CSS Cascading Style Sheets

Última atualização do doc. 06.Nov.2021

> *Para falar/escrever bom português não é necessário saber o dicionário todo...*
> *Com as CSS é exatamente o mesmo. Não é necessário conhecer todas as propriedades existentes (nem ajuda à aprendizagem!).*

> *CSS reference: MDN Web Docs*

Sintaxe

```
h1 {
    color: #00F;
}
```

Explicação da regra acima:

```
seletor {                   |
    propriedade: valor;     | Regra
}                           |
    |___Declaração____|
```

Seletores

Em CSS, os seletores são padrões usados para selecionar o(s) elemento(s) aos quais queremos aplicar estilos.

Seletor (exemplo)	Nome	Descrição
`*`	Universal Selector	Seleciona todos os elementos
`#titulo`	ID Selector	Seleciona um elemento com determinado id
`.filme`	Class Selector	Seleciona todos os elementos com determinada classe
`h1`	Type Selector	Seleciona elementos com base no tipo de elemento
`h1 + p`	Adjacent Sibling Selector	Seleciona todos os elementos referentes ao 2º elemento que estejam junto do 1º e ambos partilham o mesmo pai. Isto significa que o `p` está exatamente depois de `h1`
`ul > li`	Child Selector	Seleciona todos os elementos referentes ao 2º elemento que são filhos do 1º
`h1 ~ p`	General Sibling Selector	Seleciona todos os elementos referentes ao 2º elemento que estão depois do 1º (no exemplo, seleciona todos os elementos `p` que estão depois do `h1`, com o mesmo pai partilhado)
`p a`	Descendant Selector	Seleciona todos os elementos referentes ao 2º elemento que sejam descendentes do 1º (no exemplo, são selecionados todos os elementos `a` que estão dentro de um `p`)
`div[umqqatributo="umqqvalor"]`	Attribute Selector	Seleciona os elementos com um determinado atributo a que lhe corresponde um dado valor

Cascade

Quem escreve CSS muitas das vezes não vê um determinado estilo CSS aplicado a um determinado elemento. Isso deve-se ao efeito em cascata (**Cascading**).

> *A CSS Cascade é um algoritmo utilizado pelos browsers para a resolução de conflitos na escolha de qual o estilo deve aplicar num elemento, para o qual existem duas ou mais regras, com a mesma propriedade, mas com valores diferentes.*

Perceber o *cascade algorithm* ajuda a perceber como o browser resolve os conflitos.
Este algoritmo divide-se em 4 fases distintas:

1. Importância
2. Fonte/Origem
3. Especificidade do seletor
4. Ordem/Posição

Exemplos:

1. **Importância**
   A primeira coisa que os browsers fazem quando encontram declarações em conflito é a importância da declaração. É verificado se alguma das declarações está declarada com um `!important`, ou é uma regra de animação com `@keyframes`.

```
<div>
  <button>SIR</button>
</div>
```

```
button {
  background-color: red !important;
}
button {
  background-color: blue;
}
```

> *Neste caso será aplicada a cor red*

2. **Fonte/Origem**
   É verificado a origem dos estilos, seja um estilo de *browser*, CSS de uma extensão do *browser* ou o CSS produzido (pelo *web developer*). Priorizado da seguinte forma:

   1. CSS produzido (autor)
   2. Extensão do browser (utilizador)
   3. Browser (browser)

   Se os conflitos têm a mesma origem, normalmente do *autor*, a cascata segue para a seguinte fase: especificidade dos seletores (selector specificity).

3. **Especificidade do seletor**
   Os tipos de seletores por ordem, da maior para a menor prioridade são:

   1. Inline styles
   2. IDs
   3. Classes (e pseudo-classes e atributos)
   4. Elementos (e pseudo-elementos)

ID	Classes	Elementos
0	0	0

Para cada um dos tipos de seletores nas regras com declarações em conflito, um 1 é adicionado à coluna do tipo de seletor. De entre as declarações em conflito, à declaração com o número maior é atribuída a maior prioridade e consequentemente ganha o conflito.

> *No VS Code, ao passar por cima de um determinado seletor, é mostrada a respetiva especificidade.*

4. **Ordem/Posição**
   Quando as declarações em conflito têm o mesmo peso de especificidade, a regra que aparece em último

vence a *batalha da cascata*.

Pseudo-seletores

Pseudo-class selectors

As CSS pseudo-classes são utilizadas para adicionar estilos a seletores, mas apenas quando estes seletores correspondem a certas condições. Uma *pseudo-class* é expressa adicionando dois pontos (:) depois de um seletor CSS, seguida de uma pseudo-class tal como "hover", "focus", ou "active".

Pseudo-class sintaxe

```
selector:pseudo-class {
    propriedade: valor;
}
```

Alguns exemplos (***Dynamic pseudo-classes***):

`:link`
Refere a tag `<a>` com o atributo `href`.

`:active`
Refere a hiperligação quando for ativada (clicada).

`:visited`
Refere a hiperligação quando já foi visitada.

`:hover`
Esta é uma das mais utilizadas. Refere a hiperligação quando o cursor do rato passa por cima dela.

Exemplo:

HTML

```
<div class="container">
  <h1>Top 5 linguagens de programação</h1>
  <div class="list">
    <ul>
      <li class="list-item"> <a href="#"> Javascript </a> </li>
      <li class="list-item"> <a href="#"> Python </a> </li>
      <li class="list-item"> <a href="#"> Java </a> </li>
      <li class="list-item"> <a href="#"> PHP </a> </li>
      <li class="list-item"> <a href="#"> C# </a> </li>
    </ul>
  </div>
</div>
```

CSS

```
.list-item:hover {
    background-color: aliceblue;
}

.list a:link{
```

```
    color: black;
}

.list a:active{
    color: green;
}

.list a:visited{
    color: red;
}
```

Alguns exemplos (***Structural pseudo-classes***):

As ***Structural pseudo-classes*** referem-se a elementos de acordo com a sua posição na árvore do documento (*document tree*) e na relação com outros elementos.

`:root`
Seleciona o elemento que está na raiz do documento, especificamente o elemento `<html>`. A utilização da pseudo-class `:root` serve para a declaração de variáveis globais a utilizar pelas CSS.

`:first-child`
Seleciona o primeiro elemento (*child*) dentro de outro elemento (*parent*).

`:last-child`
Seleciona o último elemento (*child*) dentro de outro elemento (*parent*).

`:nth-child()`
Seleciona os elementos com base em expressões algébricas (por exemplo "2n" ou "2n-1"). O `2n` pode ser utilizado para selecionar posições par e `2n-1` para posições ímpares. Pode ser utilizado, ainda e por exemplo, para selecionar os seis primeiros elementos.

```
.list-item:nth-child(2n-1) {
    background-color: #DDD;
}
```

`:first-of-type`
Seleciona o primeiro elemento dentro de outro elemento de determinado tipo. Por exemplo: se temos dois `<div>` cada um com um `<p>`, um `<a>`, um `<p>` e um `<a>`, então `div a:first-of-type` seleciona a primeira hiperligação (`<a>`) dentro de cada um dos `<div>`

`:last-of-type`
Tem o mesmo funcionamento do `:first-of-type` mas seleciona o último elemento em vez do último.

`:nth-of-type()`
Funciona como o `:nth-child`. Por exemplo, para selecionar todos os parágrafos em posição ímpar, dentro de um `<div>`, temos `div p:nth-of-type(odd)`

```
.list-item:nth-of-type(odd) {
    background-color: #DDD;
}
```

`:only-of-type`
Seleciona o elemento apenas se for do tipo definido e for único dentro de determinado elemento (*parent*).

`:nth-last-of-type()`
Funciona como o `:nth-of-type` mas a contagem começa a partir do fim em vez de começar do início.

`:nth-last-child()`
Funciona como o `:nth-child`, mas a contagem começa a partir do fim em vez de começar do início

Pseudo-element selectors

Um *pseudo-element* é adicionado a um seletor que permite aplicar estilos a uma parte específica do(s) elemento(s) selecionado(s). Por exemplo, `::first-line` pode ser usado para alterar a fonte da primeira linha de um parágrafo.
Um *pseudo-element* é expresso adicionando duas vezes dois pontos (::) depois de um seletor CSS.

Pseudo-element sintaxe:

```
selector::pseudo-element {
    propriedade: valor;
}
```

`::first-line`
Adiciona estilo à primeira linha de texto conforme o seletor associado.

```
p::first-line {
    font-size: 16px;
}
```

`::first-letter`
Adiciona estilo à primeira letra do texto conforme o seletor associado.

```
p::first-letter {
    color: red;
    font-size: 32px;
}
```

`::before`
Adiciona conteúdo antes de um elemento.

```
div::before {
    content: ">";
    color: blue;
}
```

`::after`
Adiciona conteúdo depois de um elemento.

```
div::after {
    content: "<";
    color: blue;
}
```

`::selection` Utilizado para selecionar o conteúdo de um elemento e, a partir disso, personalizar algumas propriedades dessa área, como a cor de fundo e do texto.

```
p::selection {
    background-color: #F00;
    color: #FFF;
}
```

Cores

Formato	Sintaxe	Exemplo
Hex Code	#RRGGBB	`p { color: #FF0000; }`
Short Hex Code	#RGB	`p { color: #6A7; }`
RGB %	rgb(rrr%,ggg%,bbb%)	`p { color: rgb(50%,50%,50%); }`
RGB Absolute	rgb(rrr,ggg,bbb)	`p { color: rgb(0,0,255); }`
keyword	aqua, tomato, chocolate, ...	`p { color: tomato; }`

Unidades de medida

Unidade	Descrição	Observações
px	Valor absoluto em *pixels*	
rem	Valor relativo ao tamanho da fonte do elemento raiz, ou seja, relativa ao tamanho da fonte do *browser*	
em	Relativa ao tamanho da fonte do elemento pai (*parent*)	
%	Relativa ao elemento pai	
vw	Relativa à largura da *viewport* (*viewport's width*)	1vw = 1% * viewport width
vh	Relativa à altura da *viewport* (*viewport's height*)	1vh = 1% * viewport height
vmin	Relativa a dimensão menor da *viewport*	1vmin = min(1vh, 1vw)
vmax	Relativa a dimensão maior da *viewport*	vmax = max(1vh, 1vw)
ch	Relativo à largura do glifo (carácter) "0" da fonte do elemento	
in	Polegadas (*Inches*)	1in = 2.54cm = 96px
pc	Picas	1pc = 1in / 6 = 16px
pt	Points	1pt = 1in / 72 = 1.333px (aproximadamente)
cm	Centímetros	1cm = 96px / 2.54 = 37.8px (aproximadamente)
mm	Milímetros	1mm = 1cm / 10 = 3.78px (aproximadamente)

Listas

list-style-type

para listas não ordenadas `<ul>`

Propriedade	Descrição
none	Oculta os *marcadores* da lista
disc (valor por defeito)	Círculo preenchido
circle	Círculo sem preenchimento
square	Quadrado preenchido

list-style-type

para listas ordenadas `<ol>`

Propriedade	Descrição	Exemplo
decimal	Números	1,2,3,4,5
decimal-leading-zero	0 antes dos números	01, 02, 03, 04, 05
lower-alpha	Letras em minúsculas	a, b, c, d, e
upper-alpha	Letras em maiúsculas	A, B, C, D, E
lower-roman	Numeração romana em minúsculas	i, ii, iii, iv, v
upper-roman	Numeração romana em maiúsculas	I, II, III, IV, V
...	...	...
lower-greek	Alfabeto grego em minúsculas	alpha, beta, gamma
...	...	...
katakana-iroha	The marker is katakana-iroha	I, RO, HA, NI, HO, HE, TO

Exemplo:

```
ol.movies {
    list-style-type: lower-alpha;
}
```

list-style-position

Propriedade	Descrição
inside	Se o texto do item for para uma segunda linha, o texto aparece por baixo do *marker*. É feita, ainda, uma indentação a toda a ilsta.
outside	Se o texto do item for para uma segunda linha, o texto aparece alinhado com o início da primeira linha (à direita do *marker*).

Exemplo:

```
ol.movies {
    list-style-type: lower-alpha;
    list-style-position: outside;
}
```

list-style

A propriedade `list-style` permite definir todas as propriedades de uma lista (`list-*`) numa expressão única. Estas propriedades podem aparecer em qualquer ordem.

Exemplo:

```
ol.movies {
    list-style: outside upper-alpha;
}
```

Fundos

Um fundo pode ser atribuído a vários elementos HTML. As propriedades são as seguintes:

`background-color`
Define a cor de fundo de um elemento.

```
div {
    background-color: #F00;
}
```

`background-image`
Define a imagem de fundo de um elemento.

```
div {
    background-image: url("./assets/imgs/jumanji.jpg");
}
```

`background-repeat`
Controla a repetição de uma imagem de fundo.

```
body {
    background-image: url("./assets/imgs/jumanji.jpg");
    background-repeat: repeat-x;
}
```

`background-position`
Controla a posição de uma imagem de fundo.

```
body {
    background-image: url("./assets/imgs/jumanji.jpg");
    background-repeat: no-repeat;
    background-position: 100px 200px;
}
```

O exemplo acima define a posição da imagem de fundo para que fique distanciada 100 pixels da esquerda e 200 pixels do topo.

`background-attachment`
Controla o *scroll* de uma imagem de fundo.

```
body {
    background-image: url("./assets/imgs/jumanji.jpg");
    background-repeat: no-repeat;
    background-attachment: fixed;
}
```

`background`
A propriedade `background` é uma forma abreviada (*shorthand*) de definir várias propriedades do fundo de um elemento, em simultâneo.

```
div {
    background: url("./assets/imgs/jumanji.jpg") repeat fixed;
}
```

Fontes

Podem ser definidas as seguintes propriedades num elemento, relativas à fonte (de texto):

`font-family`
Utilizada para alterar o tipo de letra.

`font-style`
Utilizada para fazer itálico (italic).

`font-variant`
Utilizada para criar o efeito de maiúsculas *pequenas* (*small-caps effect*).

`font-weight`
Utilizada para definir o peso da fonte (*bold* ou *light*).

`font-size`
Utilizada para aumentar ou diminuir o tamanho da fonte.

`font`
Forma abreviada (*shorthand*) de definir várias propriedades da fonte de um elemento, em simultâneo.

Texto

Podem ser definidas as seguintes propriedades num elemento, relativas a texto:

`color`
Define a cor.

`letter-spacing`
Utilizada para adicionar ou subtrair espaço entre as letras de uma palavra.

`word-spacing`
Utilizada para adicionar ou subtrair espaço entre as palavras de uma frase.

`text-indent`
Define a indentação do texto de um parágrafo.

`text-align`
Define o alinhamento de texto num documento.

`text-decoration`
Utilizada para sublinhar (*underline*), traçar por cima (*overline*) ou cortar o texto (*strikethrough*).

`text-transform`
Utilizada para capitular texto, converter para maiúsculas ou converter para minúsculas.

`text-shadow`
Define uma sombra no texto.

Exemplo:

```
.texto-sombra {
    text-shadow: 1px 1px 2px black;
}
```

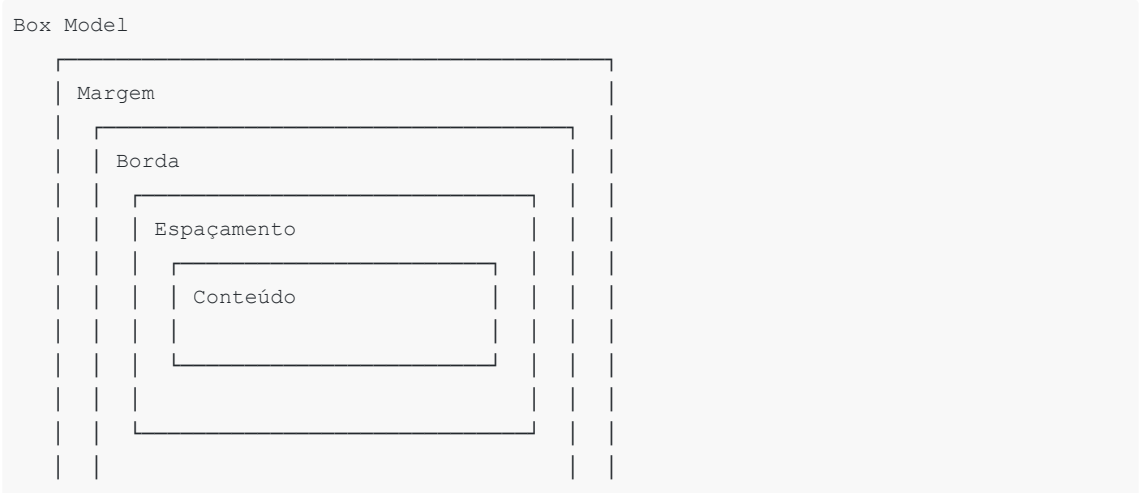
No exemplo acima os parâmetros significam, pela ordem indicada:

- deslocamento em x
- deslocamento em y
- raio de desfoque
- cor

Box Model

Box Model é composta por quatro partes:

- conteúdo (*Content box*)
- espaçamento (*Padding box*)
- bordas (*Border box*)
- margens (*Margin box*)



```
    |  └──────────────────────────────────┘  |
    |                                        |
    └────────────────────────────────────────┘
```

CSS

```css
.exemplo {
    width: 50px;
    height: 50px;
    border: 1px solid gray;
    padding: 10px 20px;
}
```

O elemento fica com a dimensão de 72 pixels de altura e 92 pixels de largura.

Altura

```
50 (altura definida) +
10 (padding top) +
10 (padding bottom) +
1 (border top) +
1 (border bottom)

= 72 pixels de altura
```

Largura

```
50 (largura definida) +
20 (padding left) +
20 (padding right) +
1 (border left) +
1 (border right)

= 92 pixels de largura
```

box-sizing

A propriedade `box-sizing` permite que o comportamento do box-model seja manipulado.
Por defeito, todos os elementos tem o valor `content-box` para essa propriedade (com o comportamento do descrito acima).

Se tivermos, por exemplo:

```css
.exemplo {
    box-sizing: border-box;
}
```

O valor altera o comportamento do box-model, fazendo com que o *browser* calcule a largura/altura do elemento contando não apenas o seu conteúdo (como visto no content-box), mas também considerando o padding (espaçamento) e border (borda) do elemento.

Exemplo para comparação entre `content-box` e `border-box`:

```
.contentbox, .contentbox2, .borderbox, .borderbox2 {
    background-color: #FF0;
    width: 50%;
    border: 10px solid #000;
    padding: 20px;
}

.contentbox2 {
    border: 20px solid #000;
}

.borderbox {
    box-sizing: border-box;
}

.borderbox2 {
    box-sizing: border-box;
    border: 20px solid #000;
}
```

Posição

O posicionamento com as CSS permite aos *designers* e programadores o posicionamento dos elementos HTML numa página web. A posição com as CSS pode ser definida como `static`, `relative`, `absolute` ou `fixed`.

position: fixed

Define/fixa a posição de um elemento HTML para um ponto específico numa página. O elemento fixo permanece o mesmo, independentemente do deslocamento da página (scrolling) como, por exemplo, numa barra de navegação que fica sempre fixa no topo.

position: absolute

O valor absoluto remove completamente um elemento da sua posição "normal" num documento. Através das propriedades `top`, `left`, `bottom` e `right`, um elemento pode ser posicionado em qualquer local da página.

position: relative

O valor `relative` da propriedade `position` permite que um elemento seja posicionado em relação ao local em que estaria originalmente na página.

> *A propriedade `display` determina como é mostrado um bloco de um elemento. Os valores mais comuns para esta propriedade são `block`, `inline` e `inline-block`.*

display: block

Os elementos `block-level` ocupam toda a largura de seu *container* com quebras de linha antes e depois, e podem ter sua altura e largura ajustadas manualmente.

display: inline

Os elemento *inline* (em linha) ocupam o mínimo de espaço possível, fluem horizontalmente e não podem ter a sua largura ou altura ajustadas manualmente.

display: inline-block

Os elementos `inline-block` podem aparecer lado a lado e podem ter a sua largura e altura ajustadas manualmente.

float

A propriedade `float` das CSS determina a que distância à esquerda ou à direita um elemento deve flutuar no seu elemento pai. O valor `left` flutua um elemento para o lado esquerdo do seu *container* e o valor `right` flutua um elemento para o lado direito do seu *container*. Para a propriedade `float`, a largura do *container* deve ser especificada ou o elemento assumirá a largura total do elemento que o contém.

```
.exemplo {
    float: right;
}
```

z-index

A propriedade z-index especifica a que distância para trás ou a que distância para a frente um elemento aparecerá numa página quando se sobrepõe a outros elementos.

A propriedade `z-index` usa valores inteiros, que podem ser positivos ou negativos. O elemento com o valor de índice z mais alto fica em primeiro plano, enquanto o elemento com o valor de índice z mais baixo ficará mais atrás.

```
.element1 {
  position: absolute;
  z-index: 1;
}

.element2 {
  position: absolute;
  z-index: -1;
}
```

No exemplo, acima, o elemento com a classe `element1` sobrepõe-se ao elemento com a classe `element2`.

Flexbox

A flexbox tem os mesmos princípios da *grid* (mas não exatamente os mesmos) na construção do *layout* de uma página e respetivos elementos, ajustando-os automaticamente conforme necessário. Uma flexbox, em vez de criar uma grelha completa com várias linhas e colunas, trabalha o posicionamento dos elementos filhos, em relação uns aos outros, dentro do elemento pai.

> *Se for necessário desenhar em duas direções (linhas e colunas) deve ser utilizada a grid. No caso de ser necessário trabalhar apenas uma direção (apenas uma linha ou uma coluna) deve ser utilizada a flexbox.*

O princípio de funcionamento consiste em *encaixar* tudo numa linha, ou numa coluna. Caso não seja possível, e quisermos um ajuste automático, deve ser utilizada a propriedade `flex-wrap` (ver exemplo abaixo). No caso de o `flex-wrap` não ser especificado a flexbox comprime todos os elementos para que fiquem todos na mesma linha (se nenhuma largura mínima `min-width` for especificada para os elementos filhos), ou permitir que eles excedam a área de visualização (se uma `min-width` for especificada).

Exemplo CSS:

```css
.flex {
    display: flex;
    flex-wrap: wrap;
}

.flex > div {
    width: 300px
}
```

Exemplo HTML:

```html
<div class="flex">
    <div>a</div>
    <div>b</div>
    <div>c</div>
</div>
```

Mais detalhe... - *disponível na próxima aula*

Grid

A grid tem como foco principal a criação de uma grelha com várias linhas e colunas que podem ser preenchidas com elementos. Esta criação é conseguida através de um *container* `grid` que é composto por elementos filhos. Há um grande número de personalizações que pode ser feito nas linhas, colunas e células da grid.

A grid tem a capacidade de repetir colunas ou linhas automaticamente, bem como redimensionar, automaticamente, as colunas no espaço disponível.

Exemplo CSS:

```css
.grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
}
```

Exemplo HTML:

```html
<div class="grid">
    <div>a</div>
    <div>b</div>
    <div>c</div>
</div>
```

No exemplo acima, a grid transforma o conteúdo em colunas que, naturalmente, ocupam o espaço disponível. Cada coluna fica com uma largura mínima de pelo menos 300px e uma largura máxima de não mais que 1fr. O 1fr é uma unidade CSS que significa fração, que informa o *browser* para dividir o espaço de forma igual e dar a cada coluna uma fração desse espaço. Neste caso, são criadas três colunas iguais, cada uma ocupando 1/3 do espaço disponível, mas nunca menor que 300px. No caso de ser adicionado outro `div`, ou se fosse retirado um, a grid trata a nova estrutura recalculando automaticamento os ajustes necessários. No caso do dispositivo de visualização ser de pequenas dimensões, sem capacidade para acomodar as colunas lado a lado, haverá um quebra automática das colunas restantes para a próxima linha.

Mais detalhe... - *disponível na próxima aula*

Comentários

Quer o comentário de linha única, quer o de múltiplas linhas, especifica-se (começa e termina) da mesma forma.

```
/* O meu comentário */
```

ou (multi-linha)

```
/*
p {
    color: #00F;
}
*/
```

Conteúdo dinâmico (variáveis)

A capacidade de podermos definir uma variável para algo, tal como uma cor, e podermos fazer uso dela numa CSS, permite o desenvolvimento consistente e de fácil manutenção de folhas de estilo. Isto sem esquecer o DRY (*Don't repeat yourself*).
Para o efeito devem ser usadas as variáveis nativas das CSS (*CSS Custom Properties*).

```
:root {
  --color-ESTG: #f2a900;
}

header {
  background-color: var(--color-ESTG);
}

footer {
  color: var(--color-ESTG);
}
```

Responsive: Media Queries

As *media queries* são um componente importante do *responsive design*
Uma *media querie* é utilizada para aplicar um conjunto de estilos baseado num conjunto de características, da área de visualização, tais como largura(*width*) e altura (*height*).

Sintaxe:

```
@media media-type (media-feature){
    /* lista de estilos */
}
```

Media type (tipo de dispositivo)	Descrição
`all`	todos os dispositivos

`print`	dispositivos no modo de visualização de impressão
`screen`	dispositivos com ecrã
`speech`	dispositivos de leitura de ecrã, onde o conteúdo é lido em voz alta para o utilizador

O *media type* pode ser omitido, sendo que o valor por defeito é `all`. A sintaxe, neste caso, fica:

```
@media (media-feature){
    /* lista de estilos */
}
```

Media feature descreve as características específicas de um determinado dispositivo de saída.
Lista completa de media features: MDN Web Docs

Para a criação de *media queries* mais complexas podem ser utilizados os seguintes operadores lógicos:

Operador lógico	Descrição
`and`	Utilizado para juntar várias *media features*. Se todas as *media features* forem verdadeiras então os estilos entre as chavetas são aplicados na página.
`not`	Negação da *query*, o que significa tornar uma *query* falsa em verdadeira e vice-versa.
`,`	A vírgula separa múltiplas *media features* e aplica os estilos contidos entre chavetas se uma das condições for verdadeira.

Exemplos:

- alterar a cor de fundo quando a largura do dispositivo for 576px ou inferior.

```
@media (max-width: 576px) {
body {
  background-color: #87ceeb;
}
/* ... */
}
```

- alterar a cor de fundo se o dispositivo tiver uma largura entre 576px e 768px. Para o efeito utiliza-se o operador `and`.

```
@media (min-width: 576px) and (max-width: 768px) {
body {
  background-color: #de3163;
}
/* ... */
}
```

CSS Frameworks

(*por ordem de relevância (desc.) - dados de Openbase a 01.Novembro.2021*)

- Bootstrap
- tailwindcss
- Ulkit
- Carbon Design
- BULMA