



MATRICULE-SE

PROGRAMAÇÃO _

DATA SCIENCE _

DEVOPS _

MOBILE _

FRONT-END _

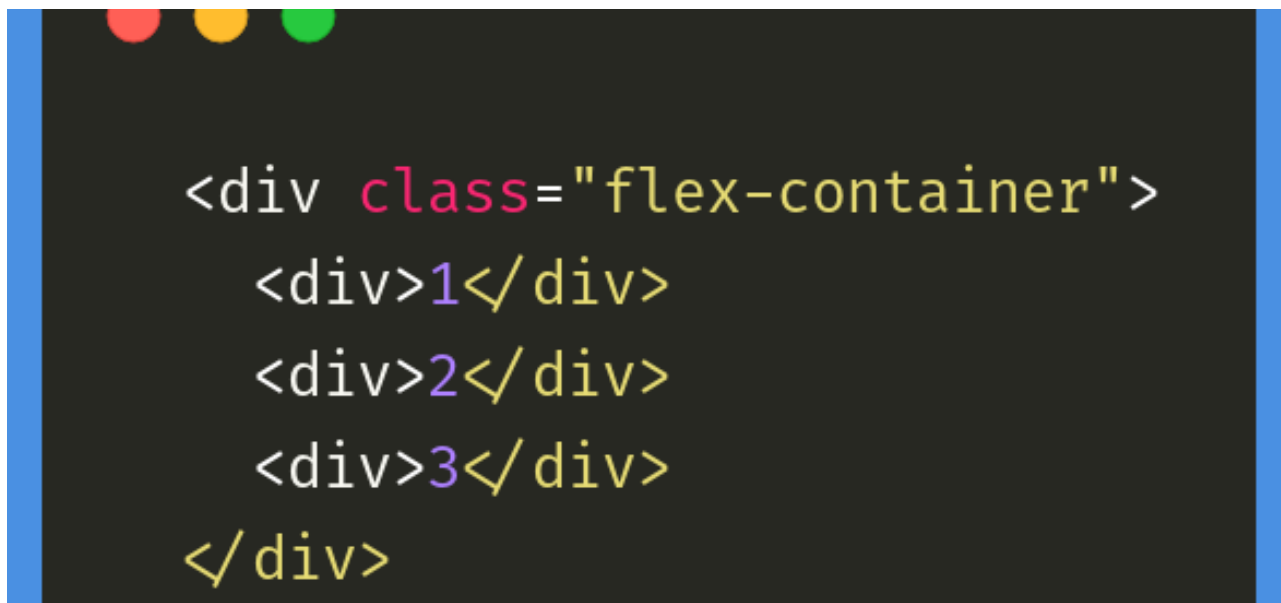
INTELIGÊNCIA ARTIFICIAL _

UX & DESIGN _

INOVAÇÃO & GESTÃO _

Artigos > **Front-end**

Flexbox CSS: Guia Completo, Elementos e Exemplos



Juliana Amoasei

Atualizado em 18/09/2023

COMPARTILHE



- [O que é o Flexbox](#)
- [Elementos](#)
- [Propriedades para o elemento-pai](#)
- [Propriedades para elementos-filhos](#)
- [Importante!](#)
- [Vamos praticar?](#)
- [Links úteis](#)

O Flexbox tem como meta ser um modo mais eficiente para criar leiautes, alinhar e distribuir espaços entre itens em um container, mesmo quando as dimensões destes itens são desconhecidas e/ou dinâmicas (daí o termo "flex").

Vamos aprender os fundamentos do CSS Flexbox para alinhamento e posicionamento, e como utilizar suas funcionalidades corretamente.

O que é o *Flexbox*

Por muito tempo, as únicas ferramentas disponíveis para criar leiautes em CSS e posicionar elementos com boa compatibilidade entre browsers eram `float` e `position`. Porém, essas ferramentas possuem algumas limitações muito frustrantes, especialmente no que diz respeito à responsividade. Algumas tarefas que consideramos básicas em um leiaute, como centralização vertical de um elemento-filho com relação a um elemento-pai ou fazer com que elementos-filhos ocupem a mesma quantidade de espaço, ou colunas terem o mesmo tamanho independente da quantidade de conteúdo interno, eram impossíveis ou muito difíceis de serem manejadas com floats ou position, ao menos de forma prática e *flexível*.

A ferramenta Flexbox (de *Flexible Box*) foi criada para tornar essas tarefas mais simples e funcionais: os filhos de um elemento com Flexbox podem se posicionar em qualquer direção e pode ter dimensões flexíveis para se adaptar.



COM PYTHON DO ZERO

CINCO AULAS GRATUITAS PARA VOCÊ APRENDER A USAR PYTHON NA PRÁTICA E DESENVOLVER HABILIDADES ESSENCIAIS EM ANÁLISE DE DADOS

INSCREVA-SE AGORA

100% gratuito

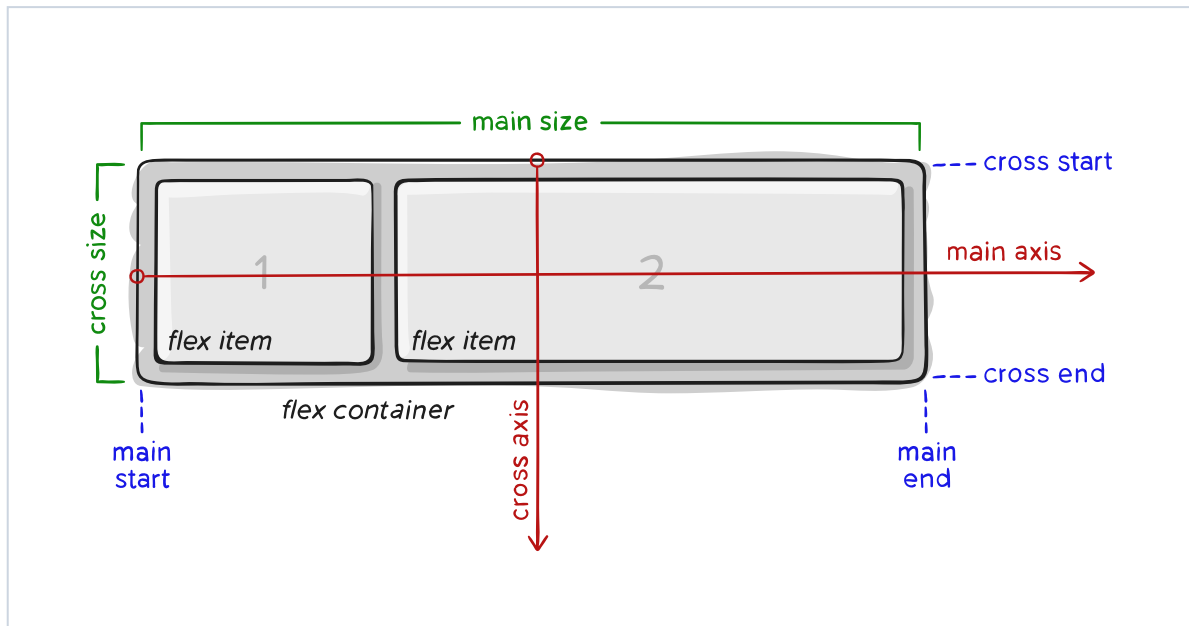
De 25 a 29 de Março

Certificado Alura

Elementos

O Flexbox é um módulo completo e não uma única propriedade; algumas delas devem ser declaradas no container (o elemento-pai, que chamamos de *flex container*), enquanto outras devem ser declaradas nos elementos-filhos (os *flex items*).

Se o leiaute "padrão" é baseado nas direções `block` e `inline`, o leiaute Flex é baseado em direções "flex flow". Veja abaixo um diagrama da especificação, explicando a ideia central por trás do leiaute Flex.



Os itens serão dispostos no leiaute seguindo ou o eixo principal ou o transversal.

- Eixo principal: o eixo principal de um *flex container* é o eixo primário e ao longo dele são inseridos os *flex items*. **Cuidado:** O eixo principal não é necessariamente horizontal; vai depender da propriedade `flex-direction` (veja abaixo).



da direção do container, é o tamanho principal do item. A propriedade de tamanho principal de um *flex item* pode ser tanto `width` quanto `height`, dependendo de qual delas estiver na direção principal.

- Eixo transversal: O eixo perpendicular ao eixo principal é chamado de eixo transversal. Sua direção depende da direção do eixo principal.
- *cross-start* / *cross-end*: Linhas flex são preenchidas com itens e adicionadas ao container, começando pelo lado *cross start* do *flex container* em direção ao lado *cross end*.
- *cross size*: A largura ou altura de um *flex item*, dependendo do que estiver na dimensão transversal, é o *cross size* do item. A propriedade *cross size* pode ser tanto a largura quanto a altura do item, o que estiver na transversal.

Flex container é o elemento que envolve sua estrutura. Você define que um elemento é um Flex Container com a propriedade `display` e valores `flex` ou `inline-flex`.

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

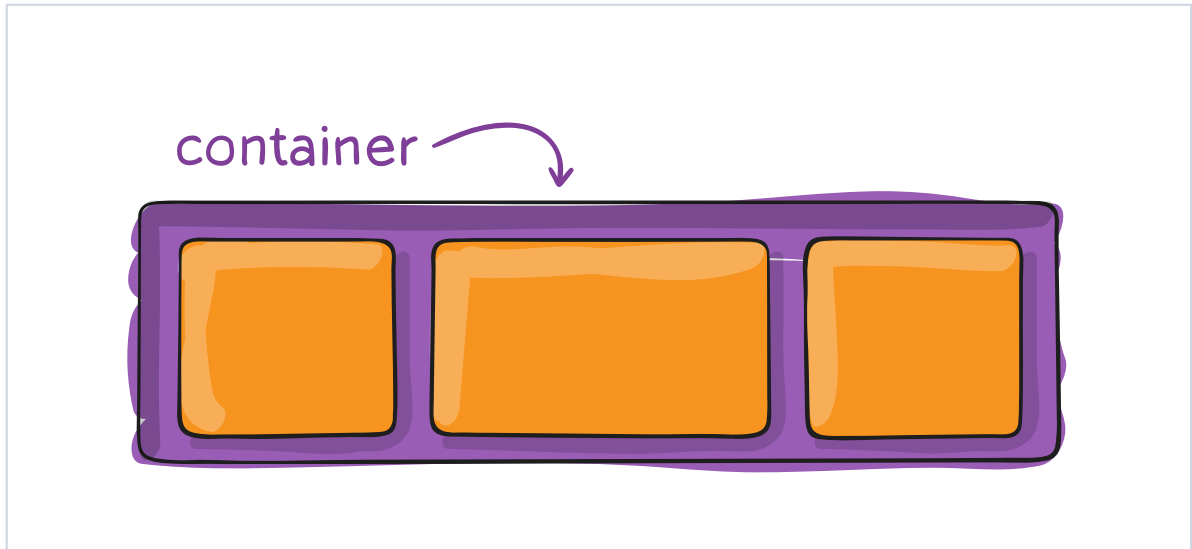
```
.flex-container {
  display: flex;
}
```

Flex Item são elementos-filhos do flex container.

Eixos ou Axes são as duas direções básicas que existem em um Flex Container: *main axis*, ou eixo principal, e *cross axis*, ou eixo transversal.



elemento-pai



Quando utilizamos o *Flexbox*, é muito importante saber quais propriedades são declaradas no elemento-pai (por exemplo, uma `div` que irá conter os elementos a serem alinhados) e quais serão declaradas nos elementos-filhos. Abaixo, seguem propriedades que devem ser declaradas utilizando o elemento-pai como seletor (para alinhar elementos-filhos):

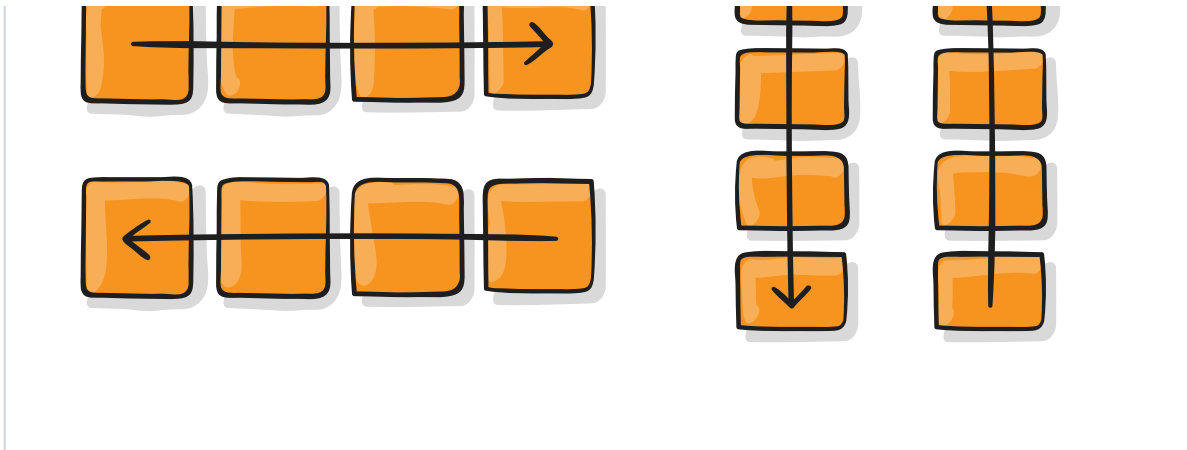
display

Esta propriedade define um *flex container*, inline ou block dependendo dos valores passados. Coloca todos os elementos-filhos diretos num contexto Flex.

```
.container {  
  display: flex; /* or inline-flex */  
}
```

Note que a propriedade de CSS `columns` não tem efeito em um *flex container*.

flex-direction

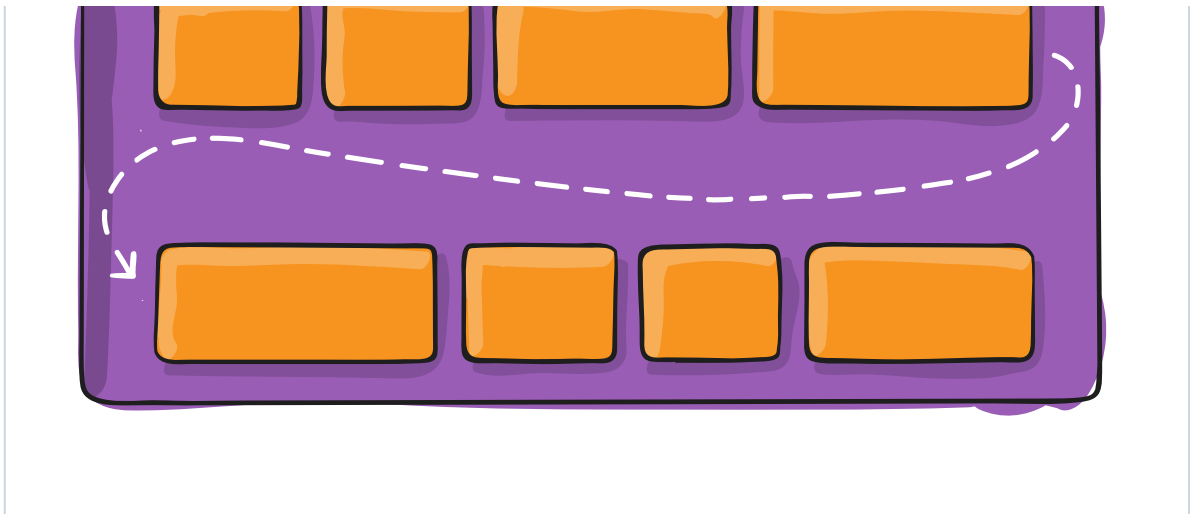


Estabelece o eixo principal, definindo assim a direção em que os *flex items* são alinhados no *flex container*. O Flexbox é (com exceção de um wrapping opcional) um conceito de layout de uma só direção. Pense nos *flex items* inicialmente posicionais ou em linhas horizontais ou em colunas verticais.

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

- `row` (padrão): esquerda para a direita em `ltr` (left to right), direita para a esquerda em `rtl` (right to left)
- `row-reverse`: direita para a esquerda em `ltr`, esquerda para a direita em `rtl`
- `column`: mesmo que `row`, mas de cima para baixo
- `column-reverse`: mesmo que `row-reverse` mas de baixo para cima

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.

```
.flex-container {  
  flex-wrap: nowrap | wrap | wrap-reverse;  
}
```

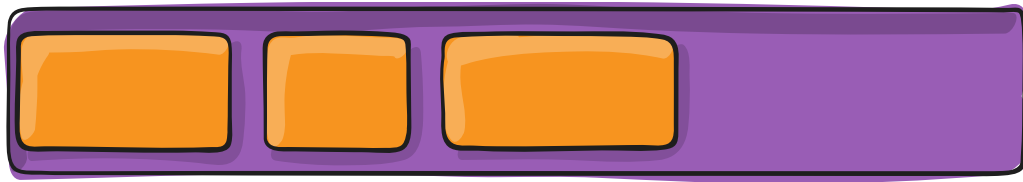
- `nowrap` (padrão): todos os *flex items* ficarão em uma só linha
- `wrap` : os *flex items* vão quebrar em múltiplas linhas, de cima para baixo
- `wrap-reverse` : os *flex items* vão quebrar em múltiplas linhas de baixo para cima

flex-flow

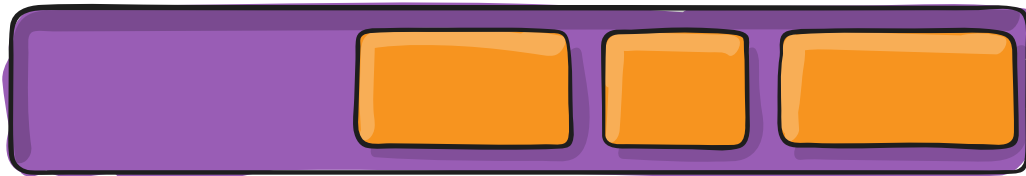
A propriedade **flex-flow** é uma propriedade *shorthand* (uma mesma declaração inclui vários valores relacionados a mais de uma propriedade) que inclui `flex-direction` e `flex-wrap` . Determina quais serão os eixos principal e transversal do container. O valor padrão é `row nowrap` .



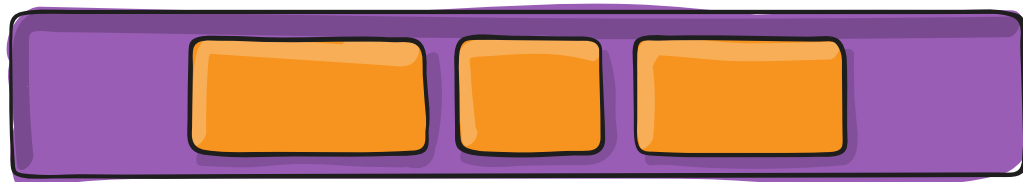
justify-content



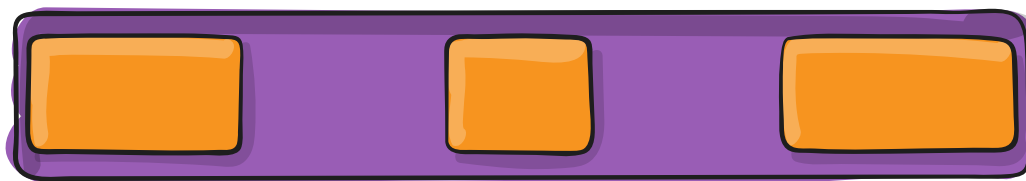
flex-end



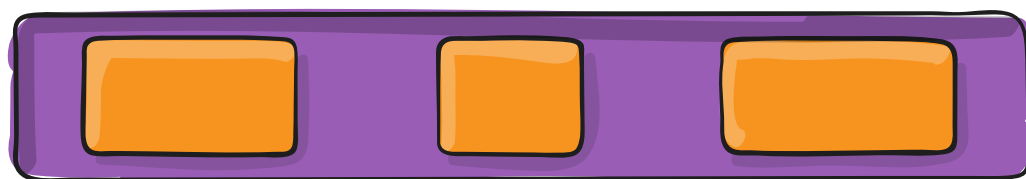
center



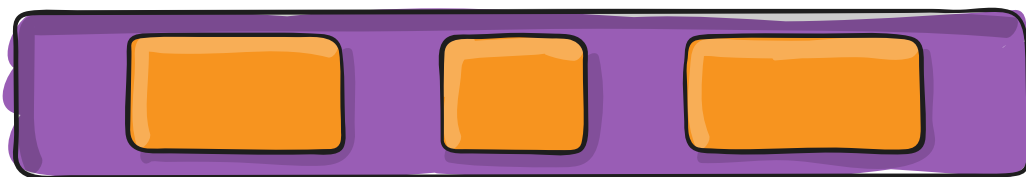
space-between



space-around



space-evenly



Esta propriedade define o alinhamento dos ítems ao longo do eixo principal. Ajuda a distribuir o espaço livre que sobra no container tanto se todos os flex items em uma linha são inflexíveis, ou são flexíveis mas já atingiram seu



```
.flex-container {  
  justify-content: flex-start | flex-end | center | space-between  
}
```

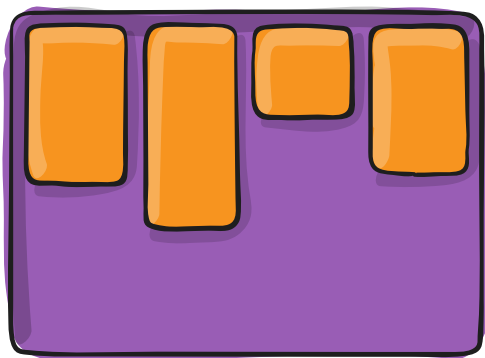
- `flex-start` (padrão): os itens são alinhados junto à borda de início (start) de acordo com qual for a `flex-direction` do container.
- `flex-end` : os itens são alinhados junto à borda final (end) de acordo com qual for a `flex-direction` do container.
- `start` : os itens são alinhados junto à borda de início da direção do `writing-mode` (modo de escrita).
- `end` : os itens são alinhados junto à borda final da direção do `writing-mode` (modo de escrita).
- `left` : os itens são alinhados junto à borda esquerda do container, a não ser que isso não faça sentido com o `flex-direction` que estiver sendo utilizado. Nesse caso, se comporta como `start` .
- `right` : os itens são alinhados junto à borda direita do container, a não ser que isso não faça sentido com o `flex-direction` que estiver sendo utilizado. Nesse caso, se comporta como `start` .
- `center` : os itens são centralizados na linha.
- `space-between` : os itens são distribuídos de forma igual ao longo da linha; o primeiro item junto à borda inicial da linha, o último junto à borda final da linha.
- `space-around` : os itens são distribuídos na linha com o mesmo espaçamento entre eles. Note que, visualmente, o espaço pode não ser igual, uma vez que todos os itens tem a mesma quantidade de espaço dos dois lados: o primeiro item vai ter somente uma unidade de espaço junto à borda do container, mas duas unidades de espaço entre ele e o próximo item, pois o próximo item também tem seu próprio espaçamento que está sendo aplicado.
- `space-evenly` : os itens são distribuídos de forma que o espaçamento entre quaisquer dois itens da linha (incluindo entre os itens e as bordas) seja igual.



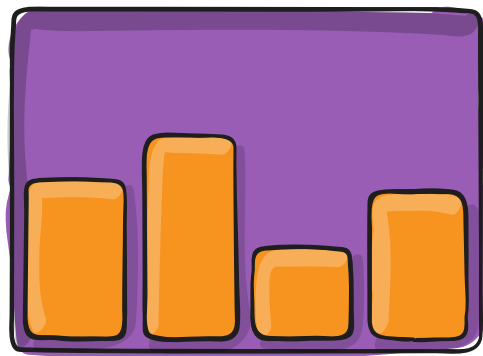
implementados no Chrome. Para tabelas detalhadas, consulte o MDN. Os valores mais seguros são `flex-start`, `flex-end` e `center`.

Também existem duas palavras-chave adicionais que você pode usar em conjunto com estes valores: `safe` e `unsafe`. `Safe` garante que, independente da forma que você faça esse tipo de posicionamento, não seja possível "empurrar" um elemento e fazer com que ele seja renderizado para fora da tela (por exemplo, acima do topo), de uma forma que faça com que o conteúdo seja impossível de movimentar com a rolagem da tela (o CSS chama isso de "perda de dados").

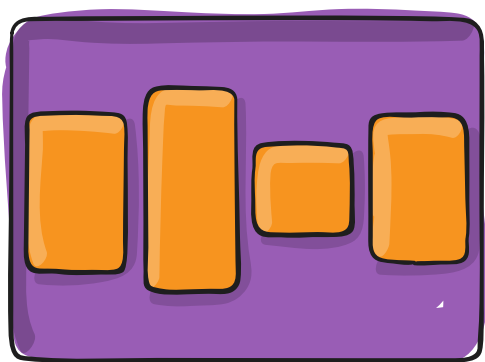
align-items



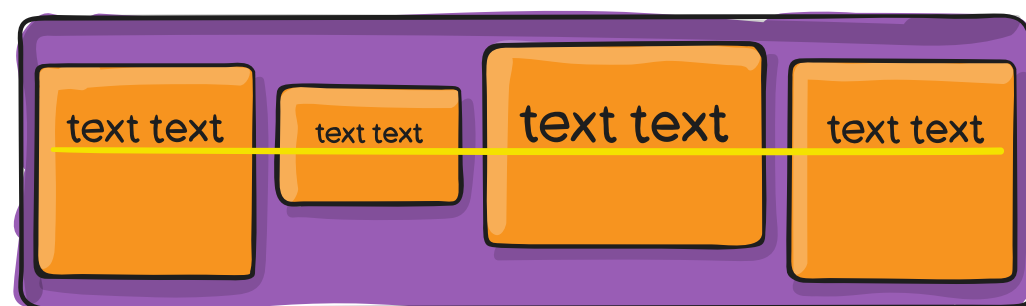
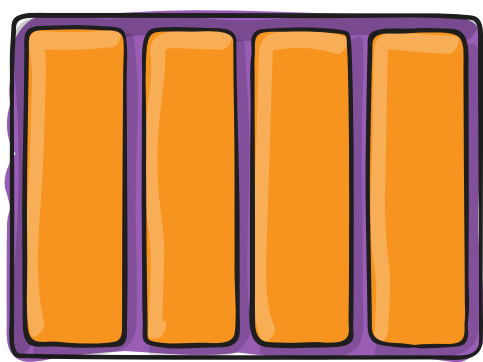
center



stretch



baseline



define o comportamento padrão de como *flex items* são alinhados de acordo com o eixo transversal (*cross axis*). De certa forma, funciona de forma similar ao `justify-content`, porém no eixo transversal (perpendicular ao eixo principal).

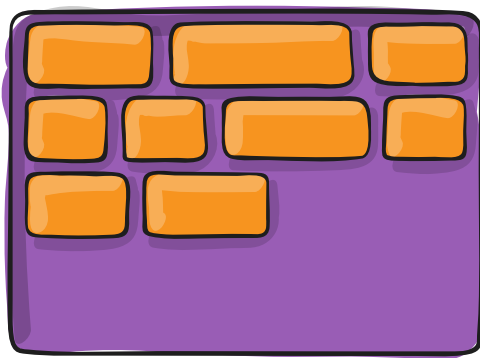
```
.flex-container {  
  align-items: stretch | flex-start | flex-end | center | baseline  
}
```



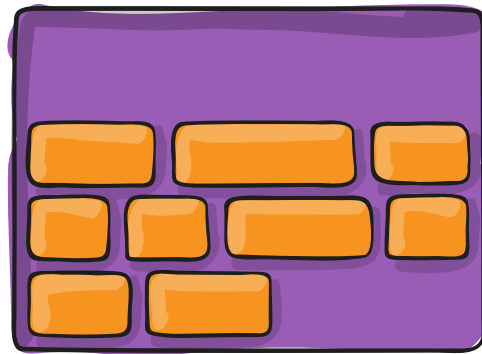
- `flex-start` / `start` / `self-start` : ítems são posicionados no início do eixo transversal. A diferença entre eles é sutil e diz respeito às regras de `flex-direction` OU `writing-mode`.
- `center` : ítems são centralizados no eixo transversal.
- `baseline` : ítems são alinhados de acordo com suas baselines.

Os modificadores `safe` e `unsafe` pode ser usados em conjunto com todas essas palavras-chave (favor conferir o suporte de cada navegador) e servem para prevenir qualquer alinhamento de elementos que faça com que o conteúdo fique inacessível (por exemplo, para fora da tela).

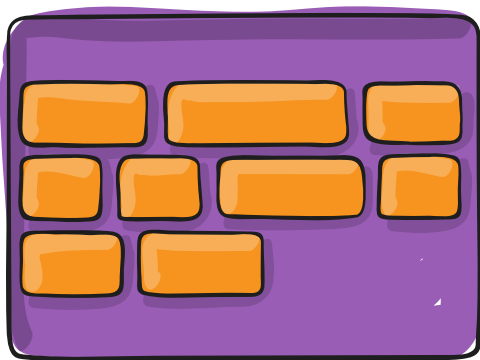
align-content



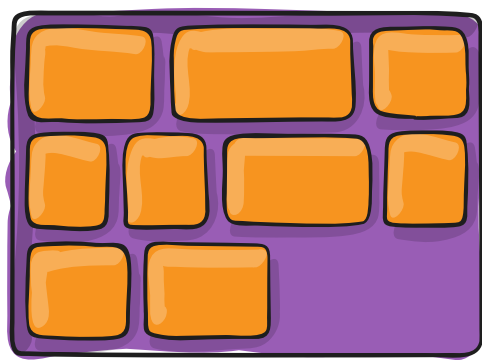
center



stretch



space-between



space-around

Organiza as linhas dentro de um flex container quando há espaço extra no eixo transversal, similar ao modo como `justify-content` alinha itens individuais dentro do eixo principal.

Importante: Esta propriedade não tem efeito quando há somente uma linha de flex items no container.



- `flex-start` / `start` : ítems alinhados com o início do container. O valor (com maior suporte dos navegadores) `flex-start` se guia pela `flex-direction`, enquanto `start` se guia pela direção do `writing-mode`.
- `flex-end` / `end` : ítems alinhados com o final do container. O valor (com maior suporte dos navegadores) `flex-end` se guia pela `flex-direction`, enquanto `end` se guia pela direção do `writing-mode`.
- `center` : ítems centralizados no container.
- `space-between` : ítems distribuídos igualmente; a primeira linha junto ao início do container e a última linha junto ao final do container.
- `space-around` : ítems distribuídos igualmente com o mesmo espaçamento entre cada linha.
- `space-evenly` : ítems distribuídos igualmente com o mesmo espaçamento entre eles.
- `stretch` (padrão): ítems em cada linha esticam para ocupar o espaço remanescente entre elas.

Os modificadores `safe` e `unsafe` pode ser usados em conjunto com todas essas palavras-chave (favor conferir o suporte de cada navegador) e servem para prevenir qualquer alinhamento de elementos que faça com que o conteúdo fique inacessível (por exemplo, para fora da tela).

Propriedades para elementos-filhos

A seguir, veremos propriedades que devem ser declaradas tendo como seletor os elementos-filhos, ou seja:

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

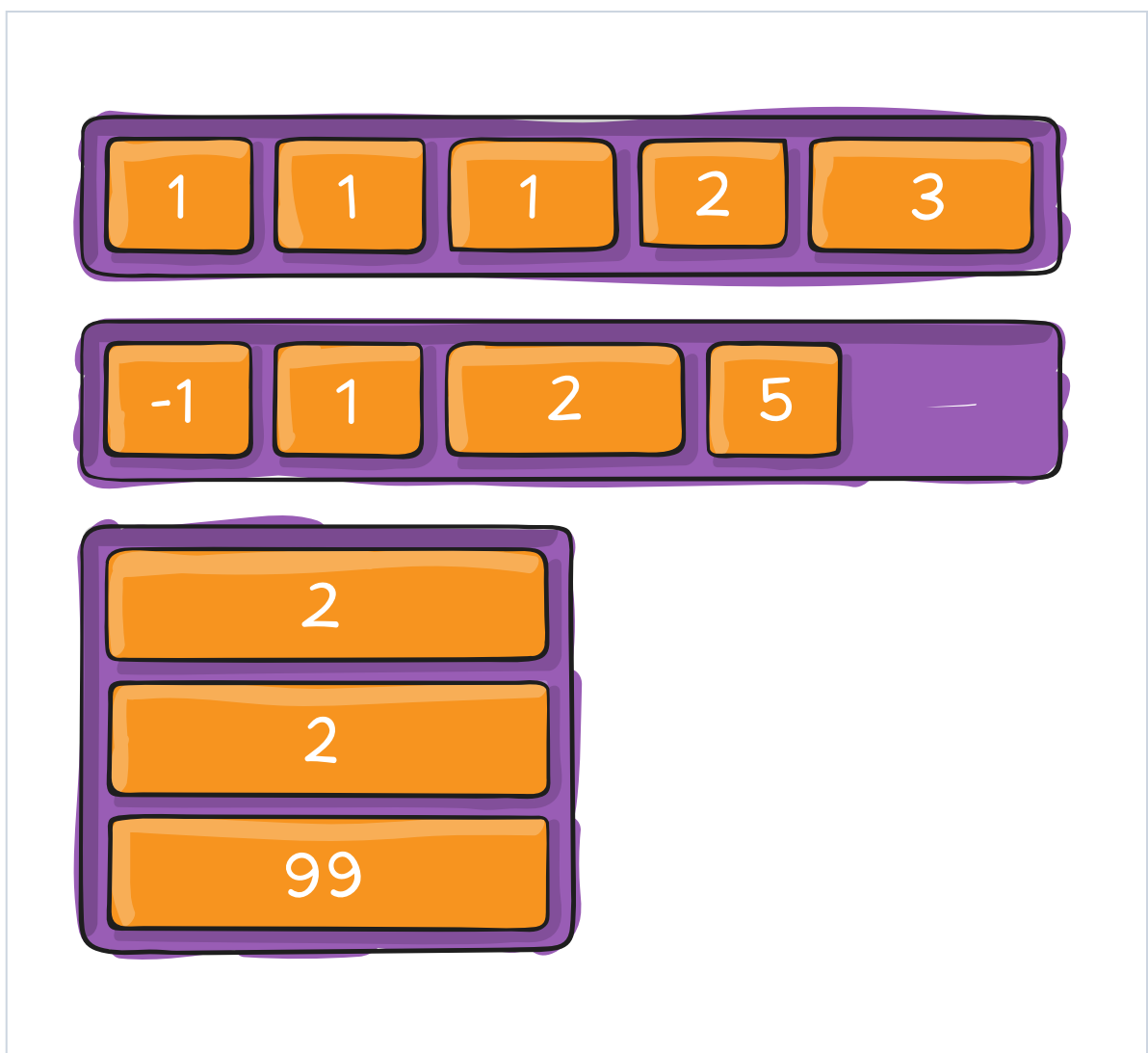


Isso significa que, onde existe um elemento-pai com propriedade *flex* (o *flex-container*), é possível atribuir propriedades flex específicas também para as elementos-filhos (*flex-item*).

Você pode definir as propriedades abaixo para apenas um dos elementos-filhos através de um identificador, como uma classe específica.

order

Determina a ordem em que os elementos aparecerão.

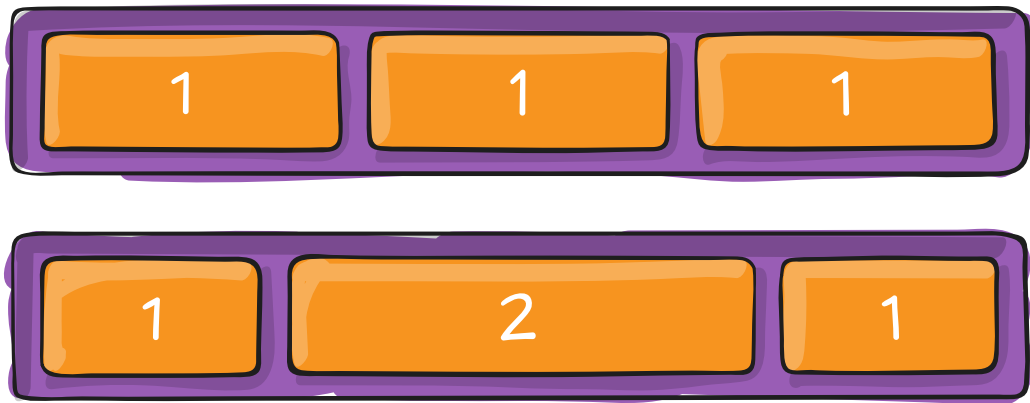


Por padrão os flex items são dispostos na tela na ordem do código. Mas a propriedade `order` controla a ordem em que aparecerão no container.



```
order: <número>; /* o valor padrão é 0 */  
}
```

flex-grow



Define a habilidade de um flex item de crescer, caso necessário. O valor dessa propriedade é um valor numérico sem indicação de unidade, que serve para cálculo de proporção. Este valor dita a quantidade de espaço disponível no container que será ocupado pelo item.

Se todos os itens tiverem flex-grow definido em 1, o espaço remanescente no container será distribuído de forma igual entre todos. Se um dos itens tem o valor de 2, vai ocupar o dobro de espaço no container com relação aos outros (ou pelo menos vai tentar fazer isso).

```
.flex-item {  
  flex-grow: <numero>; /* o valor default(padão) é 0 */  
}
```

Valores negativos não são aceitos pela propriedade.



Define a habilidade de um flex item de encolher, caso necessário.

```
.flex-item {  
  flex-shrink: <número>; /* o valor padrão é 0 */  
}
```

Valores negativos não são aceitos pela propriedade.

flex-basis

Define o tamanho padrão para um elemento antes que o espaço remanescente do container seja distribuído. Pode ser um comprimento (por exemplo, 20%, 5rem, etc) ou uma palavra-chave. A palavra-chave `auto` significa "observe minhas propriedades de altura ou largura" (o que era feito pela palavra-chave `main-size`, que foi depreciada). A palavra-chave `content` significa "estabeleça o tamanho com base no conteúdo interno do item" - essa palavra-chave ainda não tem muito suporte, então não é fácil de ser testada, assim como suas relacionadas: `max-content`, `min-content` e `fit-content`.

```
.flex-item {  
  flex-basis: flex-basis: | auto; /* o valor padrão é auto */  
}
```

Com o valor de `0`, o espaço extra ao redor do conteúdo não é considerado. Com o valor de `auto`, o espaço extra é distribuído com base no valor de `flex-grow` do item.

flex

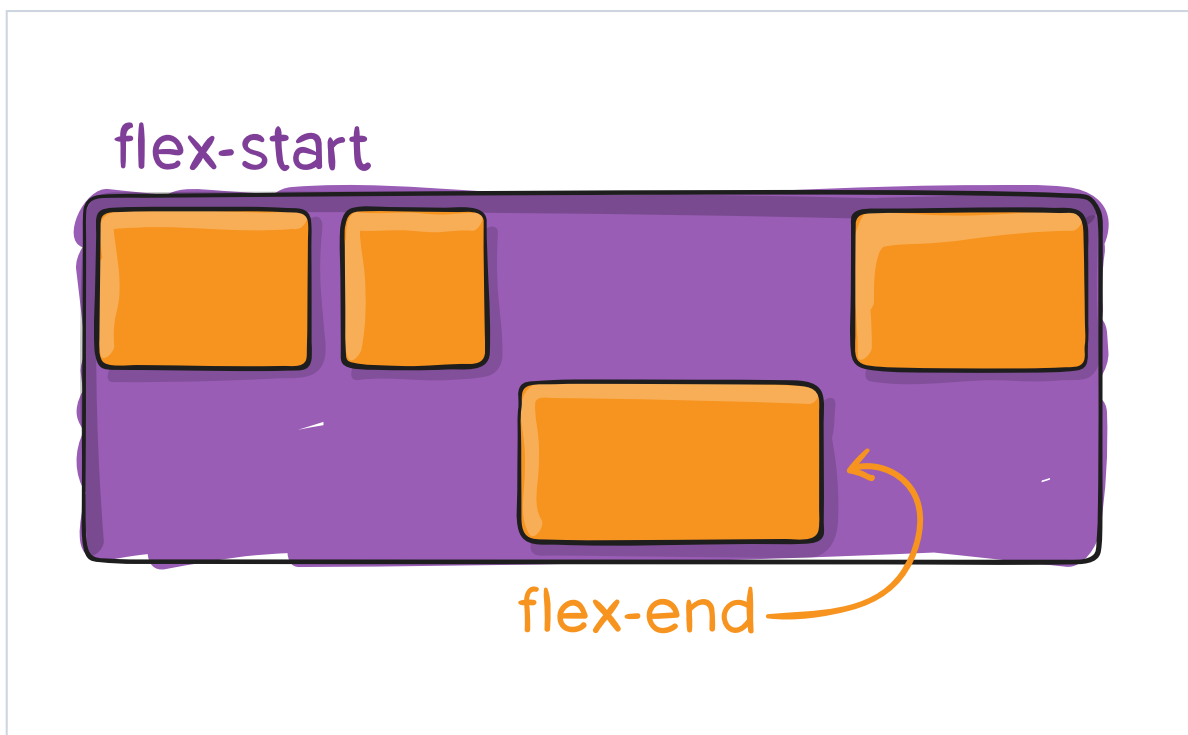
Esta é a propriedade *shorthand* para `flex-grow`, `flex-shrink` e `flex-basis`, combinadas. O segundo e terceiro parâmetros (`flex-shrink` e `flex-basis`) são



```
.item {  
  flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]  
}
```

É recomendado que você utilize esta propriedade *shorthand* ao invés de definir cada uma das propriedades em separado. O *shorthand* define os outros valores de forma inteligente.

align-self



Permite que o alinhamento padrão (ou o que estiver definido por `align-items`) seja sobrescrito para ítems individuais.

Por favor veja a explicação da propriedade `align-items` para entender quais são os possíveis valores.



Importante!

- O CSS só enxerga a hierarquia pai-filho; não vai aplicar as propriedades Flex para elementos que não estejam diretamente relacionados;
- Para que as propriedades funcionem nos elementos-filhos, as pais devem ter propriedade `display: flex;` .
- As propriedades `float` , `clear` e `vertical-align` não têm efeito em flex-items.

Vamos praticar?

[Flexbox Froggy](#)

Links úteis

Assim como qualquer outra nova funcionalidade que aprendemos, é fundamental praticar bastante e pesquisar sempre que temos dúvidas. A seguir, alguns links úteis.

Este conteúdo se baseia em grande parte no tutorial da [CSS Tricks](#), com nossos agradecimentos! Guarde nos seus favoritos para consultar sempre que precisar.

O guia do CSS Tricks também aborda alguns outros aspectos importantes do Flex: suporte dos navegadores, bugs, propriedades relacionadas, prefixos e etc. Caso tenha alguma dúvida que não foi abordada neste artigo, você pode consultar estes temas relacionados no link.

- [Flexbox no MDN](#)



[end](#) onde você vai aprender mais sobre **HTML e CSS**



Juliana Amoasei

Desenvolvedora JavaScript com background multidisciplinar, sempre aprendendo para ensinar e vice-versa. Acredito no potencial do conhecimento como agente de mudança pessoal e social. Atuo como instrutora na Escola de Programação da Alura e, fora da tela preta, me dedico ao Kung Fu e a nerdices em geral.

[Artigo Anterior](#)

CSS Funcional e outros nomes

[Próximo Artigo](#)

Vamos criar um site com HTML e CSS na Imersão Hipsters.CSS!

Veja outros artigos sobre
[Front-end](#)

Quer mergulhar em tecnologia e aprendizagem?

Receba a newsletter que o nosso CEO escreve pessoalmente, com insights do mercado de trabalho, ciência e desenvolvimento de software



MATRICULE-SE

ME INSCREVA

Nossas redes e apps



Institucional

[Sobre nós](#)

[Trabalhe conosco](#)

[Para Empresas](#)

[Para Sua Escola](#)

[Política de Privacidade](#)

[Compromisso de Integridade](#)

[Termos de Uso](#)

[Status](#)

A Alura

[Formações](#)

[Como Funciona](#)

[Todos os cursos](#)

[Depoimentos](#)

[Instrutores\(as\)](#)

[Dev em <T>](#)

[Luri by ChatGPT](#)



MATRICULE-SE

Imersões

Perguntas frequentes

Artigos

Podcasts

Artigos de educação
corporativa

Novidades e Lançamentos

Email*

ENVIAR

CURSOS

Cursos de Programação

Lógica | Python | PHP | Java | .NET | Node JS | C | Computação | Jogos | IoT

Cursos de Front-end

HTML, CSS | React | Angular | JavaScript | jQuery

Cursos de Data Science

Ciência de dados | BI | SQL e Banco de Dados | Excel | Machine Learning | NoSQL | Estatística

Cursos de Inteligência Artificial

IA para Programação | IA para Dados

Cursos de DevOps

AWS | Azure | Docker | Segurança | IaC | Linux

Cursos de UX & Design

Usabilidade e UX | Vídeo e Motion | 3D

Cursos de Mobile

React Native | Flutter | iOS e Swift | Android, Kotlin | Jogos



MATRICULE-SE