



# CSS Flexbox

October 2022

# Agenda

---

**1** WHY FLEXBOX

**2** FLEX MODEL

**3** FLEXBOX PROPERTIES FOR THE PARENT

**4** FLEXBOX PROPERTIES FOR THE CHILDREN

## CSS Flexbox



## WHY FLEXBOX

# Why Flexbox

---

For a long time, the only reliable cross-browser compatible tools available for creating CSS layouts were features like [floats](#) and [positioning](#). These work, but in some ways they're also limiting and frustrating.

The following simple layout designs are either difficult or impossible to achieve with such tools in any kind of convenient, flexible way:

- Vertically centering a block of content inside its parent.
- Making all the children of a container take up an equal amount of the available width/height, regardless of how much width/height is available.
- Making all columns in a multiple-column layout adopt the same height even if they contain a different amount of content.

As you'll see in subsequent sections, flexbox makes a lot of layout tasks much easier. Let's dig in!

# Why Flexbox

## WITHOUT FLEXBOX

### Sample flexbox example

#### First article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

#### Second article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

#### Third article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

Cray food truck brunch, XOXO +1 keffiyeh pickled chambray waistcoat ennui. Organic small batch paleo 8-bit. Intelligentsia umami wayfarers pickled, asymmetrical kombucha letterpress kitsch leggings cold-pressed squid chartreuse put a bird on it. Llisticle pickled man bun cornhole heirloom art party.

## WITH FLEXBOX

### Sample flexbox example

#### First article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

#### Second article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

#### Third article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

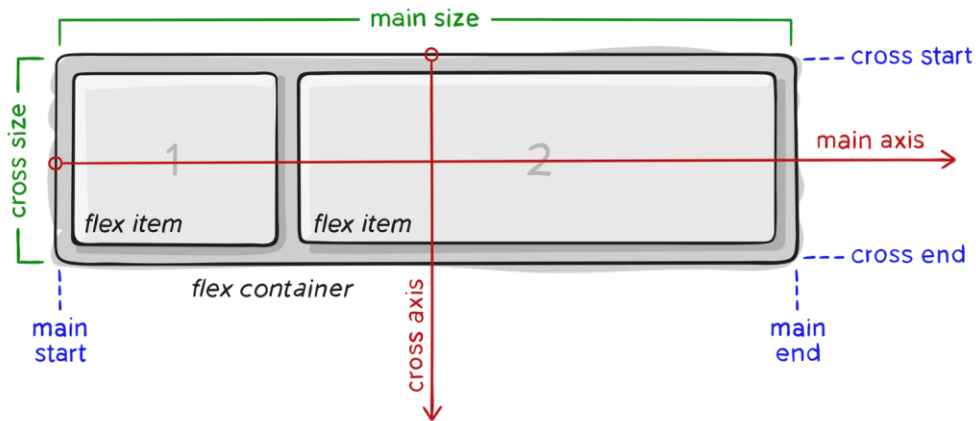
Cray food truck brunch, XOXO +1 keffiyeh pickled chambray waistcoat ennui. Organic small batch paleo 8-bit. Intelligentsia umami wayfarers pickled, asymmetrical kombucha letterpress kitsch leggings cold-pressed squid chartreuse put a bird on it. Llisticle pickled man bun cornhole heirloom art party.

## FLEX MODEL

# Flex model

When elements are laid out as flex items, they are laid out along two axes

- The **main axis** is the axis running in the direction the flex items are laid out in (for example, as rows across the page, or columns down the page.) The start and end of this axis are called the **main start** and **main end**.
- The **cross axis** is the axis running perpendicular to the direction the flex items are laid out in. The start and end of this axis are called the **cross start** and **cross end**.
- The parent element that has display: flex set on it is called the **flex container**.
- The items laid out as flexible boxes inside the flex container are called **flex items**.



## **FLEXBOX PROPERTIES FOR THE PARENT**

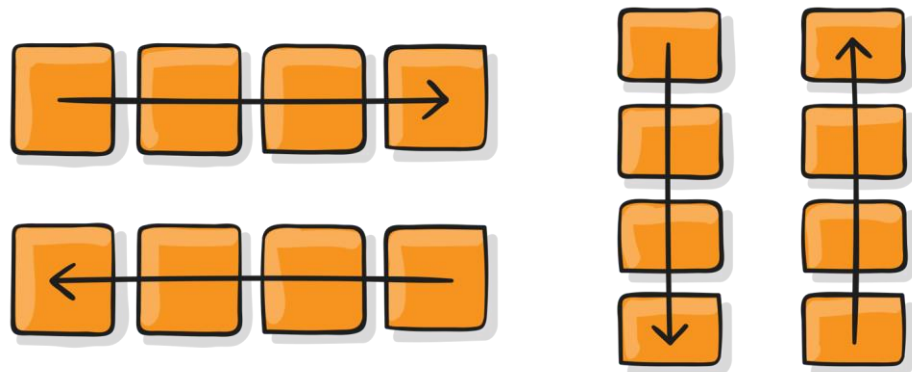


# flex-direction

This establishes the main-axis, thus defining the direction flex items are placed in the flex container. Flexbox is (aside from optional wrapping) a single-direction layout concept. Think of flex items as primarily laying out either in horizontal rows or vertical columns.

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

- **row (default)**: left to right in ltr; right to left in rtl
- **row-reverse**: right to left in ltr; left to right in rtl
- **column**: same as row but top to bottom
- **column-reverse**: same as row-reverse but bottom to top

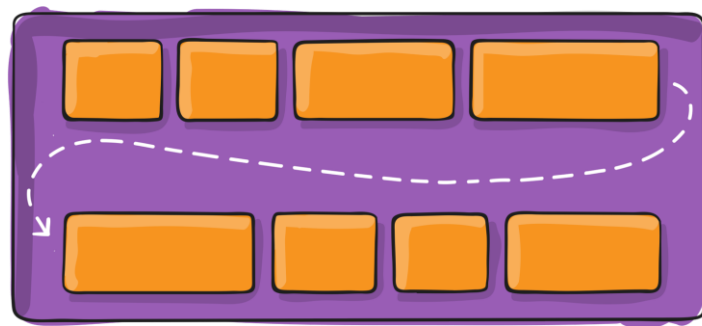


# flex-wrap

By default, flex items will all try to fit onto one line. You can change that and allow the items to wrap as needed with this property.

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


- **nowrap (default)**: all flex items will be on one line
- **wrap**: flex items will wrap onto multiple lines, from top to bottom.
- **wrap-reverse**: flex items will wrap onto multiple lines from bottom to top.



# flex-flow

---

This is a shorthand for the flex-direction and flex-wrap properties, which together define the flex container's main and cross axes. The default value is row nowrap.



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

# justify-content

This defines the alignment along the main axis. It helps distribute extra free space leftover when either all the flex items on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

```
.container {  
  justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly | start | end | left | right ... + safe | unsafe;  
}
```

flex-start



flex-end



center



space-between



space-around



space-evenly



# align-items

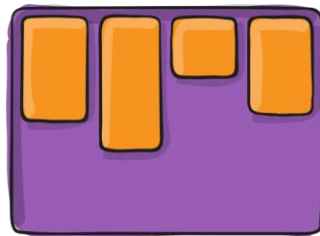
This defines the default behavior for how flex items are laid out along the **cross axis** on the current line.

Think of it as the justify-content version for the cross-axis (perpendicular to the main-axis).

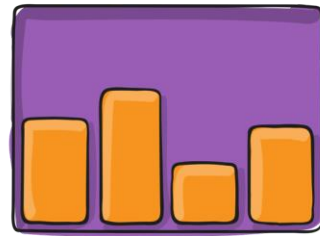


```
.container {  
  align-items: stretch | flex-start | flex-end | center | baseline | first baseline |  
  last baseline | start | end | self-start | self-end + ... safe | unsafe;  
}
```

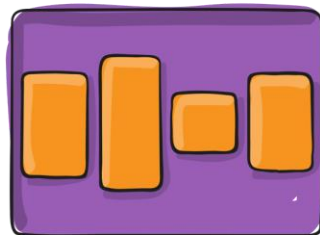
flex-start



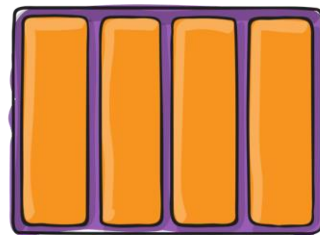
flex-end



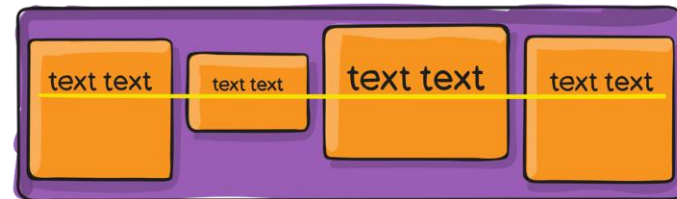
center



stretch



baseline

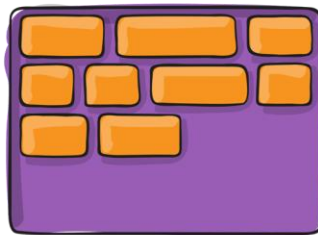


# align-content

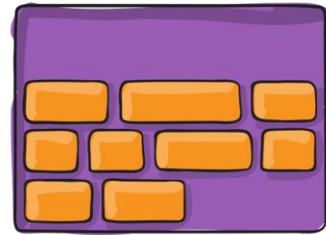
This aligns a flex container's lines within when there is extra space in the cross-axis, similar to how justify-content aligns individual items within the main-axis.

```
.container {  
  align-content: flex-start | flex-end | center | space-between | space-around |  
  space-evenly | stretch | start | end | baseline | first baseline | last baseline +  
  ... safe | unsafe;  
}
```

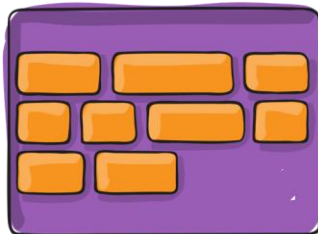
flex-start



flex-end



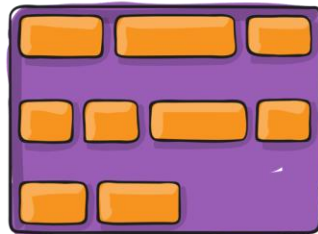
center



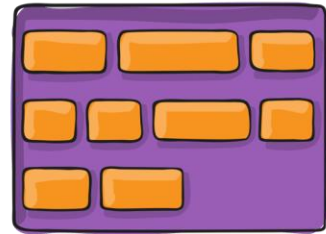
stretch



space-between



space-around



# gap, row-gap, column-gap

[The gap property](#) explicitly controls the space between flex items. It applies that spacing *only between items* not on the outer edges.

```
.container {  
  display: flex;  
  ...  
  gap: 10px;  
  gap: 10px 20px; /* row-gap column gap */  
  row-gap: 10px;  
  column-gap: 20px;  
}
```

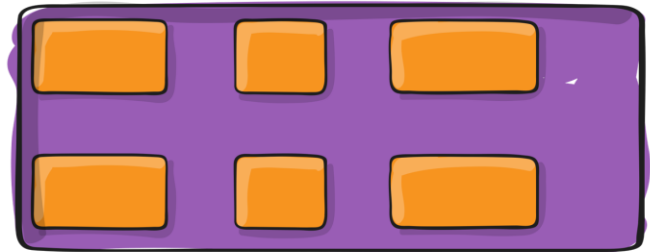
The behavior could be thought of as a *minimum* gutter, as if the gutter is bigger somehow (because of something like justify-content: space-between;) then the gap will only take effect if that space would end up smaller.

It is not exclusively for flexbox, gap works in grid and multi-column layout as well.

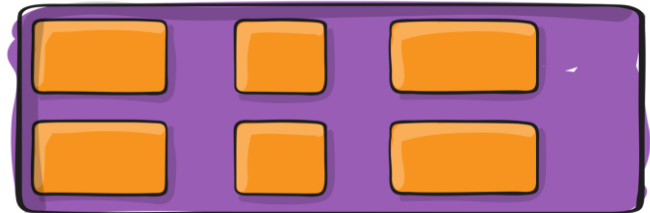
gap: 10px



gap: 30px



gap: 10px 30px



## FLEXBOX PROPERTIES FOR THE CHILDREN



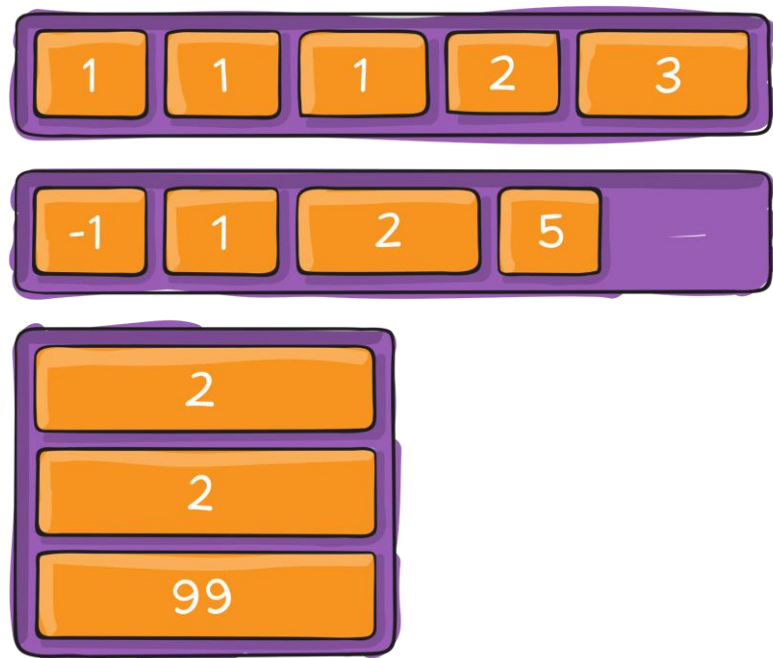
# order

By default, flex items are laid out in the source order.

However, the order property controls the order in which they appear in the flex container.

```
.item {  
  order: 5; /* default is 0 */  
}
```

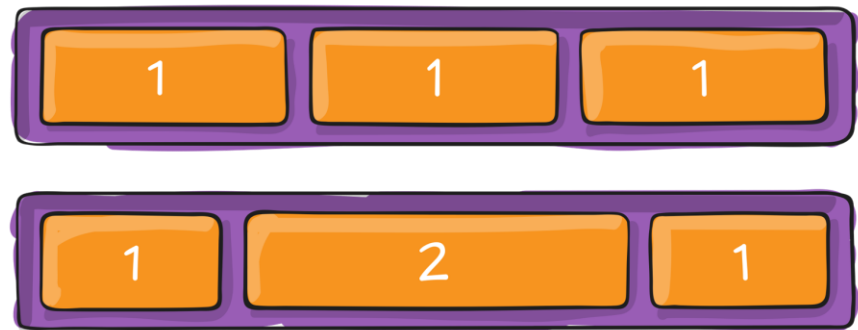
Items with the same order revert to source order.



# flex-grow

This defines the ability for a flex item to grow if necessary. It accepts a unitless value that serves as a proportion. It dictates what amount of the available space inside the flex container the item should take up.

If all items have flex-grow set to 1, the remaining space in the container will be distributed equally to all children. If one of the children has a value of 2, that child would take up twice as much of the space either one of the others (or it will try, at least).



```
.item {  
  flex-grow: 4; /* default 0 */  
}
```

Negative numbers are invalid.

# flex-basis, flex-shrink

## FLEX-BASIS

This defines the default size of an element before the remaining space is distributed. It can be a length (e.g. 20%, 5rem, etc.) or a keyword. The auto keyword means “look at my width or height property” (which was temporarily done by the main-size keyword until deprecated).

The content keyword means “size it based on the item’s content” – this keyword isn’t well supported yet, so it’s hard to test and harder to know what its brethren max-content, min-content, and fit-content do.

```
.item {  
  flex-basis: | auto; /* default auto */  
}
```

If set to 0, the extra space around content isn’t factored in. If set to auto, the extra space is distributed based on its flex-grow value.

## FLEX-SHRINK

This defines the ability for a flex item to shrink if necessary.

```
.item {  
  flex-shrink: 3; /* default 1 */  
}
```

Negative numbers are invalid.

# flex

---

This is the shorthand for flex-grow, flex-shrink and flex-basis combined. The second and third parameters (flex-shrink and flex-basis) are optional. The default is 0 1 auto, but if you set it with a single number value, like flex: 5;, that changes the flex-basis to 0%, so it's like setting flex-grow: 5; flex-shrink: 1; flex-basis: 0%;.

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

**It is recommended that you use this shorthand property** rather than set the individual properties. The shorthand sets the other values intelligently.

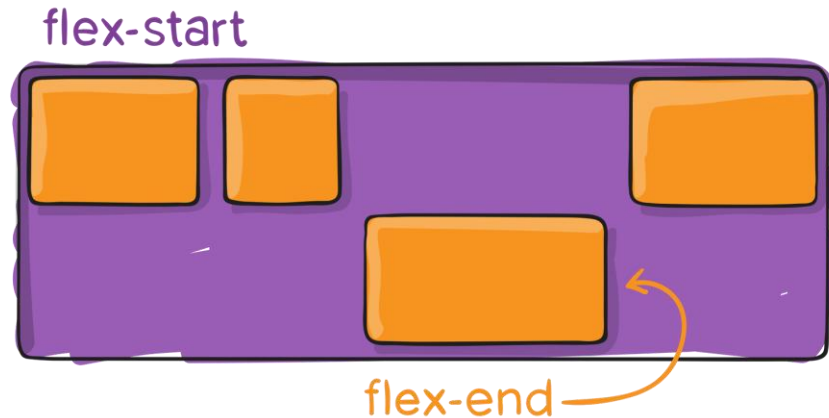
# align-self

This allows the default alignment (or the one specified by align-items) to be overridden for individual flex items.

Please see the align-items explanation to understand the available values.

```
.item {  
  align-self: auto | flex-start | flex-end | center | baseline | stretch;  
}
```

Note that float, clear and vertical-align have no effect on a flex item.



**THANK YOU!**