

Desenvolvimento Web

CSS Flexbox



Índice

Introdução	3
Conceitos	4
Direção do eixo	5
Recipiente flexbox multi-linha	5
Atalho: Direção do eixo	7
Propriedades de alinhamento	7
Sobre o eixo primário	7
No eixo secundário	10
Atalho: Alinhamentos	14
Propriedades do filso	14
Atalho: Propriedades do filso	15
Lacunas (gaps)	15
Atalho: Gaps	15
Ordem dos ítems	16

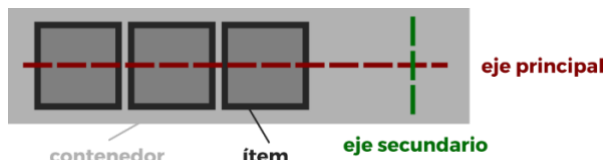
Introdução

Tradicionalmente, o CSS tem usado posicionamento ([static](#), [relative](#), [absolute](#)...), elementos em linha ou em bloco (e derivados) ou elementos [float](#), que em termos gerais ainda era um sistema de criação de design bastante tradicional que não se ajustava aos desafios que enfrentamos hoje: sistemas de desktop, dispositivos móveis, múltiplas resoluções, etc...

[Flexbox](#) é um sistema de elementos flexíveis que vem com a idéia de esquecer estes mecanismos e se acostumar a uma mecânica mais poderosa, limpa e personalizável, na qual os elementos HTML são automaticamente adaptados e colocados e é mais fácil personalizar os desenhos. Ela foi especialmente projetada para criar, por meio do CSS, estruturas unidimensionais.

Conceitos

Para começar a utilizar o **flexbox**, a primeira coisa a fazer é conhecer alguns dos elementos básicos deste novo esquema, que são os seguintes:



- **Container:** Este é o elemento principal que cada um dos itens flexíveis terá dentro dele. Note que ao contrário de muitas outras estruturas CSS, como regra geral, no Flex nós definimos as propriedades para o elemento pai.
- **Eixo principal:** As Flexboxes devem ter uma orientação principal específica. Por padrão, o eixo principal do recipiente flexbox é horizontal (em uma fileira).
- **Eixo secundário:** Da mesma forma, os recipientes flexíveis devem ter uma orientação secundária, perpendicular à principal. Se a orientação primária for horizontal, a orientação secundária será vertical (e vice-versa).
- **Item:** Cada uma das crianças que o recipiente terá em seu interior.

Uma vez esclarecido este ponto, imaginemos o seguinte cenário:

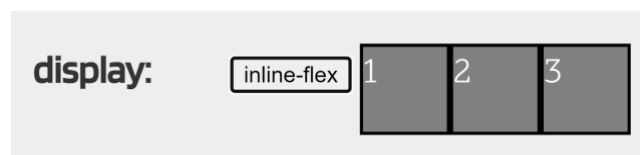
```
<div class="container"> <!-- Flex
container -->
  <div class="item item-1">1</div> <!--
Flex items -->
  <div class="item item-2">2</div>
  <div class="item item-3">3</div>
</div>
```

Para ativar o modo **flexbox**, utilizamos a propriedade de **display** no elemento do recipiente e especificamos

o valor **flex** ou **inline-flex** (dependendo de como queremos que o recipiente se comporte):

Tipo de elemento	Descrição
inline-flex	Configura um recipiente em linha, semelhante a inline-block (ocupa apenas o conteúdo).
flex	Configura um recipiente em bloco, semelhante a block (ocupa toda a largura do pai).

Por padrão, e somente com isto, observaremos que os elementos estão todos dispostos na mesma linha. Isto porque estamos usando o modo **flexbox** e estaremos trabalhando com itens flex básicos, garantindo que eles não transbordarão ou mostrarão os problemas que, por exemplo, as porcentagens têm em elementos que não usam a **flexbox**.



Direção do eixo

Há duas propriedades principais para manipular a direção e o comportamento dos itens ao longo do eixo principal do recipiente. Eles são os seguintes:

Propriedade	Valor	Significado
flex-direction	row row-reverse column column-reverse	Muda a orientação do eixo principal.

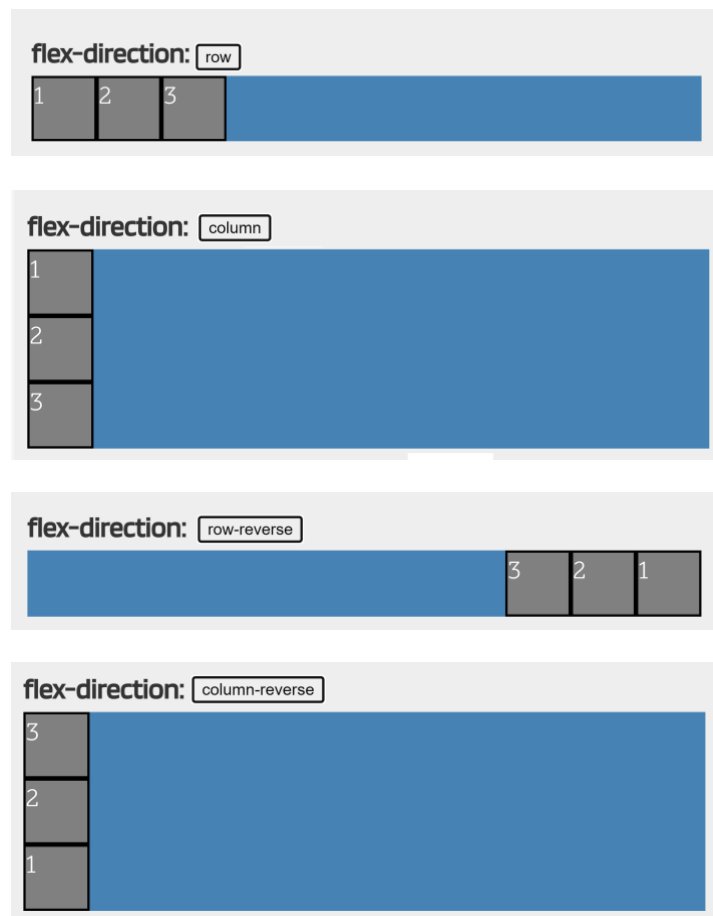
Usando a propriedade de **flex-direction**, podemos mudar a direção do eixo principal do recipiente para ser orientado horizontalmente (padrão) ou verticalmente. Além disso, também podemos incluir o sufixo -reverse para indicar a colocação dos itens em ordem inversa.

Valor	Descrição
row	Define a direção do eixo principal para horizontal.
row-reverse	Define a direção do eixo principal para horizontal (invertido).
column	Define a direção do eixo principal para vertical.
column-reverse	Define a direção do eixo principal para vertical (invertido).

Isto nos permite ter um nível muito alto de controle sobre a ordem dos elementos em uma página. Vejamos a aplicação destas propriedades no exemplo anterior, para modificar o fluxo do eixo principal do recipiente:

```
.container {  
  display: flex;  
  flex-direction: column;  
  background: steelblue;  
}  
  
.item {  
  background: grey;  
}
```

Aqui estão alguns exemplos:



Recipiente flexbox multi-linha

Por outro lado, há outra propriedade chamada **flex-wrap** com a qual podemos especificar o comportamento do recipiente no que diz respeito a evitar que ele transborde (**nowrap**, o valor padrão) ou permitir que ele o faça; nesse caso, estaríamos falando de um recipiente flexbox com várias linhas.

Propriedade	Valor	Significado
flex-wrap	nowrap wrap wrap-reverse	Impede ou permite o transbordo (multilinha).

Os valores que esta propriedade pode tomar são os seguintes:

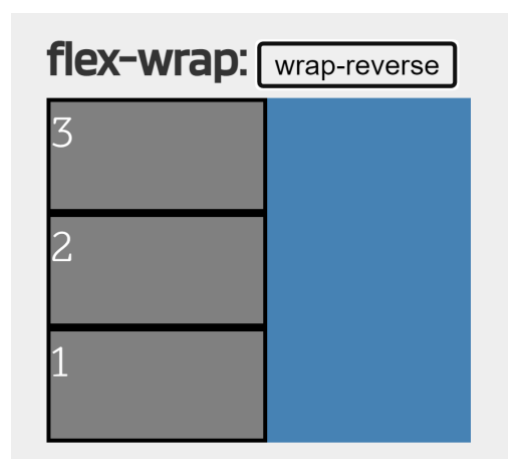
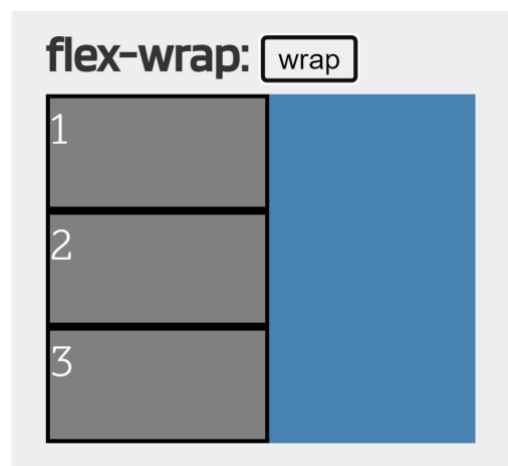
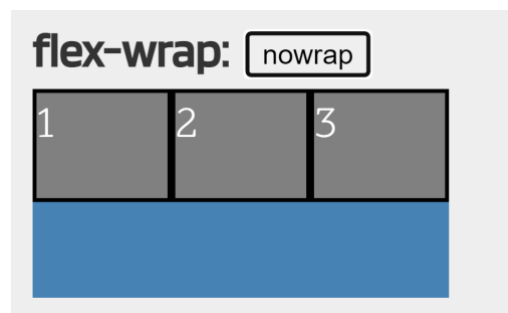
Valor	Descrição
nowrap	Define os itens em uma única linha (não permite que o contêiner transborde).
wrap	Define os itens no modo multilinha (permite que o contêiner transborde).
wrap-reverse	Define os itens no modo multilinha, mas na direção inversa.

Levando em conta estes valores da propriedade **flex-wrap**, podemos alcançar coisas como:

```
.container {  
  display: flex;  
  flex-wrap: wrap; /* Comportamiento  
por defecto: nowrap */  
  background: steelblue;  
  width: 200px;  
}  
  
.item {  
  background: grey;  
  width: 50%;  
}
```

No caso de especificar **nowrap** (ou omitir a propriedade **flex-wrap**) no contêiner, todos os 3 itens seriam exibidos na mesma linha do contêiner. Nesse caso, cada item teria que ter 50% de largura (ou seja, 100px do recipiente de 200px). Um tamanho de 100px por item somaria um total de 300px, o que não caberia no contêiner de 200px, então o **flexbox** reajusta os itens flexíveis para caberem todos na mesma linha, mantendo as mesmas proporções.

No entanto, se especificarmos o **wrap** na propriedade de **flex-wrap**, o que permitimos é que o container possa transbordar, tornando-se um container de várias linhas, que mostraria os itens 1 e 2 na primeira linha (com um tamanho de 100px cada) e o item 3 na linha seguinte, deixando um espaço livre para um possível item 4.



Atalho: Direção do eixo

Lembre-se que existe uma propriedade de atalho (short-hand) chamada **flex-flow**, com a qual podemos resumir os valores das propriedades de **flex-direction** e **flex-wrap**, especificando-os em uma única propriedade e poupando-nos de usar as propriedades específicas:

```
.container {
  /* flex-flow: <flex-direction>
  <flex-wrap>; */
  flex-flow: row wrap;
}
```

Propriedades de alinhamento

Agora que temos um controle básico do recipiente desses itens flexíveis, precisamos conhecer as propriedades existentes dentro da **flexbox** para organizar os itens dependendo de nosso objetivo. Vejamos 4 propriedades interessantes para isto, a primeira delas atua no eixo principal, enquanto o resto no eixo secundário:

Propiedad	Valor	Eixo
justify-content	flex-start flex-end center space-between space-around space-evenly	1
align-content	flex-start flex-end center space-between space-around space-evenly stretch	2
align-items	flex-start flex-end center stretch baseline	2
align-self	auto flex-start flex-end center stretch baseline	2

Desta pequena lista, a primeira e terceira propriedades são as mais importantes (as outras

duas são casos particulares, que explicaremos mais adiante):

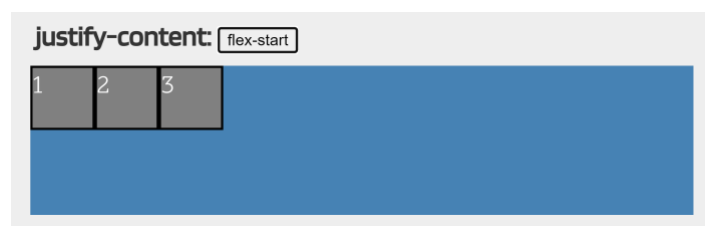
- **justify-content**: Utilizado para alinhar os itens no eixo principal (por padrão, o eixo horizontal).
- **align-items**: Usado para alinhar os itens no eixo secundário (por padrão, vertical).

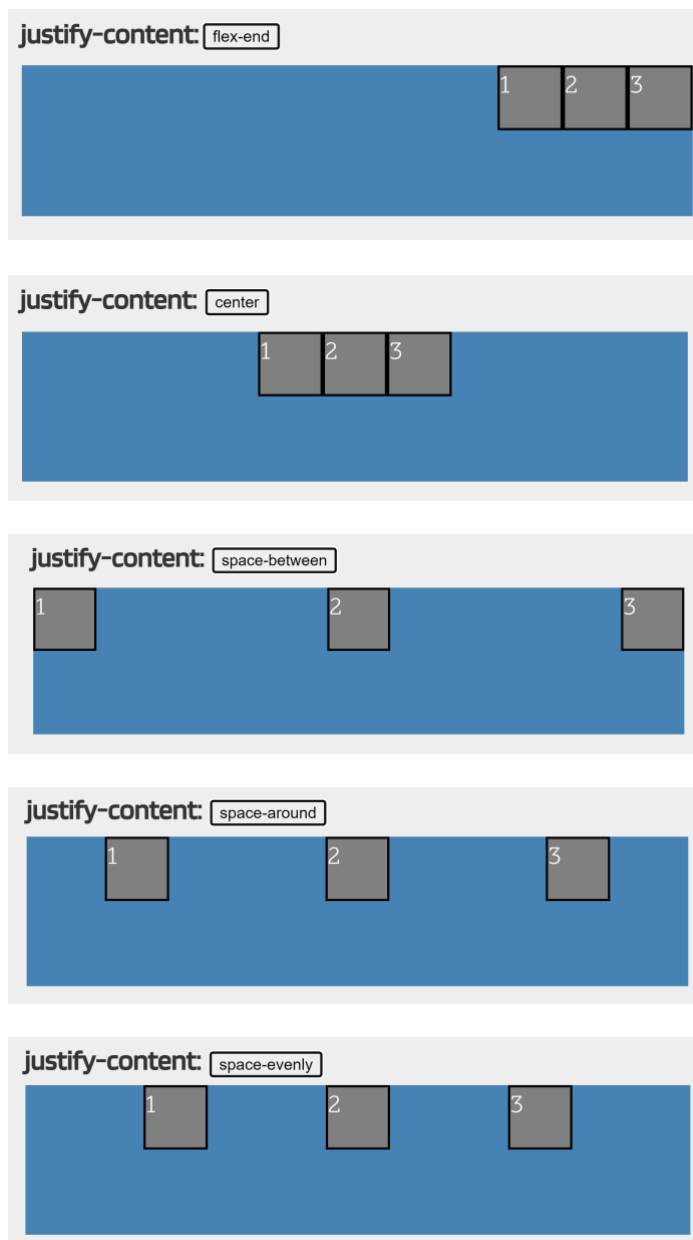
Sobre o eixo primário

A primeira propriedade, **justify-content**, é utilizada para colocar os itens em um recipiente em um determinado arranjo ao longo do eixo principal:

Valor	Descrição
flex-start	Items do grupo no início do eixo principal.
flex-end	Agrupe os itens no final do eixo principal.
center	Items do grupo no centro do eixo principal.
space-between	Distribua os itens deixando o máximo de espaço possível para separá-los.
space-around	Distribuir os itens deixando o mesmo espaço ao seu redor (esquerda/direita).
space-evenly	Distribua os itens deixando o mesmo espaço (sobrepostos) para a esquerda e para a direita.

Com cada um desses valores, modificaremos a disposição dos itens no recipiente onde é aplicado, e eles serão colocados como mostrado no seguinte exemplo interativo (observe os números para observar a ordem de cada item):





Uma vez entendido este caso, devemos prestar atenção à propriedade de `align-content`, que é um caso particular do caso anterior. Será útil quando estivermos lidando com um contêiner flex multilinha, que é um contêiner em que os itens não cabem na largura disponível e, portanto, o eixo principal é dividido em várias linhas (por exemplo, usando `flex-wrap: wrap`).

Assim, o `align-content` será usado para alinhar cada uma das linhas do contêiner multi-linhas.

Os valores que pode assumir são os seguintes:

Valor	Descrição
flex-start	Items do grupo no início do eixo principal.
flex-end	Agrupe os items no final do eixo principal.
center	Items do grupo no centro do eixo principal.
space-between	Distribua os items do início ao fim.
space-around	Distribua os items deixando o mesmo espaço em ambos os lados de cada item.
stretch	Esticar os items para ocupar todo o espaço de maneira uniforme.

Com estes valores, vemos como mudamos o layout vertical (porque partimos de um exemplo em que estamos usando a `flex-direction: row`, e o eixo principal é horizontal) dos items que estão dentro de um contêiner com várias linhas.

No exemplo a seguir, veremos que, indicando um recipiente de 200 pixels de altura com itens de 50px de altura e um `flex-wrap` para ter recipientes multilinha, podemos usar a propriedade de `align-content` para alinhar verticalmente os itens de modo que eles permaneçam na área inferior do recipiente:

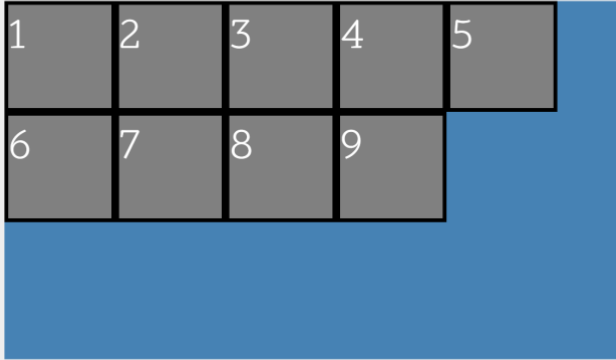
```
.container {
  background: #CCC;
  display: flex;
  width: 200px;
  height: 200px;

  flex-wrap: wrap;
  align-content: flex-end;
}

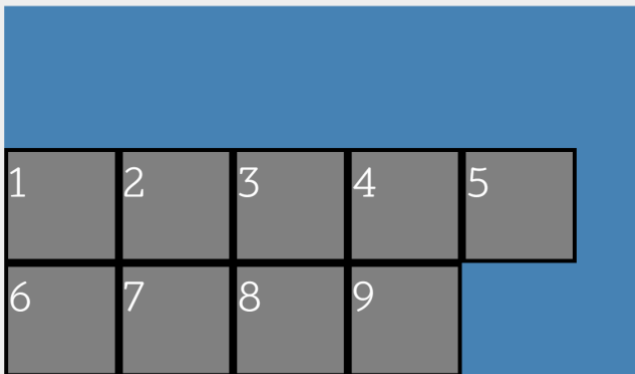
.item {
  background: #777;
  width: 50%;
  height: 50px;
}
```


Veja como funciona a propriedade `align-content` no seguinte exemplo interativo:

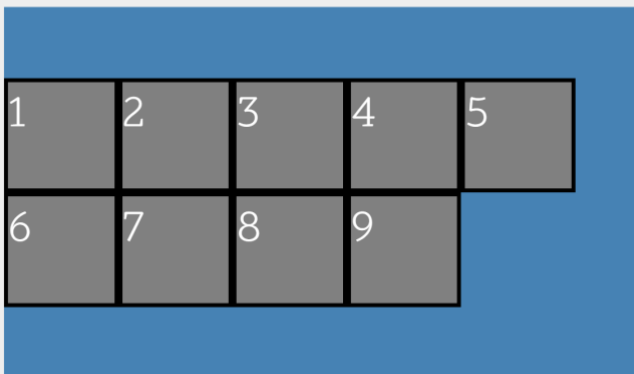
align-content: `flex-start`



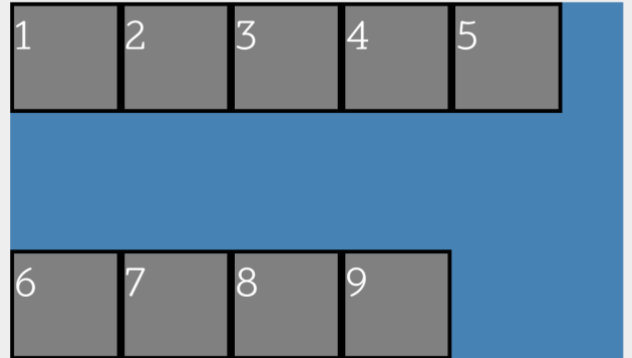
align-content: `flex-end`



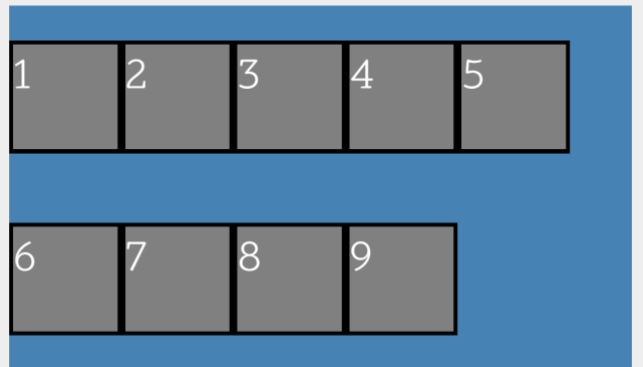
align-content: `center`



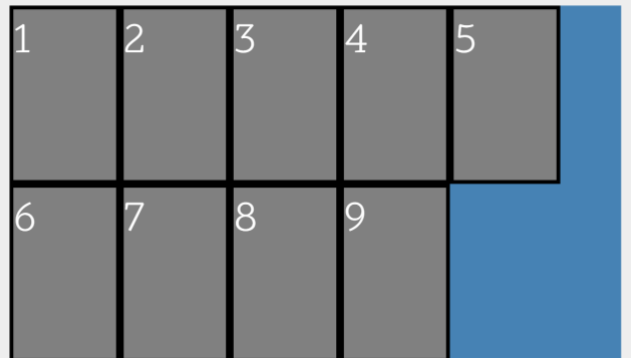
align-content: `space-between`



align-content: `space-around`



align-content: `stretch`

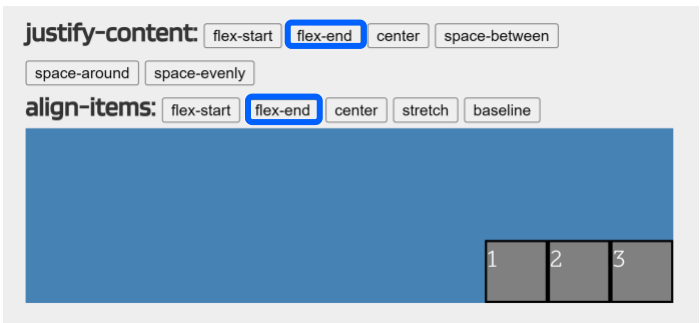
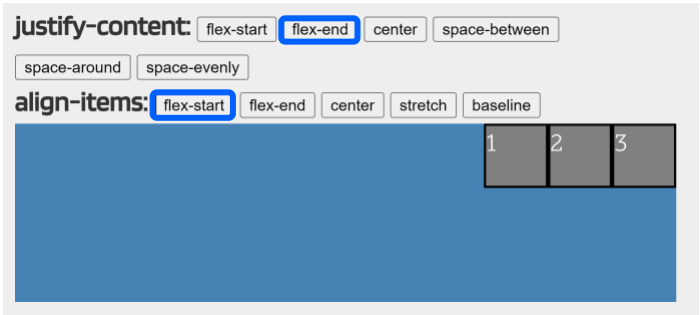
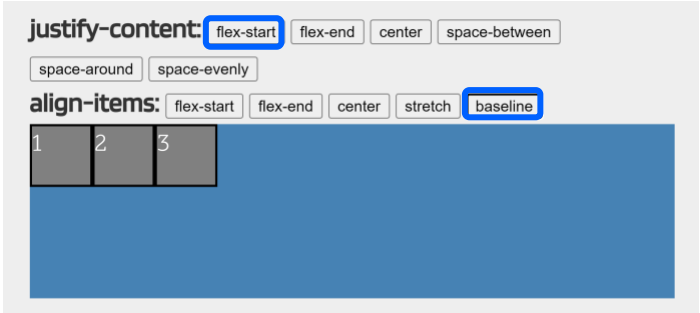
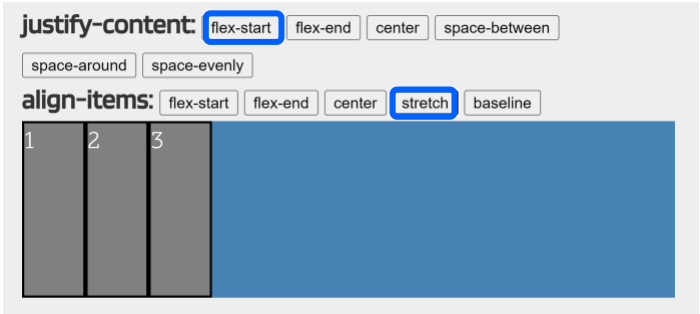
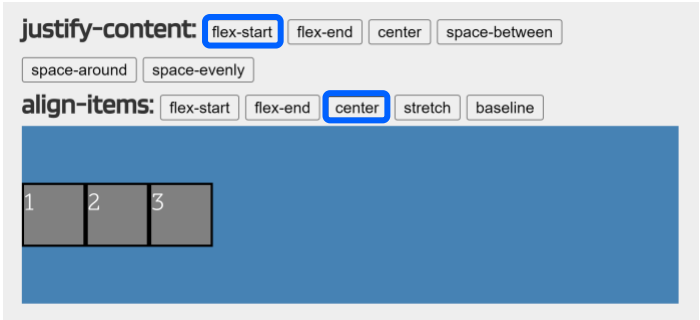
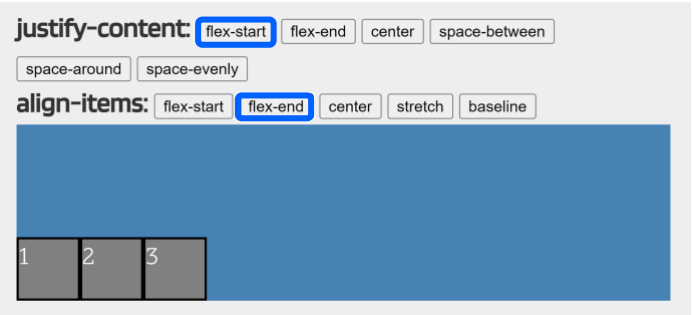
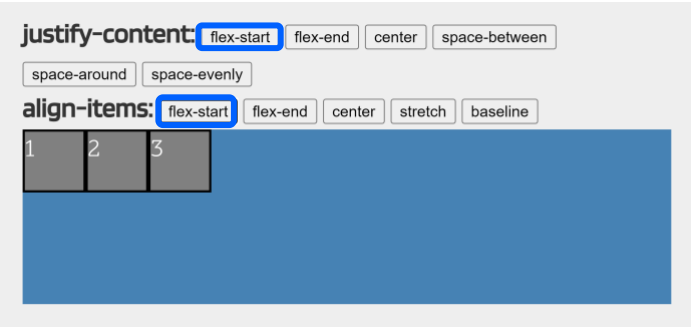


No eixo secundário

A outra propriedade importante nesta seção é o `align-items`, que é responsável pelo alinhamento dos itens no eixo secundário do recipiente. Tome cuidado para não confundir `align-content` com `align-items`, já que o conteúdo de alinhamento atua em cada linha de um recipiente de várias linhas (não tem efeito em recipientes de uma linha), enquanto os `align-items` atuam na linha atual. Os valores que pode assumir são os seguintes:

Valor	Descrição
<code>flex-start</code>	Alinhar os itens no início do eixo secundário.
<code>flex-end</code>	Alinhe os itens no final do eixo secundário.
<code>center</code>	Alinea los ítems al centro del eje secundario.
<code>stretch</code>	Alinhe os itens esticando-os de modo que cubram desde o início até o final do contêiner.
<code>baseline</code>	Alinha os itens no contêiner com base no conteúdo dos itens no contêiner.


Vamos olhar para exemplos com `justify-content` e `align-items`:



justify-content: flex-start **flex-end** center space-between

space-around space-evenly

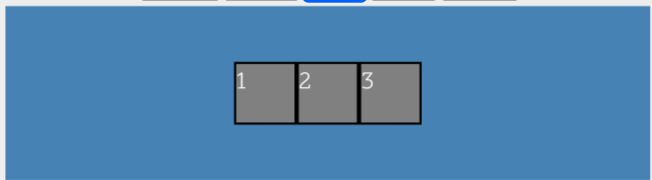
align-items: flex-start flex-end **center** stretch baseline



justify-content: flex-start flex-end **center** space-between

space-around space-evenly


align-items: flex-start flex-end **center** stretch baseline



justify-content: flex-start **flex-end** center space-between

space-around space-evenly

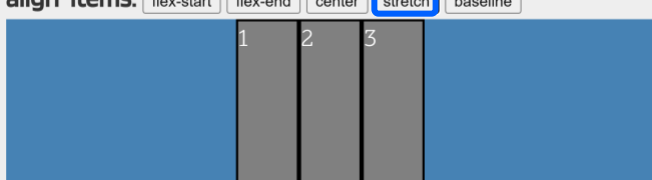
align-items: flex-start flex-end center **stretch** baseline



justify-content: flex-start flex-end **center** space-between

space-around space-evenly


align-items: flex-start flex-end center **stretch** baseline



justify-content: flex-start **flex-end** center space-between

space-around space-evenly

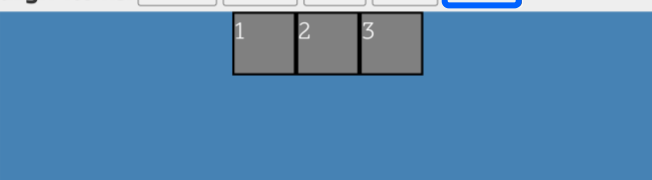
align-items: flex-start flex-end center stretch **baseline**



justify-content: flex-start flex-end **center** space-between

space-around space-evenly

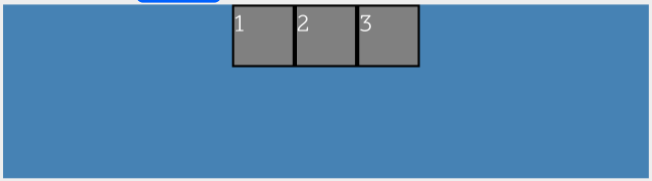
align-items: flex-start flex-end center stretch **baseline**



justify-content: flex-start flex-end **center** space-between

space-around space-evenly

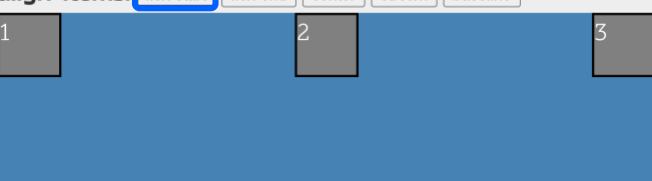
align-items: **flex-start** flex-end center stretch baseline



justify-content: flex-start flex-end center **space-between**

space-around space-evenly

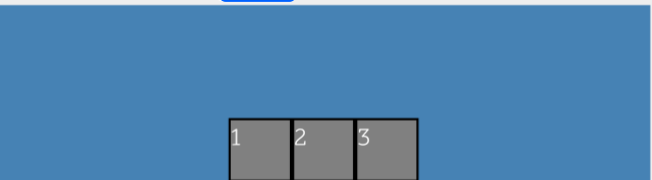
align-items: **flex-start** flex-end center stretch baseline



justify-content: flex-start flex-end **center** space-between

space-around space-evenly

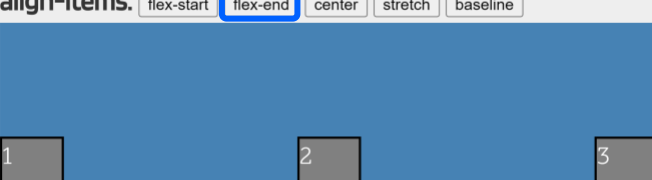
align-items: flex-start **flex-end** center stretch baseline



justify-content: flex-start flex-end center **space-between**

space-around space-evenly

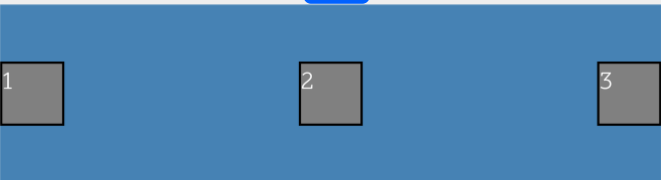
align-items: flex-start **flex-end** center stretch baseline



justify-content: flex-start flex-end center **space-between**

space-around space-evenly

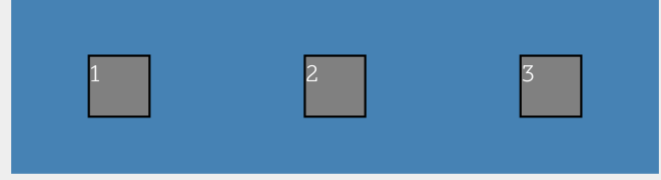
align-items: flex-start flex-end **center** stretch baseline

A blue rectangular container holds three gray square items labeled 1, 2, and 3. The items are distributed evenly across the width of the container, with the largest gaps at the far left and far right. All three items are vertically centered within the container.

justify-content: **space-around** flex-end center space-between

space-evenly

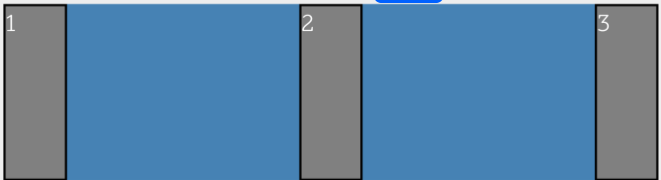
align-items: flex-start flex-end **center** stretch baseline

A blue rectangular container holds three gray square items labeled 1, 2, and 3. The items are distributed evenly across the width of the container, with the largest gaps at the far left and far right. All three items are vertically centered within the container.

justify-content: flex-start flex-end center **space-between**

space-around space-evenly

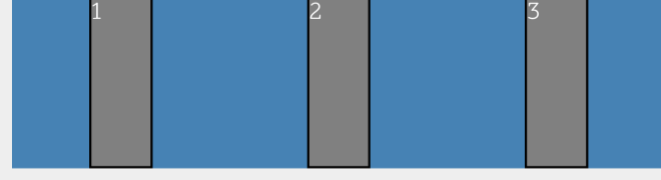
align-items: flex-start flex-end center **stretch** baseline

A blue rectangular container holds three gray square items labeled 1, 2, and 3. The items are distributed evenly across the width of the container, with the largest gaps at the far left and far right. The items are stretched vertically to fill the height of the container.

justify-content: **space-around** flex-end center space-between

space-evenly

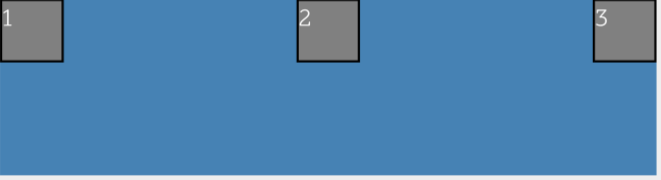
align-items: flex-start flex-end center **stretch** baseline

A blue rectangular container holds three gray square items labeled 1, 2, and 3. The items are distributed evenly across the width of the container, with the largest gaps at the far left and far right. The items are stretched vertically to fill the height of the container.

justify-content: flex-start flex-end center **space-between**

space-around space-evenly

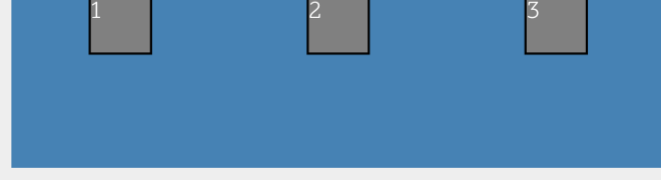
align-items: flex-start flex-end center stretch **baseline**

A blue rectangular container holds three gray square items labeled 1, 2, and 3. The items are distributed evenly across the width of the container, with the largest gaps at the far left and far right. The items are aligned to the baseline, which is the bottom of the container.

justify-content: flex-start flex-end center space-between

space-around space-evenly

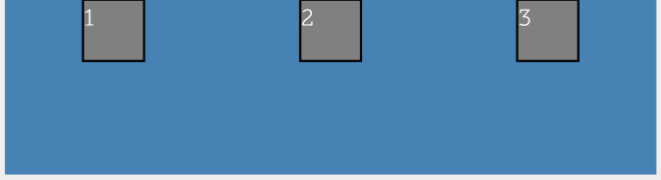
align-items: flex-start flex-end center stretch **baseline**

A blue rectangular container holds three gray square items labeled 1, 2, and 3. The items are distributed evenly across the width of the container, with the largest gaps at the far left and far right. The items are aligned to the baseline, which is the bottom of the container.

justify-content: flex-start flex-end center space-between

space-around space-evenly

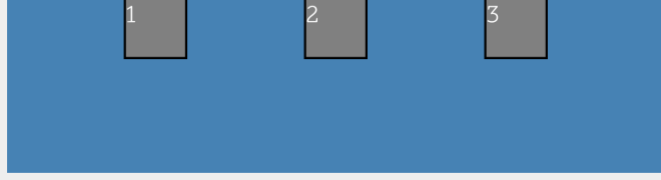
align-items: **flex-start** flex-end center stretch baseline

A blue rectangular container holds three gray square items labeled 1, 2, and 3. The items are distributed evenly across the width of the container, with the largest gaps at the far left and far right. The items are aligned to the top of the container.

justify-content: flex-start flex-end center space-between

space-around **space-evenly**

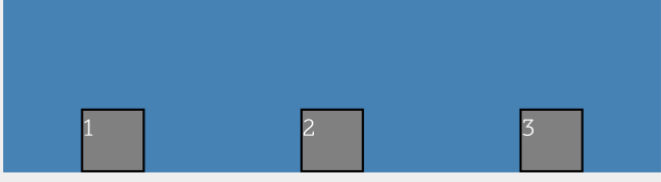
align-items: **flex-start** flex-end center stretch baseline

A blue rectangular container holds three gray square items labeled 1, 2, and 3. The items are distributed evenly across the width of the container, with the largest gaps at the far left and far right. The items are aligned to the top of the container.

justify-content: flex-start flex-end center space-between

space-around space-evenly

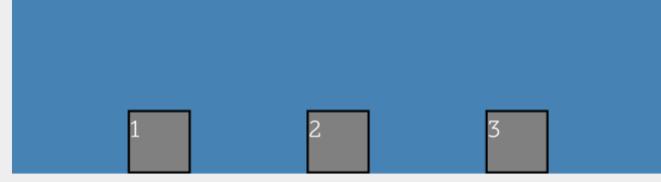
align-items: flex-start **flex-end** center stretch baseline

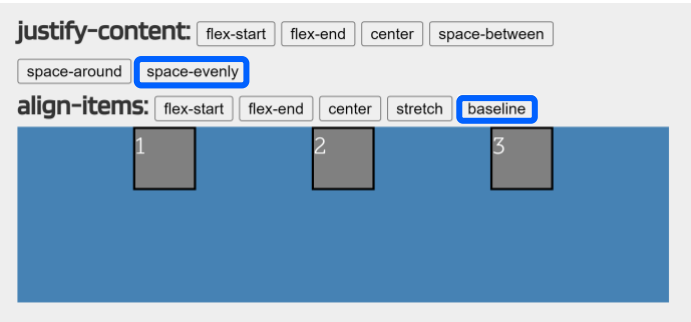
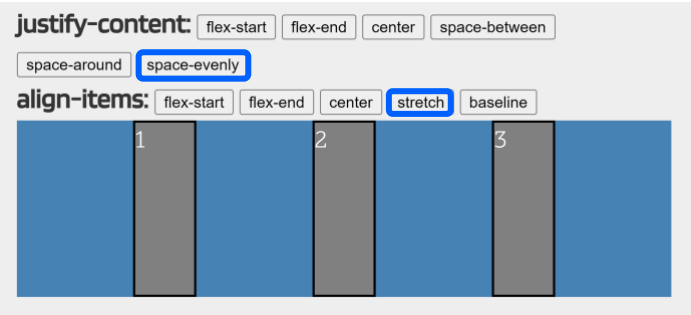
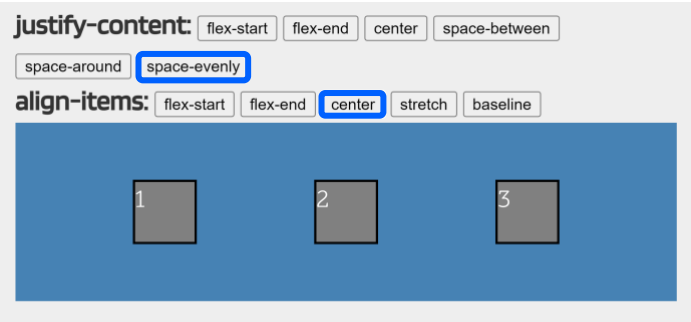
A blue rectangular container holds three gray square items labeled 1, 2, and 3. The items are distributed evenly across the width of the container, with the largest gaps at the far left and far right. The items are aligned to the bottom of the container.

justify-content: flex-start flex-end center space-between

space-around **space-evenly**

align-items: flex-start **flex-end** center stretch baseline

A blue rectangular container holds three gray square items labeled 1, 2, and 3. The items are distributed evenly across the width of the container, with the largest gaps at the far left and far right. The items are aligned to the bottom of the container.



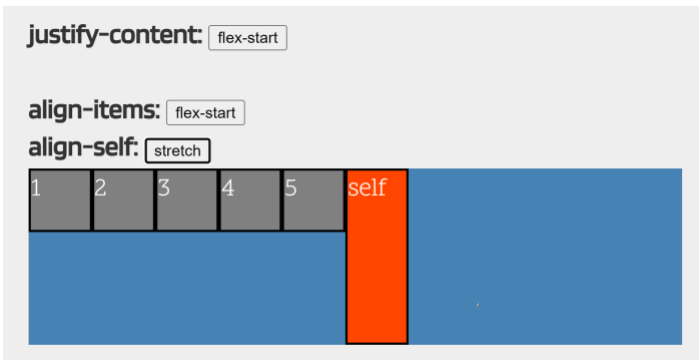
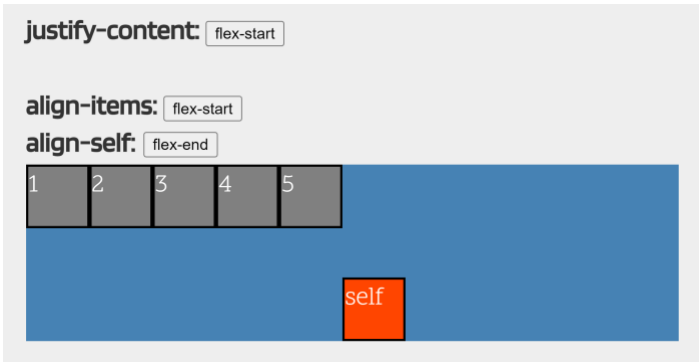
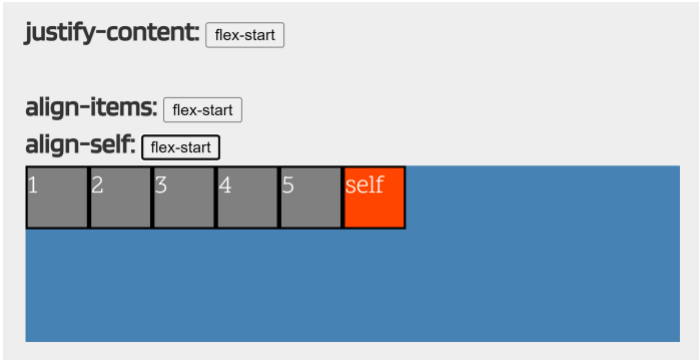
Por outro lado, a propriedade `align-self` age exatamente da mesma forma que as propriedades `align-items`, no entanto, é a primeira propriedade `flexbox` que vemos utilizada em um item específico de criação e não no elemento do contêiner. Exceto por este detalhe, funciona exatamente da mesma forma que os `align-items`.

Graças a esse detalhe, o `align-self` nos permite mudar o comportamento de `align-items` e anulá-lo com comportamentos específicos para itens específicos que não queremos ter o mesmo comportamento que os demais. O imóvel pode assumir os seguintes valores:

Valor	Descrição
<code>flex-start</code>	Alinha os itens no início do contêiner.
<code>flex-end</code>	Alinha os itens no final do contêiner.

<code>center</code>	Alinhe os itens ao centro do recipiente.
<code>stretch</code>	Alinhe os itens esticando-os ao tamanho do recipiente.
<code>baseline</code>	Alinha os itens no contêiner de acordo com a base dos itens.
<code>auto</code>	Herda o valor dos <code>align-items</code> do pai (se não estiver definido, é <code>stretch</code>).

Se o valor `auto` é especificado para a propriedade `align-self`, o navegador atribui o valor da propriedade `align-items` do recipiente pai, e se ele não existir, o valor padrão: `stretch`. Vejamos alguns exemplos para vê-lo em funcionamento:



Atalho: Alinhamentos

Existe uma propriedade de atalho com a qual os valores de `align-content` e `justify-content` podem ser definidos de uma só vez, chamada de `place-content`:

```
.container {  
  display: flex;  
  place-content: flex-start flex-end;  
end;  
  
/* Equivalente a... */  
align-content: flex-start;  
justify-content: flex-end;  
}
```

Propriedades do filso

Com exceção da propriedade `align-self`, todas as propriedades que vimos até agora se aplicam ao elemento recipiente. As seguintes propriedades, no entanto, aplicam-se aos itens filhos.

Propriedad	Valor	Descrição
flex-grow	0 number	Número que indica o fator de crescimento do item em relação ao restante.
flex-shrink	1 number	Número que indica o fator decrescente do item em relação ao restante.
flex-basis	Size content	Tamanho básico dos itens antes de aplicar a variação.
order	0 number	Número (peso) indicando a ordem de aparecimento dos itens.

Em primeiro lugar, temos a propriedade de `flex-grow` para indicar o fator de crescimento dos itens caso eles não tenham uma largura específica. Por exemplo, se com o `flex-grow` estabelecermos um valor de 1 para todos os seus itens, todos eles teriam o mesmo tamanho. Mas se fixarmos um valor de 1 para todos os elementos, exceto um deles, que fixamos em 2, esse item será maior do que os anteriores. Os itens para os quais nenhum valor é especificado terão um valor padrão de 0.

Em segundo lugar, temos a propriedade `flex-shrink`, que é o oposto de `flex-grow`. Enquanto o anterior indica um fator de crescimento, o `flex-shrink` faz exatamente o contrário, ele aplica um fator decrescente. Desta forma, os itens com valor numérico maior serão menores, enquanto aqueles com valor numérico menor serão maiores, exatamente o oposto de como funciona a propriedade `flex-grow`.

Finalmente, temos a propriedade `flex-basis`, que define o tamanho padrão (base) que os itens terão antes que a alocação de espaço seja aplicada. Geralmente, é aplicado um tamanho (unidades, porcentagens, etc.), mas você também pode aplicar a palavra-chave `content` que ajusta automaticamente o tamanho ao conteúdo do item, que é seu valor padrão.

Atalho: Propriedades do flex

Há uma propriedade chamada flex que serve como atalho para essas três propriedades de artigos infantis. Funciona da seguinte maneira:

```
.item {
  /* flex: <flex-grow> <flex-shrink> <flex-basis> */
  flex: 1 3 35%;
}
```

Lacunas (gaps)

Há duas propriedades do flexbox que surgiram recentemente: **row-gap** e **column-gap**. Estas propriedades permitem definir o tamanho de uma "lacuna" entre os itens do elemento pai que contém a criança, e sem a necessidade de usar **padding** ou **margin** sobre os elementos da criança.

Propiedad	Valor	Descrição
row-gap	normal size	Espaçamento entre fileiras (somente se flex-direction: column)
column-gap	normal size	Espaçamento entre colunas (somente se flex-direction: row)

Observe que apenas uma das duas propriedades terá efeito, dependendo se a propriedade de **flex-direction** estiver definida para **column** ou **row**. No entanto, é possível utilizar ambos se tivermos a propriedade de **flex-wrap** programado para **wrap**, portanto, tivermos **flexbox multicolumna**.

Atalho: Gaps

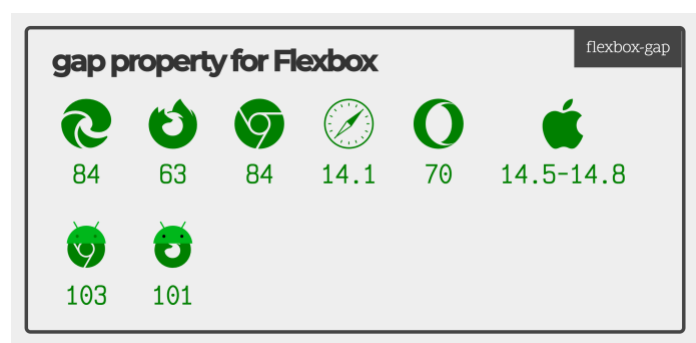
Caso queiramos usar uma propriedade de atalho para as lacunas, podemos usar a propriedade gap:

Propiedad	Valor	Descrição
gap	0 size	Aplique o tamanho indicado para a folga em ambos os eixos.
gap	0 0 size size	Aplique os tamanhos indicados para a abertura do eixo X e para a abertura do eixo Y.

E vamos vê-lo na prática:

```
.container {
  /* gap: <row> <column> */
  gap: 4px 8px;

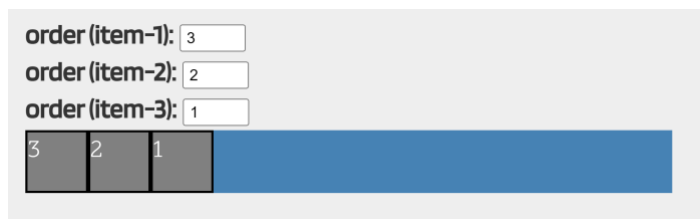
  /* 1 parâmetro: usa el mismo para ambos */
  gap: 4px;
}
```



Ordem dos ítems

Finalmente, e talvez uma das propriedades mais interessantes, é a [order](#), que modifica e estabelece a ordem dos ítems de acordo com uma sequência numérica.

Por padrão, todos os ítems [flex](#) têm uma [order: 0](#) implícito, mesmo que não esteja especificada. Se indicarmos uma [order](#) com um valor numérico, ela reposicionará os ítems de acordo com seu número, colocando primeiro os ítems com números menores (mesmo valores negativos) e depois os ítems com números maiores.



Desta forma, podemos reposicionar facilmente os ítems mesmo usando [media queries](#) ou [responsive design](#).

