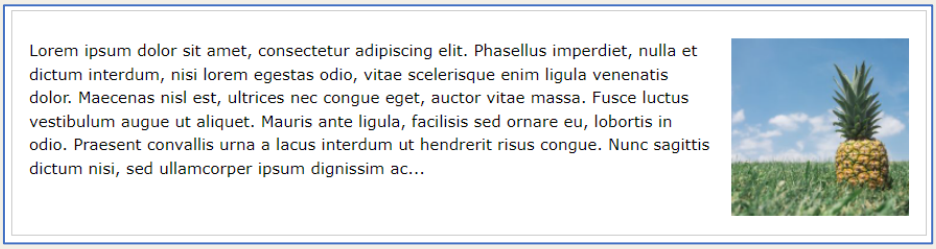


***Figura 43** – Flutuar para a direita no CSS (**Fonte:** https://www.w3schools.com/css/css_float.asp)*

O exemplo seguinte especifica que uma imagem deve flutuar para a **esquerda** num texto:

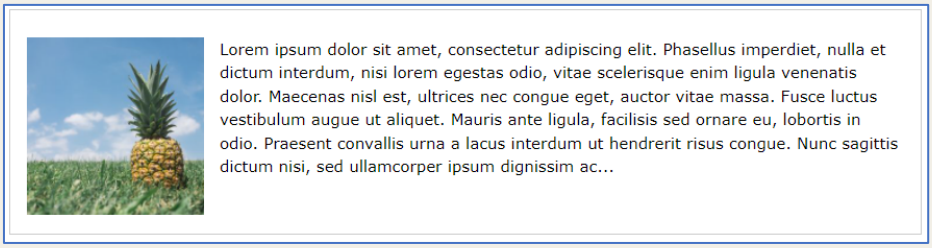


***Figura 44** – Uso da propriedade float-left (**Fonte:** https://www.w3schools.com/css/css_float.asp)*

Propriedade *inline-block*

- *Display: inline-block* permite definir uma largura e altura no elemento;
- Com *display: inline-block*, as margens/preenchimentos superiores e inferiores são respeitadas;
- *Display: inline-block* não adiciona uma quebra de linha após o elemento, então o elemento pode ficar ao lado de outros elementos.

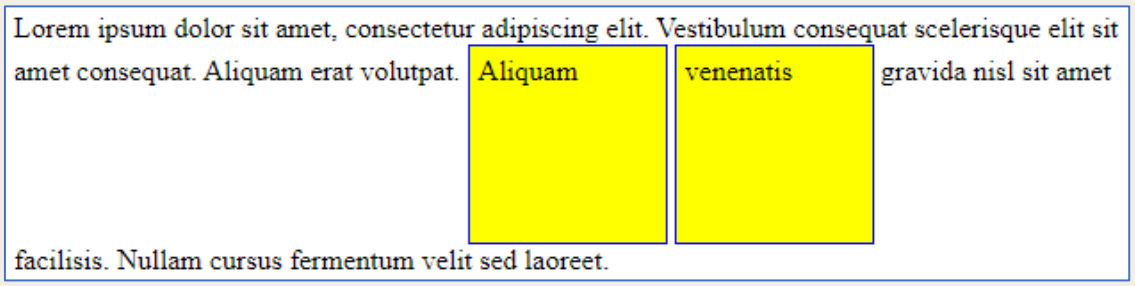


***Figura 45** – A propriedade Inline-block (**Fonte:** https://www.w3schools.com/css/css_inline-block.asp)*

A propriedade *align*

Para centralizar horizontalmente um elemento de bloco (como), usar *margin: auto.* A regulação da largura do elemento impedirá que este se estique até ao limite marginal do seu recipiente.

O elemento ocupará então a largura especificada, e o espaço restante será dividido igualmente entre as duas margens:

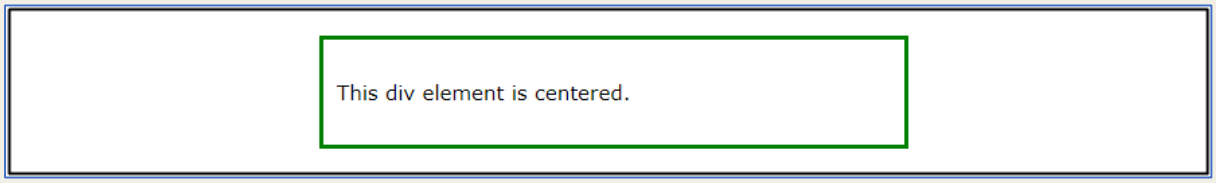


***Figura 46** – Alinhar elementos em CSS (**Fonte:** https://www.w3schools.com/css/css_align.asp)*

Combinadores do CSS

Um seletor CSS pode conter mais de um seletor simples. Entre os seletores simples, podemos incluir um combinador. Existem quatro combinadores diferentes no CSS:

- seletor descendente (espaço) -> ex. div p {background-color: yellow}
- seletor filho (>) -> ex. div > p {background-color: yellow}
- seletor de irmãos adjacente (+) -> ex. div + p {background-color: yellow}
- seletor geral de irmãos (~) -> ex. div ~ p {background-color: yellow}

Pseudo-classes de CSS

Uma pseudo-classe é utilizada para definir um estado especial de um elemento. Por exemplo, pode ser usado para:

- Estilizar um elemento quando um utilizador passa o cursor nele;
- Estilizar hiperligações visitadas e não visitadas de forma diferente;
- Estilizar um elemento quando ele estiver em foco.

```css
/* unvisited link */
a:link {
  color: #FF0000;
}

/* visited link */
a:visited {
  color: #00FF00;
}

/* mouse over link */
a:hover {
  color: #FF00FF;
}

/* selected link */
a:active {
  color: #0000FF;
}
```

Figura 47 *– Pseudo-Classes no CSS (**Fonte:** https://www.w3schools.com/css/css_pseudo_classes.asp)*

Pseudo-elementos no CSS

Um pseudo-elemento CSS é usado para estilizar partes especificadas de um elemento.

Por exemplo, pode ser usado para:

- Estilizar a primeira letra, ou linha, de um elemento;
- Inserir conteúdo antes, ou depois, do conteúdo de um elemento.

```css
p::first-line {
  color: #ff0000;
  font-variant: small-caps;
}
```

Figura 48 *– Pseudo element (**Fonte:** https://www.w3schools.com/css/css_pseudo_elements.asp)*

O pseudo-elemento de primeira linha é utilizado para acrescentar um estilo especial à primeira linha de um texto.

Opacidade no CSS

A propriedade de opacidade especifica a opacidade/transparência de um elemento. A propriedade de opacidade pode ter um valor de 0,0 - 1,0. Quanto mais baixo for o valor, mais transparente será o seu elemento.

Figura 49 *– Opacidade no CSS (**Fonte:** https://www.w3schools.com/css/css_image_transparency.asp)*

A barra de navegação do CSS

Com o CSS pode transformar menus HTML aborrecidos em barras de navegação atrativas.

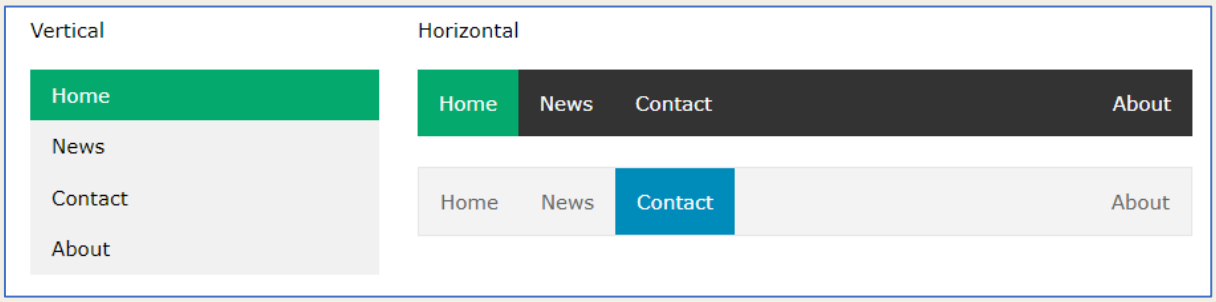


Figura 50 *– A barra de navegação do CSS (**Fonte:** https://www.w3schools.com/css/css_navbar.asp)*

Menus em cascata do CSS

Com o CSS, é possível transformar menus HTML aborrecidos em barras de navegação atrativas.

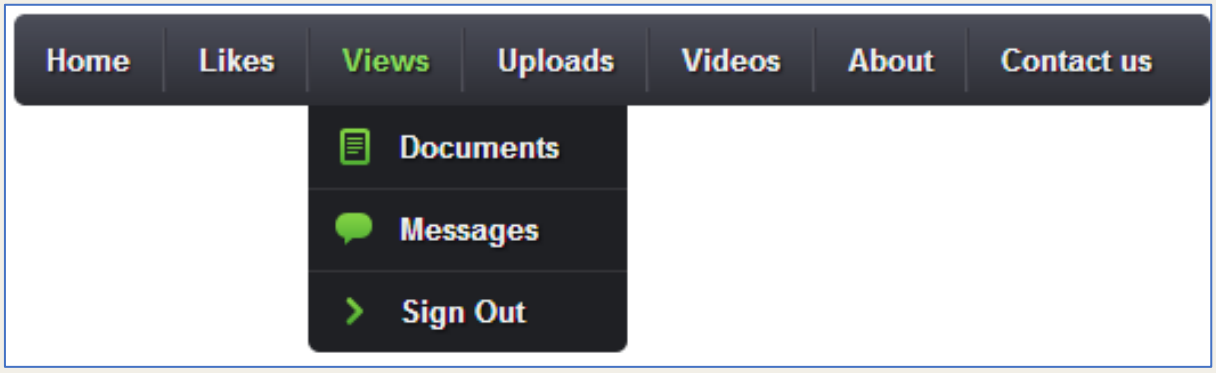


Figura 51 *– Menu em cascata (**Fonte:** https://www.w3schools.com/css/css_dropdowns.asp)*

Galeria de Imagens do CSS

O CSS pode ser criado para criar uma galeria de imagens.

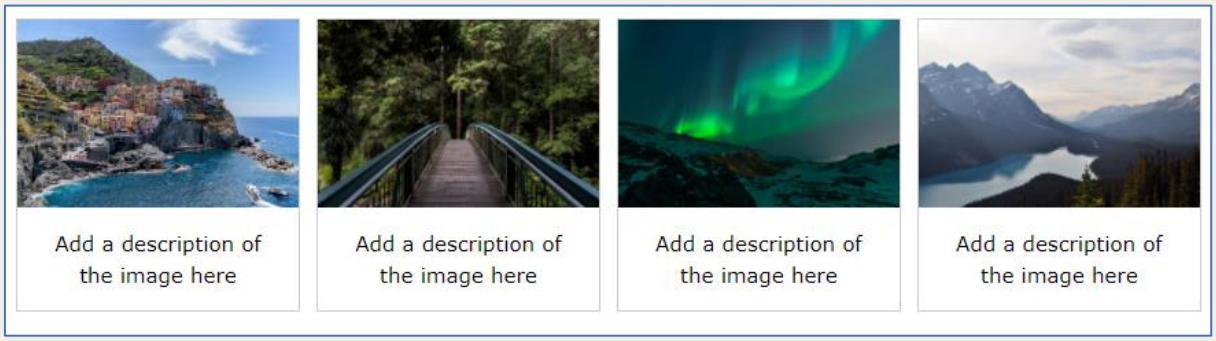


Figura 52 *– Galeria de Imagens (**Fonte:** https://www.w3schools.com/css/css_image_gallery.asp)*

Sprites de imagem do CSS

- Um sprite de imagem é uma coleção de imagens colocadas numa única imagem.
- Uma página web com muitas imagens pode demorar muito tempo a carregar e gera múltiplos pedidos de servidor.
- A utilização de sprites de imagem reduzirá o número de pedidos de servidor e poupará a largura de banda.

Figura 53 *– Sprites de imagem (**Fonte:** https://www.w3schools.com/css/css_image_sprites.asp)*

Selecionadores de atributos do CSS

O seletor [attribute] é usado para selecionar elementos com um atributo especificado.

- O exemplo seguinte seleciona todos os elementos com um atributo alvo:

```
a[target] {
  background-color: yellow;
}
```

Figura 54 *– Selecionador de atributos (**Fonte:** https://www.w3schools.com/css/css_attribute_selectors.asp)*

Formulários do CSS

O aspeto de um formulário HTML pode ser fortemente incrementado com o CSS:

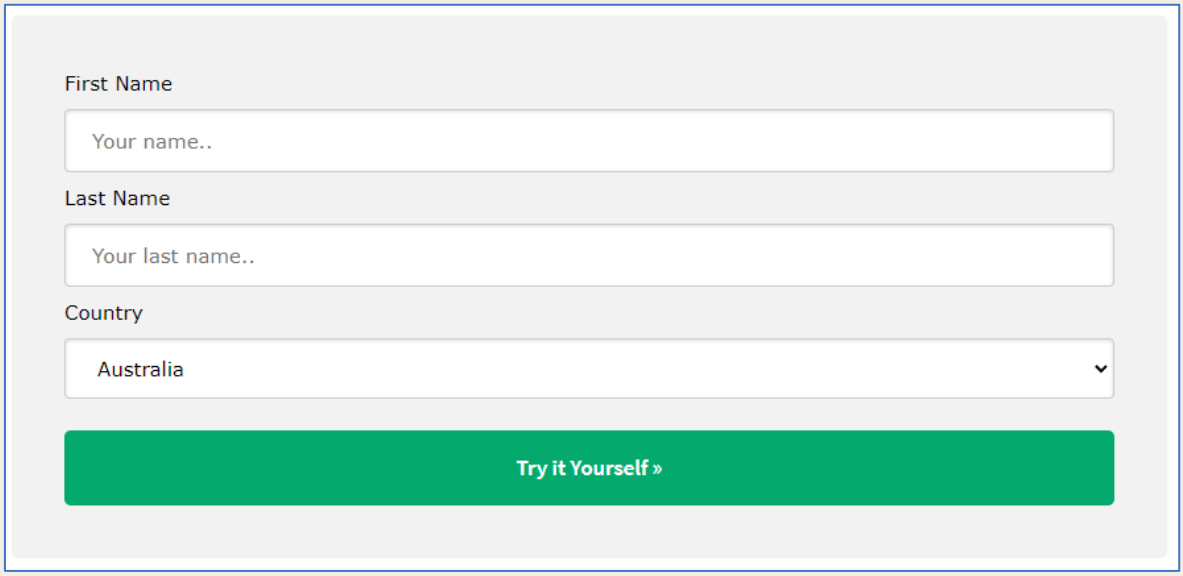


***Figura 55** – Formulários do CSS (**Fonte:** https://www.w3schools.com/css/css_form.asp)*

Contadores do CSS

Os contadores do CSS são "variáveis" mantidas pelo CSS cujos valores podem ser incrementados por regras do próprio CSS (para rastrear quantas vezes são utilizados). Os contadores permitem ajustar a aparência do conteúdo com base na sua colocação no documento.

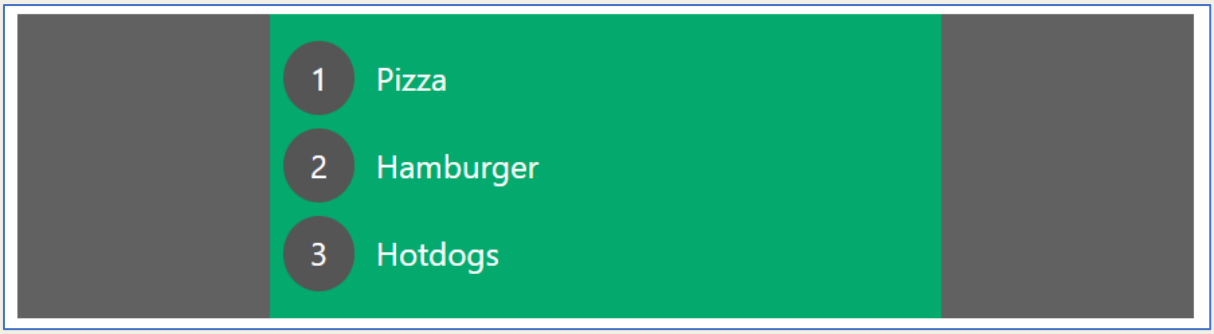


Figura 56 *– Contador do CSS (**Fonte:** https://www.w3schools.com/css/css_counters.asp)*

Layout do website do CSS

Um site geralmente é dividido em cabeçalhos, menus, conteúdo e rodapé:

Figura 57 *– Layout de um website (**Fonte:** https://www.w3schools.com/css/css_website_layout.asp)*

Unidades do CSS

O CSS tem várias unidades diferentes para expressar um comprimento. Muitas propriedades do CSS tomam valores de "comprimento", tais como largura, margem, preenchimento, tamanho da fonte, etc.

Unit	Description
cm	centimeters
mm	millimeters
in	inches (1in = 96px = 2.54cm)
px *	pixels (1px = 1/96th of 1in)
pt	points (1pt = 1/72 of 1in)
pc	picas (1pc = 12 pt)

Figura 58 *– Units (**Fonte:** https://www.w3schools.com/css/css_units.asp)*

Especificidade no CSS

Se houver duas ou mais regras CSS que apontem para o mesmo elemento, o selecionador com o maior valor de especificidade irá prevalecer, e a sua declaração de estilo será aplicada a esse elemento HTML. Devemos olhar para a especificidade como uma pontuação que determina qual a declaração de estilo que, em última análise, é aplicado a um elemento.

```
<html>
<head>
  <style>
    p {color: red;}
  </style>
</head>
<body>

<p>Hello World!</p>

</body>
</html>
```

Figura 59 *– Especificidade no CSS (**Fonte:** https://www.w3schools.com/css/css_specificity.asp)*

Neste exemplo, utilizámos o elemento "p" como selector e especificámos uma cor vermelha para este elemento. O texto será vermelho.

A regra “!important” no CSS

A regra "!important" no CSS é usada para acrescentar mais importância a uma propriedade/valor do que o normal. Na verdade, se a regra "!important" for usada, esta substituirá **todas** as regras de estilo anteriores para essa propriedade específica nesse elemento.

```
#myid {
  background-color: blue;
}

.myclass {
  background-color: gray;
}

p {
  background-color: red !important;
}
```

Figura 60 *– A regra !important (**Fonte:** https://www.w3schools.com/css/css_important.asp)*

A função math no CSS

As funções matemáticas do CSS permitem que expressões matemáticas sejam usadas como valores de propriedade. Algumas funções matemáticas são: funções calc(), max() e min().

Exemplo: o uso de calc() para calcular a largura de um elemento.

```
#div1 {
  position: absolute;
  left: 50px;
  width: calc(100% - 100px);
  border: 1px solid black;
  background-color: yellow;
  padding: 5px;
}
```

Figura 61 *– Função matemática no CSS (**Fonte:** https://www.w3schools.com/css/css_math_functions.asp)*

3.2. CSS – nível avançado

Cantos arredondados do CSS#

A propriedade border-radius define o raio dos cantos de um elemento. Esta propriedade permite adicionar cantos arredondados aos elementos.

Figura 62 *– Cantos arredondados no CSS, ex.1 (**Fonte:** https://www.w3schools.com/css/css3_borders.asp)*

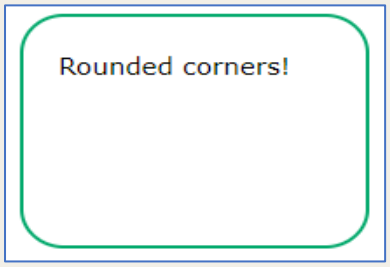


Figura 63 *– Cantos arredondados no CSS, ex.2 (**Fonte:** https://www.w3schools.com/css/css3_borders.asp)*

A propriedade *border-image*

A propriedade *border-image* permite especificar uma imagem a ser utilizada no lugar do limite de margem convencional em torno de um elemento.

Esta propriedade tem três pontos a ter em conta:

- A imagem a ser usada como limite de margem;
- O local onde se faz o corte da imagem;
- Definir se as secções intermédias devem ser repetidas ou esticadas.

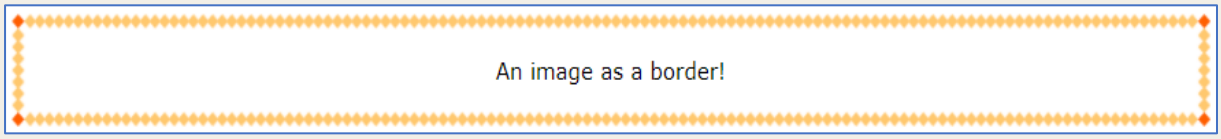


Figura 64 *– A propriedade border-image (**Fonte:** https://www.w3schools.com/css/css3_border_images.asp)*

Fundos do CSS

CSS permite adicionar várias imagens de fundo para um elemento, através da propriedade background-image.

O exemplo seguinte tem duas imagens de fundo, a primeira imagem é uma flor (alinhada para baixo e para a direita) e a segunda imagem é um fundo de papel (alinhada para o canto superior esquerdo).

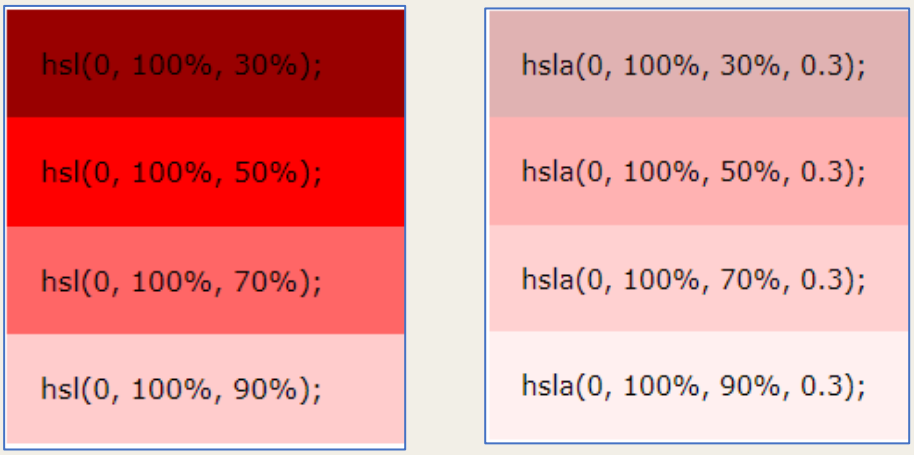
```
#example1 {
  background-image: url(img_flwr.gif), url(paper.gif);
  background-position: right bottom, left top;
  background-repeat: no-repeat, repeat;
}
```

***Figura 65** – A propriedade background-image no CSS (**Fonte:** https://www.w3schools.com/css/css3_backgrounds.asp)*

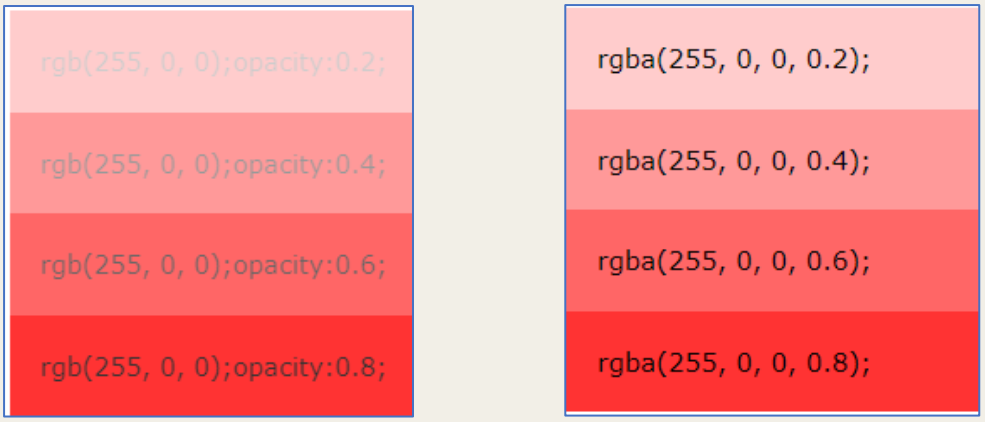
***Figura 66** – Imagem de fundo (**Fonte:** https://www.w3schools.com/css/css3_backgrounds.asp)*

Cores no CSS

O CSS suporta mais de 140 nomes de cores, valores HEX, valores RGB, valores RGBA, valores HSL, valores HSLA e opacidade.



Figuras 67 e 68 *– Parâmetros HSL e HSLA do CSS (**Fonte:** https://www.w3schools.com/css/css3_colors.asp)*



Figuras 69 e 70 *– Parâmetros RGB e RGBA do CSS (**Fonte:** https://www.w3schools.com/css/css3_colors.asp)*

Palavras-chave relativas a cores no CSS

Esta secção propõe-se a explicar de que tratam as palavras-chave *transparent, currentcolor* e inherit.

A palavra-chave *transparent* é utilizada para tornar uma cor transparente. Esta é frequentemente utilizada para fazer uma cor de fundo transparente para um elemento.

A palavra-chave *currentcolor* é como uma variável que detém o valor atual da propriedade da cor de um elemento. Esta palavra-chave pode ser útil se quiser que uma cor específica seja consistente num elemento ou numa página.

A palavra-chave inherit especifica que uma propriedade deve herdar o seu valor do seu "elemento-pai". A palavra-chave *inherit* pode ser usada para qualquer propriedade CSS e em qualquer elemento HTML.

Gradiência no CSS

Os gradientes CSS permitem mostrar transições suaves entre duas ou mais cores especificadas.

CSS define três tipos de gradientes:

- Gradientes Lineares (abaixo/acima/à esquerda/à direita/na diagonal);
- Gradientes radiais (definidos pelo seu centro);
- Gradientes Cónicos (girados em torno de um ponto central).

Figura 71 *– Gradiência nos fundos do CSS (**Fonte:** https://www.w3schools.com/css/css3_gradients.asp)*

Sombras no CSS

Com o CSS, é possível adicionar sombras ao texto e aos elementos.
Falamos das seguintes propriedades:

- Text-shadow
- Box-shadow

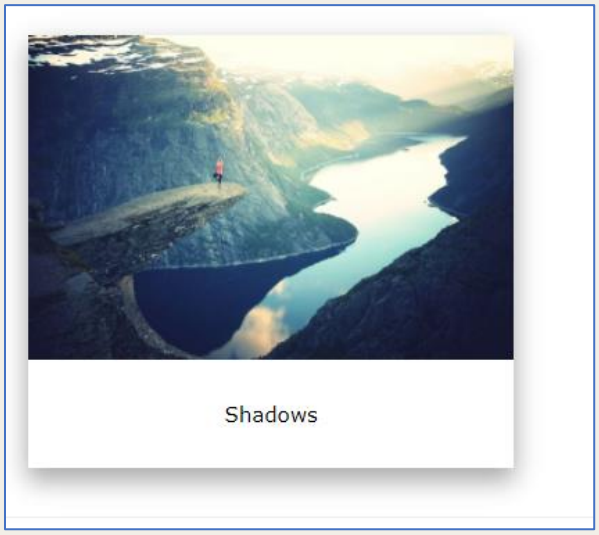


***Figura 72** – Exemplo de sombras numa caixa (**Fonte:** https://www.w3schools.com/css/css3_shadows.asp)*

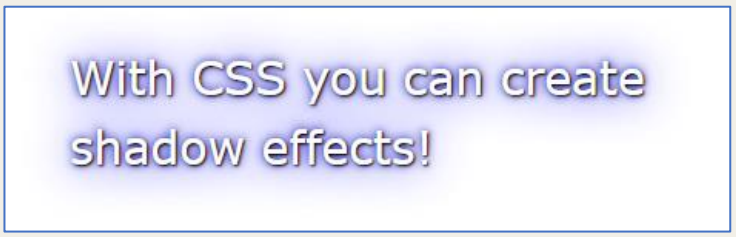


***Figura 73** – Exemplo de sombras em texto (**Fonte:** https://www.w3schools.com/css/css3_shadows.asp)*

Efeitos de texto no CSS

Neste capítulo, aprenderás sobre as funcionalidades seguintes: text-overflow, word-wrap, word-break, writing-mode.

- A propriedade text-overflow especifica como o conteúdo transbordado que não é exibido deve ser sinalizado ao utilizador.
- A propriedade CSS word-wrap permite que palavras longas possam ser quebradas e colocadas na linha seguinte.
- A propriedade CSS word-break especifica as regras de quebra de linha.

- A propriedade writing-mode especifica se as linhas de texto são dispostas horizontalmente ou verticalmente.

Tipos de letra da web do CSS

Os tipos de letra da web permitem aos *web designers* utilizar tipos de letra que não estão instalados no computador do utilizador. Quando tiver encontrado/comprado a fonte que deseja utilizar, basta incluir o ficheiro da fonte no seu servidor web, e este será automaticamente descarregado para o utilizador quando necessário. As fontes a utilizar são definidas com a regra font-face.

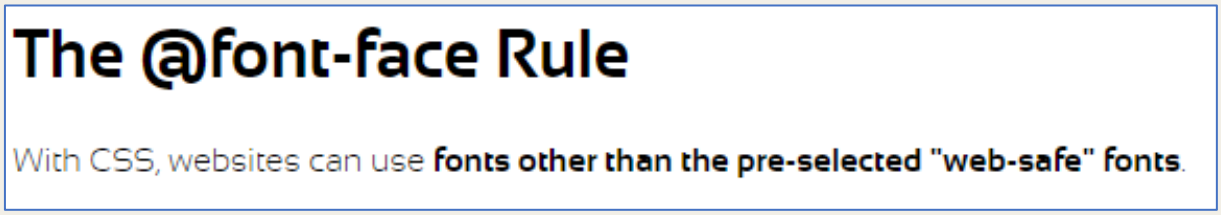


Figura 74 *– A regra font-face (**Fonte:** https://www.w3schools.com/css/css3_fonts.asp)*

Transformações em 2D do CSS

As transformações CSS permitem mover, girar, dimensionar e inclinar elementos.

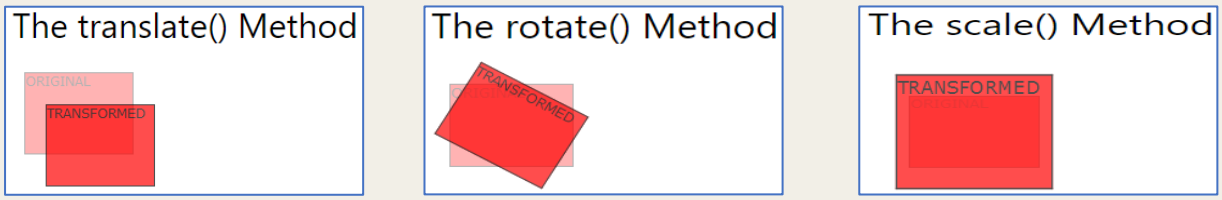


Figura 75 *– Transformações em 2D do CSS e os seus 3 métodos – translate(), rotate() e scale() (**Fonte:** https://www.w3schools.com/css/css3_2dtransforms.asp)*

Transformações em 3D do CSS

Com a propriedade de transformação CSS, é possível utilizar os seguintes métodos de transformação 3D:

- RotateX()
- RotateY()
- RotateZ()

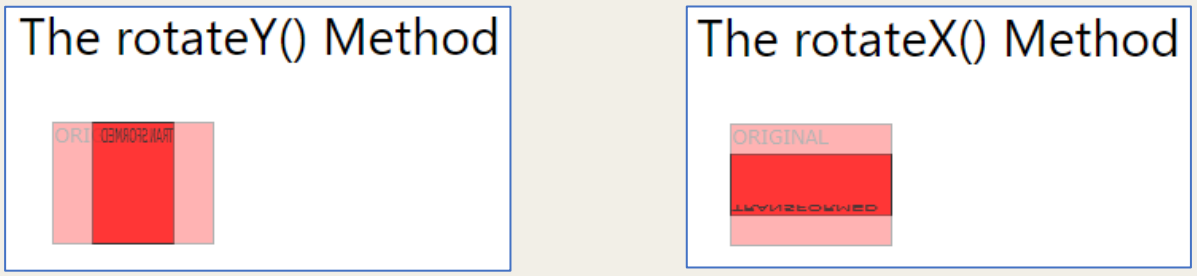


***Figura 76** – Métodos de transformação 3D no CSS (**Fonte:** https://www.w3schools.com/css/css3_3dtransforms.asp)*

Transições no CSS

As transições CSS permitem que se altere os valores das propriedades sem problemas, durante um determinado período.

Para criar um efeito de transição, é necessário especificar duas coisas:

- a propriedade CSS a que se pretende acrescentar um efeito;
- a duração do efeito.

O exemplo seguinte mostra um elemento vermelho de 100px * 100px. O elemento também especificou um efeito de transição para a propriedade da largura, com uma duração de 2 segundos:

```
div {
  width: 100px;
  height: 100px;
  background: red;
  transition: width 2s;
}
```

Figura 77 *– Transições do CSS (**Fonte:** https://www.w3schools.com/css/css3_transitions.asp)*

Animações do CSS

Uma animação permite que um elemento mude gradualmente de um estilo para outro. É possível alterar quantas propriedades CSS forem desejadas, as vezes que forem necessárias. Para utilizar animação CSS, é necessário primeiro especificar alguns quadros-chave para a animação. Os quadros-chave contêm os estilos que o elemento terá em determinados momentos.

Estas são as principais propriedades da animação:

- `@keyframes`
- `animation-name`
- `animation-duration`
- `animation-delay`
- `animation-iteration-count`
- `animation-direction`
- `animation-timing-function`
- `animation-fill-mode`
- `animation`

Figura 78 *– Animações do CSS (**Fonte:** https://www.w3schools.com/css/css3_animations.asp)*

Tooltips do CSS

- Uma *tooltip* é frequentemente utilizada para especificar informação extra sobre algo quando o utilizador move o ponteiro do rato sobre um elemento:

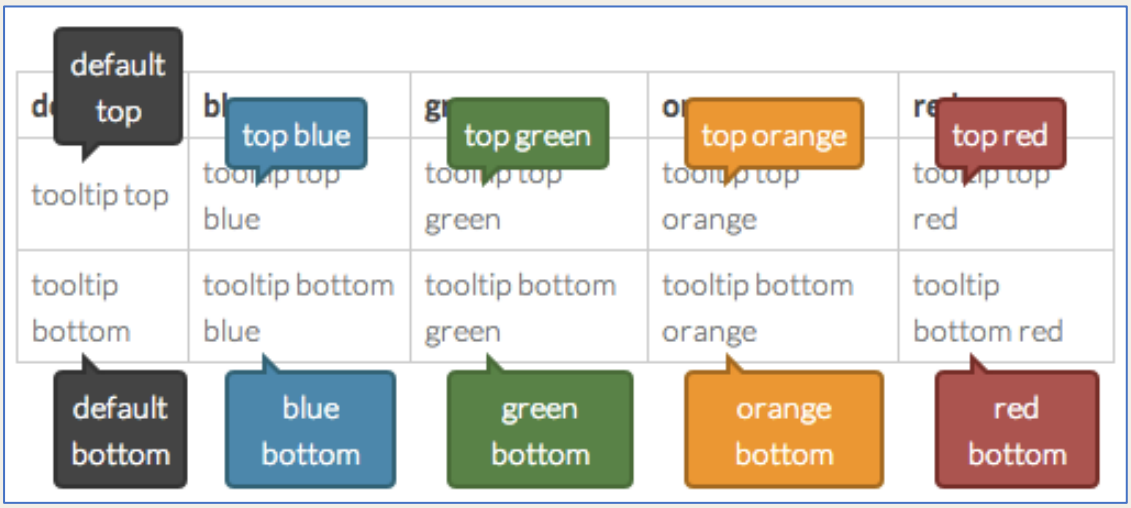


Figura 79 *– Tooltip do CSS (**Fonte:** https://www.w3schools.com/css/css3_animations.asp)*

Estilizar imagens no CSS

A utilização do CSS para estilizar imagens permite especificar uniformemente como as imagens devem aparecer no website com apenas alguns conjuntos de regras, tais como adicionar uma margem, mudar a forma e o tamanho da imagem, etc.

Figura 80 *– Imagens estilizadas a partir do CSS (**Fonte:** https://www.w3schools.com/css/css3_image_reflection.asp)*

A propriedade box-reflect do CS

A propriedade box-reflect é utilizada para criar um reflexo de imagem.O valor da propriedade box-reflect pode ser: abaixo, acima, esquerda ou direita.

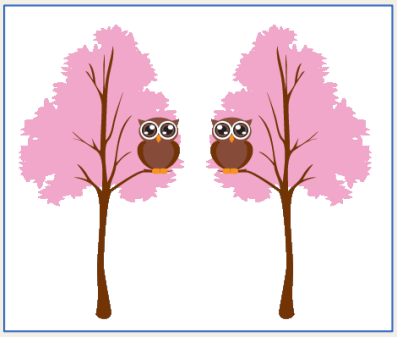


Figura 81 *– Image reflection (**Fonte:** https://www.w3schools.com/css/css3_image_reflection.asp)*

A propriedade object-fit do CSS

A propriedade do CSS é utilizada para especificar como uma <img> ou <video> deve ser redimensionado para caber no seu recipiente. Essa propriedade informa o conteúdo para preencher o recipiente de várias maneiras; como "preservar essa proporção" ou "esticar e ocupar o máximo de espaço possível".

Veja-se a seguinte imagem de Paris. Esta imagem tem 400 pixels de largura e 300 pixels de altura:

***Figura 82** – A propriedade object-fit (**Fonte:** https://www.w3schools.com/css/css3_object-fit.asp)*

A propriedade object-position do CSS

Digamos que a parte da imagem que é mostrada não está posicionada como desejamos. Para posicionar a imagem, utilizaremos a propriedade object-position. Aqui utilizaremos a propriedade object-position para posicionar a imagem de modo a que o grande edifício antigo esteja no centro:

***Figura 83** – A propriedade object-position (**Fonte:** https://www.w3schools.com/css/css3_object-position.asp)*

Propriedade de máscara no CSS

Com a máscara CSS cria-se uma camada de máscara para colocar sobre um elemento para esconder porções do elemento, seja de forma parcial ou totalmente. A propriedade mask-image do CSS especifica uma imagem de camada de máscara.

A imagem da camada de máscara pode ser uma imagem PNG, uma imagem SVG, um gradiente CSS ou um elemento SVG.

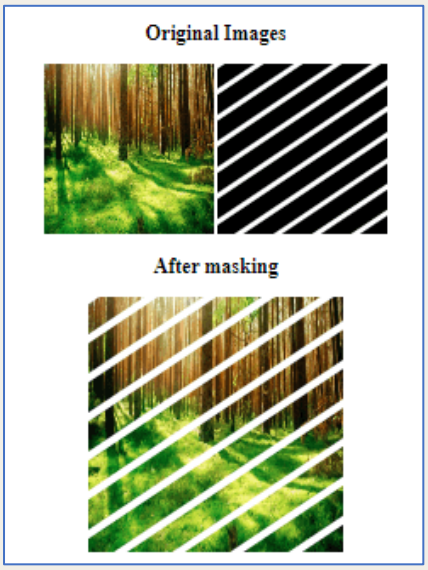


Figura 84 *– Propriedade de máscara no CSS (**Fonte:** https://www.w3schools.com/css/css3_masking.asp)*

Botões do CSS

Existem muitas propriedades para estilizar um botão no CSS. Abaixo segue um exemplo:

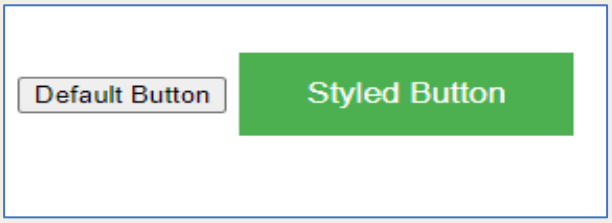


Figura 85 *– O botão "convencional" (**Fonte:** https://www.w3schools.com/css/css3_buttons.asp)*

```css
.button {
  background-color: #4CAF50;
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  font-size: 16px;
  margin: 4px 2px;
}
```

Figura 86 *– Código para conceber um botão "convencional" (**Fonte:** https://www.w3schools.com/css/css3_buttons.asp)*

Paginação no CSS

Caso tenhas um website com muitas páginas, poderás desejar adicionar algum tipo de paginação a cada página. Estes são alguns exemplos:

Figura 87 *– Paginação no CSS – ex.1 (**Fonte:** https://www.w3schools.com/css/css3_pagination.asp)*

Figura 88 *– Paginação no CSS – ex.2 (**Fonte:** https://www.w3schools.com/css/css3_pagination.asp)*

Múltiplas colunas no CSS

O *layout* multi-column do CSS permite a fácil definição de múltiplas colunas de texto, tal como se vê nos jornais:

***Figura 89** – Múltiplas Colunas no CSS (**Fonte:** https://www.w3schools.com/css/css3_multiple_columns.asp)*

Propriedades de interface do utilizador no CSS

Neste capítulo, aprenderás sobre as seguintes propriedades da interface de utilizador CSS:

- A propriedade resize especifica se (e como) um elemento deve ser redimensionável pelo utilizador.

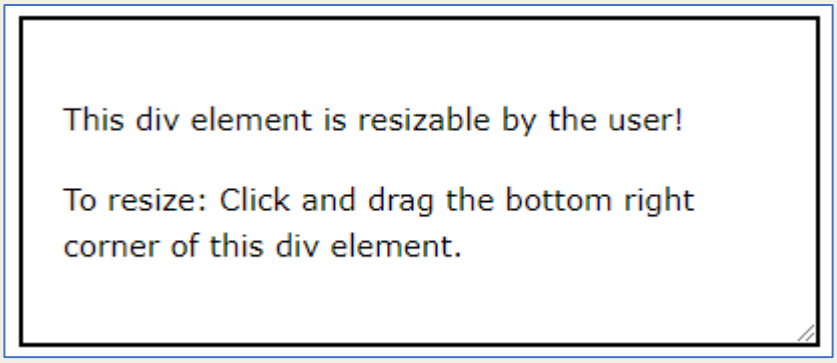


Figura 90 *– Como aplicar a propriedade resize (**Fonte:** https://www.w3schools.com/css/css3_user_interface.asp)*

- A propriedade outline-offset adiciona espaço entre o contorno e o canto ou limite de um elemento.

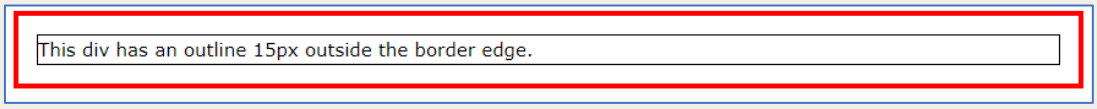


Figura 91 *– A propriedade outline-offset (**Fonte:** https://www.w3schools.com/css/css3_user_interface.asp)*

Variáveis do CSS

A função var() é utilizada para inserir o valor de uma variável CSS.

As variáveis CSS têm acesso ao DOM, o que significa que pode criar variáveis com alcance local ou global, alterar as variáveis com JavaScript, e alterar as variáveis com base em consultas de media.

Uma boa maneira de aplicar variáveis CSS dá-se ao tratar das cores de um design. Em vez de copiar e colar as mesmas cores uma e outra vez, poderemos colocá-las em variáveis.

A propriedade de box-sizing no CSS

A propriedade box-sizing em CSS define como o utilizador deve calcular a largura e altura total de um elemento, ou seja, se preenchimento e bordas devem ser incluídos ou não.

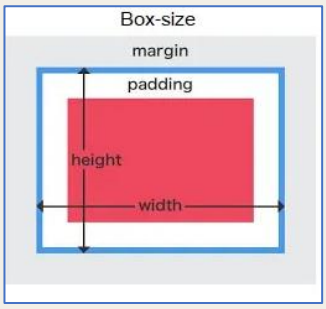


Figura 92 *– A propriedade de box-sizing (**Fonte:** https://www.w3schools.com/css/css3_box-sizing.asp)*

Consultas de aparelhos multimédia no CSS

As consultas de aparelhos multimédia no CSS3 alargaram o espetro dos tipos de multimédia da variante CSS2: agora, em vez de procurarem um tipo de dispositivo, olham para a capacidade do dispositivo.

As consultas por parte de aparelhos multimédia podem ser usadas para verificar várias coisas, tais como:

- largura e altura da janela de visualização;
- largura e altura do dispositivo;
- orientação (o tablet/telefone está no modo paisagem ou retrato?);
- resolução.

A utilização de consultas de media é uma técnica popular para fornecer uma folha de estilo adaptada aos computadores de secretária, portáteis, tablets e *smartphones* (tais como iPhone e Android).

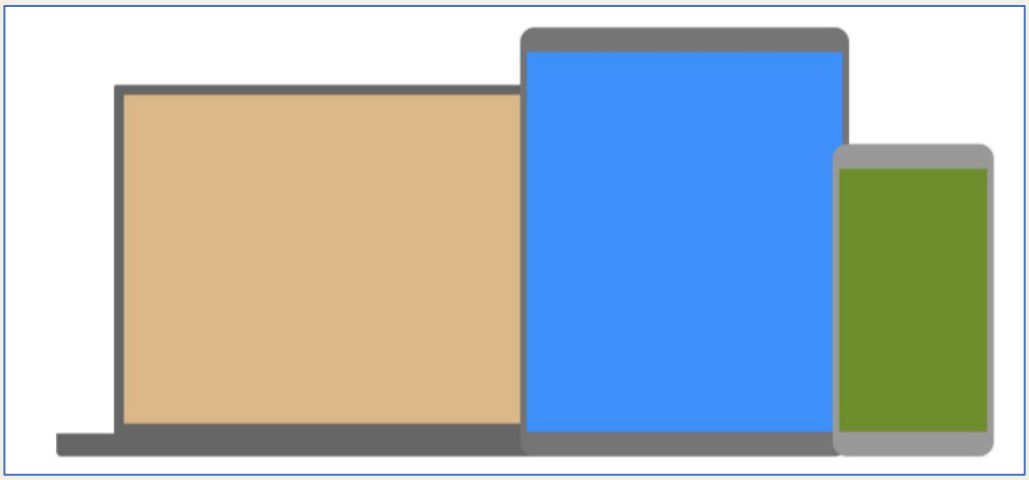


***Figura 93** – Exemplo de consultas de aparelhos multimédia no CSS (**Fonte:** https://www.w3schools.com/css/css3_mediaqueries_ex.asp)*

O módulo de layout Flexbox

O Módulo de Layout de Flexbox (“Caixa Flexível”) torna mais fácil projetar uma estrutura de layout reativa e flexível sem usar flutuação ou posicionamento.

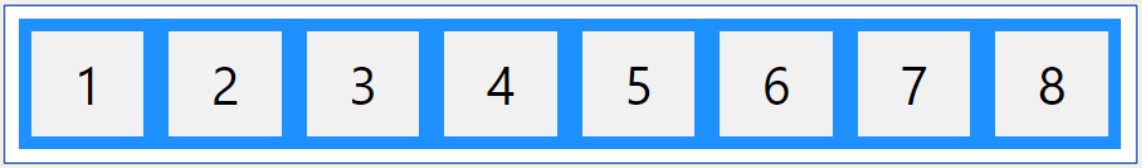


***Figura 94** – O módulo de layout Flexbox (**Fonte:** https://www.w3schools.com/css/css3_flexbox.asp)*

3.3.. Web Design Reativo do CSS

Introdução ao WDR

O design reactivo da web faz com que a tua página web tenha bom aspecto em todos os dispositivos.

Web design reativo usa apenas HTML e CSS.

As páginas da web podem ser visualizadas usando muitos dispositivos diferentes: desktops, tablets e telefones. A tua página da web deve ter uma boa aparência e ser fácil de usar, independentemente do dispositivo.

As páginas Web não devem deixar de fora a informação para caber em dispositivos mais pequenos, mas sim adaptar o seu conteúdo para caber em qualquer dispositivo:

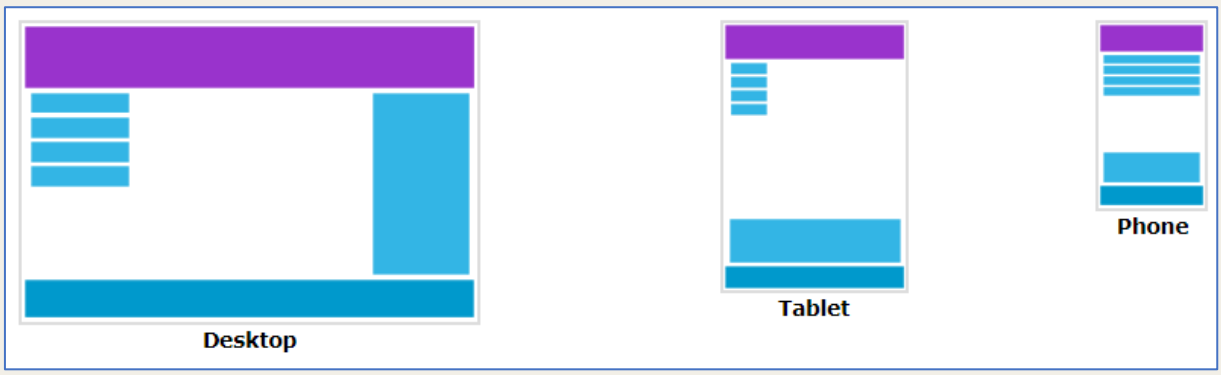


***Figura 95** – Exemplo do WDR (**Fonte:** https://www.w3schools.com/css/css_rwd_intro.asp)*

Ao ato de usar CSS e HTML para redimensionar, esconder, encolher, ampliar, ou mover o conteúdo para o fazer parecer bem em qualquer ecrã chamamos "*web design reativo*".

Janela de visualização

A janela de visualização ("*viewport*") é a área visível do utilizador de uma página web.

O *viewport* varia com o dispositivo e será mais pequeno num telemóvel do que num ecrã de computador. Antes dos *tablets* e *smartphones*, as páginas web eram concebidas apenas para ecrãs de computador, e era comum que as páginas web tivessem um design estático e um tamanho fixo.

Então, quando começamos a navegar na internet usando tablets e telefones celulares, as páginas da web de tamanho fixo eram muito grandes para caber na janela de visualização. Para corrigir isso, os navegadores desses dispositivos reduziram toda a página da Web para caber no ecrã.

Esta não foi, de todo, uma solução perfeita! Tratou-se de uma solução rápida.

Configuração do *viewport*

O HTML5 introduziu um método para deixar os *web designers* assumirem o controlo do *viewport*, através da tag <meta>.

Deverá incluir-se o seguinte elemento de viewport <meta> em todas as suas páginas web:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Figura 96 *– Definição do viewport (**Fonte:** https://www.w3schools.com/css/css_rwd_viewport.asp)*

Isto dá ao navegador instruções sobre como controlar as dimensões e escalas da página. A fração *width=device-width* define a largura da página para seguir a largura do ecrã do dispositivo (que variará dependendo do dispositivo).

A parte *initial-scale=1.0* define o nível de *zoom* inicial quando a página é carregada pela primeira vez pelo navegador.

Aqui está um exemplo de uma página web sem a meta tag *viewport*, e a mesma página web com a meta tag *viewport*:

Figura 97 *– Páginas com e sem viewport (**Fonte:** https://www.w3schools.com/css/css_rwd_viewport.asp)*

Visualização em Grelha

Muitas páginas web são baseadas numa visualização em grelha, o que significa que a página é dividida em colunas:

Figura 98 *– Visualização em grelha (**Fonte:** https://www.w3schools.com/css/css_rwd_grid.asp)*

Muitas páginas *web* são baseadas numa visualização em grelha, o que significa que a página é dividida em colunas:

Figura 99 *– Divisão da página em colunas (**Fonte:** https://www.w3schools.com/css/css_rwd_grid.asp)*

Construir uma visualização em grelha reativa

Primeiro, é necessário certificar-se de que todos os elementos HTML têm a propriedade *box-sizing* definida como *border-box*. Isso garante que o preenchimento e o limite de margem sejam incluídos na largura e altura totais dos elementos.

O seguinte código deve ser adicionado:

```
* {
  box-sizing: border-box;
}
```

Figura 100 *– Construir uma visualização em grelha reativa (**Fonte:** https://www.w3schools.com/css/css_rwd_grid.asp)*

O exemplo seguinte mostra uma página web simples e reativa, com duas colunas:

25%	75%

Figura 101 *– Visualização em grelha reativa (**Fonte:** https://www.w3schools.com/css/css_rwd_grid.asp)*

```
.menu {
  width: 25%;
  float: left;
}
.main {
  width: 75%;
  float: left;
}
```

Figura 102 *– Codificação para grelhas reativas (**Fonte:** https://www.w3schools.com/css/css_rwd_grid.asp)*

Consultas de aparelhos multimédia

Como já foi visto, as consultas de aparelhos multimédia em WDR são uma técnica introduzida no CSS3. Utiliza-se a regra *@media* para incluir um bloco de propriedades CSS apenas se uma determinada condição for verdadeira.

Se a janela do navegador for 600px ou menor, a cor de fundo será azul-claro:

```
@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

Figura 103 *– Consultas de aparelhos multimédia em WDR (**Fonte:** https://www.w3schools.com/css/css_rwd_mediaqueries.asp)*

Adicionar um ponto de interrupção

Para este propósito, as consultas de aparelhos multimédia podem ser úteis. Podemos adicionar um ponto de interrupção onde certas partes do *design* se comportarão de maneira diferente em cada lado do ponto de interrupção.

Figura 104 *– Consultas de aparelhos multimédia em WRD (**Fonte:** https://www.w3schools.com/css/css_rwd_mediaqueries.asp)*

Deve usar-se uma consulta de aparelhos multimédia para adicionar um ponto de interrupção em 768px. Exemplo: Quando o ecrã (janela do navegador) fica menor que 768px, cada coluna deve ter uma largura de 100%.

```
/* For desktop: */
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}

@media only screen and (max-width: 768px) {
  /* For mobile phones: */
  [class*="col-"] {
    width: 100%;
  }
}
```

Figura 105 *– Consulta de aparelhos multimédia em WDR (**Fonte:** https://www.w3schools.com/css/css_rwd_mediaqueries.asp)*

Imagens em WRD

Se a propriedade largura for definida para uma percentagem e a propriedade altura for definida para "auto", a imagem será reativa e escalada para cima e para baixo:

```
img {
  width: 100%;
  height: auto;
}
```

Figura 106 *– Código de imagens em WRD (**Fonte:** https://www.w3schools.com/css/css_rwd_images.asp)*

Se a propriedade max-width estiver definida como 100%, a imagem será reduzida se for necessário, mas nunca será maior do que seu tamanho original:

```
img {
  max-width: 100%;
  height: auto;
}
```

Figura 107 *– A propriedade max-width (**Fonte:** https://www.w3schools.com/css/css_rwd_images.asp)*

Note-se que no exemplo acima, a imagem pode ser ampliada para ser maior do que o seu tamanho original. Uma solução melhor, em muitos casos, será a utilização da propriedade max-width.

Emprego da propriedade max-width:

A propriedade max-width é definida como 100%, a imagem será reduzida se for necessário, mas nunca será aumentada para ser maior que seu tamanho original:

```
img {
  max-width: 100%;
  height: auto;
}
```

Figura 108 *– A propriedade max-width (**Fonte:** https://www.w3schools.com/css/css_rwd_images.asp)*

Vídeos em DRW

Se a propriedade width estiver definida como 100%, o reprodutor de vídeo responderá, aumentando ou diminuindo em conformidade:

```
video {
  width: 100%;
  height: auto;
}
```

***Figura 109** – Vídeo em DRW – a propriedade width (**Fonte:** https://www.w3schools.com/css/css_rwd_videos.asp)*

Se a propriedade max-width estiver definida como 100%, o player de vídeo será reduzido se necessário, mas nunca será maior do que o tamanho original:

```
video {
  max-width: 100%;
  height: auto;
}
```

***Figura 110** – Vídeo em DRW – a propriedade max-width (**Fonte:** https://www.w3schools.com/css/css_rwd_videos.asp)*

Observe-se que, no exemplo acima, o reprodutor de vídeo pode ser dimensionado para ser maior do que seu tamanho original. Uma solução melhor, em muitos casos, será usar a propriedade max-width.

Emprego da propriedade max-width:

Se a propriedade max-width estiver definida como 100%, o player de vídeo será reduzido se necessário, mas nunca será maior do que o tamanho original:

```
video {
  max-width: 100%;
  height: auto;
}
```

Figura 111 *– Propriedade max-width em vídeo (**Fonte:** https://www.w3schools.com/css/css_rwd_videos.asp)*

Frameworks em WRD

Existem vários frameworks no CSS gratuitos, que possibilitam um design reativo. Uma ótima maneira de criar um design responsivo é usar uma folha de estilo reativa, como a W3.CSS, que facilita o desenvolvimento de sites, que ficam bem em qualquer tamanho.

Outra estrutura popular é o Bootstrap, que utiliza HTML e CSS para fazer páginas web reativas.

3.4.. Grelhas no CSS

Introdução

O CSS Grid Layout Module oferece um sistema de layout baseado em grelha, com linhas e colunas, facilitando a conceção de páginas web sem ter de usar flutuadores e posicionamento.

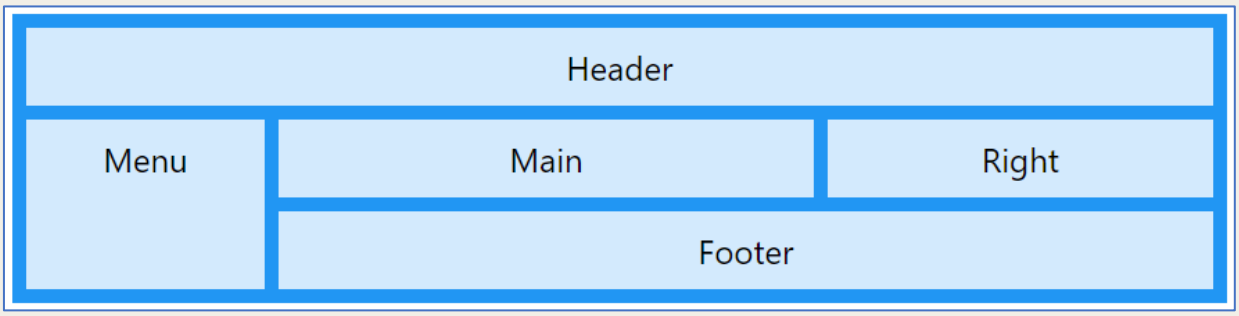


Figura 112 *– Grelhas no CSS (**Fonte:** https://www.w3schools.com/css/css_grid.asp)*

Grid Containers

To make an HTML element behave as a grid container, you have to set the display property to grid or inline-grid. Para que um elemento HTML se comporte como um *grid container*, é necessário definir a propriedade de exibição como *grid* ou *inline-grid*.

Grid containers consistem em itens de grelha colocados dentro de colunas e filas.

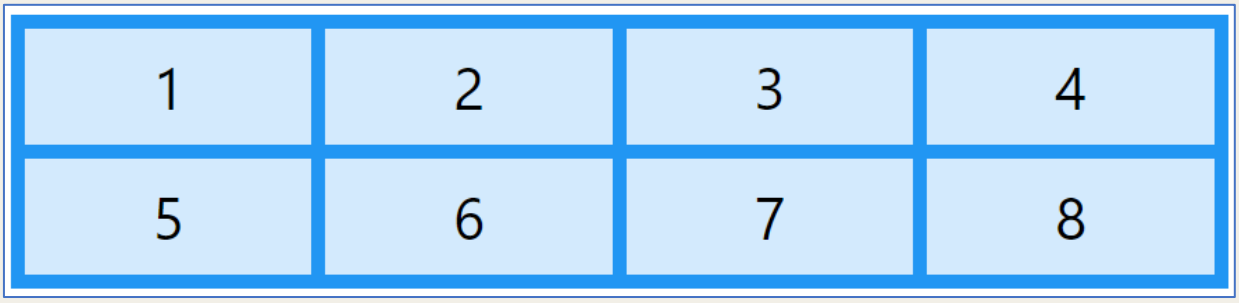


Figura 113 *– Grid Containers (**Fonte:** https://www.w3schools.com/css/css_grid_container.asp)*

Elementos da *grid*

Uma grid *container* contém elementos de *grid*. Por defeito, um *container* possui um item de grelha para cada coluna, em cada linha, mas pode estilizar os itens da grelha de modo a que abranjam várias colunas e/ou filas. A propriedade grid-column define em qual(is) coluna(s) colocar um elemento. Aqui, define-se onde o elemento irá começar, e onde irá terminar.

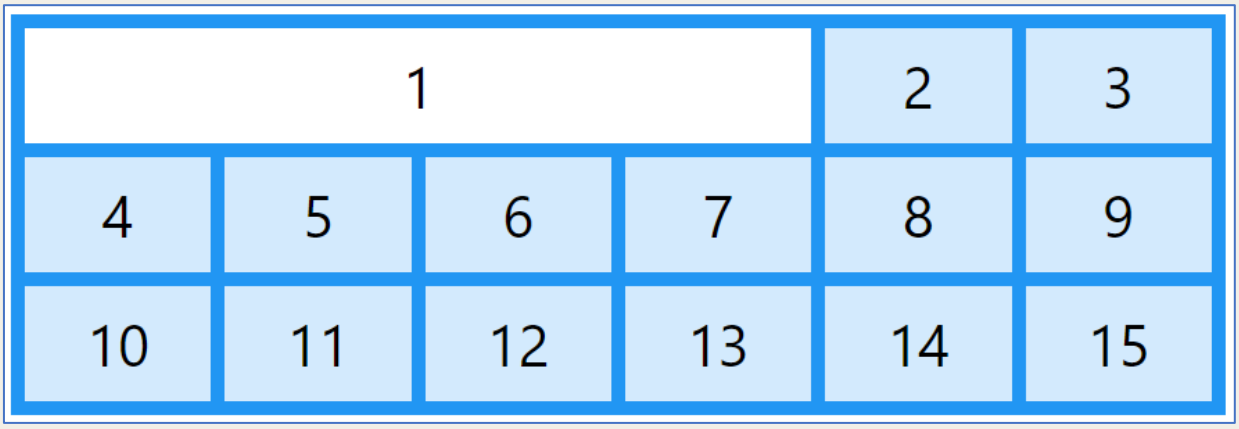


Figura 114 *– Elementos da grid (**Fonte:** https://www.w3schools.com/css/css_grid_item.asp)*

3.5. CSS SASS

Introdução

O que significa SASS?

SASS significa "Syntactically Awesome Stylesheet" (*Folha de Estilos Sintaticamente Fantásticos*), tratando-se de um pré-processador de CSS, compatível com todas as suas versões. É conhecido por reduzir repetições, poupando imenso tempo. Foi concebido por Hampton Catlin e desenvolvido por Natalie Weizenbaum, em 2006. O seu download e utilização são gratuitos.

Porquê usar SASS?

As folhas de estilo estão a ficar maiores, mais complexas, e mais difíceis de manter. É aqui que um pré-processador de CSS pode ajudar.

O SASS permite-te usar recursos que não existem no CSS, como variáveis, *nested rules*, *mixins*, importações, heranças, funções internas, entre outros.

Um exemplo simples do porquê de SASS ser útil

Digamos que temos um website com três cores principais:

Figura 115 – *Cores em SASS (**Fonte:** https://www.w3schools.com/sass/sass_intro.php)*

Então, quantas vezes é necessário digitar esses valores HEX? Exato, muitas vezes.

E as variações das mesmas cores?

Em vez de digitar muitas vezes os valores acima, poderá usar-se “SASS” e digitar o seguinte:

```
/* define variables for the primary colors */
$primary_1: #a2b9bc;
$primary_2: #b2ad7f;
$primary_3: #878f99;

/* use the variables */
.main-header {
  background-color: $primary_1;
}

.menu-left {
  background-color: $primary_2;
}

.menu-right {
  background-color: $primary_3;
}
```

***Figura 116** – Uma das variáveis “SASS” (**Fonte:** https://www.w3schools.com/sass/sass_variables.php)*

Assim, quando se usa SASS e a cor primária muda, só é necessário mudá-la num único local.

Como funciona o SASS?

Um navegador não entende o código SASS. Por conseguinte, necessitarás de um pré-processador SASS para converter o código SASS em CSS padrão.

Este processo chama-se “*transpiling*”. Portanto, é necessário dar a um *transpiler* (algum tipo de programa) algum código SASS e depois obter algum código CSS de volta.

Tipos de ficheiros do SASS

Os ficheiros SASS têm a extensão de ficheiro “. scss".

Comentários do SASS

O SASS suporta comentários CSS padrão /* comment */ e, além disso, suporta comentários inline // comment:

```
/* define primary colors */
$primary_1: #a2b9bc;
$primary_2: #b2ad7f;

/* use the variables */
.main-header {
  background-color: $primary_1; // here you can put an inline comment
}
```

Figura 117 – *Comentários do SASS (**Fonte:** https://www.w3schools.com/sass/sass_intro.php)*

Variáveis do SASS

As variáveis são uma forma de armazenar informação que se poderá reutilizar mais tarde.

Através do SASS, é possível armazenar informação em variáveis, como por exemplo:

- *strings*
- números
- cores
- booleanos

- listas
- *nulls*

O SASS utiliza o símbolo $, seguido de um nome, para declarar variáveis:

```
$variablename: value;
```

Figura 118 *– Variável do SASS (**Fonte:** https://www.w3schools.com/sass/sass_variables.php)*

O exemplo seguinte declara quatro variáveis, denominadas "myFont", "myColor", "myFontSize" e "myWidth". Depois das variáveis serem declaradas, poder-se-á utilizá-las onde for necessário:

```
$myFont: Helvetica, sans-serif;
$myColor: red;
$myFontSize: 18px;
$myWidth: 680px;

body {
  font-family: $myFont;
  font-size: $myFontSize;
  color: $myColor;
}

#container {
  width: $myWidth;
}
```

Figura 119 *– SASS variable (**Fonte:** https://www.w3schools.com/sass/sass_variables.php)*

Assim, quando o ficheiro Sass é transposto, toma as variáveis (myFont, myColor, etc.) e produz CSS normais com os valores das variáveis colocadas no CSS da seguinte forma:

```
body {
  font-family: Helvetica, sans-serif;
  font-size: 18px;
  color: red;
}

#container {
  width: 680px;
}
```

***Figura 120** – Variáveis SASS afetas ao tipo de letra (**Fonte:** https://www.w3schools.com/sass/sass_variables.php)*

O âmbito das variáveis SASS

As variáveis de baixo nível só estão disponíveis ao nível de nidificação ("nesting") onde são definidas.

Observando o seguinte exemplo:

```
$myColor: red;

h1 {
  $myColor: green;
  color: $myColor;
}

p {
  color: $myColor;
}
```

***Figura 121** – Variável SASS de cor de letra (**Fonte:** https://www.w3schools.com/sass/sass_variables.php)*

A cor do texto dentro da tag <p> será vermelha ou verde? Pois, será vermelha!

A outra definição, $myColor: green, insere-se dentro da regra <h1> e estará apenas disponível aí.

Desta forma, o resultado CSS será o seguinte:

```
h1 {
  color: green;
}

p {
  color: red;
}
```

***Figura 122** – Variável SASS relativa ao tipo de letra (**Fonte:** https://www.w3schools.com/sass/sass_variables.php)*

Este é o comportamento por defeito para o âmbito das variáveis em SASS.

Como usar o SASS !global

O comportamento padrão para o âmbito variável pode ser anulado utilizando a função *!global*. !global indica que uma variável é global, o que significa que é acessível a todos os níveis. Observe-se o seguinte exemplo (o mesmo que acima; mas com o !global acrescentado):

```
$myColor: red;

h1 {
  $myColor: green !global;
  color: $myColor;
}

p {
  color: $myColor;
}
```

Figura 123 *– !global no SASS (**Fonte:** https://www.w3schools.com/sass/sass_variables.php)*

Agora a cor do texto dentro da tag <p> será verde!

Assim, o resultado seria o seguinte:

```
h1 {
  color: green;
}

p {
  color: green;
}
```

Figura 124 *– Resultado da função !global na cor verde (**Fonte:** https://www.w3schools.com/sass/sass_variables.php)*

Regras "*nested*" do SASS

O SASS permite fazer o "*nesting*" (encadear um seletor dentro de outro) de seletores da mesma forma que o HTML. Vejamos um exemplo de um código SASS para a navegação de um site:

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }
  li {
    display: inline-block;
  }
  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

***Figura 125** – O "nesting" do SASS (**Fonte:** https://www.w3schools.com/sass/sass_nesting.php)*

Note-se que, em SASS, o "*nesting*" dos seletores ul e li foi feito dentro do seletor nav. Enquanto no CSS, as regras são definidas uma a uma (não é feito o "*nesting*"):

```
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

***Figura 126** – Disposição do "nesting" em SASS (**Fonte:** https://www.w3schools.com/sass/sass_nesting.php)*

Sendo possível fazer o "*nesting*" de propriedades em SASS, a leitura torna-se mais limpa e simples em relação ao CSS padrão.

Propriedades em "nesting" do SASS

Muitas propriedades do CSS têm o mesmo prefixo, como font-family, font-size e font-weight ou text-align, text-transform e text-overflow. Com o SASS, poderão digitar-se como propriedades em "*nesting*":

```
font: {
  family: Helvetica, sans-serif;
  size: 18px;
  weight: bold;
}

text: {
  align: center;
  transform: lowercase;
  overflow: hidden;
}
```

Figura 127 *– Propriedades em "nesting" do SASS (**Fonte:** https://www.w3schools.com/sass/sass_nesting.php)*

O "*transpiler*" SASS converterá o código acima referido em CSS normal:

```
font-family: Helvetica, sans-serif;
font-size: 18px;
font-weight: bold;

text-align: center;
text-transform: lowercase;
text-overflow: hidden;
```

Figura 128 *– O "transpiler" do SASS em ação (**Fonte:** https://www.w3schools.com/sass/sass_nesting.php)*

A diretiva @import e os "*partials*"

O SASS incorpora o código CSS de forma DRY ("Don't Repeat Yourself" - Não Se Repita a Si Mesmo). Uma forma de digitar o código em DRY passa por manter o código respetivo em ficheiros separados.
É possível criar pequenos ficheiros com trechos de CSS para incluir noutros ficheiros SASS. Exemplos de tais ficheiros podem ser: ficheiro de reset, variáveis, cores, tipos de letra, tamanhos de fontes, etc.

Importar Ficheiros no SASS

Assim como o CSS, o SASS também suporta a diretiva @import. A diretiva @import permite incluir o conteúdo de um arquivo noutro. A diretiva CSS @import tem um grande inconveniente devido a questões de desempenho, dado que cria um pedido HTTP extra cada vez que é invocada. No entanto, a diretiva SASS @import inclui o ficheiro no CSS e, assim, não é necessário invocar o HTTP de novo enquanto o ficheiro executa.

```
@import filename;
```

Figura 129 *– Importar ficheiros no SASS (**Fonte:** https://www.w3schools.com/sass/sass_import.php)*

Dica: Não é necessário especificar uma extensão de ficheiro, dado que o SASS assume automaticamente que é feita uma referência a um ficheiro .sass ou .scss. Também é possível importar ficheiros CSS. A diretiva @import importa o ficheiro e quaisquer variáveis ou mixins definidas no ficheiro importado podem então ser utilizadas no ficheiro principal.

Poderás importar todos os ficheiros que precisares no ficheiro principal:

```
@import "variables";
@import "colors";
@import "reset";
```

Figura 130 *– Importação de vários elementos usando a diretiva @import (**Fonte:** https://www.w3schools.com/sass/sass_import.php)*

Vejamos um exemplo: Vamos supor que temos um arquivo de *reset* chamado "reset.scss", semelhante ao exemplo seguinte:

```
html,
body,
ul,
ol {
  margin: 0;
  padding: 0;
}
```

Figura 131 *– Características de um elemento prestes a ser importado no SASS (**Fonte:** https://www.w3schools.com/sass/sass_import.php)*

E agora queremos importar o ficheiro "reset.scss" para outro ficheiro denominado "standard.scss". **Solução**: acrescentar a diretiva @import no topo de um ficheiro. Desta forma, o conteúdo do mesmo terá um âmbito global:

```
@import "reset";

body {
  font-family: Helvetica, sans-serif;
  font-size: 18px;
  color: red;
}
```

Figura 132 *– Resultado do ato de importação (**Fonte:** https://www.w3schools.com/sass/sass_import.php)*

Assim, quando o ficheiro "standard.css" for transposto, o código CSS assemelhar-se-á ao seguinte:

```
html, body, ul, ol {
  margin: 0;
  padding: 0;
}

body {
  font-family: Helvetica, sans-serif;
  font-size: 18px;
  color: red;
}
```

Figura 133 *– O ficheiro "standard.css" após a sua transposição (**Fonte:** https://www.w3schools.com/sass/sass_import.php)*

Os "partials" no SASS

Por defeito, o SASS transporta todos os ficheiros .scss diretamente. No entanto, quando for pretendido importar um ficheiro, não é necessário que o mesmo seja transposto diretamente. O SASS tem um mecanismo pronto para lidar com a situação. Se o nome de um ficheiro iniciar com um "*underscore*" (_), o SASS não o transporá. Os ficheiros nomeados desta forma são chamados de "*partials*".

Desta forma, um ficheiro "*partial*" em SASS é invocado após colocação prévia de um "*underscore*":

```
_filename;
```

Figura 134 *– Exemplo de invocação de um ficheiro "partial" (**Fonte:** https://www.w3schools.com/sass/sass_import.php)*

O exemplo seguinte mostra um ficheiro "partial" de SASS denominado "_colors.scss". (Este ficheiro não será transposto diretamente para "color.css"):

```
$myPink: #EE82EE;
$myBlue: #4169E1;
$myGreen: #8FBC8F;
```

Figura 135 *– O ficheiro SASS "colors.scss" (**Fonte:** https://www.w3schools.com/sass/sass_import.php)*

Desta feita, se for importado o ficheiro "*partial*", deverá omitar-se o "*underscore*". O SASS entenderá que deve importar o arquivo "_colors.scss":

```
@import "colors";

body {
  font-family: Helvetica, sans-serif;
  font-size: 18px;
  color: $myBlue;
}
```

Figura 136 *– Resultado após importação do ficheiro "partial" (**Fonte:** https://www.w3schools.com/sass/sass_import.php)*

O "@mixin" e o "@include" em SASS

Os "*mixins*" em SASS

A diretiva @mixin permite criar código CSS que deve ser reutilizado em todo o website.

Definição de um "*mixin*"

Um "*mixin*" é definido com a diretiva @mixin.

```
@mixin name {
  property: value;
  property: value;
  ...
}
```

Figura 137 *– O @mixin em SASS (**Fonte:** https://www.w3schools.com/sass/sass_mixin_include.php)*

O exemplo a seguir cria um "*mixin*" chamado "*important-text*":

```
@mixin important-text {
  color: red;
  font-size: 25px;
  font-weight: bold;
  border: 1px solid blue;
}
```

Figura 138 *– O @mixin em SASS e seus elementos (**Fonte:** https://www.w3schools.com/sass/sass_mixin_include.php)*

Usar um *mixin*

A diretiva *@include* é usada para incluir um arquivo *mixin*.

```
selector {
  @include mixin-name;
}
```

Figura 139 *– A diretiva @include (**Fonte:** https://www.w3schools.com/sass/sass_mixin_include.php)*

Assim, de forma a incluir o *mixin* important-text criado acima, dever-se-á:

```
.danger {
  @include important-text;
  background-color: green;
}
```

Figura 140 *– A importância do important-text (**Fonte:** https://www.w3schools.com/sass/sass_mixin_include.php)*

O "*transpiler*" SASS converterá o código acima referido em CSS normal:

```
.danger {
  color: red;
  font-size: 25px;
  font-weight: bold;
  border: 1px solid blue;
  background-color: green;
}
```

Figura 141 *– Conversão do SASS em CSS normal (**Fonte:** https://www.w3schools.com/sass/sass_mixin_include.php)*

Um *mixin* também pode incluir outros *mixins*:

```
@mixin special-text {
  @include important-text;
  @include link;
  @include special-border;
}
```

Figura 142 *– Presença de um mixin noutro mixin (**Fonte:** https://www.w3schools.com/sass/sass_mixin_include.php)*

Como passar variáveis para um mixin

Os *mixins* aceitam argumentos. Desta forma, é possível passar variáveis para um *mixin*:

```
/* Define mixin with two arguments */
@mixin bordered($color, $width) {
  border: $width solid $color;
}

.myArticle {
  @include bordered(blue, 1px);  // Call mixin with two values
}

.myNotes {
  @include bordered(red, 2px); // Call mixin with two values
}
```

Figura 143 *– Passar variáveis para um mixin através de argumentos (**Fonte:** https://www.w3schools.com/sass/sass_mixin_include.php)*

Nota que os argumentos são definidos como variáveis e, de seguida, usados como os valores (cor e largura) da propriedade de limite de margem.

Após a compilação, o código CSS terá o seguinte aspeto:

```
.myArticle {
  border: 1px solid blue;
}

.myNotes {
  border: 2px solid red;
}
```

Figura 144 *– Código de CSS após compilação (**Fonte:** https://www.w3schools.com/sass/sass_mixin_include.php)*

A diretiva @extend no SASS

A diretiva @extend permite compartilhar um conjunto de propriedades CSS de um seletor para outro. A diretiva @extend é especialmente útil caso existam elementos de estilo quase idênticos, que diferem apenas em alguns pequenos detalhes.
O seguinte exemplo de SASS cria primeiro um estilo básico para botões (este estilo será utilizado para a maioria dos botões). De seguida, criaremos um estilo para um botão "Report" e um estilo para um botão "Submit". Tanto o botão "Report" como o botão "Submit" herdam todas as propriedades CSS da classe .button-basic, através da diretiva @extend. Além disso, têm as suas próprias cores definidas, como pode ser visto no exemplo seguinte:

```
.button-basic  {
  border: none;
  padding: 15px 30px;
  text-align: center;
  font-size: 16px;
  cursor: pointer;
}

.button-report  {
  @extend .button-basic;
  background-color: red;
}

.button-submit  {
  @extend .button-basic;
  background-color: green;
  color: white;
}
```

Figura 145 *– A diretiva extend no SASS (**Fonte:** https://www.w3schools.com/sass/sass_extend.php)*

Após a compilação, o código CSS terá o seguinte aspeto:

```css
.button-basic, .button-report, .button-submit {
  border: none;
  padding: 15px 30px;
  text-align: center;
  font-size: 16px;
  cursor: pointer;
}

.button-report  {
  background-color: red;
}

.button-submit  {
  background-color: green;
  color: white;
}
```

***Figura 146** – A diretiva extend após compilação no código CSS (**Fonte:** https://www.w3schools.com/sass/sass_extend.php)*

Ao utilizar a diretiva @extend, não é necessário especificar várias classes para um elemento no código HTML, como por exemplo "Report This". É apenas necessário

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

4. SQL – Linguagem de Consulta Estruturada

Informação acerca do Tópico

Tópico:

4. SQL

Pré-requisitos:

Literacia informática básica, *software* básico instalado e conhecimento básico sobre como trabalhar com ficheiros.

Duração:

10 horas

Descrição:

Este tema cobre as noções básicas de SQL, de forma a familiarizar os aprendentes com o mundo da programação e encorajá-los a ganhar competências de SQL. São explicados os atributos, síntaxe e outros termos relevantes que os aprendentes já poderão ter ouvido ou com que poderão estar familiarizados, e ainda de que forma estes se enquadram na linguagem de programação. É ainda facultada uma detalhada descrição da lógica e síntaxe usadas em SQL, a sua estrutura e outras funções básicas e essenciais.

Objetivos de aprendizagem:

- Reconhecer o conceito e uso da SQL em Sistemas de Gestão de Bases de Dados Relacionais (SGBDR);
- Formular síntaxe básica para consulta SQL;
- Usar a SQL para aceder, gerir e manipular SGBD.

Material necessário:

- Computador ou portátil;
- Ligação à internet;

- Compilador *online* SQL (https://www.mycompiler.io/new/sql);
- Editor de texto *online* (https://www.w3schools.com/sql/default.asp);

Cenário de aula:

No total, serão dedicadas 10 horas a este tema, sendo que fica ao critério de cada formador(a)/professor(a) decidir o tempo a alocar a cada subtema. Sugerimos o uso das apresentações PowerPoint sobre este tema criadas para a formação para facilitar o processo de ensino e aumentar a eficiência temporal. Estas apresentações abrangem o seguinte:

- Desenvolvimento progressivo dos subtemas e conceitos centrais a reter;
- Exercícios recomendados.

Consoante as preferências do(a) formador(a)/professor(a), o desenvolvimento progressivo das apresentações permite a conclusão da sessão SQL no tempo estipulado, 10 horas. As apresentações também podem ser disponibilizadas aos formandos para estudo individual.

Subtemas:

- 4.1. Noções básicas de SQL
- 4.2. Bases de Dados SQL
- 4.3. Referências SQL
- 4.4. Exemplos de SQL

Recursos adicionais:

- W3Schools – Guia sobre todas as palavras-chave e funções de SQL e exemplos de cada um deles;
- Tutorialspoint – Outro guia detalhado sobre palavras-chave e funções SQL e vários exemplos de cada um deles.

4.1. Noções básicas de SQL

O que é a SQL?

SQL é o acrónimo de *Structured Query Language*, ou a tradução literal para português, linguagem de consulta estruturada. A SQL é uma linguagem de programação padrão especificamente criada para armazenar, recuperar, gerir ou manipular dados de um Sistema de Gestão de Bases de Dados Relacionais (SGBDR). Uma Base de Dados Relacional é um conjunto de dados com uma relação pré-definida entre eles. Estes itens estão organizados num conjunto de tabelas com colunas e linhas.

A SQL tornou-se um padrão ISO em 1987.

Esta linguagem de programação é a linguagem de bases de dados mais implementada e é suportada por sistemas relacionais de bases de dados populares, tais como MySQL, SQL Server, and Oracle. A SQL foi desenvolvida pela IBM no início dos anos 1970 e, originalmente, o seu nome era SEQUEL (Structured English Query Language), que mais tarde mudou para SQL.

Usos da SQL

A SQL é uma das linguagens de consulta mais usadas em bases de dados. Alguns dos seus diversos usos são:

- Permitir aos utilizadores aceder aos dados nos sistemas de gestão de bases de dados relacionais;
- Permitir aos utilizadores descrever os dados;
- Permitir aos utilizadores definir e manipular os dados de uma base de dados.

A síntaxe da SQL

As instruções de SQL são diretas, como inglês simples, mas têm uma síntaxe específica.

Uma instrução de SQL é composta por uma sequência de palavras-chave, identificadores, etc., e finalizada com uma semi-vírgula (;).

Exemplo:
SELECT emp_name, hire_date, salary FROM employees WHERE salary > 5000;

Para melhor legibilidade, a mesma instrução pode ser escrita da seguinte forma:

```
SELECT emp_name, hire_date, salary
FROM employees
WHERE salary > 5000;
```

No final de cada instrução de SQL deve ser usada uma semi-vírgula (;) — este sinal de pontuação indica o final da instrução ou submete a informação ao servidor da base de dados.

O uso de maiúsculas e minúsculas em SQL

Considere a seguinte instrução de SQL que faculta registos de uma tabela “Employees”:

SELECT emp_name, hire_date, salary FROM employees;

A mesma instrução pode também ser escrita da seguinte forma:

select emp_name, hire_date, salary from employees;

O uso de letras maiúsculas ou minúsculas é indiferente em SQL, o que significa que ‘SELECT’ é o mesmo que ‘select’.

Contudo, as bases de dados e nomes de tabelas podem ser sensíveis ao uso de maiúsculas e minúsculas dependendo do sistema operativo. De maneira geral, as plataformas Unix e Linux são sensíveis a maiúsculas e minúsculas, e as plataformas Windows não.

O comando ‘Select’ em SQL

O comando ‘SELECT’ seleciona ou obtém os dados de uma ou mais tabelas. Este comando pode ser usado para obter a informação de todas as linhas de uma tabela de uma vez só, ou para selecionar apenas as linhas que correspondem a uma condição específica ou a uma combinação de condições.

Suponhamos que temos na nossa base de dados uma tabela com o nome “Employees” (“Funcionários”) e que contém os seguintes registos:

```
| emp_id| emp_name| hire_date| salary| dept_id|

+--------+-------------+-----------+--------+---------+

|      1 | Ethan Hunt   | 2001-05-01 |   5000 |       4 |

|      2 | Tony Montana | 2002-07-15 |   6500 |       1 |

|      3 | Sarah Connor | 2005-10-18 |   8000 |       5 |

|      4 | Rick Deckard | 2007-01-03 |   7200 |       3 |

|      5 | Martin Blank | 2008-06-24 |   5600 |    NULL |

+--------+-------------+-----------+--------+---------+
```

Selecionar todos os dados da tabela

Para obter os dados de todas as linhas da tabela, deverá ser usada a seguinte instrução:

>> SELECT * FROM employees;

Selecionar colunas específicas da tabela

Se não necessitarmos de todos os dados da tabela, podemos selcionar colunas específicas através da seguinte instrução:

```
SELECT emp_id, emp_name, hire_date, salary
FROM employees;
```

Após executar as instruções supramencionadas, obteremos o seguinte *output*:

| emp_id| emp_name| hire_date| salary|

```
+--------+-------------+-----------+--------+

|      1 | Ethan Hunt   | 1995-10-30 |   5000 |

|      2 | Tony Montana | 1990-07-15 |   6500 |

|      3 | Sarah Connor | 2011-04-13 |   5600 |

|      4 | Rick Deckard | 2005-10-18 |   7200 |

|      5 | Martin Blank | 1996-05-24 |   8000 |

+--------+-------------+-----------+--------+
```

O comando 'SELECT DISTINCT' em SQL

O comando 'SELECT DISTINCT' omite valores duplicados numa consulta. É possível encontrar valores duplicados numa tabela, mas, por vezes, queremos consultar apenas valores únicos.

Síntaxe:

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Nos exemplos seguintes, iremos usar a tabela “Customers”, que contém dados sobre os nossos clientes.

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

*Tabela 1 – Exemplo de uso do comando ‘SELECT DISTINCT’ com a tabela “Customers” (**Fonte**: https://www.w3schools.com/sql/sql_distinct.asp)*

Para selecionar todos os valores da coluna “Country” da tabela “Customers”, deve ser usada a seguinte instrução:

```
SELECT Country FROM Customers;
```

Contudo, esta instrução incluiría valores duplicados.

De forma a omitir os valores duplicados, deverá ser usado o commando ‘SELECT DISTINCT’:

```
SELECT DISTINCT Country FROM Customers;
```

Suponhamos agora que queremos listar os diferentes países clientes, usaríamos a seguinte instrução:

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

Note-se que este exemplo não funciona no Firefox, pois o comando ‘COUNT(DISTINCT column_name)’ não é suportado em MS Access.

Para obter um resultado equivalente em Access, poderá ser usada a seguinte instrução:

```
>> SELECT Count(*) AS DistinctCountries
    FROM (SELECT DISTINCT Country FROM Customers);
```

O comando ‘WHERE’ em SQL

Aprendemos, anteriormente, como obter todos os registos de uma tabela ou colunas de uma tabela.

Contudo, em casos reais, normalmente é necessário selecionar, atualizar ou apagar apenas os registos que correspondem a determinadas condições, como por exemplo utilizadores que pertencem a uma determinada faixa etária, país, etc.

O comando ‘WHERE’ é usado com ‘select, ‘update’ e ‘DELETE’.
Este comando é usado com ‘SELECT’ para extrair apenas os registos que correspondem a determinadas condições.

A síntaxe base é a seguinte:

```
SELECT column_list
FROM table_name
WHERE condition;
```

Analisemos agora alguns exemplos demonstrativos.

Vamos supor que temos na nossa base de dados a tabela “Employees” com os seguintes registos:

```
| emp_id| emp_name| hire_date| salary| dept_id|
+--------+-------------+-----------+--------+---------+
|      1 | Ethan Hunt   | 2001-05-01 |   5000 |       4 |
|      2 | Tony Montana | 2002-07-15 |   6500 |       1 |
|      3 | Sarah Connor | 2005-10-18 |   8000 |       5 |
|      4 | Rick Deckard | 2007-01-03 |   7200 |       3 |
+--------+-------------+-----------+--------+---------+
```

A seguinte instrução SQL apresentará como resultados todos os funcionários com um salário superior a 7000 que estão presentes na tabela:

```
SELECT * FROM employees WHERE salary > 7000;
```

O comando 'WHERE' apenas filtra e exclui os dados indesejados.

Outro exemplo seria selecionar todos os funcionários com o valor 'department id=1':

```
SELECT * FROM employees WHERE dept_id=1;
```

A tabela em baixo faculta a lista de operadores que podem ser usados com o comando 'WHERE':

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Tabela 2 -*Tabela "Operators" com o comando 'WHERE' (**Fonte**: https://www.w3schools.com/sql/sql_where.asp)*

Os operadores 'AND', 'OR' e 'NOT' em SQL

O comando 'WHERE' pode ser usado com os operadores 'AND', 'OR', e 'NOT'.

Os operadores 'AND' e 'OR' são usados para filtrar registos tendo por base mais do que uma condição.

O operador 'AND' exibe um registo caso as condições envolvidas sejam verdadeiras.

A síntaxe 'AND':

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Exemplo: Selecionar todos os campos da tabela "Customers" em que o valores em "Country" e "City" sejam "Germany" e "Berlin", respetivamente.

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

O operador 'OR' exibe um registo caso as condições envolvidas sejam verdadeiras.

A síntaxe 'OR':

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Exemplo 1: Selecionar todos os campos da tabela "Customers" em que o valor em "City" seja "Berlin" ou "München".

```
SELECT * FROM Customers.
WHERE City='Berlin' OR City='München';
```

Exemplo 2: Selecionar todos os campos da tabela "Customers" em que o valor em "Country" seja "Germany" ou "Spain".

```
SELECT * FROM Customers
WHERE Country='Germany' OR Country='Spain';
```

O operador 'NOT' exibe um registo caso as condições envolvidas não sejam verdade.

A síntaxe 'NOT':

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Exemplo: Selecionar todos os campos da tabela "Customers" em que o valor no campo "Country" não seja "Germany".

```
SELECT * FROM Customers
WHERE NOT Country='Germany';
```

Conjugar os operadores 'AND', 'OR' e 'NOT'

Exemplo 1: Selecionar todas as linhas da tabela "Customers" nas quais o campo "Country" seja "Germany" e o campo 'City' seja "Berlin" ou "München".

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

Exemplo 2: Selecionar todas as linhas da tabela "Customers" nas quais o campo "Country" não seja "Germany" nem "USA".

```
SELECT * FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

O comando ‘ORDER BY’ em SQL

O comando ‘ORDER BY’ apresenta os resultados requeridos por ordem crescente ou decrescente. De forma a organizar os resultados por ordem decrescente, use ‘DESC’.

A síntaxe ‘ORDER BY’:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Nos exemplos seguintes será usada a tabela “Customers” apresentada em baixo:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Tabela 3 – *Tabela “Customers” com um exemplo de uso do comando ‘ORDER BY’* ***(Fonte:*** https://www.w3schools.com/sql/sql_orderby.asp*)*

Exemplo 1: Selecionar todos os clientes da tabela “Customers” e organizá-los por coluna “Country”.

```
SELECT * FROM Customers
ORDER BY Country;
```

Exemplo 2: Selecionar todos os clientes da tabela e organizá-los por coluna “Country” em ordem decrescente.

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

Exemplo 3: Selecionar todos os clientes da mesma tabela e organizá-los por “Country” e “Customer Name”.

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

Neste exemplo, os dados são, inicialmente, organizados por “Country”. Contudo, se existirem linhas com o mesmo valor no campo “Country”, estas serão organizadas por “Customer Name”.

Exemplo 4: Selecionar todos os clientes da mesma tabela e organizar os dados por “Country” em ordem crescente, e por “Customer Name” em ordem decrescente.

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

O comando ‘INSERT INTO’ em SQL

O comando ‘INSERT INTO’ insere novos registos numa tabela.

De forma a que o Código seja processado corretamente, os nomes das colunas e os valores a ser inseridos deverão ser especificados.

Síntaxe:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

Tabela 4 – *Exemplo do comando 'INSERT INTO'* ***(Fonte:*** https://www.w3schools.com/sql/sql_insert.asp*)*

Por exemplo, para acrescentar um novo registo à tabela "Customers", use a seguinte instrução:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

Valores 'NULL' em SQL

Um campo com um valor 'NULL' é um campo sem um valor. Se um campo de uma tabela for opcional, é possível inserir um novo registo ou atualizar um registo sem acrescentar um novo valor a este campo.

Nota: Um valor 'NULL' é diferente de um valor zero ou de um campo que contém espaços. Um campo com um valor 'NULL' é um campo que foi **deixado em branco** durante a criação do registo!

Teste de valores 'NULL'

É impossível testar para obter valores 'NULL' com operadores de comparação, como '=', '<' ou '<>'.

Em vez destes operadores, teremos de usar os operadores **'IS NULL'** e **'IS NOT NULL'**.

A Síntaxe 'IS NULL':

```
SELECT column_names
```

```
FROM table_name
WHERE column_name IS NULL;
```

A Síntaxe 'IS NOT NULL':

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

Os exemplos seguintes usam a tabela em baixo:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

***Tabela 5 –** Tabela "Customers" em uso num exemplo com valores 'NULL'. **(Fonte:** https://www.w3schools.com/sql/sql_null_values.asp)*

Exemplo de 'IS NULL'

Seleciona todos os clientes com valores 'NULL' (ex., valores vazios) na coluna 'Address':

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

Para procurar valores 'NULL', devemos usar sempre 'IS NULL'.

Exemplo de 'IS NULL'

Seleciona todos os clientes com valores 'NOT NULL' (ex., valores que não estejam vazios) na coluna 'Address':

```
SELECT CustomerName, ContactName, Address
```

```
FROM Customers
WHERE Address IS NOT NULL;
```

O comando 'UPDATE' em SQL

O comando 'UPDATE' altera os registos existentes de uma tabela.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

É necessário algum cuidado ao atualizar os registos de uma tabela. Note-se o uso do comando 'WHERE' na instrução 'UPDATE'. O comando 'WHERE' especifíca que registos devem ser atualizados.

Se o comando 'WHERE' for **omitido**, todos os registos na tabela serão atualizados!

Exemplo

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Tabela 6 – *Tabela "Customers" em uso num exemplo do comando 'UPDATE'* ***(Fonte:*** https://www.w3schools.com/sql/sql_update.asp*)*

Para atualizar 'CustomerID=1' da tabela "Customers", utilize a seguinte instrução:

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
```

```
WHERE CustomerID = 1;
```

Após a inserção da instrução anterior, a tabela “Customers” terá este aspeto:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Tabela 7 – *Tabela “Customers” com um exemplo do uso de ‘UPDATE’* ***(Source:*** https://www.w3schools.com/sql/sql_update.asp*)*

O comando ‘WHERE’ determina quantos registos serão atualizados.

A seguinte instrução SQL atualizará o campo “ContactName” para “Juan” em todos os registos nos quais o campo “Country” tem o valor “Mexico”:

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

O comando ‘DELETE’ em SQL

O comando ‘DELETE’ apaga os registos da tabela existentes.

```
DELETE FROM table_name WHERE condition;
```

Note-se que é necessária alguma caução aquando apagar registos da tabela! Repare no uso do comando 'WHERE' na instrução 'DELETE'. O comando 'WHERE' determina que registo(s) deve(m) ser eliminado(s).

Se o comando 'WHERE' for **omitido**, todos os registos da tabela serão eliminados!

Exemplo:

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Apagar todos os registos

É possível eliminar todas as linhas de uma tabela sem eliminar a tabela. Quer isto dizer que a estrutura, os atributos e índices da tabela ficarão intactos:

```
DELETE FROM table_name;
```

O comando 'SELECT TOP' em SQL

O comando 'SELECT TOP' é usado para determinar o número de registos que serão acedidos.

Este comando é útil para tabelas grandes com milhares de registos. No entanto, exibir um grande número de registos pode afetar o desempenho.

Note-se que nem todos os sistemas de bases de dados suportam o comando 'SELECT TOP'. O MYSQL suporta o comando 'LIMIT' para selecionar um número limitado de registos, ao passo que o Oracle usa 'FETCH FIRST n ROWS ONLY' e 'ROWNUM'.

SQL Server /Síntaxe MS Access:

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

A Síntaxe MySQL:

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

A Síntaxe Oracle 12:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s)
FETCH FIRST number ROWS ONLY;
```

A Síntaxe Older Oracle:

```
SELECT column_name(s)
FROM table_name
WHERE ROWNUM <= number;
```

A Síntaxe Older Oracle Syntax com ORDER BY:

```
SELECT *
FROM (SELECT column_name(s) FROM table_name ORDER BY column_name(s))
WHERE ROWNUM <= number;
```

Para os exemplos seguintes será usada a tabela “Customers” em baixo:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Tabela 8 – *Tabela "Customers" com exemplos de 'SELECT TOP'* ***(Fonte:*** *https://www.w3schools.com/sql/sql_top.asp)*

Exemplos de 'TOP', 'LIMIT' e 'FETCH FIRST'

Use a seguinte instrução para selecionar os primeiros três registos da tabela 'Customers' em SQLServer/MS Access:

```
SELECT TOP 3 * FROM Customers;
```

Para executar uma consulta com o mesmo resultado que em cima mas em MySQL, use a seguinte instrução:

```
SELECT * FROM Customers
LIMIT 3;
```

Para executar uma consulta com o mesmo resultado em Oracle:

```
SELECT * FROM Customers
FETCH FIRST 3 ROWS ONLY;
```

Exemplos de 'TOP PERCENT'

Para selecionar 50% dos registos da tabela "Customers", execute a seguinte instrução em SQL Server/MS Access:

```
SELECT TOP 50 PERCENT * FROM Customers;
```

A seguinte instrução equivale à de cima em Oracle:

```
SELECT * FROM Customers
```

```
FETCH FIRST 50 PERCENT ROWS ONLY;
```

Exemplos do comando 'ADD a WHERE'

No exemplo seguinte, iremos selecionar os primeiros três registos da tabela "Customers", em que o campo "Country" seja o valor "Germany" para SQL Server/MS Access:

```
>> SELECT TOP 3 * FROM Customers
    WHERE Country='Germany';
```

A instrução equivalente em MySQL:

```
SELECT * FROM Customers
WHERE Country='Germany'
LIMIT 3;
```

A instrução equivalente em Oracle:

```
SELECT * FROM Customers
WHERE Country='Germany'
FETCH FIRST 3 ROWS ONLY;
```

A função 'MIN' e 'MAX' em SQL

A função 'MIN()' apresenta os valores mais pequenos das colunas selecionadas.

A Síntaxe 'MIN()'

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

A função 'MAX()' apreesenta os valores mais altos das colunas selecionadas.

A Síntaxe 'MAX()'

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

Nos exemplos seguintes, será usada a tabela "Products" que segue em baixo:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

***Tabela 9 -** Tabela "Products" com exemplos de 'MIN' e 'MAX' **(Fonte:** https://www.w3schools.com/sql/sql_min_max.asp)*

Exemplo 1: Para encontrar o preço do produto mais barato:

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

Exemplo 2: Para encontrar o preço do produto mais caro:

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

As funções 'COUNT', 'AVG' E 'SUM' EM SQL

A função 'COUNT' apresenta o número de linhas que correspondem a critérios específicos.

A Síntaxe 'COUNT()'

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

A função 'AVG()' apresenta o valor médio de uma coluna numérica.

A Síntaxe 'AVG()'

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

A função 'SUM()' apresenta a soma total de uma coluna numérica.

A Síntaxe 'SUM()'

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

Para os exemplos seguintes será usada a tabela "Products" em baixo:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

Tabela 10 – *Tabela "Products" com exemplos de 'Count', 'Avg', 'Sum'* ***(Fonte:*** *https://www.w3schools.com/sql/sql_count_avg_sum.asp)*

Exemplo de 'COUNT()'

Para executar uma consulta para enocntrar o número de produtos:

```
SELECT COUNT(ProductID)
FROM Products;
```

Exemplo de 'AVG()'

Para executar uma pesquisa para encontrar o preço médio de todos os produtos:

```
SELECT AVG(Price)
FROM Products;
```

OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

***Tabela 11** – Tabela "OrdersDetails" com exemplos 'Count', 'Avg', 'Sum' **(Fonte:** https://www.w3schools.com/sql/sql_count_avg_sum.asp)*

No exemplo seguinte, será usada a tabela "OrdersDetails", apresentada em cima, para procurer o total de 'Quantity':

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

O operador 'LIKE' em SQL

O operador 'LIKE' é usado numa instrução 'WHERE' para pesquisar um padrão específico numa coluna.

A Síntaxe 'LIKE'

```
SELECT column1, column2, ...
FROM table_name
WHERE column LIKE pattern;
```

Em baixo estão alguns exemplos de diferentes operadores 'LIKE' com carateres '%' e '_':

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

***Tabela 12** – Operadores 'LIKE' (**Fonte**: https://www.w3schools.com/sql/sql_like.asp)*

Veremos agor alguns exemplos com a tabela "Customers" que segue em baixo:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

***Tabela 13** – Tabela "Customers" com exemplos 'LIKE' (**Fonte:** https://www.w3schools.com/sql/sql_like.asp)*

Exemplo 1: Para selecionar todos os clientes com nome a começar por 'a':

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

Exemplo 2: Para selecionar todos os clientes cujo nome termina em 'a':

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

Exemplo 3: Para selecionar todos os clientes cujo nome contém ‘or’ em qualquer posição:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

Exemplo 4: Para selecionar todos os nomes de clientes que têm ‘r’ como segunda letra:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

Exemplo 5: Para selecionar todos os nomes de clientes que comecem com ‘a’ e tenham pelo menos três carateres:

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

Exemplo 6: Para selecionar os nomes de clientes que comecem com ‘a’ e terminem em ‘o’:

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%o';
```

Exemplo 7: Para selecionar os nomes de clientes que não comecem por ‘a’:

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

Wildcards em SQL

Um *wildcard* substitui um ou mais carateres numa cadeia. É usado com o operador ‘LIKE’.

O operador ‘LIKE’ também é usado com o comando ‘WHERE’ para pesquisar um padrão específico numa coluna, tal como vimos na subsecção anterior.

Wildcards em MS Access

Symbol	Description	Example
*	Represents zero or more characters	bl* finds bl, black, blue, and blob
?	Represents a single character	h?t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
!	Represents any character not in the brackets	h[!oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt
#	Represents any single numeric character	2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295

***Tablela 14** – 'Wildcards' em Access (**Fonte:** https://www.w3schools.com/sql/sql_wildcards.asp)*

Wildcards no SQL Server

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt

***Tabela 15** – 'Wildcards' em SQL Server (**Fonte**: https://www.w3schools.com/sql/sql_wildcards.asp)*

Os *wildcards* podem ser combinados, considere o exemplo em baixo:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a__%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

***Tabela 16** - Exemplos de 'Wildcards' com '%' and '_' (**Fonte**: https://www.w3schools.com/sql/sql_wildcards.asp)*

Consideremos agora alguns exemplos com a tabela "Customers":

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen	T2F 8M4	Canada
11	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 5NT	UK

Tabela 17 – *Tabela "Customers" com exemplos de 'WILDCARDS' (**Fonte**: https://www.w3schools.com/sql/sql_wildcards.asp)*

Exemplos com o *wildcard* '%'

Neste exemplo, selecionamos todos os clientes com uma "City" a começar por "ber":

```
SELECT * FROM Customers
WHERE City LIKE 'ber%';
```

No exemplo seguinte, selecionamos todos os clientes com uma "City" que contenha "es":

```
SELECT * FROM Customers
WHERE City LIKE '%es%';
```

Exemplos com o *wildcard* '_'

Neste exemplo, são selecionados todos os clientes com uma "City" que se inicie por qualquer letra seguida de "ondon":

```
SELECT * FROM Customers
WHERE City LIKE '_ondon';
```

Para selecionar todos os clientes com uma "City" a iniciar-se com a letra 'L', seguida por qualquer carater seguido por "on":

```
SELECT * FROM Customers
WHERE City LIKE 'L_n_on';
```

Exemplos com o *wildcard* '[charlist]'

Para selecionar todos os clientes com uma "City" a iniciar com 'b', 's' ou 'p':

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

No exemplo seguinte, selecionaremos todos os clientes com uma "City" a começar com 'a', 'b', ou 'c':

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

Exemplos com o *wildcard* '[!charlist]'

O ponto de exclamação mostra carateres que não contêm uma cadeia específica. Por exemplo, queremos selecionar todos os clientes com uma "City" que não começa por 'b', 's' ou 'p':

```
SELECT * FROM Customers
WHERE City LIKE '[!bsp]%';
```

Em alternativa, podemos usar a seguinte instrução:

```
SELECT * FROM Customers
WHERE City NOT LIKE '[bsp]%';
```

O operador 'IN' em SQL

O operador 'IN' é usado para especificar valores múltiplos numa instrução 'WHERE'. Pode considerar-se que satisfaz várias condições.

Síntaxe 1:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Síntaxe 2:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

Há duas formas de usar o operador 'IN', como vimos anteriormente.

Suponhamos que temos uma tabela "Customers" que contém as seguintes colunas: "CustomerID", "CustomerName", "ContactName", "Address", "City", "PostalCode" and "Country".

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain

***Tabela 18** – Tabela "Customers" com exemplos de operador 'IN' (**Fonte:** https://www.w3schools.com/sql/sql_in.asp)*

Como exemplo, queremos selecionar todos os clientes localizados na Alemanha ('Germany'), França ('France') ou Reino Unido ('UK'):

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK')
```

Outro exemplo, é selecionar todos os clientes que **não** estão localizados na Alemanha ('Germany'), França ('France') ou Reino Unido ('UK'):

```
SELECT * FROM Customers
WHERE Country NOT IN ( 'Germany',  'France', 'UK')
```

Consideremos ainda um terceiro exemplo no qual queremos selecionar os clientes que são do mesmo país que os fornecedores (*suppliers*):

```
SELECT * FROM Customers
WHERE Country  IN (SELECT Country FROM Suppliers)
```

O operador 'BETWEEN' em SQL

O operador 'BETWEEN' faculta um conjunto de valores dos quais escolher. Os valores podem ser texto, números ou datas. O operador 'BETWEEN' inclui os valores iniciais e finais.

Síntaxe:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Consideremos a seguinte tabela, que contém informação sobre diferentes produtos:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	1	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	1	2	36 boxes	21.35

***Tabela 19** – Exemplo de operador 'BETWEEN' (**Fonte**: https://www.w3schools.com/sql/sql_between.asp)*

Seguem vários exemplos com os seguintes operadores: 'BETWEEN', 'NOT BETWEEN', 'BETWEEN' com 'IN', 'BETWEEN' e 'NOT BETWEEN' com valores em texto e datas 'BETWEEN'.

Exemplo de 'BETWEEN': Para selecionar todos os produtos com um intervalo de preço entre 10 e 20:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

Exemplo de 'NOT BETWEEN': Exibe todos os produtos fora do intervalo definido no exemplo anterior:

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

Exemplo de 'BETWEEN' com 'IN': Seleciona todos os produtos com um intervalo de preços entre 10 e 20 e não mostra os produtos com a "CategoryID" 1, 2 ou 3:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

Exemplos de 'BETWEEN' com valores em texto: Seleciona todos os produtos com um "ProductName" entre "Carnarvorn Tigers" e "Mozzarella di Giovanni":

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

Exemplos de 'NOT BETWEEN' com valores em texto: Seleciona todos os produtos com um "ProductName" que não esteja entre "Carnarvorn Tigers" e "Mozzarella di Giovanni":

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

Consideremos agora a seguinte tabela com informação relativa a diferentes “Orders” (encomendas):

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	7/4/1996	3
10249	81	6	7/5/1996	1
10250	34	4	7/8/1996	2
10251	84	3	7/9/1996	1
10252	76	4	7/10/1996	2

__Tabela 20__ – Exemplo de operador ‘BETWEEN’ (__Fonte__: https://www.w3schools.com/sql/sql_between.asp)

Exemplo de datas ‘BETWEEN’: Selecionar todas as encomendas com data entre ’01-July-1996’ e ’31-July-1996’.
Isto pode ser feito de duas formas, usando um cardinal (#) ou aspas (“”):

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;
```

OU

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

Aliases em SQL

Os aliases atribuem um nome temporário a uma tabela ou coluna de uma tabela. Um alias existe apenas durante uma pesquisa e é, normalmente, usado para tornar os nomes das colunas mais legíveis. Um aliasé criado através do uso da palavra-chave **AS.**

Síntaxa para colunas alias:

```
SELECT column_name AS alias_name
FROM table_name;
```

Síntaxe para tabelas alias:

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

Colunas aliases

Analisemos um exemplo que cria dois aliases, um para cada coluna:

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Outro exemplo para criar dois aliases, novamente:

```
SELECT CustomerName AS Customer, ContactName AS [Contact Person]
FROM Customers;
```

Note-se que este é colocado entre parêntesis retangulares ([ ]) porque o alias contém espaços. As aspas podem ser usadas em alternativa aos parêntesis.

Podemos também criar um alias com uma ou mais colunas, consideremos o exemplo em baixo:

```
SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' + Country AS Address
FROM Customers;
```

A instrução em cima citada difere em MySQL:

```
SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Address
FROM Customers;
```

Tabelas Aliases

O exemplo seguinte seleciona todas as encomendas da tabela de clientes com "CustomerID=4" ("Around the Horn").

Neste exemplo, são usados aliases para encurtar a pesquisa:

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
```

Uma pesquisa em aliases assemelhar-se-á a isto:

```
SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName
FROM Customers, Orders
WHERE Customers.CustomerName='Around the Horn' AND Customers.CustomerID=Orders.CustomerID;
```

A operação 'JOIN' em SQL

A operação 'JOIN' combina linhas de duas ou mais tabelas tendo por base uma coluna comum em ambas as tabelas.

Consideremos as tabelas "Orders" e "Customers":

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

***Tabelas 21 & 22** – Tabelas "Orders" e "Customers" num exemplo 'JOIN' (**Fonte:** https://www.w3schools.com/sql/sql_join.asp)*

Ao analisar as duas tabelas é possóvel verificar que estas têm uma coluna em comum, "CustomerID". Com base nesta coluna comum, poderemos criar uma instrução SQL com 'INNER JOIN', que seleciona registos com valores correspondentes em ambas as tabelas.

Exemplo:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Este comando criará algo semelhante ao que consta na tabela em baixo:

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

***Tabela 23** – Exemplo 'JOIN' (**Fonte**: https://www.w3schools.com/sql/sql_join.asp)*

Existem quatro junções ('joins') diferentes em SQL:

1. **(INNER) JOIN:** exibe registos com valores equivalentes e mambas as tabelas;
2. **LEFT (OUTER) JOIN:** apresenta todos os registos da tabela da esquerda e os registos correspondentes da tabela da direita;
3. **RIGHT (OUTER) JOIN:** apresenta todos os registos da tabela da direita e os valores correspondentes da tabela da esquerda;

4. **FULL (OUTER) JOIN:** exibe todos os registos quando há uma correspondência na tabela da esquerda ou da direira.

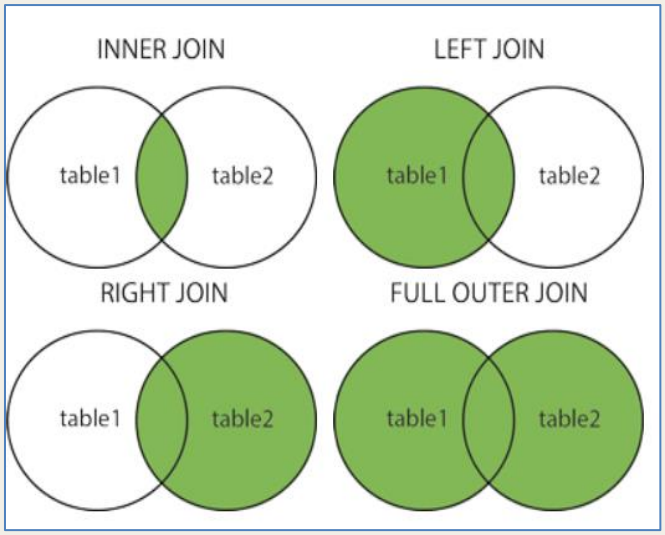


***Figura 1** – Diferentes tipos de 'JOINS' (**Fonte**: https://www.w3schools.com/sql/sql_join.asp)*

'INNER JOIN' em SQL

A operação 'INNER JOIN' seleciona registos que têm valores equivalentes em ambas as tabelas.

Síntaxe:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

Para este exemplo foram usadas as mesmas tabelas que na subsecção anterior.

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

***Tabelas 24 & 25** – Tabelas "Orders" e "Customers" em exemplo 'JOIN' (**Fonte**: https://www.w3schools.com/sql/sql_join_inner.asp)*

Neste exemplo, queremos obter os nomes dos clientes e as suas "Order IDs" respetivas:

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Note-se que a instrução 'INNER JOIN' selecionará todas as linhas correspondentes de ambas as tabelas. Se os registos da tabela "Orders" não tiverem registos equivalentes na tabela "Customers", não serão selecionados.

No exemplo seguinte, veremos como juntar três tabelas com dados referentes a clientes e informação de envio:

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

'LEFT JOIN' em SQL

A operação 'LEFT JOIN' exibe todos os registos da tabela da esquerda e os registso equivalentes da tabela da direita. Se não forem encontrados dados correspondentes, não serão apresentados quaisquer registos da tabela da direita como resultado.

Síntaxe:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Atenção, a operação 'LEFT JOIN' denomina-se por 'LEFT OUTER JOIN' em algumas bases de dados.

Como exemplo, vamos selecionar todos os clientes e quaisquer encomendas que estes possam ter feito:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note-se que todos os registos da tabela da esquerda,"Customers", serão exibidos, mesmo que não existam registos correspondentes na tabela da direita, "Orders".

'RIGHT JOIN' em SQL

A operação 'RIGHT JOIN' segue a mesma lógica descrita anteriormente, mas do lado direito em vez do lado esquerdo.

Esta operação exibe todos os registos da tabela da direita e os registos correspondentes da tabela da esquerda, se estes existirem.

Síntaxe:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Consideremos as seguintes tabelas, ‘Orders’ e ‘Employees’:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

Tabela 26 *– Tabela “Orders” com um exemplo ‘RIGHT JOIN’ (**Fonte**:* *https://www.w3schools.com/sql/sql_join_right.asp)*

EmployeeID	LastName	FirstName	BirthDate	Photo
1	Davolio	Nancy	12/8/1968	EmpID1.pic
2	Fuller	Andrew	2/19/1952	EmpID2.pic
3	Leverling	Janet	8/30/1963	EmpID3.pic

Tabela 27 *– Tabela “Employees” com um exemplo ‘RIGHT JOIN’ (**Fonte:*** *https://www.w3schools.com/sql/sql_join_right.asp)*

O exemplo seguinte exibirá todos os funcionários e quaisquer encomendas que possam ter feito:

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Note-se que a operação ‘RIGHT JOIN’ exibirá todos os registos da tabela da direita, “Employees”, mesmo que não exista correpondência com a tabela da esquerda, “Orders”. Aplica-se a mesma lógica que vimos anteriormente na operação ‘LEFT JOIN’.

‘FULL JOIN’ em SQL

A operação ‘FULL JOIN’ recupera todos os registos quando são encontrados registos correspondentes quern a tabela da direita, quern a tabela da esquerda.

Note-se que 'FULL OUTER JOIN' e 'FULL JOIN' são a mesma coisa e 'FULL JOIN' pode exibir um número maior de resultados.

Síntaxe:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Consideremos as seguintes tabelas, a tabela "Orders" e a tabela "Customers":

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

***Tabela 28** – Tabela "Orders" com um exemplo 'FULL JOIN' (**Fonte:** https://www.w3schools.com/sql/sql_join_full.asp)*

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

***Tabela 29** – Tabela "Customers" com um exemplo 'FULL JOIN' (**Fonte**: https://www.w3schools.com/sql/sql_join_full.asp)*

Um exemplo que seleciona todos os clientes (*customer*s) e encomendas (*orders*):

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

O resultado deste 'FULL JOIN' assemelhar-se-á a algo como isto:

CustomerName	OrderID
Null	10309
Null	10310
Alfreds Futterkiste	*Null*
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	*Null*

***Tabela 30** – Exemplo de 'FULL JOIN' com tabelas "Customers" e "Orders" (**Fonte**: https://www.w3schools.com/sql/sql_join_full.asp)*

Aqui podemos verificar que esta operação exibe todos os resultados correspondentes de ambas as tabelas, mesmo que estes não sejam encontrados. Neste caso, é atribuído o valor *nul*l.

'SELFJOIN' em SQL

Um 'SELF JOIN' é considerado uma operação 'JOIN' normal, mas a tabela é incluída também.

Síntaxe:

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

'T1' e 'T2' são aliases usados para a mesma tabela.

Consideremos a tabela "Customers" como exemplo:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

***Tabela 31** – Exemplo de 'SELF JOIN' em tabela 'Customers' (**Fonte**: https://www.w3schools.com/sql/sql_join_self.asp)*

Neste exemplo, queremos selecionar clientes da mesma cidade:

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

'UNION' em SQL

O operador 'UNION' é usado para combiner o conjunto de resultados de duas ou mais instruções 'SELECT'.

Há alguns requisitos para viabilizar um 'UNION':

1. Ao utilizar o operador 'UNION', todas as instruções 'SELECT' devem ter o mesmo número de colunas;
2. As colunas devem ter tipos de dados semelhantes;
3. As colunas devem estar na mesma ordem em todas as instruções 'SELECT'.

Síntaxe:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

* Note-se que o operador 'UNION' seleciona apenas valores diferentes por defeito.

Para permitir valores duplicados, use 'UNION ALL':

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

* Note-se que os nomes das colunas são normalmente iguais nas duas instruções 'SELECT'.

Vejamos agor alguns exemplos de 'UNION', 'UNION ALL' e 'UNION' com instruções 'WHERE' para podermos compreender melhor como usá-los.

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

*Tabela 32 – Exemplo 'UNION' com tabela "Customers" (**Fonte:** https://www.w3schools.com/sql/sql_union.asp)*

SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

*Tabela 33 – Sexemplo 'UNION' com tabela "Suppliers" (**Fonte:** https://www.w3schools.com/sql/sql_union.asp)*

O primeiro exemplo é usado para obter cidades diferentes de ambas as tabelas exibidas em cima:

```
SELECT City FROM Customers
UNION
```

```
SELECT City FROM Suppliers
ORDER BY City;
```

Por estarmos a usar 'UNION', os fornecedores (*suppliers*) da mesma cidade serão listados apenas uma vez. Se quisermos ver os valores duplicados, devemos usar 'UNION ALL'.

O exemplo seguinte apresentará como resultado os valores duplicados de ambas as tabelas:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

Neste exemplo, através do comando 'WHERE' serão exibidas as diferentes cidades alemãs presents nas tabelas "Customers" e "Suppliers":

```
SELECT City, Country FROM Customers
WHERE Country = 'Germany'
UNION
SELECT City, Country FROM Customers
WHERE Country = 'Germany'
ORDER BY City;
```

Este exemplo é semelhante ao anterior, mas, possivelmente, serão exibidos valores duplicados:

```
SELECT City, Country FROM Customers
WHERE Country = 'Germany'
UNION ALL
SELECT City, Country FROM Customers
WHERE Country = 'Germany'
ORDER BY City;
```

O próximo exemplo lista todos os clientes e fornecedores:

```
SELECT 'Customer' AS Type, ContactName, City, Country
FROM Customers
UNION
SELECT 'Supplier', ContactName, City, Country
FROM Suppliers;
```

Repare que neste caso usamos ‘AS’ para criar um alias para a pesquisa que irá desaparecer assim que esta fique concluída.

‘GROUP BY’ em SQL

A instrução ‘GROUP BY’ agrupa linhas com os mesmos valores em linhas resumidas. Suponhamos, por exemplo, que queremos consultar o número de clientes de cada país. Esta instrução é também frequentemente usada com funções combinadas, tais como ‘COUNT()’, ‘MAX()’, ‘MIN()’, ‘SUM()’, ‘AVG()’, para agrupar os resultados em uma ou mais colunas.

Síntaxe:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Para os exemplos seguintes, iremos usar a tabela “Customers” em baixo:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

***Tabela 34** – Exemplo de 'GROUP BY' com tabela "Customers" (**Fonte:** https://www.w3schools.com/sql/sql_groupby.asp)*

Neste primeiro exemplo, iremos listar o número de clientes de cada país:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

No segundo exemplo, iremos listar novamente o número de clientes em cada país, mas por ordem decrescente:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
ORDER BY COUNT(CustomerID) DESC;
```

No exemplo seguinte, iremos usar 'GROUP BY' com 'JOIN' com as tabelas "Orders" e "Shippers":

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

***Tabela 35** – Exemplo 'GROUP BY' com tabela "Orders" (**Fonte**: https://www.w3schools.com/sql/sql_groupby.asp)*

ShipperID	ShipperName
1	Speedy Express
2	United Package
3	Federal Shipping

*Tabela 36 – Exemplo de 'GROUP BY' com tabela "Shippers" (**Fonte:** https://www.w3schools.com/sql/sql_groupby.asp)*

Neste exemplo, iremos listar o número de encomendas (*orders*) enviados por cada exportador (*shipper*):

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

Note-se que começamos por selecionar os nomes dos exportadores (*shippers*) na tabela "Shippers" e contar as encomendas tendo por base os "OrderID" guardados como um alias.

Em seguida, executamos um 'LEFT JOIN' para juntar a tabela "Shippers" (table 2) com a tabela "Orders" (tabela 2) e agrupamo-las por nome de exportador.

'HAVING' em SQL

A instrução 'HAVING' foi adicionada à SQL porque o comando 'WHERE' não pode ser usado com funções combinadas.

Síntaxe:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

***Tabela 37** – Exemplo de 'HAVING' com tabela "Customers" (**Fonte:** https://www.w3schools.com/sql/sql_having.asp)*

Para este exemplo, usaremos novamente a tabela "Customers". Neste caso, queremos listar o número de clientes de cada país, mas também queremos incluir países que têm mais do que cinco clientes.

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

Neste exemplo, queremos listar o número de clientes por país, novamente, e incluir os países com mais de cinco clientes por ordem decrescente.

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

Tentemos mais alguns exemplos para combiner o que aprendemos até agora.

Nos dois exemplos seguintes iremos usar as tabelas “Orders” e “Employees”:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

*Tabela 38 – Exemplo de ‘HVING’ com tabela “Orders” (**Fonte:** https://www.w3schools.com/sql/sql_having.asp)*

EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	1968-12-08	EmpID1.pic	Education includes a BA....
2	Fuller	Andrew	1952-02-19	EmpID2.pic	Andrew received his BTS....
3	Leverling	Janet	1963-08-30	EmpID3.pic	Janet has a BS degree....

*Tabela 39 – Exemplo de ‘HAVING’ com tabela “Orders” (**Fonte:** https://www.w3schools.com/sql/sql_having.asp)*

O exemplo seguinte lista os funcionários que registaram mais de dez encomendas:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

Neste exemplo, iremos listar os funcionários de “Davolio” ou “Fuller” que tenham registado encomendas mais do que 25 vezes:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
WHERE LastName = ‘Davolio’ OR LastName = ‘Fuller’
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

'SELECT INTO' em SQL

A instrução 'SELECT INTO' copia dados de uma tabela para uma nova tabela.

Síntaxe para copiar todas as colunas para uma nova tabela:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

Síntaxe para copiar apenas algumas colunas para uma nova tabela:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

A tabela nova irá manter os nomes e tipos das colunas. É possível criar novas colunas com o comando 'AS'.

Exemplo de como criar uma cópia de segurança da tabela "Customers":

```
SELECT * INTO CustomersBackup2017
FROM Customers;
```

Exemplo do uso do comando 'IN' para copiar a tabela para uma tabela nova em outra base de dados:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

Exemplo de como copiar apenas algumas colunas para uma tabela nova:

```
SELECT CustomerName, ContactName INTO CustomersBackup2017
FROM Customers;
```

Exe4mplo de como copiar apenas os clientes alemães para uma tabela nova:

```
SELECT * INTO CustomersGermany
FROM Customers
WHERE Country = 'Germany';
```

Exemplo de como copiar dados de várias tabelas para uma tabela nova:

```
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomersOrderBackup2017
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

'SELECT INTO' também pode ser usado para criar uma tabela nova e vazia usando o mesmo esquema de outra tabela.

Para tal, temos de acrescentar um comando 'WHERE' que não obtém quaisquer dados:

```
SELECT * INTO newtable
FROM oldtable
WHERE 1 = 0;
```

'INSERT INTO SELCT' em SQL

A instrição 'INSERT INTO SELECT' copia dados de uma tabela e insere-os em outra tabela. Isto requere que o tipo de dados da fonte correspondam ao tipo de dados da tabela para onde estes serão copiados.

Síntaxe para copiar todas as colunas de uma tabela para outra:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Síntaxe para copiar apenas algumas colunas de uma tabela para outra:

```
INSERT INTO table2 (column1, column2, column3, ...)
```

```
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

***Tabela 40** – Exemplo 'INSERT INTO SELECT' com tabela "Customers" (**Fonte:** https://www.w3schools.com/sql/sql_insert_into_select.asp)*

SupplierID	SupplierName	ContactName	Address	City	Postal Code	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	Londona	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

***Tabela 41** – Exemplo de 'INSERT INTO SELECT' com tabela "Suppliers" (**Fonte:** https://www.w3schools.com/sql/sql_insert_into_select.asp)*

Para os exemplos seguintes, usaremos as tabelas "Customers" e "Suppliers".

O primeiro exemplo copia a tabela "Suppliers" para a tabela "Customers" (note-se que as colunas sem dados serão registadas com valores 'null'):

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;
```

Este exemplo copia a tabela "Suppliers" para a tabela "Customers" para preencher todas as colunas:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
SELECT SupplierName, ContactName, Address, City, PostalCode, Country FROM Suppliers;
```

O terceiro exemplo copia apenas os fornecedores alemães para a tabela “Customers”:

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers
WHERE Country='Germany';
```

‘CASE’ em SQL

A instrução ‘CASE’ repassa uma série de condições e apresenta um valor quando a primeira condição é atendida. Pense nesta instrução como uma instrução ‘IF’ e depois ‘ELSE. Quando uma condição é verificada como verdadeira, esta isntrução pára de procurar. Se nenhuma condição for dada como verdadeira, esta instrução apresentará como resultado o valor na instrução ‘ELSE’.

Note-se que se não existir uma instrução ‘ELSE’ e nenhuma condição for dada como verdadeira, o resultado será o valor ‘NULL’.

Syntax:

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

__Tabela 42__ – Exemplo ‘CASE’ com tabela “Orders” (__Fonte:__ https://www.w3schools.com/sql/sql_case.asp)

Nos exemplos seguintes, iremos usar a tabela “Orders”.

O primeiro exemplo analisa uma série de condições e apresenta um valor quando a primeira condição for atendida:

```
SELECT OrderID, Quantity,
CASE
        WHEN Quantity > 30 THEN 'The quantity is greater than 30'
        WHEN Quantity = 30 THEN 'The quantity is 30’
        ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

No segundo exemplo, iremos ordenar os clientes por cidade (*city*). Note-se que se o campo “City” for ‘NULL’ será ordenado por país (*country*).

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
    (CASE
        WHEN City IS NULL THEN Country
        ELSE City
END);
```

Funções ‘NULL’ em SQL

As funções ‘NULL’ incluem: ‘IFNULL()’, ‘ISNULL()’, ‘COALESCE()’ e ‘NVL()’.

Para os exemplos seguintes, usaremos a tabela “Products” em baixo:

P_Id	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder
1	Jarlsberg	10.45	16	15
2	Mascarpone	32.56	23	
3	Gorgonzola	15.67	9	20

Tabela 43 – *Exemplo de funções 'NULL' com tabela "Products" (**Fonte:** https://www.w3schools.com/sql/sql_isnull.asp)*

Digamos que a coluna "UnitsOnOrder" é opcional e pode conter valores 'NULL'.

Exemplo:

```
SELECT ProductName, UnitPrice * (UnitsInStock + UnitsOnOrder)
FROM Products;
```

Aqui podemos ver que se algum dos valores de "UnitsOnOrder" for 'null', o resultado também será 'null'.

Vejamos agora como Podemos ultrapassar este problema.

Em MySQL, Podemos usar a função 'ISNULL()' que permite obter um valor alternative se uma expressão for 'null':

```
SELECT ProductName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
FROM Products;
```

Ou podemos usar a função 'COALESCE()':

```
SELECT ProductName, UnitPrice * (UnitsInStock + COALESCE(UnitsOnOrder, 0))
FROM Products;
```

Em SQL Server, a função 'ISNULL()' faz o mesmo que em MySQL:

```
SELECT ProductName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
FROM Products;
```

Em MS Access, a função 'IsNull()' apresenta 'TRUE(-1)' se a expressão for um valor 'null', em caso contrário, apresenta 'FALSE (0)':

```
SELECT ProductName, UnitPrice * (UnitsInStock + IIF(IsNull(UnitsOnOrder), 0, UnitsOnOrder))
FROM Products;
```

Em Oracle, a função 'NVL()' faz a mesma coisa:

```
SELECT ProductName, UnitPrice * (UnitsInStock + NVL(UnitsOnOrder, 0))
FROM Products;
```

Comentários em SQL

Os comentários em SQL explicam as seções das isntruções SQL ou previnem a sua execução.

Note-se que os exemplos nesta secção não são suportados por Firefox ou Microsoft Edge, que são bases de dados Microsoft Access. Normalmente, os comentários não são suportados por bases de dados Microsoft Access.

Os comentários de uma linha começam com - - (dois travessões):

```
--Select all:
SELECT * FROM Customers;
```

Ou podem ser usados da seguinte forma para ignorar o fim da linha:

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

Ou para ignorar uma instrução:

```
--SELECT * FROM Customers;
SELECT * FROM Products;
```

Comentários com linhas múltiplas começam por /* e terminam em */. Qualquer texto escrito entre estes dois símbolos será ignorado.

Exemplo:

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

Para ignorar parte de uma instrução também podemos usar /**/.

Exemplo 1:

```
SELECT CustomerName, /*City,*/ Country FROM Customers;
```

Exemplo 2:

```
SELECT * FROM Customers WHERE (CustomerName LIKE 'L%'
OR CustomerName LIKE 'R%' /*OR CustomerName LIKE 'S%'
OR CustomerName LIKE 'T%'*/ OR CustomerName LIKE 'W%')
AND Country='USA'
ORDER BY CustomerName;
```

Operadores SQL

Operadores Aritméticos usados em SQL:

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

Tabela 44 *– Operadores Aritméticos (**Fonte:** https://www.w3schools.com/sql/sql_operators.asp)*

Operadores *Bitwise* usados em SQL:

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

Tabela 45– *Operadores Bitwise (**Fonte**: https://www.w3schools.com/sql/sql_operators.asp)*

Operadores Comparativos usados em SQL:

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Tabela 46 *– Operadores Comparativos (**Fonte**: https://www.w3schools.com/sql/sql_operators.asp)*

Operadores Compostos usados em SQL:

Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

Tabela 47 *- Operadores Compostos (**Fonte:** https://www.w3schools.com/sql/sql_operators.asp)*

Operadores Lógicos usados em SQL:

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

Tabela 48 *- Operadores Lógicos (**Fonte**: https://www.w3schools.com/sql/sql_operators.asp)*

4.2. Bases de Dados SQL

Tal como mencionado na secção anterior dedicada às instruções básicas usadas em SQL, esta linguagem de programação é maioritariamente usada em bases de dados relacionais. Nesta secção, iremos aprender como criar, alterar e manipular uma base de dados com SQL.

Iremos começar com instruções simples e avançar progressivamnete para instruções mais complicadas.

'CREATE DATABASE' em SQL

A instrução 'CREATE DATABASE' cria uma nova base de dados SQL.

Síntaxe:
CREATE DATABASE DatabaseName;

Nota: Lembre-se, o nome da base de dados deve ser único no Sistema de Gestão de Base de dados Relacional que está a usar, e garanta que é o administrador do mesmo antes de criar qualquer base de dados.

Imaginemos que queremos criar a base de dados 'testDB', iremos usar a seguinte instrução:

CREATE DATABASE testDB;

'DROP DB' em SQL

A instrução 'DROP DATABASE' elimina as bases de dados SQL existentes.

DROP DATABASE DatabaseName;

Antes de apagar a base de dados, garante que não precisa da informação que a mesma contém, porque esta será eliminada na sua totalidade.

Recorda-se da base de dados que criamos, ‘testDB’? Agora vamos apagá-la.

Exemplo:

```
DROP DATABASE testDB;
```

‘BACKUP DB’ em SQL

A instrução ‘BACKUP DATABASE’ faxz uma cópia de segurança complete de uma base de dados SQL existente.

Para usar esta instrução, é necessário facultar duas coisas: o nome da base de dados e a localizaçã do ficheiro.

```
BACKUP DATABASE DatabaseName
TO DISK = ‘filepath’
```

Exemplo:

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak';
```

Nota: Para evitar problemas técnicos, é melhor guardar a cópia de segurança numa unidade de disco diferente daquel onde se encontra a base de dados.

Existe outra opção na qual é feita uma cópia de segurança diferenciada com base nas mudanças que foram feitas desde a última cópida de segurança feita. Este tipo de cópia de segurança reduaz o tempo desta operação.

Para tal, use a seguinte síntaxe:

```
BACKUP DATABASE DatabaseName
TO DISK = 'filepath'
WITH DIFFERENTIAL;
```

Exemplo:

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak'
```

```
WITH DIFFERENTIAL;
```

‘CREATE TABLE’ em SQL

A instrução ‘CREATE TABLE’ cria uma nova tabela numa base de dados.

Síntaxe:

```
CREATE TABLE table_name (
        column1 datatype,
        column2 datatype,
        column3 datatype,
        .....
);
```

Nesta instrução, é necessário especificar os **nomes das colunas** e os **tipos de dados** que a coluna irá conter.

Há muitos tipos de dados, como *integer*, *date* ou *varchar*. Dependendo do tipo de dados que quer armazenar, deve escolher a opção mais adequada. Por exemplo, se tem uma coluna com o nome “Date of Birth”, então escolheria *date* como tipo de dados.

Exemplo:

```
CREATE TABLE Persons (
        PersonID int,
        LastName varchar(255),
        FirstName varchar(255),
        Address varchar(255),
        City varchar(255)
);
```

Este exemplo criará uma tabela com o nome ‘Persons’ e com 5 colunas.

A coluna “PersonID” irá conter um *integer* (int); as colunas “LastName”, “FirstName”, “Address” e “City” irão conter carateres com extenão máxima de 255.

A tabela deverá assemelhar-se a este exemplo:

PersonID	LastName	FirstName	Address	City

Tabela 49 *– Exemplo de ‘CREATE TABLE’ com tabela vazia (**Fonte:** https://www.w3schools.com/sql/sql_create_table.asp)*

Também pode criar uma tabela ao usar outra tabela e escolher que colunas quern a tabela nova. Tenha em consideração que os dados da tabela existente serão usados para preencher as entradas da tabela nova.

A síntaxe é a seguinte:

```
CREATE TABLE new_table_name AS
SELECT column1, column2,...
FROM existing_table_name
WHERE ....;
```

Tal como aprendemos na sectção anterior:

- ‘SELECT’ especifica as colunas da tabela existente;
- ‘FROM’ especifica o nome da tabela existente;
- ‘WHERE’ pode ser usado se quisermos um conjunto de registos que preencham uma condição específica.

Exemplo:

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

‘DROP TABEL’ em SQL

Similarmente a ‘DROP DATABASE’, esta instrução apaga uma tabela existente numa base de dados.

Lembre-se que deve ter a certeza de que não precisa da informação que consta nessa tabela antes de a apagar.

Síntaxe:

```
DROP TABLE TableName;
```

Exemplo:

```
DROP TABLE Persons;
```

Também pode escolher apagar apenas os dados da atebla e não a tabela em si.

Imagine que cria uma tabela nova a partir de uma tabela existente, a tabela tem a estrutura que pretendia mas quer acrescentar novas entradas. A instrução ‘TRUNCATE TABLE’ é útil num caso destes.

Síntaxe:

```
TRUNCATE TABLE TableName;
```

Exemplo:

```
TRUNCATE TABLE Persons;
```

‘ALTER TABLE’ em SQL

A instrução ‘ALTER TABLE’ pode adicionar, apagar ou alterar colunas de uma tabela existente. Pode também ser usada para acrescentar ou eliminar limitações de uma tabela

Vejamos a síntaxe usada para acrescentar uma coluna:

```
ALTER TABLE TableName
ADD column_name datatype;
```

Esta instrução assemelha-se à forma como criamos uma tabela ao determinar o nome da coluna e o tipo de dados a incluir nessa coluna.

Exemplo:

```
ALTER TABLE Customers
ADD Email varchar(255);
```

Para apagar uma coluna de uma tabela, usamos a instrução 'DROP'.

Lembre-se que alguns sistemas de bases de dados não permitem que os utilizadores apaguem colunas.

Síntaxe:

```
ALTER TABLE TableName
DROP COLUMN ColumnName;
```

Por exemplo, vamos apagar a coluna que criámos:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

Para alterar o tipo de dados de uma coluna, pode usar a seguinte instrução dependendo do SGBD que está a usar:

- 'ALTER COLUMN' (para SQL Server/MS Access);
- 'MODIFY COLUMN' (para My SQL/ Oracle anterior à versão 10G);
- 'MODIFY' (para Oracle version 10G e mais recentes).

Síntaxe:

```
ALTER TABLE TableName
ALTER COLUMN ColumnName datatype;
```

Note-se que a segunda instrução é a que muda de 'ALTER COLUMN' para 'MODIFY COLUMN' ou 'MODIFY', dependendo do SGBD que está a usar. O resto mantém-se inalterável.

Vejamos agora um exemplo. A table em baixo é a “Persons” e contém informação sobre pessoas diferentes.

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Tabela 50 *– Exemplo ‘ALTER TABLE’ (**Fonte:** https://www.w3schools.com/sql/sql_alter.asp)*

Imaginemos que queremos acrescentar a esta tabela uma coluna com o nome “DateofBirth”. Para tal, usaremos a instrução seguinte:

```
ALTER TABLE Persons
ADD DateofBirth date;
```

A nova coluna que acrescentamos à tabela tem como tipo de dados *date*, o que significa que armazena dados em formato de data. Em baixo, é possível ver a tabela com as colunas acrescentadas.

ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Tabela 51 *– Exemplo ‘ALTER TABLE’ (**Fonte**: https://www.w3schools.com/sql/sql_alter.asp)*

Contudo, caso mude de ideias e queira alterar o tipo de dados da coluna nova, pode usar a instrução ‘ALTER COLUMN’. Por exemplo, Podemos alterar o tipo de dadosda nova coluna para *year* (ano). Para tal, use a seguinte instrução:

```
ALTER TABLE Persons
ALTER COLUMN DateofBirth year;
```

O tipo de dados *year* apresenta um ano em formato numérico com 2 ou 4 dígitos.

Para apagar a coluna que acabamos de alterar, usamos a instrução ‘DROP COLUMN’.

```
ALTER TABLE Persons
DROP COLUMN DateofBirth;
```

Neste caso, a nossa tabela voltará ao que era no início.

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Tabela 52 *– Exemplo 'ALTER TABLE' (**Fonte**: https://www.w3schools.com/sql/sql_alter.asp)*

'CONSTRAINTS' em SQL

Em SQL, 'Constraints' é usado quando a tabela é criada com a instrução 'CREATE TABLE' ou depois da tabela ter sido criada com a instrução 'ALTER TABLE'.

Síntaxe:

```
CREATE TABLE table_name (
        column1 datatype constraint,
        column2 datatype constraint,
        column3 datatype constraint,
        ....
);
```

O termo 'Constraints' (limitações) é usado para especificar um conjunto de regras e restrições que se aplicam a uma coluna ou tabela. Estas regras e restrições são usadas para assegurar a integridade, rigor e fiabilidade dos dados. Quando aplicado a uma tabela, todas as colunas têm de ceder às limitações impostas.

As seguintes *constraints* são as mais comuns:

- 'NOT NULL';

- ‘UNIQUE’;
- ‘PRIMARY KEY’;
- ‘FOREIGN KEY’;
- ‘CHECK’;
- ‘DEFAULT’;
- ‘CREATE INDEX’.

Iremos agora abordar cada uma destas *constraints* para explicar o seu uso e síntaxe através de exemplos.

‘NOT NULL’ em SQL

Em SQL, as colunas podem ter valores ‘null’ por defeito. A *constraint* ‘NOT NULL’ é usada para evitar valores ‘null’ em colunas. Isto é importante para assegurar que todos os campos necessários são preenchidos quando uma nova entrada é introduzida na tabela.

Imaginemos que queremos criar uma tabela com o nome “Persons” e queremos garantir que as colunas “ID”, “LastName” e “FirstName” não têm qualquer valor ‘null’:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255) NOT NULL,
        Age int
);
```

Se quiser alterar uma tabela ao acrescentar limitações, pode usar a seguinte instrução:

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

A limitação ‘UNIQUE’ em SQL

A limitação ‘UNIQUE’ é usada para assegurar que todos os valores de uma coluna não se repetem nas linhas da tabela. Para clarificar, pense na variante ‘ID’. Não é conveniente que duas pessoas tenham a mesma ‘ID’, usamos então a limitação ‘UNIQUE’ para o evitar.

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
        ID int NOT NULL UNIQUE,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

MySQL:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        UNIQUE (ID)
);
```

Como pode ver, dependendo do SGBD que está a usar, há alguns ajustes nos quais a limitação ‘UNIQUE’ é incluída no código.

Caso queira definir uma limitação ‘UNIQUE’ em várias colunas, use o seguinte:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
```

```
        FirstName varchar(255),
        Age int,
        CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

Também pdoemos acrescentar uma limitação 'UNIQUE' depois de a tabela ter sido criada através da instrução 'ALTER TABLE', que aprendemos anterioremente.

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD UNIQUE (ID);
```

Para definir uma limitação 'UNIQUE' em várias colunas já existentes, usamos a seguinte instrução:

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Persons UNIQUE (ID, LastName);
```

Para eliminar a limitação 'UNIQUE', usamos a seguinte instrução:

MySQL:

```
ALTER TABLE Persons
DROP INDEX UC_Persons;
```

SQL Server/Oracle/ MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT UC_Persons;
```

A limitação 'PRIMARY KEY' em SQL

A limitação 'PRIMARY KEY' é usada apenas para identificar cada linha ou registo numa tabela. Note-se que *primary keys* deve conter valores únicos, mas não pode conter valores *null*.

Uma tabela pode ter apenas **UM** *primary key*, que deve ter uma ou várias colunas.

SQL Server/Oracle/MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

O exemplo em baixo permite definir uma limitação 'PRIMARY KEY' em várias colunas:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

Note-se que a 'PRIMARY KEY' continua a ser apenas uma, mas o valor abrange duas colunas.

Podemos também aplicar uma limitação 'PRIMARY KEY' a uma tabela existente através da seguinte instrução:

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

Para acrescentar e definir uma limitação 'PRIMARY KEY', use a seguinte instrução:

```
ALTER TABLE Persons
ADD CONSTRAINT PK_Persons PRIMARY KEY (ID, LastName);
```

Para eliminar uma limitação 'PRIMARY KEY', use a seguinte instrução, de acordo com o seu SGBDR:

MySQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

A limitação 'FOREIGN KEY' em SQL

A limitação 'FOREIGN KEY' representa as colunas de uma tabela que estão ligadas a uma limitação 'PRIMARY KEY' de outra tabela. À tabela que tem a limitação 'FOREIGN KEY'chama-se chama-se *child table*, e à tabela 'PRIMARY KEY' chama-se *referenced table* ou *parent table*.

Este tipo de limitação é usado para prevenir quaisquer ações que poderiam destruir ligações entre tabelas *child* e *parent*.

Consideremos as seguintes tabelas:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

***Tabela 53** – Exemplo de 'FOREIGN KEY' com tabela 'Persons' (**Fonte**: https://www.w3schools.com/sql/sql_foreignkey.asp)*

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

***Tabela 54** – Exemplo de 'FOREIGN KEY' com tabela 'Oders' (**Fonte**: https://www.w3schools.com/sql/sql_foreignkey.asp)*

Estas duas tabelas estão ligadas à coluna "PersonID". A *primary key* está na tabela 'Persons', e a *foreign key* corresponde à "PersonID" na tabela 'Orders'.

A limitação 'FOREIGN KEY' previne a inserção de dados inválidos na coluna *foreign key*, porque está ligada à tabela e os seus valores têm de ser idênticos.

Para usar a limitação 'FOREIGN KEY' ao criar uma tabela, podemos usar a seguinte instrução, de acordo com o nosso SGBDR:

SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
```

```
        OrderNumber int NOT NULL,
        PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

MySQL:

```
CREATE TABLE Orders (
        OrderID int NOT NULL,
        OrderNumber int NOT NULL,
        PersonID int,
        PRIMARY KEY (OrderID),
        FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

Esta instrução ligou a tabela “Orders” À tabela “Persons” com a limitação ‘FOREIGN KEY’ tendo por base a coluna “PersonID”.

A limitação ‘CHECK’ em SQL

A limitação ‘CHECK’ é usada para especificar valores numa coluna ou em determinadas colunas de uma tabela tendo por base valores encontrados noutras colunas da mesma linha.

Exemplo de limitação ‘CHECK’ em ‘CREATE TABLE’

O exemplo seguinte é usado par agarantir que uma pessoa não tem menos de 18 anos, para tal é acrescentada a limitação ‘CHECK’ à coluna “Age”:

MySQL:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
```

```
        CHECK (Age>=18)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int CHECK (Age>=18)
);
```

Se quiser nomear uma limitação 'CHECK' e usá-la em várias colunas, pode usar a seguinte instrução:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        City varchar(255),
        CONSTRAINT CHK_Person CHECK (Age>=18 AND City= 'Sandnes')
);
```

Exemplo de limitação 'CHECK' em 'ALTER TABLE'

Para criar uma limitação para uma tabela já existente, use a seguinte instrução:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
To name a constraint and create it on multiple columns, you can use:
ALTER TABLE Persons
```

```
ADD CONSTRAINT CHK_Person CHECK (Age>=18 AND City= ‘Sandnes’);
```

Exemplo de limitação ‘DROP a CHECK’

Para eliminar uma limitação ‘CEHCK’, pode usar a seguinte instrução de acordo com a o seu SGBDR:

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;
```

MySQL:

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

A limitação ‘DEFAULT’ em SQL

A limitação ‘DEFAULT’ é usada para especificar um valor por defeito para uma coluna. Se não existirem valores especificados, o valor por defeito será acrescentado a todos os novos registos.

Exemplo de limitação ‘DEFAULT’ em ‘CREATE TABLE’

O exemplo seguinte acrescenta um valor por defeito à coluna “City” quando a tabela “Persons” é criada:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
```

Esta limitação também pode ser uasada para inserir valores de Sistema com funções como ‘GETDATE()’:

```
CREATE TABLE Orders (
        ID int NOT NULL,
        OrderNumber int NOT NULL,
        OrderDate date DEFAULT GETDATE()
);
```

Exemplo de limitação ‘DEFAULT’ em ‘ALTER TABLE’

Neste exemplo, a coluna “Cirty” é usada para criar uma limitação ‘DEFAULT’ quando estamos a alterar uma tabela existente:

MySQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

SQL Server:

```
ALTER TABLE Persons
ADD CONSTRAINT df_City
DEFAULT 'Sandnes' FOR City;
```

MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City SET DEFAULT ‘Sandnes’;
```

Oracle:

```
ALTER TABLE Persons
MODIFY City DEFAULT 'Sandnes';
```

Exemplo de limitação 'DROP a DEFAULT'

MySQL:

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT;
```

O comando 'CREATE INDEX' em SQL

A instrução 'CREATE INDEX' cria um índice numa tabela. Os índices são úteis quando queremos aceder a dados rapidamente.

Note-se que a atualização de tabelas com índices é mais demorada em comparação com a atualização de tabelas sem índices. Nesta senda, sugere-se a criação de índices apenas em colunas frequentemente consultadas.
Para operar o comando 'CREATE INDEX' numa tabela em que são permitidos valores duplicados, use a seguinte síntaxe:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

Para operar o comando 'CREATE UNIQUE INDEX' numa tabela em que não são permitidos valores duplicados, use a seguinte síntaxe:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

Lembre-se que a criação de índices varia de base de dados para base de dados, por isso é importante verificar sempre qual a síntaxe a utilizar para criar um na sua base de dados.

Exemplos de 'CREATE INDEX'

Neste exemplo, vamos criar um índice na coluna "LastName" ao especificar o nome "idx_lastname":

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

Para criar um índice numa combinação de colunas, use a seguinte instrução:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

Se quiser, pode acrescentar mais colunas entre os parêntesis.

Exemplos de 'DROP INDEX'

Se quiser apagar um índice, use a seguinte instrução de acordo com o eu SGBDR:

MS Access:

```
DROP INDEX index_name ON table_name;
```

SQL Server:

```
DROP INDEX table_name.index_name;
```

DB2/Oracle:

```
DROP INDEX index_name;
```

MySQL:

```
ALTER TABLE table_name
DROP INDEX index_name;
```

Auto Increment em SQL

O *Auto increment* é usado para gerar automaticamente números únicos quando um novo registo é introduzido numa tabela. Isto é normalmente usado em *primary key* para assegurar que nenhuma pessoa tem a mesma *ID*.

Este recurso usa diferentes síntaxes em MySQL, SQL Server, Access e Oracle. Em seguida, iremos ver alguns exemplos.

MySQL:

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

Em MySQL, 'AUTO_INCREMENT' acrescenta o elemento *auto-increment* por defeito, o valor estabelecido é 1 e este vai aumentando 1 de cada vez.

Se quer que a sequência inicie num valor diferente, use a seguinte instrução:

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

Se introduzir um novo registo na tebla "Persons", não terá de especificar o valor para a coluna "PersonsID", pois este será Gerado automaticamente:

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen');
```

SQL Server

Aqui, estamos a usar o mesmo exemplo que em cima, no qual a coluna “PersonsID” é usada como a p*rimary key na tabela “Persons”:*

```
CREATE TABLE Persons (
        Personid int IDENTITY(1,1) PRIMARY KEY,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

Em SQL Server, o recurso *auto-increment* usa a palavra ‘IDENTIFY’ para ser ativado. Os dois avlores entre parêntesis indicam (o valor inicial e o valor a acrescentar para cada novo registo). Iniciará em 1 e irá aumentando de 1 em 1 a cada valor novo acrescentado.

Por exemplo, se quiser iniciar em 10 e aumentar esse valor de 5 em 5 a cada registo acrescentado, terá de escrever o seguinte ‘IDENTIFY (10,5)’.

Ao acrescentar novos registos não precisa de especificar a “PersonID”, esta será gerada automaticamente, como demonstrado no exemplo em cima.

MS Access

```
CREATE TABLE Persons (
        Personid AUTOINCREMENT PRIMARY KEY,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

O MS Access usa a palavra-chave 'AUTOINCREMENT' para ativar o recurso *auto-increment.* Similarmente aos outros dois, o valor inicial é 1 e a cada novo registo é acrescentado é acrescentado 1.

Pode indicar diferentes valores para valor inicial, como 10, e estabelecer que a cada novo registo o valor vá crescendo de 5 em 5 com "AUTOINCREMENT(10,5)".

Novamente, note que cada vez que acrescentamos um novo registo não é necessário especificar o valor "PersonID", este é gerado automaticamente.

Oracle

Em Oracle, o código é um pouco mais complicado. Para criar um campo *auto-increment*, é necessário criar uma sequência de números.

```
CREATE SEQUENCE seq_person
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10;
```

Esta sequência cria um elemento sequência chamado "seq_person", estabelece o valor inicial mínimo (neste caso é 1), e especifíca o aumento por 1. O *cache* especifíca quantos valores de sequência devem ser armazenados na memória para um acesso mais rápido.

Ao contrário dos exemplos anteriores, para introduzir um novo registo na tabela "Persons", é necessário usar a função *nextval.* Esta função é usada para obter o próximo valor do elemento sequência que criámos.

```
INSERT INTO Persons (Personid,FirstName,LastName)
VALUES (seq_person.nextval,'Lars','Monsen');
```

Neste exemplo, podemos ver que a coluna “PersonsID” é selecionada para ser atribuído o próximo número do elemento sequência “seq_person” que criámos.

Datas em SQL

Um dos aspetos mais difíceis ao usar datas é assegurar que o formato que estamos a tentar introduzir é o mesmo da coluna referente a datas na base de dados.

É importante notar que os dados que contêm apenas datas funcionarão como é expectável em consultas. Contudo, se houver também informação referente à hora, as coisas complicam-se um bocado.

Tipos de datas encontrados em MySQL:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

Tipos de data encontrados em SQL Server:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: a unique number

Lembre-se que os tipos de data são escolhidos aquando a criação de uma nova tabela na sua base de dados.

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
2	Camembert Pierrot	2008-11-09
3	Mozzarella di Giovanni	2008-11-11
4	Mascarpone Fabioli	2008-10-29

***Tabela 55** – Exemplos de datas com a tabela "Orders" (**Fonte**: https://www.w3schools.com/sql/sql_dates.asp)*

Usaremos a tabela "Orders" no nosso exemplo para selecionar os registos com a "OrderDate" "2088-11-11".

Exemplo:

```
SELECT *
FROM Orders
WHERE OrderDate='2008-11-11';
```

O resultado esperado assemelhar-se-á a isto:

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
3	Mozzarella di Giovanni	2008-11-11

***Tabela 56** – Exemplo de resultado de pesquisa de "OrderDate" em "Dates" (**Fonte**: https://www.w3schools.com/sql/sql_dates.asp)*

Note-se que duas datas podem ser facilmente comparadas quando não há registo de hora envolvido.

Suponha que tem a tabela “Orders”, mas que a coluna “OrderDate” contém também o registo da hora.

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11 13:23:44
2	Camembert Pierrot	2008-11-09 15:45:21
3	Mozzarella di Giovanni	2008-11-11 11:12:01
4	Mascarpone Fabioli	2008-10-29 14:56:59

Tabela 57 *– Tabela “Orders” com registo de hora em “Dates” (**Fonte**: https://www.w3schools.com/sql/sql_dates.asp)*

Neste exemplo, se tentar usar a mesma pesquisa que usamos em cima não obterá qualquer resultado, pois a pesquisa não considera o registo da hora. É recomendado que o registo da hora não seja usado, a menos que necessário.

```
SELECT *
FROM Orders
WHERE OrderDate='2008-11-11';
```

Views em SQL

Em SQL, uma *view* é uma tabela virtual de um conjunto de resultados criado a partir de uma determinada pesquisa. Uma *view* é útil quando queremos visualizar e apresentar dados através de uma combinação de tabelas.

Síntaxe:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note-se que sempre que o utilizador consulta uma *view, esta* exibe dados atualizados desde que a base de dados recrie a tabela virtual.

Exemplo de como consultar todos os clientes do Brasil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

Para consultar a *view:*

```
SELECT * FROM [Brazil Customers];
```

Outro exemplo é criar uma *view* que selecione todos os produtos da tabela “Products” com o preço superior ao preço médio:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

Para consultar a *view* em cima, use a seguinte instrução:

```
SELECT * FROM [Products Above Average Price];
```

Para atualizar a *view* em cima, use a instrução ‘CREATE OR REPLACE VIEW’:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

O exemplo seguinte adiciona a coluna “City” à *view* “Brazil customers” que criámos anteriormente:

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
```

```
WHERE Country = 'Brazil';
```

Para apagar uma *view*, use a instrução 'DROP VIEW':

```
DROP VIEW view_name;
```

Por exemplo, suponha que quer apagar a view "Brazil customers":

```
DROP VIEW [Brazil Customers];
```

4.3. Referências SQL

Palavras-chave SQL

Palavra-chave	Descrição
ADD	Adiciona a coluna a uma tabela existente
ADD CONSTRAINT	Adiciona uma limitação depois de uma tabela ter sido criada
ALL	Apresenta resultados como verdadeiros se os valores de subconsulta corresponderem à condição
ALTER	Acrescenta, apaga ou altera colunas de uma tabela, ou muda o tipo de dados da coluna de uma tabela
ALTER COLUMN	Altera o tipo de dados da coluna de uma tabela
ALTER TABLE	Acrescenta, apaga ou altera as colunas de uma tabela
AND	Apenas inclui linhas em que ambas as condições sejam verdade
ANY	Apresenta resultados como verdadeiros se qualquer um dos valores da subconsulta corresponder à condição

AS	Atribui um nome nove a uma coluna ou tabela com um alias
ASC	Organiza os resultados por ordem crescente
BACKUP DATABASE	Cria uma cópia de segurança de uma base de dados existente
BETWEEN	Seleciona valores dentro de um dado intervalo
CASE	Cria diferentes resultados com base nas condições
CHECK	Uma limitação que restringe o valor a ser introduzido em cada coluna
COLUMN	Altera o tipo de dados de uma coluna ou apaga a coluna de uma tabela
CONSTRAINT	Acrescenta ou elimina uma limitação
CREATE	Cria uma base de dado, um índice, *view*, tabela ou procedimento
CREATE DATABASE	Cria uma nova base de dados SQL

CREATE INDEX	Cria um índice numa tabela (permite valores duplicados)
CREATE OR REPLACE VIEW	Atualiza uma *view*
CREATE TABLE	Cria uma nova tabela numa base de dados
CREATE PROCEDURE	Cria um processo armazenado
CREATE UNIQUE INDEX	Cria um único índice numa tabela (não permite valores duplicados)
CREATE VIEW	Cria uma *view* com base no conjunto de resultados de uma instrução 'SELECT'
DATABASE	Cria ou apaga uma base de dados SQL
DEFAULT	Uma limitação que faculta um valor por defeito para uma coluna
DELETE	Apaga linhas de uma tabela
DESC	Apresenta os resultados por ordem decrescente

DISTINCT	Seleciona apenas valores diferentes
DROP	Apaga uma coluna, limitações, bases de dados, índices, tabelas ou *views*
DROP COLUMN	Apaga a coluna de uma tabela
DROP CONSTRAINT	Apaga uma limitação 'UNIQUE', 'PRIMARY KEY', 'FOREIGN KEY' ou 'CHECK'
DROP DATABASE	Apaga uma base de daos SQL existente
DROP DEFAULT	Apaga uma limitação 'DEFAULT'
DROP INDEX	Apaga o índice de uma tabela
DROP TABLE	Apaga uma tabela de uma base de dados
DROP VIEW	Apaga uma *view*
EXEC	Executa um procedimento armazenado
EXISTS	Testa a existência de registos numa subconsulta

FOREIGN KEY	Uma limitação usada para associar duas tabelas
FROM	Especifica de que tabela selecionar ou apagar dados
FULL OUTER JOIN	Apresenta todas as linhas nas quais existe correspondência entre a tabela da esquerda ou da direita
GROUP BY	Agrupa os resultados (usada com as funções combinadas: 'COUNT', 'MAX', 'MIN', 'SUM', 'AVG')
HAVING	Usado em vez de 'WHERE' com funções combinadas
IN	Permite especificar vários valores numa instrução 'WHERE'
INDEX	Cria ou apaga um índice numa tabela
INNER JOIN	Apresenta linhas que têm valores correspondentes em ambas as tabelas
INSERT INTO	Insere novas linhas numa tabela
INSERT INTO SELECT	Copia dados de uma tabela para outra

IS NULL	Testa valores vazios (*null)*
IS NOT NULL	Testa valores não vazios (not *null)*
JOIN	Junta tabelas
LEFT JOIN	Exibe todas as linhas da tabela da esquerda e as linhas correspondentes da tabela da direita
LIKE	Procura um padrão específico numa coluna
LIMIT	Determina o número de registos a apresentar nos resultados
NOT	Inclui apenas linhas nasquais a condição não seja verdade
NOT NULL	Uma limitação que impõe que uma coluna não aceite valores *null*
OR	Inclui linhas em que uma ou outra condição sejam verdade
ORDER BY	Organiza os resultados por ordem crescente ou decrescente
OUTER JOIN	Exibe todas as linhas quando existe correspondência na tabela da esquerda ou da direita

PRIMARY KEY	Uma limitação que apenas identifica cada registo numa tabela da base de dados
PROCEDURE	Um procedimento armazenado
RIGHT JOIN	Apresenta todas as linhas da tabela da direita e as linhas correspondentes da tabela da esquerda
ROWNUM	Especifica o número de registos a apresentar nos resultados
SELECT	Seleciona dados de uma base de dados
SELECT DISTINCT	Seleciona apenas valores diferentes
SELECT INTO	Copia dados de uma tabela para um atabela nova
SELECT TOP	Determina o número de registos a apresentar nos resultados
SET	Determina que colunas e valores devem ser atualizados numa tabela
TABLE	Cria uma tabela ou adiciona, apaga ou altera colunas de uma tabelaCreates a table, or adds, deletes, or modifies columns in a table, ou apaga uma tabela ou os dados de uma tabela

TOP	Determina o número de registos a apresentar nos resultados
TRUNCATE TABLE	Apaga os dados de uma tabela, mas não a tabela
UNION	Combina os resultados de duas ou mais instruções 'SELECT' (apenas valores diferentes)
UNION ALL	Combina os resultados de duas ou mais instruções 'SELECT' (permite valores duplicados)
UNIQUE	Uma limitação que garante que todos os valores de uma coluna são únicos
UPDATE	Atualiza as linhas de uma tabela
VALUES	Determina os valores de uma instrução 'INSERT INTO'
VIEW	Cria, atualiza ou apaga uma *view*

Tabela 68 *– Palavras-chave SQL e respetivas descrições (**Fonte:** https://www.w3schools.com/sql/sql_ref_keywords.asp)*

Funções em MySQL

Para mais informações sobre funções usadas especificamente em MySQL, os aprendentes podem consultar o seguinte link.

Funções em SQL Server

Para mais informações sobre funções usadas especificamente em SQL Server, os aprendentes podem consultar o seguinte link

Funções em MS Access

Para mais informações sobre funções usadas especificamente em MS Access, os aprendentes podem consultar o seguinte link.

SQL Quick Ref

Para saber mais sobre instruções usadas em SQL e as suas síntaxes, os aprendentes podem consultar este link.

Dados em SQL

Regra geral, todas as colunas de uma tabela requerem um nome e tipo de dados.

Um programador SQL terá de decidir que tipo de dados irão ser armazenados em cada coluna quando criar a tabela. O tipo de dados é usado para a SQL compreender os dados que cada coluna irá conter e como esta nirá interagir com os dados.

Tenha com consideração que o tipo de dados pode ter nomes diferentes em bases de dados diferentes. Confirme sempre os documentos, mesmo que o nome seja o mesmo, pois os detalhes podem diferir em alguns aspetos, como, por exemplo, em tamanho.

Tipos de dados em MySQL (Versão 8.0)

O MySQL tem três tipos de dados: *string*, numérico e data/hora.

Tipos de dados *String*

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

***Tabela 58** – Tipos de dados 'String' (MySQL) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Tipos de dados Numéricos

Data type	Description
BIT(*size*)	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.
TINYINT(*size*)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(*size*)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255)
MEDIUMINT(*size*)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The *size* parameter specifies the maximum display width (which is 255)
INT(*size*)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)
INTEGER(*size*)	Equal to INT(size)
BIGINT(*size*)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The *size* parameter specifies the maximum display width (which is 255)
FLOAT(*size*, *d*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(*p*)	A floating point number. MySQL uses the *p* value to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(*size*, *d*)	A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter
DOUBLE PRECISION(*size*, *d*)	
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.
DEC(*size*, *d*)	Equal to DECIMAL(size,d)

Tabela 59 *– Tipos de dados Numéricos (MySQL) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Tipos de dados Data/Hora

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(*fsp*)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(*fsp*)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

Tabela 60 *– Tipos de dados Data/Hora (MySQL) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Tipos de dados em SQL Server

Tipos de dados *String*

Data type	Description	Max size	Storage
char(n)	Fixed width character string	8,000 characters	Defined width
varchar(n)	Variable width character string	8,000 characters	2 bytes + number of chars
varchar(max)	Variable width character string	1,073,741,824 characters	2 bytes + number of chars
text	Variable width character string	2GB of text data	4 bytes + number of chars
nchar	Fixed width Unicode string	4,000 characters	Defined width x 2
nvarchar	Variable width Unicode string	4,000 characters	
nvarchar(max)	Variable width Unicode string	536,870,912 characters	
ntext	Variable width Unicode string	2GB of text data	
binary(n)	Fixed width binary string	8,000 bytes	
varbinary	Variable width binary string	8,000 bytes	
varbinary(max)	Variable width binary string	2GB	
image	Variable width binary string	2GB	

***Tabela 61** – Tipos de dados 'String' (SQL Server) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Tipos de dados Numéricos

Data type	Description	Storage
bit	Integer that can be 0, 1, or NULL	
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
smallmoney	Monetary data from -214,748.3648 to 214,748.3647	4 bytes
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.	4 or 8 bytes
real	Floating precision number data from -3.40E + 38 to 3.40E + 38	4 bytes

***Tabela 62** – Tipos de dados Numéricos (SQL Server) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Tipos de dados Data/Hora

Data type	Description	Storage
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds	8 bytes
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds	6-8 bytes
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute	4 bytes
date	Store a date only. From January 1, 0001 to December 31, 9999	3 bytes
time	Store a time only to an accuracy of 100 nanoseconds	3-5 bytes
datetimeoffset	The same as datetime2 with the addition of a time zone offset	8-10 bytes
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable	

Tabela 63 *– Tipos de dados Data/Hora (SQL Server) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Outros tipos de dados

Data type	Description
sql_variant	Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML formatted data. Maximum 2GB
cursor	Stores a reference to a cursor used for database operations
table	Stores a result-set for later processing

Tabela 64 *– Outros tipos de dados (SQL Server) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Tipos de dados em MS Access

Data type	Description	Storage
Text	Use for text or combinations of text and numbers. 255 characters maximum	
Memo	Memo is used for larger amounts of text. Stores up to 65,536 characters. **Note:** You cannot sort a memo field. However, they are searchable	
Byte	Allows whole numbers from 0 to 255	1 byte
Integer	Allows whole numbers between -32,768 and 32,767	2 bytes
Long	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
Single	Single precision floating-point. Will handle most decimals	4 bytes
Double	Double precision floating-point. Will handle most decimals	8 bytes
Currency	Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places. **Tip:** You can choose which country's currency to use	8 bytes
AutoNumber	AutoNumber fields automatically give each record its own number, usually starting at 1	4 bytes
Date/Time	Use for dates and times	8 bytes
Yes/No	A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to -1 and 0). **Note:** Null values are not allowed in Yes/No fields	1 bit
Ole Object	Can store pictures, audio, video, or other BLOBs (Binary Large Objects)	up to 1GB
Hyperlink	Contain links to other files, including web pages	
Lookup Wizard	Let you type a list of options, which can then be chosen from a drop-down list	4 bytes

Tabela 65 *– Tipos de dados em Access (**Fonte:** https://www.w3schools.com/sql/sql_datatypes.asp)*

4.4. Exemplos SQL

Exemplos SQL

Existe uma vasta lista de exemplos no *website* W3Schools que os aprendentes podem consultar para estudo individual e praticar as suas competências em SQL.

Questionário SQL

Para os aprendentes que quiserem avaliar o seu conhecimento e competeências em SQL, por favor indique um dos seguintes *websites:*

- W3Schools
- Tutorialspoint

5. JavaScript

Informação do tópico

Tópico:

5. JavaScript

Pré-requisitos:

Para aprender JavaScript, os formandos devem saber o básico de HTML e CSS. Para um conhecimento prático de JavaScript e a maioria de projetos baseados na *web*, este conhecimento será suficiente. Para projetos e skills mais avançados, recomenda-se saber o conhecimento básico dos conceitos de OOP e de uma linguagem baseada no OOP (como o Java).

Carga Horária:

15 horas.

Descrição:

O JavaScript é uma linguagem de programação que permite ao programador efetuar mudanças no conteúdo de uma página *web* de uma forma dinâmica. O JavaScript é uma das linguagens de programação mais poderosas e flexíveis na *web*.

Este tópico cobre todos os conceitos fundamentais de programação, incluindo tipos de dados, operações, criar e usar variáveis, gerar resultados, estruturar código para tomar decisões em programas ou para repetir o mesmo bloco de código múltiplas vezes, criando e manipulando matrizes, definindo e usando funções, etc.

Quando os formandos estiverem confortáveis com o básico, estes irão avançar para o próximo nível que explica a ideia dos objetos, o Document Object Model (DOM) e o Navegador Object Model (BOM), da mesma forma que aprendem a usar objetos nativos do JavaScript como Date, Math, etc., e realizar esse tipo de conversões.

Para além disso, usarão outros conceitos avançados como *event listeners*, propagação de eventos, métodos de empréstimo de outros objetos, *hoisting*, codificar e descodificar dados JSON, assim como a exploração de uma visão geral detalhada das novas funcionalidades introduzidas no ECMA*Script* 6 (ou ES6).

Resultados de Aprendizagem:

Os formandos irão perceber como é que o JavaScript permite ao programador alterar o conteúdo de forma dinâmica e modificar o HTML ou a informação CSS no lado do cliente, quando a página *web* está a ser mostrada ao utilizador. Desta forma, os formandos irão estudar a estruturas do código JavaScript; vão saber como mudar o código HTML/CSS no carregamento da página e aprenderão a criar uma função e interligá-la a um evento.

Material necessário:

- Computador ou portátil
- Conexão à internet
- Editor de texto (*online* ou offline): Sublime Text/Brackets/W3Schools *online* editor

Cenário de aula:

O tempo total para este tópico é de 15 horas e caberá ao formador/professor decidir a quantidade de tempo que dedica a cada subtópico de aprendizagem. De forma a aproveitar ao máximo o tempo disponível, propomos que se use o material de aprendizagem criado para este projeto (apresentação PPT), que foram criados tendo em conta o uso aproveitado de tempo. Estas apresentações são compostas pelos seguintes elementos:

- Desenvolvimento do subtópico e as principais ideias a reter;
- Atividades/Exercicios propostos.

Pondo isto, se os professores/formadores seguirem a sequência lógica dos PPTs, ele/ela irá, certamente, ser capaz de completar a sessão no tempo limite estipulado. Estas apresentações podem também ser disponibilizadas aos formandos para um estudo individual.

Subtópicos:

- 5.1. JavaScript num nível básico
- 5.2. JavaScript e DOM
- 5.3. JavaScript e BOM
- 5.4. JavaScript Avançado

Recursos adicionais:

- Tutorial de JavaScript: w3schools
- Curso *Online* de JavaScript: CodeAcademy

5.1. JavaScript num nível básico

JavaScript (JS) é a linguagem mais abrangente no *scripting* da visão do cliente (o *scripting* da visão do cliente está associada à execução num navegador *da web*).JS pretende adicionar às páginas *web* efeitos interativos e dinâmicos através da manipulação do conteúdo devolvido de um servidor *web*.

O JavaScript foi inicialmente desenvolvido como FreeScript pelo programador de Netscape, Brendan Eich, em 1995. Foi mais tarde renomeado de JavaScript e tornou-se um standard ECMA (Associação Europeia de Fabricantes de Computadores), em 1997. Hoje em dia, o JavaScript é a norma da linguagem de *scripting* da visão do cliente para aplicações baseadas na *web* e é compatível com praticamente todos os motores de busca *web* disponíveis (Chrome, Firefox, Safari, etc.).

O JavaScript é uma linguagem orientada do objeto, tendo também algumas semelhanças na síntaxe da linguagem de programação Java, mesmo que não seja relacionado, de todo, ao Java.

O JavaScript pode ser usado para vários fins:

- Modificar o conteúdo de uma página *web* ao adicionar ou remover elementos;
- Alterar o estilo e posição dos elementos numa página *web*;
- Monitorizar eventos como o clicar do rato, *hover*, etc. e reagir a esses estímulos;
- Fazer e controlar transições na página *web*;
- Produzir alertas *pop-up* para mostrar mensagens de informação e aviso para o utilizador;
- Completar operações com base no contributo do utilizador e na exibição dos resultados;
- Validar o contributo do utilizador antes de o submeter no servidor;
- E muitos outros propósitos interessantes que serão mostrados posteriormente.

Iniciar com o JavaScript

Neste ponto, os formandos irão entender o quão simples pode ser adicionar interatividade numa página *web* usando o JavaScript.

Existem 3 formas de adicionar JS a uma página *web*:

a) Incorporar o código JavaScript entre a *tag* <*script*> e </*script*>;
b) Criar um ficheiro JavaScript com a extensão .js e depois abrir dentro da página através do atributo da *tag* <*script*>.
c) Colocar o código JavaScript diretamente dentro da *tag* HTML usando a *tag* especial de atributo como onclick, onmouseover, onkeypress, onload, etc.

a) Incorporar o código JavaScript entre a *tag* <*script*> e </*script*>;

O código JavaScript pode ser incorporado diretamente no interior de uma página *web* ao colocá-lo entre as *tags* <*script*> e <*script*>. A *tag* <*script*> indica ao navegador que as afirmações presentes devem ser interpretadas como um guião exequível e não como HTML, tal como mostrado no seguinte exemplo:

```
<!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!

Figura 1 *– Incorporar o código* JavaScript *entre a tag <script> e <scipt>.* (***Fonte:*** *https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php*)

b) Criar um ficheiro JavaScript com a extensão .js e depois abrir dentro da página através do atributo da *tag* <*script*>.

O código JavaScript pode ser colocado, também, em ficheiros separados com a extensão .js, sendo então chamado para o ficheiro no mesmo documento através do atributo src da *tag* <*script*>, assim:

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

Esta opção é especialmente útil caso o programador quiser que os mesmos *script*s estejam disponíveis em vários documentos. Ao seguir este procedimento, ele/ela vai evitar repetir a mesma tarefa várias vezes, e faz com que o seu *website* seja mais fácil de manter.

Seguindo para a afirmação em cima, será criado um ficheiro JavaScript denominado como “hello.js” e será introduzido o código seguinte:

```
1  // A function to display a message
2  function sayHello() {
3      alert("Hello World!");
4  }
5
6  // Call function on click of the button
7  document.getElementById("myBtn").onclick = sayHello;
```

Figura 2 *–Criar um ficheiro e utilizar a função em JS (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Neste momento, o programador pode utilizar o ficheiro JS acima numa *web*page usando a *tag* <*script*>, tal como exemplificado na imagem seguinte:

Figura 3 *– Criar um ficheiro e utilizar a função em JS (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

c) Colocar o código JavaScript diretamente dentro da *tag* HTML usando a *tag* especial de atributo como onclick, onmouseover, onkeypress, onload, etc.

O código JavaScript pode ser introduzido *inline* ao inseri-lo diretamente dentro da *tag* HTML através dos atributos especiais como onclick, onmouseover, onkeypress, onload, etc.

De qualquer forma, não é aconselhável que se coloque grandes quantidades de código JavaScript *inline,* pois pode desordenar o HTML com o JavaScript e fazer com que o código JS possa ser mais difícil de manter. A *Figura 4* mostra um exemplo (neste caso, uma mensagem de alerta é exibida ao clicar no elemento de botão):

Figura 4 *– Colocar código JS inline (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

O elemento <*script*> pode ser posicionado na secção <*head*> ou <*body*> de um documento HTML. No entanto, os *script*s devem ser preferencialmente posicionados no fim da secção do corpo, antes de fechar a *tag* <*/body*>. Este procedimento permite às páginas *web* carregar mais rapidamente, pois evita obstruções que possam acontecer na renderização inicial da página. Cada *tag* de <*script*> bloqueia o processo de renderização até este estar completamente descarregado e executado pelo código JavaScript, colocando-os, então, na secção de *head* (p.e.: o elemento <*head*>) do documento sem qualquer razão válida que irá impactar, significativamente, o desempenho do *website*.

Existem diferenças entre o *scripting* da visão de cliente e a visão de servidor. A linguagem de visão do cliente (p.e., o JavaScript ou VB*Script* são interpretados e executados pelo navegador da *web*, ao contrário do *scripting* da linguagem da visão

do servidor (p.e.; PHP, ASP, Java, Python, Ruby, etc.), o que corre num servidor *web* e o seu contributo é enviado de volta para o navegador de *web* em formato HTML.

O *scripting* da visão do cliente tem várias vantagens, comparativamente ao *scripting* tradicional da visão do sevidor. Por exemplo, o JavaScritpt pode ser usado para verificar se o utilizador inseriu dados inválidos em campos de formulário e mostrar a notificação, em tempo real, dos erros cometidos antes de submeter o formulário ao servidor *web* para a validação final dos dados e o processamento seguinte, de forma a evitar usos desnecessários de *bandwidth* de canais e do uso incorreto de recursos de sistemas de servidores.

Com isto, a resposta do *script* da visão do servidor é bastante mais lenta comparada com o *script* da visão do cliente, pois o *script* da visão do servidor é processado num computador remoto (e não num local).

Síntaxe do JavaScript

É altura de aprender como se escreve código JS.

Primeiro, é importante entender a síntaxe do JavaScript.

A síntaxe do JS é um **conjunto de regras** que compilam uma programação bem estruturada. JS envolve afirmações que são colocadas entre as *tag*s HTML <*script*> e </*script*> numa página *web*, ou dentro do ficheiro externo de JavaScript com a extensão .js.

A *Figura 5* apresenta um exemplo de uma afirmação em JS:

```
<!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

***Figura 5** – Exemplo de uma afirmação em* JavaScript *(**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Os formandos devem notar que o JavaScript é **sensível a maiúsculas e minúsculas**. Desta forma, as variáveis, palavras-chave de linguagem, nomes de funções e outros identificadores devem ser escritos sempre iguais. Desta forma, a variável myVar deve ser sempre escrita desta forma (e não, “MYVAR”, “myvar”, etc.). Isto é aplicável em **todos os casos**.

De acordo com os tópicos anteriores, o JavaScript permite escrever comentários ao longo das linhas de código. Os comentários são inseridos maioritariamente porque dão informação extra à fonte de código, mas também porque pode ajudar programadores a entender os seus códigos após algum tempo, é útil em trabalho de equipas, etc.

É possível adicionar tanto linhas únicas como várias linhas de comentários no JavaScript. Os comentários de linha única começam como uma dupla barra para a frente (//), seguindo-se o texto com o comentário:

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

***Figura 6** – Comentário de linha única em* JavaScript *Fonte: https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Para um comentário com mais de uma linha, o ponto de começo são uma barra e um asterisco (/*), acabando com um asterisco e uma barra (*/):

```
1  /* This is my first program
2  in JavaScript */
3  document.write("Hello World!");
```

***Figura 7** – Comentário com várias linhas em* JavaScript *(**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Variáveis em JavaScript

Para armazenar dados em JavaScript, os programadores criam variáveis. Estas são a chave de todas as linguagens de programação e são usadas para armazenar dados, por exemplo, uma linha de texto, números ou outros elementos. Quando necessitar, o programador pode determinar, atualizar ou recuperar dados ou valores guardados nas variáveis. As variáveis podem ser entendidas como nomes simbólicos para os valores.

Uma variável pode ser criada usando a palavra-chave var, onde o operador designado ("=") é usado para atribuir um valor à variável, como no seguinte exemplo: `var varName = value`

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

***Figura 8** – Criar uma variável (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Neste exemplo foram criadas três variáveis:
Número 1: valor da linha
Número 2: número
Número 3: valor *booleano*

Em JS, algumas variáveis são criadas com o objetivo de guardar valores como os comentários do utilizador. Posto isto, elas podem ser declaradas sem tere, nenhum valor inicial associado, como no seguinte exemplo:

```
1  // Declaring Variable
2  var userName;
3
4  // Assigning value
5  userName = "Clark Kent";
```

Figura 9 *– Criar uma variável sem qualquer valor inicial atribuído (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Para além disto, o programador pode também declarar várias variáveis e, numa única afirmação, definir os seus valores iniciais. Cada variável é separada com vírgulas:

```
1  // Declaring multiple Variables
2  var name = "Peter Parker", age = 21, isMarried = false;
3
4  /* Longer declarations can be written to span
5  multiple lines to improve the readability */
6  var name = "Peter Parker",
7  age = 21,
8  isMarried = false;
```

Figura 10 *– Declarar múltiplas variáveis (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

A última revisão de JavaScriot (ECMA*Script* 2015 ou ES6) introduziu duas novas palavras-chave para declarar variáveis: let e const.

A palavra-chave "*const*" funciona da mesma forma que a "*let*". No entanto, as variáveis declaradas com "*const*" não pode ser reatribuídas mais tarde no código, tal como no exemplo que se segue:

```
// 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
```

Figura 11 *– Declarar múltiplas variáveis (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Contrariamente à palavra-chave "*var*", que declara a variável *function-scoped*, tanto "let" como "const" são palavras-chave que declaram variáveis, delimitando-as a um nível de bloco ({}). A delimitação por bloco significa que uma nova delimitação é criada dentro de um par de chavetas.

As variáveis JavaScript têm **regras específicas** para serem nomeadas:

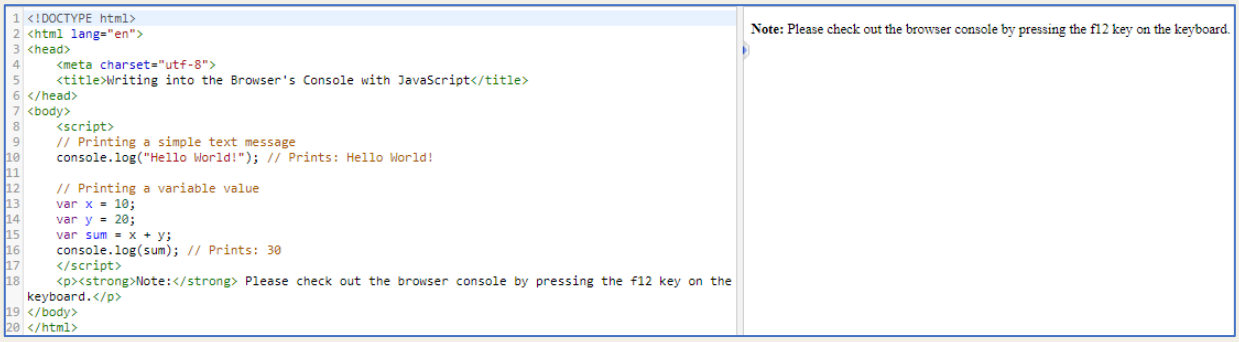
- O nome de uma variável deve começar com uma letra, *underscore* (_), ou cifrão ($).
- O nome de uma variável não pode começar por um número.
- O nome de uma variável só pode ter caracteres alfa-numéricos (A-Z, 0-9) e *underscores*.
- O nome de uma variável não pode conter espaços
- O nome de uma variável não pode ser uma palavra-chave do JavaScript ou uma palavra reservada para fins de programação JavaScript.

Gerar *outputs* em JavaScript

Existem certas situações em que os programadores podem precisar de criar *outputs* ("*resultados*") do código JS, por exemplo: verificar qual o valor de uma variável, escrever uma mensagem no controlo de navegador, etc. Em JavaScript, existem algumas maneiras de criar *outputs*, incluindo a sua inscrição na janela do navegador

ou no controlo do navegador, mostrando resultados nas caixas de diálogo, escrevendo os resultados num elemento HTML, etc.

Não é difícil criar *outputs* de uma mensagem ou escrever os dados no controlo do navegador (pode ser acedido ao carregar na tecla F12). Para tal, deve ser aplicado o *console*.log () – um método simples, contudo, bastante eficaz.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Writing into the Browser's Console with JavaScript</title>
</head>
<body>
    <script>
    // Printing a simple text message
    console.log("Hello World!"); // Prints: Hello World!

    // Printing a variable value
    var x = 10;
    var y = 20;
    var sum = x + y;
    console.log(sum); // Prints: 30
    </script>
    <p><strong>Note:</strong> Please check out the browser console by pressing the f12 key on the keyboard.</p>
</body>
</html>
```

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

Figura 12 *– Gerar outputs usando o console.log (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Para escrever o conteúdo do documento corrente enquanto o documento está a ser construído, é necessário *desconstruí-lo*. O método document. write () pode ser aplicado.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Writing into an Browser Window with JavaScript</title>
</head>
<body>
    <script>
    // Printing a simple text message
    document.write("Hello World!"); // Prints: Hello World!

    // Printing a variable value
    var x = 10;
    var y = 20;
    var sum = x + y;
    document.write(sum); // Prints: 30
    </script>
</body>
</html>
```

Hello World!30

Figura 13 *– Gerar resultados ao usar o console .log (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Se o método **document.write** estiver a ser usado depois da página ter sido carregada, irá reescrever todo o conteúdo existente naquele documento, como explicado neste *link*.

Caixas de diálogos de alerta podem também ser adicionados para mostrar os dados de mensagem ou resultado ao utilizador. Para criar este diálogo de alerta, o método alert() é aplicado, tal como no seguinte exemplo:

```
1  // Displaying a simple text message
2  alert("Hello World!"); // Outputs: Hello World!
3
4  // Displaying a variable value
5  var x = 10;
6  var y = 20;
7  var sum = x + y;
8  alert(sum); // Outputs: 30
```

Figura 14 *– Gerar caixas de diálogos de alerta (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Os resultados podem ser inseridos ou digitados dentro de um elementoo HTML usando a propriedade **innerHTML**. Contudo, o programador deve selecionar o elemento antes de digitar o *output*, usando o método getElementById ().

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>Writing into an HTML Element with JavaScript</title>
6  </head>
7  <body>
8      <p id="greet"></p>
9      <p id="result"></p>
10
11     <script>
12     // Writing text string inside an element
13     document.getElementById("greet").innerHTML = "Hello World!";
14
15     // Writing a variable value inside an element
16     var x = 10;
17     var y = 20;
18     var sum = x + y;
19     document.getElementById("result").innerHTML = sum;
20     </script>
21 </body>
22 </html>
```

Hello World!

30

Figura 15 *– Usando o método getElementById () (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Tipos de Dados em JavaScript

Os tipos de dados, essencialmente, estipulam qual é o tipo de dados que podem ser guardados e manipulados dentro de um programa. Existem seis tipos básicos de dados em JS, os quais podem ser divididos em três categorias principais:

- Primitivo (ou primário) – Linha, Número e Booleano são exemplos de tipos de dados primitivos, que apenas conseguem representar um valor de cada vez;
- Composto (ou referência) – Objeto, Matriz e Função (que são tipos de objetos) são tipos de dados compostos. Estes podem representar coleções de valores e de entidades mais complexas; e
- Tipos de dados especiais – Indefinidos e vazios são tipos de dados especiais.

O tipo de dados "*string*"

É usado para incorporar dados textuais (por exemplo, uma sequência de caracteres). As linhas são criadas usando aspas uma ou duas vezes, rodeando um ou mais caracteres:

```
<!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!

Figura 16 *– Tipos de linha de dados (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Deve ser realçado que as aspas podem estar incluídas dentro da linha, mas não podem combinar com as aspas finais, como neste exemplo.

O tipo de dados de número

O tipo de dados de números é especialmente útil na hora de exibir números positivos ou negativos, com ou sem casas decimais, ou números escritos usando uma anotação exponencial, por exemplo: 1.5e-4 (equivalente a $1.5x10^{-4}$).

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Number Data Type</title>
</head>
<body>
    <script>
    // Creating variables
    var a = 25;
    var b = 80.5;
    var c = 4.25e+6;
    var d = 4.25e-6;

    // Printing variable values
    document.write(a + "<br>");
    document.write(b + "<br>");
    document.write(c + "<br>");
    document.write(d);
    </script>
</body>
</html>
```

```
25
80.5
4250000
0.00000425
```

Figura 17 *–Tipo de linha de dados (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

O tipo de dados de número comprime também alguns valores especiais: infinito, -infinito e NaN. O infinito representa o infinito matemático (∞), que é maior que qualquer número. O infinito é o resultado da divisão de um número maior que 0 por 0, como pode ser verificado na Figura 18:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Infinity</title>
</head>
<body>
    <script>
    document.write(16 / 0);
    document.write("<br>");
    document.write(-16 / 0);
    document.write("<br>");
    document.write(16 / -0);
    </script>
</body>
</html>
```

```
Infinity
-Infinity
-Infinity
```

Figura 18 *– O Tipo de Dados de Número (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

No entanto, NaN representa um valor especial: *Not-a-Number* value. É o resultado de uma operação matemática inválida ou indefinida, por exemplo, quando se usa a raiz quadrada de -1 ou quando se divide zero por zero, entre outras.

O tipo de dados *booleano*

Existem dois valores que podem ser ingrados neste tipo de dados: verdadeiro ou falso. É, por norma, utilizado para valores de *stock* como sim (verdadeiro) ou não (falso), ligado (verdadeiro) ou desligado (falso), etc., tal como mostra o exemplo abaixo:

```
<!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
```

Figura 19 *– O Tipo de Dados Booleano (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Os valores *Booleanos*, desta forma, apresentam-se como o resultado de uma comparação num programa. Este exemplo mostra duas variáves e apresenta o resultado numa caixa de diálogo de alerta.

Tipos de Dados Indefinidos

O tipo de dados indefinidos pode apenas representar um valor – o valor especial indefinido. Se uma variável for declarada, mas não lhe tiver sido atribuída um valor, o valor deverá ser declarado como indefinido.

```
<!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!
```

Figura 20 *– Tipos de Dados Indefinidos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Tipo de Dado Nulo

Este é um tipo de dados especial, que apenas pode ter um único valor – o valor nulo. Um valor nulo significa que não existe um valor. Não é equivalente a uma linha vazia (“”) ou zero. É, simplesmente, nada. Esta variável pode ter os seus conteúdos esvaziados através da atribuição de um **valor nulo**.

```html
<!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
```

Figura 21 *– Tipo de Dados Nulos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Tipo de Dados do Objeto

O objeto é um tipo de dados multifacetado que permite armazenar coleções de dados. Um objeto contem propriedades definidas por um par de valores chave. Uma chave de propriedade (nome) é sempre uma linha, mas o seu valor pode ser qualquer tipo de dados (linhas, números, *booleanos*, ou tipos de dados complexos, como matrizes, funções e outros objetos). A forma mais simples de criar um objeto em JavaScript é mostrada de seguida:

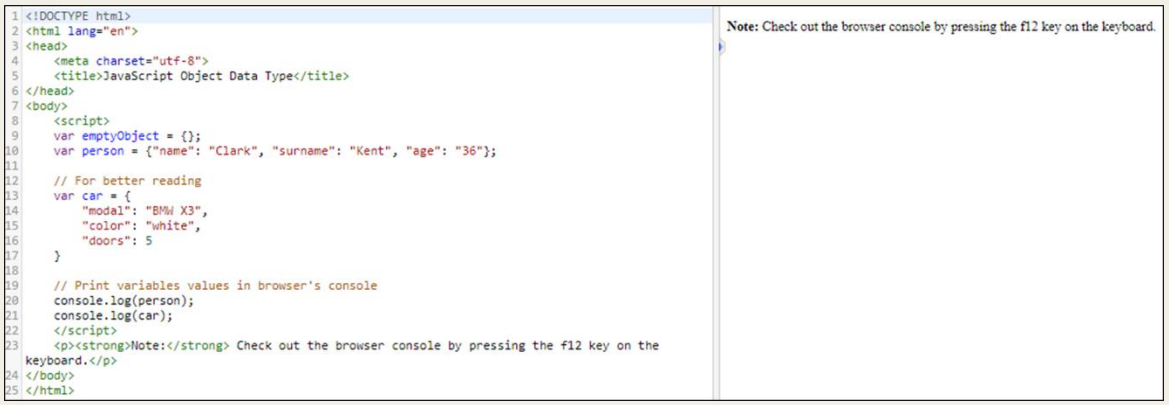


Figura 22 *– The simplest way for creating an object in JS (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

As citações em torno do nome da propriedade podem ser omitidas se o nome for um nome válido do JS. Isto significa que as citações são necessárias em torno de "first-name", mas não são obrigatórias em torno de "firstname". Assim, o objecto *car* na Figura 22 também pode ser escrito da seguinte forma.:

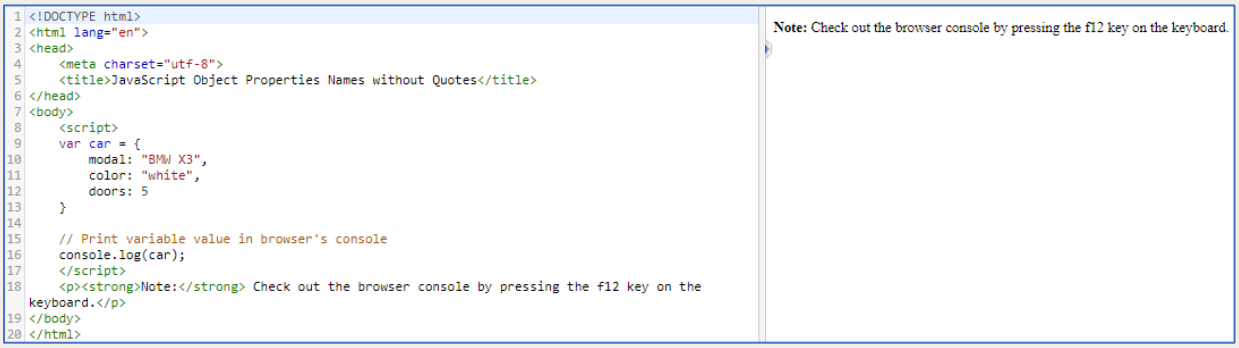


Figura 23 *– Reescrever o objeto car da Figura 22 (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Tipos de Dados de Matrizes

Uma matriz é um tipo de objeto usado para empacotar múltiplos valores numa só variável. Cada valor (também conhecido por 'elemento') numa matriz tem uma posição numérica, conhecida como o index, e poderá compreender dados de qualquer tipo numérico, linhas, *booleanos*, funções, objetos e até mesmo outras matrizes. O index da

matriz começa no 0 (zero), para que o primeiro elemento da matriz seja arr[0] e não arr[1].

A forma mais simples de fazer uma matriz é através da estipulação dos elementos da matriz em listas separadas por vírgulas, fechadas por parentises retos, tal como aqui:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Array Data Type</title>
</head>
<body>
    <script>
    // Creating arrays
    var colors = ["Red", "Yellow", "Green", "Orange"];
    var cities = ["London", "Paris", "New York"];

    // Printing array values
    document.write(colors[0] + "<br>");    // Output: Red
    document.write(cities[2]);    // Output: New York
    </script>
</body>
</html>
```

```
Red
New York
```

Figura 24 *– Reescrever o objeto car da Fig. 22 (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Tipos de Dados de Função

A função é um objeto que faz um bloco de código. As funções são objetos, daí ser possível designar-lhes variáveis, assim:

```
<!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!
```

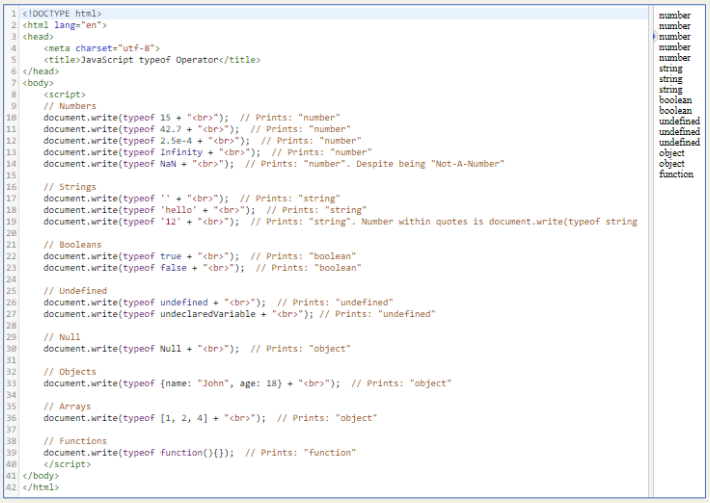
Figura 25 *– Tipos de Dados de Funções (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Na verdade, as funções podem ser usadas em qualquer lado, e quaisquer valores podem ser usados. Estas podem ser guardadas em variáveis, objetos e matrizes. As funções podem ser aprovadas como argumentos em outras funções, e podem ser devolvidas como funções.

Operador Typeof

O operador typeof pode ser usado para perceber o tipo de dados cobertos por uma variável. Pode ser usado com ou sem parênteses (typeof(x) ou typeof x). O operador typeof é, na sua maioria, benéfico no processo de valorização de diferentes tipos numa forma diferente.

No entanto, o programador deve ser cauteloso, pois pode produzir, em alguns casos, resultados imprevistos:



```
<!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
```

Figura 26 *– Operador Typeof (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Como destacado no exemplo da figura 26, quando um valor nulo é testado através do operadot typeof (linha 22), o “objeto” é devolvido como “nulo”.

Este é uma falha permanente no JavaScript, mas, como a maior parte do código da *web* é escrito com basde neste comportamento, e arranjar iria gerar ainda mais problemas, esta questão foi excluída pelo comitê que desenha e mantem o JavaScript.

Operadores de JavaScript

Os operadores são símbolos ou palavras-chave que informam o mecanismo de JavaScript a realizar alguma ação. Por exemplo, o símbolo de adição (+) é um operador que diz ao motor JavaScript para introduzir duas variáveis ou valores. No entanto, os sinais igual (==), maior que (>) ou menor que (<) são os operadores que dizem ao JavaScript para comparar duas variáveis, valores, etc.

Entre os diferentes operadores usados em JavaScript, os primeiros a serem descritos são os **Operadores Aritméticos** do JavaScript. Estes são colocados em ação de forma a realizarem operações aritméticas comuns (adições, subtrações, multiplicação e tudo o resto). A lista completa segue-se abaixo:

Operador	Descrição	Exemplo	Resultado
+	Adição	x + y	A soma de x e y.
-	Subtração	x - y	A diferença entre x e y.
*	Multiplicação	x * y	O produto de x com y.
/	Divisão	x / y	Quociente de x e y.
%	Percentagem	x % y	O resto de x dividido por y.

Tabela 1 *– Operadores Aritméticos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Aqui estão alguns exemplos práticos acerca de como é que os operadores anteriormente mencionados podem ser usados:

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>JavaScript Arithmetic Operators</title>
6  </head>
7  <body>
8      <script>
9      var x = 10;
10     var y = 4;
11     document.write(x + y); // Prints: 14
12     document.write("<br>");
13
14     document.write(x - y); // Prints: 6
15     document.write("<br>");
16
17     document.write(x * y); // Prints: 40
18     document.write("<br>");
19
20     document.write(x / y); // Prints: 2.5
21     document.write("<br>");
22
23     document.write(x % y); // Prints: 2
24     </script>
25 </body>
26 </html>
```

```
14
6
40
2.5
2
```

Figura 27 *– Operadores Aritméticos em JS (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Existe também **Operadores JS de linha (“*string*”)**:

Operador	Descrição	Exemplo	Resultado
+	Concatenação	str1 + str2	Concatenação de str1 e str2
+=	Atribuição de Concatenação	str1 += str2	Apêndice de str2 para str1

Tabela 2 *– Operadores de Linhas (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Um exemplo prático:

```
<!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!
```

Figura 28 *– Operadores Aritméticos de JS (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Os **Operadores de incremento e decremento do JS** são usados para incrementar/decrementar o valor de uma variável.

Operador	Nome	Efeito
++x	Pré-incremento	Incrementa x por um, depois retoma a x
x++	Pós-incremento	Retoma a x, depois incrementa x por um
--x	Pré-decremento	Decrementa x por um, depois retoma a x

x--	Pós-decremento	Retorna a x, depois decrementa x por um

Tabela 3 *– Operadores de Incremento e Decremento (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Um exemplo real:

```
<!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
```

Figura 29 *– Operadores de Incremento e Decremento de JS (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Para concentrar declarações condicionais são usados os **Operadores Lógicos de JS.**

Operador	Nome	Exemplo	Resultado
&&	E	x && y	Verdadeiro se ambos x e y forem verdadeiros
||	Ou	x || y	Verdadeiro se o x ou y for verdadeiro
!	Não	!x	Verdadeiro se x não for verdadeiro

*Tabela 4 – Operadores Lógicos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Aqui está um exemplo prático:

```
<!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.
```

*Figura 30 – Operadores de Incremento e Decremento de JS (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Em relação aos Operadores de Comparação de JS, os programadores usam-nos para compararem dois valores no estilo de *Booleano*.

Operador	Nome	Exemplo	Resultado
==	Igual	x == y	Verdadeiro se x for igual a y.
===	Idêntico	x === y	Verdadeiro se x for igual a y e se eles são do mesmo tipo.

!=	Não igual	x != y	Verdadeiro se o x não for igual ao y.
!==	Não identicos	x !== y	Verdadeiro se o x não for igual ao y ou se não forem do mesmo tipo.
<	Menor do que	x < y	Verdadeiro se x for menor que y.
>	Maior do que	x > y	Verdadeiro se o x for maior do que o y.
>=	Maior ou igual do que	x >= y	Verdadeiro se o x for maior ou igual do que y.
<=	Menor ou igual do que	x <= y	Verdadeiro se o x for menor ou igual do que o y.

Tabela 5 *– Operadores de Comparação (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Como funcionam:

```
<!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
```

Figura 31 *– Operadores de Incremento e Decremento de JS (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Eventos de JavaScript

Antes de aprofundarmos esta secção, é importante saber o que é, neste contexto, um evento. Um evento é algo que ocorre cada vez que um utilizador interage com a página *web*, através, por exemplo, do clique de um *link* ou de um botão, o texto dá entrada numa caixa de contributo ou uma área de texto, a seleção é feita na caixa de seleção, a tecla é carregada no teclado, o cursor do rato é movido, um formulário é submetido, etc. De vez em quando, o navegador é capaz de desencadear o evento por si próprio (por exemplo, quando está a carregar uma página).

Quando se dá um evento, os programadores podem usar um manipulador (ou ouvinte) de eventos JavaScript de forma e identificá-los, executando tarefas específicas. Através da conversão, os nomes para os manipuladores de eventos começam sempre pela palavra "*on*", para que um manipulador de um evento de clique seja apelidado de "onclick", da mesma forma, um manipulador para um evento de *load* é chamado de "onload", um manipulador para o evento de *blur* é chamado de "onblur", etc.

Existem várias formas de atribuir um manipulador de eventos. A forma mais simples de adicioná-los diretamente é no início da *tag* de um elemento HTML, através dos atributos especiais de um manipulador de eventos. Por exemplo, é possível atribuir um manipulador de clique a um elemento de botão, tal como se segue:

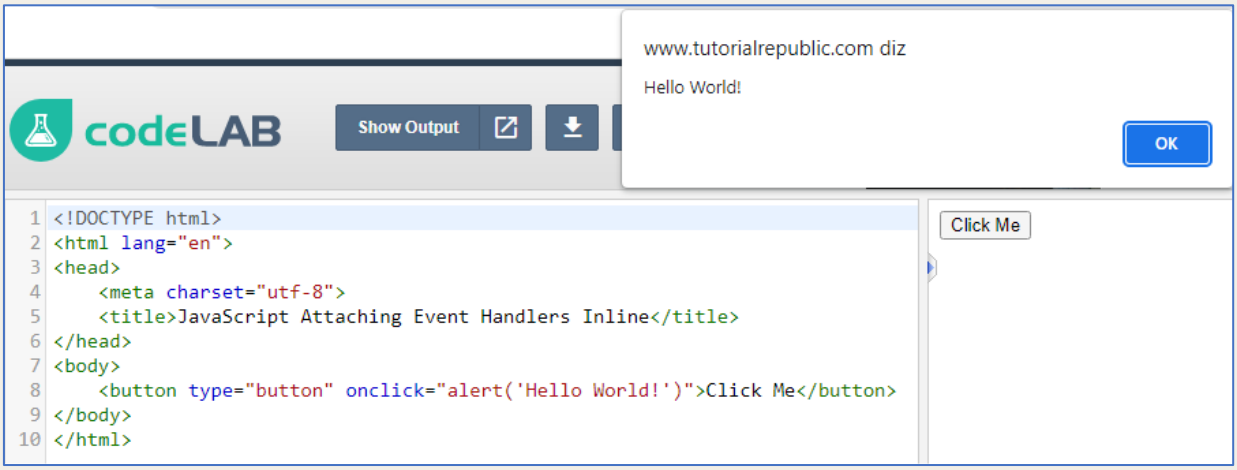


***Figura 32** – Operadores de Incremento e Decremento de JS (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

No entanto, para manter o JavaScript separado do HTML, os programadores podem definir um manipulador de eventos num ficheiro de JavaScript externo ou dentro das *tags* <*script*> e </*script*>, tal como no seguinte exemplo:

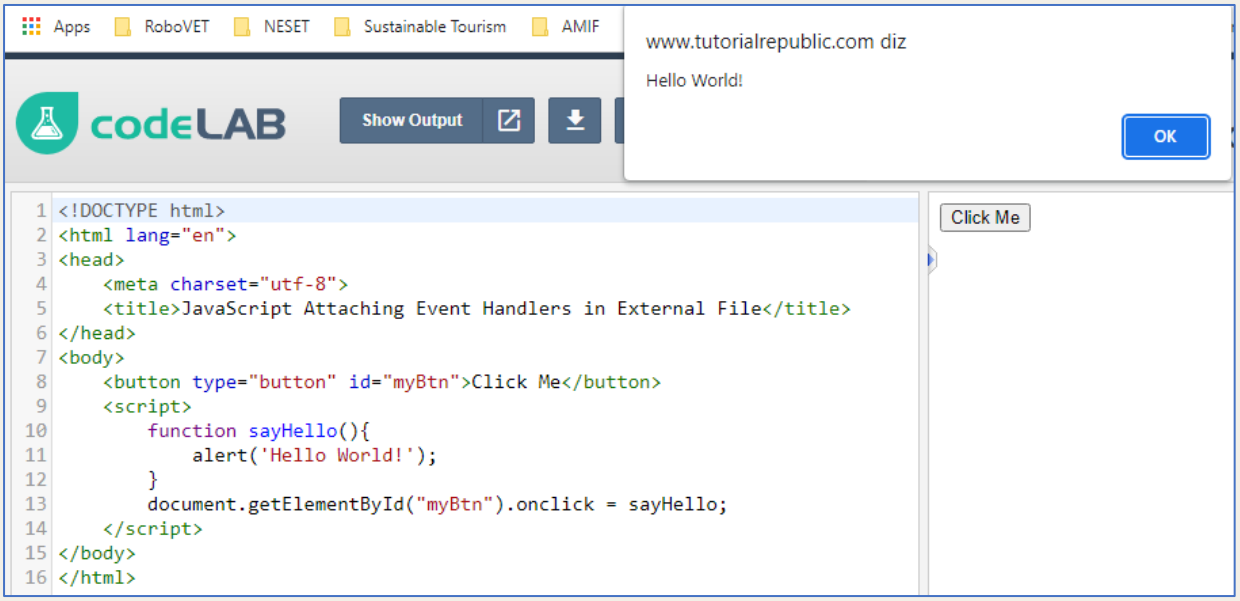


Figura 33 *– Atribuir um manipulador de eventos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Geralmente, os eventos podem ser categorizados em quatro grupos principais: eventos de rato, eventos de teclado, eventos de formulário e eventos de documento/janela. Existem muitos outros eventos, que serão explorados mais à frente. Os eventos mais úteis podem ser observados na lista que se segue:

- **Eventos de Rato**

Um evento de rato é ativado no momento em que o utilizador carrega num elemento, move o cursor do rato sobre um elemento, entre outros. Alguns eventos de rato importantes e respetivos manipuladores de eventos são os seguintes:

  - **O Evento do Clique (onclick):** O evento do clique acontece quando um utilizador carrega num elemento de uma página *web*. Por norma, tratar-se-ão de elementos respeitantes a um formulário. Um evento de clique

pode ser manipulado através de um manipulador *onclick* de um evento. O exemplo seguinte representa um caso de uma mensagem de alerta, que aparece no momento em que o utilizador clica no elemento.

- **O Evento Contextmenu (oncontextmenu):** ocorre quando o utilizador clica com o botão direito do rato num elemento, abrindo um menu relacionado com o elemento onde clicou. O manipulador de evento *oncontextmenu* manipula um evento de *context-menu*. Neste exemplo, é exibida uma mensagem de alerta logo após o utilizador clicar nos elementos com o botão direito do rato.
- **O Evento Mouseover (onmouseover):** dá-se quando o utilizador move o cursor do rato por cima de um elemento. Pode ser manipulado com o manipulador do evento *onmouseover*. O seguinte exemplo exibe uma mensagem de alerta quando o cursor do rato é colocado por cima dos elementos.
- **O Evento Mouseout (onmouseout):** acontece quando o utilizador move o cursor do rato para fora de um elemento. Pode ser manipulado ao usar o manipulador de eventos *onmouseout*. O seguinte exemplo reproduz uma mensagem de alerta assim que o evento *mouseout* ocorre.

- **Eventos de teclado**

Um evento de teclado acontece quando o utilizador carrega ou solta uma tecla no teclado. Alguns dos eventos de teclado mais importantes e os seus manipuladores de evento são os seguintes:

- **O Evento Keydown (onkeydown):** dá-se quando o utilizador carrega numa tecla de um teclado. Pode ser manipulado ao usar o manipulador do evento *onkeydown*. Este exemplo mostra uma mensagem de alerta cada vez que ocorre um evento *keydown*.
- **O Evento Keyup (onkeyup):** ocorre quando um utilizador solta uma tecla no teclado. Manipulado pelo manipulador de evento *onkeyup*. Este exemplo exibe uma mensagem de alerta quando este ocorre.

- **O Evento Keypress (onkeypress):** acontece quando um utilizador carrega numa tecla que tem um valor de caracter ligado a si. Por exemplo: Ctrl, Shift, Alt, Esc, teclas de setas, etc. Este evento não irá gerar um evento *keypress*, mas sim um evento *keydown* e *keyup*. O manipulador do evento *onkeypress* manipula o evento *keypress*. Pode ser verificado aqui através de uma mensagem alerta.

- **Eventos de Formulário**

Um evento de formulário é desencadeado quando o controlo de formulário recebe ou perde foco ou quando um utilizador modifica um valor de controlo do formulário (ex.: ao escrever texto numa entrada de texto), seleciona uma opção numa caixa de seleção, etc.

- **O Evento Focus (onfocus):** acontece quando o utilizador atribui foco a um elemento de uma página *web*. Este exemplo irá destacar o fundo de uma entrada de texto em cor amarela quando obtém o foco.
- **O Evento Blur (onblur):** acontece quando o utilizador retira o foco de uma janela ou de um elemento. Este pode ser manipulado com o mainipulador de eventos *onblur*. O seguinte exemplo irá mostrar uma mensagem de alerta assim que o elemento de entrada de texto falhar o enfoque.
- **O Evento Change (onchange):** acontece logo após o utilizador mudar o valor de um elemento de formulário. O manipulador do evento: *onchange*. Este exemplo mostra uma mensagem alerta quando a opção numa caixa de seleção for alterada.
- **O Evento Submit (onsubmit):** acontece apenas quando o utilizador submete um formulário numa página *web*. Manipulador de evento: *onsubmit*. Este exemplo mostra uma mensagem de alerta enquando se submete um formulário.

- **Eventos de Documentos/Janela**

Situações onde a página foi carregada ou redimensionada no navegador da janela podem, também, despoletar eventos.

  - **O Evento Load (onload):** acontece quando uma página *web* é completamente carregada num navegador *web*. Manipulador de evento: *onload*. O seguinte exemplo emite uma mensagem alerta assim que a página carrega.
  - **O Evento Unload (onunload):** quando o utilizador sai da *web*page atual, este evento ocorre. Manipulador do evento: *onunload*. Este exemplo exibe um alerta *pop-up* quando o utilizador tenta sair da página.
  - **O Evento Resize (onresize):** acontece quando o utilizador redimensiona, minimiza ou maximiza a janela do navegador. Manipulador de evento: *onresize*. Este exemplo exibe uma mensagem mesmo antes do utilizador redimensionar a janela do navegador.

Linhas em JavaScript

Em JavaScript, as linhas representam um papel principal na estrutura geral de uma página *web*, tratando-se de uma sequência de letras, números, caracteres especiais e valores aritméticos, podendo até mesmo ser uma combinação de todos os anteriores.

Estas podem ser criadas através do desdobramento de uma linha literal (ex.: linha de caracteres), quer estejam dentro de aspas únicas (‘) ou aspas duplas (‘’), como no exemplo que se segue:

```
<!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!
```

Figura 34 *– Exemplos de linhas em* JavaScript (***Fonte:*** *https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php*)

As aspas podem ser usadas dentro de uma linha, mas estas não devem corresponder às que rodeiam as linhas:

```
<!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'
```

Figura 35 *– Exemplos de linhas em* JavaScript *II* (***Fonte:*** *https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php*)

No entanto, aspas únicas podem ser postas dentro de uma linha com aspas únicas ou aspas duplas dentro de linhas com aspas duplas, através da separação das aspas com uma barra inclinada para trás, tal como mostra a *Figura 36*. A barra inclinada para trás

é um termo que representa um **caracter de saída** e as sequências \' e \" são **sequências de saída**.

```
<!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'
```

Figura 36 *– Caracter de saída da barra inclinada para trás (\) (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

As sequências de saída são valiosas para adicionar caracteres que não podem ser inseridos através do teclado. Algumas das outras sequências de saída mais utilizadas são:

- \n é substituído pelo caracter numa nova linha
- \t é substituído pela *tab* do caracter
- \r é substituído pelo caracter de *carriage-return*
- \b é substituído pelo caracter de *backspace*
- \\ é substituído pela barra única (\)

A *Figura 37* mostra como funcionam as sequências de saída:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Escape Sequences</title>
</head>
<body>
    <script>
    // Creating variables
    var str1 = "The quick brown fox \n jumps over the lazy dog.";
    document.write("<pre>" + str1 + "</pre>"); // Create line break

    var str2 = "C:\Users\Downloads";
    document.write(str2 + "<br>"); // Prints C:UsersDownloads

    var str3 = "C:\\Users\\Downloads";
    document.write(str3); // Prints C:\Users\Downloads
    </script>
</body>
</html>
```

```
The quick brown fox
 jumps over the lazy dog.

C:UsersDownloads
C:\Users\Downloads
```

Figura 37 *– Como funcionam as sequências de saída (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Realizar Operações em Linhas

O JavaScript faz com que várias propriedades e métodos possam fazer operações em valores de linha. Tendo isto em conta, vários objetos podem ter propriedades e métodos. No entanto, em JavaScript, tipos de dados primitivos podem comportar-se como objetos quando o programador se refere a eles com notação de acesso à propriedade. O JavaScript possibilita-o através da criação de um objeto *embalado* provisório para tipos de dados primitivos. Este procedimento pode ser feito automaticamente pelo intérprete JS no *background*.

Definir o Comprimento de uma Linha

A propriedade do comprimento define o comprimento de uma linha, que é o número de caracteres delimitados numa linha, incluindo o número de caracteres especiais, como /t ou /n. Os programadores devem ter especial atenção ao usarem parênteses depois do comprimento (ex.: *str.length*()), pois a forma correta é *str.length* (se não for esta, irá dar erro).

```
<!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

Figura 38 *– Como definir o comprimento de uma linha (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Descobrir uma Linha Dentro de Outra Linha

O método *indexOf*() pode ser usado para descobrir uma sublinha ou uma linha dentro de outra linha. Esta técnica devolve o *index* ou a posição do primeiro incidente de uma linha dentro de uma linha específica.

```
<!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

Figura 39 *– Descobrir uma linha dentro de uma linha (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Desta forma, a técnica *lastIndexOf*() pode ser usada para descobrir o index ou a posição da última ocorrência de uma linha dentro de uma linha específica, como explicado de seguida:

```
<!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); // 0utputs: 46
    </script>
</body>
</html>
```

46

Figura 40 *– Descobrir uma linha dentro de uma linha com o método lastIndexOf()(**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Tanto o *indexOf*() como lastIndexOf() devolvem o resultado de -1 se a sublinha não for encontrada. Ambos os métodos aceitam, também, um parâmetro inteiro opcional que estipula a posição dentro da linha que inicia a busca.

Procurar um Padrão Dentro de uma Linha

Este método de procura pode ser usado para encontrar uma peça de texto ou padrão particular dentro de uma linha. Tal como o *indexOf(),search()* também volta ao index da primeira combinação, e devolve -1 se nenhum par for encontrado, mas, ao contrário do *indexOf(),search()* pode também usar as expressões regulares como um argumento para entregar aptidões de busca avançadas. Deve ser notado que o método de *search()* não funciona com buscas globais, pois ignora a bandeira ou modificador *g* no argumento regular de expressão.

```
<!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>"); // 0utputs: 30

    // Case insensitive search using regexp
    var pos2 = str.search(/color/i);
    document.write(pos2); // 0utputs: 0
    </script>
</body>
</html>
```

```
30
0
```

Figura 41 *– Procurar um padrão dentro de uma linha (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Extrair uma Sublinha a uma Linha

Para extrair uma parte ou uma sublinha de uma linha, pode usar-se o método *slice*(). Isto requer dois parâmetros: *start index* (index onde começa a extração) e, opcionalmente, *index end* (index depois onde termina a extração), como *str.slice (startIndex, endIndex).*

O exemplo abaixo corta uma porção de uma linha da posição 4 para a posição 15:

```html
<!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
```

Figura 42 *– Extrair uma Sublinha de uma Linha usando o método slice()(**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Os valores negativos podem ser específicos também. Estes valores são tratados como *strLength* + *startIndex*, onde *strLength* é o comprimento da linha (como no caso de *str.length*), por exemplo, de o startIndex é -5, este é tratado como strLength – 5. Se o startIndex é maior ou igual do que o comprimento da linha, o método *slice*() retoma uma linha vazia. Paralelamente, se o *endIndex* opcional não for específicado ou for omitido, o método *slice*() extrai-se para o fim da linha.

```html
<!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.
```

Figura 43 *– Específicar valores negativos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

O método de sub*string*() para extrair a secção dada de uma linha baseada nos indexes de inicio e de fim, como str.sub*string*(startIndex, endIndex). O método de sub*string*() é bastante comparável ao do slice(), exceto algumas diferenças:

- Se algum dos argumentos for menor do que 0 ou for NaN, este é tratado como 0.
- Se algum dos argumentos for maior do que *str.length*, este é tratado como se fosse *str.length*.
- Se o *startIndex* for maior do que o *endIndex*, então o *substring*() vai trocar esses dois argumentos, por ex.: str.sub*string*(5, 0) == str.sub*string*(0, 5).

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Extract substring from a String</title>
</head>
<body>
    <script>
    var str = "The quick brown fox jumps over the lazy dog.";
    document.write(str.substring(4, 15) + "<br>"); // Prints: quick brown
    document.write(str.substring(9, 0) + "<br>"); // Prints: The quick
    document.write(str.substring(-28, -19) + "<br>"); // Prints nothing
    document.write(str.substring(31)); // Prints: the lazy dog.
    </script>
</body>
</html>
```

quick brown
The quick

the lazy dog.

***Figura 44** – O método substring()para extrair uma secção de uma linha baseado nos indexes de inicio e fim.*

*(**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Extraír um Número Fixo de Caracteres de uma Linha

O JavaScript também fornece a técnica substr(), que é parecida à do slice() com uma pequena diferença: o segundo parâmetro estipula o número de caracteres a extraír em vez do index fina, como str.substr(startIndex, length). Se o comprimento for 0 ou um número negativo, é devolvida uma linha vazia.

```
<!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.
```

Figura 45 *– Extrair um Número Fixo de Caracteres de uma Linha (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Substituir o Conteúdo de uma Linha

A técnica de *replace*() é usada para substituir uma parte de uma linha por outra linha. Esta abordagem usa uma expressão regular para combinar ou uma sublinha com dois parâmetros e uma linha de substituição, por ex.: *str.replace* (regexp|substr, newSubstr). Este método *replace()* devolve uma nova linha, que não interfere com a linha original que se manterá intacta.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Replace Part of a String with another String</title>
</head>
<body>
    <script>
    var str = "Color red looks brighter than color blue.";
    var result = str.replace("color", "paint");
    document.write(result); // 0utputs: Color red looks brighter than paint blue.
    </script>
</body>
</html>
```

```
Color red looks brighter than paint blue.
```

Figura 46 *– Substituir os Conteúdos de uma Linha usando a técnica replace() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Por norma, a técnica *replace*() substitui apenas a primeira combinação e é sensível a letras maiúsculas e minúsculas. Para substituir a sublinha dentro de uma linha numa

forma sensível a maiúsculas e minúsculas, a *regular expression (regexp)* com um modificador i pode ser usada tal como no exemplo seguinte:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Replace Part of a String with another String</title>
</head>
<body>
    <script>
    var str = "Color red looks brighter than color blue.";
    var result = str.replace(/color/i, "paint");
    document.write(result); // 0utputs: paint red looks brighter than color
blue.
    </script>
</body>
</html>
```

```
paint red looks brighter than color blue.
```

Figura 47 *– Substituir os Conteúdos de uma Linha usando a técnica replace() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Para substituir todas as incidências de uma sublinha dentro de uma linha de uma forma sensível a maiúsculas e minúsculas, podem ser usados o modificador *g* com o modificador *i*, tal como usado abaixo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Replace All Occurrences of a Substring in a
String</title>
</head>
<body>
    <script>
    var str = "Color red looks brighter than color blue.";
    var result = str.replace(/color/ig, "paint");
    document.write(result); // 0utputs: paint red looks brighter than paint
blue.
    </script>
</body>
</html>
```

```
paint red looks brighter than paint blue.
```

Figura 48 *– Substituir todos os incidentes numa sublinha dentro de uma linha de forma sensível a maiúsculas e minúsculas (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Transformar uma Linha para Maiúsculas ou Minúsculas

O método *toUpperCase*() é usado para converter uma linha para maiúsculas, tal como no exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Convert a String to Uppercase Characters</title>
</head>
<body>
    <script>
    var str = "Hello World!";
    var result = str.toUpperCase();
    document.write(result); // Prints: HELLO WORLD!
    </script>
</body>
</html>
```

HELLO WORLD!

Figura 49 *– Coverter uma Linha para Maiúsculas usando o método toUpperCase() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Desta forma, o método toLowerCase() é usado para converter uma linha em minúsculas:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Convert a String to Lowercase Characters</title>
</head>
<body>
    <script>
    var str = "Hello World!";
    var result = str.toLowerCase();
    document.write(result); // Prints: hello world!
    </script>
</body>
</html>
```

hello world!

Figura 50 *– Coverter uma Linha para Minusculas usando o método toLowerCase()(**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Concatenação de Duas ou Mais Linhas

Duas ou mais linhas podem ser concatenadas ou combinadas através do uso de operadores atribuídos + e +=.

```html
<!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

Figura 51 *– Concatenação de Duas ou Mais Linhas (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Aceder a Caracteres Individuais de uma Linha

O método charAt() pode ser usado para aceder a caracteres individuais de uma linha, como str.charAt(index). O index específico deve ser um número entre 0 e str.length - 1. Se não for atribuído qualquer index, o primeiro caracter da linha é devolvido, pois o valor base é 0.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Extract a Single Character from a String</title>
</head>
<body>
    <script>
    var str = "Hello World!";
    document.write(str.charAt() + "<br>");  // Prints: H
    document.write(str.charAt(6) + "<br>"); // Prints: W
    document.write(str.charAt(30) + "<br>"); // Prints nothing
    document.write(str.charAt(str.length - 1)); // Prints: !
    </script>
</body>
</html>
```

H
W
!

Figura 52 *– Aceder a Caracteres Individuais de uma Linha (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

No entanto, existem boas alternativas para este procedimento. Desde ECMAScript 5, as linhas podem ser tratadas como matrizes de leitura, e caracteres individuais podem

ser vistos de uma linha usando parênteses retos ([]) em vez da técnica de charAt(), como no exemplo em baixo:

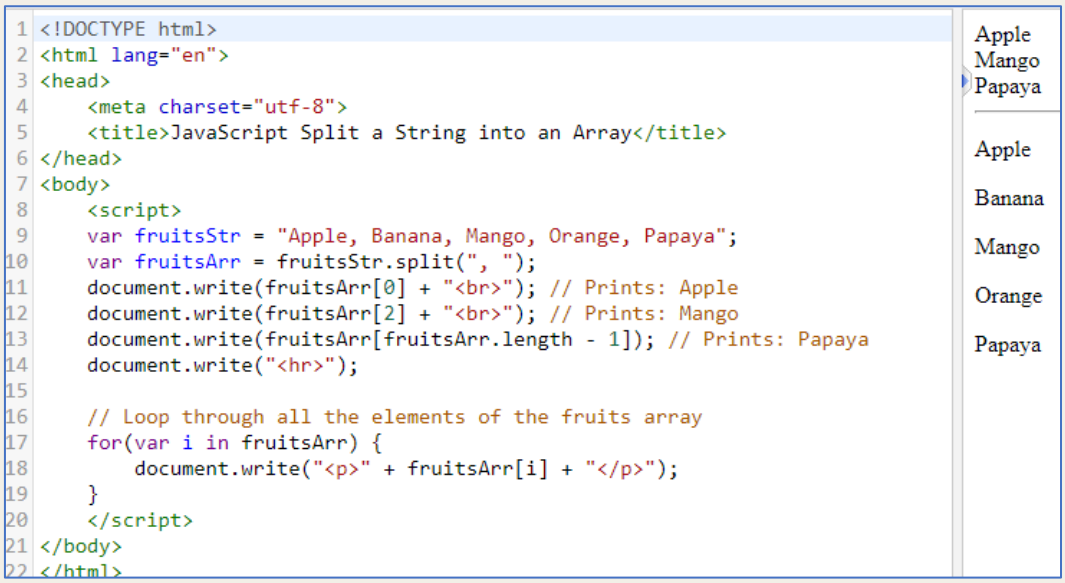
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Extract a Single Character from a String</title>
</head>
<body>
    <script>
    var str = "Hello World!";
    document.write(str[0] + "<br>"); // Prints: H
    document.write(str[6] + "<br>"); // Prints: W
    document.write(str[str.length - 1] + "<br>"); // Prints: !
    document.write(str[30]); // Prints: undefined
    </script>
</body>
</html>
```

```
H
W
!
undefined
```

***Figura 53** – Aceder a Caracteres Individuais de uma Linha usando parênteses retos ([]) em vez da técnica charAt() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Dividir uma Linha numa Matriz

O método de *split*() pode ser usado para fragmentar uma linha numa matriz de linhas, usando a síntaxe *str.split(separator, limit)*. O argumento de separador estipula qual a linha onde deve acontecer a divisão. Se o argumento de separador for omitido ou não for encontrado numa linha específica, a linha completa é alocada para o primeiro elemento da matriz.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Split a String into an Array</title>
</head>
<body>
    <script>
    var fruitsStr = "Apple, Banana, Mango, Orange, Papaya";
    var fruitsArr = fruitsStr.split(", ");
    document.write(fruitsArr[0] + "<br>"); // Prints: Apple
    document.write(fruitsArr[2] + "<br>"); // Prints: Mango
    document.write(fruitsArr[fruitsArr.length - 1]); // Prints: Papaya
    document.write("<hr>");

    // Loop through all the elements of the fruits array
    for(var i in fruitsArr) {
        document.write("<p>" + fruitsArr[i] + "</p>");
    }
    </script>
</body>
</html>
```

```
Apple
Mango
Papaya

Apple
Banana
Mango
Orange
Papaya
```

Figura 54 *– Dividir uma Linha numa Matriz (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-get-started.php)*

Para dividir uma linha em matrizes de caracteres, deve ser estipulada uma linha vazia ("") como separador, tal como demonstrado neste exemplo.

Números em JavaScript

Existem dois tipos de números em JavaScript:

- **Números Regulares** em JavaScript são armazenados num formato de 64-bit IEEE-754, também conhecido como "*double precision floating point numbers*". Estes são o tipo de números mais comum e mais usados.
- **Números BigInt**, representam valores inteiros aleatórios de comprimento. Estas são necessários, pois um número regural não pode exceder, com segurança, $\mathbf{2^{53}}$ ou menos do que $\mathbf{-2^{53}}$.

Existem mais formas de escrever um número. Por exemplo, a forma mais óbvia de escrever 1 milhão seria '1000000000' or '1_000_000_000', usando o *underscore* como separador. Neste caso, o *underscore* é um "syntactic sugar", que torna o número mais

fácil de ler. O mecanismo de JS ignora os *underscore*s entre os dígitos, sendo o mesmo 1 milhão do primeiro caso.

No entanto, na vida real, toda a gente irá evitar escrever longas sequências de zeros. Algo como “1bn” para um milhão ou “2.5bn” para dois milhões e 500 mil parece bastante mais razoável. Os mesmos princípios aplicam-se na maioria dos números grandes. Posto isto, é possível encurtar um número em JS ao adicionar a letra “e” ao mesmo que se especifica a quantidade de zeros:

```
1  let billion = 1e9;  // 1 billion, literally: 1 and 9 zeroes
2
3  alert( 7.3e9 );  // 7.3 billions (same as 7300000000 or 7_300_000_000)
```

Figura 55 *– Específicar a quantidade de zeros (**Fonte:** https://JavaScript.info/number)*

Portanto, o “e” multiplica o número por 1 com o número de zeros atribuídos.

```
1e3 === 1 * 1000; // e3 means *1000
1.23e6 === 1.23 * 1000000; // e6 means *1000000
```

Este mesmo princípio aplica-se também a números pequenos. Por exemplo, o que deve ser escrito para 1 microssegundo (um milhão de um segundo)?

```
let mcs = 0.000001;
```

Tal como o que foi mencionado antes com os números grandes, usar o "e" pode ser benéfico. Para evitar escrever os zeros explicitamente, pode ser escrito o seguinte:

```
let mcs = 1e-6; // six zeroes to the left from 1
```

Existem 6 zeros em 0.000001. Então, é fácil concluir que 1e-6.

Por consequente, um número negativo antes do “e” implica a divisão por 1 com o número de zeros atribuídos, como no exemplo:

```
1  // -3 divides by 1 with 3 zeroes
2  1e-3 === 1 / 1000; // 0.001
3
4  // -6 divides by 1 with 6 zeroes
5  1.23e-6 === 1.23 / 1000000; // 0.00000123
```

Figura 56 *– A divisão de 1 pelo número atribuído de zeros (**Fonte:** https://JavaScript.info/number)*

Números hex, binários e octal

Númeroes hexadecimais são comuns no JavaScript para a atribuição das cores, codificação de caracteres e muitos outros propósitos. Portanto, obviamente, existe uma forma mais rápida de escrevê-los: 0x e depois o número, tal como abaixo:

```
1  alert( 0xff ); // 255
2  alert( 0xFF ); // 255 (the same, case doesn't matter)
```

Figura 57 *– O 0x como forma de acelerar os processamentos em JS (**Fonte:** https://JavaScript.info/number)*

Sistemas numerais binários e octais não são utilizados com muita frequência, mas são também compatíveis através do uso dos prefixos de 0b e 0o (zero, o):

```
1  let a = 0b11111111; // binary form of 255
2  let b = 0o377; // octal form of 255
3
4  alert( a == b ); // true, the same number 255 at both sides
```

Figura 58 *– Os prefixos 0b e 0o para sistemas numerias binários e octais (**Fonte:** https://JavaScript.info/number)*

Existem apenas 3 sistemas numerais com este suporte. Para mais sistemas numerais deve ser usado a função *parseInt*.

Método to*String*(base)

O método num.to*String*(base) devolve uma linha de representação de um sistema numérico com as bases providenciadas, como no exemplo:

```
1  let num = 255;
2
3  alert( num.toString(16) );  // ff
4  alert( num.toString(2) );   // 11111111
```

Figura 59 *– O método num.toString(base)* (***Fonte:*** *https://JavaScript.info/number)*

A base pode variar de 2 a 36. Por definição é 10.

Existem casos de uso geral para isto:

- **base=16** é usado para as cores hex, descodificação de caracteres, etc., os dígitos podem ser 0..9 ou A..F.
- **base=2** é maioritariamente usado para a correção de operações *bitwise*, os dígitos podem ser 0 ou 1.
- **base=36** é o máximo, os dígitos podem ser 0..9 ou A..Z. Todo o alfabeto latino pode ser usado para representar um número. Um caso curioso, mas útil, para o 36 é quando é necessário tornar um identificador longo numérico em algo mais curto, por exemplo, para fazer um URL curto.

Existe um caso específico que deve ser considerado. Quando dois pontos são localizados em 123456..to*String*(36), não se trata de uma gralha. Quando o programador quer aplicar um método diretamente a um número, é preciso colocar os dois pontos ('..') de seguida.

Se apenas for colocado um único ponto ('123456.to*String*(36)'), isto seria um erro, pois a síntaxe de JavaScript determina a parte decimal após o primeiro ponto. E, se o programador colocar mais um ponto, então o JavaScript sabe que a parte decimal está vazia e procede para o método.

Arredondamento

Uma das operações mais aplicadas aquando da operação de números é o **arredondamento**.

Aqui estão algumas funções built-in para arredondamento:

- **Math.floor**

Arredonda para baixo: 3.1 fica 3, e -1.1 altera para -2.

- **Math.ceil**

Arredonda para cima: 3.1 torna-se 4, e -1.1 fica -1.

- **Math.round**

Arredonda para o número inteiro mais próximo: 3.1 fica 3, 3.6 fica 4, o caso do meio: 3.5 arredonda para 4 também.

- **Math.trunc (não suportado no Internet Explorer)**

Elimina tudo após o ponto decimal sem arredondar: 3.1 fica 3, -1.1 arredonda para -1.

Estas funções cobrem todas as formas possíveis para lidar com a parte decimal de um número.
Mas então como é que se arredonda um número ao dígito n-th após o decimal?
Por exemplo, para arredondar 1.2345 em dois dígitos (1.23).

Estas são as duas maneiras de o fazer:

1. **Multiplicar e dividir.**

Por ex., para arredondar o número ao 2º digíto depois do decimal, podemos multiplicar o número por 100 (ou uma potência maior de 10), aplicar a função de arredondamento e depois dividi-lo novamente.

2. O método **toFixed(n)** arredonda o número para n dígitos depois do ponto e retoma a representação do resultado numa linha.

```
1  let num = 12.34;
2  alert( num.toFixed(1) ); // "12.3"
```

Figura 60 *– O método toFixed(n) (**Fonte:** https://JavaScript.info/number)*

Isto arredonda para cima ou para baixo do valor mais perto, tal como o *Math.round*:

```
let num = 12.36;
alert( num.toFixed(1) ); // "12.4"
```

***Figura 61** – O método toFixed(n) – continuação (**Fonte:** https://JavaScript.info/number)*

Deve ser sublinhado que o resultado do *toFixed* é uma linha. Se a parte decimal for mais curta do que o pedido, serão atribuídos zeros no final:

```
let num = 12.34;
alert( num.toFixed(5) ); // "12.34000", added zeroes to make exactly 5 digits
```

***Figura 62** – O método toFixed(n) – exposição final (**Fonte:** https://JavaScript.info/number)*

Cálculos Imprecisos

No interior, um número é representado num formato 64-bit IEEE-754, o que quer dizer que existem precisamente 64 bits para armazenar um número; 52 para armazenar os dígitos; 11 para armazenar a posição do ponto decimal (zero para os números inteiros), e 1 bit para o sinal.

Se o número for muito longo, irá ultrapassar o armazenamento de 64-bits, dando, possivelmente, infinito.

Algo que acaba por acontecer com alguma frequência é a perda de precisão.

Considere o seguinte teste:

```
alert( 0.1 + 0.2 == 0.3 ); // false
```

***Figura 63** – Primeiro teste de cálculos imprecisos (**Fonte:** https://JavaScript.info/number)*

Tal como parece, o resultado de 0.1+0.2 é, estranhamente, um 0.3 falso. O que poderá ser então?

```
alert( 0.1 + 0.2 ); // 0.30000000000000004
```

***Figura 64** – Segundo teste de cálculos imprecisos (**Fonte:** https://JavaScript.info/number)*

Tal como pode ser imaginado, existem mais consequências do que uma comparação incorreta neste caso. Imaginemos que uma compra está a ser feita numa loja *online* e

o visitante põe artigos de 0.10$ e 0.20$ no seu carrinho. O total desta compra seria $0.30000000000000004. Isso seria uma surpresa.

Porque é que esta situação acontece?

Um número é armazenado na memória num formato binário, uma sequência de bits – uns e zeros. No entanto, frações como 0.1 e 0.2 que paracem simples num sistema decimal numérico são, na verdade, frações intermináveis num formato binário.

Isto para dizer, o que é 0.1? É um dividido por 10 1/10, um décimo. Em sistemas numéricos decimais, estes números são facilmente representados. Comparando-o com um terço: 1/3. Torna-se uma fração infinita 0.33333(3).

Então, a divisão por potências de 10 funciona garantidamente bem no sistema decimal, mas a divisão por 3 não é possível. Por esta mesma razão, no sistema numeral binário, a divisão por potências de 2 funciona garantidamente, mas 1/10 torna-se numa fração binária infinita.

Não existe forma de armazenar exatamente 0.1 ou 0.2 usando o sistema binário, assim como não há forma de armazenar um terço como fração decimal.

O formato numérico IEEE-754 resolve isto ao arredondar para o número mais perto possível. Estas regras de arredondamento, por norma, não permitem ver as “pequenas perdas de precisão”, mas estas existem.

```
1  alert( 0.1.toFixed(20) ); // 0.10000000000000000555
```

Figura 65 *– Resultado final dos cálculos imprecisos (**Fonte:** https://JavaScript.info/number)*

Quando dois números são somados, as suas “perdas de precisão” juntam-se. Por isso, 0.1+0.2 não é precisamente 0.3.

O método mais fiável para lidar com esta situação é o **toFixed(n)**:

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

Figura 66 *– O método toFixed(in) (**Fonte:** https://JavaScript.info/number)*

Deve ser notado que o toFixed(n) devolve uma linha todas as vezes. Garante que tem 2 digitos depois do ponto decimal. Isto é útil quando existe a necessidade de mostrar 0.30$ no contexto de compras *online*.

Testes: isFinite e isNaN

Anteriormente, estes dois valores numéricos especiais foram descritos. **Infinity** (e **-Infinity**) são um valor número especial que é maior (menor) que alguma coisa. **NaN** representa um erro. Ambos pertecem ao tipo **número**, mas não são números “normais”, então existem funções especiais para estes:

- **isNaN(value)** converte o argumento para um número e depois testa se é **NaN**:

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

Figura 67 *– O método isNaN(value) (**Fonte:** https://JavaScript.info/number)*

- **isFinite(value)** converte o seu argumento num número e fará um retorno como sendo **verdadeiro** se se tratar de um número regular, e não **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
```

Figura 68 *– O método isFinite(value)(**Fonte:** https://JavaScript.info/number)*

parseInt e parse*Float*

A conversão numérica usando um mais (+) ou **Number()** é rigorosa. Se o valor não for exatamente um número, falha. A única excepção são espaços no início ou no fim da linha, pois estes são ignorados.

No entanto, na vida real, existem outros valores em unidades, como "100px" ou "12pt" em CSS. Desta forma, em muitos países, o símbolo da moeda fica depois da quantidade, por exemplo “19€". Quantas vezes o programador desejaria extrair um valor numérico exato deste número? É para este fim que se destinam o **parseInt** e o **parse*Float***.

Eles *“lêem”* um número de uma linha até não poderem. Em caso de erro, o número recolhido é devolvido. A função **parseInt** retoma um número inteiro, onde o **parse*Float*** devolve um número em ponto-flutuante:

```
1 alert( parseInt('100px') ); // 100
2 alert( parseFloat('12.5em') ); // 12.5
3
4 alert( parseInt('12.3') ); // 12, only the integer part is returned
5 alert( parseFloat('12.3.4') ); // 12.3, the second point stops the reading
```

Figura 69 *– parseInt e parseFloat (**Fonte:** https://JavaScript.info/number)*

Outras funções matemáticas

O JavaScript é detentor de um *built-in* objeto *Math*, que inclui uma pequena biblioteca de funções e constantes matemáticas:

- **Math.random()**

  Devolve um número aleatório do 0 ao 1 (não incluindo o 1).

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

  Devolve o maior/menor valor do número arbitrário de argumentos.

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

Devolve n elevado à potência dada.

Expressões JavaScript If…Else

Tal como outras linguagens de programação, o JavaScript também permite escrever código que execute ações diferentes baseadas nas condições de testes lógicos ou comparativos na altura em que estes foram executados. Assim, as condições de teste podem ser criadas como expressões que avaliam como verdadeiro ou falso e, baseado nestes resultados, certas ações podem ter efeito.

Existem várias expressões de condições em JavaScript que podem ser usadas para tomar decisões:

- A expressão **if** ;
- A expressão **if...else**;
- A expressão **if...else if....else**;
- A expressão **switch...case**.

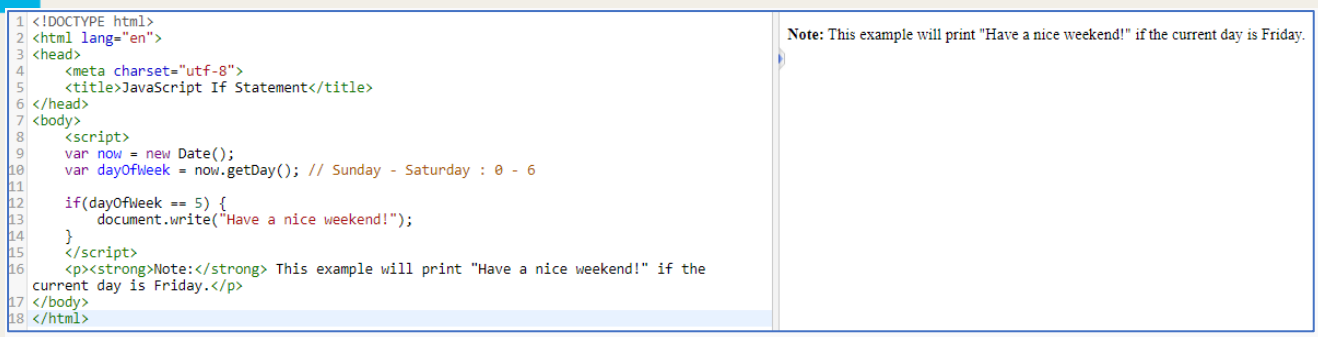
A expressão "if"

A expressão "*if*" é usada para executar um bloco de código apenas se a condição específica de avaliação for verdadeira. Esta é a expressão condicional mais simples de JS e pode ser escrita como:

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

Figura 70 *– Código para criar uma condição (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-if-else-statements.php)*

Um exemplo prático pode ser uma demonstração da utilidade desta característica. Se hoje for sexta-feira, o códiogo mostrado na *Figura 68* irá mostrar uma mensagem a dizer "Tenha um bom fim-de-semana":



```
<!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>
```


***Figura 71** – Expressão if… criada para mostrar a mensagem "Tem um bom fim-de-semana" se for sexta-feira (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-if-else-statements.php)*

A expressão "if…else"

O processo de tomar decisões em JS pode ser melhorado ao adicionar uma escolha alternativa através de **expressões else** à **expressão**. A expressão if…else permite executar um bloco de código se as condições específicas avaliadas como verdadeiras e outro bloco de código se avaliadas como *falsas*.

```
if(condition) {
    // Code to be executed if condition is true
} else {
    // Code to be executed if condition is false
}
```

***Figura 72** – Código para criar uma condição if…else (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-if-else-statements.php)*

O exemplo da *Figura 68* será aplicada ao teste da expressão if…else. Desta vez, a mensagem "Tenha um bom fim-de-semana!" será mostrada se o dia de hoje for sexta-feira (tal como no caso anterior), no entanto, irá emitir a mensagem "Tenha um bom dia" se não for sexta-feira.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript If-Else Statement</title>
</head>
<body>
    <script>
    var now = new Date();
    var dayOfWeek = now.getDay(); // Sunday - Saturday : 0 - 6

    if(dayOfWeek == 5) {
        document.write("Have a nice weekend!");
    } else {
        document.write("Have a nice day!");
    }
    </script>
</body>
</html>
```

Have a nice day!

***Figura 73** – Código para criar uma condição if…else, mostrando uma mensagem de "Tenha um bom fim-de-semana" ou "Tenha um bom dia" de acordo com o caso (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-if-else-statements.php)*

O operador ternário

O operador ternário dá-nos uma **forma abreviada de escrever expressãos "if…else"**. É caracterizado pelo símbolo de ponto de interrogação (?) e requer três operações: uma condição para analisar, um resultado para constituir uma "verdade" e um resultado para "falso". A sua síntaxe básica segue no seguinte exemplo:

```
var result = (condition) ? value1 : value2
```

Se a condição for avaliada como verdadeira, o value1 será devolvido, se não será devolvido o value2.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Typical Conditional Statement</title>
</head>
<body>
    <script>
    var userType;
    var age = 21;
    if(age < 18) {
        userType = 'Child';
    } else {
        userType = 'Adult';
    }
    document.write(userType); // Prints Adult
    </script>
</body>
</html>
```

Adult

Figura 74 *– Como usar um operador Ternary (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-if-else-statements.php)*

Expressões “Switch…Case”

A expressão “**switch..case”** é um cenário alternativo para as expressãos “**if...else if...else”**, que são quase o mesmo. As expressões “**switch...case”** analisam a variável ou a expressão perante uma série de valores até que encontre uma combinação, e depois executa um bloco de código correspondente para combinar. A sua síntaxe apresenta-se da seguinte forma:

```
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
}
```

Figura 75 – *Síntaxe para expressãos switch…case (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-if-else-statements.php)*

Este exemplo mostra o nome do dia da semana em que o utilizador está.

A expressão "**switch...case"** diverge da expressão "**if...else"** de uma forma expressiva. A expressão "**switch"** executa linha por linha e, quando o JavaScript descobre uma cláusula de caso que avalie se é verdadeiro, efetua, não apenas o código correspondente a essa cláusula de caso, como também executa todas as cláusulas de causa automaticamente até ao fim do bloco "**switch**".

De forma a prever isto, deve ser incluída uma expressão de quebra depois de todos os casos (como se verifica na Figura 73). A expressão de quebra informa o intérprete JS que tem de sair do bloco da expressão de **switch...case** uma vez que execute o código associado ao primeiro caso verdadeiro.

A expressão de quebra ainda não é requerida para o caso ou cláusulas pré-definidas quando aparece no final de uma expressão *switch*. No entanto, é uma boa prática de programação dispensar o último caso ou cláusula pré-definida numa expressão *switch* com uma quebra. Evita um possível erro de programação que surja mais tarde caso outra expressão esteja incluída na expressão *switch*.

A cláusula pré-definida é voluntária, o que estipula as ações a acontecerem se nenhum caso combinar com a expressão *switch*. As cláusulas pré-definidas não têm que ser a última cláusula a ser encontrada numa expressão *switch*.

Cada valor de caso deve ser exclusivo dentro de uma expressão switch. Ainda assim, casos diferentes não precisam de ter uma ação única. Vários casos podem partilhar a mesma ação.

Matrizes JavaScript

As matrizes são variáveis complexas que permitem armazenar mais do que um valor ou um grupo dentro de um único nome de variável. As matrizes JavaScript podem ser armazenar qualquer valor válido, incluindo linhas, números, objetos, funções, e, ao

mesmo tempo, outras matrizes, por consequência permitindo a criação de dados estruturados mais complexos como uma matriz de objetos ou uma matriz de matrizes.

No seguinte exemplo, o nome das cores pode ser armazenado num código JavaScript:

```
<!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
```

Figura 76 – *Guardar o nome de cores num código JS (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

No entanto, pode ser um pouco difícil e cansativo armazenar elementos em variáveis, ao contrário os três casos em cima. Para além disso, usar muitas variáveis ao mesmo tempo e acompanhá-los pode ser uma tarefa bastante complicada. E aqui existem matrizes que entram em cena. A matriz resolve este problema ao providenciarem uma estrutura ordenada para o armazenamento de múltiplos valores de armazenamento ou de vários grupos de valores.

Criar uma Matriz

A forma mais simples de criar uma matriz em JavaScript é de envolver uma lista de valores em parênteses retos ([]) separados por vírgulas, tal como está na seguinte síntaxe:

```
var myArray = [element0, element1, ..., elementN];
```

```html
<!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
```

Figura 77 –*Criar uma matriz(**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

Aceder aos elementos de uma Matriz

Os elementos de uma matriz podem ser acedidos pelo index, utilizando uma anotação dos parênteses retos. Um index é um elemento que representa a posição de um elemento numa matriz.

As matrizes são baseadas no 0. Isto significa que o primeiro iten de uma matriz é armazenada no index 0 e não no último, pela segunda vez que os itens são armazenados no index 2, e etc. No entanto, uma matriz de cinco elementos terá indexes do 0 ao 4.

```html
<!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
```

Figura 78 –*Como obter um elemento individual de uma matriz pelo index (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

Descobrir o Comprimento de uma Matriz

A propriedade de comprimento devolve o comprimento de uma matriz, que é o número total de elementos incluindo a matriz. O comprimento de uma matriz é sempre maior que o index de todos os elementos nela.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Get the Length of an Array</title>
</head>
<body>
    <script>
    var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
    document.write(fruits.length); // 0utputs: 5
    </script>
</body>
</html>
```

```
5
```

Figura 79 – *Descobrir o comprimento de uma matriz (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

Içamento através dos elementos de uma Matriz

O içamento pode ser usado para ganhar acesso a cada elemento de uma matriz numa ordem sequencial, como no exemplo que se segue:

```
<!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
```

Figura 80 – *Içamento através dos elementos de uma matriz (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

ECMAScript 6 veio introduzir uma forma mais simples de interar por cima de elementos de matrizes, o que é um içamento for-of. Neste içamento aqui, não existe necessidade para inicializar a variável de con*tag*em de variáveis (i).

Adicionar Novos Elementos a uma Matriz

Para adicionar um novo elemento no final de uma matriz, use, simplesmente o método *push*(), tal como demonstrado de seguida:

```
<!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
```

Figura 81 – *Adicionar Novos Elementos a uma Matriz (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

Desta forma, para adicionar um novo elemento no início de uma matriz devemos usar o método *unshift*() tal como explicado no seguinte exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Add a New Element at the Beginning of an Array</title>
</head>
<body>
    <script>
    var colors = ["Red", "Green", "Blue"];
    colors.unshift("Yellow");

    document.write(colors + "<br>"); // Prints: Yellow,Red,Green,Blue
    document.write(colors.length); // Prints: 4
    </script>
</body>
</html>
```

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

Figura 82 – *Adicionar um novo elemento no inicio de uma matriz utilizando o método unshift() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

Remover Elementos de uma Matriz

Para eliminar o último elemento de uma matriz podemos usar o método *pop*(). Este método devolve o valor que foi retirado.

```
<!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
```

Figura 83 – *Remover Elementos de uma Matriz (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

Adicionar ou Remover Elementos em Qualquer Posição

O método *splice*() é um método de matriz bastante flexível que permite adicionar ou remover elementos de qualquer síntaxe, usando `arr.splice(startIndex, deleteCount, elem1, ..., elemN).`

Este método requer três parâmetros: o primeiro é o index onde será para iniciar a divisão da matriz, é obrigatório; o segundo é o número de elementos a remover (zero deve ser usado caso o programador não queira retirar nenhum elemento), é opcional; e o terceiro parâmetro é um conjunto de elementos de substituição, também estes opcionais.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Add or Remove Array Elements at any Index</title>
</head>
<body>
    <script>
    var colors = ["Red", "Green", "Blue"];
    var removed = colors.splice(0,1); // Remove the first element

    document.write(colors + "<br>"); // Prints: Green,Blue
    document.write(removed + "<br>"); // Prints: Red (one item array)
    document.write(removed.length + "<br>"); // Prints: 1

    removed = colors.splice(1, 0, "Pink", "Yellow"); // Insert two items at position one
    document.write(colors + "<br>"); // Prints: Green,Pink,Yellow,Blue
    document.write(removed + "<br>"); // Empty array
    document.write(removed.length + "<br>"); // Prints: 0

    removed = colors.splice(1, 1, "Purple", "Voilet"); // Insert two values, remove one
    document.write(colors + "<br>"); //Prints: Green,Purple,Voilet,Yellow,Blue
    document.write(removed + "<br>"); // Prints: Pink (one item array)
    document.write(removed.length); // Prints: 1
    </script>
</body>
</html>
```

```
Green,Blue
Red
1
Green,Pink,Yellow,Blue

0
Green,Purple,Voilet,Yellow,Blue
Pink
1
```

Figura 84 – *Adicionar ou Remover Elementos em Qualquer Posição (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

O método *splice* () devolve uma matriz aos elementos apagados, ou uma matriz vazia se não tiverem sido eliminados nenhuns elementos, tal como pode ser visto na *Figura 81*. Se o segundo argumento está em falta, todos os elementos desde o início até ao fim da matriz são removidos. Ao contrário dos métodos de *slice ()* e *concat ()*, o método *splice()* modifica a forma como a matriz é chamada.

Criar uma Linha Através de uma Matriz

Podem existir situações onde o programador pretende, simplesmente, criar uma linha ao juntar os elementos de uma matriz. Para isto, este pode usar o método *join ()*. Este método retira um parâmetro opcional que é uma linha separadora que é adicionada entre cada elemento. Se omitirmos o separador, o JavaScript usará, por definição, uma vírgula (,).

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Join All Elements of an Array into a String</title>
</head>
<body>
    <script>
    var colors = ["Red", "Green", "Blue"];

    document.write(colors.join() + "<br>"); // Prints: Red,Green,Blue
    document.write(colors.join("") + "<br>"); // Prints: RedGreenBlue
    document.write(colors.join("-") + "<br>"); // Prints: Red-Green-Blue
    document.write(colors.join(", ")); // Prints: Red, Green, Blue
    </script>
</body>
</html>
```

```
Red,Green,Blue
RedGreenBlue
Red-Green-Blue
Red, Green, Blue
```

Figura 85 – *Criar uma Linha Através de uma Matriz (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

Uma matriz pode também ser convertida em linhas separadas por vírgulas ao usar to*String*(). Este método não permite o parâmetro de separação como *join()*.

Juntar Duas ou Mais Matrizes

O método *concat()* pode ser usado para combinar duas ou mais matrizes. Este método não altera as matrizes dominantes. Em vez disso, devolve uma nova matriz.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JavaScript Merge Two Arrays</title>
</head>
<body>
    <script>
    var pets = ["Cat", "Dog", "Parrot"];
    var wilds = ["Tiger", "Wolf", "Zebra"];

    // Creating new array by combining pets and wilds arrays
    var animals = pets.concat(wilds);
    document.write(animals); // Prints: Cat,Dog,Parrot,Tiger,Wolf,Zebra
    </script>
</body>
</html>
```

```
Cat,Dog,Parrot,Tiger,Wolf,Zebra
```

Figura 86 – *Juntar Duas ou Mais Matrizes (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

O método *concat()* pode levar qualquer número de argumentos de matrizes de forma a que uma matriz possa ser criada apartir de qualquer outro número de matrizes, tal como demonstrado no seguinte exemplo:

```
<!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
```

Figura 87 – *O método concat()(**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

Procurar Através de uma Matriz

Os métodos de *indexOf()* e *lastIndexOf()* podem ser usados para procurar, numa matriz, um valor específico. Se o valor for encontrado, ambos os métodos devolvem um index que representa o elemento da matriz. Se o valor não for encontrado, -1 será devolvido. O método indexOf() devolve o primeiro encontrado, enquanto que o lastIndexOf() devolve o último. Ambos os métodos reconhecem, também, um parâmetro opcional inteiro do index, que específica o index dentro da matriz na qual começou a busca.

```
<!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
```

Figura 88 – *Procurar Através de uma Matriz (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

O método includes() pode também ser usado para descobrir se uma matriz envolve um certo elemento ou não. Este método usa os mesmos parâmetros que os métodos indexOf() e lastIndexOf(), mas retorna resultados verdadeiros ou falsos em vez de um número index.

```
<!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
```

Figura 89 – *Procurar através de uma matriz usando o método includes() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

Para pesquisar uma matriz baseada em certas condições, podemos usar o método *find()* do JavaScript, que foi recentemente introduzido no ES6. Este método devolve o valor do primeiro elemento da matriz que preenche a função de teste dada. Se não, retoma sem identificação.

O método *find()* procura simplesmente o primeiro elemento que preencha as funções teste. Mesmo assim, se o programador pretender encontrar todos os elementos

combinados, pode usar o método *filter()*. Cria uma nova matriz com os elementos que passam, com sucesso, o teste dado, tal como se pode verificar neste exemplo.

Ordenação de Matrizes no JavaScript

A ordenação é uma tarefa popular quando se trabalha com matrizes. Pode ser usada, por exemplo, para mostrar os nomes de cidades ou países por ordem alfabética. O objeto de matrizes do JavaScript tem um método *built-in* – *sort()* – para ordenar elementos de matrizes por ordem alfabética.

```
<!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
```

Figura 90 – *O método built-in sort() para ordenar elementos de matrizes por ordem alfabética (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-arrays.php)*

Para **inverter a ordem dos elementos** de uma matriz, deve ser usado o método *reverse*(). Inverte uma matriz de uma forma a que o primeiro elemento da matriz passa para último, e vice-versa. Os métodos *sort*() e *reverse*() alteram a matriz inicial e devolvem uma referência para a mesma matriz, tal como pode ser visto aqui.

Para **ordenar matrizes numéricas**, não é aconselhável usar o método *sort*(), pois pode produzir resultados inesperados. Consequentemente, os programadores devem passar uma função comparativa, como quando uma função de comparação é específica, os elementos de uma matriz são ordenados de acordo com o valor de retomo da função de comparação.

Por exemplo, quando comparamos A com B:

- Se a função de comparação retomar um valor menor que 0, então o A vem primeiro.

- Se a função de comparação retomar um valor maior que 0, então o B vem primeiro.
- Se a função de comparação retomar 0, A e B mantêm-se inalterados em relação a si próprios, mas agrupados com respeito a todos os outros elementos.

Posto isto, como 5-20=-15, o que é menor que 0, ou seja, o 5 vem primeiro. Quando 20-10=10, que é maior que 0, o 10 vem antes do 20, onde 20-75=-55, o que é menor que 0, ou seja, o 20 vem antes do 75, em semelhança com o 50 que vem antes do 75, e sempre assim. Isto pode ser verificado neste exemplo.

De forma a **descobrir o máximo e o mínimo valor numa matriz**, deve ser usado o método apply(), em combinação com o método Math.max() e Math.min(), tal como explicado aqui. O método *apply*() oferece uma forma acessível de transformar matrizes de valores como argumentos a funções que aceitam múltiplos argumentos de uma maneira semelhante a matrizes, mas não são matrizes. Depois, o resultado proveniente da expressão Math.max.apply(null, numbers) no exemplo acima é equivalente ao Math.max (3, -7, 10, 8, 15, 2).

Por fim, para **ordenar uma matriz de objetos**, podemos usar o método *sort*(). Neste exemplo, é mostrado como ordenar uma matriz de objetos pelos valores de propriedade.

Laços (*loops*) em JavaScript

Os laços são aplicados para repetirem o mesmo bloco de código um número de vezes, caso uma certa condição seja encontrada. A ideia principal por detrás do içamento é mecanizar tarefas repetitivas dentro de um programa de forma a poupar tempo e esforço. O JavaScript suporta cinco diferentes tipos de laços:

- **while** — laços através de um bloco de código se a condição específica avaliada for verdadeira.

- **do…while** — laços através de um bloco de código uma vez; depois a condição é avaliada. Se a condição for verdadeira, a expressão é repetida se a condição específica for verdadeira.
- **for** — laços através de um bloco de código até que o contador atinja um número específico.
- **for…in** — laços através das propriedades de um objeto.
- **for…of** — laços sobre objetos iteráveis como matrizes, linhas, etc.

O içamento *while*

Este é a expressão de içamento mais fácil que é dada pelo JS. Faz içamento sobre um bloco de código se uma condição específica de avaliação se verificar verdadeira. Uma vez que esta condição não se verifique, o içamento é interrompido.
A síntaxe geral do içamento while é:

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

Neste exemplo, um içamento continua a correr enquanto as variáveis $i \leq 5$. Tal como se verifica, irá aumentar por 1 cada vez que o içamento correr. Os programadores devem ter a certeza de que a condição especificada no içamento possa, a certa altura, tornar-se negativa. Se não, o içamento nunca vai parar de correr (içamento infinito).

O içamento *do…while*

O içamento *do-while* é uma variante do içamento *while*, que acessa a condição no fim de cada içamento gerado. Com o içamento do…while, o bloco de código é executado uma vez, e depois a condição é avaliada. Se a condição for verdadeira, a ação é repetida se a condição específica de avaliação for verdadeira. A síntaxe comum é:

```
do {
   // Code to be executed
}
while(condition);
```

O código JavaScript neste exemplo define um içamento que começa com *i=1.* Irá depois produzir o resultado e aumentar o valor da variante de i por 1. Depois, esta condição será avaliada e o içamento continuará a correr se *i ≤ 5.*

O içamento for

Repete um bloco de código se uma certa condição se verificar. É normalmente usada para implementar um bloco de código para um certo número de vezes. A síntaxe é:

```
for(initialization; condition; increment) {
   // Code to be executed
}
```

Os parâmetros para as expressãos do **içamento for** têm as seguintes implicações:

- **initialisation** — é usado para estabelecer uma variável de contador e avaliado na sua totalidade antes da execução do corpo do içamento.
- **condition** — é avaliado no início de cada iteração. Se a sua avaliação for verdadeira, a expressão do içamento é arquivada. Se a avaliação for falsa, a execução do içamento acaba.
- **increment** — atualiza o contador do içamento como um novo valor de cada vez que este corre.

Este exemplo define um içamento que começa com i=1. O içamento irá correr até atingir o valor de *i ≤ 5*. A variável irá aumentar 1 cada vez que o içamento correr.

O içamento for...in

É um tipo especial de içamento que se repete nas propriedades de um objeto ou os elementos de uma matriz. A sua síntaxe é:

```
for(variable in object) {
   // Code to be executed
}
```

O contador de laços p.e.: a variável in para o içamento for-in é uma linha e não um número. Engloba o nome da propriedade atual ou o index dos elementos da matriz em uso.

Este exemplo demonstra como é que o içamento percorre todas as propriedades de um objeto JS.

O içamento *for...of*

O ES6 introduz um novo içamento for-of que permite a iteração sobre matrizes ou outros objetos iteráveis (p.e.: linhas) de uma forma simplificada. Para além disso, o código dentro do içamento é executado para cada elemento de um objeto iterável. Este exemplo mostra como é que um içamento percorre linhas e matrizes.

Funções JavaScript

Uma função é um conjunto de expressãos que realizam tarefas específicas e podem ser guardadas e mantidas de forma independente. As funções permitem criar conjuntos de código reutilizável que são mais fáceis de gerir e mais simples de limpar. Estas são algumas das van*tag*ens que se podem obter com o uso de funções:

- **Funções que diminuem a repetição de código dentro de um programa** — As funções permitem extraír blocos de código que são usados frequentemente para componentes únicos. Desta forma, é possível realizar a mesma tarefa ao

invocar esta função o que for necessário para este *script* sem ter que copiar e colar o mesmo bloco de código vezes sem conta.

- **Funções fazem como que o código seja mais fácil de manter** — Tendo em conta que as funções, uma vez criadas, podem ser aplicadas várias vezes, quaisquer alterações feitas dentro da função são automaticamente aplicadas em todos os locais, sem que os respetivos documentos fiquem afetados.
- **Funções facilitam a eliminação dos erros**— Quando o programa é dividido em funções, o programador sabe exatamente que função é que está a gerar erro e como encontrá-la. Por consequência, corrigir erros torna-se mais simples.

Definir e Nomear Funções

A declaração de uma função começa com a função no teclado, seguindo-se pela determinação do nome da função a ser criada, e depois seguem-se os parênteses e, por fim, o código da função entre chavetas {}. Para declarar uma função devemos aplicar a seguinte síntaxe:

```
function functionName() {
    // Code to be executed
}
```

Neste simples exemplo exibe-se a mensagem “Hello”.

Uma vez que a função seja definifa, pode ser chamada por qualquer local no documento, ao escrever o nome seguido por um par de parênteses, como, por exemplo, sayHello(), usado no exemplo anterior.

Adicionar Parâmetros às Funções

Os parâmetros podem ser específicados aquando da definição de uma função para aceitar valores de inputs no tempo da execução da mesma. Os parâmetros agem como

variáveis de placeholders dentro de uma função; estas são substituídas, durante a execução destas, por valores (conhecidos por *argumentos*) oferecidos à função no momento da requisição destes.

Os parâmetros são definidos na primeira linha de uma função dentro de um par de parênteses, tal como se segue:

```
function functionName(parameter1, parameter2, parameter3) {
    // Code to be executed
}
```

A função displaySum()neste exemplo recebe dois números como argumentos, adiciona-os como um e depois mostra o resultado no navegador. Simples. Não existe um limite na definição de parâmetros. Todavia, para cada parâmetro especificado, é necessário que um argumento passe para a função quando este for chamado, se isto não acontecer, o valor torna-se indefinido.

Valores Pré-Definidos para Parâmetros de Funções

Com o ES6, é possível especificar valores pré-definidos para os parâmetros das funções. Isto implica que, se nenhum argumento for dado à função quando esta for chamada, estes valores pré-definidos serão usados. Este exemplo é bastante claro sobre o quão importante esta funcionalidade é, pois, de forma a conseguir o mesmo resultado, teríamos, anteriormente, que usar este.

Devolver Valores de uma Função

Uma função pode devolver um valor de novo para o *script* que chamou a função, como um resultado, ao utilizar a expressão de retorno. Este valor pode ser de qualquer tipo (p.e.: matrizes e objetos). A expressão de retomo é, por norma, colocada na última linha da função antes de fechar com chavetas e termina com um ponto e vírgula (;), tal como mostrado aqui.

Trabalhar com Expressões de Funções

A síntaxe usada, anteriormente, para criar funções era chamada de **declaração de funções**. Existe outra síntaxe para construir uma função - expressão de funções. Uma vez que esteja armazenada num variável, esta variável pode ser usada como função. A síntaxe de uma declaração de função e de uma expressão de função parecem bastante semelhantes, mas estas variam na forma em que são avaliadas. Tal como observado no exemplo anterior, a expressão de funções permitiu uma excepção quando foi chamada antes de ser definida, mas a declaração da função foi executada com eficiência. O JS analisa a declaração da função antes de o programa a executar. Posto isto, não faz diferença se o programa chama a função antes de esta estar definida, pois o JavaScript elevou a função para o topo do escopo atual no plano de fundo. A expressão da função não foi analisada até ser atribuída a uma variável. Por consequência, esta ainda está indefinida no momento em que é chamada.

Entender o Escopo das Variáveis

As variáveis podem ser declaradas em qualquer local do JS, no entanto, a localização da declaração irá estabelecer a extensão da disponibilidade de uma variável dentro do programa do JavaScript – este processo pode também ser chamado de **escopo das variáveis**. Por defeito, as variáveis declaradas dentro de uma função têm um escopo local, o que significa que não podem ser vistas ou controladas pelo exterior dessa função, tal como mostrado aqui. Mesmo que quaisquer variáveis declaradas fora de uma função, num programa, têm um escopo global, independentemente de onde estiver localizada, no *script*, a função em caso, tal como pode ser verificado aqui.

Objetos JavaScript

O JavaScript é uma linguagem baseada num objeto e, ali, práticamente tudo é um objeto ou age como um. Portanto, para trabalhar corretamente com o JS, os programadores necessitam de entender como funcionam, como se criam e como se usam os objetos.

Um objeto JavaScript é simplesmente uma coleção de valores com nomes. Estes são tipicamente referidos como propriedades de um objeto. Sendo uma matriz uma coleção de valores, onde cada valor tem um index (uma chave numérica) que começa em zero e aumenta um para cada valor. Um objeto é como uma matriz, mas a diferença é que o programador define as chaves (nome, idade, género, etc.).

Criar um Objeto

Os programadores podem criar objetos com chavetas, incluindo uma lista voluntária de propriedades. Uma propriedade pode ser um par “key:value”, onde a chave (ou nome da propriedade) é sempre uma linha, e o valor (ou valor da propriedade) pode ser qualquer tipo de dados (linhas, números, *Booleanos*, matrizes, funções, etc.). Para além disso, as propriedades com funções a representar os valores são, frequentemente, chamados de métodos para os distinguir de outras funcionalidades. Um objeto JS pode ser isto. Este exemplo cria um objeto chamado de pessoa, que tem 3 propriedades (nome, idade e género) e um método: displayName(). Este método mostra o valor de this.name, que vai ao acordo do *person.name*. Esta é a forma ideal e mais fácil de criar um objeto novo em JS, que é conhecido como a síntaxe literal de um objeto. A propriedade de nome não precisa, geralmente, de ser citada, com excessão se forem palavras reservadas ou se contiverem espaços ou caracteres especiais (qualquer outra coisa que não letras, números e os caracteres _ e $), ou se estas começam com um número, tal como mostrado aqui.

Aceder às Propriedades dos Objetos

De forma a aceder ou ter o valor de uma propriedade, podemos aplicar o ponto (.), assim como a notação em parênteses retos ([]), tal como mostra este exemplo. A notação do ponto é fácil de ler e de escrever, mas nem sempre pode ser usada. Se o nome da propriedade não for válido (por exemplo, se tiver caracteres especiais ou espaços), a notação do ponto não pode ser usada, tendo que usar a notação de

parênteses nesses casos. Esta mostra muita mais flexibilidade do que a notação do ponto e, para além disso, permite identificar os nomes das propriedades como variáveis de forma a substituir as literais das linhas.

Laços Através das Propriedades dos Objetos

Os programadores podem iterar através de pares de valores-chave de um objeto usando o içamento for...in. É especialmente valorizado por iterar sobre as propriedades dos objetos, tal como pode ser visto aqui.

Definir as Propriedades dos Objetos

Da mesma forma, podem-se definir propriedades novas ou atualizar as existentes ao usar a notação de ponto (.) ou de parênteses retos ([]), tal como demonstrado aqui.

Eliminar as Propriedades dos Objetos

O operador de eliminação pode ser aplicado para apagar, por completo, as propriedades de um objeto. Para nos livrarmos, por completo, de uma propriedade, temos obrigatoriamente de as remover. Ajustar a propriedade para um estado de indefinido ou nulo apenas vai alterar o seu valor, não removendo as propriedades do objeto. Portanto, não tem qualquer efeito em variáveis ou funções declaradas. Tendo isto em conta, os programadores devem evitar o operador de eliminação para apagar elementos de matriz, pois não altera o comprimento da matriz, mas cria apenas um buraco nela.

Invocar os Métodos dos Objetos

O método de um objeto pode ser acedido da mesma forma que acedemos às propriedades – usando uma notação de ponto ou aplicando a notação de parênteses retos, como pode ser verificado aqui.

Manipulação de Valores vs. Referências

Os objetos JS são tipos de referências, o que significa que, quando o programador faz cópias destes, os mesmos estão simplesmente a copiar as referências para esse objeto, enquanto os valores primitivos, como linhas ou números, são atribuídos ou copiados como um valor completo. Este exemplo demonstra esta ideia. Tal como pode ser observado, já foi feita uma cópia de uma mensagem variável e foi alterado o valor dessa mesma cópia. Ambas as variáveis se mantêm distintas e separadas. No entanto, se este mesmo princípio for aplicado num objeto, o resultado recolhido será diferente.

Portanto, qualquer alteração feita à variável do utilizador também vai interferir com a variável de pessoa, pois ambas se referem ao mesmo objeto. Concluindo, a mera cópia de um objeto não o duplica, mas copia apenas a referência desse objeto.

5.2. JavaScript e DOM

O que é o Documento de Modelo do Objeto? (DOM)?

DOM é criado pelo navegador quando uma página *web* é carregada em documentos HTML ou XML. É usada para definir as estruturas lógicas deste documento e para aceder e alterar os seus elementos.

O DOM HTML refere-se ao Documento de Modelo do Objeto em documentos HTML, equanto que o DOM XML refere-se ao Documento de Modelo do Objeto em documentos XML. Neste subcapítulo focaremos-nos no DOM HTML, que pode ser usado para aceder e manipular documentos HTML em JavaScript.

O DOM é construído numa árvore hierárquica de objetos, que incluem todas as partes de um documento HTML como elementos, atributos, texto, etc. Estes objetos individuais da árvore são também conhecidos como nós, e estes são compostos por nós pais e nós filhos. Numa forma gráfica, parece-se algo semelhante ao diagrama apresentado em baixo.

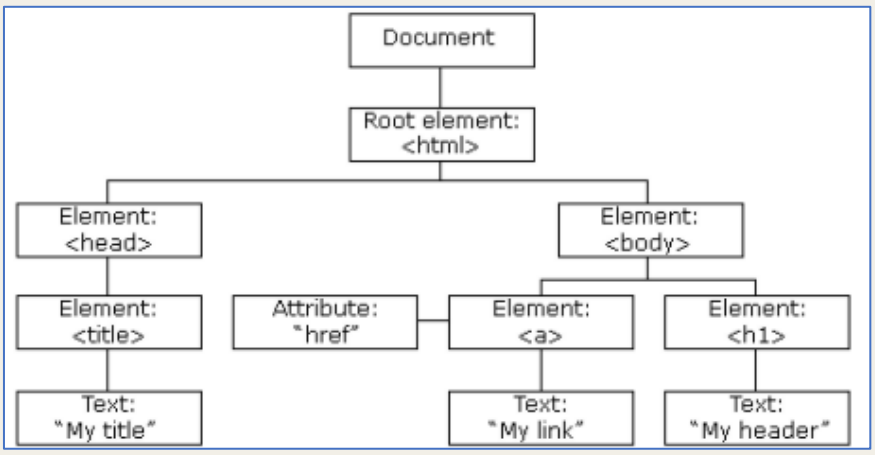


***Figura 91** – Árvore de Objetos DOM HTML (**Fonte:** https://www.w3schools.com/js/js_htmldom.asp)*

Dentro do DOM HTML, o JavaScript pode fazer os seguintes:

- mudar todos os elementos HTML na página
- mudar todos os atributos HTML na página
- mudar todos os estilos CSS na página
- remove existing HTML elements and attributes
- remover todos os elementos e atributos HTML
- adicionar todos os elementos e atributos HTML
- reagir a todos os eventos HTML existentes na página
- criar novos eventos HTML existentes na página

Selectores DOM em JavaScript

Selecionar Elementos DOM em JavaScript

O JavaScript é usado para criar ou modificar conteúdos ou valores de elementos numa página *web* HTML, assim como para aplicar alguns efeitos visuais como animações. Para ser capaz de realizar qualquer ação, é necessário encontrar ou selecionar um elemento alvo em HTML.

Iremos analisar algumas das formas mais comuns de selecionar elementos numa página e de os manipular com JavaScript.

Selecionar os Elementos Topmost

Os elementos *topmost* podem ser acedidos diretamente como propriedades de um documento.

Por exemplo, para aceder ao elemento <*html*>, deve-se usar a propriedade documentElement. Para o elemento <*head*>, podemos usar a propriedade *document.head* e, para o elemento <*body*>, usamos a propriedade document.*body*.

Vejamos este exemplo para entendermos um pouco melhor:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Topmost Elements</title>
</head>
<body>
    <script>
    // Display lang attribute value of html element
    alert(document.documentElement.getAttribute("lang")); // Outputs: en

    // Set background color of body element
    document.body.style.background = "yellow";

    // Display tag name of the head element's first child
    alert(document.head.firstElementChild.nodeName); // Outputs: meta
    </script>
</body>
</html>
```

Figura 92 *– Exemplo de Elementos Topmost (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-selectors.php)*

* É importante considerar que o *document.body* não deve ser usado antes do elemento <*body*>, pois, se tal acontecer, irá resultar em nulo. Este programa necessita de enveredar, primeiramente, pelo elemento <*body*> para que se possa aceder à propriedade document.*body*.

Vejamos este exemplo para entender porque é que o <body> ficaria nulo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Document.body Demo</title>
    <script>
    alert("From HEAD: " + document.body); // Outputs: null (since <body> is not parsed yet)
    </script>
</head>
<body>
    <script>
    alert("From BODY: " + document.body); // Outputs: HTMLBodyElement
    </script>
</body>
</html>
```

Figura 93 *- Exemplo de Elementos Topmost (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-selectors.php)*

Este exemplo demonstra o que vimos no incio sovre as relações hierárquicas que existem entre os nós. É necessário ser calculoso no sentido em que, de forma a aceder à propriedade *document*.*body*, irá ser necessário começar pelo elemento <*body*> para evitar valores nulos.

Selecionar Elementos pelo ID

Se quiser encontrar o selecionar um elemento HTML, a forma mais simples seria seleciona-lo com base no seu ID único. É possível fazer isto através do método getElementById().

No seguinte exemplo, o elemento que tem um atributo ID id=”mark” é selecionado e realçado:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Element by ID</title>
</head>
<body>
    <p id="mark">This is a paragraph of text.</p>
    <p>This is another paragraph of text.</p>

    <script>
    // Selecting element with id mark
    var match = document.getElementById("mark");

    // Highlighting element's background
    match.style.background = "yellow";
    </script>
</body>
</html>
```

Figura 94 *– Exemplo Selecionar Elementos pelo ID (**Fonte:*** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-selectors.php)

O método getElementById() é usado para devolver um elemento como objeto se for encontrado um objeto correspondente. De outra maneira, irá regressar nulo.

* É preciso relembrar que qualquer elemento HTML pode ter um atributo, que deve ser um **valor único na página**. Isto significa que apenas um elemento pode ter o mesmo id.

Selecionar Elementos pelo Nome da Classe

Se quiser selecionar todos os elementos com nomes de classes específicos, use o método getElementsByClassName(). Irá devolver um objeto semelhante a uma matriz com todos os seus elementos criança, que têm todos o nome de classe atribuído.

Vejamos este exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Elements by Class Name</title>
</head>
<body>
    <p class="test">This is a paragraph of text.</p>
    <div class="block test">This is another paragraph of text.</div>
    <p>This is one more paragraph of text.</p>

    <script>
    // Selecting elements with class test
    var matches = document.getElementsByClassName("test");

    // Displaying the selected elements count
    document.write("Number of selected elements: " + matches.length);

    // Applying bold style to first element in selection
    matches[0].style.fontWeight = "bold";

    // Applying italic style to last element in selection
    matches[matches.length - 1].style.fontStyle = "italic";

    // Highlighting each element's background through loop
    for(var elem in matches) {
        matches[elem].style.background = "yellow";
    }
    </script>
</body>
</html>
```

Figura 95 – Exemplo Selecionar Elementos pelo Nome da Classe Fonte: *https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-selectors.php)*

Selecionar Elementos pelo Nome da *Tag*

Se quiser selecionar elementos pelo seu nome da *tag*, use o método *getElementsByTagName*(). Este método irá também devolver um objeto semelhante a uma matriz com todos os seus elementos criança, que têm todos o nome de classe atribuído.

Vejamos este exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Elements by Tag Name</title>
</head>
<body>
    <p>This is a paragraph of text.</p>
    <div class="test">This is another paragraph of text.</div>
    <p>This is one more paragraph of text.</p>

    <script>
    // Selecting all paragraph elements
    var matches = document.getElementsByTagName("p");

    // Printing the number of selected paragraphs
    document.write("Number of selected elements: " + matches.length);

    // Highlighting each paragraph's background through loop
    for(var elem in matches) {
        matches[elem].style.background = "yellow";
    }
    </script>
</body>
</html>
```

***Figura 96** – Exemplo Selecionar Elementos pelo Nome da Tag (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-selectors.php)*

Selecionar Elementos com Seletores CSS

Os seletores CSS oferecem uma forma potente e eficiente de selecionar elementos HTML num documento. Para selecionar elementos que combinem com um sletor de CSS específico, podemos usar o método *querySelectorAll*().

Este método irá devolver uma lista de todos os elementos que combinam com os seletores específicos.

O exemplo em baixo serve para entender como se examina uma lista como uma matriz:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Select Elements with CSS Selectors</title>
</head>
<body>
    <ul>
        <li>Bread</li>
        <li class="tick">Coffee</li>
        <li>Pineapple Cake</li>
    </ul>

    <script>
    // Selecting all li elements
    var matches = document.querySelectorAll("ul li");

    // Printing the number of selected li elements
    document.write("Number of selected elements: " + matches.length + "<hr>")

    // Printing the content of selected li elements
    for(var elem of matches) {
        document.write(elem.innerHTML + "<br>");
    }

    // Applying line through style to first li element with class tick
    matches = document.querySelectorAll("ul li.tick");
    matches[0].style.textDecoration = "line-through";
    </script>
</body>
</html>
```

Figura 97 *- Exemplo Selecionar Elementos com Seletores CSS (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-selectors.php)*

*Por favor note que o método querySelectorAll() suporta pseudo-classes do CSS como :first-child, :last-child, :hover,etc. No entanto, este método irá sempre devolver uma lista vazia para pseudo-elementos CSS como: :before:, :after:, :first-line:, etc.

Estilo dos Elementos DOM em JavaScript

É também possível mudar a apresentação visual de um documento HTML de uma forma dinâmica ao usar o JavaScript para aplicar diferentes estilos aos elementos HTML. Quase todos os estilos de elementos podem ser adicionados, como fontes, cores, margens, bordas, imagens de fundo, alinhamento de texto, largura e altura, posição, entre outros.

Aqui iremos ver vários métodos que podem ser usados para definir estilos em JavaScript.

Definir Estilos *Inline* nos Elementos

Este atributo de estilo é usado para aplicar estilos *inline* diretamente no elemento HTML específico. A propriedade style é usada no JavaScript para ter ou definir o estilo *inline* num documento.

No seguinte exemplo, as propriedades das cores e fontes serão definidas para um elemento com id="intro":

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Set Inline Styles Demo</title>
</head>
<body>
    <p id="intro">This is a paragraph.</p>
    <p>This is another paragraph.</p>

    <script>
    // Selecting element
    var elem = document.getElementById("intro");

    // Appling styles on element
    elem.style.color = "blue";
    elem.style.fontSize = "18px";
    elem.style.fontWeight = "bold";
    </script>
</body>
</html>
```

Figura 98 *– Exemplo de Estilos Inline nos Elementos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-styling.php)*

Denominar Convenções de Propriedades CSS em JavaScript

É importante mencionar que muitas das propriedades de CSS contêm hífens (-) nos seus nomes como font-size, background-image, text-decoration, etc. No entanto, em JavaScript, o hifen é um operador reservado que significa o sinal menos. Posto isto, não é possível escrever uma expressão da seguinte maneira: elem.style.font-size.

De forma a resolver este problema, o nome das propriedades CSS em JavaScript que contêm um ou mais hífens são convertidos em estilos com palavras intercapatisalizadas. Isto, em resumo, significa que os hífens são removidos e a primeira letra após o hifen é maiúscula. Por exemplo, a propriedade CSS font-size, torna-se em propriedade DOM fontSize.

Obter a Informação do Estilo dos Elementos

A propriedade de estilo é também usada para aplicar os estilos aos elementos HTML. O exemplo abaixo irá obter informações de estilo sobre o elemento id="intro":

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JS Get Element's Style Demo</title>
</head>
<body>
    <p id="intro" style="color:red; font-size:20px;">This is a paragraph.</p>
    <p>This is another paragraph.</p>

    <script>
    // Selecting element
    var elem = document.getElementById("intro");

    // Getting style information from element
    alert(elem.style.color);  // Outputs: red
    alert(elem.style.fontSize);  // Outputs: 20px
    alert(elem.style.fontStyle);  // Outputs nothing
    </script>
</body>
</html>
```

Figura 99 *– Exemplo – Propriedade de Estilo (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-styling.php)*

A propriedade de estilo não é a mais útil na questão de recolher informação dos elementos, pois apenas devolve as regras de estilo que foram aplicadas no atributo de estilo do elemento e não naqueles que vieram de outros locais como regras de estilo das folhas de estilo implementadas ou em folhas de estilo externas.

Se quiser obter os valores de todas as propriedades CSS que são usadas para renderizar um objeto, pode utilizar o método *window.getComputedStyle()*, como mostrado no seguinte exemplo:

```html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>JS Get Computed Style Demo</title>
<style type="text/css">
    #intro {
        font-weight: bold;
        font-style: italic;
    }
</style>
</head>
<body>
    <p id="intro" style="color:red; font-size:20px;">This is a paragraph.</p>
    <p>This is another paragraph.</p>

    <script>
    // Selecting element
    var elem = document.getElementById("intro");

    // Getting computed style information
    var styles = window.getComputedStyle(elem);

    alert(styles.getPropertyValue("color"));  // Outputs: rgb(255, 0, 0)
    alert(styles.getPropertyValue("font-size"));  // Outputs: 20px
    alert(styles.getPropertyValue("font-weight"));  // Outputs: 700
    alert(styles.getPropertyValue("font-style"));  // Outputs: italic
    </script>
</body>
</html>
```

***Figura 100** – Exemplo - window.getComputedStyle() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-styling.php)*

* É preciso lembrar que o valor 700 para a propriedade *font-weight* do CSS é o mesmo que o negrito que usamos normalmente. A cor vermelha usada normalmente corresponde ao rgb(255,0,0), o que corresponde à notação rgb de uma cor.

Adicionar Classes CSS a Elementos

Outra forma de obter classes CSS em elementos HTML é usar a propriedade className. Uma classe é uma palavra reservada no JavaScript; posto isto, o JavaScript usa a propriedade className para se referir ao valor do atributo de classe HTML.

Observemos agora o seguinte exemplo para aprender a adicionar novas classes ou substituir as classes existentes com o elemento com o id="info":

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS Add or Replace CSS Classes Demo</title>
<style>
    .highlight {
        background: yellow;
    }
</style>
</head>
<body>
    <div id="info" class="disabled">Something very important!</div>

    <script>
    // Selecting element
    var elem = document.getElementById("info");

    elem.className = "note";  // Add or replace all classes with note class
    elem.className += " highlight";  // Add a new class highlight
    </script>
</body>
</html>
```

Figura 101 *– Exemplo Adicionar Classes aos Elementos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-styling.php)*

Uma forma ainda melhor de trabalhar com classes CSS é usando a propriedade *classList* para obter, estabelecer ou remover simplesmente as classes CSS de um elemento. Esta propriedade é suportada em todos os navegadores mais conhecidos, exceto as edições anteriores ao Internet Explorer 10.

Vejamos este exemplo:

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS classList Demo</title>
<style>
    .highlight {
        background: yellow;
    }
</style>
</head>
<body>
    <div id="info" class="disabled">Something very important!</div>

    <script>
    // Selecting element
    var elem = document.getElementById("info");

    elem.classList.add("hide");  // Add a new class
    elem.classList.add("note", "highlight");  // Add multiple classes
    elem.classList.remove("hide"); // Remove a class
    elem.classList.remove("disabled", "note"); // Remove multiple classes
    elem.classList.toggle("visible"); // If class exists remove it, if not add it

    // Determine if class exist
    if(elem.classList.contains("highlight")) {
        alert("The specified class exists on the element.");
    }
    </script>
</body>
</html>
```

***Figura 102** – Propriedade classList – Exemplo Adicionar Classes aos Elementos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-styling.php)*

Atributos Get Set DOM em JavaScript

Trabalhar com atributos

Os atributos são uma palavra especial usada, muitas vezes, dentro do início de uma *tag* de um elemento HTML de forma a controlar o comportamento da *tag* ou para obter mais informações sobre a *tag*.

Nesta seleção, iremos analisar vários métodos de adição, remoção ou alteração dos atributos de um elemento HTML.

Obter o Valor do Atributo do Elemento

Para obter o valor corrente do atributo de um elemento, temos que usar o método *getAttribute().* Se este atributo em particular não for encontrado no elemento, a resposta vai ser nula.

Atentemos ao seguinte exemplo:

```
1  <a href="https://www.google.com/" target="_blank" id="myLink">Google</a>
2
3  <script>
4      // Selecting the element by ID attribute
5      var link = document.getElementById("myLink");
6
7      // Getting the attributes values
8      var href = link.getAttribute("href");
9      alert(href); // Outputs: https://www.google.com/
10
11     var target = link.getAttribute("target");
12     alert(target); // Outputs: _blank
13 </script>
```

Figura 103 *– Método getAttribute()– Exemplo Obter o Valor do Atributo do Elemento (**Fonte:*** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-get-set-attributes.php)

Definir Atributos nos Elementos

Se quiser definir um atributo num elemento específico, poderá usar o método *setAttribute*(). Se o atributo já existir nesse elemento, o valor será atualizado. Se não, será adicionar um novo atributo com o nome e o valor específico.

No seguinte exemplo iremos adicionar uma classe e desativar um atributo com o elemento <*button*>:

```html
<button type="button" id="myBtn">Click Me</button>

<script>
    // Selecting the element
    var btn = document.getElementById("myBtn");

    // Setting new attributes
    btn.setAttribute("class", "click-btn");
    btn.setAttribute("disabled", "");
</script>
```

Figura 104 *- Método setAttribute()– Exemplo Definir o Valor do Atributo do Elemento (**Fonte:*** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-get-set-attributes.php)

Se quiser atualizar ou alterar o valor de um atributo existente de um elemento, pode usar o método *setAttribute*().

Vejamos um exemplo que irá atualizar o valor de um atributo *href* existente de uma âncora (<a>) do elemento:

```html
<a href="#" id="myLink">Tutorial Republic</a>

<script>
    // Selecting the element
    var link = document.getElementById("myLink");

    // Changing the href attribute value
    link.setAttribute("href", "https://www.tutorialrepublic.com");
</script>
```

Figura 105 *- Método setAttribute()– Exemplo Atualizar ou Alterar o Valor do Atributo do Elemento (**Fonte:*** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-get-set-attributes.php)

Remover Atributos dos Elementos

Para remover um atributo de um elemento específico, podemos usar o método *removeAttribute*().

Tenha em mente o atributo *href* que alteramos do elemento âncora; iremos agora removê-lo, como se vê no seguinte exemplo:

```html
<a href="https://www.google.com/" id="myLink">Google</a>

<script>
    // Selecting the element
    var link = document.getElementById("myLink");

    // Removing the href attribute
    link.removeAttribute("href");
</script>
```

***Figura 106** – Método removeAttribute()–Exemplo Remover o Atributo do Elemento*

*(**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-get-set-attributes.php)*

Manipulação DOM em JavaScript

Manipular Elementos DOM em JavaScript

Até agora, aprendemos como selecionar e alterar o estilo dos elementos DOM HTML. Agora, irmos aprender como adicionar ou remover elementos DOM de uma forma dinâmica, como obter os seus conteúdos e muito mais.

Adicionar Novos Elementos ao DOM

O método *document.createElement*() é usado para criar um novo elemento num documento HTML. Este cria um novo elemento, no entanto, não o adiciona ao DOM.

Para adicioná-lo ao DOM, é necessário um passo separado, tal como mostrado no exemplo abaixo:

```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
// Creating a new div element
var newDiv = document.createElement("div");

// Creating a text node
var newContent = document.createTextNode("Hi, how are you doing?");

// Adding the text node to the newly created div
newDiv.appendChild(newContent);

// Adding the newly created element and its content into the DOM
var currentDiv = document.getElementById("main");
document.body.appendChild(newDiv, currentDiv);
</script>
```

Figura 107 *– Exemplo Adicionar Novos Elementos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-manipulation.php)*

No exemplo acima, o *appendChild*() é usado para adicionar um novo elemento ao fim de cada outra no *nó filho*, dentro do nó de parente específico.

Tem também a opção de adicionar novos elementos antes de qualquer outro filho, tal como é mostrado no exemplo abaixo:

```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
// Creating a new div element
var newDiv = document.createElement("div");

// Creating a text node
var newContent = document.createTextNode("Hi, how are you doing?");

// Adding the text node to the newly created div
newDiv.appendChild(newContent);

// Adding the newly created element and its content into the DOM
var currentDiv = document.getElementById("main");
document.body.insertBefore(newDiv, currentDiv);
</script>
```

Figura 108 *– Exemplo Adicionar Novos Elementos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-manipulation.php)*

Obter ou Estabelecer Conteúdos HTML para DOM

Se quiser obter ou estabelecer os conteúdos de elementos HTML, pode usar a propriedade *innerHTML*. Esta propriedade é usada obter ou estabelecer um *markup* HTML dentro do elemento, que contem conteúdo entre as *tag*s de abertura e de fecho.

Vejamos o exemplo para entendermos melhor:

```html
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
// Getting inner HTML conents
var contents = document.getElementById("main").innerHTML;
alert(contents); // Outputs inner html contents

// Setting inner HTML contents
var mainDiv = document.getElementById("main");
mainDiv.innerHTML = "<p>This is <em>newly inserted</em> paragraph.</p>";
</script>
```

Figura 109 *- Exemplo Obter ou Estabelecer Conteúdos HTML para DOM (**Fonte:***
https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-manipulation.php*)*

Como é possível observar no exemplo, os novos elementos são inseridos com facilidade no DOM através da propriedade *innerHTML*. Mas esta propriedade substitui todo o conteúdo existente de um elemento.

Posto isto, se não quiser alterar o conteúdo existente de um elemento, deve usar o método *insertAdjacentHTML*(). Este método requer dois fatores: a inserção do HTML e a sua posição. A posição pode ser uma destas: “beforebegin", "afterbegin", "beforeend", e "afterend". É importante notar que este método é suportado por todos os navegadores de grande dimensão.

No seguinte exemplo pode-se observar como funciona o posicionamento:

```
<!-- beforebegin -->
<div id="main">
    <!-- afterbegin -->
    <h1 id="title">Hello World!</h1>
    <!-- beforeend -->
</div>
<!-- afterend -->

<script>
// Selecting target element
var mainDiv = document.getElementById("main");

// Inserting HTML just before the element itself, as a previous sibling
mainDiv.insertAdjacentHTML('beforebegin', '<p>This is paragraph one.</p>');

// Inserting HTML just inside the element, before its first child
mainDiv.insertAdjacentHTML('afterbegin', '<p>This is paragraph two.</p>');

// Inserting HTML just inside the element, after its last child
mainDiv.insertAdjacentHTML('beforeend', '<p>This is paragraph three.</p>');

// Inserting HTML just after the element itself, as a next sibling
mainDiv.insertAdjacentHTML('afterend', '<p>This is paragraph four.</p>');
</script>
```

***Figura 110** - Método insertAdjacentHTML() – Exemplo Obter ou Estabelecer Conteúdos HTML para DOM (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-manipulation.php)*

* É preciso notar que para que as posições *beforebegin* e *afterbegin* funcionem, o nó tem de estar na árvore DOM e ter um elemento parente.

* Quando se inserir um HTML numa página, é também preciso ter em atenção para que o input do utilizador não tenha sido limpo ou eliminado, de forma a prever ataques XSS.

Remover Elementos Existentes do DOM

Para remover um nó filho do DOM, pode-se usar o método *removeChild*(). Este método irá, também, devolver o nó removido.

Atentemos o exemplo abaixo:

```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
var parentElem = document.getElementById("main");
var childElem = document.getElementById("hint");
parentElem.removeChild(childElem);
</script>
```

Figura 111 *- Exemplo Remover Elementos Existentes do DOM (**Fonte:***
https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-manipulation.php)

É também possível remover o elemento filho sem saber o elemento parente. Pode encontrar o elemento filho e usar a propriedade *parentNode* para encontrar o seu parente. Irá devolver o parente desse nó à árvore DOM.

```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
var childElem = document.getElementById("hint");
childElem.parentNode.removeChild(childElem);
</script>
```

Figura 112 *– Exemplo Remover Elementos Existentes em DOM (**Fonte:***
https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-manipulation.php)

Substituir Elementos Existentes em DOM

Pode também ter a opção de substituir um elemento em HTML DOM por outro ao usar o método replaceChild(). Este método requer dois parâmetros: que o nó esteja inserido e que o nó seja substituído. A síntaxe usada é a seguinte:

```
parentNode.replaceChild(newChild, oldChild);
```

```
1  <div id="main">
2      <h1 id="title">Hello World!</h1>
3      <p id="hint">This is a simple paragraph.</p>
4  </div>
5
6  <script>
7  var parentElem = document.getElementById("main");
8  var oldPara = document.getElementById("hint");
9
10 // Creating new elememt
11 var newPara = document.createElement("p");
12 var newContent = document.createTextNode("This is a new paragraph.");
13 newPara.appendChild(newContent);
14
15 // Replacing old paragraph with newly created paragraph
16 parentElem.replaceChild(newPara, oldPara);
17 </script>
```

Figura 113 *– Exemplo Substituir Elementos Existentes em DOM (**Fonte:***
https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-manipulation.php)

Navegação DOM em JavaScript

Navegar Entre Nós DOM

Neste momento já deve ter uma melhor ideia sobre como selecionar elementos individuais numa página *web*. Existem várias ocasiões onde é preciso aceder a elementos filhos, parentes ou ancestores. Fomos falando em nós desde o ínico deste subcapítulo e vamos agora ver como podemos aceder aos diferentes tipos de nós.

Os nós DOM têm múltiplas propriedades e métodos que permitem navegar ou percorrer a estrutura das árvores DOM e fazer as alterações necessárias de uma forma simples.

Aceder a Nós Filhos

As propriedades firstChild e lastChild cedem o acesso direto ao primeiro e ao último nó respetivamente. Se um nó não tiver nenhum elemento filho, será devolvido como nulo.

Vejamos o exemplo abaixo:

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var main = document.getElementById("main");
console.log(main.firstChild.nodeName); // Prints: #text

var hint = document.getElementById("hint");
console.log(hint.firstChild.nodeName); // Prints: SPAN
</script>
```

Figura 114 *– Exemplo Aceder a Nós Filhos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-navigation.php)*

* É necessário notar que nodeName é uma propriedade apenas de leitura, o que devolve o nome do nó atual como linha. Por exemplo, irá devolver o nome da *tag* de um elemento de um nó, #text para nó text, #comment para nó comment, #document para nó document, e muitos outros.

Neste exemplo que vimos, o nodeName do nó do primeiro filho do principal elemento DIV foi devolvido como #text em vez de H1. Isto deveu-se à existência de um espaço branco, p.e.: espaços, linhas novas, *tag*s, entre outros, são considerados caracteres válidos e tornam-se parte da árvore DOM na forma de nós #text. Depois, a *tag* , que conrem uma nova linha antes do <h1>, irá criar o nó #text.

De forma a prevenir este problema com o firstChild e lastChild a devolverem nós #text e #comment, podemos usar as propriedades firstElementChild ou lastElementChild como alternativa. No entanto, isto não irá funcionar nas versões anteriores do Internet Explorer 9.

O exemplo abaixo mostra uma melhor explicação disto:

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var main = document.getElementById("main");
alert(main.firstElementChild.nodeName); // Outputs: H1
main.firstElementChild.style.color = "red";

var hint = document.getElementById("hint");
alert(hint.firstElementChild.nodeName); // Outputs: SPAN
hint.firstElementChild.style.color = "blue";
</script>
```

Figura 115 *- Exemplo Aceder a Nós Filhos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-navigation.php)*

Para aceder a todos os nós filhos de um elemento dado, podemos usar a propriedade *childNodes*. É necessário ter em conta que ao nó do primeiro filho é-lhe atribuído o index de 1.

Atente o exemplo abaixo:

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var main = document.getElementById("main");

// First check that the element has child nodes
if(main.hasChildNodes()) {
    var nodes = main.childNodes;

    // Loop through node list and display node name
    for(var i = 0; i < nodes.length; i++) {
        alert(nodes[i].nodeName);
    }
}
</script>
```

***Figura 116** - Exemplo Aceder a Nós Filhos ((**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-navigation.php)*

Aqui, os childNodes devolvem *todos os nós filhos*, incluindo nós não-elementos como nós de texto e comentários.

Se quiser ter uma *coleção de elementos apenas*, terá de usar a propriedade children em substituição.

Vejamos o exemplo em baixo para entender a propriedade:

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var main = document.getElementById("main");

// First check that the element has child nodes
if(main.hasChildNodes()) {
    var nodes = main.children;

    // Loop through node list and display node name
    for(var i = 0; i < nodes.length; i++) {
        alert(nodes[i].nodeName);
    }
}
</script>
```

Figura 117 *- Exemplo Aceder a Nós Filhos ((**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-navigation.php)*

Aceder a Nós Parentes

Para aceder ao nó parente de um nó específico da árvore DOM, podemos usar a propriedade *parentNode*.

* É de salientar que a propriedade *parentNode* irá sempre retomar valores nulos para nós em documentos, pois estes não têm parentes.

No seguinte exemplo podemos observar como funciona a propriedade *parentNode*:

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var hint = document.getElementById("hint");
alert(hint.parentNode.nodeName); // Outputs: DIV
alert(document.documentElement.parentNode.nodeName); // Outputs: #document
alert(document.parentNode); // Outputs: null
</script>
```

***Figura 118** - Exemplo Aceder a Nós Parentes ((**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-navigation.php)*

É bom saber que os nós *topmost* das árvores DOM pode ser acedidos diretamente como propriedades dos documentos. Vimos alguns exemplos sobre os nós topmost das árvores DOM em seleções anteriores como o elemento <*html*>, que pode ser acedido com a propriedade *document*.*documentElement*. Igualmente, o elemento <*head*> pode ser acedidio com a propriedade *document*.*head* e o elemento <*body*> pode ser acedido com a propriedade document.*body*.

Existe também uma opção para obter apenas um nó de elemento com o *parentElement*, tal como está exemplificado no seguinte exemplo:

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var hint = document.getElementById("hint");
alert(hint.parentNode.nodeName); // Outputs: DIV
hint.parentNode.style.backgroundColor = "yellow";
</script>
```

***Figura 119** - Exemplo Aceder a Nós Parentes (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-navigation.php)*

Aceder a Nós Irmãos

Para aceder aos anteriores e aos próximos nós numa árvore DOM, podemos usar, respetivamente, as propriedades *previousSibling* e *nextSibling*.

Vejamos o exemplo:

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p><hr>
</div>

<script>
var title = document.getElementById("title");
alert(title.previousSibling.nodeName); // Outputs: #text

var hint = document.getElementById("hint");
alert(hint.nextSibling.nodeName); // Outputs: HR
</script>
```

Figura 120 *- Exemplo Aceder a Nós Irmãos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-navigation.php)*

Para avançar qualquer nó de texto com espaços em branco, podemos usar *previousElementSiblinh* e *nextElementSibling* como alternativas para obter os elementos irmãos anteriores e posteriores. Se não for encontrado nenhum irmão, estas propriedades irão ser devolvidas como valores nulos.

Vejamos o exemplo em baixo:

```
<div id="main">
    <h1 id="title">My Heading</h1>
    <p id="hint"><span>This is some text.</span></p>
</div>

<script>
var hint = document.getElementById("hint");
alert(hint.previousElementSibling.nodeName); // Outputs: H1
alert(hint.previousElementSibling.textContent); // Outputs: My Heading

var title = document.getElementById("title");
alert(title.nextElementSibling.nodeName); // Outputs: P
alert(title.nextElementSibling.textContent); // Outputs: This is some text.
</script>
```

Figura 121 *- Exemplo Aceder a Nós Irmãos (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-navigation.php)*

A propriedade textContent que é usada aqui significa o conteúdo de um nó e todos os seus descendentes.

Tipos de Nós DOM

Uma árvore DOM é composta por diferentes tipos de nós que incluem elementos, texto, comentários e muito mais.

Todos os nós têm a propriedade nodeType que consegue ajudar a entender como pode aceder e manipular o nó em questão. A seguinte tabela oferece uma lista dos tipos de nó mais usados e mais importantes:

Constant	Value	Description
ELEMENT_NODE	1	An element node such as `<p>` or `<img>`.
TEXT_NODE	3	The actual text of element.
COMMENT_NODE	8	A comment node i.e. `<!-- some comment -->`
DOCUMENT_NODE	9	A document node i.e. the parent of `<html>` element.
DOCUMENT_TYPE_NODE	10	A document type node e.g. `<!DOCTYPE html>` for HTML5 documents.

***Figura 122** – Tabela dos Tipos mais Comuns de Tabelas DOM (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dom-navigation.php)*

5.3. JavaScript e BOM

Janela JavaScript - O Modelo de Objeto do Navegador

O Modelo do Objeto do Navegador (BOM) permite que o JavaScript "fale" para o navegador.

O Modelo do Objeto do Navegador (BOM)

Não existem *standards* oficiais para o Modelo de Objeto do Navegador (BOM).
Tendo em conta que os navegadores modernos já implementaram (praticamente) os mesmos métodos e propriedades para a interativifade do JavaScript, este é muitas vezes referido como métodos e propriedades do BOM.

O Objeto da Janela

O objeto da janela é suportado por todos os navegadores. Representa a janela do navegador.
Todos os objetos, funções e variáveis globais de JavaScript tornam-se, automaticamente, membros de um objeto da janela.
As variáveis globais são propriedades do objeto da janela.
As funções globais são métodos de um objeto da janela.
Até os objetos de um documento (HTML DOM) é uma propriedade do objeto da janela:

```
window.document.getElementById("header");
```

Figura 123 *– O Objeto da Janela em JS BOM (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window.php)*

É o mesmo que:

```
document.getElementById("header");
```

***Figura 124** – O Objeto da Janela em JS BOM – versão alternativa (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window.php)*

Tamanho da janela

Duas propriedades podem ser usadas para determinar o tamanho da janela do Navegador.

Ambas as propriedades podem ser devolvidas ao valor inicial.

Ambas propriedades retomam o tamanho em pixéis:

- *window.innerHeight* – o tamanho interior da altura da janela do navegador (em pixeis)
- *window.innerWidth* - o tamanho interior da largura da janela do navegador (em pixeis)

```
let w = window.innerWidth;
let h = window.innerHeight;
```

***Figura 125** – Alternativas de tamanho de janela (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window.php)*

Outros Métodos da Janela

Aguns outros médicos:

- window.open() – abrir uma nova janela
- window.close() – fechar uma tamanha aberta
- window.moveTo() – mover a porta atual
- window.resizeTo() – redimensionar a minha porta atual.

Ecrã de Janela em JavaScript

O objeto window.screen contem informação sobre o ecrã dos utilizadores.

O objeto window.screen pode ser escrito sem o prefixo “window”.

Propriedades:

- screen.width
- screen.height
- screen.availWidth
- screen.availHeight
- screen.colorDepth
- screen.pixelDepth

Largura do Ecrã da Janela

A propriedade screen.width devolve a largura do ecrã do visitante em pixéis.

Example

Display the width of the screen in pixels:

```
document.getElementById("demo").innerHTML =
"Screen Width: " + screen.width;
```

Result will be:

```
Screen Width: 1280
```

Figura 126 *– A propriedade screen.width (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window.php)*

Altura do Ecrã da Janela

A propriedade *screen.height* devolve a altura do ecrã do visitante em pixéis.

Figura 127 *– A propriedade screen.height da Janela (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window.php)*

Largura Disponível do Ecrã da Janela

A propriedade *screen.availWidth* devolve a largura do ecrã do visitante em pixéis, excluindo funcionalidades de interface como a barra de tarefas.

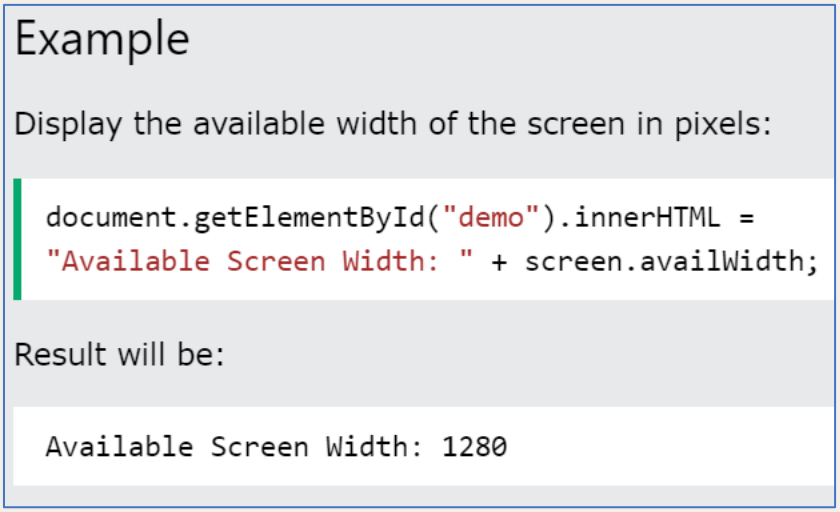


Figura 128 *– A propriedade screen.availWidth da janela (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window.php)*

Altura Disponível do Ecrã da Janela

A propriedade *screen.availHeight* devolve a altura do ecrã do visitante em pixéis, excluindo funcionalidades de interface como a barra de tarefas.

Example

Display the available height of the screen in pixels:

```
document.getElementById("demo").innerHTML =
"Available Screen Height: " + screen.availHeight;
```

Result will be:

```
Available Screen Height: 680
```

Figura 129 *– A propriedade screen.availWidth da janela (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window.php)*

Dencidade da Cor do Ecrã da Janela

A propriedade *screen.colorDepth* devolve o número de bits usados para mostrar uma cor.Todos os computadores modernos usam hardware de 24-bit ou 32-bit para a resolução da cor:

24 bits = 16,777,216 “Cores verdadeiras” diferentes
32 bits = 4,294,967,296 “Cores escuras” diferentes

Os computadores mais antigos usam 16-bit: 65,536 resoluções de "cores vivas" diferentes. Os computadores ainda mais velhos, ou telemóveis mais velhos, usam 8-bit: 256 “Cores VGA” diferentes.

Example

Display the color depth of the screen in bits:

```
document.getElementById("demo").innerHTML =
"Screen Color Depth: " + screen.colorDepth;
```

Result will be:

```
Screen Color Depth: 24
```

Figura 130 *– A propriedade screen.availWidth da janela (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window.php)*

Profundidade de Pixeis do Ecrã da Janela

A propriedade screen.pixelDepth devolve a profundidade dos pixéis no ecrã.

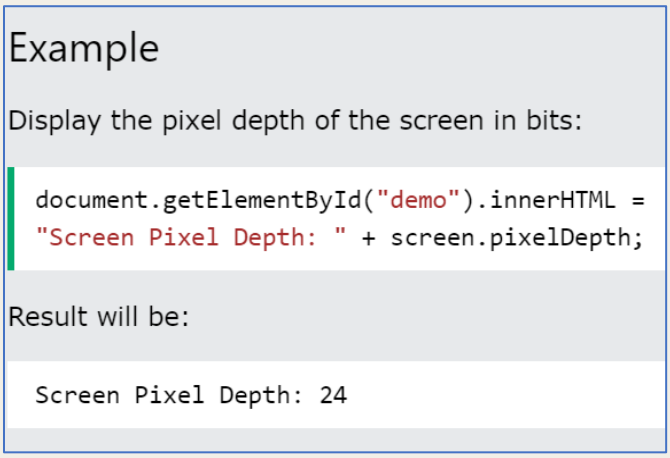


***Figura 131** – A propriedade screen.pixelDepth da janela (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window.php)*

Localização da Janela em JavaScript

O objeto *window.location* pode ser usado para obter um endereço de página atual (URL) e redirecionar o navegador para uma nova página.

Localização da Janela

O objeto window.location pode ser escrito sem o prefixo “window”.

Alguns exemplos:

- window.location.href devolve o href (URL) da página atual
- window.location.*host*name devolve o nome de domínio ao anfitrião da página
- window.location.pathname devolve o caminho e o filename da página atual
- window.location.protocol devolve o protocolo *web* usado (http: ou https:)
- window.location.assign() carrega um novo documento

Localização do Href na Janela

A propriedade *window.location*.href devolve o URL da página atual.

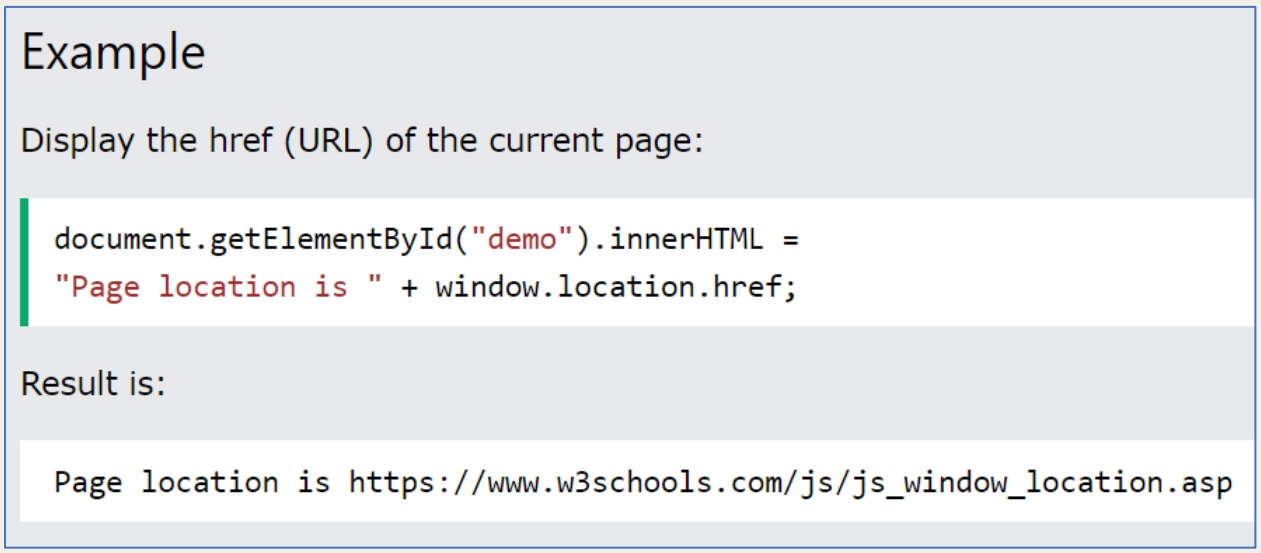


Figura 132 *– A propriedade window.location.href da janela (**Fonte:***
https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window-location.php*)*

Localização do *Host*name na Janela

A propriedade *window.location.hostname* devolve o nome do anfitrião da internet (da página atual).

Figura 133 *– A propriedade window.location.hostname da janela (**Fonte:***
https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window-location.php*)*

Localizar Pathname na Janela

A propriedade *window.location.pathname* devolve o *pathname* da página atual.

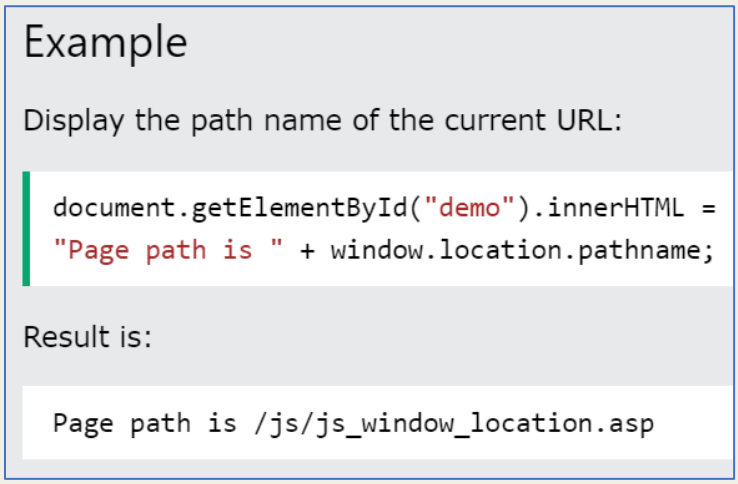


Figura 134 *– A propriedade window.location.pathname da janela (**Fonte:***
https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window-location.php*)*

Protocolo de Localização na Janela

A propriedade *window.location.protocol* devolve o protocolo *web* da página.

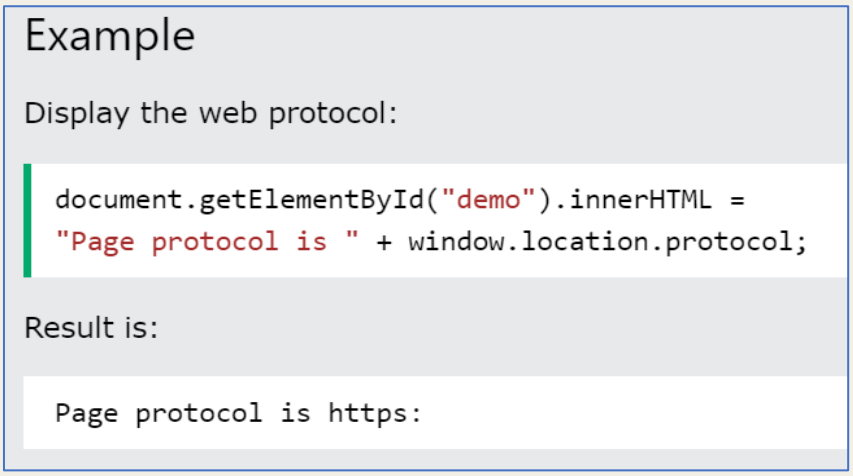


Figura 135 *– A propriedade window.location.protocol da janela (**Fonte:***
https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window-location.php*)*

Localização do Port na Janela

A propriedade *window.location.port* exibe o número de *hosts* (na página atual).

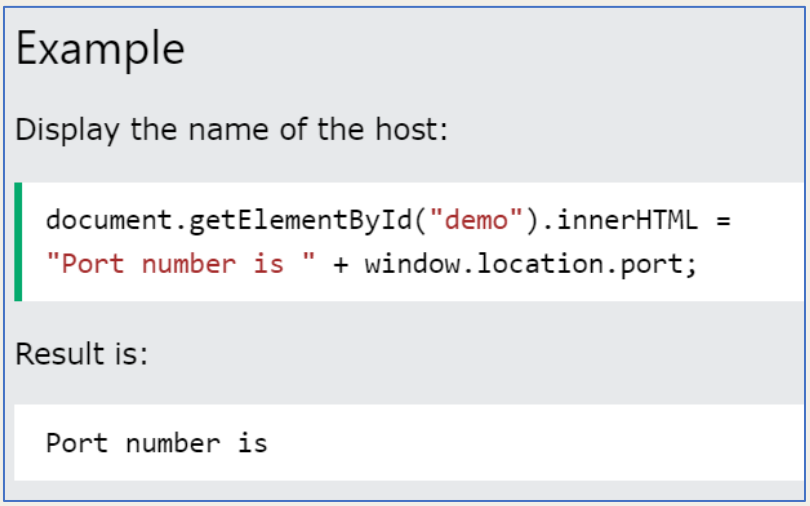


Figura 136 *– A propriedade window.location.port da janela (**Fonte:***
https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window-location.php)

Atribuição da Localização na Janela

O método *window.location.assign*() carrega um novo documento.

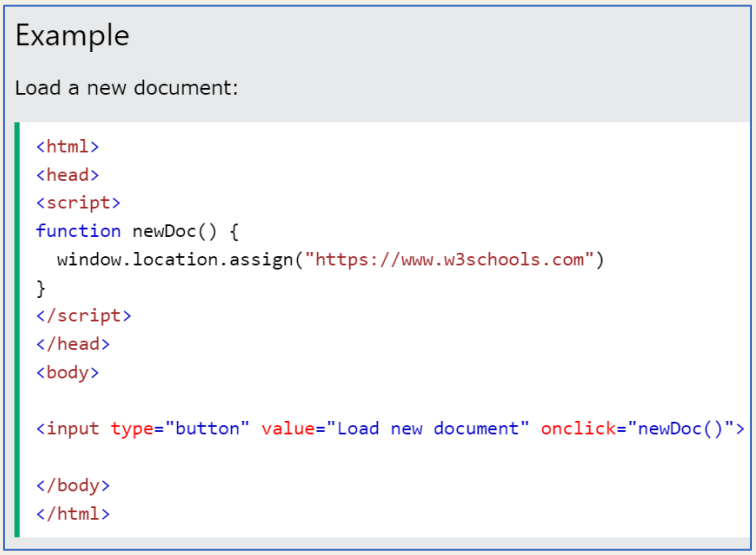


Figura 137 *– A propriedade window.location.assign() da janela (**Fonte:***
https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window-location.php)

Histórico da Janela de JavaScript

O objeto *window.history* contém o histórico do navegador.

Histórico da Janela

O objeto *window.history* pode ser escrito sem o prefixo "*window*".

Para proteger a privacidade dos utilizadores, existem limitações sobre como o JavaScript pode aceder a este objeto.

Alguns métodos:

- history.back() – O mesmo que carregar para trás no navegador.
- history.forward() – O mesmo que carregar para a frente no navegador.

Histórico da Janela para Trás

O método history.back() carrega o URL anterior na lista do histórico.

É o mesmo que carregar no botão de retroceder do navegador.

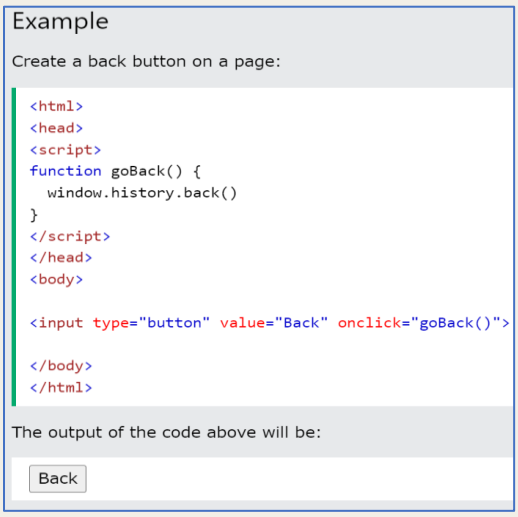


***Figura 138** – A propriedade history.back() da janela (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window-location.php)*

Histórico da Janela para a Frente

O método *history.forward*() carrega o próximo URL na lista do histórico.

Isto é o mesmo que carregar no botão de avançar no navegador.

Example

Create a forward button on a page:

```
<html>
<head>
<script>
function goForward() {
  window.history.forward()
}
</script>
</head>
<body>

<input type="button" value="Forward" onclick="goForward()">

</body>
</html>
```

The output of the code above will be:

Forward

Figura 139 *– A propriedade history.forward() da janela (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window-location.php)*

Janela de Navegação em JavaScript

O objeto *window.navigator* contem informação sobre o navegador do visitante.

Janela de Navegação

O objeto window.navigator pode ser escrito sem o prefixo "*window*".

Alguns exemplos:

- navigator.appName
- navigator.appCodeName
- navigator.platform

Cookies no Navegador

A propriedade *cookieEnabled* retoma verdadeiro se as *cookies* estiverem ativas, se não, o resultado será "*falso*":

```
document.getElementById("demo").innerHTML =
"cookiesEnabled is " + navigator.cookieEnabled;
```

***Figura 140** – A propriedade history.forward() da janela (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-window-location.php)*

Nome da Aplicação do Navegador

A propriedade *appName* devolve o nome da aplicação do navegador.

Código do Nome da Aplicação do Navegador

A propriedade *appCodeName* devolve o código do nome da aplicação do navegador.

Motor do Navegador

A propriedade *product* devolve o nome do produto do motor do navegador.

Versão do Navegador

A propriedade *appVersion* devolve informação sobre a versão do navegador.

Agente do Navegador

A propriedade *userAgent* devolve o cabeçalho do agente de utilizador enviado do navegador para o servidor.

Aviso!!!

A informação do objeto de navegação pode, por vezes, ser enganosa, e não deve ser usada para detetar as versões do navegador pelos seguintes motivos:

- Navegadores diferentes podem usar o mesmo nome;
- Os dados do navegador podem ser mudados pelo proprietário do navegador;
- Alguns navegadores não se identificam corretamente para boicotar os testes do *website;*

- Os navegadores não conseguem reportar novos sistemas de operação, lançados mais tarde do que o navegador.

Plataforma do Navegador

A propriedade *platform* devolve a plataforma do navegador (sistema operativo).

Linguagem do Navegador

A propriedade *language* devolve a linguagem do navegador.

O Navegador **é *Online*?**

A propriedade *online* retoma verdadeiro se o navegador for *online*.

O Java está ativo?

O método *javaEnabled*() retoma verdadeiro se o Java estiver ativo.

Caixas Popup em JavaScript

O JavaScript tem três tipos de caixas popup: Caixa de Alerta, Caixa de Confirmação e Caixa de Solicitação.

Caixa de Alerta

Uma caixa de alerta é muitas vezes usada para garantir que a informação chega ao utilizador.

Quando uma caixa de alerta aparece, o utilizador terá que carregar “OK” para continuar.

Síntaxe:

```
window.alert("sometext");
```

Figura 141 *– Síntaxe de caixa de alerta (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dialog-boxes.php)*

O método *window.alert*() pode ser escrito sem o prefixo "*window*".

```
alert("I am an alert box!");
```

Figura 142 *– O método window.alert() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dialog-boxes.php)*

Caixa de Confirmação

A caixa de confirmação é, por norma, usada no caso de querer que o utilizador verifique ou aceite algo.

Quando uma caixa de confirmação aparece, o utilizador terá que carregar "OK" ou "Cancelar" para continuar.

Se o utilizador carregar em "OK", a caixa retoma como verdadeira. Se o utilizador carregar em "Cancelar", a caixa retoma falsa.

Síntaxe:

```
window.confirm("sometext");
```

Figura 143 *– Método window.alert() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dialog-boxes.php)*

O método *window.confirm*() pode ser escrito sem o prefixo "*window*".

```
if (confirm("Press a button!")) {
  txt = "You pressed OK!";
} else {
  txt = "You pressed Cancel!";
}
```

Figura 144 *– Método window.confirm() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dialog-boxes.php)*

Caixa de Solicitação

A caixa de solicitação é, por norma, usada se quiser que o utilizador coloque um valor antes de entrar na página.

Quando uma caixa de solicitação aparece, o utilizador tem de carregar em “OK” ou em “Cancelar” de forma a proceder depois de introduzir um valor.

Se o utilizador carregar em “OK”, a caixa retoma um valor de entrada. Se o utilizador escolher “Cancelar”, a caixa dará o resultado de “falso”.

Síntaxe:

```
window.prompt("sometext","defaultText");
```

Figura 145 *– A Síntaxe prompt box (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dialog-boxes.php)*

O método *window.prompt*() pode ser escrito sem o prefixo “*window*”.

```
let person = prompt("Please enter your name", "Harry Potter");
let text;
if (person == null || person == "") {
  text = "User cancelled the prompt.";
} else {
  text = "Hello " + person + "! How are you today?";
}
```

Figura 146 *– Método window.prompt() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dialog-boxes.php)*

Quebra de Linha

Para mostrar quebras de linha dentro de uma caixa popup, é necessário usar a barra inclinada para trás seguida do caracter “n”.

```
alert("Hello\nHow are you?");
```

Figura 147 *–Mostrar quebras de linha dentro de uma caixa popup(**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-dialog-boxes.php)*

Eventos de Tempo em JavaScript

Eventos de Tempo

O objeto da janela permite a execução de um código em intervalos de tempo específicos.

Estes intervalos de tempo são chamados “eventos de tempo”.

Os dois métodos essenciais para usar com o JavaScript são:

- setTimeout(*function,milliseconds*)

  Executa uma função, depois espera um número específico de milissegundos.

- setInterval(function,milliseconds)
  O mesmo que o *setTimeout*(), mas repete a execução da função continuamente.

O *setTimeout*() e o *setInterval*() são ambos métodos do Objeto da Janela HTML em DOM.

Método setTimeout()

```
window.setTimeout(function, milliseconds);
```

Figura 148 *– Método setTimeout() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-timers.php)*

O método *window.setTimeout*() pode ser escrito sem o prefixo "*window*".
O primeiro parâmetro é que uma função seja exequível.

O segundo parâmetro indica o número de milissegundos antes da execução.

Example

Click a button. Wait 3 seconds, and the page will alert "Hello":

```
<button onclick="setTimeout(myFunction, 3000)">Try it</button>

<script>
function myFunction() {
  alert('Hello');
}
</script>
```

Figura 149 *– O método window.setTimeout() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-timers.php)*

Como Parar a Execução?

Os métodos *clearTimeout*() para a execução de uma função específica em *setTimeout*().

```
window.clearTimeout(timeoutVariable)
```

Figura 150 *– Método clearTimeOut() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-timers.php)*

O método *window.clearTimeout*() pode ser escrito sem o prefixo "*window*".
O método *clearTimeout*() usa a variável devolvida do *setTimeout*():

```
myVar = setTimeout(function, milliseconds);
clearTimeout(myVar);
```

Figura 151 *– Método window.clearTimeout() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-timers.php)*

Se a função ainda não tiver sido executada, podemos parar a execução ao usar o método *clearTimeout*():

Example

Same example as above, but with an added "Stop" button:

```
<button onclick="myVar = setTimeout(myFunction, 3000)">Try it</button>

<button onclick="clearTimeout(myVar)">Stop it</button>
```

Figura 152 *– Método clearTimeout() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-timers.php)*

O Método setInterval()

O método *setInterval*() repete uma função dada em cada intervalo de tempo dado.

```
window.setInterval(function, milliseconds);
```

Figura 153 *– Método setInterval() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-timers.php)*

O método *window.setInterval*() pode ser escrito sem o prefixo "*window*".

O primeiro parâmetro é uma função que seja exequível.

O segundo parâmetro indica o comprimento do intervalo de tempo entre cada execução.

Este exemplo executa uma função chamada "*myTimer*" uma vez em cada segundo (como um relógio digital).

Example

Display the current time:

```
setInterval(myTimer, 1000);

function myTimer() {
  const d = new Date();
  document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
```

Figura 154 *– Método window.setInterval() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-timers.php)*

Como Para um Execução?

O método *clearInterval*() para as execuções de uma função específica no método *setInterval*().

```
window.clearInterval(timerVariable)
```

Figura 155 *– Método clearInterval() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-timers.php)*

O método *window.clearInterval*() method pode ser escrito sem o prefixo "*window*".

O método *clearInterval*() usa a variável devolvida por *setInterval*():

```
let myVar = setInterval(function, milliseconds);
clearInterval(myVar);
```

Figura 156 *– A variável devolvida de setInterval() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-timers.php)*

Example

Same example as above, but we have added a "Stop time" button:

```
<p id="demo"></p>

<button onclick="clearInterval(myVar)">Stop time</button>

<script>
let myVar = setInterval(myTimer, 1000);
function myTimer() {
  const d = new Date();
  document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
</script>
```

Figura 157 *– A variável devolvida de setInterval() (**Fonte:** https://www.tutorialrepublic.com/JavaScript-tutorial/JavaScript-timers.php)*

5.4. JavaScript Avançado

Dia e Hora em JavaScript

Note que o objeto de data em JavaScript é exibido, por definição, como um objeto global em todos os ambientes JavaScript. Desta forma, não é necessário importá-lo.

Os objetos de dados contêm vários métodos para obter, definir e manusear a exibição dos dias e das horas.

Criar Objetos de Data

Pode criar um novo objeto de data ao passar uma linha de representação de uma data ao construtor *Date*(), ou ao passar os componentes individuais de uma data como parâmetros para o construtor *Date*() (ano, mês, dia, hora, minuto, segundo, milissegundo).

Os seguintes exemplos criam objetos de data:

- // Criar um objeto de data para a data de hoje: const now = new Date(); //
- *Outputs* da data de hoje e o formato de tempo standard *console*.log(now); //
- *Outputs* do dia, data, mês e ano *console*.log(now.toDate*String*()); //
- *Outputs* da hora, minuto e segundo *console*.log(now.toTime*String*());
- *Outputs* da hora, minuto, segundo e milissegundo *console*.log(now.toLocale*String*());
- *Outputs* do ano *console*.log(now.getFullYear());
- *Outputs* do mês (de 0-11) *console*.log(now.getMonth());
- *Outputs* do dia (de 1-31) *console*.log(now.getDate());
- *Outputs* do dia da semana (de 0-6) *console*.log(now.getDay());
- *Outputs* de hora (de 0-23) *console*.log(now.getHours());

- *Outputs* de minuto (de 0-59) *console*.log(now.getMinutes());
- *Outputs* de segundo (de 0-59) *console*.log(now.getSeconds());
- *Outputs* de milissegundo (de 0-999) *console*.log(now.getMilliseconds());
- *Outputs* do número de milissegundos desse o início do dia *console*.log(now.getTime());
- *Outputs* do número de segundos desde 01.01.1970 *console*.log(now.getTime() / 1000);

Exemplos de *outputs*:

```
Thu, 22 Aug 2019 11:37:45 GMT
Thu Aug 22 2019 11:37:45 GMT+0000 (Coordinated Universal Time)
11:37:45 GMT+0000 (Coordinated Universal Time)
11:37:45 GMT+0000 (Coordinated Universal Time) 2019 8 22 4 11 37 45
Today is a Thursday, the 22nd of August, 2019.
```

Pode ver-se um exemplo prático aqui.

O construtor *Date*() também pode usar um valor inteiro que representa o número de milissegundos desde 01.01.1970, ou uma linha de representação de data.

Se passar um valor inteiro que representa o número de milissegundos desde 01.01.1970, o construtor irá criar um objeto de data que representa este dia e hora.

Se passar uma linha de representação de data, o construtor irá tentar analisar esta data e crirar um objeto de dado que representa este dia e hora. Se a linha não puder ser analisada, o construtor irá criar um objeto de data inválido.

Criar Objetos de Data com Moment.js

Moment.js é uma biblioteca de JavaScript que faz com que trabalhar com datas e horas seja mais fácil.

Para usar *Moment*.js num projeto, é necessário, primeiro, instalá-lo primeiro com um package manager como npm ou yarn, tal como se segue:

```
npm install moment yarn add moment
```

Após a instalação, podemos importar Moment.js para o ficheiro JavaScript com a seguinte expressão de importação:

```
import moment from 'moment';
```

No seguinte exemplo, usamos Moment.js para criar um objeto de data para o dia e hora de hoje, e depois usamos o método format() para *output* o dia e a hora nos diferentes formatos:

```
import moment from 'moment'; // Create a date object for today's date and time const now = moment(); // Outputs today's date and time in standard format console.log(now.format()); // Outputs the day, date, month and year console.log(now.format('dddd, MMMM D, YYYY')); // Outputs the hour, minute and second console.log(now.format('h:mm:ss a')); // Outputs the day, date, month, year, hour, minute and // second console.log(now.format('ddd, hA'));
```

Outputs:

```
2019-08-22T15:25:33+04:00 Thursday, August 22, 2019 3:25:33 pm Thu, 3PM
```

Aritemética de Data

Os objetos de data podem ser manipulados para mudar o dia e as horas. Os objetos de data têm o método set() que pode ser usado para definir o ano, mês, dia, hora, minuto, segundo e milissegundo de uma data.

Quando usarmos set(), o objeto tem de ser atualizado no local, portanto, não é necessário criar um novo objeto de data.

O seguinte exemplo usa o método set() para definir as horas e os minutos de uma data:

```
const now = new Date(); // Outputs the hour, minute and second
console.log(now.toLocaleString()); // Changes the hour to 12 and the minute to 45 now.setHours(12); now.setMinutes(45);
console.log(now.toLocaleString());
```

Outputs:

```
11:56:09 AM 12:45:09 PM
```

O método setHours() também pode ser usado para definir as horas e os minutos, e o método setMinutes() também pode ser usado para definir os minutos e os segundos.

O objeto de data também tem getters para definir o ano, mês, dia, hora, minuto, segundo e milissegundo de uma data.

O seguinte exemplo usa o método get() para definir as horas e os minutos de uma data:

```
const now = new Date(); // Outputs the hour, minute and second
console.log(now.toLocaleString()); // Outputs the hour and minute
console.log(now.getHours() + ':' + now.getMinutes());
```

Outputs:

```
11:56:09 AM 11:56
```

Quando usarmos o método set(), o objeto é atualizado no local, portanto não tem que criar um novo objeto de data.

O método setDate() pode ser usado para definir o dia de uma data. O método setMonth() pode ser usado para definir o mês de uma data. O método setFullYear() pode ser usado para definir o ano de uma data.

O método getDate() pode ser usado para obter o dia de uma da. O método getMonth() pode ser usado para obter o mês de uma data. O método getFullYear() pode ser usado para obter o ano de uma data.

Os seguintes exemplos usam os métodos set() , setDate() , setMonth() , setFullYear() , get() , getDate() , getMonth() and getFullYear() para definir e obter o ano, mês e dia de uma data:

```
const now = new Date(); now.setDate(1); // Sets the day to the first day of the month now.setMonth(0); // Sets the month to January now.setFullYear(2019); // Sets the year to 2019 console.log(now.toLocaleString()); now.setDate(now.getDate() + 10); // Adds 10
```

Operações Mataméticas em JavaScript

Pode realizar operações numéricas usando o JavaScript. Isto pode ser feito com a ajuda de operadores de + (adição), - (subtração), * (multiplicação), / (divisão) and % (percentagem).

No exemplo em baixo usamos o operador de adição para somar dois números:

```
var x = 10; var y = 5; var z = x + y;
```

Isto irá dar-nos o valor de z como 15.

Da mesma forma, podemos usar operadores de - (subtração), * (multiplicação), / (divisão) and % (percentagem) para realizarem as respetivas operações.

O operador de percentagem devolve o resto de uma operação de divisão. Por exemplo, se dividirmos 10 por 3, o resto será 1. Portanto, 10 % 3 irá dar o resultado de 1.

Para além destas operações aritméticas básicas, o JavaScript também oferece algumas funções matemáticas built-in. Estas funções podem ser usadas para realizar operações mais complexas.

Algumas das funções matemáticas mais usadas são:

- Math.abs(x) : Esta função devolve o valor absoluto de um número. Por exemplo, Math.abs(-10) irá devolver 10.
- Math.ceil(x) : Esta função arredonda para cima um número para o valor inteiro mais próximo. Por exemplo, Math.ceil(4.7) irá devolver 5.
- Math.floor(x) : Esta função arredonda para baixo um número para o valor inteiro mais próximo. Por exemplo, Math.floor(4.7) irá devolver 4.
- Math.max(x, y, ...) : Esta função devolve o máximo de argumentos passados para esta. Por exemplo, Math.max(10, 5, 20) irá devolver 20.
- Math.min(x, y, ...) : Esta função devolve o minimo de argumentos passados para esta. Por exemplo, Math.min(10, 5, 20) irá devolver 5.
- Math.pow(x, y) : Esta função devolve o valor de x elevado à potência de y. Por exemplo, Math.pow(2, 3) irá devolver 8.
- Math.random() : Esta função devolve um número aleatório entre 0 e 1.
- Math.sqrt(x) : Esta função devolve a raiz quadrada de x. Por exemplo, Math.sqrt(16) irá devolver 4.

Pode aprender mais sobre o objeto matemático e as suas propriedades e métodos através da referência matemática do JavaScript.

Tipo de Conversões em JavaScript

É possível converter um valor para um tipo de dados específico usando métodos built-in, como *parseInt*() para converter um valor para um número interno, ou *parseFloat*() para converter um valor num *float*..

É também possível usar o método *Number*() para converter um valor num número, ou o método *Booleano*() para converter um valor num *booleano*.

No seguinte exemplo iremos converter uma linha num número usando o método Number():

```
var x = "100"; var y = Number(x); console.log(y); // 100
```

Neste exemplo temos uma linha com o valor “100”. Convertemos esta linha num número usando o método Number(), e depois imprimimos o valor do número no *console*.

Pode usar o operador *unary* + para converter um valor num número. Por exemplo:

```
var x = "100"; var y = +x; console.log(y); // 100
```

Neste exemplo nós temos uma linha com o valor de “100”. Usamos o operador *unary* + para converter esta linha num número, e depois imprimimos o valor do número no *console*.

Pode usar o método *parseInt*() para converter uma linha num número inteiro, ou o método *parseFloat*() para converter uma linha num *float*.

Por exemplo:

```
var x = "100.50"; var y = parseInt(x); console.log(y); // 100 var z = parseFloat(x); console.log(z); // 100.5
```

Neste exemplo nós temos uma linha com o valor de “100.50”. Usamos o método *parseInt*() para converter uma linha num número inteiro, e depois imprimimos o valor do número inteiro no *console*.

Também usamos o método *parseFloat*() para converter uma linha num *float*, e depois imprimimos o valor do *float* no *console*.

Pode usar o método *Booleano*() para converter um valor num *booleano*.

Por exemplo:

```
var x = "100"; var y = Booleano(x); console.log(y); // true
```

Neste exemplo temos uma linha com o valor de “100”. Convertemos a linha num *booleano* usando o método *Booleano*(), e depois imprimimos o valor do *booleano* na consola.

Pode usar-se o operador **!!** para converter um valor num *booleano*. Por exemplo:

```
var x = "100"; var y = !!x; console.log(y); // true
```

Neste exemplo temos uma linha com o valor “100”. Usamos o operador !! para converter um valor num *booleano* e depois imprimimos o valor do *booleano* no *console*.

Se quiser converter um valor numa linha, pode usar o método to*String*().

Por exemplo:

```
var x = 100; var y = x.toString(); console.log(y); // "100"
```

Neste exemplo temos um número com o valor de “100”. Convertemos o número numa linha usando o método to*String*() e depois imprimimos a linha no *console*.

Se quiser converter um valor numa matriz, pode usar o método *Array.from().*

Por exemplo:

```
var x = "100"; var y = Array.from(x); console.log(y); // [ "1", "0",
"0" ]
```

Neste exemplo temos uma linha com o valor de “100”. Convertemos a linha numa matriz usando o método *Array.from*() e depois imprimimos a matriz no *console*.

Se quiser converter um valor num objeto, pode usar o método *Object*().

Por exemplo:

```
var x = "100"; var y = Object(x); console.log(y); // { "0": "1", "1":
"0", "2": "0" }
```

Neste exemplo temos uma linha com o valor de “100”. Convertemos a linha num objeto usando o método *Object*() e depois imprimimos o objeto no *console*.

Listagem de Eventos DOM

As listagens de eventos JavaScript permite definir funções que irão correr quando ocorrer um evento em específico num elemento do DOM.

Existe um número de eventos diferentes que podem ocorrer num elemento DOM, quando, por exemplo, o elemento é carregado, passado por cima ou até mesmo quando o conteúdo dos elementos for alterado.

Para adicionar uma listagem de eventos num elemento, é necessário, primeiro, selecionar o elemento usando um dos métodos de seleção DOM, como *document.querySelector*() *ou document.getElementById*() .

Uma vez que o elemento é selecionado, pode usar o método *addEventListener*() para anexar uma listagem de eventos no elemento.

O método *addEventListener*() inclui dois argumentos: o nome dos eventos a ouvir e uma função que corra quando ocorre o evento.

Por exemplo, para adicionar um clique a uma listagem de eventos, é necessário usar o seguinte código:

```
button . addEventListener ( "click" , function ( ) { console . log ( "The button was clicked!" ) ; } ) ;
```

No código acima, quando o elemento botão é carregado, a função é passada para que o método *addEventListener*() seja executado.

Pode também passar uma função com nome para o método *addEventListener*(), em vez de uma função anónima:

```
function handleClick ( ) { console . log ( "The button was clicked!" ) ; } button . addEventListener ( "click" , handleClick ) ;
```

Tipos de Eventos DOM

Existe um número de diferentes eventos que podem ocorrer num elemento DOM.

Os eventos mais comuns são:

- click
- mouseover
- mouseout
- keypress
- change
- submit

É possível encontrar uma lista completa de eventos DOM no Mozilla Developer Network.

Objeto de Evento DOM

Quando uma função de listagem de eventos é chamada, esta passa um objeto de evento como um argumento.

O objeto de evento contem informação sobre o evento que ocorreu, como o tipo de evento, o elemento onde ocorreu o evento, e algum dado específico do evento.

Por exemplo, o objeto do evento click contem informação sobre o evento *click*, como, por exemplo, as coordenadas x e y do clique.

Por exemplo, para introduzir as coordenadas x e y de um evento *click*, o seguinte código seria usado:

```
button . addEventListener ( "click" , function ( ) { console . log ( "x: " + this . clientX + ", y: " + this . clientY ) ; } ) ;
```

No código acima, a palavra "*this*" refere-se ao objeto do evento, e as propriedades clientX e clientY são usados para aceder às coordenadas x e y do evento *click*.

Delegação de Eventos DOM

A delegação de eventos DOM é uma técnica para anexar listagens de eventos a elementos que ainda não existem no DOM.

Isto é útil quando temos um grande número de elementos que queremos adicionar às listagens de eventos, mas não quer adicionar uma listagem de eventos a cada elemento individualmente.

Por exemplo, se tem uma list com 100 itens, e quiser adicionar uma listagem do evento *click* a cada um deles, poderá uma delegação de eventos para adicionar uma única listagem de eventos *click* ao elemento parente da lista, e ter aquela função de listagem responsável pelos eventos de todos os elementos filho.

Para usar uma delegação de eventos, é primeiro necessário selecionar um elemento parente dos elementos que quer adicionar aos *event listeners*.

Pode, então, usar o método *addEventListener*() para anexar a listagem de eventos a um elemento parente.

O método *addEventListener*() tem dois argumentos: o nome do evento a listar e a função a correr quando o evento ocorre.

A função passada ao método *addEventListener*() será executada quando o evento ocorrer em qualquer elemento filho do elemento parente.

Por exemplo, para adicionar uma lis*tag*em de evento *click* a todas as listas de itens numa lista, terá de usar o seguinte código:

```
var listItems = document . querySelectorAll ( "li" ) ; listItems . forEach ( function ( listItem ) { listItem . addEventListener ( "click" , function ( ) { console . log ( "The list item was clicked!" ) ; } ) ; } ) ;
```

No código acima, uma listagem de evento *click* é adicionado a cada lista de itens individualmente.

Isto pode não se eficiente se houver um largo número de itens de lista.

Em vez disso, pode usar a delegação de eventos para adicionar listagens de eventos *click* individuais ao elemento parente dos itens de lista:

```
var list = document . querySelector ( "ul" ) ; list . addEventListener ( "click" , function ( event ) { if ( event . target . tagName === "LI" ) { console . log ( "The list item was clicked!" ) ; } } ) ;
```

No código acima, a listagem de eventos *click* é adicionada ao elemento parente dos itens lista.

A função passada ao método *addEventListener*() verifica a propriedade *Name* de um elemento que foi carregado de forma a ver se é um elemento <*li*>.

Se for um elemento <*li*>, então a função introduz uma mensagem no *console*.

Esta técnica pode ser usada para adicionar lis*tag*ens de eventos em qualquer tipo de elemento, e não apenas os elementos listados.

Por exemplo, pode usar a delegação de eventos para adicionar uma listagem de eventos *click* a todos os botões num documento:

```
var buttons = document . querySelectorAll ( "button" ) ; buttons . forEach ( function ( button ) { button . addEventListener ( "click" , function ( ) { console . log ( "The button was clicked!" ) ; } ) ; } ) ;
```

Ou pode usar uma delegação de evento para adicionar uma listagem de eventos *click* a todos os *links* no documento:

```
var links = document . querySelectorAll ( "a" ) ; links . forEach ( function ( link ) { link . addEventListener ( "click" , function ( ) { console . log ( "The link was clicked!" ) ; } ) ; } ) ;
```

Resumo da Delegação de Eventos

Neste tutorial, foram dadas noções acerca das listagens de eventos DOM em JavaScript. Aprendemos como usar o método *addEventListener*() para anexar as listagens de eventos nos elementos DOM. Aprendemos também sobre os diferentes eventos que podem ocorrer num elemento DOM e como aceder ao objeto do evento de dentro de uma função de listagem de eventos. Por fim, aprendeu sobre a delegação de eventos DOM e como o usar para anexar listagens de eventos a elementos que ainda não existem em DOM.

Propagação de Eventos em JavaScript

Quando um evento ocorre num elemento, o evento pode ser reconhecido pelo JavaScript e agirem de acordo com tal. Por exemplo, se carregar num botão numa página *web*, pode programar o JavaScript para reconhecer o evento e agir.

O evento que ocorreu foi um evento *click*.

Quando carrega num elemento, o evento *click* é feito no elemento que foi carregado.

O evento propaga, então, na árvore DOM.

Isto significa que, se houver um manipulador num elemento parente, esse manipulador de evento será executado.

Se houver um manipulador de eventos num elemento avô, esse manipulador de eventos será executado.

A propagação nas árvores DOM continua até o evento atingir o elemento de raíz da árvore DOM ou até o evento parar.

Quando o evento atinge o elemento raiz, o evento terá subido completamente a árvore DOM.

O evento irá, então, propagar, desta mesma forma, até à base da árvore DOM, do elemento raiz ao elemento que foi carregado.

Este evento chama-se ***bubbling***. O seguinte diagrama mostra como funciona o evento *bubbling*.

Pode manipular este evento em qualquer elemento na árvore DOM.

Pode, também, parar a propagação do evento.

Por exemplo, se tiver um manipulador de evento num elemento parente, e não quiser que esse manipulador de eventos seja executado quando o evento ocorre num elemento filho.

Neste caso, pode parar a propagação do evento na árvore DOM.

A isto chamamos propagação de evento.

O evento não atinge o elemento raíz e não propaga de volta à base da árvore DOM.

Pode também parar a propagação de um evento a qualquer ponto da árvore DOM.

Por exemplo, pode ter um manipulador de evento num elemento avô, e não quer que o manipulador de evento seja executado quando o evento ocorrer num elemento neto.

Neste caso, pode parar o evento de propagar para a base da árvore DOM. A isto chama-se captura de evento.

Resumindo, quando um evento ocorre num elemento, esse evento pode ser reconhecido pelo JavaScript e realizado.

O evento propaga acima da árvore DOM, e depois desce na árvore DOM.

Pode manipular eventos em qualquer elemento da árvore DOM.

Pode também parar a propagação de um evento.

Empréstimo de Métodos em JavaScript

Usaremos o exemplo de um objeto que contem uma propriedade que precisa de emprestar.

```
var myObject = {

someProperty: “foo”,

};
```

Podemos emprestar a propriedade someProperty de myObject desta forma:

```
var myProperty = myObject.someProperty;

myProperty will now contain the value “foo”.
```

Podemos também usar a mesma síntaxe para emprestar métodos de objetos.

Por exemplo, se tivemos um objeto com um método que queremos emprestar:

```
var myObject = {

someMethod: function() {

// do something

}

};
```

Podemos emprestar o método *someMethod* assim:

```
var myMethod = myObject.someMethod;

We can now call myMethod like a normal function.

 myMethod();
```

Hoisting em JavaScript

Hoisting é um comportamento único do JavaScript. É um conceito que é, muitas vezes, incompreendido. Muitos programadores acreditam que o JavaScript faz *hoist* de expressões de variáveis de funções para o topo do escopo.

Isto não é de todo correto. O que o JavaScript faz, na verdade, é mover as expressãos para o topo do escopo e não a inicialização. Isto pode levar a alguns comportamentos inesperados.

Considere o seguinte código:

```
var foo = 1; function bar() { if (!foo) { var foo = 10; } console.log(foo); } bar();
```

Qual é que acha que vai ser o resultado deste código?

Se adivinhou 10, está errado. O resultado deste código é 1.
Isto deve-se à declaração de foo estar em hoist no topo do escopo, mas a inicialização não. Quando a expressão for executada, foo é indefinido e é, assim, um conjunto de 10.

Isto pode ser um pouco confuso, mas é importante entender como funciona o *hoisting* em JavaScript. Pode ajudar a evitar alguns erros comuns.

Encerramentos em JavaScript

Os encerramentos em JavaScript são uma função interior que tem acessi às variáveis exteriores (encerramento) da cadeia de escopo de funções. O encerramento tem três cadeias de escopos: tem acesso ao seu próprio escopo (as variáveis definidas entre as chavetas), tem acesso às funções exteriores das variáveis e tem acesso às variáveis globais.

Os encerramentos de JavaScript são criados quando a função interior é feita dentro da função exterior. Os encerramentos são usados com bastante frequência nas bibliotecas de JavaScript como jQuery.

Aqui está um simples exemplo de um encerramento em JavaScript:

```
function outerFunction(x) {
  var innerFunction = function(y) {
    return x + y;
  }
  return innerFunction;
}

var add5 = outerFunction(5);
var add10 = outerFunction(10);

console.log(add5(2)); // 7
console.log(add10(2)); // 12
```

No exemplo acima, temos a função outerFunction que tem um único parâmetro x. Esta função contem uma função *innerFunction* que tem um único parâmetro y. A *innerFunction* retoma com a soma de x e y.

Quando chamamos o *outerFunction* com um valor, este retoma o *innerFunction*. Quando chamamos a função que foi devolvida pelo *outerFunction* com um valor, retoma a soma do passamos ao *outerFunction* com o valor que passamos com o *innerFunction*.

Neste exemplo acima temos dois encerramentos: add5 e add10. Quando chamamos add5(2), retoma 7, pois o 5 é passado para o *outerFunction* como o valor de x, e o 2 é passado para o *innerFunction* como o valor de y. Quando invocamos add10(2), retoma 12, pois o 10 é passado para o *outerFunction* como o valor de x, e o 2 é passado para o *outerFunction* como o valor de y.

O encerramento é uma função que define o uso de variáveis nas funções exteriores que tenham previamente sido devolvidas. No seguinte exemplo, a função interior *plus*() está a beneficiar a variável num que foi definida na função exterior *returnNotification*():

```
function returnNotification() {
  var num = 42;

  function plus() {
    return num + 1;
  }

  return plus;
}

var result = returnNotification();
```

```
console.log(result()); // 43
```

No exemplo acima, temos a função *returnNotification*(). Esta função tem uma variável local num e uma função *plus*(). A função *plus*() devolver o valor de num com 1. A função *returnNotification*() retoma como a função *plus*().

Atribuímos o valor de retoma da função *returnNotification*() ao resultado da variável. Quando chamamos *result*(), este chama a função *plus*() e retoma o valor de *num* com 1.

O encerramento é uma função que usa as variáveis definifas nas funções exteriores que tenham retomado previamente. No seguinte exemplo, a função interna *plus*() usa a variável num, que foi definida na função exterior *returnNotification*():

```
function returnNotification() {
  var num = 42;

  function plus() {
    return num + 1;
  }

  return plus;
}



var result = returnNotification();

console.log(result()); // 43
```

No exemplo acima temos a função *returnNotificaton*(). Esta função tem uma variável local *num* e a função *plus*(). A função *plus*() retoma o valor de num com 1. A função *returnNotificação*() retoma a função *plus*().
Atribuímos o valor de retoma na função *returnNotification*() ao resultado da variável. Quando chamamos o *result*(), este chama a função *plus*() e retoma o valor de num mais 1.

Os encerramentos são por norma usados nas librarias JavaScript como jQuery. No seguinte exemplo, usamos a libraria jQuery para criar um botão. Quando o botão é pressionado, é mostrada uma caixa com o texto “Hello world!”:

```
$(function() {
  var button = $('button');
  button.click(function() {
    alert('Hello world!');
  });
});
```

No exemplo acima, temos um elemento de botão com o id “button”. Usamos a biblioteca jQuery para selecionar o elemento de botão. Usamos o método *click*() para registar a função que irá ser executada quando o botão é carregado.

A função que é executada quando o botão é carregada é um encerramento. Tem acesso a variáveis na função exterior, que, neste caso, é o elemento de botão. Quando a função é executada, mostra uma caixa de alerta com o texto “Hello world!”.

Modo Rrestrito em JavaScript

O que é o modo rrestrito em JavaScript?

O modo rrestrito é uma forma de optar por uma variante restrita do JavaScript. O modo rrestrito não é apenas um subconjunto: tem, intencionalmente, diferentes semânticas do código normal. Os navegadores não são obrigados a correr código JS em modo restrito, e muitos deles não o fazem.
O modo restrito torna mais fácil escrever JavaScript “seguro”. O modo rrestrito transforma o que antes era aceite como “má síntaxe” em erros reais.

SintaxError: Avaliação ou argumentos inesperados em modo rrestrito.

O modo rrestrito resolve, também, alguns erros que, em silêncio, falharam em JavaScript, com exceções.

- Apagar uma variável ou função.
- Usar uma variável que não está declarada.
- Apagar uma propriedade de um objeto.
- Tentar mudar uma propriedade de leitura.
- Adicionar uma propriedade a um objeto que não é extendível.
- Usar um objeto que foi congelado ou selado.
- Escrever em propriedades de leitura.
- Tentar mudar o atributo DontDelete numa propriedade.
- Designar um valor para uma variável global de leitura.
- Aumentar ou diminuir uma propriedade de leitura.
- Tentar alterar o atributo DontDelete numa função.
- Designar um valor a uma função global de leitura.
- Tentar mudar o protótipo de uma função.

O modo restrito também evita, ou retira erros, quando são tomadas ações “inseguras”.

- Usar uma variável sem a declarar.
- Criar uma variável global (sem usar a expressão *var*).

- Apagar uma variável ou função.
- Apagar a propriedade de um objeto.
- Usar um objeto que foi congelado ou selado.
- Tentar alterar um atributo *DontDelete* numa propriedade.
- Atribuir um valor a uma função global de leitura.
- Aumentar ou diminuir uma propriedade de leitura.
- Tentar mudar o protótipo de uma função.

De forma a correr o código em modo restrito, é necessário adicionar “use strict”; diretamente no início do código. Esta diretiva não é uma linha, mas sim uma expressão literal que aparece no início no corpo de um ficheiro, *script* ou função.

```
"use strict";
```

A diretiva pode ser colocada ou no início de um *script* ou no início de uma função.

Exemplo 1: Colocar a diretiva no início do *script*.

```
"use strict"; x = 3.14; // This will cause an error because x is not declared.
```

Exemplo 2: Colocar a diretiva no início de uma função.

```
function myFunction() { "use strict"; y = 3.14; // This will cause an error because y is not declared. }
```

Se o modo restrito for usado dentro de uma função, o modo restrito será apenas aplicado na função. Se o modo restrito for usado no nível superior de um *script*, o modo restrito será aplicado ao *script* completo.

Se o modo restrito for usado no nível superior do *script*, o modo restrito será aplicado no *script* todo.

Se o modo restrito for usado dentro de uma função, o modo restrito irá apenas ser aplicado na função.

No modo restrito, qualquer referência a uma variável não declarada irá causar um erro de referência.

```
function myFunction() { "use strict"; x = 3.14; // This will cause an error because x is not declared. }
```

No modo restrito, qualquer atribuição a uma variável global não escrita, a um objeto de um elemento de um argumento não escrito, ou uma propriedade não escrita, irá resultar em erro.

Analisar JSON em JavaScript

O que é JSON?

JSON significa Notação do Objeto em JavaScript. É uma troca ligeira de formatos de dados que permitem mudar e armazenar os dados de uma forma simples e organizada. A melhor coisa sobre isto é que a síntaxe assemelha-se bastante ao inglês, pelo que, mesmo que não saiba nada sobre programação ou código, existem ainda hipóteses de entender o que é que este código significa.

Analisar Dados JSON em JavaScript

O seguinte exemplo mostra como analizar uma linha e criar um objeto a partir dela. Deve sempre optar por analisar os dados como está exemplificado em baixo:

```
var jsonString = '{"name":"John","age":30,"city":"New York"}'; var obj = JSON.parse(jsonString); console.log('Name: ',obj['name'],' Age:',obj['age'], ' City:' , obj ['city'] ); // Outputs Name : John, age : 30 and city as New york
```

Análise de Dados Nested JSON em JavaScript

O seguinte exemplo mostra como analisar dados *nested* de uma linha em objeto usando a função *eval*() (o que não é recomendado). Pode também usar o mesmo método para analisar objetos *nested*, mas é necessário ter precaução enquanto o faz, pois se existem alguns erros, irá exibir exceções que podem fazer com que o programa vá abaixo!

Codificar dados como JSON em JavaScript

O seguinte exemplo explica como codificar dados numa linha usando a função JSON.*string*ify():

```
var obj = { name : 'John', age: 30, city:'New York' }; var jsonString = JSON.stringify(obj); console.log('JSON String is ',jsonString ); // Outputs the stringified object as a JSON
```

Lidar com Erros em JavaScript

JavaScript é uma linguagem flexível, o que significa que existem várias maneiras de escrever código. Esta flexibilidade pode levar a erros no programa se não houver cuidado sobre a sua estruturação.

Um dos erros mais frequentes que os programadores cometem é assumir que todo o código corre corretamente. Esta suposição pode ser prejudicial porque o JavaScript é uma linguagem interpretada, o que significa que cada linha de código é corrida no momento em que está a ser lida pelo intérprete. Se existe um erro no código, o intérprete irá parar de funcionar naquele ponto e irá mostrar uma mensagem de erro.

Este tutorial irá mostrar-lhe como lidar com erros de uma forma correta para que o seu programa possa continuar a funcionar mesmo que haja problemas com a introdução dos dados e outras partes do código.

A primeira coisa que precisa de ser feita é identificar onde é que os potenciais erros podem acontecer. Por exemplo, se está a ler os dados de um ficheiro, existe uma probabilidade do ficheiro não existir ou estar corrompido. Se estiver a trabalhar com input do utilizador, existe a hipótese de que o utilizador tenha inserido dados inválidos. Uma vez que tenha identificado estas potenciais fontes de erros, pode começar a escrever código de forma a conseguir lidar come eles.

Uma forma usual de lidar com os erros é usar os blocos *try/catch*. Estes blocos permitir-lhe-ão "tentar" algum bocado de código e "apanhar" alguns erros que possam ocorrer. A síntaxe para um bloco *try/catch* é assim:

```
try { // Code to try goes here } catch (error) { // Code to handle errors goes here }
```

O código dentro do bloco "*try*" irá ser realizado primeiro. Se não ocorrerem erros, o código no bloco "catch" nunca será corrido. No entanto, se houver um erro, a execução do código irá diretamente para o bloco "*catch*" e qualquer outro código no bloco "*try*" será ultrapassado.

Vejamos um exemplo de como funciona. Imagine que tem uma função que lê dados de um ficheiro e os imprime no *console*. No entanto, existe a possibilidade de o ficheiro não existir:

```
function readFile(filename) { try { // Code to read and print the contents of "filename" goes here } catch (error) { console.log("Error reading file: " + error); } }
```

Neste exemplo, a função *readFile* irá tentar ler e imprimir os conteúdos num ficheiro. Se ocorrer um erro aquando da leitura do ficheiro, será encontrado pelo bloco "*catch*" e impresso no *console*.

É importante saber que apenas os erros que ocorrem dentro do bloco "*try*" é que podem der descobertos pelo bloco "*catch*". Se houver um erro no bloco "*catch*" (por exemplo,

esquece-se de fechar os parênteses curvos), não irá apanhar erros e o seu programa irá abaixo.

Outra forma de lidar com erros é usar o objeto Error *built-in* do JavaScript. O objeto *Error* pode ser usado para criar mensagens de erro personalizadas que podem ser passadas para expressãos:

```
throw new Error("This is a custom error message");
```

Quando este código é executado, um erro será mostrado e a execução da função atual irá parar. Qualquer código subsequente na função não será corrido.

Também pode usar o objeto *Error* para lidar com erros que ocorram dentro do código assíncrono. Por exemplo, imagine que tem uma função que faz um pedido *http* e devolve os seguintes dados:

```
function makeRequest(url) { return new Promise((resolve, reject) => { http.get(url, (response) => { // Code to handle the response goes here }); }); }
```

Esta função devolve uma garantia, que significa que os dados devolvidos pelos pedidos HTTP estarão disponíveis em alguma altura no futuro. Se ocorrer algum erro enquanto se fazem os pedidos (por exemplo, se um URL está inválido), será apanhado pelo manipulador de erros *built-in* do JavaScript e rejeitado com uma mensagem de erro.

Esta rejeição irá causar a garantia de retorno como erro, o que pode ser depois manipulado pelo código que chamou *makeRequest*:

```
makeRequest("http://www.example.com") .then((data) => { // Code to handle the data goes here }) .catch((error) => { console.log("Error making request: " + error); });
```

Neste exemplo, registamos uma função *callback* para o evento "*then*" que pode ser chamado se a premissa for resolvida corretamente. Também foi registado uma função *callback* para o evento "*catch*" que será chamado caso houver algum erro. Isto permite-nos lidar com erros de uma forma positiva e continuar e correr o código mesmo que haja problemas com o *input* dos dados ou outras partes do código.

Expressões Regulares em JavaScript

As expressões regulares são ferramentas eficazes usadas para criar padrões de combinação e funções "*search-and-replace*" no texto.

A expressão regular é, basicamente, uma sequência de caracteres que definem um padrão de busca particular. Por exemplo, o seguinte *regex* iria emparelhar qualquer linha que contenha um A maiúsculo: /A/.

As expressões regulares podem ser usadas para realizar uma variedade de tarefas, como:

- Extraír informação de uma linha atribuída (p.e.: endereço de email);
- Validar input do utilizador (por exemplo: garantir que a palavra-passe é suficientemente forte);
- Procurar e substituir texto numa linha atribuída (por exemplo: alterar todas as ocorrências de "a" para "b");
- Formatar texto (por exemplo: adicionar vírgulas entre as palavras);
- Gerar linhas aleatórias (por exemplo: criar palavras-passe e ID únicos);
- E muito mais.

Em JavaScript, as expressões regulares são também objetos. Estes objetos contêm um conjunto de métodos que podem ser usados para correr várias operações em linhas, como procurar por padrões ou substituir texto.

Criar Expressões Regulares em JavaScript

Existem duas formas de criar expressões regulares em JavaScript: usar uma expressão regular literal ou a função de construtor *RegExp*().

Uma expressão regular literal é, simplesmente, uma linha que contem um padrão que o programador deseja encontrar, fechado por duas barras (/). Por exemplo:

```
const regex = /abc/ ; console .log(regex); // => /abc/
```

Os códigos acima criam um novo objeto de expressões regulares que contêm um padrão abc e atribui-o a uma variável regex. O valor desta variável pode ser usada como qualquer objeto JavaScript. Em particular, podemos usar métodos para realizar várias operações em linhas.

A função de construtor *RegExp*() também pode ser usada para criar expressões regulares. Esta função requer dois argumentos: o primeiro é uma linha que contem o padrão que quer combinar, e o segundo é um argumento "*flags*" opcional que específica certas definições de como o regex deve trabalhar. Por exemplo:

```
const regex = new RegExp ( 'abc' ); console .log(regex); // => /abc/ const flagsRegex = new RegExp ( '123' , 'i' ); console .log(flagsRegex); // => /123/i; i stands for case-insensitive matching
```

O código acima cria dois novos objetos de expressões regulares. O primeiro contem o padrão abc e é sensível a letras maiúsculas e minúsculas (pré-definição). O segundo objeto tem um padrão de 123, mas é combinável e será sensível a letras maiúsculas e minúsculas, devido à "*flag*" i que passou como segundo argumento para *RegExp*().

Nota: Em JavaScript, os regexes criado usando as notações literais são imutáveis; estas não podem ter as propriedades e os métodos alterados depois de serem

definidos. Por outro lado, os *regexes* criados com *RegExp*() podem ser modificados em qualquer altura porque são apenas outra função de construtor como Data, Matriz, etc.

Formulário de Validação em JavaScript

O JavaScript é uma linguagem de *script* da visão do cliente, o que significa que o *script* corre no computador do visitante. Isto permite que se façam coisas como verificar se o endereço de email foi inserido corretamente antes de ser enviado para o servidor. Reduz também a tensão no servidor porque os dados dos formulários não têm que ser submetidos até depois de terem sido verificados os seus erros pelo JavaScript.

A primeira coisa a fazer é criar um formulário. Pode usar o código seguinte:

```
<form action="yourpage.php" method="post"> <p>Name:</p><input type="text" name="name"><br /> <p>Email Address:</p><input type="text" name= "email"><br /> </form>
```

Isto irá criar um formulário básico com dois campos, um para o nome e outro para o endereço de email. O atributo de ação diz ao navegador onde enviar os dados quando submetidos, neste caso *yourpage.php*. Pode mudar isto para a página onde quer que os dados do formulário sejam processados, O método attribure específica como é que os dados devem ser enviados, seja por GET ou POST. Na maioria dos casos deverá usar POST porque é mais seguro que GET, mas existem situações onde usar GET poderá ser mais indicado (por exemplo, se quiser que as pessoas sejam capazes de marcar resultados de pesquisa). Para mais informações sobre estes métodos, veja os nossos Tutoriais de Formulários HTML: artigo de introdução que os explica detalhadamente.

A próxima coisa a fazer é adicionar um botão de submissão para que os visitantes possam enviar os seus detalhes:

```
<input type="submit" value="Submit">
```

Pode pôr este botão em qualquer lado dentro das *tag*s de formulário. Agora, se guardar a sua página e a experimentar, quando carregar no botão de submeter, irá abrir uma nova janela com yourpage.php nela (assumindo que é isto que está definido na sua ação). Isto acontece porque não dissemos ao navegador para fazer outra coisa diferente ainda – neste momento, todos que submitem os formulários enviam-nos para outra página, o que não é útil.

Precisamos que o JavaScript verifique que não há erros antes de enviar algum dado, portanto, vamos acrescentar algum código:

```
<script type="text/JavaScript"> function checkForm() { var errorMsg = ""; // Check each field to make sure it has a value. if (document.form1.name == "") { errorMsg += "- Please enter your name

"; } else if (document.form1 .email == "" || document .form 1 .email_confirm != document form 1 email)  {errorMsg += "- You must provide an e-mail address and confirm it by entering the same address again."; } return true; // This line submits the form only when there are no errors in any of the fields, otherwise we display our message telling them what they need to do: alert(errormsg); return false ; </script> <body onload="checkForm();" >

 ... rest of page here...

</body>
```

Cookies em JavaScript

Criar *cookies* em JavaScript é bastante simples. É apenas necessário usar o objeto do documento e o seu método *createCookie*(). A síntaxe desta função parece-se com o seguinte:

```
document.cookie = "name=value; expires=date";
```

O parâmetro do nome representa o nome da cookie, enquanto que o valor significa o seu conteúdo (linha). Se quiser estabelecer uma data de validade, adicione outro atributo chamado expires com uma linha válida de Date() como valor, ou passe, simplesmente, para nulo, caso não precise dele.

```
document.cookie="username=John Doe;expires"+Date(30); //creates username cookies that will expire after 30 days from now
document.cookie="username=John Doe;expires"+Date(30*24); //creates username cookies that will expire after 30 days from now
```

Ler uma *cookie* em JavaScript é bastante simples, também. Apenas tem que usar o objeto documento e o método *getCookie*(). A síntaxe parece-se com isto:

```
var name = getCookie("name");
```

O único parâmetro representa o nome da cookie (linha). Esta função retoma nula se não existir essa *cookie* ou se já tiver expirado. Também pode ler todas as *cookies* existentes usando outra propriedade built-in chamado *document.cookie* que contém uma matriz com todas as que estão disponíveis na página separadas por “,”. Por exemplo:

```
alert(document.cookies) ;//will display something like : userName1=value1;userName2=value2 etc... /*this code displays nothing because there are no any active cookiess*/
```

Atualizar uma cookie em JavaScript é bastante simples também. Terá apenas que criá-la novamente com com o memso nome e o mesmo valor, mas desta vez é também necessário estabelecer uma data de validade que é maior que a data atual (se houver).

Por exemplo:

```
document.cookie = "name=newValue; expires=" + Date(Date().getTime()+1); /*this code will update your cookies*/
document.cookie = "username=John Doe;expires"+Date(30*24); //creates username cookies that will expire after 30 days from now /*this code creates new cookies if there are no active ones or updates existing ones */
```

Requisitos de Ajax em JavaScript

Ajax é uma técnica de desenvolvimento para criar aplicações *web* interativas. O objetivo do Ajax é fazer páginas *web* sentirem-se com maior resposta ao trocar pequenas quanridades de dados com o servidor nos bastidores, para que a página completa não tenha que ser recarregada cada vez que o utilizador faz alguma alteração.

De forma a usar Ajax, é necessário usar a biblioteca JavaScript como jQuery ou Prototype. Estas bibliotecas dão uma forma simplificada de enviar e receber dados de um servidor sem ter que recarregar a página.

Uma vez que tenha incluído a biblioteca Ajax na sua página *web*, pode começar a fazer requisitos ao servidor. Por exemplo, se quisesse carregar alguns dados de um ficheiro num servidor, deve usar o seguinte código:

```
$.ajax({ url: 'data.json', success: function(data) { // do something with the data } });
```

Este código fará um pedido a Ajax para o URL *'data.json'*. Se este pedido for bem-sucedido, a função *callback* em "*success*" será corrida com os dados do servidor como argumento.

Se quiser enviar dados ao servidor, pode usar a opção 'data' do método '$.ajax()':

```
$.ajax({ url: 'saveData.php', type: 'POST', data: { name: 'John', age: 20 } });
```

Este código irá fazer um requisito POST ao URL 'saveData.php'. Os dados que estão a ser enviados para o servidor é específicado na opção 'data' como um objeto. Neste exemplo, estamos a enviar dois pedaços de dados: 'name' e 'age'.

Os seguintes exemplos irão mostrar como fazer um requisito AjaxGET em JavaScript:

```
$.ajax({ url: 'getData.php', type: 'GET', success: function(data) { // do something with the data } });
```