

Programação Web

- ✓ CSS
 - ✓ Conceitos iniciais e sintaxe básica
 - ✓ Como inserir o CSS nas páginas
 - ✓ Seletores CSS
 - ✓ Exemplos
 - ✓ Exercícios
- ✓ Introdução ao Javascript

CSS (Cascading Style Sheet) = Folha de Estilo em Cascata

O CSS é uma linguagem que nos **permite adicionar estilos as nossas páginas**, em outras palavras, seria a **formatação visual** (aparência, layout) das páginas, como tipo de fonte, cores, espaçamentos, entre outros.

A W3C (World Wide Web Consortium) também especifica o CSS, existem diversos módulos que os navegadores aceitam em relação a última versão.

Com o uso do CSS externo, diminuímos o trabalho devido a possibilidade de reutilização dos estilos em diferentes tags e principalmente em diferentes páginas.

CSS



Para inserir o CSS em uma página HTML, podemos escolher as seguintes opções:

- CSS externo
- CSS incorporado
- CSS inline (deve ser evitado, porque mistura a estrutura com o visual)

Toda a configuração de formatação fica dentro de um arquivo .css que é chamado no cabeçalho (HEAD) do documento HTML com a tag link

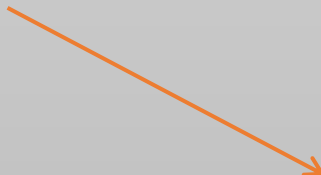
Ex:

```
<link rel="stylesheet" href="estilo.css" type="text/css" />
```

As configurações definidas no arquivo **estilo.css** valem para todas as páginas que fazem referência ao arquivo.

Exemplo simples:

```
<!--exemplo1.html-->
<html>
<head>
  <title> Teste CSS externo </title>
  <link rel="stylesheet" href="estilo.css" type="text/css" />
</head>
<body>
<p> Somente um parágrafo </p>
</body>
</html>
```



```
/*estilo.css*/
p {
  font-family: "verdana";
  color: red;
}
```

Todas as configurações ficam dentro do cabeçalho (HEAD) da página e são delimitadas pelas tags **<style>...</style>**

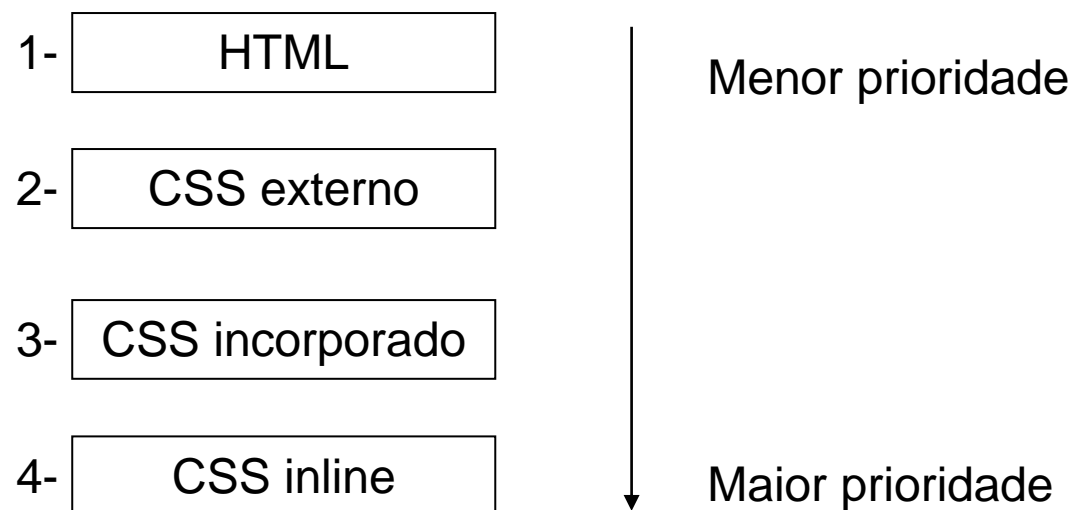
As configurações valem para a página inteira.

Este tipo de CSS não permite a reutilização, como seria no CSS externo.

Exemplo simples:

```
<!--exemplo2.html-->
<html>
<head>
  <title> Teste CSS embutido </title>
  <style type="text/css">
    p {
      font-family: "verdana";
      color: red;
    }
  </style>
</head>
<body>
  <p> Somente um parágrafo </p>
</body></html>
```

Caso você utilize as três formas de inserir CSS em um documento HTML, deve ficar atento a prioridade entre eles, conforme demonstrado abaixo:



Na figura acima, o CSS inline sobrepõe os demais estilos, caso este não exista, o CSS incorporado sobrepõe o externo e assim ocorre com as demais variações.

Um seletor CSS é um padrão criado para ser aplicado ao elemento(s) desejado(s) no HTML.

Existem diversos seletores, aqui iremos abordar as principais, porém um estudo aprofundado pode ser obtido através dos links:

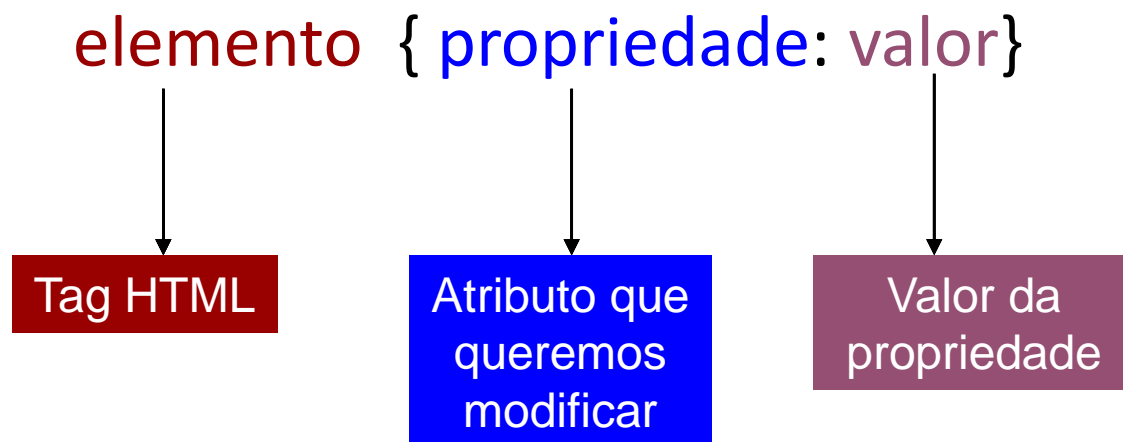
- http://www.maujor.com/tutorial/seletores_css21_parte1.php
- <http://www.w3.org/TR/css3-selectors/>
- http://www.w3schools.com/cssref/css_selectors.asp

Iremos utilizar os seguintes seletores (básicos):

- Para um elemento específico (seletor tipo)
- Para classes genéricas (seletor classe)
- Para estilos individuais (seletor id)

Podemos definir nossos estilos para qualquer elemento do HTML.

Sintaxe:



```
p { color:blue}
b {
  color:red;
  font-family: verdana;
}
i {
  color:red;
  font-family: "sans serif";
}
```

Um padrão que é utilizado para a criação dos estilos é inserir uma propriedade em cada linha, isso facilita a leitura posterior.

```
body {
  color:green;
  font-family: "sans serif";
}
```

```
<!--exemplo3.html-->
<html><head>
<style type="text/css">
  p { color:blue}
  b { color:red; font-family: verdana; }
  i { color:red; font-family: "sans serif"; }
  body {
    color:green;
    font-family: "sans serif";
  }
</style>
</head>
<body>
  <p> parágrafo </p>
  <b> negrito</b><br />
  <i> itálico </i><br />
  Texto do body.
</body></html>
```

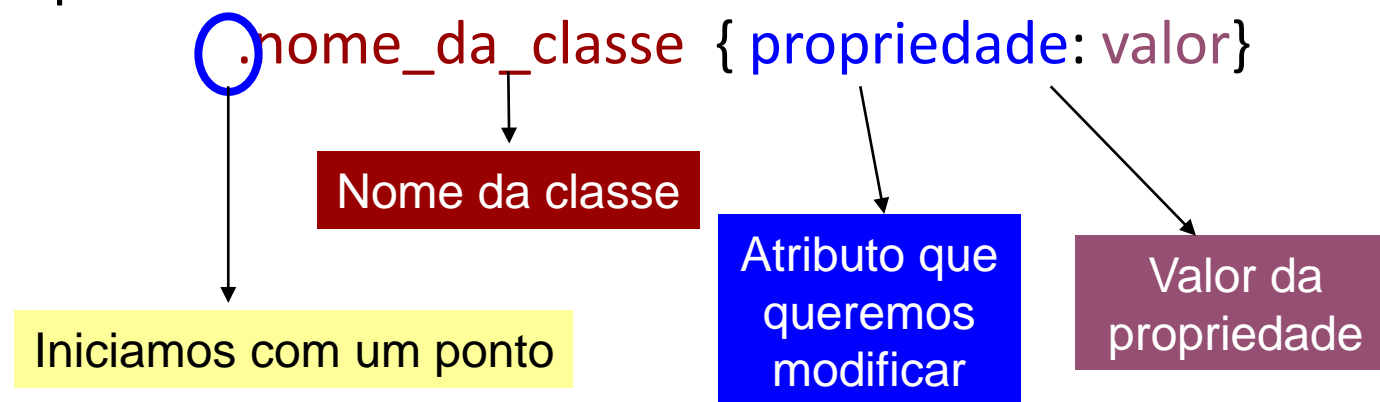
Até o momento, definimos configurações de apresentação para elementos específicos do HTML. Sempre que usamos o elemento formatado em nosso documento, ele assume a formação definida no CSS. Surge então um problema: como formatar um mesmo elemento de diferentes formas em nosso documento?

Para resolver esse problema, existem as classes e os estilos individuais.

Podemos criar classes genéricas (não estão associadas a nenhum elemento) que podem ser aplicadas a qualquer elemento do HTML, sempre que for necessário, assim, uma classe pode ser usada várias vezes em um mesmo documento.

Para utilizar uma classe em um marcador, acrescentamos o atributo **class**.

Sintaxe para definir o estilo:



```
<!--exemplo4.html-->
```

```
<html>
```

```
<head>
```

```
<title>Exemplo</title>
```

```
<style type="text/css">  
  .cor_azul { color:blue}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<p> parágrafo 1 sem formatação </p>
```

```
<p class="cor_azul"> parágrafo 2 com formatação</p>
```

```
<i class="cor_azul"> itálico com formatação </i><br />
```

```
Texto do body.
```

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

Cria a classe "cor_azul" genérica, logo, pode ser usada em qualquer tag do HTML.

parágrafo 1 sem formatação

parágrafo 2 com formatação

itálico com formatação

Texto do body.

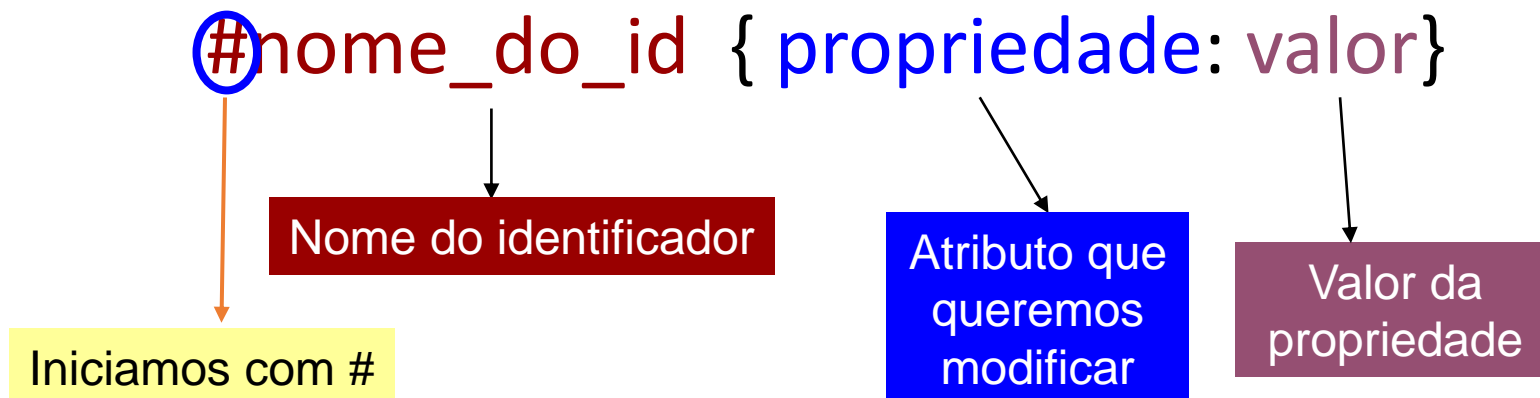
Aplica a classe "cor_azul" na tag p e na tag i. Veja que para aplicar o estilo, devemos chamar a classe com o atributo "class".

Assim como as classes genéricas, os estilos individuais podem ser aplicados a qualquer elemento do HTML. A diferença é que esses estilos são únicos, logo, só podem ser utilizados uma única vez dentro do mesmo documento.

São utilizados também para identificar um elemento HTML para posteriormente ser manipulado através da linguagem JavaScript. Este estilo é a identificação da tag.

Para utilizar um estilo individual em um marcador, acrescentamos o atributo **id**.

Sintaxe para definir o estilo:



```
<!--exemplo5.html-->
```

```
<html>
```

```
<head>
```

```
  <title>Exemplo</title>
```

```
  <style type="text/css">
```

```
    #cor_azul { color:blue}
```

```
  </style>
```

```
</head>
```

```
<body>
```

```
  <p> parágrafo 1 sem formatação </p>
```

```
  <p id="cor_azul"> parágrafo 2 com formatação</p>
```

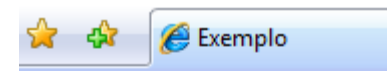
```
  <i> itálico </i><br />
```

```
  Texto do body.
```

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

Cria o id "cor_azul", pode ser usado em qualquer tag do HTML, porém somente um vez.

Aplica o id "cor_azul" na tag p. Veja que para aplicar o estilo, devemos chamar o identificador com o atributo "id".



parágrafo 1 sem formatação

parágrafo 2 com formatação

itálico

Texto do body.

Podemos criar classes genéricas que podem ser aplicadas a qualquer tag do HTML. Essas classes podem ser usadas mais de uma vez dentro do documento. Para acioná-la, usamos o atributo class na tag desejada.

Podemos criar um estilo individual que aplica um estilo e identifica uma tag do HTML. Esta forma de estilo só pode ser usada uma vez dentro do documento. Para acioná-lo, usamos o atributo id na tag desejada.

Podemos ter em uma mesma tag, os atributos class e id.

Ao chamar a classe em uma tag com o atributo class, colocamos somente o nome da classe, ou seja, na chamada da classe, não utilizamos o ponto.

O mesmo ocorre quando chamamos o estilo individual com o atributo id, também não inserimos o #, somente o nome do estilo.

Existem, também, os chamados CSS Combinators, para estabelecer estilos a tags que se encontram relacionadas com outras pela colocação dentro do documento.

Por exemplo, podemos construir estilos hierárquicos, de forma a aplicar um estilo apenas a tags que se encontrem dentro de outra, com as sintaxes:

`tag1 > tag2`

para uma tag2 filha de tag1

`tag1 tag2`

para uma tag2 descendente de tag1

```
<style type="text/css">
  body{
    background-color:#ebebeb;
  }
  p {
    color:black;
  }
  div > p {
    color:red;
  }
  div span {
    color:blue;
  }
</style>
```

```
<style type="text/css">
  body{
    background-color:#ebebeb;
  }
  p {
    color:black;
  }
  div > p {
    color:red;
  }
  div span {
    color:blue;
  }
</style>
```

Estilos hierárquicos

Parágrafo que é filho direto da div alterado com o estilo hierárquico `div > p`

Parágrafo que não é filho direto da div não é alterado com o estilo `div > p`

Span que é filho direto da div alterado com o estilo hierárquico `div span`

Span que não é filho direto da div alterado com o estilo hierárquico `div span`

```
<body>
  <h3>Estilos hierárquicos</h3>

  <div>
    <p>Parágrafo que é filho direto da div alterado com o estilo hierárquico div > p </p>
    <span><p>Parágrafo que não é filho direto da div não é alterado com o estilo div > p</p></span>
  </div>

  <div>
    <span>Span que é filho direto da div alterado com o estilo hierárquico div span</span>
    <p><span>Span que não é filho direto da div alterado com o estilo hierárquico div span</span></p>
  </div>
</body>
```

Como geralmente os arquivos com CSS possuem muitas configurações, é interessante você inserir comentários descrevendo o que cada bloco de estilo faz. Para isso usamos `/*.....*/`

Exemplo:

```
<style type="text/css">  
    /*Cria um id chamado cor_azul*/  
    #cor_azul { color:blue}  
</style>
```

Os estilos CSS estudados até o momento foram aplicados a tags do HTML que já possuem um formato padrão inicial. Porém, existem situações em que precisamos de tags "limpas", sem formatação padrão. Para estes casos, temos a disposição a tag **div** e a tag **span** e as tags de estrutura semântica (article, aside, nav etc).

A tag **div** é muito utilizada na criação de layouts das páginas e áreas para conteúdos. Já a tag **span** é utilizada para aplicar configurações dentro do conteúdo, isso devido ao fato de não possuir nenhum formato pré-definido. Já as tags de estrutura semântica devem ser utilizadas no conteúdo para definir um significado para cada área do conteúdo do site.

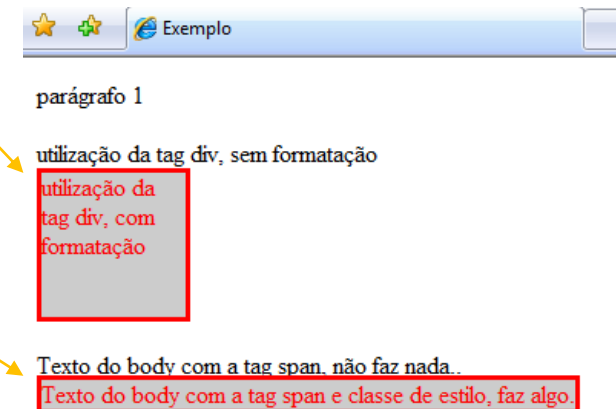
Todas as tags aceitam o uso de classes (class) e estilos individuais (id) e também aceitam o uso de ambos os atributos.

```
<!--exemplo6.html-->
<html>
<head>
<title>Exemplo</title>
<style type="text/css">
  .caixa {
    border: 3px solid #ff0000;
    background-color: #cccccc;
    color: red;
  }
  #aviso {
    width: 100px;
    height: 100px;
  }
</style>
</head>
```

```
<body>
<p> parágrafo 1 </p>
<div> utilização da tag div, sem formatação</div>
<div class="caixa" id="aviso"> utilização da tag div, com
formatação</div>
<br />
<span>Texto do body com a tag span, não faz nada.</span>
<br />
<span class="caixa">Texto do body com a tag span e classe de estilo, faz
algo.</span>
</body>
</html>
```

Uso dos atributos class e id

Faz uso da classe caixa.



```
<!DOCTYPE HTML><html><head>
<title>Untitled Document</title>
<style type="text/css">
body {
    background-color:#003366;
}

#principal {
    width: 757px;
    margin:0 auto;
}
h2 {
    color:#990000;
}
.data {
    color:#999999;
    font-size:8pt;
}
#topo_geral {
    background-image:url(imagens/topo.JPG);
    height: 62px;
    background-repeat:no-repeat;
}
```

<!--exemplo7.html-->

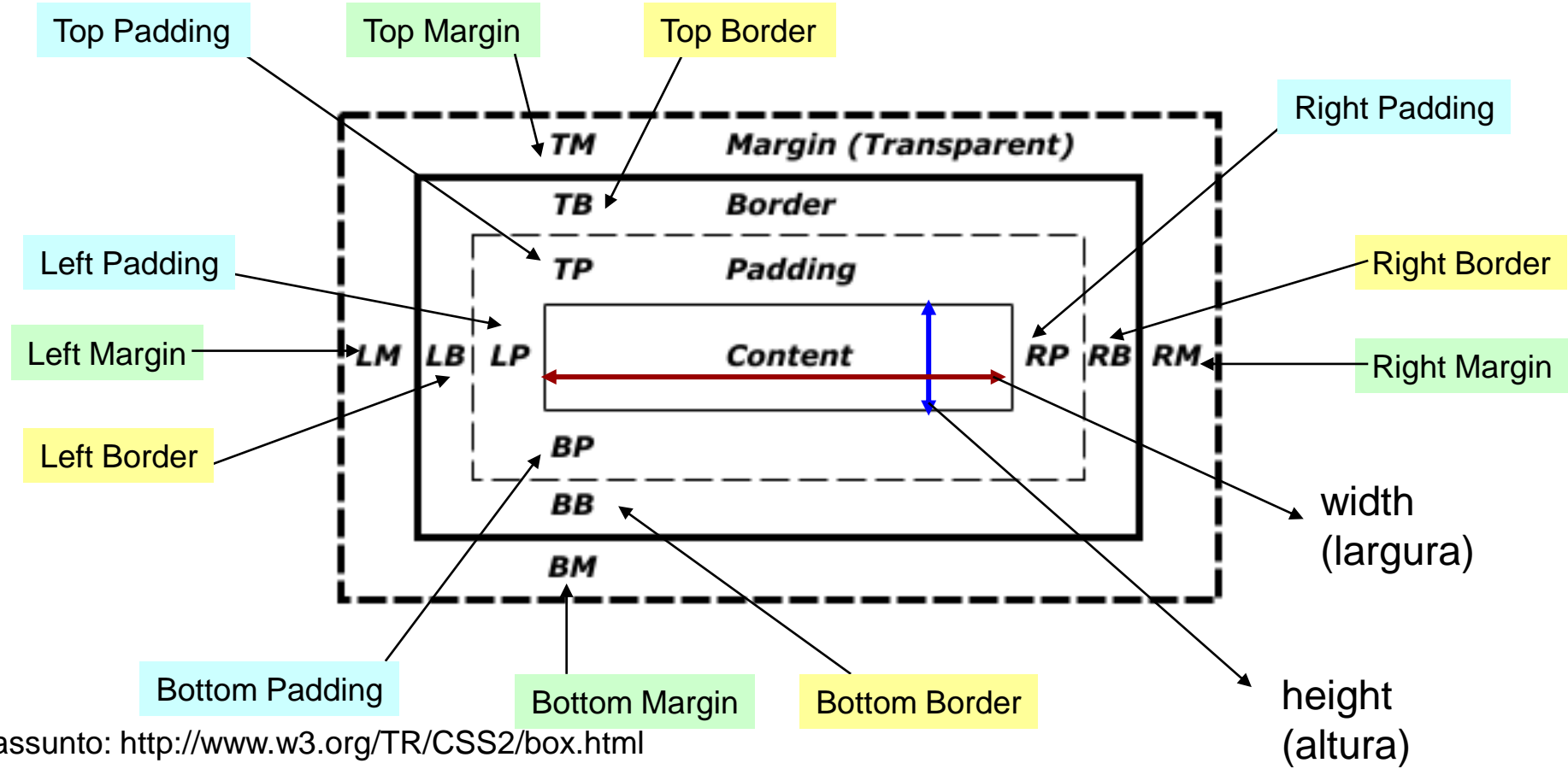
```
article>p { text-align:justify; }
article {
    background-color:#FFF;
    padding:50px;
}
ul {
    list-style:none;
    padding:0;
    margin:0;
}
li { display:inline; }
nav {
    background-color:#003300;
    color:#FFFFFF;
    padding:5px;
}
</style>
</head>
```

```
<body>
<div id="principal">
  <header id="topo_geral"></header>
  <nav>
    <ul>
      <li>HOME</li> | <li>Notícias</li> | <li>Galeria</li> | <li>Contato</li>
    </ul>
  </nav>
  <article>
    <header id="topo_article">
      <h2>Greve faz governo trocar PF por militares na Copa</h2>
      <span class="data">21 de agosto de 2012 | 22h 30</span>
    </header>
    <p>    texto...    </p>
    <p>    texto...    </p>
  </article>
</div>
</body>
```

Os boxes são criados por elementos que formam blocos, como por exemplo temos as tags: p, h1 – h6, div, aside, article, entre outras. Entenda as diferentes regiões de um box.



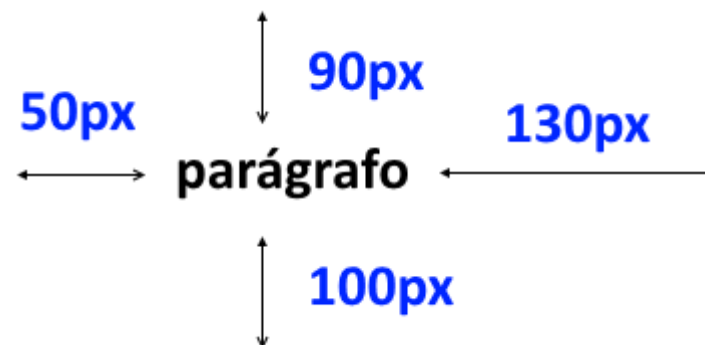
As regiões detalhadas de um box.



Para ler mais sobre o assunto: <http://www.w3.org/TR/CSS2/box.html>

Exemplo:

```
p {  
  margin-top: 90px;  
  margin-bottom: 100px;  
  margin-right: 130px;  
  margin-left: 50px;  
}
```



A propriedade *margin* poderá ter de um a quatro valores.

- **margin: 25px 50px 75px 100px;**
 - margem superior 25px
 - margem direita 50px
 - margem inferior 75px
 - margem esquerda 100px
- **margin: 25px 50px 75px;**
 - margem superior 25px
 - margem direita e esquerda 50px
 - margem inferior 75px
- **margin: 25px 50px;**
 - margem superior e inferior 25px
 - margem direita e esquerda 50px
- **margin: 25px;**
 - todas as margens 25px

Exemplo:

```
p {  
    margin: 70px 50px;  
}
```

O atributo **border** poderia especificar, adicionalmente, cor, estilo e largura da borda.

O atributo **border-radius** permite criar bordas arredondadas em elementos do tipo box.

`border-radius` = define tamanho de arredondamento dos cantos

Exemplos: `border-radius: 10px 50px 50px 10px;`

`border-radius: 10px;`

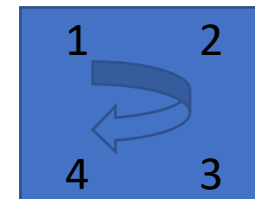
ou podemos definir separadamente o arredondamento de cada esquina:

`border-top-left-radius: tamanho;`

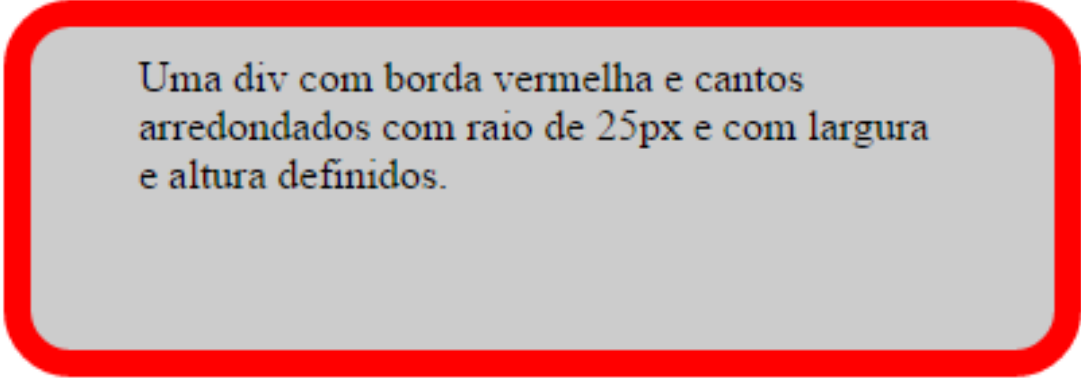
`border-top-right-radius: tamanho;`

`border-bottom-right-radius: tamanho;`

`border-bottom-left-radius: tamanho;`



```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Exemplos</title>
  <style>
    div {
      width: 300px;
      height: 100px;
      padding: 10px 40px;
      background: #ccc;
      border: 10px solid #f00;
      border-radius: 25px;
    }
  </style>
</head>
<body>
  <div>Uma div com borda vermelha e cantos
    arredondados com raio de 25px e com
    largura e altura definidos.
  </div>
</body>
</html>
```



Uma div com borda vermelha e cantos arredondados com raio de 25px e com largura e altura definidos.

Uma borda pode ser: solid, dashed, dotted, double, inset, outset,...

O atributo **box-shadow** adiciona uma sombra em algum box.

box-shadow: h-sombra v-sombra borrão propagação cor ...

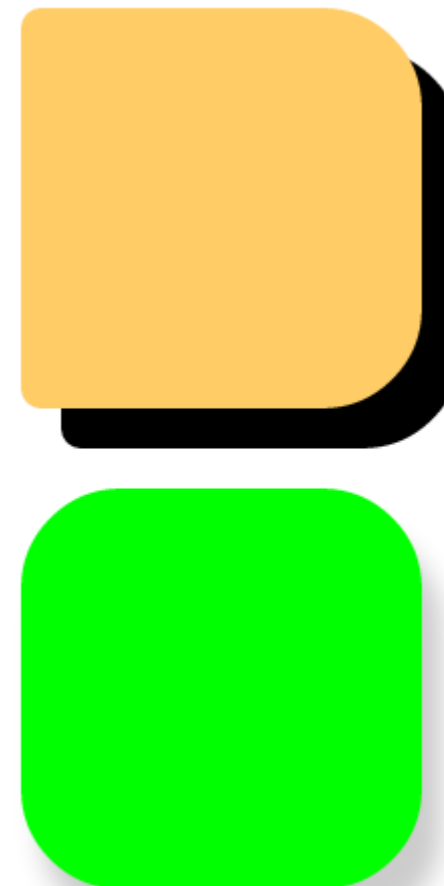
h-sombra, v-sombra são as posições da sombra horizontal e vertical (podem ser valores negativos)

borrão ou *blur* é a distância deste efeito (opcional)

propagação (disseminação) ou *spread* é o tamanho da sombra (opcional)

cor é a cor da sombra (opcional)

```
<style>
  .borda1 {
    width: 200px;
    height: 200px;
    background-color: #FFCC66;
    border-radius: 10px 50px 50px 10px;
    box-shadow: 20px 20px #000000;
  }
  .borda2 {
    width: 200px;
    height: 200px;
    background-color: #00FF00;
    border-radius: 50px;
    box-shadow: 10px 20px 20px #CCCCCC;
  }
</style>
```



As cores são exibidas através da combinação **RGB** (vermelho, verde, azul: **Red**, **Green**, **Blue**)

As cores em CSS podem ser especificadas por:

- Valores hexadecimais

- Valores RGB

- Valores RGBA

- Valores HSL

- Valores HSLA

- Predefinidos (nomes)

Valores hexadecimais

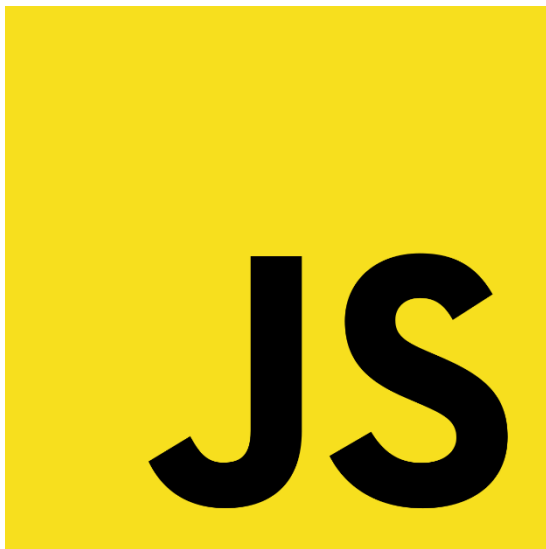
Combinação dos valores para RGB no formato hexadecimal, o menor valor seria 00 e o maior seria FF (255).

Exemplo: #FF0000; (cor vermelha)

#F00; (também cor vermelha)

Caso a sequência de valores se repita como o exemplo anterior, poderemos utilizar somente três dígitos; para as demais cores devemos utilizar os 6 dígitos

Exemplo: #6495ED (tom de azul)





- Fortemente tipada
- Linguagem de programação orientada a objetos
- Linguagem compilada
- Java é usado principalmente para backend



- Tipagem flexível
- Linguagem de script baseada em objetos
- Linguagem interpretada
- Javascript é usado tanto para o frontend quanto para o backend.

<https://www.geeksforgeeks.org/difference-between-java-and-javascript/>

https://www.java.com/pt-BR/download/help/java_javascript_pt-br.html

<https://natahouse.com/pt/saiba-as-diferencas-entre-java-e-javascript-de-uma-vez-por-todas>

<https://medium.com/@scarlett8285/java-vs-javascript-everything-you-need-to-know-3362b5fa4128>

Criado por Brendan Eich



https://pt.wikipedia.org/wiki/Brendan_Eich

JavaScript é um nome patenteado pela Oracle (antiga Sun Microsystem)

O nome (oficial) foi definido como ECMAScript, junção de ECMA (European Computer Manufactures Association) com JavaScript. Contudo o nome mercadológico mais comum é JavaScript e ECMAScript só especifica a versão.

Uma versão importante é a ECMAScript 2015 conhecida como ECMAScript 6 ou ES6, nesta versão foram introduzidos novas conceitos como declaração de constantes, classes, arrow functions, etc.

Especificação do ECMAScript:

<https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>

Última versão é a **ECMAScript 2022 (ES13)**, aprovada pela ECMA em **Jun/2022**.

<https://www.ecma-international.org/publications/standards/Ecma-262.htm>

<https://www.ecma-international.org/>

Linguagem integrada ao navegador por padrão.

No front-end usa o chamado "**modelo de execução controlado por eventos**", ou seja, o código JavaScript só é executado quando o evento ao qual está associado é acionado, mas, em alguns casos, podemos inserir o código sem a associação a eventos e o mesmo é executado conforme o navegador interpreta a página.

O JavaScript pode ser executado não apenas no navegador (que é seu padrão), mas também no servidor ou em qualquer dispositivo (como em apps) que tenha um tipo de máquina capaz de interpretar o código.

É uma linguagem interpretada, diferente de outras linguagens que fazem o processo de compilação antes de executar o programa, como por exemplo a linguagem Java.

Versões ECMAScript ou ES são: ES5, ES6, ES7, ...

Algumas versões mais antigas de navegadores não dão suporte ao ECMAScript 6 ou posterior, possuindo problemas de compatibilidade. A versão mais estável é a ES5, contudo o mercado faz referência ao ES6 ou ES6+.

6th Edition – ECMAScript 2015 [\[edit \]](#)

The 6th edition, ECMAScript 6 (**ES6**) and later renamed to ECMAScript 2015, was finalised in June 2015.^{[11][32]} This update adds significant new syntax for writing complex applications, including class declarations (`class Foo { ... }`), ES6 modules like `import * as moduleName from "..."; export const Foo`, but defines them semantically in the same terms as ECMAScript 5 strict mode. Other new features include iterators and `for...of` loops, Python-style generators, arrow function expression (`() => { ... }`), `let` keyword for local declarations, `const` keyword for constant local declarations, binary data, typed arrays, new collections (maps, sets and WeakMap), `promises`, number and math enhancements, reflection, proxies (metaprogramming for virtual objects and wrappers) and `template literals` for strings.^{[33][34]} The complete list is extensive.^{[35][36]} As the first "ECMAScript Harmony" specification, it is also known as "ES6 Harmony."

7th Edition – ECMAScript 2016 [\[edit \]](#)

The 7th edition, or ECMAScript 2016, was finalised in June 2016.^[12] Its features include block-scoping of variables and functions, destructuring patterns (of variables), proper tail calls, exponentiation operator `**` for numbers, `await`, `async` keywords for asynchronous programming (as a preparation for ES2017), and the `Array.prototype.includes` function.^{[12][37]}

The exponentiation operator is equivalent to `Math.pow`, but provides a simpler syntax similar to languages like Python, F#, Perl, and Ruby. `async` / `await` was hailed as an easier way to use promises and develop asynchronous code.

8th Edition – ECMAScript 2017 [\[edit \]](#)

The 8th edition, or ECMAScript 2017, was finalised in June 2017.^[13] Its features include the `Object.values`, `Object.entries` and `Object.getOwnPropertyDescriptors` functions for easy manipulation of Objects, `async/await` constructions which use generators and promises, and additional features for concurrency and `atomics`.^{[38][13]}

9th Edition – ECMAScript 2018 [\[edit \]](#)

The 9th edition, or ECMAScript 2018, was finalised in June 2018.^[14] New features include the spread operator, rest parameters, asynchronous iteration, `Promise.prototype.finally` and additions to `RegExp`.^[14]

The spread operator allows for the easy copying of object properties, as shown below.

```
let object = {a: 1, b: 2}

let objectClone = Object.assign({}, object) // before ES9
let objectClone = {...object} // ES9 syntax

let otherObject = {c: 3, ...object}
console.log(otherObject) // -> {c: 3, a: 1, b: 2}
```

10th Edition – ECMAScript 2019 [\[edit \]](#)

The 10th edition, or ECMAScript 2019, was published in June 2019.^[15] Added features include, but are not limited to, `Array.prototype.flat`, `Array.prototype.flatMap`, changes to `Array.sort` and `Object.fromEntries`.^[15]

`Array.sort` is now guaranteed to be stable, meaning that elements with the same sorting precedence will appear in the same order in the sorted array. `Array.prototype.flat(depth=1)` flattens an array to a specified depth, meaning that all subarray elements (up to the specified depth) are concatenated recursively.

11th Edition – ECMAScript 2020 [\[edit \]](#)

The 11th edition, or ECMAScript 2020, was published in June 2020.^[16] In addition to new functions, this version introduces a `BigInt` primitive type for arbitrary-sized integers, the `nullish coalescing operator`, and the `globalThis` object.^[16]

https://en.wikipedia.org/wiki/ECMAScript#6th_Edition_-_ECMAScript_2015

Podemos inserir nossos códigos JavaScript das seguintes formas:

- Dentro do corpo da página `<body>....</body>`
- Dentro do cabeçalho `<head>....</head>`
- Dentro de um marcador html
- Em um arquivo separado do HTML: este arquivo deve ter a extensão **.js**

Se fizermos uma comparação com CSS, esse modo de inserir JavaScript em uma página, seria o modo incorporado.

Dentro do body ou do head, utilizamos a tag <script>

Ex:

```
<html>
<head>
  <title>Exemplo JavaScript</title>
  <script>
    alert("Eu estou no cabeçalho.");
  </script>
</head>
<body>
  <script>
    document.write("<strong class= 'teste'>Eu estou no corpo do documento.</strong>");
  </script>
</body>
</html>
```

OBS: Podemos colocar tags dentro das aspas inclusive com os atributos class e id.

Quando inserimos um código JavaScript em uma tag, este é associado a um evento. Os eventos serão vistos mais adiante, por enquanto faremos um exemplo simples com o evento onclick que é acionado quando o usuário clica no elemento (esse uso inline deve ser evitado).

Ex:

```
<html>
<head>
<title>Exemplo JavaScript</title>
</head>
<body>
<button type="button" onclick="alert('Olá.....')">Clique aqui</button>
</body>
</html>
```

Verifique o uso de
aspas duplas " e
simples '

Assim como em CSS, também podemos criar um arquivo separado do html com nossos códigos em JavaScript. Esse arquivo deve ser salvo com a extensão .js e é chamado no cabeçalho da página com a tag <script>

Ex:

```
<html>
<head>
<title>Exemplo JavaScript</title>
<script src="ex3.js"></script>
</head>
<body>
Conteúdo da página. Também podemos fazer aqui chamadas para blocos de
códigos do arquivo .js.
</body>
</html>
```

Código dentro do arquivo ex3.js

```
alert("Estou em um arquivo .js");
```

Mauricio SAMY Silva. **HTML 5 - A Linguagem de Marcação que revolucionou a WEB.** São Paulo: Novatec, 2011

MAURICIO SAMY SILVA. **Construindo Sites Com Css e (X)Html.** São Paulo: Novatec, 2007.

ERIC FREEMAN; ELISABETH FREEMAN. **Use a Cabeça! Html Com Css e Xhtml.** São Paulo: Alta Books, 2008.

ADOBE CREATIVE TEAM. **Dreamweaver Cs3 - Classroom In a Book.** São Paulo: Artmed, 2008.

<http://www.w3.org/TR/html5-diff/>

<http://www.whatwg.org/specs/web-apps/current-work/#is-this-html5>

http://www.w3schools.com/html/html5_intro.asp

<http://dev.w3.org/html5/spec/>

<http://www.w3.org/International/getting-started/language.pt-br.php?changelang=pt-br>

[http://msdn.microsoft.com/en-us/library/ms533052\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms533052(v=vs.85).aspx)

<http://www.w3c.br/Cursos/CursoCSS3>

<http://www.css3.info/selectors-test/>

<http://fmbip.com/>

[http://www.456bereastreet.com/archive/200601/css 3 selectors explained/](http://www.456bereastreet.com/archive/200601/css_3_selectors_explained/)

<http://www.findmebyip.com/litmus/>

<http://css3generator.com/>

http://www.quackit.com/css/css_color_codes.cfm

<http://www.w3schools.com/tags/default.asp>

<http://www.w3schools.com/cssref/default.asp>

<http://www.w3schools.com/html/default.asp>

<http://www.w3schools.com/css/default.asp>