

COMBINADORES CSS	3
DESCENDANT SELECTOR (SPACE).....	3
CHILD SELECTOR (>)	3
ADJACENT SIBLING SELECTOR (+).....	3
GENERAL SIBLING SELECTOR (~)	4
SELETORES DE ATRIBUTOS.....	4
CSS [ATTRIBUTE] SELECTOR	4
CSS [ATTRIBUTE="VALUE"] SELECTOR	5
CSS [ATTRIBUTE~="VALUE"] SELECTOR	5
CSS [ATTRIBUTE ="VALUE"] SELECTOR	5
CSS [ATTRIBUTE^="VALUE"] SELECTOR	6
CSS [ATTRIBUTE\$="VALUE"] SELECTOR	6
CSS [ATTRIBUTE*="VALUE"] SELECTOR	6
PSEUDO-ELEMENTOS CSS	7
PSEUDO-ELEMENTOS REFERÊNCIA.....	7
PSEUDO-CLASSES CSS	8
PSEUDO-CLASSES REFERÊNCIA	8
FUNÇÃO CALC(_)	10
TRANSFORMAÇÕES	10
MÉTODO TRANSLATE()	10
MÉTODO ROTATE()	10
MÉTODO SCALE()	11
MÉTODO SKEW().....	11
MÉTODO MATRIX()	11
REFERÊNCIA PROPRIEDADES DE TRANSFORMAÇÃO	12
REFERÊNCIA MÉTODOS DE TRANSFORMAÇÃO	12
TRANSFORMAÇÃO 3D	13
MÉTODO ROTATEX()	13
MÉTODO ROTATEY()	13
REFERÊNCIA PROPRIEDADES DE TRANSFORMAÇÃO 3D	13
REFERÊNCIA MÉTODOS DE TRANSFORMAÇÃO 3D	13
COMPATIBILIDADE DE NAVEGADORES.....	14
VERIFICAR SUPORTE DOS NAVEGADORES	14
VARIÁVEIS CSS.....	15

VARIÁVEL DE ESCOPO GLOBAL	15
VARIÁVEL DE ESCOPO LOCAL	16
ALTERAR VARIÁVEIS CSS COM JAVASCRIPT	17
FLEXBOX.....	18
FLEX-DIRECTION.....	18
FLEX-WRAP	19
FLEX-FLOW	20
JUSTIFY-CONTENT	20
ALIGN ITEMS	21
ALIGN-CONTENT	22
CENTRALIZAÇÃO PERFEITA	24
FLEX-ITEMS	24
FLEX-GROW	24
FLEX-SHRINK	25
ORDER	25
ALIGN-SELF	25
CSS GRID	26
DISPLAY GRID	26
GRID COLUMNS.....	26
GRID ROWS	26
GRID GAP	27
GRID LINES	27
GRID-TEMPLATE-COLMNS/ROWS.....	28
ALINHAMENTO CSS GRID	29
PLACE-CONTENT E PLACE-ITEMS	29
JUSTIFY-CONTENT	30
ALIGN-CONTENT	31
JUSTIFY-ITEMS	32
ALIGN-ITEMS	32
JUSTIFY-SELF.....	33
ALIGN-SELF	33
POSICIONAMENTO CSS GRID	34
GRID-COLUMN	34
GRID-ROW	34
GRID-AREA.....	35
GRID-TEMPLATE-AREAS	35
TIPOS DE ALINHAMENTOS CSS.....	36

COMBINADORES CSS

Um combinador é algo que determina especifica uma relação entre dois seletores. Um seletor CSS pode conter mais de um seletor simples, entre eles pode ser incluso um combinador.

DESCENDANT SELECTOR (SPACE)

O seletor de descendentes corresponde a todos os elementos que são descendentes de um elemento específico. O seguinte exemplo seleciona todos os elementos `<p>` dentro de um elemento `<div>`.

<code><head></code>	
<code><style></code>	
<code>div p {</code>	
<code>background-color: yellow;</code>	
<code>}</code>	
<code></style></code>	
<code></head></code>	
<code><body></code>	
<code><div></code>	Paragraph 1 in the div.
<code><p>Paragraph 1 in the div.</p></code>	Paragraph 2 in the div.
<code><p>Paragraph 2 in the div.</p></code>	
<code><section><p>Paragraph 3 in the div.</p></section></code>	Paragraph 3 in the div.
<code></div></code>	
<code><p>Paragraph 4. Not in a div.</p></code>	Paragraph 4. Not in a div.
<code><p>Paragraph 5. Not in a div.</p></code>	
<code></body></code>	Paragraph 5. Not in a div.

CHILD SELECTOR (>)

O seletor de filho seleciona todos os elementos que são filhos de um elemento específico. O seguinte exemplo seleciona todos os elementos `<p>` que são filhos de um elemento `<div>`.

<code><head></code>	
<code><style></code>	
<code>div > p {</code>	
<code>background-color: yellow;</code>	
<code>}</code>	
<code></style></code>	
<code></head></code>	
<code><body></code>	
<code><div></code>	
<code><p>Paragraph 1 in the div.</p></code>	Paragraph 1 in the div.
<code><p>Paragraph 2 in the div.</p></code>	Paragraph 2 in the div.
<code><section></code>	
<code><!-- not Child but Descendant --></code>	Paragraph 3 in the div (inside a section element).
<code><p>Paragraph 3 in the div (inside a section element).</p></code>	Paragraph 3 in the div (inside a section element).
<code></section></code>	
<code><p>Paragraph 4 in the div.</p></code>	Paragraph 4 in the div.
<code></div></code>	
<code><p>Paragraph 5. Not in a div.</p></code>	Paragraph 5. Not in a div.
<code><p>Paragraph 6. Not in a div.</p></code>	Paragraph 6. Not in a div.
<code></body></code>	

ADJACENT SIBLING SELECTOR (+)

O seletor de irmãos adjacentes é usado para selecionar um elemento que está imediatamente após um elemento específico. Sibling (irmãos), pois, eles têm o mesmo parent (pai), e adjacente (adjacente) para significar “imediatamente a seguir”. O seguinte exemplo seleciona o primeiro elemento `<p>` que está localizado imediatamente após o elemento `<div>`.

```

<head>
<style>
div + p {
  background-color: yellow;
}
</style>
</head>
<body>
<div>
  <p>Paragraph 1 in the div.</p>
  <p>Paragraph 2 in the div.</p>
</div>
<p>Paragraph 3. After a div.</p>
<p>Paragraph 4. After a div.</p>
</body>

```

GENERAL SIBLING SELECTOR (~)

O seletor geral de irmãos seleciona todos os próximos elementos irmãos de um elemento específico. Sibling (irmãos), pois, eles têm o mesmo parent (pai). O seguinte exemplo seleciona todos os elementos <p> que são os próximos irmãos de um elemento <div>.

```

<head>
<style>
div ~ p {
  background-color: yellow;
}
</style>
</head>
<body>
<p>Paragraph 1.</p>
<div>
  <p>Paragraph 2.</p>
</div>
<p>Paragraph 3.</p>
<code>Some code.</code>
<p>Paragraph 4.</p>
<p>Paragraph 5.</p>
<p>Paragraph 6.</p>
</body>

```

SELETORES DE ATRIBUTOS

É possível alterar o estilo de um elemento que tenha um atributo específico ou valor de atributo específico através dos seletores de atributos.

CSS [ATTRIBUTE] SELECTOR

O seletor **[atributo]** é usado para selecionar elementos com um atributo específico. O seguinte exemplo seleciona todos os elementos <a> com o atributo "target".

```

<head>
<style>
a[target] {
  background-color: yellow;
}
</style>
</head>
<body>
<a href="https://www.w3schools.com">w3schools.com</a>
<a href="http://www.disney.com" target="_blank">disney.com</a>
<a href="http://www.wikipedia.org" target="_top">wikipedia.org</a>
</body>

```

CSS [ATTRIBUTE="VALUE"] SELECTOR

O seletor **[atributo="valor"]** é usado para selecionar um elemento com um valor específico de atributo.

```
<head>
<style>
a[target=_blank] {
  background-color: yellow;
}
</style>
</head>
<body>
<a href="https://www.w3schools.com">w3schools.com</a>
<a href="http://www.disney.com" target="_blank">disney.com</a>
<a href="http://www.wikipedia.org" target="_top">wikipedia.org</a>
</body>
```

w3schools.com **disney.com** wikipedia.org

CSS [ATTRIBUTE~="VALUE"] SELECTOR

O seletor **[atributo~="valor"]** é usado para selecionar elementos com um valor específico de um atributo contendo uma palavra específica.

```
<head>
<style>
[title~=flower] {
  border: 5px solid yellow;
}
</style>
</head>
<body>



</body>
```



CSS [ATTRIBUTE|= "VALUE"] SELECTOR

O seletor **[atributo|= "valor "]** é usado para selecionar elementos com um atributo específico e um valor específico ou esse valor seguido de um hífen (-). O valor deve ser uma palavra inteira, sozinha, como class="top", ou seguida por um hífen(-), como class="top-text".

```
<head>
<style>
[class|=top] {
  background: yellow;
}
</style>
</head>
<body>
<h1 class="top-header">Welcome</h1>
<p class="top-text">Hello world!</p>
<p class="topcontent">Are you learning CSS?</p>
</body>
```

Welcome


Hello world!

Are you learning CSS?

CSS [ATTRIBUTE^="VALUE"] SELECTOR

O seletor **[atributo^="valor"]** é usado para selecionar um elemento com atributo específico do qual o valor começa com um valor específico. O valor não precisa ser uma palavra inteira.

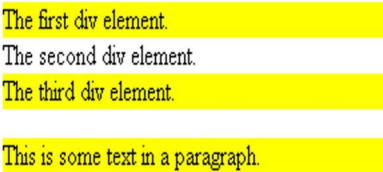
```
<head>
<style>
[class^="top"] {
  background: yellow;
}
</style>
</head>
<body>
<h1 class="top-header">Welcome</h1>
<p class="top-text">Hello world!</p>
<p class="topcontent">Are you learning CSS?</p>
</body>
```



CSS [ATTRIBUTE\$="VALUE"] SELECTOR

O seletor **[atributo\$="valor"]** é usado para selecionar elementos do qual o valor de atributo terminar com um valor específico. O valor não precisa ser uma palavra inteira.

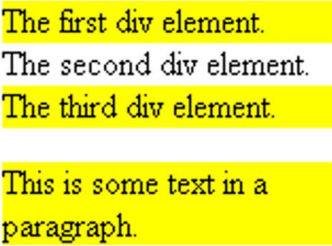
```
<head>
<style>
[class$="test"] {
  background: yellow;
}
</style>
</head>
<body>
<div class="first_test">The first div element.</div>
<div class="second">The second div element.</div>
<div class="my-test">The third div element.</div>
<p class="mytest">This is some text in a paragraph.</p>
</body>
```



CSS [ATTRIBUTE*="VALUE"] SELECTOR

O seletor **[atributo*="valor"]** é usado para selecionar elementos do qual o valor de atributo contém um valor específico em qualquer parte. O valor não precisa ser uma palavra inteira.

```
<head>
<style>
[class*="te"] {
  background: yellow;
}
</style>
</head>
<body>
<div class="first_test">The first div element.</div>
<div class="second">The second div element.</div>
<div class="aateaa">The third div element.</div>
<p class="mytest">This is some text in a paragraph.</p>
</body>
```



PSEUDO-ELEMENTOS CSS

Em CSS pseudo-elementos são utilizados para estilizar partes específicas de um elemento. Como por exemplos, o estilo da primeira letra, primeira linha, inserir um conteúdo antes ou depois do conteúdo de um elemento.

A sintaxe dos pseudo-elementos é:

```
selector::pseudo-element {
  property: value;
}
```

O seguinte exemplo exibirá a primeira letra dos parágrafos com class="intro", em vermelho e em tamanho maior.

```
<head>
<style>
p.intro::first-letter {
  color: #ff0000;
  font-size: 200%;
}
</style>
</head>
<body>
<p class="intro">This is an introduction.</p>
<p>This is a paragraph with some text. A bit more text
even.</p>
</body>
```

This is an introduction.

This is a paragraph with some text. A bit more text even.

PSEUDO-ELEMENTOS REFERÊNCIA

SELETOR	EXEMPLO	DESCRIÇÃO DO EXEMPLO
<u>::after</u>	p::after	Inserir algo após o conteúdo de cada elemento <p>
<u>::before</u>	p::before	Inserir algo após o conteúdo de cada parágrafo <p>
<u>::first-letter</u>	p::first-letter	Seleciona a primeira letra de cada elemento <p>
<u>::first-line</u>	p::first-line	Seleciona a primeira linha de cada elemento <p>
<u>::marker</u>	::marker	Seleciona o marcador dos itens de uma lista
<u>::selection</u>	p::selection	Seleciona a porção de um elemento que está sendo selecionada pelo usuário

PSEUDO-CLASSES CSS

A pseudoclasse em CSS é utilizada para definir um estado especial de um elemento. Como por exemplo estilizar um elemento quando o mouse está sobre ele, estilizar um link que ainda não foi visitado, estilizar um elemento que está em foco, etc.

A sintaxe das pseudoclasses é:

```
selector:pseudo-class {
  property: value;
}
```

No exemplo a seguir, o seletor corresponde a qualquer elemento <p> que seja o primeiro filho de qualquer elemento.

```
<head>
<style>
p:first-child {
  color: blue;
}
</style>
</head>
<body>
<p>This is some text.</p>
<p>This is some text.</p>
<div>
  <p>This is some text.</p>
  <p>This is some text.</p>
</div>
</body>
```

This is some text.

This is some text.

This is some text.

This is some text.

PSEUDO-CLASSES REFERÊNCIA

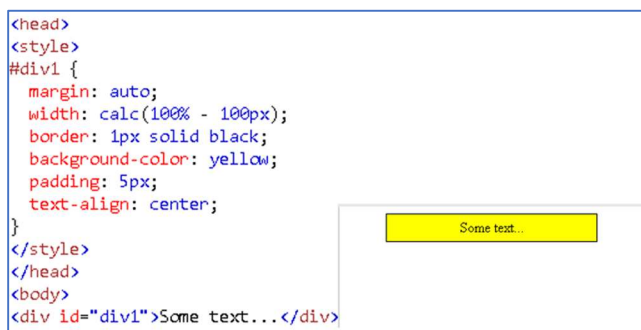
SELETOR	EXEMPLO	DESCRIÇÃO DO EXEMPLO
<u>:active</u>	a:active	Seleciona um link que já foi clicado (ativo)
<u>:checked</u>	input:checked	Seleciona todos os elementos input checked (preenchido / marcado)
<u>:disabled</u>	input:disabled	Seleciona todos os elementos input que estão desabilitados
<u>:empty</u>	p:empty	Seleciona todos os elementos <p> que não possuem filhos ou vazios
<u>:enabled</u>	input:enabled	Seleciona todos os elementos input que estão habilitados
<u>:first-child</u>	p:first-child	Seleciona todos os elementos <p> que são os primeiros filhos de qualquer elemento
<u>:first-of-type</u>	p:first-of-type	Seleciona todos os elementos <p> que são os primeiros de seu tipo
<u>:focus</u>	input:focus	Seleciona o elemento <input> que está em foco
<u>:hover</u>	a:hover	Seleciona o link em que o mouse está sobre ele

<u>:in-range</u>	input:in-range	Seleciona um elemento <input> com um valor dentro de um intervalo determinado
<u>:invalid</u>	input:invalid	Seleciona todos os elementos <input> com um valor invalido
<u>:lang(language)</u>	p:lang(it)	Seleciona todos elementos <p> com um atributo de linguagem começado com "it"
<u>:last-child</u>	p:last-child	Seleciona todos os elementos <p> que são os últimos filhos de um pai
<u>:last-of-type</u>	p:last-of-type	Seleciona todos elementos <p> que são os últimos do seu tipo
<u>:link</u>	a:link	Seleciona todos os links não visitados
<u>:not(selector)</u>	:not(p)	Seleciona todos os elementos que não são um elemento <p>
<u>:nth-child(n)</u>	p:nth-child(2)	Seleciona todos os elementos <p> que são os segundos filhos de um pai
<u>:nth-last-child(n)</u>	p:nth-last-child(2)	Seleciona todos os elementos <p> que são os segundos filhos de um pai, contando a partir do ultimo
<u>:nth-last-of-type(n)</u>	p:nth-last-of-type(2)	Seleciona todos os elementos <p> que são os segundos do seu tipo, contando a partir do ultimo
<u>:nth-of-type(n)</u>	p:nth-of-type(2)	Seleciona todos os elementos <p> que são os segundos do seu tipo
<u>:only-of-type</u>	p:only-of-type	Seleciona todos os elementos <p> que são os únicos do seu tipo de um elemento pai
<u>:only-child</u>	p:only-child	Seleciona todos os elementos <p> que são os únicos filhos de um elemento pai
<u>:optional</u>	input:optional	Seleciona os elementos <input> que não tenham o atributo "required" especificado
<u>:out-of-range</u>	input:out-of-range	Seleciona todos os elementos <input> com o valor fora de um intervalo determinado
<u>:read-only</u>	input:read-only	Seleciona todos os elementos <input> com o atributo "readonly" especificado
<u>:read-write</u>	input:read-write	Seleciona todos os elementos <input>
<u>:required</u>	input:required	Seleciona os elementos <input> que tenham o atributo "required" especificado
<u>:root</u>	root	Seleciona o elemento raiz do documento
<u>:target</u>	#news:target	Seleciona o elemento #news que está ativo atualmente (clicado em um URL que contém esse nome de âncora)
<u>:valid</u>	input:valid	Seleciona todos os elementos <input> com um valor válido
<u>:visited</u>	a:visited	Seleciona todos os links que já foram visitados/acessados

FUNÇÃO CALC()

A função `calc()` executa um cálculo que pode ser usado como valor de uma propriedade. É necessária uma expressão matemática em que o resultado será usado como o valor. Os seguintes operadores podem ser usados: `+` `-` `*` `/`.

No seguinte exemplo a largura de um elemento é determinada com a função `calc()` que especifica que a largura de um elemento `<div>` deve ser `100% - 100px`, ou seja, um espaço de 50px entre as duas bordas da janela.

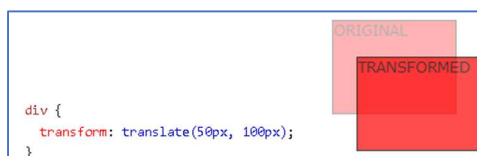


TRANSFORMAÇÕES

Transformações CSS permitem mover, rotacionar, dimensionar e torcer elementos.

MÉTODO TRANSLATE()

O método `translate()` move um elemento de sua posição atual de acordo com os parâmetros fornecidos para o eixo X e o eixo Y. O seguinte exemplo move um elemento 50 pixels para direita e 100 pixels para baixo de sua posição atual.



MÉTODO ROTATE()

O método `rotate()` rotaciona um elemento no sentido horário ou anti-horário de acordo com um determinado grau. O uso de valores negativos girará o elemento no sentido anti-horário. O exemplo a seguir gira o elemento `<div>` no sentido horário com 20 graus.

```
div {
  transform: rotate(20deg);
}
```



MÉTODO SCALE()

O método `scale()` aumenta ou diminui o tamanho de um elemento de acordo com parâmetros fornecidos como largura e altura. O seguinte exemplo aumenta o elemento `<div>` para ser duas vezes maior que o tamanho original de largura e três vezes maior que o tamanho original de altura.

```
div {
  transform: scale(2, 3);
}
```



MÉTODO SKEW()

O método `skew()` “torce” um elemento ao longo do eixo X e Y de acordo com um ângulo fornecido. Se o segundo parâmetro não for especificado o valor do eixo Y vai ser 0. O seguinte exemplo torce um elemento `<div>` 20 graus ao longo do eixo X e 10 graus ao longo do eixo Y.

```
div {
  transform: skew(20deg, 10deg);
}
```



MÉTODO MATRIX()

O `matrix()` método combina todos os métodos de transformação em um. O método `matrix()` aceita seis parâmetros, contendo funções matemáticas, que permitem girar, dimensionar, mover (traduzir) e inclinar elementos. Os parâmetros são os seguintes:

`matrix(scaleX(),skewY(),skewX(),scaleY(),translateX(),translateY())`

```
div {
  transform: matrix(1, -0.3, 0, 1, 0, 0);
}
```

REFERÊNCIA PROPRIEDADES DE TRANSFORMAÇÃO

Propriedade	Descrição
<u>transform</u>	Aplica uma transformação 2D ou 3D em um elemento
<u>transform-origin</u>	Permite alterar a posição de um elemento transformado

REFERÊNCIA MÉTODOS DE TRANSFORMAÇÃO

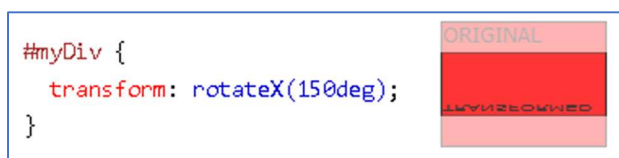
Função	Descrição
<code>matrix(<i>n,n,n,n,n,n</i>)</code>	Define uma transformação 2D usando uma matriz de 6 valores
<code>translate(<i>x,y</i>)</code>	Define uma interpretação 2D do elemento, movendo-o ao longo do eixo X e eixo Y
<code>translateX(<i>n</i>)</code>	Define uma interpretação 2D do elemento, movendo-o ao longo do eixo X
<code>translateY(<i>n</i>)</code>	Define uma interpretação 2D do elemento, movendo-o ao longo do eixo Y
<code>scale(<i>x,y</i>)</code>	Define uma transformação de dimensão alterando a largura e altura de um elemento
<code>scaleX(<i>n</i>)</code>	Define uma transformação de dimensão alterando a largura de um elemento
<code>scaleY(<i>n</i>)</code>	Define uma transformação de dimensão alterando a altura de um elemento
<code>rotate(<i>angle</i>)</code>	Define uma rotação 2D, o ângulo deve ser especificado nos parâmetros
<code>skew(<i>x-angle,y-angle</i>)</code>	Define uma transformação de torção 2D ao longo do eixo X e do eixo
<code>skewX(<i>angle</i>)</code>	Define uma transformação de torção 2D ao longo do eixo X
<code>skewY(<i>angle</i>)</code>	Define uma transformação de torção 2D ao longo do eixo Y

TRANSFORMAÇÃO 3D

CSS também suporta transformações 3D.

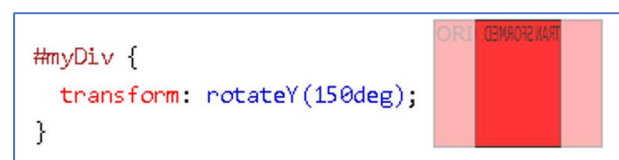
MÉTODO ROTATEX()

O método rotateX() gira um elemento em torno de seu eixo X em um determinado grau



MÉTODO ROTATEY()

O rotateY() método gira um elemento em torno de seu eixo Y em um determinado grau.



REFERÊNCIA PROPRIEDADES DE TRANSFORMAÇÃO 3D

Propriedade	Descrição
<u>transform</u>	Aplica uma transformação 2D ou 3D a um elemento
<u>transform-origin</u>	Permite alterar a posição de elementos transformados
<u>transform-style</u>	Especifica como elementos aninhados são renderizados em um espaço 3D
<u>perspective</u>	Especifica a perspectiva em que elementos 3D são visualizados
<u>perspective-origin</u>	Especifica a posição inferior de um elemento 3D
<u>backface-visibility</u>	Define se um elemento deve ser visível quando não está voltado para a tela

REFERÊNCIA MÉTODOS DE TRANSFORMAÇÃO 3D

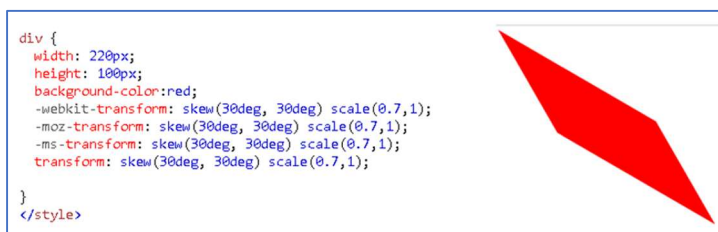
Função	Descrição
matrix3d (n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n)	Define uma transformação 3D usando uma matriz 4x4 de 16 valores
translate3d(x,y,z)	Define uma interpretação 3D do elemento
scale3d(x,y,z)	Define um dimensionamento 3D do elemento
rotate3d(x,y,z,angle)	Define uma rotação 3D
perspective(n)	Define uma visão de perspectiva 3D de um elemento transformado

COMPATIBILIDADE DE NAVEGADORES

Antes de começar a implementar o HTML5 e o CSS3 em seus projetos, é importante você ter consciência de quais marcações e propriedades são suportadas pelas diferentes versões dos diferentes navegadores.

Na verdade, o suporte as linguagens não é feito exatamente pelo navegador, e sim pelo motor de renderização que ele utiliza. Por exemplo, temos o motor Webkit que é usado pelos navegadores Chrome e Safari, o motor Gecko que é utilizado pelo navegador Firefox, o motor Trident usado pelo navegador Internet Explorer, e o motor Presto usado pelo navegador Opera. O Webkit é derivado (fork) do motor KHTML, do navegador Konqueror (facilmente encontrado nas distribuições Linux que utilizam a interface KDE). Atualmente, o Webkit é o motor de renderização que tem maior compatibilidade com o HTML5 e CSS3. Se o seu código HTML5 e CSS3 estiver bem compatível com o Webkit, ele com certeza funcionará bem não só nos desktops com Chrome e Safari, mas também nos celulares e tablets — sejam eles iPhone e iPad ou produtos que levam o Android como sistema operacional. (Edu Agni)

No seguinte exemplo o uso da propriedade transform não apresenta suporte a todos os navegadores ou a versões anteriores, por isso o código foi adaptado para funcionar corretamente em, versões inferiores do Google Chrome (“-webkit-”), versões inferiores do internet explore (“-ms-”) e com versões do navegador Mozilla Firefox (“-moz-”).



VERIFICAR SUPORTE DOS NAVEGADORES

- <https://caniuse.com/> - informações completas sobre o suporte
- <https://shouldiprefix.com/> - informações sobre prefixos
- <https://html5readiness.com/> - infográfico interativo.
- <https://modernizr.com/> - Técnicas para compatibilidade com biblioteca JS.

VARIÁVEIS CSS

A função `var()` é usada para inserir o valor de uma variável CSS. As variáveis CSS podem ser acessadas pelo DOM, o que significa que é possível criar e alterar variáveis CSS com JavaScript. A sintaxe da função das variáveis CSS é:

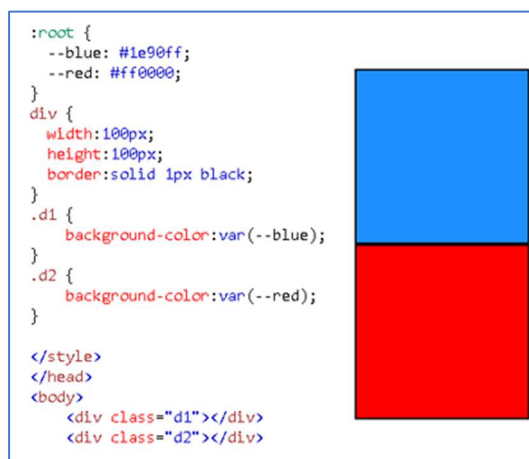
```
var(--name, value)
```

O nome da variável deve sempre começar com dois dashes (--) quando for declarada.

VARIÁVEL DE ESCOPO GLOBAL

Para criar primeiro é preciso declarar o seletor **“:root”**, esse seletor corresponde ao elemento raiz do documento, ou seja, ao “HTML”. Após isso as variáveis podem ser declaradas dentro desse seletor, os seus nomes devem começar com dois dashes (--) e são case sensitive. Após isso essas variáveis podem ser usadas como valores de propriedades para alterar estilos de elementos como uso da função `var()`.

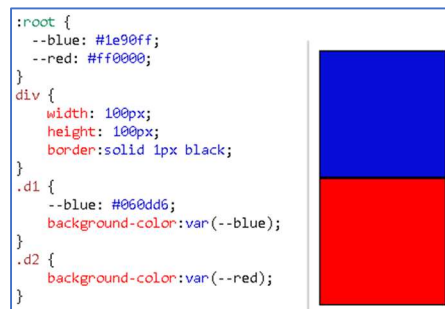
No seguinte exemplo foram declaradas duas variáveis (`--blue` e `--red`), e então utilizada a função `var()` para utilizar o valor dessas variáveis em alguns elementos.



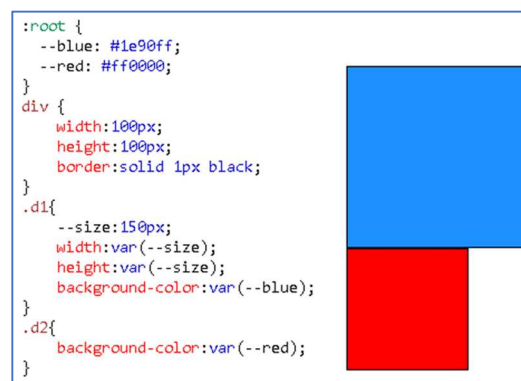
As vantagens de usar `var()` é tornar o código mais fácil de ler (mais compreensível) e tornar muito mais fácil alterar os valores de cor. Para alterar a cor `--blue` e `--red` para um azul e vermelho mais suave, basta alterar os dois valores das variáveis.

VARIÁVEL DE ESCOPO LOCAL

Às vezes queremos que as variáveis mudem apenas em uma seção específica da página. No seguinte exemplo suponha que queremos uma cor diferente de azul para o elemento `<div class="d1">`. Então, podemos declarar novamente a variável `--blue` dentro do seletor da `<div>`. Quando for utilizada a função `var(--blue)` dentro deste seletor, ela usará o valor da variável local `--blue`.



As variáveis de escopo local são declaradas no próprio seletor e não necessitam do seletor `:root`. No seguinte exemplo foi alterado o tamanho do elemento utilizando uma variável de escopo local (`--size`).

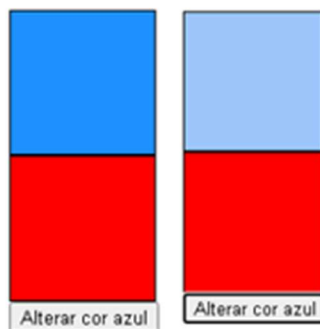


ALTERAR VARIÁVEIS CSS COM JAVASCRIPT

As variáveis CSS têm acesso ao DOM, o que significa que você pode alterá-las com JavaScript. É possível alterar e acessar as variáveis CSS através dos métodos **getComputedStyle()**, para pegar todos as propriedades e valores de estilo de um elemento, **getPropertyValue()**, para pegar o valor de uma propriedade e o método **setProperty()** para definir uma nova variável CSS ou alterar uma variável existente. No exemplo a seguir é demonstrado como criar um script para alterar a variável --blue dos exemplos usados anteriormente.

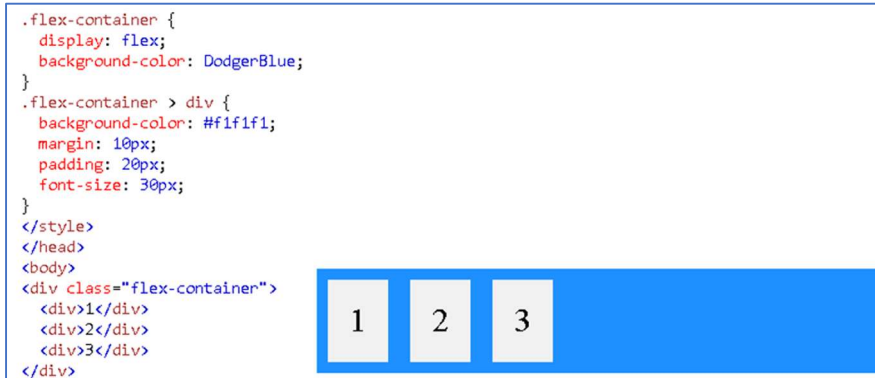
```
<head>
<style>
:root {
  --blue: #1e90ff;
  --red: #ff0000;
}
div {
  width:100px;
  height:100px;
  border:solid 1px black;
}
.d1 {
  background-color:var(--blue);
}
.d2 {
  background-color:var(--red);
}
</style>
</head>

<body>
<div class="d1"></div>
<div class="d2"></div>
<button type="button" onclick="myFunction_set()">Alterar cor azul</button>
<script>
var r = document.querySelector(':root');
function myFunction_set() {
  r.style.setProperty('--blue', '#9cc6fc');
}
</script>
</body>
```



FLEXBOX

Flexbox é um módulo de layout de caixa flexível, através dele é possível deixar o design mais fácil, responsivo e estruturar o layout sem o uso das técnicas de posicionamento. **Para utilizar o modelo Flexbox é preciso definir um “flex container”** (recipiente flexível), esse container se torna flexível quando é atribuído a ele a propriedade de “display:flex”.



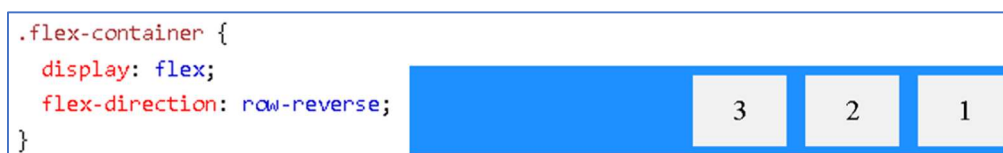
FLEX-DIRECTION

A propriedade **flex-direction** define em qual direção o container vai empilhar/organizar seus flex-items.

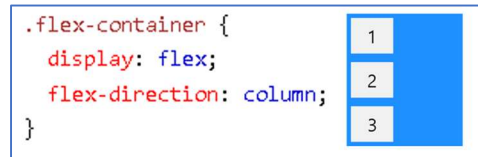
Por padrão a propriedade é definida como **flex-direction:row**, ou seja, os flex-items dentro do container serão organizados horizontalmente em uma “linha”, começando da esquerda para direita.



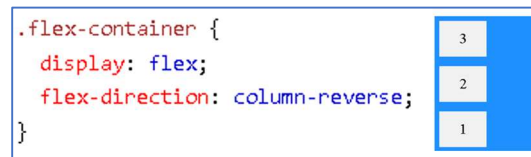
A propriedade definida como **flex-direction:row-reverse** organiza os flex-items de maneira semelhante, porém começando direita para esquerda.



A propriedade definida como **flex-direction:column** organiza os flex-items verticalmente como uma coluna de cima para baixo.



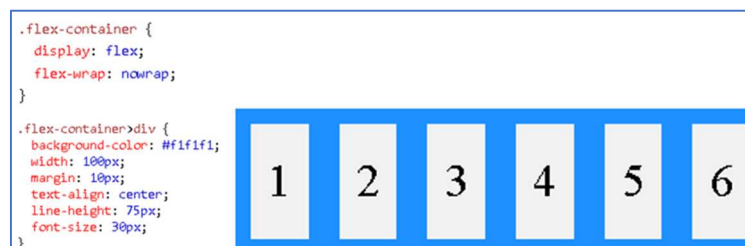
A propriedade definida como **flex-direction:column-reverse** organiza os flex-items de maneira semelhante, porém começando de baixo para cima.



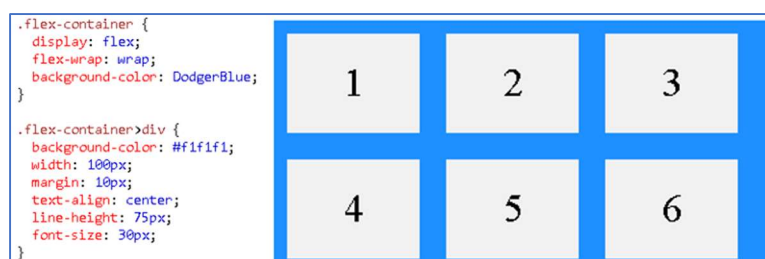
FLEX-WRAP

A propriedade **flex-wrap** especifica se os flex-items devem ser “wrapped” (embrulhados, agrupados), ou seja, essa propriedade define se os tamanhos podem ou não ser alterados para caber dentro do container em uma linha ou coluna.

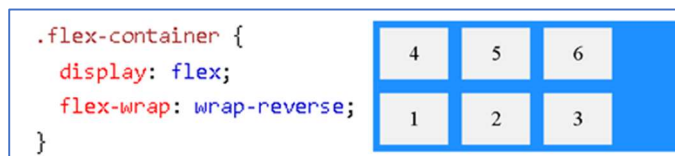
Por padrão o valor da propriedade é **flex-wrap:nowrap**, ou seja, os flex-items serão “espremidos” conforme o tamanho da janela é diminuído para caber dentro do container.



A propriedade definida como **flex-wrap:wrap** vai “espremer” os flex-items somente se necessário (quando o tamanho da janela for menor que o tamanho do item).

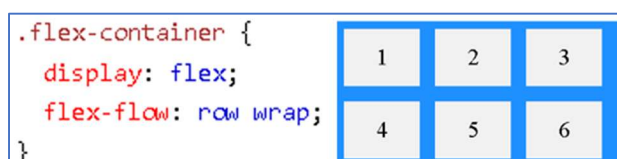


A propriedade definida como **flex-wrap:wrap-reverse** especifica que os itens flexíveis serão agrupados na ordem inversa.



FLEX-FLOW

A propriedade **flex-flow** é uma maneira encurtada para definir as propriedades flex-direction e flex-wrap em uma única linha.



JUSTIFY-CONTENT

A propriedade **justify-content** é usada para alinhar os flex-items de maneira **horizontal** de acordo com a largura dos itens e do container.

Por padrão a propriedade é definida como **justify-content :flex-start**, os seja, os flex-items vão ser alinhados no começo do container.



A propriedade definida com o valor **justify-content :flex-end** alinha os flex-items no final do container.



A propriedade definida com o valor **justify-content :center** alinha os flex-items no centro do container.



A propriedade definida com o valor **justify-content :space-around** exibe os flex-items com espaços antes, entre e depois das linhas ou colunas.



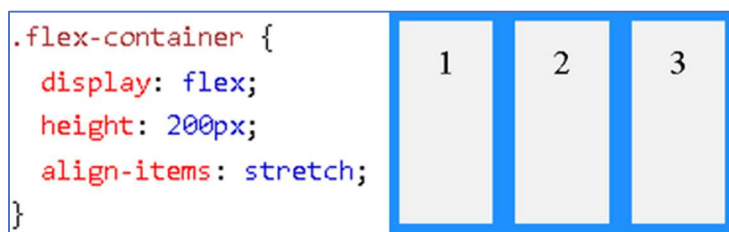
A propriedade definida com o valor **justify-content:space-between** exibe os flex-items com espaços entre eles.



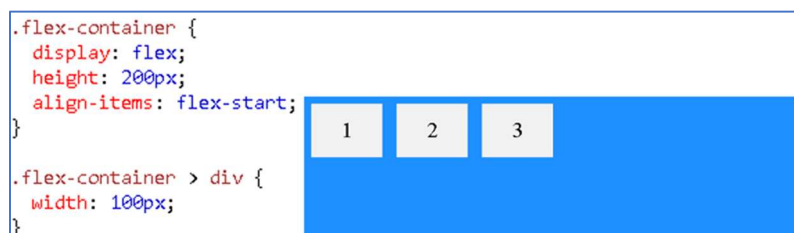
ALIGN ITEMS

A propriedade **align-items** também é usada para alinhar os flex-items, porém de maneira **vertical** de acordo com o tamanho dos itens e do container.

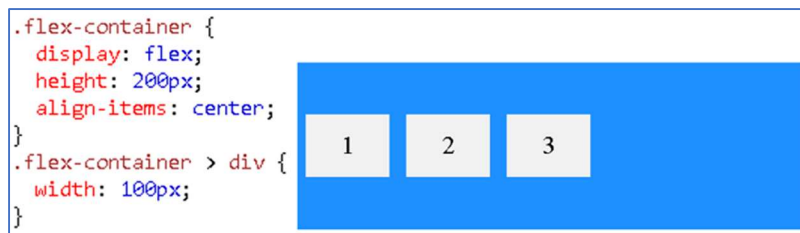
Por padrão a propriedade é definida com o valor **align-items:stretch**. Os flex-items serão “esticados” para preencher o container se o mesmo tiver uma altura definida.



A propriedade definida com o valor **align-items:flex-start** alinha os flex-items na parte de cima do container.



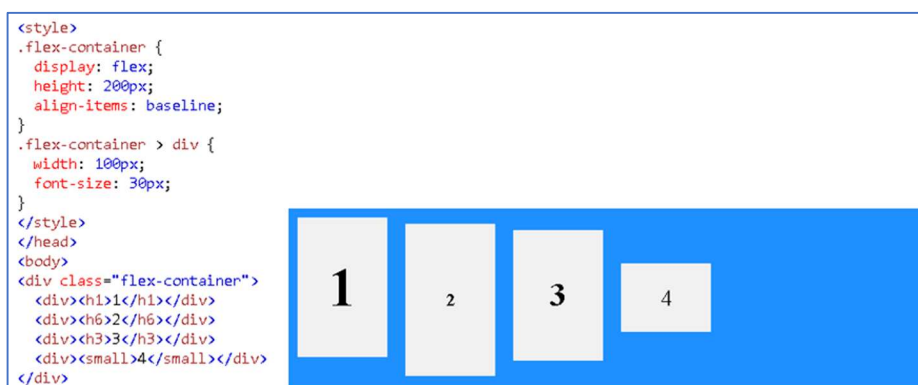
A propriedade definida com o valor **align-items:center** alinha os flex-items no centro do container.



A propriedade definida com o valor **align-items:flex-end** alinha os flex-items na parte de baixo do container.



A propriedade definida com o valor **align-items:baseline** alinha os flex-items de acordo com as suas bases de linha. Tamanhos de fonte diferentes definem linhas de base diferentes, a propriedade com o valor baseline alinha essas bases de linha e por fim altera o tamanho dos flex-items de acordo com o tamanho do container e a linha de base.

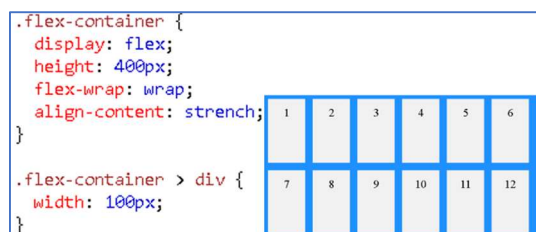


ALIGN-CONTENT

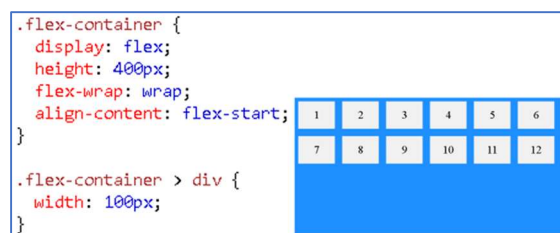
Como já foi mencionado, por padrão os flex-items possuem a propriedade **flex-wrap:nowrap**, ou seja, o tamanho do item vai ser modificado para todos os itens caberem dentro de uma única linha ou coluna do container. Porém caso a propriedade for definida como **flex-wrap:wrap** o container vai criar mais linhas ou colunas para que caber todos os flex-items tentando manter seus tamanhos previamente definidos. A propriedade **align-content** **alinha as linhas e colunas dos flex-items**.



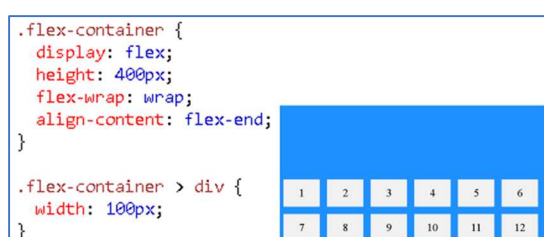
Por padrão a propriedade é definida como **align-content:stretch**, as linhas vão ser esticadas para preencher o espaço do container.



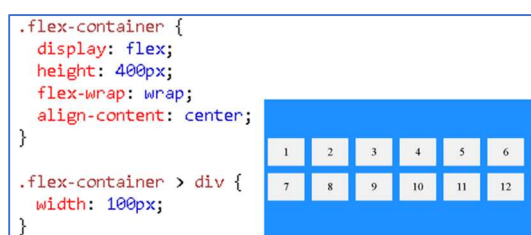
A propriedade definida com o valor **align-content:flex-start** vai e exibir as linhas no começo do container.



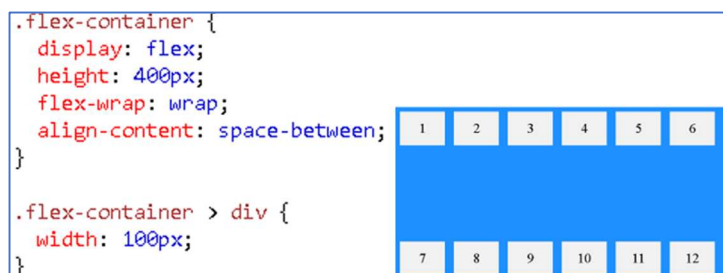
A propriedade definida com o valor **align-content:flex-end** vai e exibir as linhas no final do container.



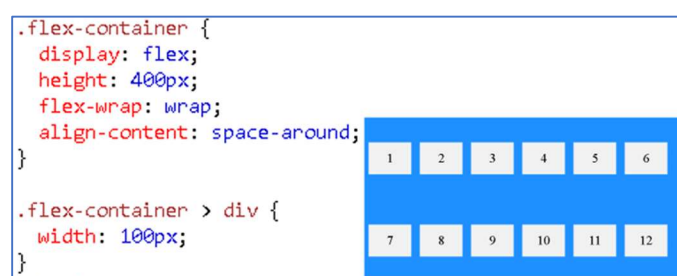
A propriedade definida com o valor **align-content:center** vai e exibir as linhas no centro do container.



A propriedade definida com o valor **align-content: space-between**, exibe as linhas flexíveis com um espaço igual entre elas.

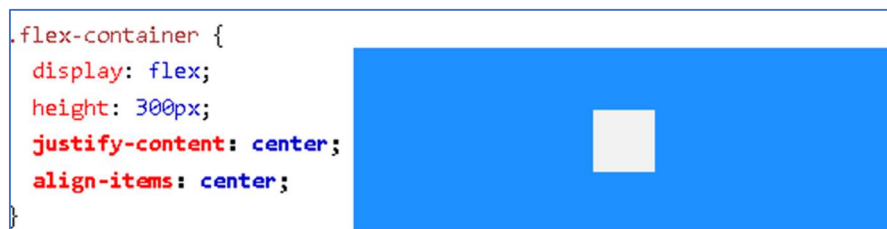


A propriedade definida com o valor **align-content: space-around**, exibe as linhas flexíveis com espaço antes, entre e depois delas.



CENTRALIZAÇÃO PERFEITA

Definindo as propriedades **justify-content** e **align-items** com o valor **center** o item flexível ficará perfeitamente centralizado no container.



FLEX-ITEMS

Os elementos “filhos” de um container com **display: flex** são automaticamente flex-items (flex-items). As propriedades dos flex-items podem ser alteradas.

FLEX-GROW

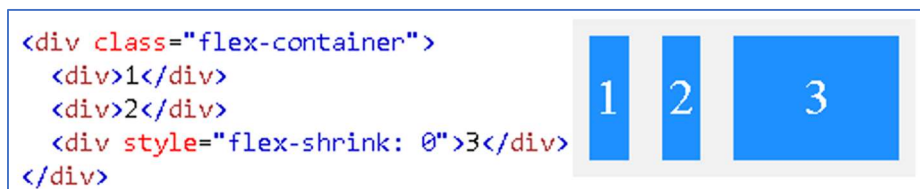
A propriedade **flex-grow** especifica o quanto um flex-item vai aumentar em relação ao restante dos outros flex-items para preencher o container. No

seguinte exemplo o terceiro elemento vai aumentar oito vezes mais que o restante.



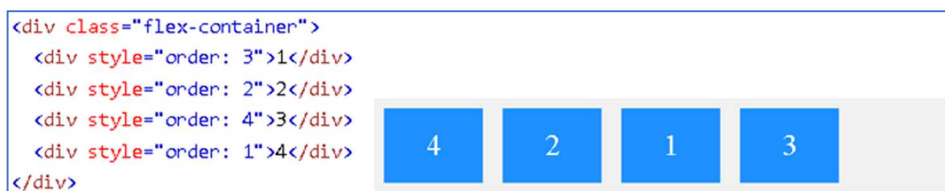
FLEX-SHRINK

A propriedade **flex-shrink** define se um flex-item vai encolher quando o tamanho da janela for diminuído. A propriedade aceita dois valores, sendo “1” o seu valor padrão e “0” que especifica que o item não vai ser encolhido.



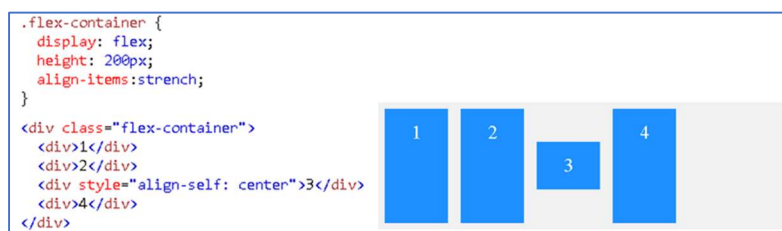
ORDER

A propriedade order especifica a ordem que os flex-items devem aparecer dentro do container.



ALIGN-SELF

A propriedade **align-self** especifica o alinhamento de um item selecionado dentro de um container flexível. A propriedade align-self substitui o alinhamento padrão definido pela propriedade align-items do contêiner.

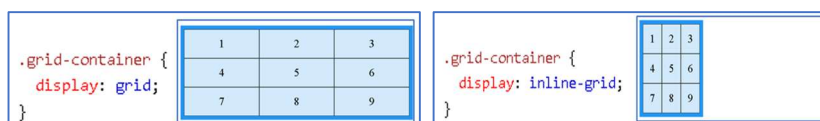


CSS GRID

O módulo de layout GRID CSS oferece um sistema de layout baseado em grade, com linhas e colunas, tornando mais fácil projetar o design de web pages sem utilizar técnicas de objetos flutuantes e de posicionamento.

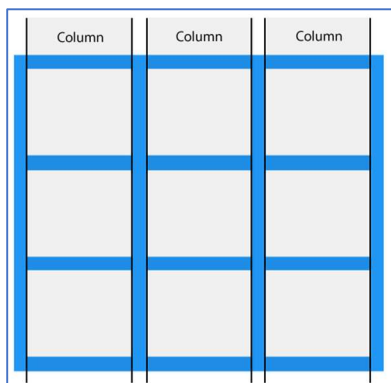
DISPLAY GRID

Um elemento HTML se torna um **grid container** quando a propriedade **display** é definida como **grid** ou **inline-grid**. Todos os filhos diretos de um grid container se tornam automaticamente **grid items**.



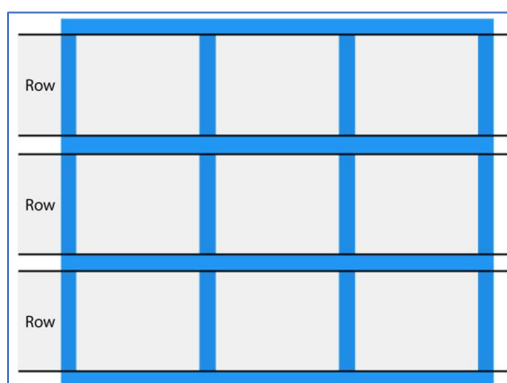
GRID COLUMNS

As fileiras verticais formadas por grid items são chamadas de columns (colunas).



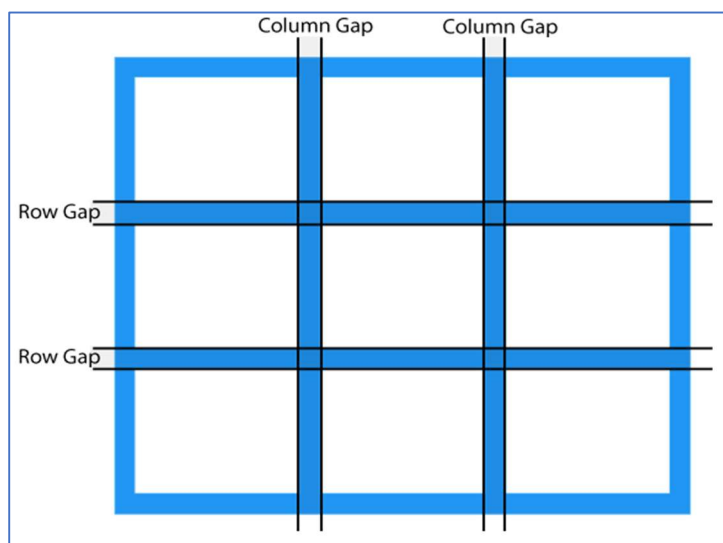
GRID ROWS

As fileiras horizontais formadas por grid items são chamadas de rows (linhas).






GRID GAP

Os espaços entre cada column/row são chamados de gaps (vão).

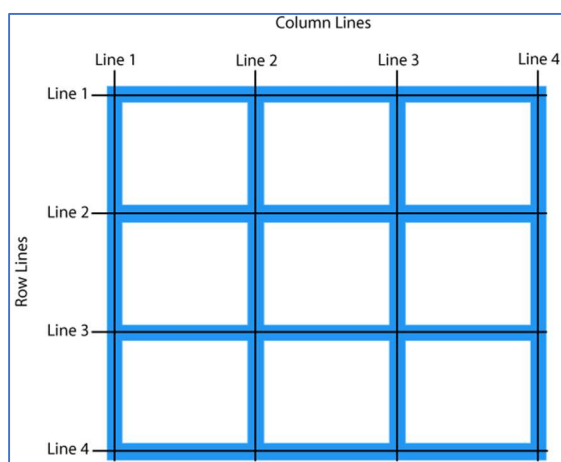


É possível ajustar o tamanho do gap utilizando as seguintes propriedades: **column-gap**, **row-gap** e **gap**.

<pre>.grid-container { display: grid; column-gap: 50px; }</pre> 	<pre>.grid-container { display: grid; row-gap: 50px; }</pre> 	<pre>.grid-container { display: grid; gap: 50px 100px; }</pre> 
---	--	--

GRID LINES

As linhas entre as columns são chamadas de **column-lines**. As linhas entre as rows são chamadas de **row-lines**.



Cada linha possui um número correspondente que será utilizado como referência para posicionar os grid items em um grid container.

No seguinte exemplo o grid item “item1” vai ser posicionado de uma maneira que ele comece na column line 1 e termine na column line 3.

<code>.item1 {</code>	1	2
<code>grid-column-start: 1;</code>	3	4
<code>grid-column-end: 3;</code>	6	7
<code>}</code>	8	

No seguinte exemplo o grid item “item1” vai ser posicionado de uma maneira que ele comece na row line 1 e termine na row line 3.

<code>.item1 {</code>	1	2	3
<code>grid-row-start: 1;</code>		4	5
<code>grid-row-end: 3;</code>	6	7	8
<code>}</code>			

grid-template-columns/rows

A propriedade **grid-template-columns** define o número de colunas no grid layout e permite definir o tamanho de cada coluna. No seguinte exemplo foram definidas 4 colunas no grid layout, os tamanhos (largura) das colunas foram definidos como auto, ou seja, os tamanhos vão ser iguais de acordo com o dimensionamento da janela.

<code>.grid-container {</code>	1	2	3	4
<code>display: grid;</code>	5	6	7	8
<code>grid-template-columns: auto auto auto auto;</code>				
<code>}</code>				

No seguinte exemplo foram definidas 4 colunas, porém, as larguras de algumas colunas foram especificadas, ou seja, o tamanho dessas colunas será mantido independente da alteração do tamanho da janela.

<code>.grid-container {</code>	1	2	3	4
<code>display: grid;</code>	5	6	7	8
<code>grid-template-columns: 80px 200px auto 40px;</code>				
<code>}</code>				

O mesmo se aplica a propriedade **grid-template-rows**, que define o número de rows no grid layout e permite definir o tamanho de cada row. No seguinte exemplo foi definido um grid layout com duas linhas com tamanhos(altura) específicos.

<code>.grid-container {</code>	1	2	3
<code>display: grid;</code>	4	5	6
<code>grid-template-rows: 80px 200px;</code>			
<code>}</code>			

Um modo para definir a quantidade de rows e columns é através da propriedade **grid-template** utilizando o sinal gráfico “/” para demonstrar a separação dos parâmetros de definição . A definição de grid row vem deve vir antes da definição de grid column. No seguinte exemplo foram definidas duas linhas uma com uma altura fixa de 150px e outra com um tamanho automático, foram definidas ainda na mesma propriedade três colunas, as unidades de medidas utilizadas nos tamanhos das colunas permitem um dimensionamento responsivo de acordo com o tamanho da página, duas colunas possuem um tamanho correspondente a uma fração, e uma coluna possui um tamanho correspondente a duas frações, ou seja, um tamanho duas vezes maior que as outras duas.

<code>.grid-container {</code>	1	2	3
<code>display: grid;</code>	4	5	6
<code>grid-template: 150px auto / 1fr 2fr 1fr;</code>			
<code>}</code>			

ALINHAMENTO CSS GRID

place-content e place-items

A propriedade **place-content** define as propriedades **align-items** e **justify-items** em uma única declaração.

A propriedade **place-items** define as propriedades **align-items** e **justify-items** em uma única declaração.

<code>.grid-container {</code>	1	2	3
<code>display: grid;</code>	4	5	6
<code>place-content: center;</code>			
<code>}</code>			

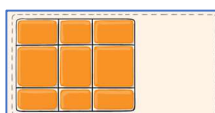
<code>.grid-container {</code>	1	2	3
<code>display: grid;</code>	4	5	6
<code>place-items: start end;</code>			
<code>}</code>			

justify-content

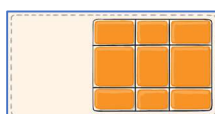
Caso o tamanho total da grid for menor que o container, essa propriedade **alinha a grid** dentro de um container (basicamente ajuste do padding). Alinha a grid referente ao eixo inline (**row ou eixo horizontal**).

Valores:

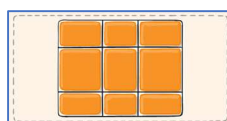
start – alinha a grid para nivelar com o começo do grid container;



end – alinha a grid para nivelar com o final do grid container;



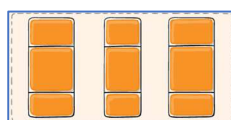
center – alinha a grid no centro do grid container;



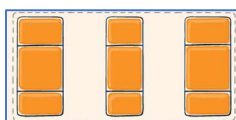
stretch – redimensiona os grid items para permitir que a grid preencha toda a largura do grid container;



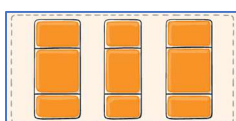
space-around – posiciona igualmente espaços entre cada grid item (na horizontal) com metade desses espaços nas extremidades;



space-between – posiciona igualmente espaços entre cada grid item (na horizontal) sem espaços nas extremidades;



space-evenly – posiciona igualmente espaços entre cada grid item (na horizontal) e nas extremidades;

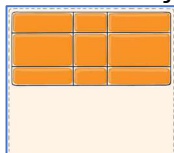


align-content

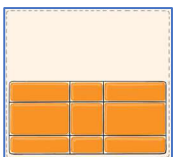
Caso o tamanho total da grid for menor que o container essa propriedade **alinha a grid** dentro de um container (basicamente ajuste do padding). Alinha a grid referente ao eixo block (**column ou eixo vertical**).

Valores:

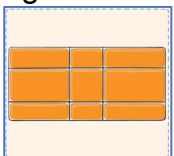
start – alinha a grid para nivelar com o começo do grid container;



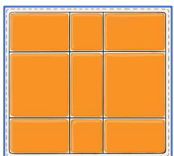
end – alinha a grid para nivelar com o final do grid container;



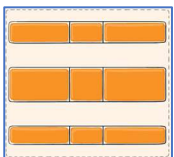
center – alinha a grid no centro do grid container;



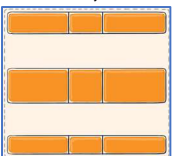
stretch – redimensiona os grid items para permitir que a grid preencha toda a largura do grid container;



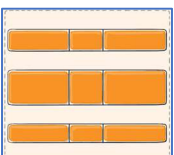
space-around – posiciona igualmente espaços entre cada grid item (na vertical) com metade desses espaços nas extremidades;



space-between – posiciona igualmente espaços entre cada grid item (na vertical) sem espaços nas extremidades;



space-evenly – posiciona igualmente espaços entre cada grid item (na vertical) e nas extremidades;

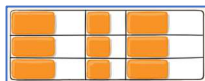


justify-items

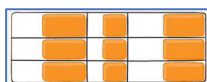
Alinha **todos os itens** dentro de uma célula. Alinha os grid items referente ao eixo inline (**row ou eixo horizontal**).

Valores:

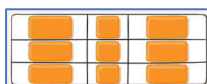
start – alinha os itens para nivelar com o começo da célula;



end – alinha os itens para nivelar com o final da célula;



center – alinha a grid no centro da célula;



stretch –(default) redimensiona os grid items para preencher todo espaço da célula;

align-items

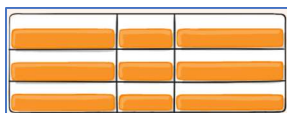
Alinha **todos os itens** dentro de uma célula (basicamente ajuste do margin). Alinha os grid items referente ao eixo block (**column ou eixo vertical**).

Valores:

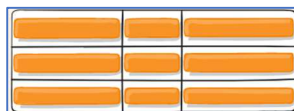
start – alinha os itens para nivelar com o começo da célula;



end – alinha os itens para nivelar com o final da célula;



center – alinha a grid no centro da célula;



stretch – (default) redimensiona os grid items para preencher todo espaço da célula;

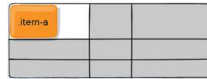
baseline – alinhar itens ao longo da linha de base do texto.

justify-self

Alinha **um item** dentro de uma célula (basicamente ajuste do margin).
Alinha o grid item referente ao eixo inline (**row ou eixo horizontal**).

Valores:

start – alinha o grid item para nivelar com o começo da célula;



end – alinha o grid item para nivelar com o final da célula;



center – alinha o grid item no centro da célula;



stretch – (default) redimensiona o grid item para preencher todo espaço da célula;



align-self

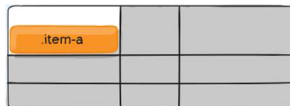
Alinha **um item** dentro de uma célula (basicamente ajuste do margin).
Alinha o grid item referente ao eixo block (**column ou eixo vertical**).

Valores:

start – alinha o grid item para nivelar com o começo da célula;



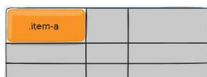
end – alinha o grid item para nivelar com o final da célula;



center – alinha o grid item no centro da célula;



stretch – (default) redimensiona o grid item para preencher todo espaço da célula;



POSICIONAMENTO CSS GRID

grid-column

Os grid-items podem ser posicionados verticalmente através de duas propriedades, **grid-column-start** e **grid-column-end**. A propriedade **grid-column** é uma maneira encurtada de definir essas propriedades. Primeiro é definido a column-line onde o item começará e após o uso do sinal de “/” é definido a column-line onde o item irá terminar. O posicionamento do grid-item é feito utilizando o número correspondente a column-line desejada, ou utilizando “span” que define quantas columns o item vai expandir. No seguinte exemplo o “item1” começa na column-line 1 e vai até a column-line 5.

<code>.item1 {</code>	1	2	3
<code> grid-column: 1 / 5;</code>	4	5	6
<code>}</code>	7	8	9
	10	11	12
	13	14	15

No seguinte exemplo o “item1” começa na column 1 e expande 3 columns.

<code>.item1 {</code>	1	2	3	4
<code> grid-column: 1 / span 3;</code>	5	6	7	8
<code>}</code>	9	10	11	12
	13	14	15	16

grid-row

Os grid-items são posicionados horizontalmente através de duas propriedades, **grid-row-start** e **grid-row-end**. A propriedade **grid-row** é uma maneira encurtada de definir essas propriedades. Primeiro é definido a row-line onde o item começará e após o uso do sinal de “/” é definido a row-line onde o item irá terminar. O posicionamento do grid-item é feito utilizando o número correspondente a row-line desejada, ou utilizando “span” que define quantas rows o item vai expandir. No seguinte exemplo o “item1” começa na row-line 1 e vai até a row-line 4.

<code>.item1 {</code>	1	2	3	4	5	6
<code> grid-row: 1 / 4;</code>	7	8	9	10	11	
<code>}</code>	12	13	14	15	16	

No seguinte exemplo o “item1” começa na row-line 1 e expande 2 rows.

<code>.item1 {</code>	1	2	3	4	5	6
<code> grid-row: 1 / span 2;</code>		7	8	9	10	11
<code>}</code>	12	13	14	15	16	17

grid-area

A propriedade **grid-area** é uma maneira encurtada de definir de as propriedades **grid-row-start**, **grid-column-start**, **grid-row-end** e **grid-column-end**. Primeiro é definido a row-line e column-line onde o item começará e depois é definido a row-line e column-line onde o item irá terminar. O posicionamento do grid-item é feito utilizando o número correspondente a line desejada, ou utilizando “span” que define quantas columns ou rows o item vai expandir. No seguinte exemplo o “item8” começa na row-line 1 e column-line 2, e então vai até a row-line 5 e column-line 6.

<code>.item8 {</code>	1	2	3	4	5	6
<code> grid-area: 1 / 2 / 5 / 6;</code>	3	8				4
<code>}</code>	5					6
	7					9
	10	11	12	13	14	15

No seguinte exemplo o “item8” começa na row-line 2 e column-line 1, e expande 2 rows e 3 columns.

<code>.item8 {</code>	1	2	3	4	5	6
<code> grid-area: 2 / 1 / span 2 / span 3;</code>	8			7	9	10
<code>}</code>				11	12	13

grid-template-areas

A propriedade **grid-area** também pode ser usada para atribuir nomes aos grid-items, os itens nomeados podem então ser referenciados através da propriedade **grid-template-areas** no grid container. Cada row é definida por apóstrofos (' '). As columns em cada row são definidas dentro dos apóstrofos, separadas por um espaço.

<code>.item1 { grid-area: header; }</code>	Header			
<code>.item2 { grid-area: menu; }</code>	Menu	Main		Right
<code>.item3 { grid-area: main; }</code>				
<code>.item4 { grid-area: right; }</code>				
<code>.item5 { grid-area: footer; }</code>	Footer			

```

.grid-container {
  grid-template-areas:
    'header header header header header'
    'menu main main main right right'
    'menu footer footer footer footer footer';
}

```

Um sinal de ponto “.” representa um grid item sem nome

<code>.item1 {</code>	1	2	3	4
<code> grid-area: myArea;</code>				
<code>}</code>	5	6	7	

```

.grid-container {
  display: grid;
  grid-template-areas: 'myArea myArea . . .'
                      'myArea myArea . . .';
}

```

TIPOS DE ALINHAMENTOS CSS

As propriedades de alinhamento de caixa no CSS são definidas como 6 propriedades que controlam o alinhamento de caixas dentro de outras caixas (containers). Elas podem ser classificadas como:

- Em qual dimensão/eixo a propriedade se refere (main/inline ou cross/block).
- O controle do posicionamento está relacionado a caixa dentro do parent, ou relacionado ao conteúdo dentro da própria caixa.

A seguinte imagem explica isso.

Common	Axis	Aligns	Applies to
‘justify-content’	main/inline	content within element (effectively adjusts padding)	block containers , flex containers , and grid containers
‘align-content’	cross/block		
‘justify-self’	inline	element within parent (effectively adjusts margins)	block-level boxes, absolutely-positioned boxes, and grid items
‘align-self’	cross/block		absolutely-positioned boxes, flex items , and grid items
‘justify-items’	inline	items inside box (controls child items’ ‘justify-self: auto’)	block containers and grid containers
‘align-items’	cross/block		flex containers and grid containers