

Todas as tags do HTML

- <!--...--> → Define um comentário
- <!DOCTYPE> → Define o tipo de documento
- <a> → Define um Hyperlink
- <address> → Define informação de contato do autor/dono do documento
- <area> → Define uma área dentro de um mapa de imagem
- <article> → Define um article
- <aside> → Define conteúdo diferente do conteúdo da página
- <audio> → Define conteúdo de áudio embutido
- → Define texto em negrito
- <blockquote> → Define uma secção que cita outra fonte
- <body> → Define o corpo do documento
-
 → Um único quebrador de linha
- <button> → Define um botão clicável
- <canvas> → Usado para desenhar gráfico ao vivo, por Script (Javascript)
- <caption> → Define uma legenda de uma tabela
- <cite> → Define um título de um trabalho
- <code> → Define um pedaço de um código de computador
- <col> → Define propriedades de coluna para coluna de um <colgroup>
- <colgroup> → Especifica um grupo de colunas num formato de tabela
- <data> → Adiciona uma tradução machine-readable de um dado conteúdo
- <datalist> → Especifica uma lista pré-definida de opções de input
- → Define um texto que foi deletado do documento
- <details> → Define detalhes adicionais que o usuário pode ver ou esconder
- <dialog> → Define uma caixa ou janela de diálogo
- <div> → Define uma seção do documento
- <dl> → Define uma descrição de lista

< dt > → Define um termo dentro de uma lista de descrição

< em > → Enfatiza texto

< embed > → Define um container para uma aplicação externa

< fieldset > → Grupo de elementos relacionados em um form

< figcaption > → Define o Título para um elemento de figura

< figure > → Especifica um conteúdo independente

< footer > → Define o rodapé de um documento

< form > → Define um Form HTML para receber Inputs do usuário

< h1 > to < h6 > → Define cabeças do HTML

< head > → Contém metadados/informação do documento

< header > → Define o cabeçalho do conteúdo

< hr > → Define a mudança temática de conteúdo

< html > → Define a raiz de um documento HTML

< i > → Define a parte do texto em uma voz/humor alternativo

< iframe > → Define um iframe em linha

< img > → Define uma imagem

< input > → Define um controle de input

< ins > → Define um texto que foi inserido em um documento

< kdb > → Define um input de teclado

< label > → Define uma label para um < input >

< legend > → Define uma rubrica para um < fieldset >

< li > → Define um item de uma lista

< link > → Define a relação entre o documento e uma fonte externa

< main > → Especifica o conteúdo principal de um documento

< map > → Define uma imagem mapa

< mark > → Define um texto em destaque

< meta > → Define metadata sobre o documento HTML

< meter > → Define uma medição escalar dentro de um intervalo conhecido

< nav > → Define links navegacionais

< object > → Define um container para uma aplicação externa

< ol > → Define uma lista não ordenada

< optgroup > → Define grupo de opções relacionadas em uma lista drop-down

< option > → Define uma opção em uma lista drop-down

< output > → Define o resultado de um cálculo

< p > → Define um parágrafo

< Picture > → Define um container para múltiplos recursos de imagem

< pre > → Define texto pré-formatado

< progress > → Representa o progresso de uma tarefa

< q > → Define uma citação pequena

< s > → Define um texto que não está mais em contexto

< sample > → Define um exemplo de um output de um programa

< script > → Define um script

< section > → Define uma seção de um documento

< select > → Define uma drop-down list

< small > → Define texto pequeno

< strong > → Define texto importante

< style > → Define estilo de informação de um documento

< sub > → Define um texto subscrito

< summary > → Define um cabeçalho visível para um elemento < details >

< sup > → Define um texto sobrescrito

< svg > → Define um container para gráficos SVG

< table > → Define uma tabela

< tbody > → Grupo de elementos de conteúdo de uma tabela

< td > → Célula de uma tabela

- < template > → Define um container para conteúdo que fica escondido quando abre a página
- < textarea > → Define múltiplos input de controle (área de texto)
- < tfoot > → Agrupa o rodapé de uma tabela
- < th > → Define o cabeçalho de uma célula da tabela
- < thead > → Agrupa o cabeçalho de uma tabela
- < time > → Define um tempo específico ou data
- < title > → Define um título para um documento
- < tr > → Define a fileira de uma tabela
- < track > → Define pistas de textos para elementos de mídia
- < u > → Define algum texto não articulado e estilo diferentes de um texto normal
- < ul > → Define uma lista não ordenada
- < var > → Define uma variável
- < video > → Define um vídeo
- < wbr > → Define um possível quebrador de linha

Resumo de CSS

Seletores

- Seletor de Elemento

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

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

- Seletor de Classe

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

- Seletor Universal (muda todos os elementos da página)

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

- Seletor de grupo

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

Cores

- Background-color

```
.classe{  
    background-color: DodgerBlue;  
}
```

- Cor do texto

```
.classe{  
    color: Tomato;  
}
```

- Cor da borda

```
.classe{  
    border: 2px solid Violet;  
}
```

Backgrounds

- background-color

```
body{  
    background-color: lightblue;  
}
```

- Opacidade/Transparência

```
div{  
    background-color: green;  
    opacity: 0.3 // 0 a 1, quanto menor, mais transparente  
}
```

- background-image

```
body{  
    background-image: url("paper.jpg");  
}
```

Bordas

- Estilos de borda

- dotted - Define uma borda pontilhada
- dashed - Define uma borda tracejada
- solid - Define uma borda sólida
- double - Define uma borda dupla
- groove - Define uma borda 3d sulcada
- ridge - Define uma borda 3d com cume
- inset - Define uma borda 3d intercalar
- outset - Define uma borda 3d outset
- none - Define nenhuma borda
- Hidden - Define uma borda escondida

```
#id{  
    border: 2px solid black  
}
```

- Cor da borda

```
.one{  
    border-color: red green blue yellow;  
    /* vermelho em cima, verde na direita  
    azul em baixo e amarelo na esquerda */  
}
```

Margin

```
.margem4{  
    margin: 25px 50px 75px 100px;  
    /* 25px em cima, 50px na direita  
    75px em baixo, 100px na esquerda */  
};  
  
.margem3{  
    margin: 25px 50px 75px;  
    /* 25px em cima, 50px na direita e  
    na esquerda, 75px em baixo */  
};  
  
.margem2{  
    margin: 25px 50px;  
    /* 25px em cima e em baixo,  
    50px na direita e na esquerda */  
};
```

- Valor auto
 - Você setar a margem para AUTO para centralizar o elemento horizontalmente no container que está contido. Ele pega o sobrou de espaço e divide igualmente entre a esquerda e a direita

Padding

```
.padding4{
    padding: 25px 50px 75px 100px;
    /* 25px em cima, 50px na direita
    75px em baixo, 100px na esquerda */
};

.padding3{
    padding: 25px 50px 75px;
    /* 25px em cima, 50px na direita e
    na esquerda, 75px em baixo */
};

.padding2{
    padding: 25px 50px;
    /* 25px em cima e em baixo,
    50px na direita e na esquerda */
};
```

Height and Width

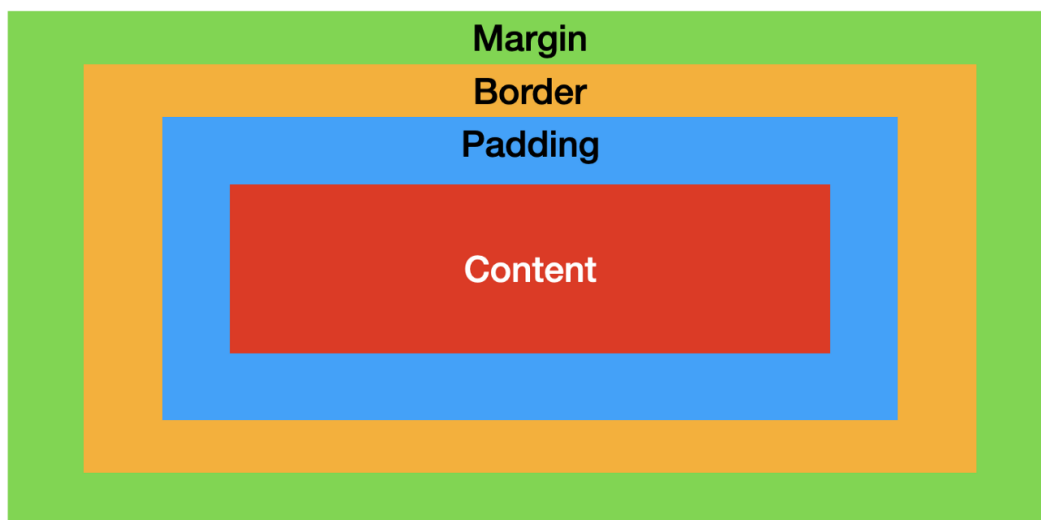
A altura e largura podem ser definidas com os seguintes valores:

- auto - Esse é default. O navegador calcula auto a altura e largura
- length - Define a altura/largura com px, cm, etc.
- % - Define a altura/largura com a porcentagem do bloco contido
- initial - Seta a altura/largura para o valor default
- inherit - A altura/largura é herdada do valor da tag pai

*Quando usar % é necessário que a tag pai tem altura/largura definida

- **max-width**
 - Serve para setar a máxima largura de um elemento, funciona bem para garantir a responsividade de uma página
- **min-width**
 - Serve para setar a mínima largura de um elemento, funciona bem para garantir a responsividade de uma página
- **max-height**
 - Serve para setar a máxima altura de um elemento, funciona bem para garantir a responsividade de uma página
- **min-height**
 - Serve para setar a mínima altura de um elemento, funciona bem para garantir a responsividade de uma página

Box Model



- **Content** - O conteúdo da box, onde o texto e imagens aparecem
- **Padding** - Espaçamento entre o conteúdo e a borda. O padding é transparente
- **Border** - Uma borda que vai envolta do padding e o conteúdo
- **Margin** - Espaçamento entre a área externa e a borda. A margin é transparente

Texto

- Color

```
body{  
    color: blue;  
}
```

- Alinhamento

- Alinha o texto de forma horizontal

```
h1 {  
    text-align: center;  
}
```

```
h2{  
    text-align: left;  
}
```

```
h3{  
    text-align: right;  
}
```

- A propriedade "justify" estica o texto para que toda linha tenha a mesma largura

```
div{  
    text-align: justify;  
}
```

- Propriedade que especifica o alinhamento da última linha do texto

```
p.a {  
    text-align-last: right;  
}
```

```
p.b{  
    text-align-last: center;  
}
```

- Propriedade que pode ser usada para mudar a direção do texto de um elemento

```
p{  
    direction: rtl;  
    unicode-bidi: bidi-override;  
}
```

- Propriedade que seta o alinhamento vertical de um elemento

```
img.a{  
    vertical-align: baseline;  
}
```

```
img.b{  
    vertical-align: text-top;  
}
```

```
img.c{  
    vertical-align: text-bottom;  
}
```

```
img.d{  
    vertical-align: sup;  
}
```

```
img.e{  
    vertical-align: super;  
}
```

- Decoração

- Propriedade usada para adicionar decoração de linha no texto

```
h1 {  
    text-decoration-line: overline;  
}
```

```
h2 {
    text-decoration-line: line-through;
}

h3 {
    text-decoration-line: underline;
}

p {
    text-decoration-line: overline underline;
}
```

- Propriedade usada para setar o cor linha de decoração

```
h1 {
    text-decoration-line: overline;
    text-decoration-color: red;
}

h2 {
    text-decoration-line: line-through;
    text-decoration-color: blue;
}

h3 {
    text-decoration-line: underline;
    text-decoration-color: green;
}

p {
    text-decoration-line: overline underline;
    text-decoration-color: purple;
}
```

- Propriedade usada para setar o estilo da linha de decoração

```
h1 {
    text-decoration-line: underline;
    text-decoration-style: solid;
```

```
}  
h2 {  
  text-decoration-line: underline;  
  text-decoration-style: double;  
}
```

```
h3 {  
  text-decoration-line: underline;  
  text-decoration-style: dotted;  
}
```

```
p.ex1 {  
  text-decoration-line: underline;  
  text-decoration-style: dashed;  
}
```

```
p.ex2 {  
  text-decoration-line: underline;  
  text-decoration-style: wavy;  
}
```

```
p.ex3 {  
  text-decoration-line: underline;  
  text-decoration-color: red;  
  text-decoration-style: wavy;  
}
```

- Propriedade usada para setar a grossura da linha de decoração

```
h1 {  
  text-decoration-line: underline;  
  text-decoration-thickness: auto;  
}
```

```
h2 {  
    text-decoration-line: underline;  
    text-decoration-thickness: 5px;  
}  
  
h3 {  
    text-decoration-line: underline;  
    text-decoration-thickness: 25%;  
}  
  
p {  
    text-decoration-line: underline;  
    text-decoration-color: red;  
    text-decoration-style: double;  
    text-decoration-thickness: 5px;  
}
```

- Espaçamento

- Propriedade usada para especificar a indentação da primeira linha do texto

```
p {  
    text-indent: 50px;  
}
```

- Propriedade usada para determinar o espaçamento entre os caracteres

```
h1 {  
    letter-spacing: 5px;  
}
```

```
h2 {  
    letter-spacing: -2px;
```

```
}
```

- Propriedade usada para determinar o espaçamento entre linhas

```
p.small {
```

```
    line-height: 0.8;
```

```
}
```

```
p.big {
```

```
    line-height: 1.8;
```

```
}
```

- Propriedade usada para determinar o espaçamento entre as palavras

```
p.one {
```

```
    word-spacing: 10px;
```

```
}
```

```
p.two {
```

```
    word-spacing: -2px;
```

```
}
```

- Sombreamento

```
h1 {
```

```
    text-shadow: 2px 2px red;
```

```
}
```

Fonte

- Família

```
.p1 {
```

```
    font-family: "Times New Roman", Times, serif;
```

```
}
```

```
.p2 {
```

```
    font-family: Arial, Helvetica, sans-serif;
```

```
}  
.p3 {  
    font-family: "Lucida Console", "Courier New", monospace;  
}
```

- Tamanho

```
h1 {  
    font-size: 40px;  
}  
h2 {  
    font-size: 30px;  
}  
p {  
    font-size: 14px;  
}
```

Link

- Estilizando Links

```
/* unvisited link */  
a:link {  
    color: red;  
}  
/* visited link */  
a:visited {  
    color: green;  
}  
/* mouse over link */  
a:hover {  
    color: hotpink;
```



```
}  
/* selected link */
```

```
a:active {  
    color: blue;  
}
```

- Decoração dos Links

```
a:link {  
    text-decoration: none;  
}  
a:visited {  
    text-decoration: none;  
}  
a:hover {  
    text-decoration: underline;  
}  
a:active {  
    text-decoration: underline;  
}
```

- Botões de Links

```
a:link, a:visited {  
    background-color: #f44336;  
    color: white;  
    padding: 14px 25px;  
    text-align: center;  
    text-decoration: none;  
    display: inline-block;  
}  
a:hover, a:active {  
    background-color: red;  
}
```

Listas

Unordered Lists:

- Coffee
- Tea
- Coca Cola

- Coffee
- Tea
- Coca Cola

Ordered Lists:

1. Coffee
2. Tea
3. Coca Cola

- I. Coffee
- II. Tea
- III. Coca Cola

- Propriedade que especifica o tipo de marcador da lista

```
ul.a {  
  list-style-type: circle;  
}  
ul.b {  
  list-style-type: square;  
}  
ol.c {  
  list-style-type: upper-roman;  
}  
ol.d {  
  list-style-type: lower-alpha;  
}
```

- Uma imagem com marcador de lista

```
ul {  
  list-style-image: url('sqpurple.gif');  
}
```

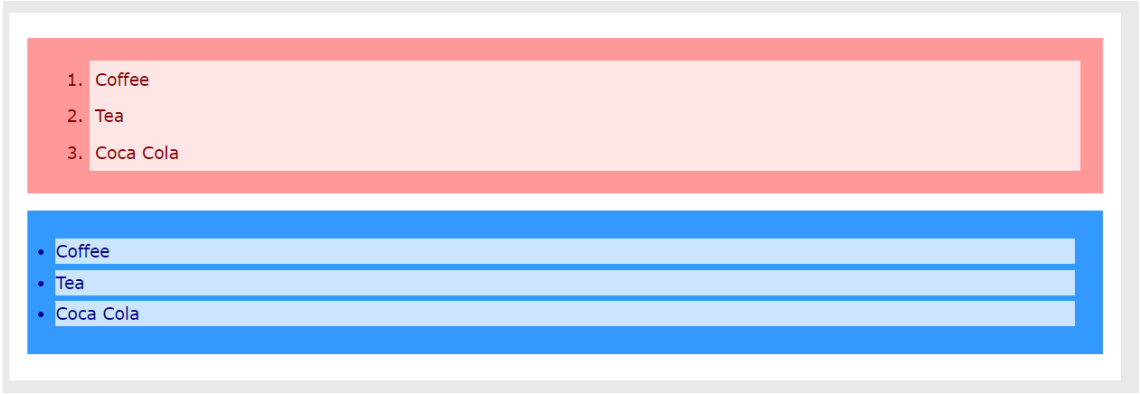
- Propriedade que especifica a posição dos marcadores de lista

```
ul.a {  
  list-style-position: outside;  
}  
ul.b {  
  list-style-position: inside;  
}
```

- Remover configurações default

```
ul {
```

- ```
list-style-type: none;
margin: 0;
padding: 0;
}
• Estilizando listas com cores
ol {
 background: #ff9999;
 padding: 20px;
}
ul {
 background: #3399ff;
 padding: 20px;
}
ol li {
 background: #ffe5e5;
 color: darkred;
 padding: 5px;
 margin-left: 35px;
}
ul li {
 background: #cce5ff;
 color: darkblue;
 margin: 5px;
}
```

- 
1. Coffee
  2. Tea
  3. Coca Cola

- Coffee
- Tea
- Coca Cola

# Tabelas

- Bordas

| Firstname | Lastname |
|-----------|----------|
| Peter     | Griffin  |
| Lois      | Griffin  |

```
table, th, td {
 border: 1px solid;
}
```

- Tabela que ocupa 100% da largura

| Firstname | Lastname |
|-----------|----------|
| Peter     | Griffin  |
| Lois      | Griffin  |

```
table {
 width: 100%;
}
```

- Borda colapsada

| Firstname | Lastname |
|-----------|----------|
| Peter     | Griffin  |
| Lois      | Griffin  |

```
table {
 border-collapse: collapse;
}
```

- Se você colocar borda apenas na tag <table> esse será o resultado

| Firstname | Lastname |
|-----------|----------|
| Peter     | Griffin  |
| Lois      | Griffin  |

```
table {
```

border: 1px solid;

}

- Altura e largura

| Firstname | Lastname | Savings |
|-----------|----------|---------|
| Peter     | Griffin  | \$100   |
| Lois      | Griffin  | \$150   |
| Joe       | Swanson  | \$300   |

table {

width: 100%;

}

th {

height: 70px;

}

| Firstname | Lastname | Savings |
|-----------|----------|---------|
| Peter     | Griffin  | \$100   |
| Lois      | Griffin  | \$150   |
| Joe       | Swanson  | \$300   |

table {

width: 50%;

}

- Alinhamento
  - Horizontal

| Firstname | Lastname | Savings |
|-----------|----------|---------|
| Peter     | Griffin  | \$100   |
| Lois      | Griffin  | \$150   |
| Joe       | Swanson  | \$300   |

td {

text-align: center;

}

| Firstname | Lastname | Savings |
|-----------|----------|---------|
| Peter     | Griffin  | \$100   |
| Lois      | Griffin  | \$150   |
| Joe       | Swanson  | \$300   |

```
th {
 text-align: left;
}
```

- Vertical

| Firstname | Lastname | Savings |
|-----------|----------|---------|
| Peter     | Griffin  | \$100   |
| Lois      | Griffin  | \$150   |
| Joe       | Swanson  | \$300   |

```
td {
 height: 50px;
 vertical-align: bottom;
}
```

- Estilos
  - Padding

| First Name | Last Name | Savings |
|------------|-----------|---------|
| Peter      | Griffin   | \$100   |
| Lois       | Griffin   | \$150   |
| Joe        | Swanson   | \$300   |

```
th, td {
 padding: 15px;
 text-align: left;
}
```

- Divisores Horizontais

| First Name | Last Name | Savings |
|------------|-----------|---------|
| Peter      | Griffin   | \$100   |
| Lois       | Griffin   | \$150   |
| Joe        | Swanson   | \$300   |

```
th, td {
 border-bottom: 1px solid #ddd;
}
```

- Hover Table

| First Name | Last Name | Savings |
|------------|-----------|---------|
| Peter      | Griffin   | \$100   |
| Lois       | Griffin   | \$150   |
| Joe        | Swanson   | \$300   |

```
tr:hover {background-color: coral;}
```

- Tabelas sem linhas

| First Name | Last Name | Savings |
|------------|-----------|---------|
| Peter      | Griffin   | \$100   |
| Lois       | Griffin   | \$150   |
| Joe        | Swanson   | \$300   |

```
tr:nth-child(even) {background-color: #f2f2f2;}
```

- Cor de Tabela

| First Name | Last Name | Savings |
|------------|-----------|---------|
| Peter      | Griffin   | \$100   |
| Lois       | Griffin   | \$150   |
| Joe        | Swanson   | \$300   |

```
th {
 background-color: #04AA6D;
 color: white;
}
```

# Display

| Value        | Description                                                                                                                                             | Play it                   |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|
| inline       | Displays an element as an inline element (like <span>). Any height and width properties will have no effect                                             | <a href="#">Demo &gt;</a> |
| block        | Displays an element as a block element (like <p>). It starts on a new line, and takes up the whole width                                                | <a href="#">Demo &gt;</a> |
| contents     | Makes the container disappear, making the child elements children of the element the next level up in the DOM                                           |                           |
| flex         | Displays an element as a block-level flex container                                                                                                     |                           |
| grid         | Displays an element as a block-level grid container                                                                                                     |                           |
| inline-block | Displays an element as an inline-level block container. The element itself is formatted as an inline element, but you can apply height and width values |                           |

- Esconder um elemento

```
h1.hidden {
 display: none;
}

h1.hidden {
 visibility: hidden;
}
```

## Posição

- Static
  - É sempre posicionada de acordo com o flow da pagina, não é afetada por propriedades top, bottom, left e right

```
div.static {
 position: static;
 border: 3px solid #73AD21;
}
```

- Relative
  - Posicionada de acordo com sua posição relativa normal, afetada por propriedades top, bottom, left e right

```
div.relative {
```



```
position: relative;
left: 30px;
border: 3px solid #73AD21;
}
```

- Fixed

- Posição relativa com o viewport, fica sempre no mesmo lugar mesmo que página seja escrolada

```
div.fixed {
position: fixed;
bottom: 0;
right: 0;
width: 300px;
border: 3px solid #73AD21;
}
```

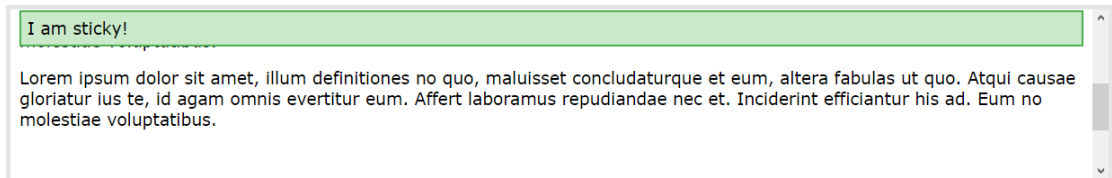
- Absolute

- É posicionada de acordo com relativamente com o ancestral posicionado mais próximo

```
div.absolute {
position: absolute;
top: 80px;
right: 0;
width: 200px;
height: 100px;
border: 3px solid #73AD21;
}
```

- Sticky

- É posicionado baseado na posição do scroll do usuário



```
div.sticky {

 position: -webkit-sticky; /* Safari */

 position: sticky;

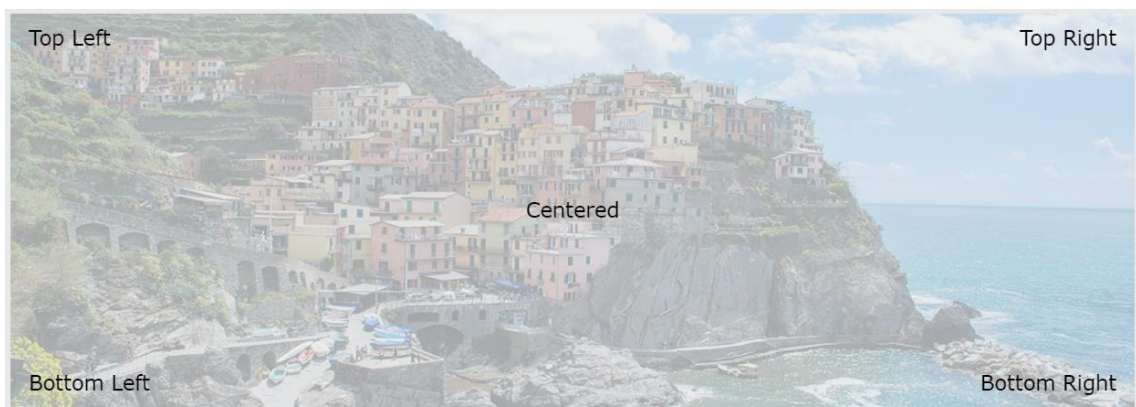
 top: 0;

 background-color: green;

 border: 2px solid #4CAF50;

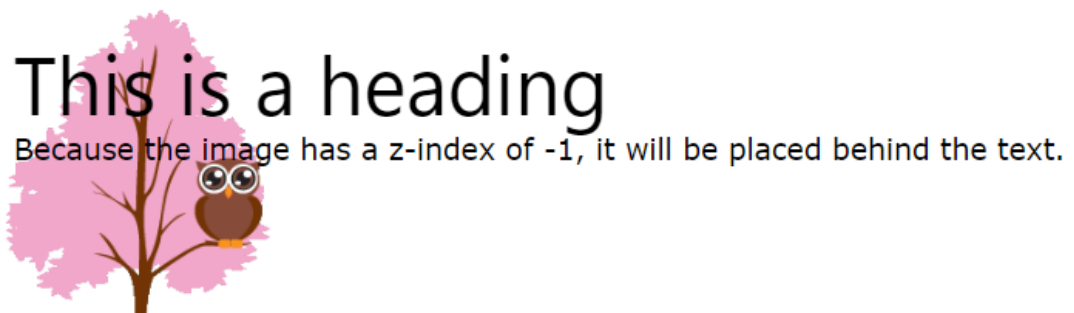
}
```

- Texto em imagem



## Z-Index

- Serve para posicionar um elemento sobre o outro



```
img {

 position: absolute;
```

```
left: 0px;
top: 0px;
z-index: -1;
}
```

## Overflow

- Visível

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

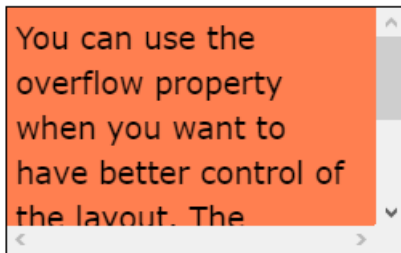
```
div {
 width: 200px;
 height: 65px;
 background-color: coral;
 overflow: visible;
}
```

- Escondido

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what

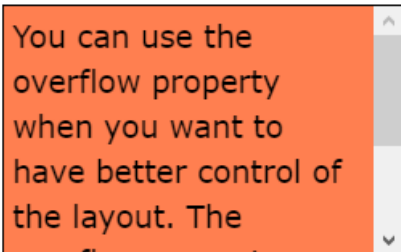
```
div {
 overflow: hidden;
}
```

- Scroll



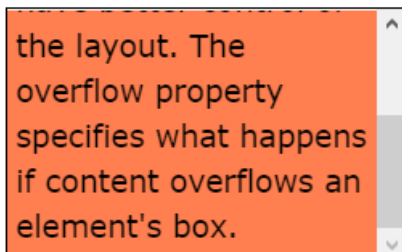
```
div {
 overflow: scroll;
}
```

- Auto



```
div {
 overflow: auto;
}
```

- Overflow-x e Overflow-y



```
div {
 overflow-x: hidden; /* Hide horizontal scrollbar */
 overflow-y: scroll; /* Add vertical scrollbar */
}
```

## Float

- `left` - The element floats to the left of its container
- `right` - The element floats to the right of its container
- `none` - The element does not float (will be displayed just where it occurs in the text). This is default
- `inherit` - The element inherits the float value of its parent

## Alinhamento Vertical e Horizontal

- Center Align Elements

This div element is centered.

```
.center {
 margin: auto;
 width: 50%;
 border: 3px solid green;
 padding: 10px;
}
```

- Center Aling Text

This text is centered.

```
.center {
 text-align: center;
 border: 3px solid green;
}
```

- Center na Image

```
img {
 display: block;
 margin-left: auto;
 margin-right: auto;
```

```
width: 40%;
}
```

- Left and Right Aling - Using position

```
.right {
 position: absolute;
 right: 0px;
 width: 300px;
 border: 3px solid #73AD21;
 padding: 10px;
}
```

- Left and Right Aling - Using float

```
.right {
 float: right;
 width: 300px;
 border: 3px solid #73AD21;
 padding: 10px;
}
```

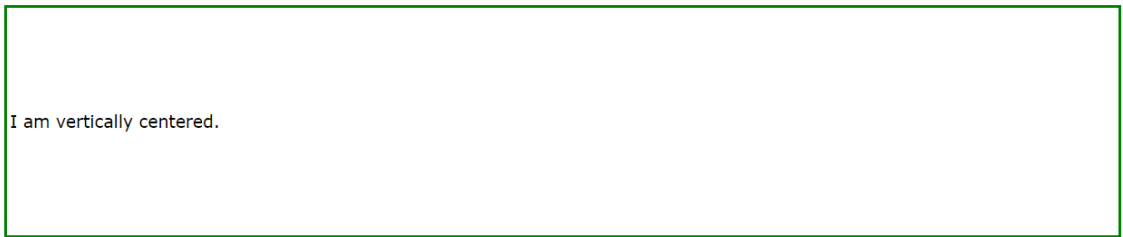
- The clearfix Hack



```
.clearfix::after {
 content: "";
 clear: both;
 display: table;
```

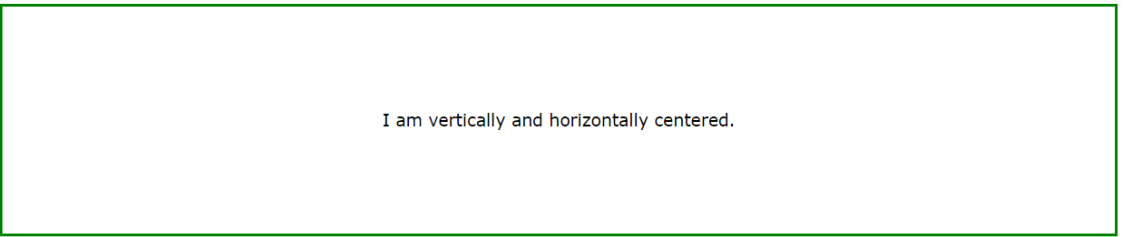
```
}
```

- Center Vertically - Using padding



I am vertically centered.

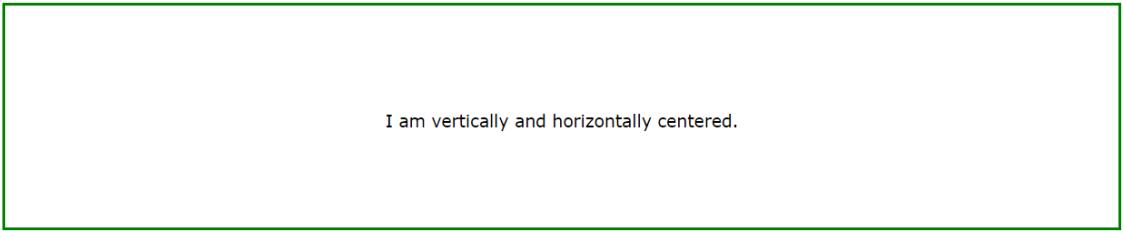
```
.center {
 padding: 70px 0;
 border: 3px solid green;
}
```



I am vertically and horizontally centered.

```
.center {
 padding: 70px 0;
 border: 3px solid green;
 text-align: center;
}
```

- Center Vertically - Using line-height

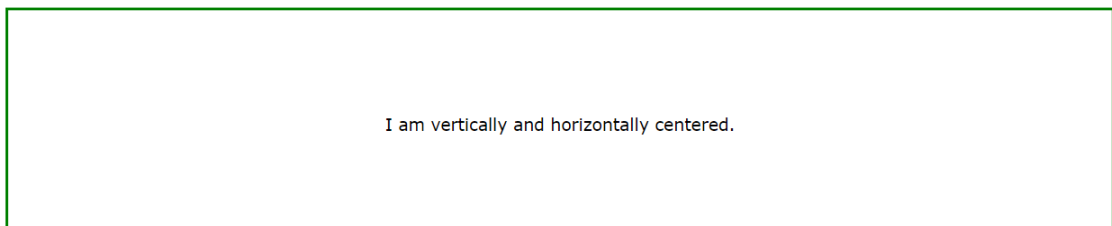


I am vertically and horizontally centered.

```
.center {
 line-height: 200px;
```

```
height: 200px;
border: 3px solid green;
text-align: center;
}
/* If the text has multiple lines, add the following: */
.center p {
 line-height: 1.5;
 display: inline-block;
 vertical-align: middle;
}
```

- Center Vertically - Using position and transform



```
.center {
 height: 200px;
 position: relative;
 border: 3px solid green;
}
.center p {
 margin: 0;
 position: absolute;
 top: 50%;
 left: 50%;
 transform: translate(-50%, -50%);
}
```



- Center Vertically - Using Flexbox

I am vertically and horizontally centered.

```
.center {
 display: flex;
 justify-content: center;
 align-items: center;
 height: 200px;
 border: 3px solid green;
}
```

## @media Queries

```
<!-- CSS media query dentro de um stylesheet -->
@media (max-width: 600px)
{
 .facet_sidebar
 {
 display: none;
 }
}
```

```
@media tv and (min-width: 700px) and (orientation: landscape) { ... }
```

```
@media (min-width: 700px), handheld and (orientation: landscape) { ... }
```

```
@media not all and (monochrome) { ... }
```

Isto significa que a *query* é avaliada assim:

```
@media not (all and (monochrome)) { ... }
```

... em vez disso:

```
@media (not all) and (monochrome) { ... }
```

```
media_query_list: <media_query> [, <media_query>]*
media_query: [[only | not]? <media_type> [and <expression>]*
 | <expression> [and <expression>]*
expression: (<media_feature> [: <value>]?)
media_type: all | aural | braille | handheld | print |
 projection | screen | tty | tv | embossed
media_feature: width | min-width | max-width
 | height | min-height | max-height
 | device-width | min-device-width | max-device-width
 | device-height | min-device-height | max-device-height
 | aspect-ratio | min-aspect-ratio | max-aspect-ratio
 | device-aspect-ratio | min-device-aspect-ratio | max-device-aspect-ratio
 | color | min-color | max-color
 | color-index | min-color-index | max-color-index
 | monochrome | min-monochrome | max-monochrome
 | resolution | min-resolution | max-resolution
 | scan | grid
```

*Media queries* são *case insensitive*. *Media queries* envolvidas em *media types* desconhecidos serão sempre falsas.





# Resumo das formas de aplicação do CSS

## Resumo de HTML

Para começar vamos fazer uma breve recapitulação sobre para que serve o CSS (Cascading Style Sheets). Mas antes disso, para entender a funcionalidade do CSS primeiro precisamos entender o que é o HTML (HyperText Markup Language). HTML nada mais é do que uma forma de escrever informações em um arquivo por meio de tags. Cada tag começa com um nome entre sinais de maior e menor **<nomeDaTag>** e termina igual, mas com um barra na frente **</nomeDaTag>**. O conteúdo de cada tag fica entre esse começo e fim. Segue um exemplo abaixo:

```
<html>
 <title>Conteúdo</title>
</html>
```

Para que o seu navegador de internet possa saber o que ele deve exibir quando se abre uma página na internet, ele precisa de um arquivo que ele possa ler contendo essas informações. Esse arquivo é composto de HTML, CSS e JavaScript. O HTML é o esqueleto da página, nele é escrito tudo que a página contém.

## HTML Prática

Para demonstrar a funcionalidade do HTML vamos fazer um teste prático.

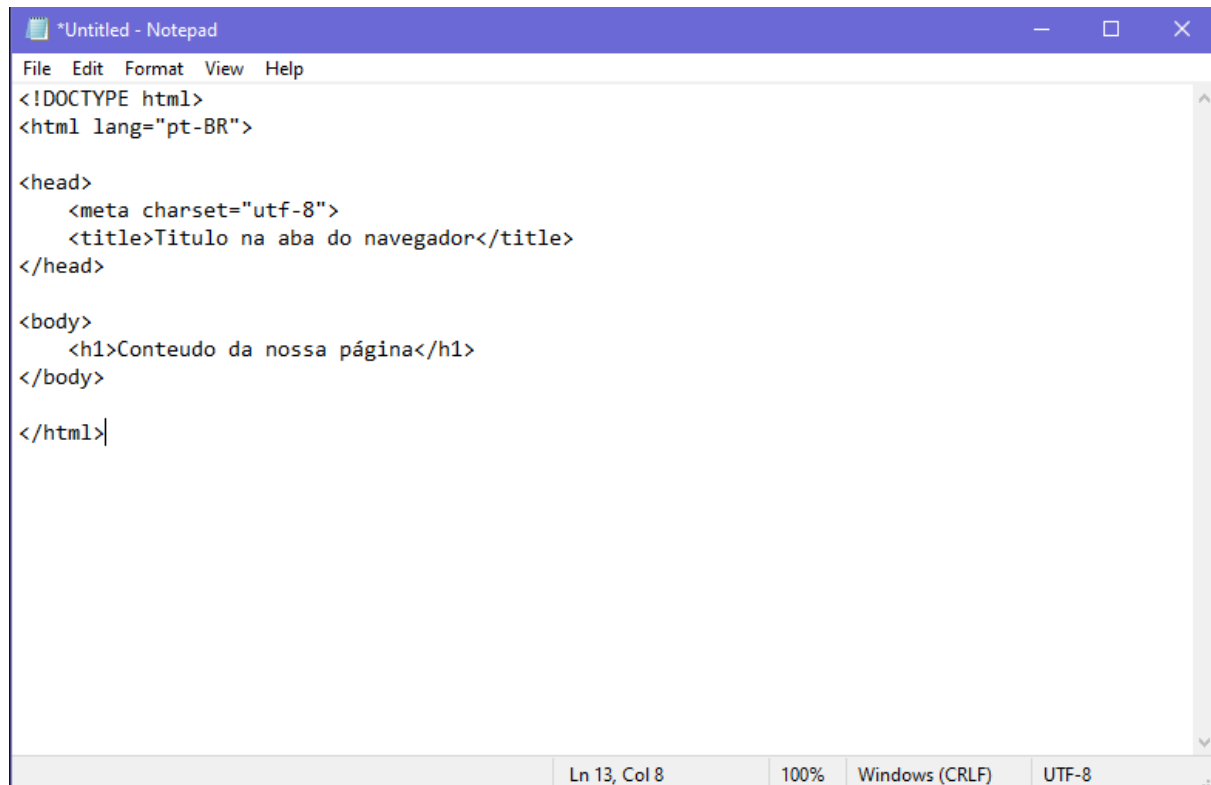
Abra o bloco de notas e escreva o texto abaixo:

```
<!DOCTYPE html>
<html lang="pt-BR">

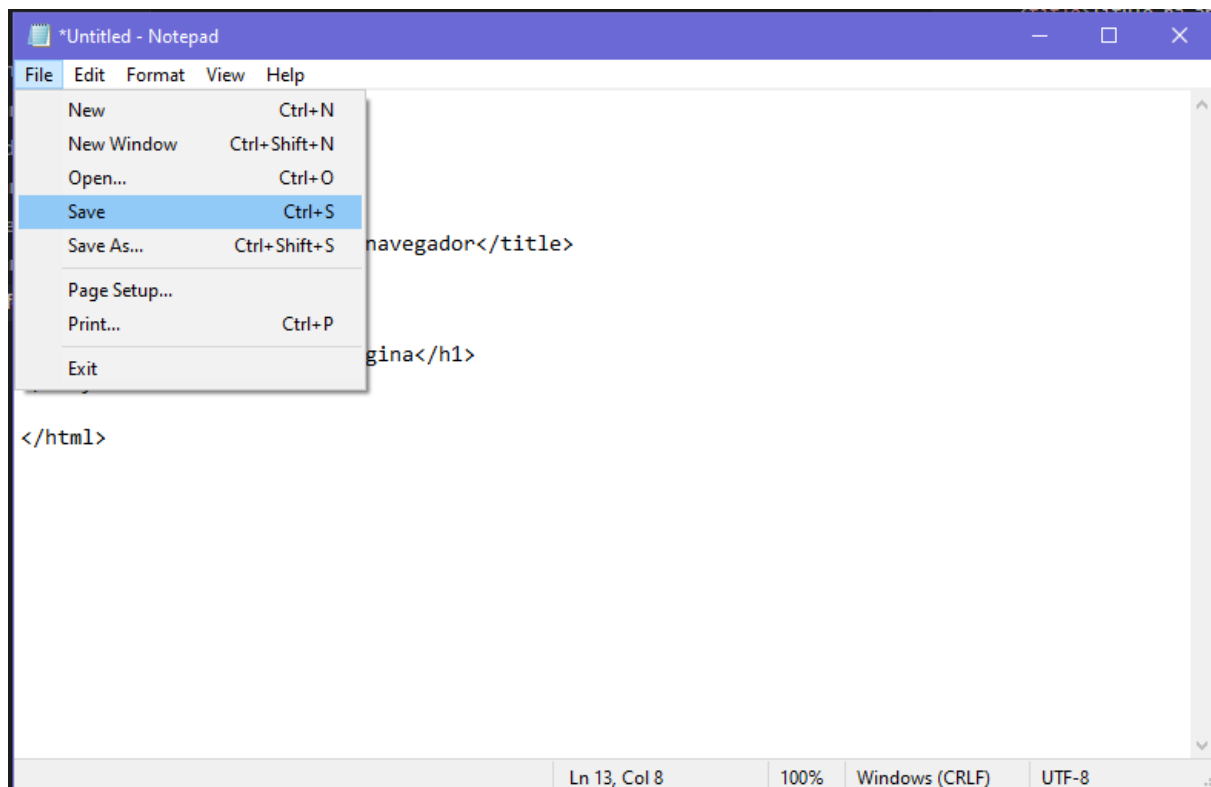
<head>
 <meta charset="utf-8">
 <title>Titulo na aba do navegador</title>
</head>
```

```
<body>
 <h1>Conteudo da nossa página</h1>
</body>

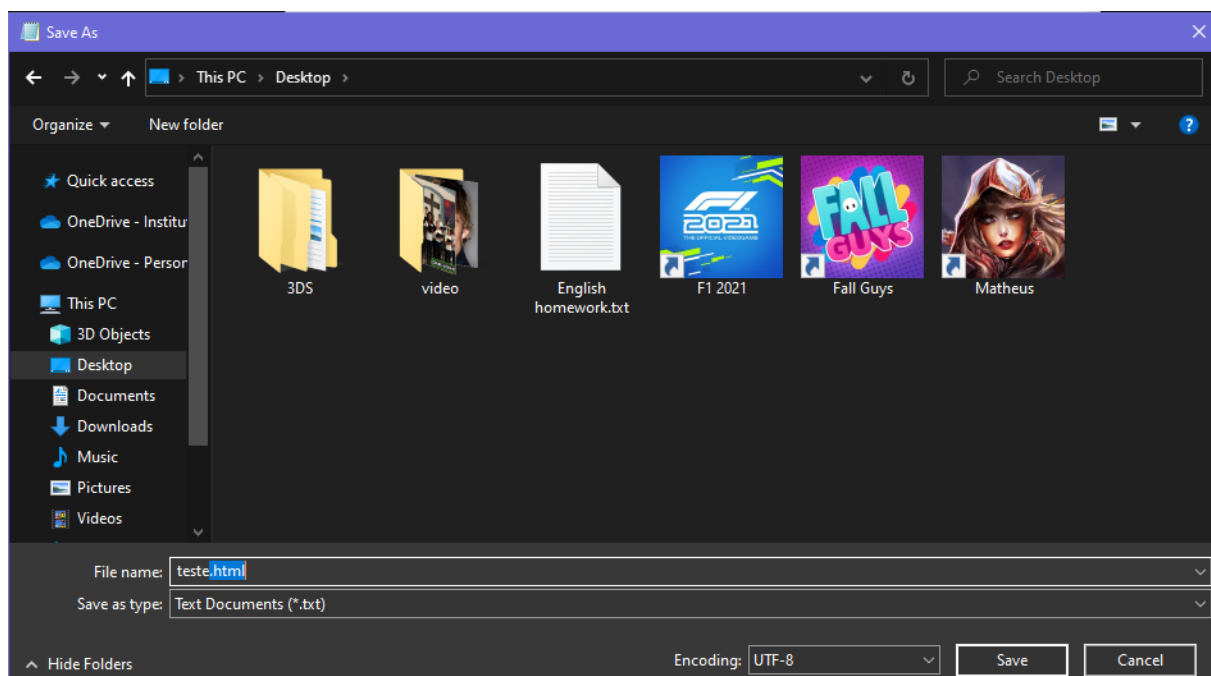
</html>
```



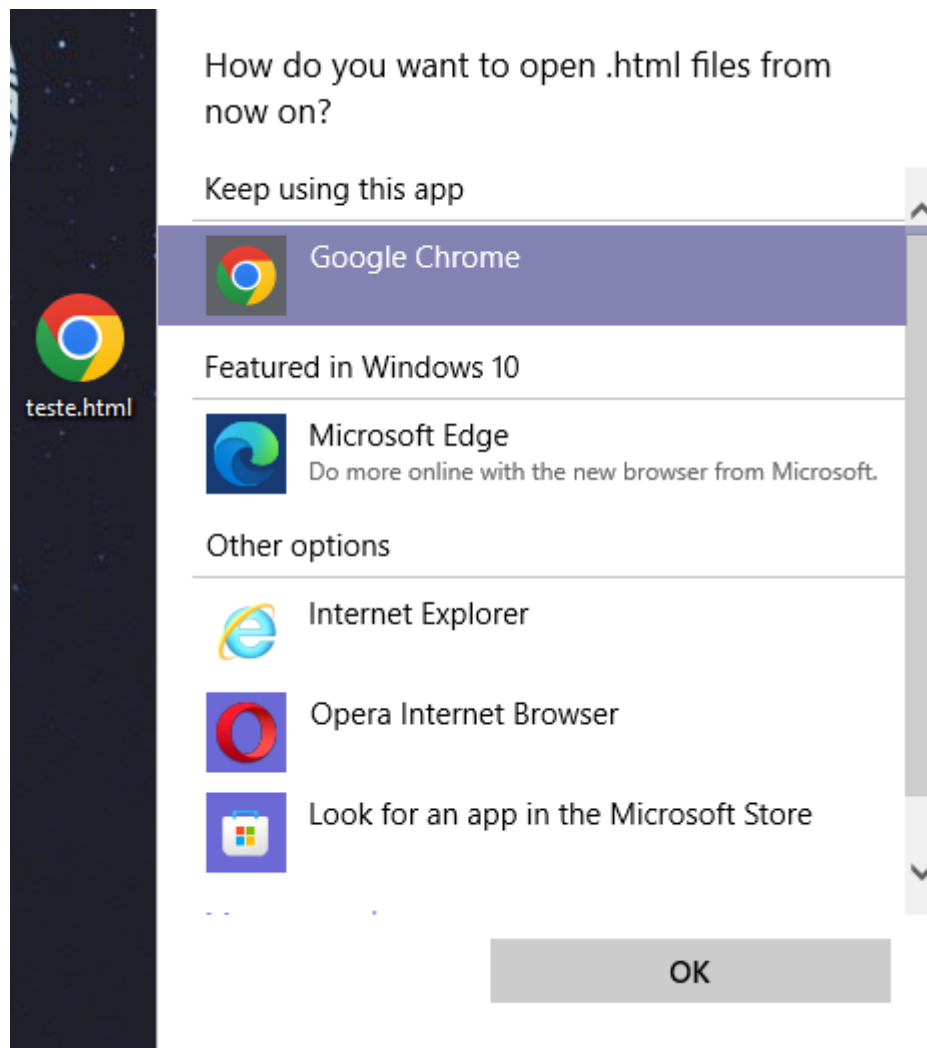
Clique em **File** ou **Arquivo** na parte superior esquerda do bloco de notas e depois clique em **Salvar**.



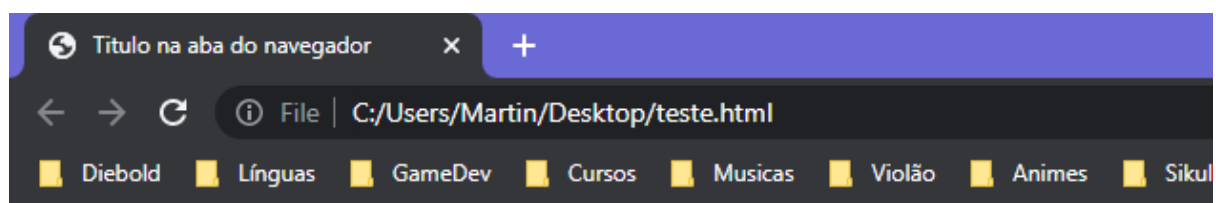
Mude a extensão do arquivo para **.html** e clique em **Save** ou **Salvar**.



Agora abra o arquivo com qualquer navegador de sua preferência.



Como podemos ver o navegador interpretou o arquivo **html** e exibiu todo o conteúdo que estava descrito nele.



## Conteúdo da nossa página

# Resumo de CSS

Agora que a página contém todo o conteúdo que precisamos, seria bom se tivesse um jeito para que desse para personalizar a nossa página, mudar seu **estilo**. É aí que entra o **CSS**. Com o CSS conseguimos escrever informações para que o navegador entenda como ele deve customizar o conteúdo da nossa página. O CSS serve para controlar a aparência da página. Existem três maneiras diferentes para se aplicar o CSS.

## CSS inline

A primeira forma de escrever CSS, e a mais simples, é o escrevendo diretamente nas tags HTMLs que você deseja personalizar. Toda tag possui um atributo **style** no qual você pode escrever código CSS, como no exemplo a seguir:

```
<h1 style="color: aqua;">Conteúdo da nossa página</h1>
```

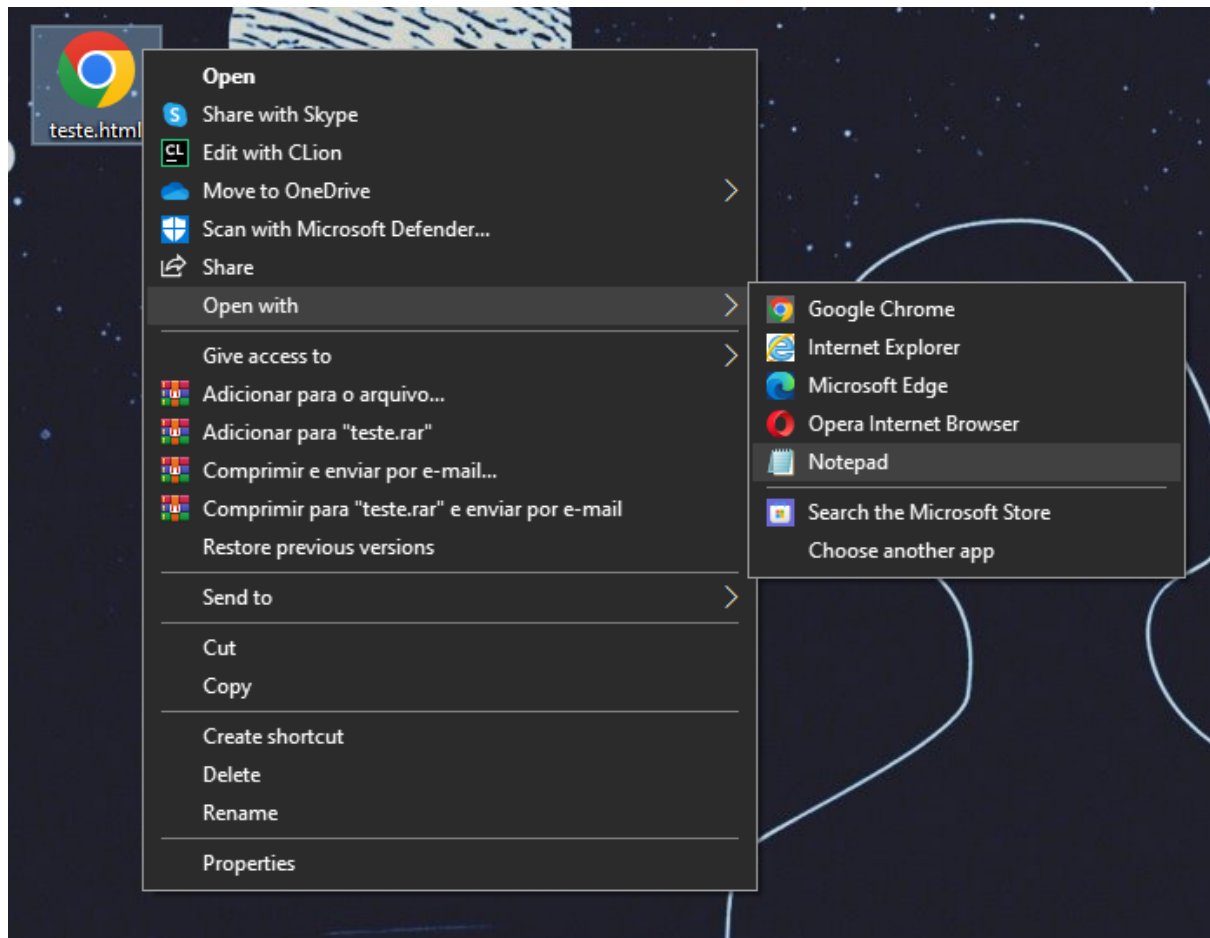
- **color** é uma propriedade CSS
- **aqua** é um valor associado à propriedade color
- **color: blue;** escritos desta forma, caracterizam uma regra CSS

## CSS inline prática

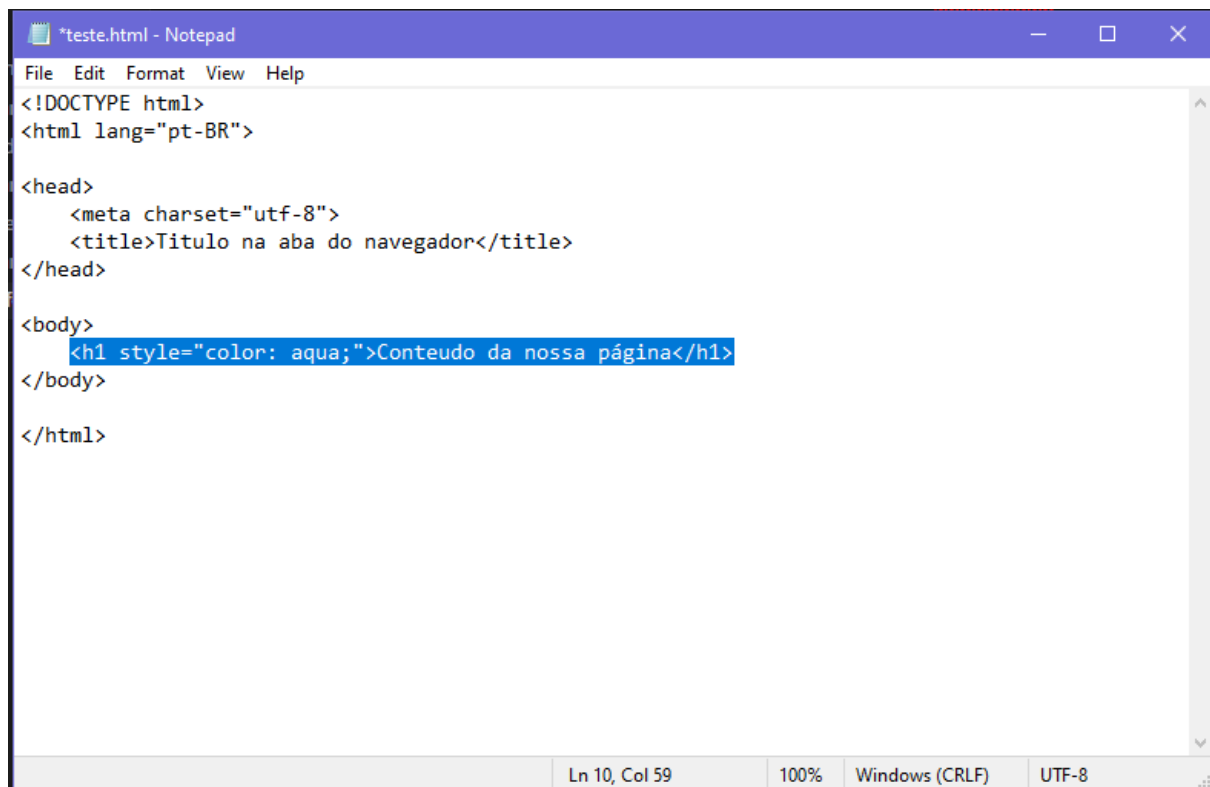
Se você alterar o arquivo utilizado anteriormente no exemplo de prática de html, colocando style, você verá que a cor do conteúdo da tag h1 irá mudar de cor preta para aqua.

Abra o arquivo teste.html novamente com o bloco de notas.





Altere a tag **h1** adicionando a propriedade style.



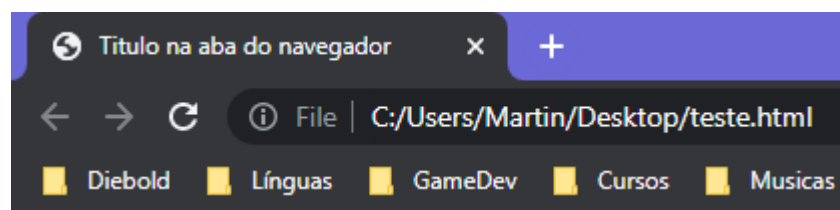
```
<!DOCTYPE html>
<html lang="pt-BR">

<head>
 <meta charset="utf-8">
 <title>Titulo na aba do navegador</title>
</head>

<body>
 <h1 style="color: aqua;">Conteudo da nossa página</h1>
</body>

</html>
```

Salve e abra o arquivo novamente com o navegador.



Conteudo da nossa página

Podemos ver que a cor do conteúdo de h1 mudou.

## CSS usando tag style

Com o CSS inline conseguimos customizar tags separadamente de forma simples. Como no exemplo a seguir.

```
<!DOCTYPE html>
<html lang="pt-BR">

<head>
 <meta charset="utf-8">
```

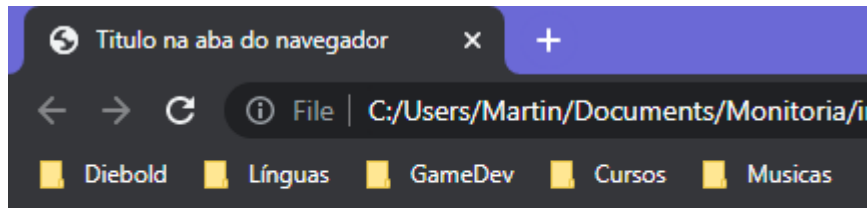
```

<title>Titulo na aba do navegador</title>
</head>

<body>
 <h1 style="color: aqua;">Conteudo da nossa página 1 </h1>
 <h1 style="color: green;">Conteudo da nossa página 2 </h1>
 <h1 style="color: red;">Conteudo da nossa página 3 </h1>
</body>

</html>

```



Conteudo da nossa página 1

Conteudo da nossa página 2

Conteudo da nossa página 3

Mas agora imagine que você precise criar uma página com várias tags **h1** e que tivessem todos a mesma cor. Você teria que escrever o CSS inline em cada tag. E caso você queira alterar essa cor em algum momento, também seria necessário reescrever o CSS inline em cada tag.

Para resolver esse problema é possível se utilizar da tag HTML **style**. Dentro dessa tag é possível escrever código CSS específico para várias tags ou para uma única tag. A seguir um exemplo utilizando-se o **CSS inline** e outro com a tag **style**.

## CSS inline

```

<!DOCTYPE html>
<html lang="pt-BR">

<head>
 <meta charset="utf-8">
 <title>Titulo na aba do navegador</title>
</head>

```

```

<body>
 <h1 style="color: aqua;">Conteudo da nossa página 1 </h1>
 <h1 style="color: aqua;">Conteudo da nossa página 2 </h1>
 <h1 style="color: aqua;">Conteudo da nossa página 3 </h1>
</body>

</html>

```

## CSS com tag style

```

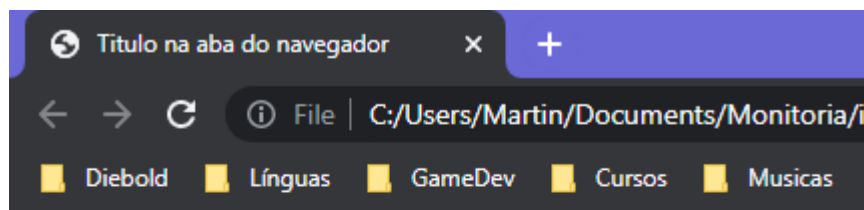
<!DOCTYPE html>
<html lang="pt-BR">

<head>
 <meta charset="utf-8">
 <title>Titulo na aba do navegador</title>
 <style>
 h1 {
 color: aqua;
 }
 </style>
</head>

<body>
 <h1> Conteudo da nossa página 1 </h1>
 <h1> Conteudo da nossa página 2 </h1>
 <h1> Conteudo da nossa página 3 </h1>
</body>

</html>

```



Conteudo da nossa página 1

Conteudo da nossa página 2

Conteudo da nossa página 3

Como podemos ver o resultado é o mesmo nos dois casos, porém em caso de uma alteração de cor ou adição de mais tags **h1** de mesma cor, o trabalho utilizando a tag **style** é muito menor do que utilizando a o **CSS inline** em casa tag.

## CSS em arquivo separado.

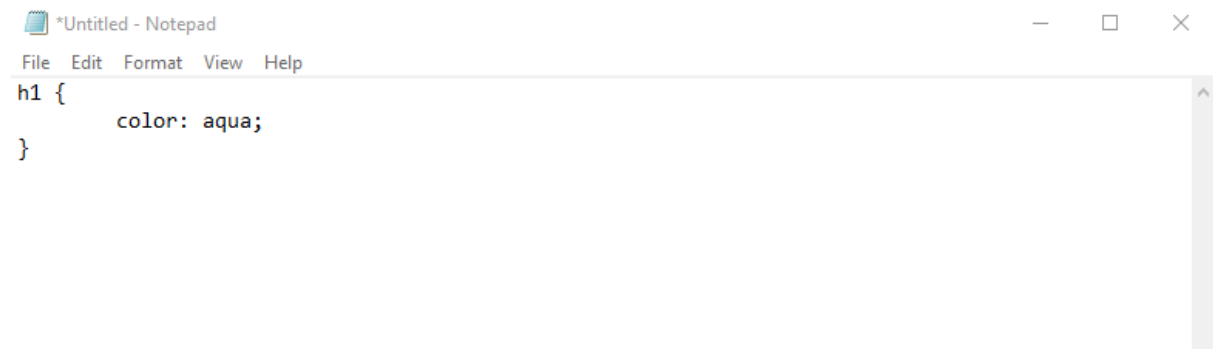
O CSS usando a tag style facilita a reutilização da customização em vários conjuntos de tags, fazendo com que não seja necessário reescrever o código várias vezes. Porém deixar o código CSS misturado com o código HTML tudo no mesmo arquivo acaba deixando ele com muito informação e deixa mais complicada a leitura e o entendimento.

Por isso para deixar o código melhor organizado e mais legível conseguimos separar o CSS da tag style para um arquivo separado.

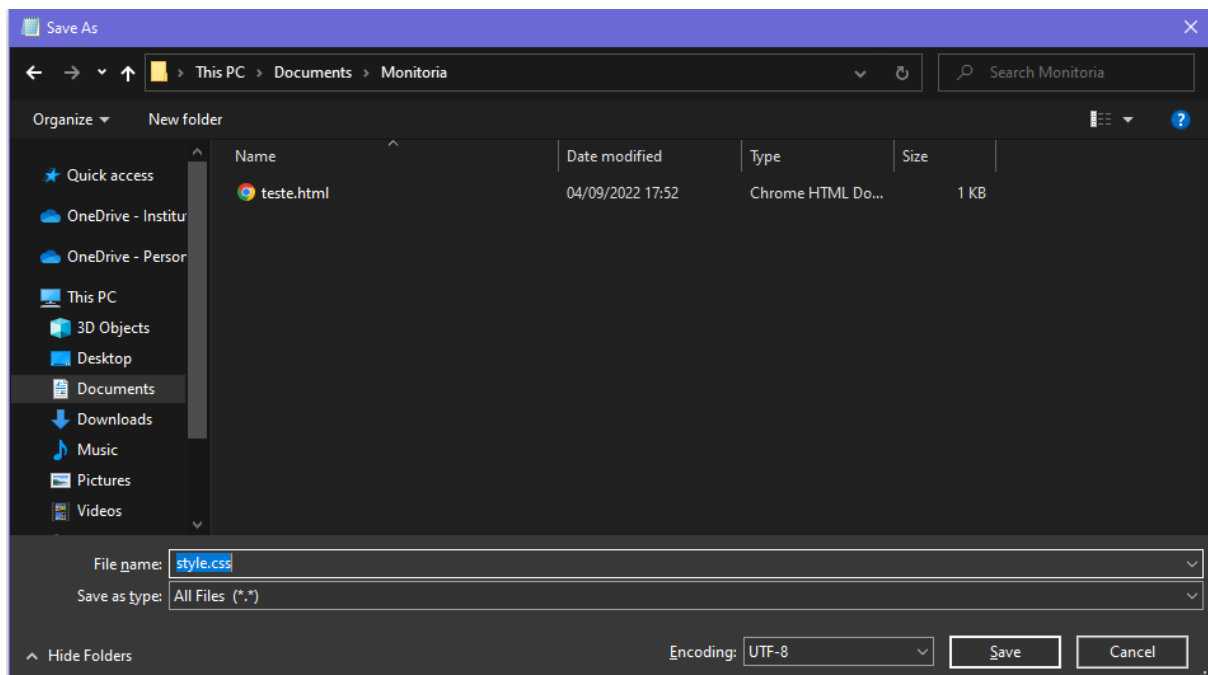
## CSS em arquivo separado prática

Abre um novo bloco de notas e cole o código CSS abaixo:

```
h1 {
 color: aqua;
}
```



Salve ele com o nome **style.css** na mesma pasta que o arquivo **teste.html** criado anteriormente:



Agora que escrevemos o código CSS em um arquivo separado **style.css**, é necessário fazer uma referência a ele no arquivo **teste.html** para que ele saiba onde pegar as informações de CSS.

Abra o arquivo teste.html com o bloco de notas e adicione a **linha de referência** para o arquivo style.css.

### Linha de referência:

```
<link rel="stylesheet" href="./style.css">
```

## Arquivo teste.html

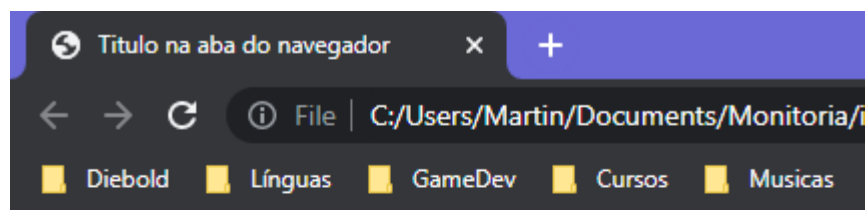
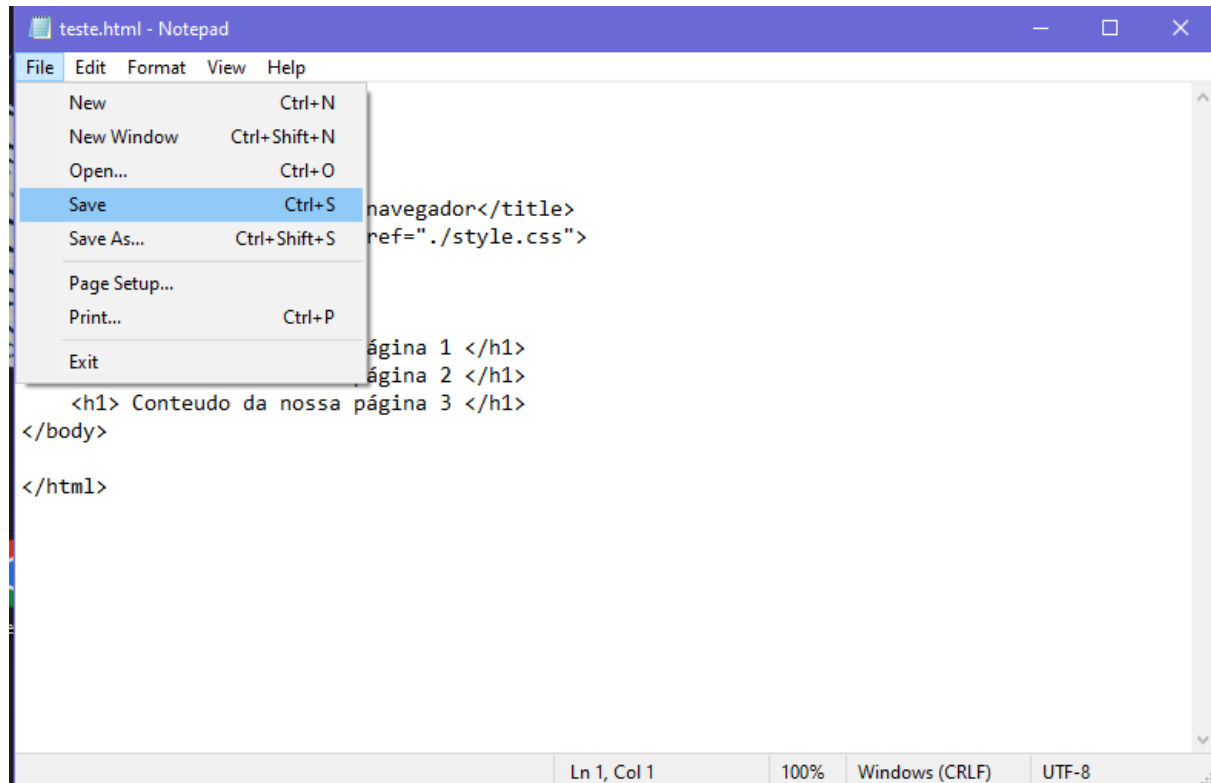
```
<!DOCTYPE html>
<html lang="pt-BR">

<head>
 <title>Titulo na aba do navegador</title>
 <link rel="stylesheet" href="./style.css">
</head>

<body>
 <h1> Conteudo da nossa página 1 </h1>
 <h1> Conteudo da nossa página 2 </h1>
 <h1> Conteudo da nossa página 3 </h1>
</body>

</html>
```

salve o arquivo teste.html e o abra novamente com o navegador.



Podemos ver que o resultado final é o mesmo, porém agora os códigos HTML e CSS estão separados e organizados cada um em seu arquivo.

# TTI107 – Seletor CSS

## 1. Style e CSS

Até agora você estava vendo sobre o elemento **style** da tag, que pode modificar a forma de como o elemento vai ser exibido, e descobriu que pode declará-lo tanto no corpo da tag (inline), no elemento head (internal) e/ou em um arquivo separado **.css** (external).

Apenas uma rápida lembrada, um arquivo **.css** deve ser semelhante a isso:

```
body {}
h1 {
 color: blue;
 background-color: white;
 border-style: solid;
 border-radius: 10px;
 text-align: center;
 font-family:'Courier New', Courier, monospace;
}
p {
 text-align: justify;
 border-left: 10px;
 border-left-style: solid;
 border-left-color: blue;
}
```

*CódigoCSS1.1*

Você já deve ter percebido que o mesmo nome de tag que usamos no arquivo **.html**, também usamos para definir o elemento que queremos mudar a aparência. Então se não gostamos do estilo “padrão do html”, ou se quisermos mudar para deixar nossa página mais bonita, devemos mudar seu estilo (**style**). Logo passamos de algo assim:

## Página super legal

Página muito bonita e estilizada.

*Imagem1.1*

Para algo assim:



# Página super legal

■ Página muito bonita e estilizada.

Imagem1.2

Mas, e se quisermos vários elementos diferentes? Cada título com uma formatação diferente de acordo com o tópico, ou parágrafos diferentes um do outro, para destacar um que seja mais importante que o outro. Enfim, continuar usando os mesmos elementos, porém com estilos diferentes?

## 2. Seletores CSS

No caso dos títulos/tópicos pode-se sugerir mudar o estilo de h2 para ficar parecido com h1 e depois mudar uma coisa ou outra, mas e se eu tiver mais de 6 títulos/tópicos com estilos diferentes? Ou quiser usar um parágrafo com texto grande e outro com texto pequeno?

### 2.1 Seletor CSS por tipo

Como citado antes, já deve ter reparado que usamos o nome da tag no arquivo `.css` para definir o elemento que queremos alterar sua aparência, vamos ver mais afundo alguns aspectos interessantes antes de avançar no assunto.

Já percebeu que mudar o elemento body acaba por afetar a página toda? E os elementos presentes nela?

Faça um teste, usando a tag div (`<div></div>`), dentro do elemento body, crie um elemento parágrafo (`<p></p>`) dentro do elemento div e um parágrafo fora do elemento div, no arquivo `.css`, altere o estilo do elemento div. Deve ter acontecido algo como isso:

Parágrafo fora do elemento div

Parágrafo dentro do elemento div

Imagem2.1.1

```
body {}
div {
 color: blue;
 border-style: solid;
 border-color: red;
}
p {}
```

```
[...]
<body>
 body {}
 div {
 color: blue;
 border-style: solid;
 border-color: red;
 }
 p {
 color: black;
 border-style: solid;
 border-color: black;
 }
}
```

*CódigoCSS2.1.2*

Diagrama de uma página web com uma barra de título e um conteúdo principal. A barra de título contém o texto "Parágrafo fora do elemento div". O conteúdo principal contém o texto "Parágrafo dentro do elemento div".

*Imagem 2.1.2*

## 2.2 Seletor CSS por classe

O tópico anterior abordou um assunto intuitivo: mudar estilo conforme o nome da tag e que uma tag dentro de outra herda seus estilos, como a tag pai (mais externa) e a tag filha (tag interna). Mas apenas isso não nos dá uma liberdade de mudar o estilo como o [css inline](#), para poder fazer uma seleção que não se prenda ao tipo e ainda assim pode ser usada em outros elementos é o atributo [classe](#).

O elemento classe é declarado no arquivo [.css](#) como `‘.nomeClasse’`, e no arquivo [.html](#) é declarado como o elemento [style](#) na tag: `‘<p class="nomeClasse"></p>’`.

```
body {
 .bonito {
 color: blueviolet;
 font-family: 'Courier New', Courier, monospace;
 }
}
```

```
<body>
 <h1 class="bonito">Um título bonito</h1>
 <p class="bonito">Um texto bonito</p>
 <h1>Um título normal</h1>
 <p>Um texto normal</p>
</body>
```

*CódigoCSS2.2.1*

*CódigoHTML2.2.1*

Um título bonito

Um texto bonito

Um título normal

Um texto normal

*Imagem2.2.1*

## 2.3 Seletor CSS por ID

Usar uma classe pode ajudar bastante na hora de definir os estilos e usá-los, mas é possível usar algo ainda mais específico que o tipo de uma tag e sua classe, que é o ID. O ID é declarado no arquivo [.css](#) como `‘#nomeId’`, e no arquivo [.html](#) é declarado no nome da tag como o [style](#): `‘<p id="nomeId"></p>’`.

```
.bonito {
 color: blueviolet;
 font-family: 'Courier New', Courier, monospace;
}
#especial {
 color: aliceblue;
 background-color: darkslategray;
```

```
<body>
 <h1 class="bonito">Um título bonito</h1>
 <p id="especial" class="bonito">Um texto bonito e especial</p>
 <h1>Um título normal</h1>
 <p>Um texto normal</p>
</body>
```

Um título bonito

Um texto bonito e especial

Um título normal

Um texto normal

Imagem2.3.1

Como pode ver no texto bonito e especial, é possível usar classe e id junto, e isso se aplica aos tipos também. Note que a cor de fonte que permanece é branca, declarado no id, porém a família da fonte (font-family) não é a padrão, é a declarada na classe.

Como pode perceber, há um grau de importância entre os seletores, aonde o que for mais específico sobrescreve os outros, e o que não for declarado no mais específico é buscado no menos específico, até chegar no padrão.

### 3 Maneiras de agrupar

Os seletores CSS podem ser agrupados para ter parâmetros mais específicos ou apenas juntar aqueles parâmetros que tem um estilo semelhante.

No caso de juntar seletores por ter um estilo semelhante, pode nos ajudar a economizar tempo e linhas, além de deixar o arquivo **.css** mais organizado.

```
body {}
.bonito, h1 {
 color: blueviolet;
 font-family: 'Courier New', Courier, monospace;
}
#textoEspecial, #tituloEspecial {
 color: aliceblue;
 background-color: darkslategray;
```

Como pode ver é possível agrupar qualquer tipo de seletor, basta separá-los por vírgula.

Também é possível juntar esses seletores para ter estilos mais específicos, pode-se limitar o funcionamento de uma classe/id apenas para certo tipo, ou ainda ter um comportamento polimórfico por parte das classes.

```
.centro#borda {
 border-style: dotted;
 border-color: gold;
 text-align: left;
 color: black;
}
p.centro {
 text-align: center;
 color: red;
}
h1.centro {
 text-align: center;
 color: blue;
}
```

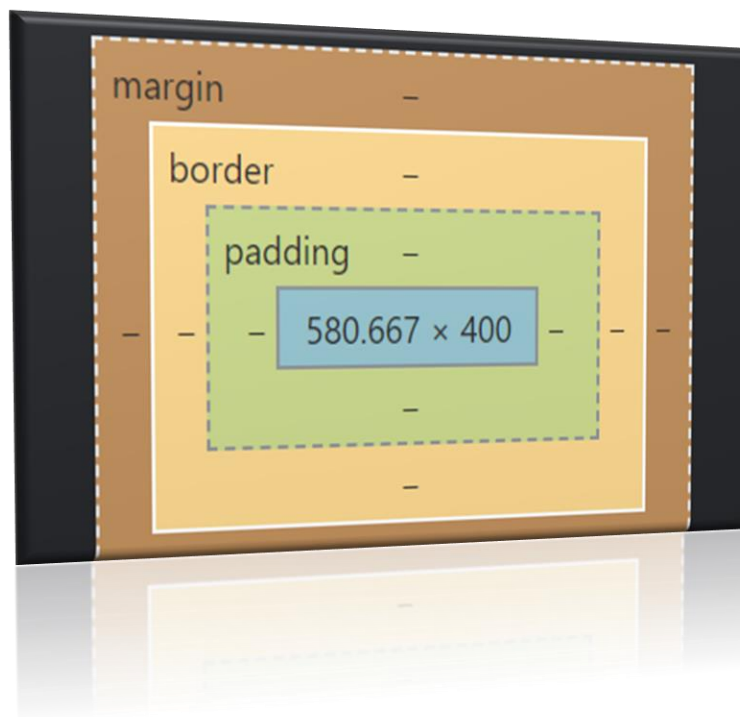
Algo que pode ser feito também que não fica no arquivo **.css** é selecionar mais de uma classe para os elementos no arquivo **.html**, assim pode fazer classes que mudam algo bem específico e juntar elas quando declarar o elemento.

```
<h1 class="centro bonito">Um título bonito</h1>
<p class="centro textoGrande bordaComum" id="perigo"> Cuidado</p>
```

Autor: João Vitor Quirino Sarti  
GitHub:<https://github.com/NULLBYTE-RGH>



## CSS Box model



### O que é:

O modelo Box-Model, é o modelo que rege e orchestra tudo que está em uma página web. Tudo em uma página se comporta como uma caixa (podendo ela ser retangular ou quadrada).

Ele é composto por 4 propriedades principais:

- Content
- Padding
- Border
- Margin

Esse modelo é aplicado desde um título <H1> que possui um significado semântico ao HTML até uma <DIV> genérica.

**(Uma outra maneira de se referir ao modelo box é como Container. Muito utilizado ao se fazer uso do Bootstrap)**

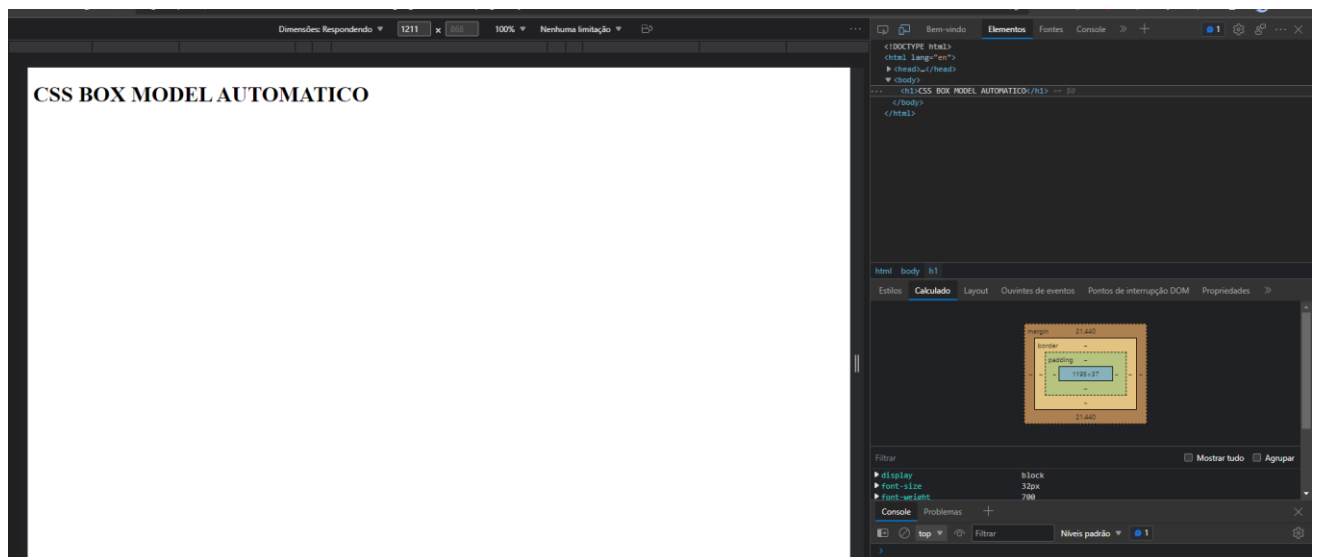
O navegador possui regras CSS pré-carregadas, sendo assim, ao se criar um título por exemplo, e não se colocar as medidas do Box-model, essas regras padrões, serão aplicadas ao conteúdo.

Exemplo:

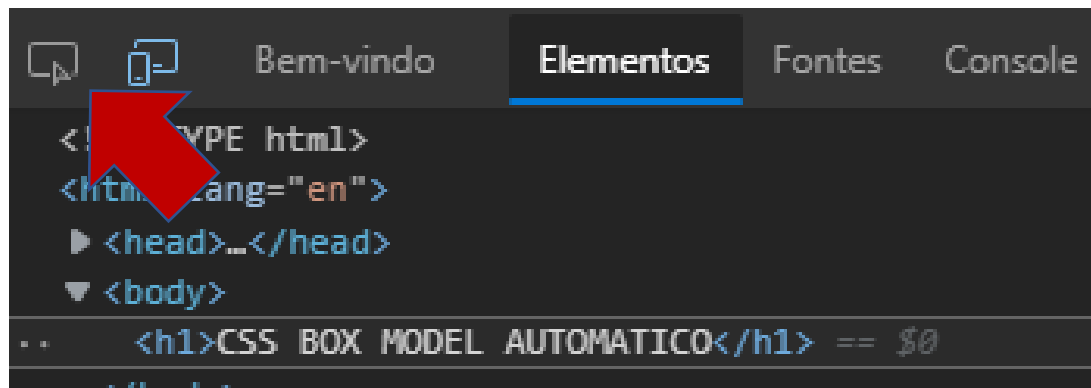
-Criando uma tag H1:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Document</title>
</head>
<body>
 <h1>CSS BOX MODEL AUTOMATICO</h1>
</body>
</html>
```

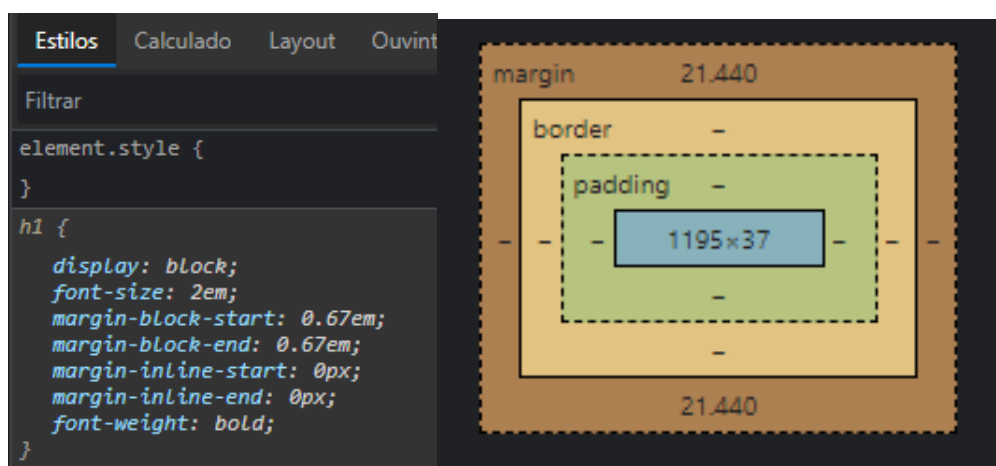
Ao se abrir a página no navegador e executar o inspecionador de elemento com a (tecla F12):



Caso o Box model não apareça, ou caso não seja o box model do elemento que se deseja inspecionar, selecione:



Após isso, basta clicar no texto, no nosso caso, e deverá aparecer as seguintes opções:



Assim, já pode ser ver as regras CSS que o navegador aplicou ao elemento.

Ele nos mostra que o display, tipo de exibição, do elemento, usa as propriedades padrões do tipo bloco. Mas a frente será mostrada que existem variações do tipo bloco. Onde essas variações não deixam de ser um bloco, porém com a capacidade de se ter mais de um elemento na mesma linha por exemplo.



Agora, o que são as outras propriedades? Bom, vamos começar pela Content.

Essa propriedade como o próprio nome diz, é o conteúdo, ou seja, tudo o que definimos dentro da div, podendo ser um texto ou tags filhas.

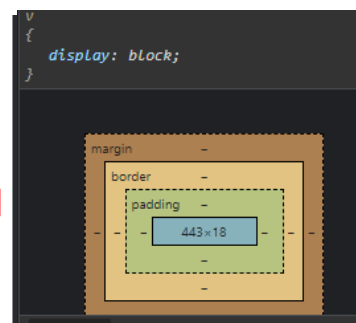
Vamos fazer um exemplo para ficar mais claro:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <style>
 .de{
 color: blue;
 background-color: red;
 }
 </style>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Document</title>
</head>
<body>
 <h1>CSS BOX MODEL AUTOMATICO</h1>
 <div class="de">Eu sou um content!</div>
</body>
</html>
```

Obtemos:

# CSS BOX MODEL AUTOMATICO

Eu sou um content!



Podemos ver que diferentemente do H1, a caixa de uma DIV, não possui margem definida, isso se dá pelo próprio contexto da DIV ser algo genérico e com o intuito de ser altamente manipulada.

Também há de se notar que não se possui propriedades de margem e padding (preenchimento) definidas até o momento.

Será que é por conta disso que a cor do background foi até o final da página?

Bom, se sua resposta foi SIM, você acertou!

O conteúdo ou content é uma região reservada da caixa, ou seja, a prioridade de espaço é sempre dela. A partir dela que se ditam as regras mais externas, como padding e margin. Mais do que isso: o conteúdo pode extrapolar o tamanho que lhe foi definido no content e até mesmo ultrapassar o padding.

Agora, a pergunta é: como definimos o tamanho desse content?

As 2 propriedades são: width (largura/eixo X) e height (altura/eixo y)

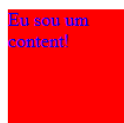
(Essas propriedades se utilizadas com a propriedade padrão box-sizing terá o comportamento de o tamanho definido com o width e height pertencer apenas ao content, mas a frente será apresentada os tipos de box-sizing.)

**Vamos supor que queiramos fazer nosso texto ocupar um espaço de 100px X 100px**

Vamos a um exemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <style>
 .de{
 color: blue;
 background-color: red;
 width: 100px;
 height: 100px;
 }
 </style>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Document</title>
</head>
<body>
 <h1>CSS BOX MODEL AUTOMATICO</h1>
 <div class="de">Eu sou um content!</div>
</body>
</html>
```

## CSS BOX MODEL AUTOMATICO

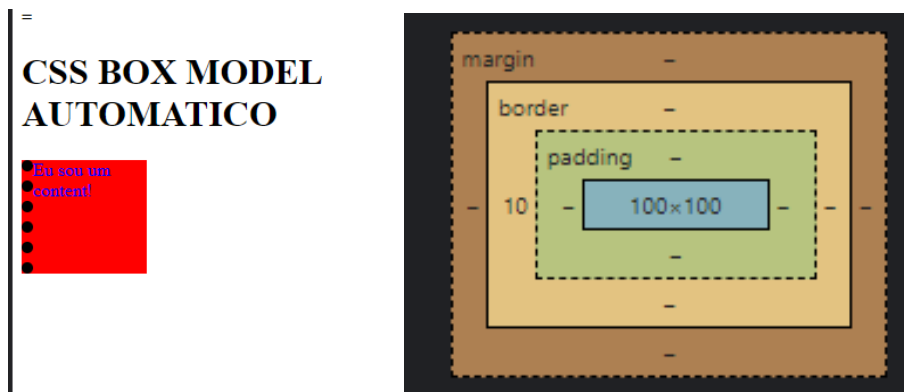


Nesse momento, nosso texto inteiro, junto a cor de fundo ocupam um espaço total de 100x100px, podemos ver que até ocorreu uma quebra de linha automática para o texto se encaixar na largura definida.

**Agora, vamos dizer que queremos colocar uma borda na nossa caixa. Para isso utilizaremos as mesmas propriedades vistas na lista 1:**

```
border-left: 10px;
border-left-style: dotted;
border-color: black;
```

obtemos:



Isso nos mostra que agora temos mais uma propriedade ativada na nossa box model, e como só adicionamos a borda no lado esquerdo da nossa Box, os 10px só se aplicam a esse pedaço. Porém, como podemos ver, os círculos estão ocupando um espaço grande e estão muito perto do nosso conteúdo.

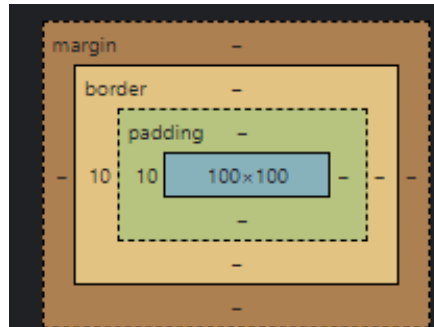
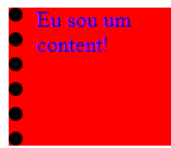
**Agora entra a propriedade Padding (preenchimento), ela vai ser a camada entre a margem e o conteúdo.**

Vamos adicionar a seguinte propriedade a nossa tag style:

```
padding-left: 10px;
```

obtemos:

# CSS BOX MODEL AUTOMATICO



*(Todas as propriedades mostradas até agora, podem ser aplicadas tanto individualmente como foi feito: `padding-left`, `padding-top` etc. Como também representando todos os lados, utilizando simplesmente `padding` por exemplo)*

Agora vamos ativar a última propriedade principal do Box-Model, a margin:

A margem é utilizada para distanciar uma caixa da outra, sendo elas no modelo Block ou Inline Block.

Mas se ainda não definimos nada para se distanciar da outra caixa <H1> por qual razão ele tem uma separação?

Bom essa distância se dá pois como vimos, a tag H1 possui suas propriedades já definidas, e uma delas é a margin. A margem é a distância de uma box a outras caixas, ou até mesmo as bordas da janela do navegador, sendo assim, a margem não só influencia a caixa em que ele foi definida, como todas as caixas próximas a ela.

Para definirmos uma margem, é muito simples; a propriedade é: margin

Vamos adicionar essa propriedade a nossa caixa, e para deixar bem evidente vamos utilizar uma distância de 100px

*(Dica: ao se utilizar Pixels como medida seu site não fica responsivo, pois é uma unidade engessada, fixa, para o tornar responsivo utilize por exemplo VW, outras medidas podem ser consultadas nesse site : [Guia de Unidades no CSS | Alura](#))*

```
<style>
.de{
 color: blue;
 background-color: red;
 width: 100px;
 height: 100px;
 border-left: 10px;
 border-left-style: dotted;
 border-color: black;
 padding-left: 10px;
 margin-left: 100px;
}
</style>
```

Resultado:



Muito bem, até agora tudo certo?

Entao.... existe uma outra propriedade da Box-Model, chamada box-sizing. Nós estávamos a utilizando até agora, só que não estávamos definindo-a explicitamente, ou seja, estávamos utilizando o seu valor Default.

Mas no final das contas, o que essa propriedade faz?

A verdade é que ela vai mudar a forma como os tamanhos são calculados.

Lembra que definimos a largura e a altura como 100px X 100px e foi dito que ela pertencia ao tamanho do content? Bom, isso só é verdade se utilizado (padrão/Default):

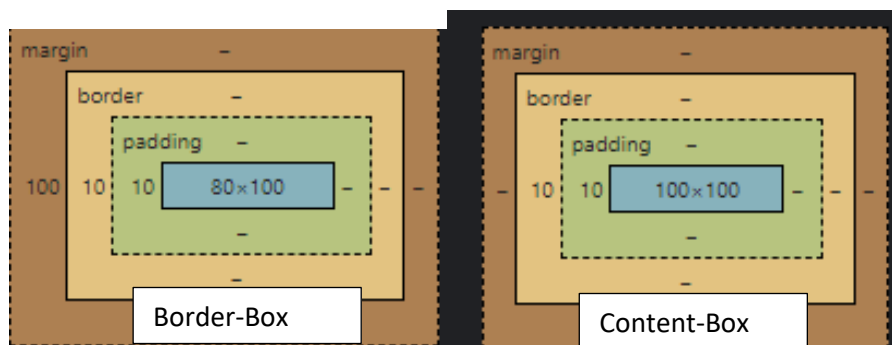
```
box-sizing: content-box;
```

Agora, se utilizarmos a propriedade como:

```
box-sizing: border-box;
```

O tamanho definido no width e height pertencem a caixa toda (content, padding e border).

A partir do momento que for definido como border-box, todas as medidas que passamos serão recalculadas para fazer toda a caixa se encaixar nos novos parâmetros:



Com o Border-Box podemos verificar o tamanho da nossa nova caixa, fazendo a simples soma de tudo no eixo X e tudo no Y:

Em X: Content + padding + border = 80 + 10 + 10 = 100px

Em Y: content + padding + border = 100 + 0 + 0 = 100px

Essa propriedade vai ser útil quando se quer ter certeza de que seu elemento da tag, não vá ocupar mais espaço do que o dimensionado.

### Extra: (OVERFLOW)

Ao se definir um conteúdo grande, ele pode ultrapassar o tamanho da caixa, causando algo chamado overflow:



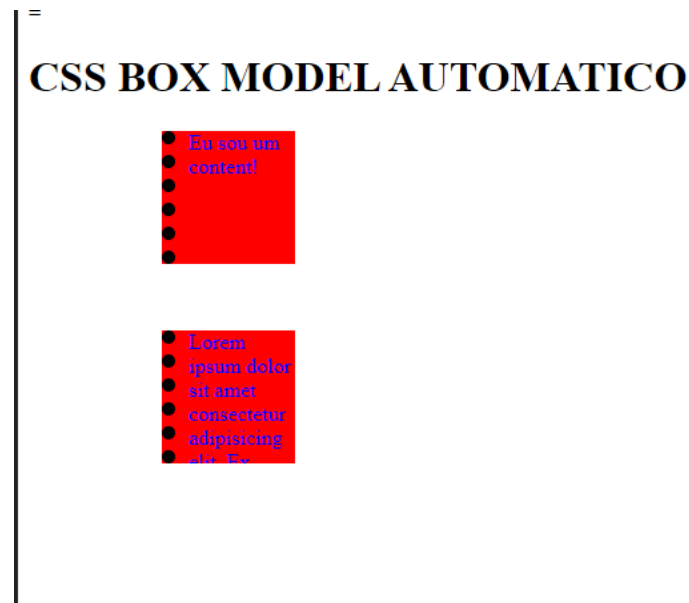
Isso é algo indesejável, e como pode ser resolvido? Bom mais para frente vocês verão outras propriedades CSS que vão lidar muito bem com elementos variáveis e darão a responsividade necessária.

Entao não tem nada que se possa fazer com o Box-Model e display: block? E a resposta é, SIM!

Uma delas é utilizar a propriedade Overflow:

```
OVERFLOW: hidden;
```

Ela simplesmente faz uma corte no conteúdo excedente:



Más pode ser que você não queira que o conteúdo simplesmente sumo e fique dessa forma estranha, como se faltasse uma parte, bom, para isso você pode utilizar a propriedade:

```
OVERFLOW: scroll;
```

Como o próprio nome sugere, você agora tem a opção de usar o scroll do mouse para passar o conteúdo dentro da sua div!

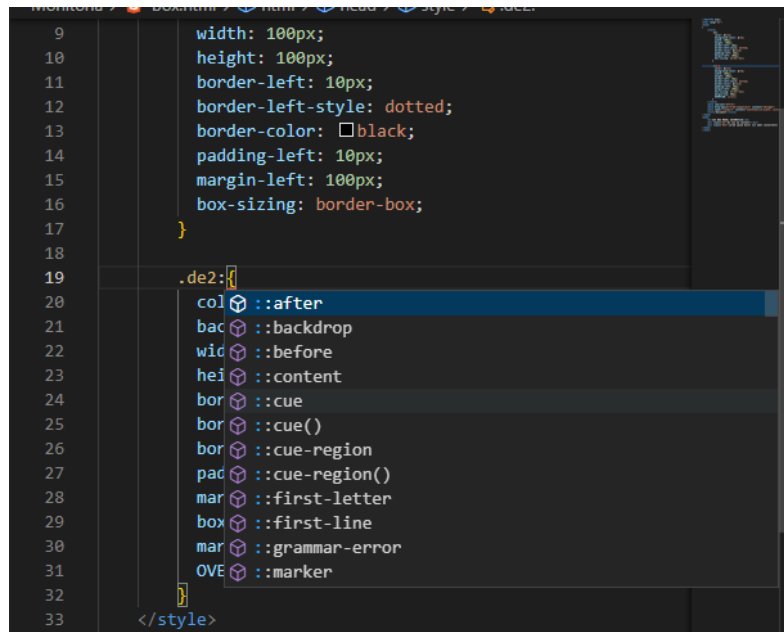
Vou me estender um pouco agora, pois pode ser que nem seja mostrado a vocês isso, mas acho que seja pertinente mostrar.

As propriedades CSS podem ser ativadas por condições, isso sem uso de java-script.

Como?

Dessa forma:

Ao se digitar (:) após uma classe ou ID, o visual code irá sugerir várias propriedades que podem ser utilizadas:



Cada uma dessas propriedades indica que essa classe, no caso de2, será ativada quando um desses eventos ocorrer.

Vou exemplificar com uma propriedade chamada (hover), ele é um evento que indica que o mouse passou em cima da DIV que tem esse evento associado.

E qual a utilidade disso? Responsividade e animação!

Exemplo prático, tomando o Moodle como exemplo; quando o mouse é passado em cima das matérias ele não tem uma animação de aumentar de tamanho da matéria e fazer um sombreado aparecendo ao fundo para dar uma sensação de que ele se descolou da tela? Então, ele utiliza exatamente essa propriedade CSS, só que muito provavelmente faz uso também da biblioteca bootstrap, com uma classe chamada CARD

Falei....falei e agora vou exemplificar:

```
.de2:hover{
 color: blue;
 background-color: red;
 width: 200px;
 height: 200px;
 border-left: 10px;
 border-left-style: dotted;
 border-color: black;
 padding-left: 10px;
 margin-left: 100px;
 box-sizing: border-box;
 margin-top: 50px;
 OVERFLOW: normal;
```



```

}
.de2{
 color: blue;
 background-color: red;
 width: 100px;
 height: 100px;
 border-left: 10px;
 border-left-style: dotted;
 border-color: black;
 padding-left: 10px;
 margin-left: 100px;
 box-sizing: border-box;
 margin-top: 50px;
 OVERFLOW: hidden
}

```

Ao usar propriedades na classe, se deve duplicar a classe, assim, por padrão a classe somente com `.de2{}` vai ser executada, agora quando o mouse passar pela DIV que faz uso da classe `.de2`, as propriedades do `.de2:hover {}` é que vão ser utilizadas.

Resultado:

=

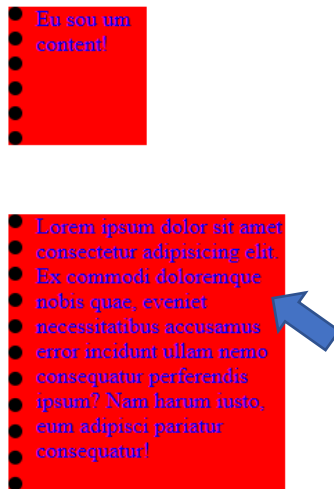
## CSS BOX MODEL AUTOMATICO

• Eu sou um  
• content!

• Lorem  
• ipsum dolor  
• sit amet  
• consectetur  
• adipiscing  
• elit. Et

*Figura 1 DIV sem interação do mouse*

## CSS BOX MODEL AUTOMATICO



*Figura 2 DIV com interação do mouse*

Sites recomendados:

[Bootstrap em Português · O mais popular framework front-end responsivo e focado para dispositivos móveis do mundo. \(getbootstrap.com.br\)](https://getbootstrap.com.br/)

[Guia de Unidades no CSS | Alura](#)

[W3Schools CSS overflow-wrap demonstration](#)

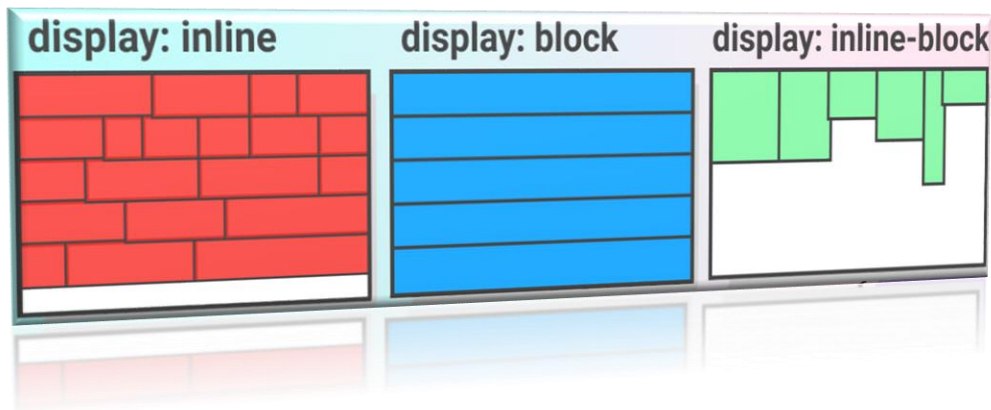
[CSS :hover Selector \(w3schools.com\)](https://www.w3schools.com/css/css_hover_selector.asp)

Autor: João vitor Quirino Sarti

GitHub: [NULLBYTE-RGH \(github.com\)](https://github.com/NULLBYTE-RGH)



## CSS (Display): Block, Inline e Inline-Block



### O que é:

A propriedade display nada mais é o que a forma como um determinado elemento irá se comportar na árvore DOM, nesse caso, o comportamento é como o elemento irá ser renderizado e se comportará em relação aos elementos ao seu redor.

Existem vários tipos de Displays, uns mais flexíveis e outros menos, nesse momento serão abordados 3 tipos:

- Block
- Inline
- Inline-Block

### Display: Block

O display Block é o mesmo que foi utilizado até o momento, que é o padrão para a maioria das TAGs HTML (se utilizado o style sheet padrão do navegador) ele funciona com as propriedades mencionadas na lista passada (Box-Model), ou seja, content, padding, margin e border.

Esse tipo de display tem a característica de exibir apenas 1 conteúdo por “Linha”, independente de outro conteúdo caber ou não ao lado.

```
display: block;
```

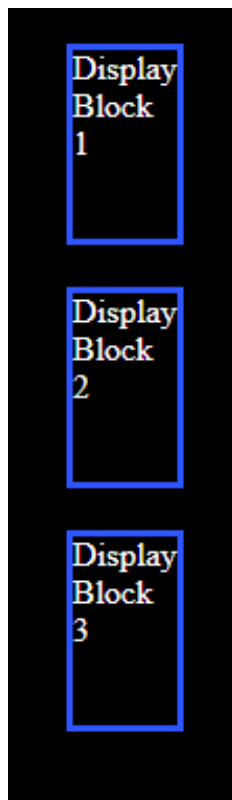
Exemplo:

-Criando Divs com display Block:

```
<!DOCTYPE html>
<html lang="en">
 <head>
 <meta charset="UTF-8" />
 <meta http-equiv="X-UA-Compatible" content="IE=edge" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0"
 />
 <title>Displays</title>
</head>
<body style="background-color:black; color:white">
 <div
 style="
 display: block;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: rgb(47, 85, 255);
 "
 >
 Display Block 1
 </div>
 <div
 style="
 display: block;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: rgb(47, 85, 255);
 "
 >
 Display Block 2
 </div>
 <div
 style="
 display: block;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: rgb(47, 85, 255);
 "
 >
 Display Block 3
```

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

O resultado esperado é:



Esse resultado não é nada mais do que você já sabia e estava aplicando até agora. Cada Box-Model tem seu espaço reservado na linha, independente da largura da caixa. Agora e se você quiser aplicar várias TAGs na mesma linha? Bom, para isso existe o tipo de display (**Inline**).

## **Display:Inline**

Esse display é utilizado quando se quer mostrar várias TAGs em sequência, o que é muito útil quando se quer fazer um menu horizontal por exemplo, porém o display **Inline** apenas de ser um Box-Model, as propriedades altura e largura, não tem efeito sobre o conteúdo dessas caixas, isso porque o intuito é realmente se comportar como se fosse tudo um “texto” contínuo.

```
display: inline;
```

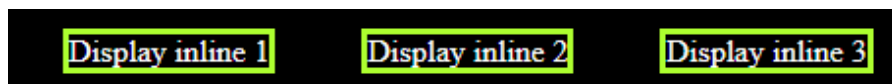
```
width: 50px; → Não tem efeito
```

```
height: 90px; → Não tem efeito
```

Exemplo:

```
<div
 style="
 display: inline;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: greenyellow;
 "
>
 Display inline 1
</div>
<div
 style="
 display: inline;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: greenyellow;
 "
>
 Display inline 2
</div>
<div
 style="
 display: inline;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: greenyellow;
 "
>
 Display inline 3
</div>
```

A saída é:



Agora temos diferentes blocos alinhados atuando como um único bloco!

## Display:Inline-Block

O display Inline-Block é similar ao Inline, porém sua diferença fica no quesito de os blocos no Display Inline não atuarem como fazendo parte de um único Box model, e sim como cada bloco tendo seu próprio Box-Model. Em suma, esse display é como se os blocos que antes ficavam cada um em uma linha, agora podem ocupar o mesmo espaço em uma linha.

E pelo fato de cada bloco ser individualizado todas as propriedades do display Block estão valendo, ou seja, largura e altura agora podem ter efeito sobre os elementos.

```
display: inline-block;
```

Exemplo:

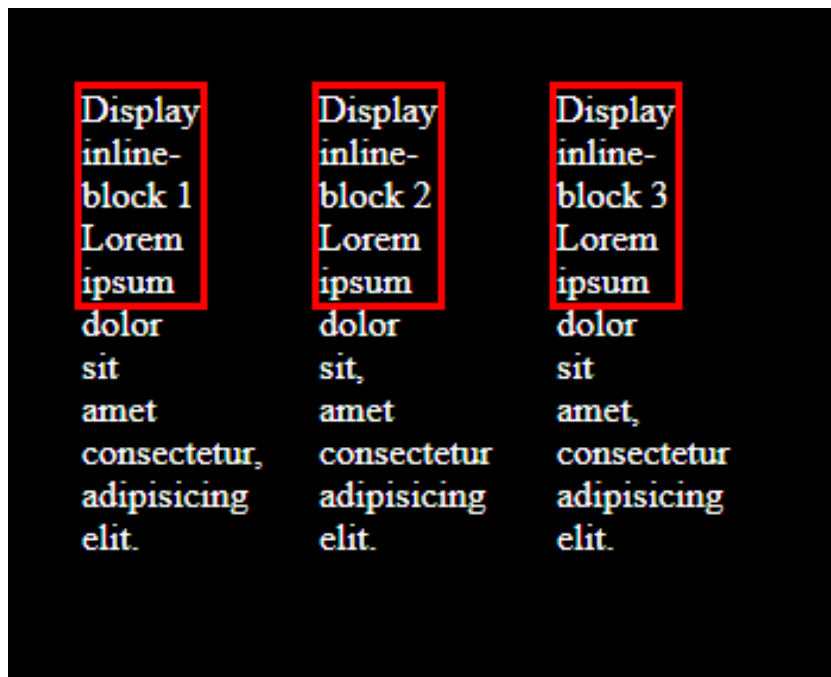
```
<div
 style="
 display: inline-block;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: red;
 "
>
 Display inline-block 1 Lorem ipsum dolor sit amet consectetur,
adipisicing
 elit.
</div>
<div
 style="
 display: inline-block;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: red;
 "
>
 Display inline-block 2 Lorem ipsum dolor sit, amet consectetur
adipisicing
 elit.
</div>
<div
 style="
 display: inline-block;
 border: 3px solid;
 width: 50px;
 height: 90px;
```

```

 margin: 20px;
 border-color: red;
 "
 >
 Display inline-block 3 Lorem ipsum dolor sit amet, consectetur
adipisicing
 elit.
 </div>

```

Resultado:



Vemos que agora nossas caixas estão alinhadas e até mesmo o sentido do overflow do texto, já que limitamos a largura do Box-Model e sua altura. Esse vazamento pode ser resolvido pelo método Overflow, abordado na lista anterior ([Box\\_Model](#)).

Juntando todo o código temos um exemplo de fácil visualização:

```

<!DOCTYPE html>
<html lang="en">
 <head>
 <meta charset="UTF-8" />
 <meta http-equiv="X-UA-Compatible" content="IE=edge" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0"
 />
 <title>Displays</title>
</head>
<body style="background-color: black; color: white">
 <p style="display: flex; border: 4px dotted; justify-content:
center">

```



```

 Display Block
 </p>
 <div
 style="
 display: block;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: rgb(47, 85, 255);
 "
 >
 Display Block 1
 </div>
 <div
 style="
 display: block;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: rgb(47, 85, 255);
 "
 >
 Display Block 2
 </div>
 <div
 style="
 display: block;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: rgb(47, 85, 255);
 "
 >
 Display Block 3
 </div>

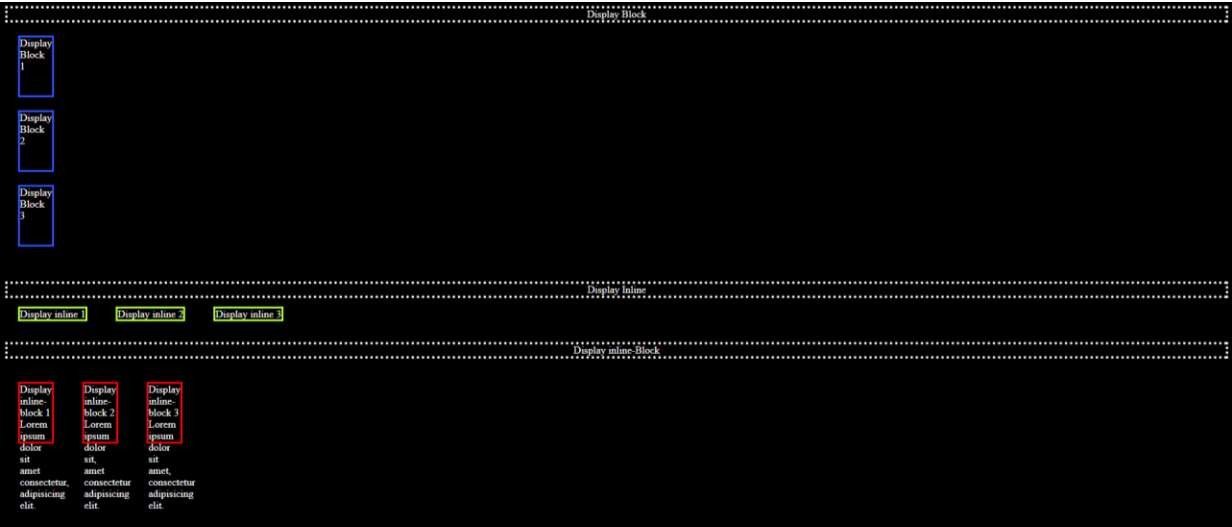
 <p style="display: flex; border: 4px dotted; justify-content:
center">
 Display Inline
 </p>
 <div
 style="
 display: inline;
 border: 3px solid;
 width: 50px;
 height: 90px;

```

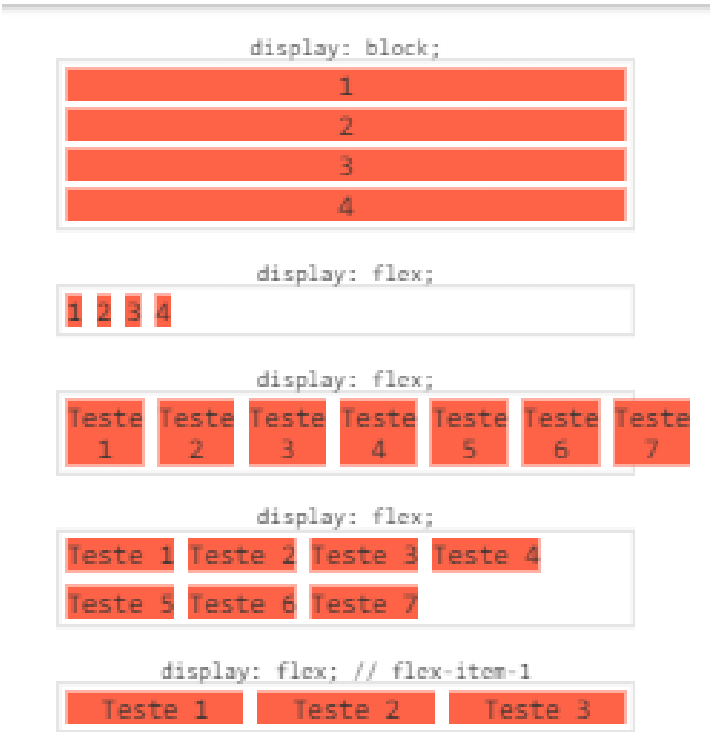


```
</div>
<div
 style="
 display: inline-block;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: red;
 "
>
 Display inline-block 2 Lorem ipsum dolor sit, amet consectetur
adipisicing
 elit.
</div>
<div
 style="
 display: inline-block;
 border: 3px solid;
 width: 50px;
 height: 90px;
 margin: 20px;
 border-color: red;
 "
>
 Display inline-block 3 Lorem ipsum dolor sit amet, consectetur
adipisicing
 elit.
</div>
</body>
</html>
```

Saída:



Nos títulos foi utilizado o Display do Tipo Flex, que não será abordado nesse momento, mas caso queira saber mais: [Guia completo de Flexbox - CSS - display: flex; \(origamid.com\)](#)



Sites recomendados:

[CSS display property \(w3schools.com\)](#)

[CSS Height, Width and Max-width \(w3schools.com\)](#)

# CSS: display (flex)

Autor: Martin Röpke

## Código para testar propriedades:

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

<head>
 <style>
 .flex-container {
 display: flex;
 }

 div {
 outline: 3px solid red;
 width: fit-content;
 height: fit-content;
 margin: 10px;
 padding: 10px;
 }
 </style>
</head>

<body>
 <div class="flex-container">
 <div>1</div>
 <div>2</div>
 <div>3</div>
 </div>
</body>

</html>
```

## Para que serve o display flex?

A ferramenta display flex serve para tornar mais simples e funcional as tarefas de criação de leiaute.

Ele ajuda em funcionalidades como alinhar e distribuir espaços entre itens em um container, mesmo quando as dimensões destes itens são desconhecidas.



## Como funciona?

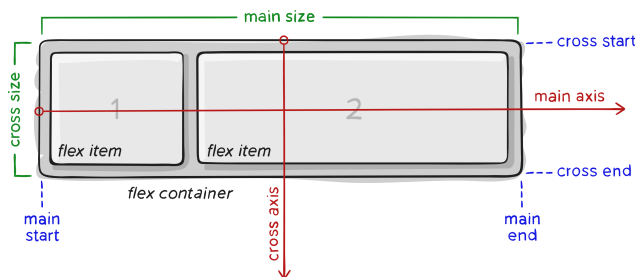
O display flex não é apenas uma única propriedade aplicada pelo CSS, mas sim um conjunto de propriedades CSS, sendo algumas aplicadas em elementos filhos de um elemento container e outras no próprio elemento pai container.

Você define um elemento pai container como flex com a propriedade **display** e o valor **flex**

```
.flex-container {
 display: flex;
}
```

```
<div class="flex-container">
 <div>1</div>
 <div>2</div>
 <div>3</div>
</div>
```

O display flex é baseado em dois **eixos principais**, o **main axis** e o **cross axis**. Segue uma imagem de exemplo:



Os itens dentro do container serão dispostos no layout seguindo ou o **main axis** ou o **cross axis**, tudo depende de qual eixo foi escolhido como eixo principal.

## Algumas propriedades para elementos pai container

### Flex-direction

A propriedade flex-direction serve para definir qual eixo é usado como eixo principal, assim alterando a forma como os itens são dispostos.



Existem quatro valores possíveis para essa propriedade. **Row, row-reverse, column e column-reverse**.

As versões **reverse** de cada valor da propriedade apenas alteram se os itens são dispostos da esquerda para direita ou da direita para esquerda no eixo escolhido.

```
.flex-container {
 display: flex;
 flex-direction: row-reverse;
}
```

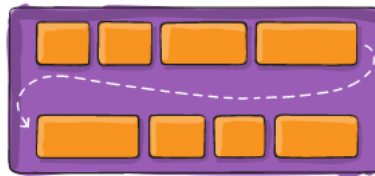


```
.flex-container {
 display: flex;
 flex-direction: column;
}
```



## Flex-wrap

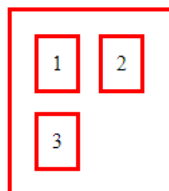
Por padrão, os *flex items* vão todos tentar se encaixar em uma só linha. Com esta propriedade você pode modificar esse comportamento e permitir que os itens quebrem para uma linha seguinte conforme for necessário.



Os valores possíveis para essa propriedade são **nowrap** (padrão), **wrap**, **wrap-reverse**.

```
.flex-container {
 flex-wrap: wrap;
}
```

```
<div class="flex-container" style="width: 100px;">
 <div>1</div>
 <div>2</div>
 <div>3</div>
</div>
```



## Justify-content

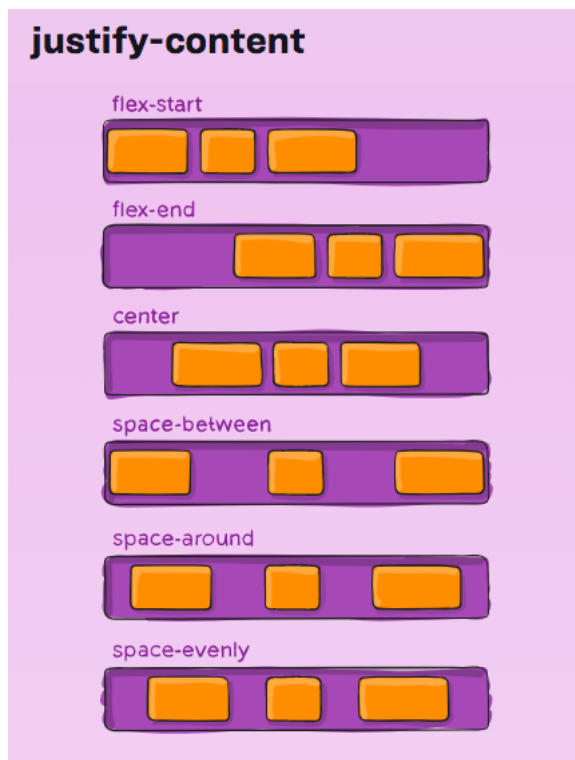
A propriedade **Justify-content** serve para alterar a forma como os itens se alinham em relação em eixo em que eles estão dispostos.

```
.flex-container {
 display: flex;
 justify-content: flex-start;
}
```

```
<div class="flex-container" style="width: 350px;">
 <div>1</div>
 <div>2</div>
 <div>3</div>
</div>
```



Segue abaixo os valores possíveis para essa propriedade:



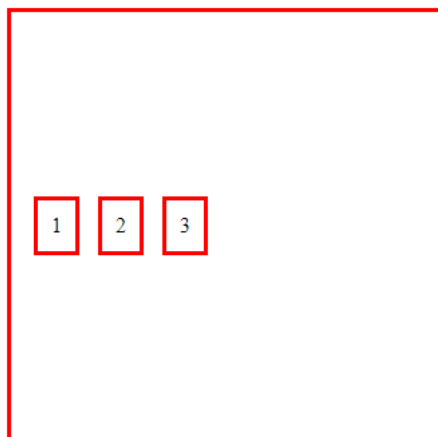
## align-items

A propriedade **align-items** serve para alterar a forma como os itens se alinham em relação ao eixo perpendicular ao eixo principal no qual eles estão dispostos.

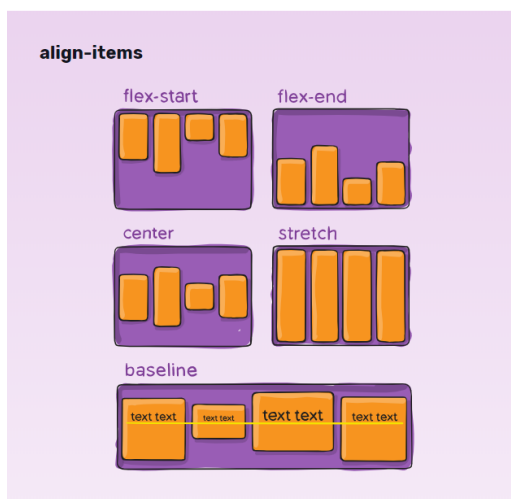
```
.flex-container {
 display: flex;
 align-items: center;
}
```

```
<div class="flex-container" style="width: 300px; height: 300px;">
 <div>1</div>
 <div>2</div>
 <div>3</div>
</div>
```





Segue abaixo os valores possíveis para essa propriedade:



## Propriedades para elementos filhos

É possível aplicar propriedades CSS específicas para elementos filhos de um elemento pai container flex e assim mudar a forma como eles são exibidos individualmente ao longo do eixo.

```
<div class="flex-container">
 <div class="flex-item1">1</div>
 <div class="flex-item2">2</div>
 <div class="flex-item3">3</div>
</div>
```

### Order

Determina a ordem em que os elementos aparecerão.

```
.flex-item1 {
 order: 2;
}

.flex-item2 {
 order: 3;
}

.flex-item3 {
 order: 1;
}
```



## Referências

Se você quiser se aprofundar mais nas propriedades apresentadas e aprender algumas novas propriedades você pode acessar esses sites:

[https://www.alura.com.br/artigos/css-guia-do-flexbox?](https://www.alura.com.br/artigos/css-guia-do-flexbox?gclid=Cj0KCQjw4omaBhDqARIsADXULuXLXYmgnYgkuAQISw13hioZNR14IzD3Q0t6awwgp_6YFWmYSxn42t8aAvTvEALw_wcq)

[gclid=Cj0KCQjw4omaBhDqARIsADXULuXLXYmgnYgkuAQISw13hioZNR14IzD3Q0t6awwgp\\_6YFWmYSxn42t8aAvTvEALw\\_wcq](https://www.alura.com.br/artigos/css-guia-do-flexbox?gclid=Cj0KCQjw4omaBhDqARIsADXULuXLXYmgnYgkuAQISw13hioZNR14IzD3Q0t6awwgp_6YFWmYSxn42t8aAvTvEALw_wcq)

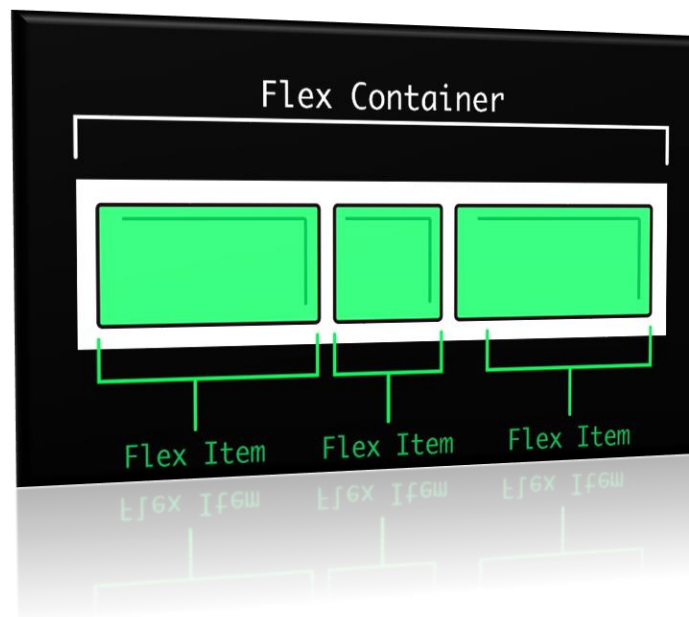
<https://origamid.com/projetos/flexbox-guia-completo/>

Autor: João vitor Quirino Sarti

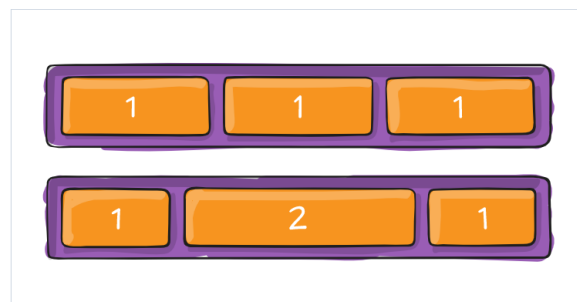
GitHub: [NULLBYTE-RGH \(github.com\)](https://github.com/NULLBYTE-RGH)



## CSS (Display): Flex



### Flex-Grow:



A propriedade Grow, faz com que em uma determinada largura de container, determinada por exemplo por width, um dos blocos (Numerados na imagem de 1 a 3) tenham um crescimento maior em relação aos outros. Por padrão a propriedade tem valor 0:

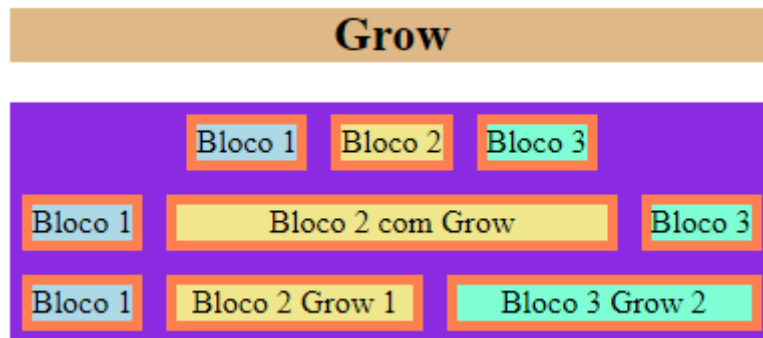
```
flex-grow: 0;
```

Isso significa que todos os blocos estão em igualdade para dividir o espaço do container, porém, ao definir um valor de crescimento, o bloco que receber o valor acima de 0 terá prioridade em relação aos outros.

Se apenas um bloco tiver a prioridade de crescimento Flex-Grow, o valor Máximo que se pode definir é 1, significando que ele ocupará o espaço não ocupado pelos outros blocos. Já se tiver mais de um bloco utilizando a propriedade, aí sim pode-se utilizar valores acima de 1, fazendo

com que o bloco com maior valor tenha maior prioridade e consequentemente ocupe um espaço maior no container flex.

Por exemplo:



- 1º Linha tem-se grow padrão em todos, ou seja 0.
- 2º Linha, apenas o bloco 2 tem grow, 1.
- 3º Linha bloco 2 com Grow 1 e bloco 3 com grow 3

Sendo assim, nota-se claramente o efeito do grow, em blocos concorrentes por espaço.

Código para obter essa saída:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-
scale=1.0">
 <title>Flex</title>
</head>

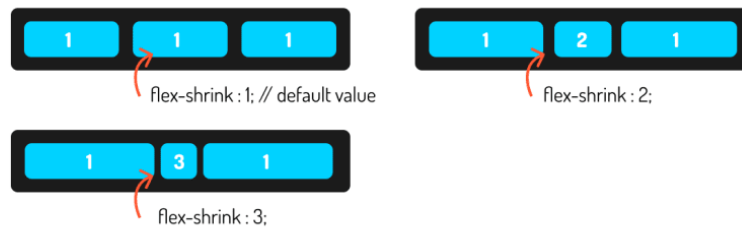
<style>
 .container{
 display: flex;
 background-color:blueviolet;
 width:20%;
 justify-content: center;
 }

 .Padrao{
 border: 5px solid coral;
 margin: 6px;
 text-align: center;
 }

 .Titulo{
 display:flex;
 background-color:burlywood;
 justify-content: center;
```



## Flex-shrink:



O Shrink funciona de forma com que o valor padrão é 1:

```
flex-shrink : 1;
```

Isso significando que, todos os blocos tendem a se encolher a fim de permitir com que o tamanho do container seja respeitado e todos os blocos ocupem um tamanho igual dentro do container:

- Blocos de tamanho Iguais
- Respeitar o tamanho do container

Ao se mudar esse valor para 0

```
flex-shrink : 0;
```

O bloco que estiver com essa propriedade vai utilizar apenas a largura definida:

```
width:600px;
```

E não ligara para os blocos ao seu arredor.

Caso o valor do Shrink seja maior do que 1, cada vez mais o bloco com essa propriedade ira encolher para que outros blocos maiores possam ocupar mais espaço, na tentativa de atingir o espaço definido na largura original, WIDTH.



Nesse caso os blocos se estenderam tanto que até passaram o tamanho do container, isso seria resolvido com o Flex-Wrap adicionado ao container.

Código:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-
scale=1.0">
 <title>Flex</title>
</head>

<style>
 .container{
 display: flex;
 background-color:blueviolet;
 width:600px;
 justify-content: center;
 margin-left:500px;
 }

 .Padrao{
 border: 5px solid coral;
 margin: 6px;
 text-align: center;
 width: 400px;
 flex-shrink: 1;
 }

 .Titulo{
 display:flex;
 background-color:burlywood;
 justify-content: center;
 width:600px;
 margin-left:500px;
 }

 .Shrink{
 flex-shrink : 0;
 }

</style>
<body>

 <h2 class="Titulo">Shrink</h2>

 <div class="container">
```

```

 <div style = "background-color:lightblue" class="Padrao">Bloco
1-->400 px</div>
 <div style = "background-color:khaki;" class="Padrao">Bloco 2-
->400 px</div>
 <div style = "background-color:aquamarine;"
class="Padrao">Bloco 3-->400 px</div>
 </div>

 <div class="container">
 <div style = "background-color:lightblue" class="Padrao">Bloco
1-->400 px Shrink 1</div>
 <div style = "background-color:khaki;" class="Padrao
Shrink">Bloco 2-->400 px Shrink 0</div>
 <div style = "background-color:aquamarine;"
class="Padrao">Bloco 3-->400 px Shrink 1</div>
 </div>

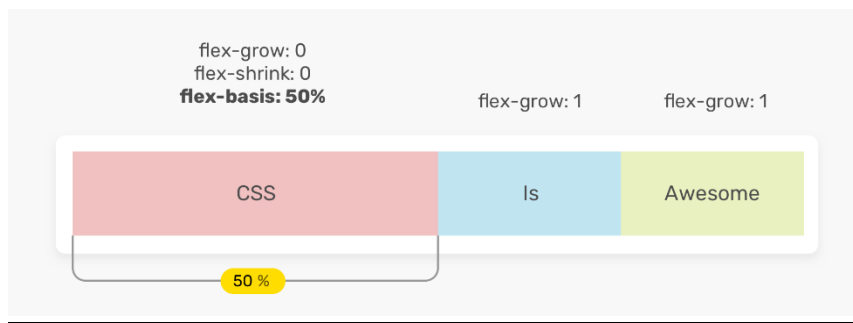
 <div class="container">
 <div style = "background-color:lightblue" class="Padrao
Shrink">Bloco 1-->400 px Shrink 0</div>
 <div style = "background-color:khaki;" class="Padrao">Bloco 2-
->400 px Shrink 1</div>
 <div style = "background-color:aquamarine;" class="Padrao
Shrink">Bloco 3-->400 px Shrink 0</div>
 </div>

</body>
</html>

```



## Flex-basis:



O Basis tem como intuito de permitir ou não o uso das propriedades do bloco definidas como:

```
width:600px;
margin-left:500px;
padding:30px;
```

Essas propriedades fazem com que o uso do display Flex, não cumpra seu papel caso os blocos cresçam e precisem ser redimensionados, para isso a propriedade:

```
flex-basis: auto;
```

Permaneces por padrão em automático. Isso significa que ela tenta usar as propriedades do BoxModel, porém caso precise redimensionar ela assim o fará.

Agora, caso deseje que o bloco tenha uma largura inicial diferente da salva no WIDTH, o comando a ser usado é:

```
flex-basis: 600px;
```

Sendo assim a largura agora passa a ser 600px, e isso faz com que todas as outras caixas que têm o Basis em AUTO, respeitem essa medida da nova caixa.



Codigo:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-
scale=1.0">
 <title>Flex</title>
</head>

<style>
 .container{
 display: flex;
 background-color:blueviolet;
 width:600px;
 justify-content: center;
 margin-left:500px;
 }

 .Padrao{
 border: 5px solid coral;
 margin: 6px;
 text-align: center;
 width: 100px;
 flex-shrink: 1;
 }

 .Titulo{
 display:flex;
 background-color:burlywood;
 justify-content: center;
 width:600px;
 margin-left:500px;
 }

 .Basis{
 flex-basis : 600px;
 }

 .BasisPadrao{
 flex-basis: auto;
 }

 .Shrink{
 flex-shrink: 0;
 }

</style>
```

```
<body>

 <h2 class="Titulo">Basis</h2>

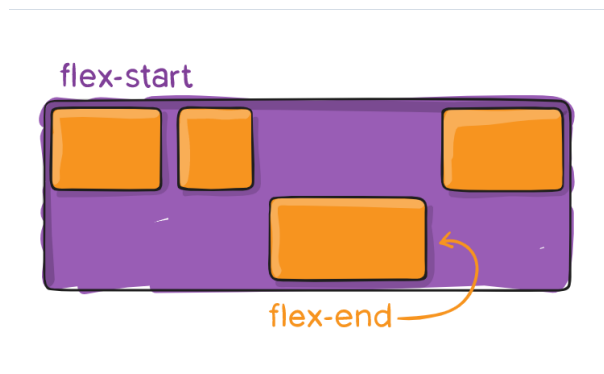
 <div class="container">
 <div style = "background-color:lightblue" class="Padrao
BasisPadrao">Bloco 1-->100 px</div>
 <div style = "background-color:khaki;" class="Padrao
BasisPadrao">Bloco 2-->100 px</div>
 <div style = "background-color:aquamarine;" class="Padrao
BasisPadrao">Bloco 3-->100 px</div>
 </div>

 <div class="container">
 <div style = "background-color:lightblue" class="Padrao
BasisPadrao">Bloco 1-->100 px</div>
 <div style = "background-color:khaki;" class="Padrao Basis">Bloco
2-->100 px</div>
 <div style = "background-color:aquamarine;" class="Padrao
BasisPadrao">Bloco 3-->100 px</div>
 </div>

 <div class="container">
 <div style = "background-color:lightblue" class="Padrao
Basis">Bloco 1-->100 px</div>
 <div style = "background-color:khaki;" class="Padrao
BasisPadrao">Bloco 2-->100 px</div>
 <div style = "background-color:aquamarine;" class="Padrao
Shrink">Bloco 3-->100 px</div>
 </div>

</body>
</html>
```

## Align-self:



A propriedade Align-Self:

```
Align-Self: stretch;
```

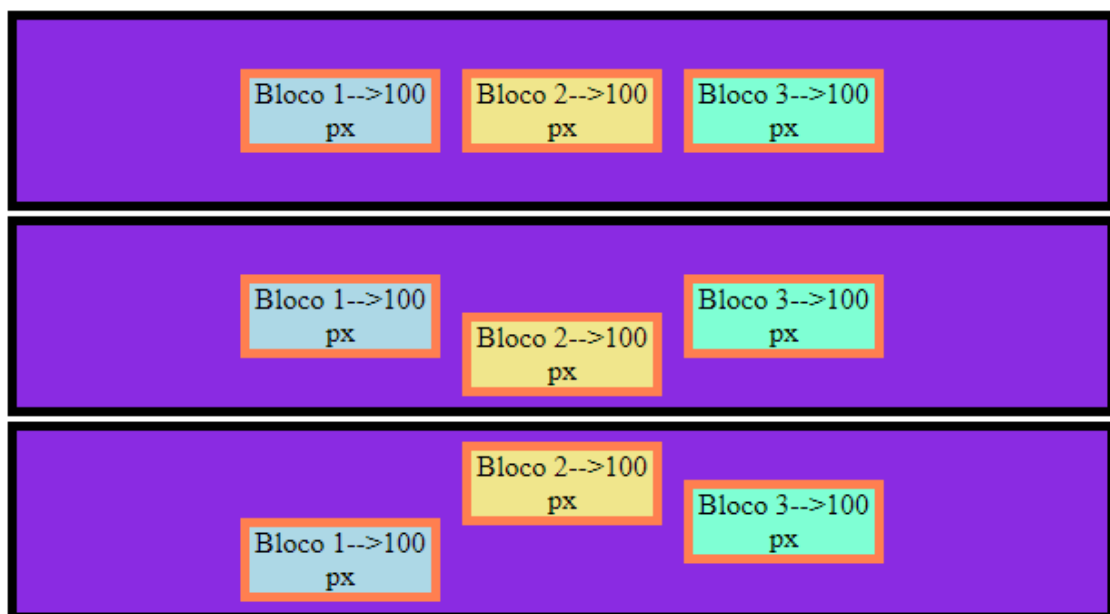
Tem como função aplicar propriedades como:

```
align-items: stretch | flex-start | flex-end | center | baseline;
```

```
align-content: flex-start | flex-end | center | space-between | space-around | stretch;
```

Só que ao invés de aplicar ao contêiner, o que acarreta que todos os blocos vão se comportar da mesma maneira, ele aplica as mesmas propriedades a blocos individuais.

### Align-Self



Codigo:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-
scale=1.0">
 <title>Flex</title>
</head>

<style>
 .container{
 display: flex;
 background-color:blueviolet;
 width:600px;
 height:100px;
 justify-content: center;
 margin-left:500px;
 border:5px solid black;
 margin-bottom :3px;
 align-items:center;
 }

 .Padrao{
 border: 5px solid coral;
 margin: 6px;
 text-align: center;
 width: 100px;
 flex-shrink: 1;
 }

 .Titulo{
 display:flex;
 background-color:burlywood;
 justify-content: center;
 width:600px;
 margin-left:500px;
 }

 .Align-Self1{
 Align-Self:flex-end;
 }

 .Align-Self1:hover{
 Align-Self:flex-start;
 }

 .Align-Self2{
```



Sites recomendados:

[flex-basis - CSS: Cascading Style Sheets | MDN \(mozilla.org\)](#)

[box-sizing - CSS: Cascading Style Sheets | MDN \(mozilla.org\)](#)

[Flexbox CSS: guia completo, elementos e exemplos | Alura](#)

# CSS: Media queries

Autor: Martin Röpke

## Para que serve?

**Media queries** são utilizadas para criar regras CSS, que são **aplicadas apenas** em determinados **tipos de mídia**, como tela de computador, impressões, televisão, etc. Ou se uma determinada **condição lógica**, em relação as características da mídia, como largura, altura e orientação(modos paisagem ou retrato) da tela, **for verdadeira**.

## Exemplo:

Se a janela do navegador tiver uma **largura de 700px ou menos**, a **cor do fundo** ficará **vermelho claro**.

CSS:

```
@media screen and (max-width: 700px) {
 body {
 background-color: lightcoral;
 }
}
```

Código HTML para testar:

```
<!DOCTYPE html>
<html>

<head>
 <style>
 body {
 background-color: lightgreen;
 }

 @media screen and (max-width: 600px) {
 body {
 background-color: lightcoral;
 }
 }
 </style>
</head>

<body>

 <p>Altere a largura da janela para mudar a cor</p>

</body>

</html>
```

## Sintaxe de uma Media Query:

@media screen and (max-width: 700px)

### ▼ @media

Regra CSS utilizada para dizer que uma media query está sendo utilizada, esta regra é uma **at-rule** CSS, clique no **at-rule** para saber mais a respeito.

### ▼ Tipos de mídia

Mídia escolhida para aplicar as propriedades CSS. Dentre as opções no tipo de mídia existem:

- **all:** Aplicar em todos os dispositivos.
- **print:** Aplica em documentos que são vistos em uma pré-visualização de uma impressão ou qualquer mídia que quebre o conteúdo em páginas com a intenção de imprimir.
- **screen:** aplica em dispositivos com uma tela.
- **speech:** aplica em dispositivos que leem o conteúdo de forma audível, como leitor de tela.



### ▼ Características da mídia

Após escolher o tipo de mídia, é possível escolher as **características** que a **mídia** deve ter para ter as regras aplicadas. No exemplo acima foi utilizada a **largura da tela**, porém existem várias outras características que podem ser usadas para aplicar as propriedades, como **cor**, **orientação** e etc. Para ver mais acesse esse link [https://developer.mozilla.org/en-US/docs/Web/CSS/@media#media\\_features](https://developer.mozilla.org/en-US/docs/Web/CSS/@media#media_features).

### ▼ Operadores

Para criar medias queries que dependem de mais de uma condição é possível utilizar **operadores lógicos**. **Tipos de mídia** e **Características da mídia** são as **condições** utilizadas para aplicar as regras CSS. No exemplo acima foi utilizado o operador **and** para juntar a condição de ser uma **mídia que tem uma tela** com a **característica de ter a largura máxima de de 700px**.

Dentro os operadores lógicos possíveis existem:

- **Not:** Inverte o significado de uma Media query inteira.

```
/* Aplica se o dispositivo não puder passar o mouse sobre os elementos */
@media not screen and (hover: hover) {
 body {
 background-color: none;
 }
}
```

- **Only:** É um operador lógico um pouco especial e oculta toda a consulta para navegadores mais antigos. Em outras palavras, os navegadores mais antigos não entendem a palavra-chave **Only**, então toda a consulta de mídia é ignorada. Caso contrário só não tem efeito.

```
@media only screen and (color) {
 /* Styles! */
}
```

- **And:** Combina uma característica de mídia com um tipo de mídia ou outras característica de mídia.

```
/* Apliaca se a tela tiver largura
maior ou igual a 320px e menor
ou igual a 768px */
@media screen (min-width: 320px) and (max-width: 768px) {
 .element {
 /* Styles! */
 }
}
```

- **Or (ou separado por virgula):** Você pode definir várias consultas separadas por vírgulas, caso em que as vírgulas atuam como operadores lógicos ou e a consulta se torna uma lista de consultas.

```
/* Aplica a telas em que o usuário prefere o modo escuro
ou a tela tem pelo menos 1200px de largura*/
@media screen (prefers-color-scheme: dark), (min-width 1200px) {
 .element {
 /* Styles! */
 }
}
```

## Referências:

[https://www.w3schools.com/css/css3\\_mediaqueries.asp](https://www.w3schools.com/css/css3_mediaqueries.asp)

[https://www.w3schools.com/css/css\\_rwd\\_mediaqueries.asp](https://www.w3schools.com/css/css_rwd_mediaqueries.asp)

[https://developer.mozilla.org/en-US/docs/Web/CSS/Media\\_Queries/Using\\_media\\_queries](https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries)

<https://css-tricks.com/a-complete-guide-to-css-media-queries/>

<https://www.digitalocean.com/community/tutorials/css-media-queries#not-logical-operator>

# Media Queries

Autor: Gustavo Lourenço Losada

Como visto anteriormente media queries definem estilos diferentes de exibição para tipos diferentes de mídia. Sintaxe:

@media <tipo de mídia> <operador> (<característica da mídia>)

E caso queira acrescentar mais características para deixar mais restritivo, pode acrescentar no final o operador ‘and’ e mais uma característica, mas caso queira adicionar mais um caso em que pode ocorrer esse efeito use ‘or’ ou uma ‘,’.

Algumas características de media querie podem ter os prefixos ‘min-’ e ‘max-’ para substituir o uso dos símbolos ‘>’ e ‘<’, que significam respectivamente ‘maior ou igual a’ e ‘menor ou igual a’.

Uma das características que suportam o ‘min/max’ e é bem fácil de usar é a largura (width), e tem também a altura (height).

- 1) Faça uma página que quando a largura for maior que 1000px fique roxa, quando menor que 200px fique azul, mas quando a altura for menor que 400px e menor que 200px a tela deve ficar preta. (Mostrar página funcionando para o monitor).  
O que acontece quando duas dessas características são verdadeiras? Qual cor que domina sobre as outras? Por que isso acontece?

Uma outra característica bem intuitiva faz a relação entre altura e largura, chamada de ‘orientation’ (orientação), que percebe quando a tela está no sentido paisagem ou retrato (celular deitado [horizontal] e celular em pé [vertical]).

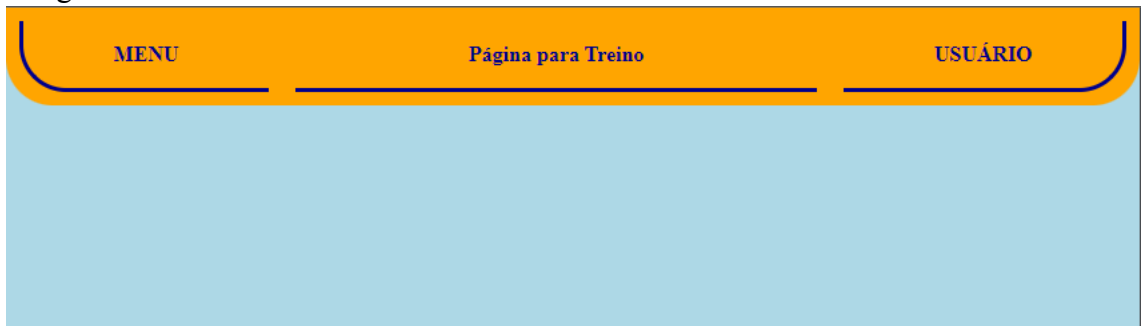
Embora as mudanças até agora só tenham tratado sobre cores, é possível alterar vários elementos e suas formas de display/exibição.

- 2) Isso inclui uma barra que fica no topo da página, possivelmente mostrando opções de rotas que o usuário pode clicar (como exemplo o login). Faça uma página que quando a tela estiver como retrato, as opções de rota fiquem empilhadas uma sobre a outra (imagem1). E quando estiver como paisagem as opções fiquem uma do lado da outra (imagem2).

Imagem1:



Imagem2:



Pode usar qualquer coisa, no exemplo foi utilizado display flex e outros elementos usando/aprendidos anteriormente.

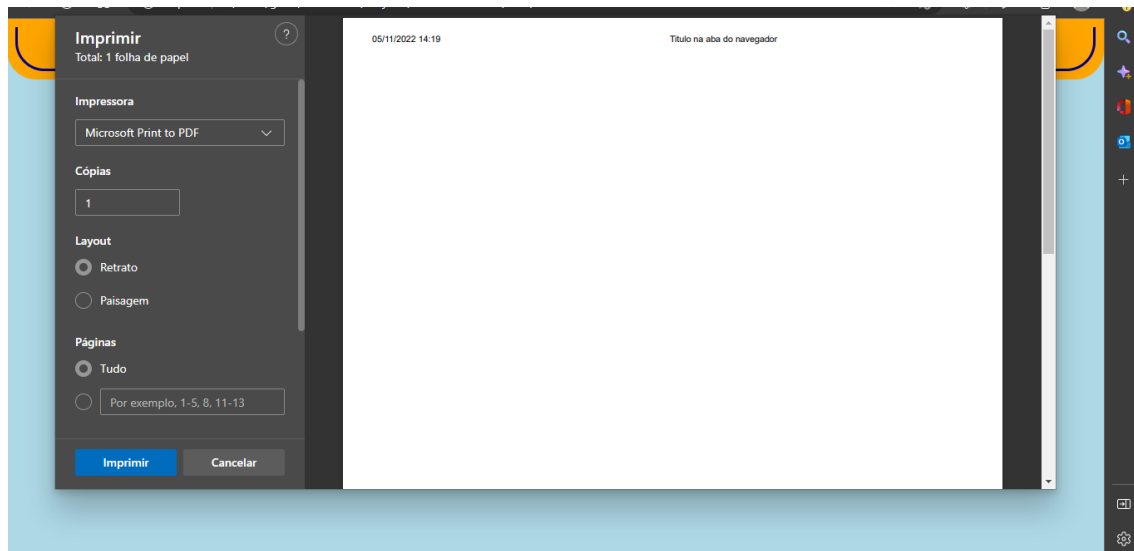
Existem outras características que podem ser usadas para mudar a página de acordo com a mídia, como exemplo a resolução (resolution) que verifica a resolução, ou densidade de pixel) no dispositivo de saída, cor (color), color-index, aspect-ratio, grid e outras características que não serão vistas.

- 3) Para finalizar, finja que você está encarregado de fazer uma página para um grupo secreto, as pessoas podem acessar a página, mas não podem imprimir o conteúdo da página. Ache um jeito de quando a pessoa clicar nos 3 pontos do canto superior

direito e selecionar a opção de imprimir (ou Ctrl + P), todos os elementos / conteúdo da página suma, para que não seja possível imprimir a página.

Dica1: Crie uma classe que não muda o elemento, e adiciona essa classe em todos os elementos.

Dica2: none, display e tipo de mídia.

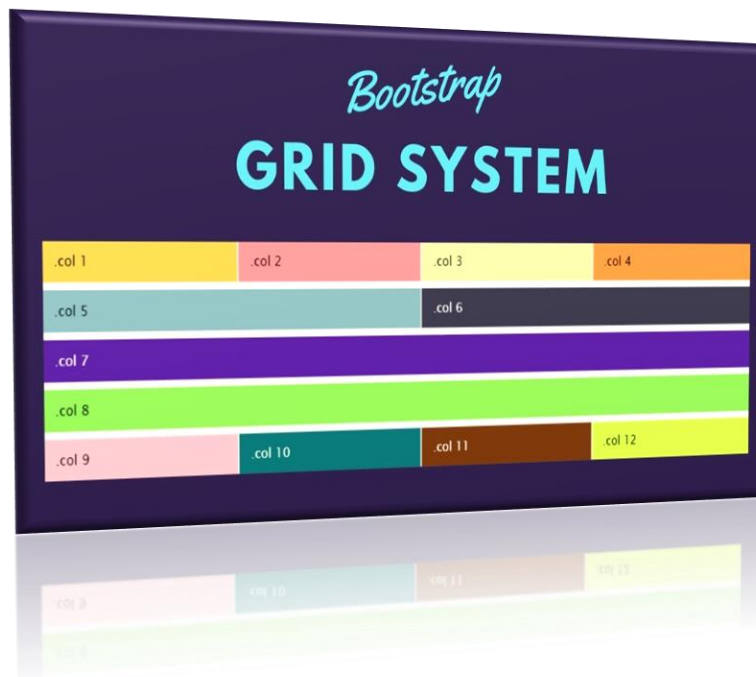


Autor: João vitor Quirino Sarti

GitHub: [NULLBYTE-RGH \(github.com\)](https://github.com/NULLBYTE-RGH)



## Bootstrap 5: Grid system



### Grid System:

O Grid nada mais é do que uma biblioteca que encapsula o tipo de (display Flex). Sendo assim o Bootstrap nada mais vai fazer do que colocar uma camada de abstração ao display, facilitando chamadas e encapsulando várias propriedades em um único comando.

Uma forma de testar e comparar ao código feito na lista 6 (display flex.), é utilizando a propriedade (container) do Bootstrap:

```
<div class="container"></div>
```

Essa propriedade nada mais é do que fazer: (em css puro)

```
.container{
 display: flex;
 justify-content: center;
}
```

Só que o Bootstrap traz a vantagem de várias propriedades de responsividade pré-definidas.

Para fazer com que elementos internos se comportem como (display flex) e (inline block) se faz uso do comando:

- Faz com que todo conteúdo filho seja em uma linha, respeitando o container Flex.

```
<div class="row"></div>
```

- Faz com que todo conteúdo filho seja parte de uma coluna.

```
<div class="col"></div>
```

---

Column	Column	Column
--------	--------	--------

Código:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-
scale=1.0">
 <title>Bootstrap 5 : Grid system</title>
 <!-- Incluindo o Bootstrap 5 via CDN-->
 <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min
.css" rel="stylesheet" integrity="sha384-
Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeu0xjzrPF/et3URy9Bv1WTRi"
crossorigin="anonymous">
 <!-- alguns recursos Bootstrap 5 precisam de código JS para
funcionar, como menus interativos -->
 <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/js/bootstrap.bundl
e.min.js" integrity="sha384-
OERcA2EqjJCMA+/3y+gxIOqMEjwtxJY7qPCqsdltbNJua0e923+mo//f6V8Qbsw3"
crossorigin="anonymous"></script>
</head>

<body>
 <div class="container">
 <div class="row">
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col ">
 Column
 </div>
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col ">
 Column
```

```

 </div>
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col ">
 Column
 </div>
 </div>
</div>
</body>
</html>

```

O Bootstrap tem 2 formas de ser utilizado: via CDN, com uso dos links:

```

<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min
.css" rel="stylesheet" integrity="sha384-
Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeu0xjzrPF/et3URy9Bv1WTRi"
crossorigin="anonymous">

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/js/bootstrap.bundl
e.min.js" integrity="sha384-
OERcA2EqjJCMA+/3y+gxIOqMEjwtxJY7qPCqsdltbNJua0e923+mo//f6V8Qbsw3"
crossorigin="anonymous"></script>

```

E podendo ser baixado a biblioteca do próprio site oficial:

<https://getbootstrap.com/>

Vantagem do CDN: não é necessário baixar, mas a cada vez que a página é carregada ela tem que ir até o link e obter o conteúdo

Vantagem do download: permite desenvolvimento offline e modificação das bibliotecas antes de fazer o envio ao cliente.

## **Tamanhos:**

O Grid faz uso de 6 unidades de medidas padrões:

- Extra small (xs)
- Small (sm)
- Medium (md)
- Large (lg)
- Extra large (xl)
- Extra extra large (xxl)

Isso quer dizer que se pode fazer todo o conteúdo ser responsivo a diferentes tamanhos de telas:

- **Xs** <576px
- **Sm** ≥576px
- **Md** ≥768px
- **Lg** ≥992px
- **Xl** ≥1200px
- **Xxl** ≥1400px

O uso dessas métricas se faz através d0 (-) após a classe:

`col-sm col-md col-lg col-xl col-xxl`

Junto ao tamanho mínimo de tele, poder se fazer alinhamentos com tamanhos pré-definidos de containers, no caso de (COL) pode-se variar de 1 a 12:

`col-sm-1 col-sm-2 col-sm-5`

ou apenas

`col-3 col-6 col-12`



```
<body>
 <div class="container">
 <div class="row">
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-sm-11">
 Column 11
 </div>
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-5 ">
 Column 5
 </div>
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-4 ">
 Column 4
 </div>
 </div>
 </div>

 <div class="container">
 <div class="row">
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-sm-7 ">
 Colum-7
 </div>
```



```

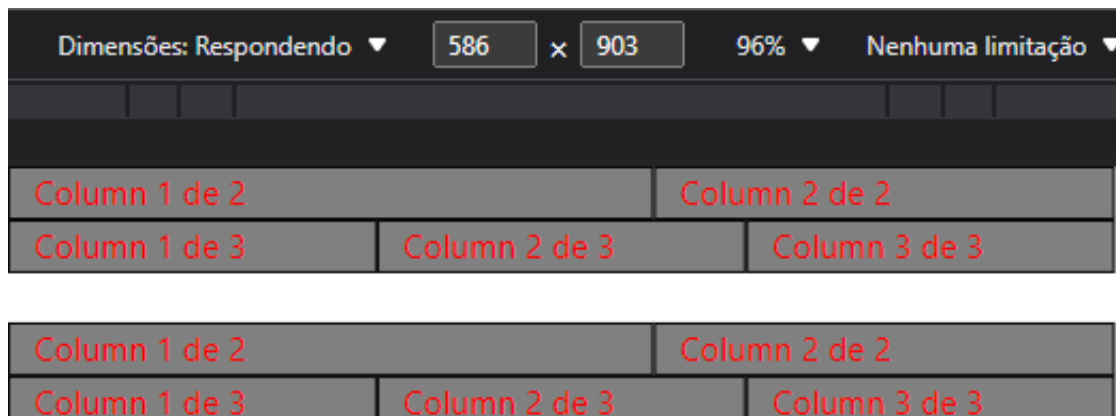
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-3">
 Colum 3
 </div>
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-2 ">
 Column 2
 </div>
 </div>
</div>
</body>

```

### Tamanho variável e empilhamento:

Assim como no display flex, o tamanho dos componentes sempre tenta ser respeitados, e assim, mantando o maior sempre maior que os elementos menores, independentemente do tamanho da tela. Que no caso do CSS puro era aplicado com o Flex-grow ou Flex-shrink.

Com o Grid System, isso ocorre de forma automática:



Dimensões: Respondendo ▾		575	×	903	96% ▾	Nenhuma limitação ▾
Column 1 de 2						
Column 2 de 2						
Column 1 de 3						
Column 2 de 3						
Column 3 de 3						
Column 1 de 2						
Column 2 de 2						
Column 1 de 3	Column 2 de 3			Column 3 de 3		

Com esse código, ao se redimensionar a tela até 576 px de largura a proporção se mantém, já se for menor as TAGS que possuem o tamanho em (SM) alteram seu comportamento perdendo a proporção estimada para o limite de 576 px (SM). E teriam que ser inserida outra propriedade para o mesmo conjunto de classes dizendo a proporção para dispositivos menores que SM, que no caso seria XS. Sendo assim, pode-se definir as proporções para cada tamanho de tela, de forma fácil em apenas uma instanciação de classe.

`col-sm-7 col-md-2`

Isso significaria que até 768px ele terá o comprimento de 2 unidades de 12, já ao atingir 576px ele ocupará 7 unidades de 12, assim consequentemente redimensionando o componente ao seu lado.

Caso o tamanho limite seja atingido e nenhuma proporção seja definida os elementos assumem o tamanho de 12 e cada um ocupa uma linha inteira para si. Dessa forma eles acabam se empilhando.

Código:

```
<body>
 <div class="container">
 <div class="row">
 <div style="color:red; border:1px solid black;background-color:grey;" class="col-sm-7">
 Column 1 de 2
 </div>
 <div style="color:red; border:1px solid black;background-color:grey;" class="col-sm-5 ">
 Column 2 de 2
 </div>
 </div>
 <div class="row">
 <div style="color:red; border:1px solid black;background-color:grey;" class="col-sm-4 ">
 Column 1 de 3
 </div>
 </div>
 </div>
```

```
 </div>
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-sm-4 ">
 Column 2 de 3
 </div>
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-sm-4 ">
 Column 3 de 3
 </div>
 </div>
</div>

<div class="container">
 <div class="row">
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-sm-7">
 Column 1 de 2
 </div>
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-5 ">
 Column 2 de 2
 </div>
 </div>
 <div class="row">
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-4 ">
 Column 1 de 3
 </div>
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-4 ">
 Column 2 de 3
 </div>
 <div style="color:red; border:1px solid black;background-
color:grey;" class="col-4 ">
 Column 3 de 3
 </div>
 </div>
</div>
</body>
```

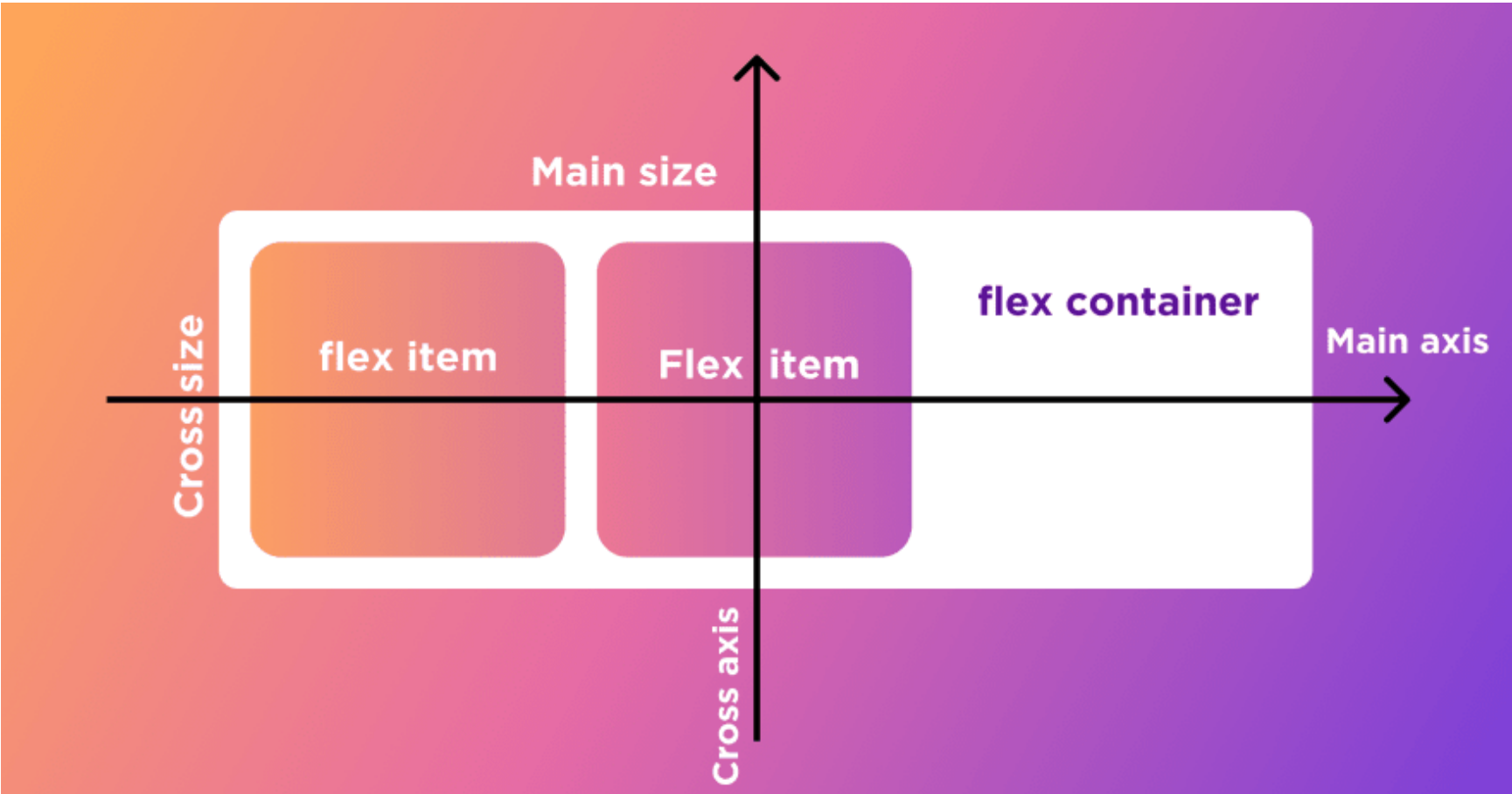
# Bootstrap 5 - Grid System

**Autor: Martin Röpke**

**OBS:** ao copiar códigos certifique-se de **não quebrar a linha do link do bootstrap**, também atente-se a **copiar o código inteiro**, as vezes o **pdf** pode quebrar as páginas.

## Introdução alinhamento no bootstrap Grid System:

Como na propriedade **display flex** do CSS padrão, o **alinhamento** do grid system do bootstrap funciona com referencia em dos **eixos principais**, o **main axis** e o **cross axis**.



O alinhamento no **main axis** é sempre feito através da classe bootstrap **justify-content** e o alinhamento no **cross axis** é sempre feito pela classe bootstrap **align-items**, essas classes serão melhor detalhadas nos capítulos abaixo.

A direção dos eixos pode ser alterada pela propriedade CSS **flex-direction**, no qual o valor padrão é **row**.

Nos exemplos a seguir será utilizado a direção padrão **row**.

## Alinhamento vertical:

### Alinhamento de todos os itens de uma linha

É possível mudar o alinhamento dos itens verticalmente através da classe **align-items**.

Existem três valores possíveis:

- **align-items-start**
- **align-items-center**
- **align-items-end**

uma de três colunas	uma de três colunas	uma de três colunas
uma de três colunas	uma de três colunas	uma de três colunas
uma de três colunas	uma de três colunas	uma de três colunas

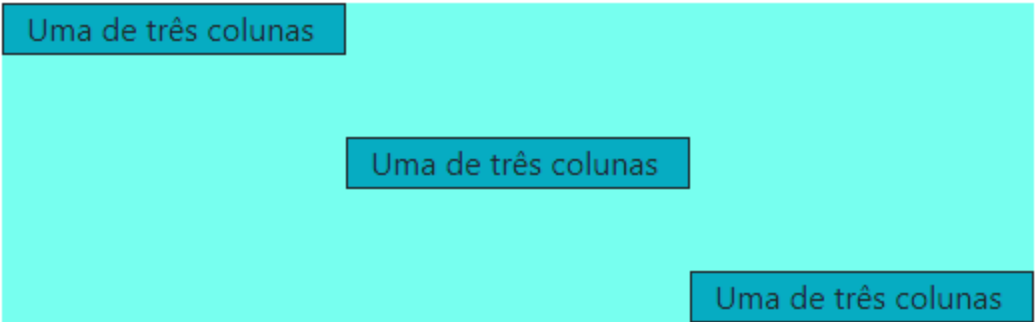
```
<!DOCTYPE html>
<html lang="en">
<head>
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-Ze
nh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeu0xjzrPF/et3URy9Bv1WTRi" crossorigin="anonymous">
 <style>
 .example .col{
 background-color: rgb(6, 172, 194);
 border-style: solid;
 border-width: 1px;
 }

 .example .row{
 height: 10rem;
 background-color: rgba(10, 255, 226, 0.553);
 margin: 0;
 margin-top: 1rem;
 }
 </style>
</head>
<body>
 <div class="container example">
 <div class="row align-items-start">
 <div class="col">
 uma de três colunas
 </div>
 <div class="col">
 uma de três colunas
 </div>
 <div class="col">
 uma de três colunas
 </div>
 </div>
 <div class="row align-items-center">
 <div class="col">
 uma de três colunas
 </div>
 <div class="col">
 uma de três colunas
 </div>
 <div class="col">
 uma de três colunas
 </div>
 </div>
 <div class="row align-items-end">
 <div class="col">
 uma de três colunas
 </div>
 <div class="col">
 uma de três colunas
 </div>
 <div class="col">
 uma de três colunas
 </div>
 </div>
 </div>
```

```
 uma de três colunas
 </div>
 </div>
 </div>
</body>
</html>
```

## Alinhamento vertical separado de cada item de uma linha

Também é possível alterar o alinhamento vertical de cada item separadamente utilizando a classe **align-self** na coluna em vez de **align-items** na linha.



```
<!DOCTYPE html>
<html lang="en">
<head>
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-Ze
nh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeu0xjzrPF/et3URy9Bv1WTRi" crossorigin="anonymous">
 <style>
 .example .col{
 background-color: rgb(6, 172, 194);
 border-style: solid;
 border-width: 1px;
 }

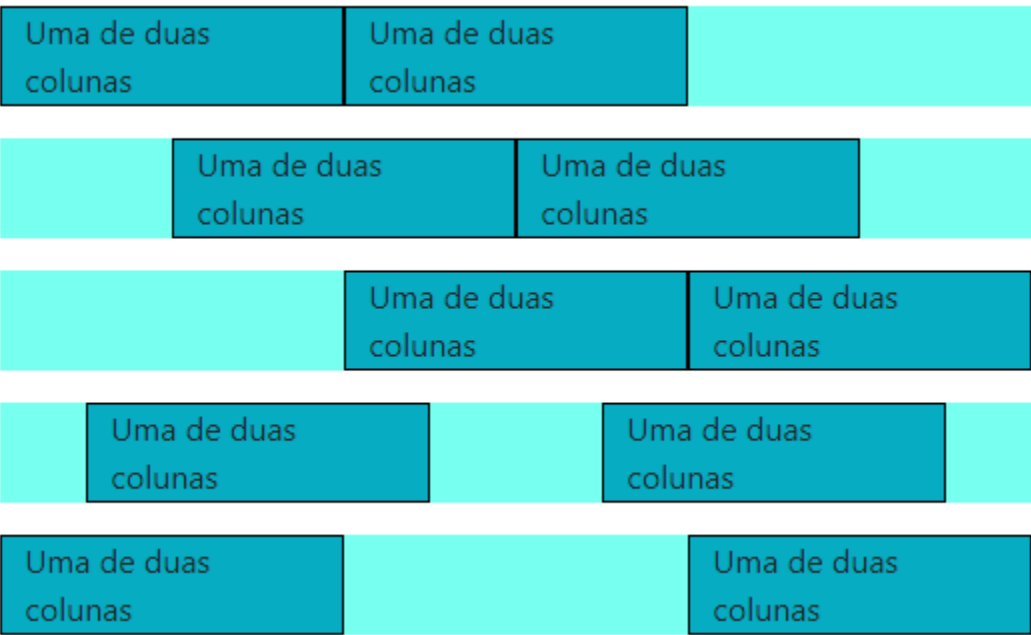
 .example .row{
 height: 10rem;
 background-color: rgba(10, 255, 226, 0.553);
 margin: 0;
 margin-top: 1rem;
 }
 </style>
</head>
<body>
 <div class="container example">
 <div class="row">
 <div class="col align-self-start">
 Uma de três colunas
 </div>
 <div class="col align-self-center">
 Uma de três colunas
 </div>
 <div class="col align-self-end">
 Uma de três colunas
 </div>
 </div>
 </div>
</body>
</html>
```

## Alinhamento Horizontal:

O alinhamento horizontal dos itens em uma linha podem ser alterados através da classe **justify-content**.

Existem cinco valores possíveis:

- **justify-content-start**
- **justify-content-center**
- **justify-content-end:**
- **justify-content-around**
- **justify-content-between**



```
<!DOCTYPE html>
<html lang="en">
<head>
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-Zenh
87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeu0xjzrPF/et3URy9Bv1WTRi" crossorigin="anonymous">
 <style>
 .example .col-4{
 background-color: rgb(6, 172, 194);
 border-style: solid;
 border-color: black;
 border-width: 1px;
 }

 .example .row{
 background-color: rgba(10, 255, 226, 0.553);
 margin: 0;
 margin-top: 1rem;
 }
 </style>
</head>
<body>
 <div class="container example">
 <div class="row justify-content-start">
 <div class="col-4">
 Uma de duas colunas
 </div>
 <div class="col-4">
 Uma de duas colunas
 </div>
 </div>
 <div class="row justify-content-center">
 <div class="col-4">
 Uma de duas colunas
 </div>
 <div class="col-4">
 Uma de duas colunas
 </div>
 </div>
 <div class="row justify-content-end">
 <div class="col-4">
 Uma de duas colunas
 </div>
 <div class="col-4">
 Uma de duas colunas
 </div>
 </div>
 <div class="row justify-content-around">
 <div class="col-4">
 Uma de duas colunas
 </div>
 <div class="col-4">
 Uma de duas colunas
 </div>
 </div>
 <div class="row justify-content-between">
 <div class="col-4">
 Uma de duas colunas
 </div>
 <div class="col-4">
 Uma de duas colunas
 </div>
 </div>
 </div>
</body>
</html>
```

