Flexbox

- What can you do with a flex layout?
- The main axis and the cross axis
- Creating a flex container
- Controlling the direction of items
  - Reversing the flow of items and accessibility
  - Writing modes and direction
- Wrapping flex items
  - The flex-flow shorthand
- Controlling space inside flex items
  - Allowing items to grow at different rates
- Reordering flex items
- Flexbox alignment overview
- Distributing space on the main axis
  - With flex-direction: column
- Distributing space between flex lines
  - The place-content shorthand
- Aligning items on the cross-axis
- Why is there no justify-self in flexbox?
- How to center an item vertically and horizontally
- Resources

A design pattern that can be tricky in responsive design is a sidebar that sits inline with some content. Where there is viewport space, this pattern works great, but where space is condensed, that rigid layout can become problematic.

{% Codepen { user: 'web-dot-dev', id: 'poRENWv', height: 420 } %}

The Flexible Box Layout Model (flexbox) is a layout model designed for one-dimensional content. It excels at taking a bunch of items which have different sizes, and returning the best layout for those items.

This is the ideal layout model for this sidebar pattern. Flexbox not only helps lay the sidebar and content out inline, but where there's not enough space remaining, the sidebar will break onto a new line. Instead of setting rigid dimensions for the browser to follow, with flexbox, you can instead provide flexible boundaries to hint how the content could display.

> {% Codepen { user: 'web-dot-dev', id: 'xxgERMp', height: 400 } %}

What can you do with a flex layout?

Flex layouts have the following features, which you will be able to explore in this guide.

- They can display as a row, or a column.
- They respect the writing mode of the document.
- They are single line by default, but can be asked to wrap onto multiple lines.
- Items in the layout can be visually reordered, away from their order in the DOM.
- Space can be distributed inside the items, so they become bigger and smaller according to the space available in their parent.
- Space can be distributed around the items and flex lines in a wrapped layout, using the Box Alignment properties.
- The items themselves can be aligned on the cross axis.

The main axis and the cross axis

The key to understanding flexbox is to understand the concept of a main axis and a cross axis. The main axis is the one set by your `flex-direction` property. If that is `row` your main axis is along the row, if it is `column` your main axis is along the column.

> {% Img src="image/VbAJIREinuYvovrBzzvEyZOpw5w1/xKtf0cHRw0xQyiyYuuyz.svg", alt="Three boxes next to each other with an arrow, pointing left to right. The arrow is labelled Main axis", width="800", height="320" %}

Flex items move as a group on the main axis. Remember: we've got a bunch of things and we are trying to get the best layout for them as a group.

The cross axis runs in the other direction to the main axis, so if `flex-direction` is `row` the cross axis runs along the column.

> {% Img src="image/VbAJIREinuYvovrBzzvEyZOpw5w1/5wCsZcBmK5L33LS7nOmP.svg", alt="Three boxes of different heights, next to each other with an arrow, pointing left to right. The arrow is labelled Main axis. There's another arrow pointing top to bottom. This

one is labelled Cross axis", width="800", height="320" %}

You can do two things on the cross axis. You can move the items individually or as a group, so they align against each other and the flex container. Also, if you have wrapped flex lines, you can treat those lines as a group to control how space is assigned to those lines. You will see how this all works in practice throughout this guide, for now just keep in mind that the main axis follows your `flex-direction`.

Creating a flex container

Let's see how flexbox behaves by taking a group of different sized items and using flexbox to lay them out.

```
<div class="container" id="container">
  <div>One</div>
  <div>Item two</div>
  <div>The item we will refer to as three</div>
</div>
```

To use flexbox you need to declare that you want to use a flex formatting context and not regular block and inline layout. Do this by changing the value of the `display` property to `flex`.

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

As you learned in the layout guide this will give you a block-level box, with flex item children. The flex items immediately start exhibiting some flexbox behavior, using their **initial values**.

{% Aside %} All CSS properties have initial values which control how they behave "out of the box" when you haven't applied any CSS to change that initial behavior. The children of our flex container become flex items as soon as their parent gets `display: flex`, so these initial values mean that we start seeing some flexbox behavior. {% endAside %}

The initial values mean that:

- Items display as a row.
- They do not wrap.
- They do not grow to fill the container.
- They line up at the start of the container.

Controlling the direction of items

Even though you haven't added a `flex-direction` property yet, the items display as a row because the initial value of `flex-direction` is `row`. If you want a row then you don't need to add the property. To change the direction, add the property and one of the four values:

- `row`: the items lay out as a row.
- `row-reverse:` the items lay out as a row from the end of the flex container.
- `column`: the items lay out as a column.
- `column-reverse` : the items lay out as a column from the end of the flex container.

You can try out all of the values using our group of items in the demo below.

{% Codepen { user: 'web-dot-dev', id: 'bGgKNXq' } %}

Reversing the flow of items and accessibility

You should be cautious when using any properties that reorder the visual display away from how things are ordered in the HTML document, as it can negatively impact accessibility. The `row-reverse` and `column-reverse` values are a good example of this. The reordering only happens for the visual order, not the logical order. This is important to understand as the logical order is the order that a screen reader will read out the content, and anyone navigating using the keyboard will follow.

You can see in the following video how in a reversed column layout, tabbing between links becomes disconnected as the keyboard navigation follows the DOM not the visual display.

{% Video src="video/VbAJIREinuYvovrBzzvEyZOpw5w1/IgpaIRZd7kOq8sd46eaR.mp4", autoplay=true, controls=true %}

Anything which can change the order of items in flexbox or grid can cause this problem. Therefore any reordering should include thorough testing to check that it will not make your site hard to use for some people.

For more information see:

- Content reordering
- Flexbox and the keyboard navigation disconnect

Writing modes and direction

Flex items lay out as a row by default. A row runs in the direction that sentences flow in your writing mode and script direction. This means that if you are working in Arabic, which has a right-to-left (rtl) script direction, the items will line up on the right. Tab order would also begin on the right as this is the way sentences are read in Arabic.

> {% Codepen { user: 'web-dot-dev', id: 'ExZgwWN' } %}

If you are working with a vertical writing mode, like some Japanese typefaces, then a row will run vertically, from top to bottom. Try changing the `flex-direction` in this demo which is using a vertical writing mode.

> {% Codepen { user: 'web-dot-dev', id: 'qBRaPXX', height: 600 } %}

Therefore the way flex items behave by default is linked to the writing mode of the document. Most tutorials are written using English, or another horizontal, left to right writing mode. This would make it easy to assume that flex items line up **on the left**, and run **horizontally**.

With the main and cross axis plus the writing mode to consider, the fact that we talk about **start** and **end** rather than top, bottom, left, and right in flexbox might be easier to understand. Each axis has a start and an end. The start of the main axis is referred to as **main-start**. So our flex items initially line up from main-start. The end of that axis is **main-end**. The start of the cross axis is **cross-start** and the end **cross-end**.

{% Img src="image/VbAJIREinuYvovrBzzvEyZOpw5w1/uSH4TxRv8KNQDTK7Vn8h.svg", alt="A labelled diagram of the above terms", width="800", height="382" %}

Wrapping flex items

The initial value of the `flex-wrap` property is `nowrap`. This means that if there is not enough space in the container the items will overflow.

> {% Img src="image/VbAJIREinuYvovrBzzvEyZOpw5w1/VTUdLS9PeBziBvbOSc4q.jpg", alt="A flex container with nine items inside it, the items have shrunk down so one word is on a line but there is not enough room to show them side by side so the flex items have extended outside the box of the container.", width="800", height="282" %}
> Once they hit min-content size flex items will start to overflow their container

Items displaying using the initial values will shrink as small as they can, down to the `min-content` size before overflow happens.

To cause the items to wrap add `flex-wrap: wrap` to the flex container.

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

{% Codepen { user: 'web-dot-dev', id: 'WNRGdNZ', height: 601 } %}

When a flex container wraps it creates multiple **flex lines**. In terms of space distribution, each line acts like a new flex container. Therefore if you are wrapping rows, it is not possible to get something in row 2 to line up with something above it in row 1. This is what is meant by flexbox being one-dimensional. You can control alignment in one axis, a row or a column, not both together as we can do in grid.

The flex-flow shorthand

You can set the `flex-direction` and `flex-wrap` properties using the shorthand `flex-flow`. For example, to set `flex-direction` to `column` and allow items to wrap:

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

Controlling space inside flex items

Assuming our container has more space than is needed to display the items, the items line up at the start and do not grow to fill the space. They stop growing at their max-content size. This is because the initial value of the `flex-` properties is:

- `flex-grow: 0`: items do not grow.
- `flex-shrink: 1`: items can shrink smaller than their `flex-basis`.
- `flex-basis: auto`: items have a base size of `auto`.

This can be represented by a keyword value of `flex: initial`. The `flex` shorthand property, or the longhands of `flex-grow`, `flex-shrink` and `flex-basis` are applied to the children of the flex container.

{% Codepen { user: 'web-dot-dev', id: 'LYxRebE' } %}

To cause the items to grow, while allowing large items to have more space than small ones use `flex:auto`. You can try this using the demo above. This sets the properties to:

- `flex-grow: 1`: items can grow larger than their `flex-basis`.
- `flex-shrink: 1`: items can shrink smaller than their `flex-basis`.
- `flex-basis: auto`: items have a base size of `auto`.

Using `flex: auto` will mean that items end up different sizes, as the space that is shared between the items is shared out *after* each item is laid out as max-content size. So a large item will gain more space. To force all of the items to be a consistent size and ignore the size of the content change `flex:auto` to `flex: 1` in the demo.

This unpacks to:

- `flex-grow: 1`: items can grow larger than their `flex-basis`.
- `flex-shrink: 1`: items can shrink smaller than their `flex-basis`.
- `flex-basis: 0`: items have a base size of `0`.

Using `flex: 1` says that all items have zero size, therefore all of the space in the flex container is available to be distributed. As all items have a `flex-grow` factor of `1` they all grow equally and the space is shared equally.

{% Aside %} There is also a value of `flex: none`, which will give you inflexible flex items that do not grow or shrink. This might be useful if you are purely using flexbox to access the alignment properties but don't want any flexible behavior. {% endAside %}

Allowing items to grow at different rates

You don't have to give all items a `flex-grow` factor of `1`. You could give your flex items different `flex-grow` factors. In the demo below the first item has `flex: 1`, the second `flex: 2` and third `flex: 3`. As these items grow from `0` the available space in the flex container is shared into six. One part is given to the first item, two parts to the second, three parts to the third.

{% Codepen { user: 'web-dot-dev', id: 'OJWRzEz' } %}

You can do the same thing from a `flex-basis` of `auto`, though you will need to specify the three values. The first value being `flex-grow`, the second `flex-shrink`, and the third `flex-basis`.

```
.item1 {
  flex: 1 1 auto;
}

.item2 {
  flex: 2 1 auto;
}
```

This is a less common use case as the reason to use a `flex-basis` of `auto` is to allow the browser to figure out space distribution. If you wanted to cause an item to grow a little more than the algorithm decides however it might be useful.

Reordering flex items

Items in your flex container can be reordered using the `order` property. This property allows the ordering of items in **ordinal groups**. Items are laid out in the direction dictated by `flex-direction`, lowest values first. If more than one item has the same value it will be displayed with the other items with that value.

The example below demonstrates this ordering.

{% Codepen { user: 'web-dot-dev', id: 'NWdRXoL' } %}

{% Aside 'warning' %} Using `order` has the same problems as the `row-reverse` and `column-reverse` values of `flex-direction`. It would be very easy to create a disconnected experience for some users. Do not use `order` because you are fixing things being out of order in the document. If the items logically should be in a different order, change your HTML! {% endAside %}

{% Assessment 'flex' %}

Flexbox alignment overview

Flexbox brought with it a set of properties for aligning items and distributing space between items. These properties were so useful they have since been moved into their own specification, you'll encounter them in Grid Layout too. Here you can find out how they work when you are using flexbox.

The set of properties can be placed into two groups. Properties for space distribution, and properties for alignment. The properties which distribute space are:

- `justify-content`: space distribution on the main axis.
- `align-content`: space distribution on the cross axis.
- `place-content`: a shorthand for setting both of the above properties.

The properties used for alignment in flexbox:

- `align-self`: aligns a single item on the cross axis.
- `align-items`: aligns all of the items as a group on the cross axis.

If you are working on the main axis then the properties begin with `justify-`. On the cross axis they begin with `align-`.

Distributing space on the main axis

With the HTML used earlier, the flex items laid out as a row, there is space on the main axis. The items are not big enough to completely fill the flex container. The items line up at the start of the flex container because the initial value of `justify-content` is `flex-start`. The items line up at the start and any extra space is at the end.

Add the `justify-content` property to the flex container, give it a value of `flex-end`, and the items line up at the end of the container and the spare space is placed at the start.

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

You can also distribute the space between the items with `justify-content: space-between`.

Try some of the values in the demo, and see MDN for the full set of possible values.

{% Codepen { user: 'web-dot-dev', id: 'JjERpGb' } %}

{% Aside %} For the `justify-content` property to do anything you have to have spare space in your container on the main axis. If your items fill the axis then there is no space to share out so the property won't do anything. {% endAside %}

With `flex-direction: column`

If you have changed your `flex-direction` to `column` then `justify-content` will work on the column. To have spare space in your container when working as a column you need to give your container a `height` or `block-size`. Otherwise you won't have spare space to distribute.

Try the different values, this time with a flexbox column layout.

{% Codepen { user: 'web-dot-dev', id: 'bGgwLgz', height: 600 } %}

Distributing space between flex lines

With a wrapped flex container you might have space to distribute on the cross axis. In this case you can use the `align-content` property with the same values as `justify-content`. Unlike `justify-content` which aligns items to `flex-start` by default, the initial value of `align-content` is `stretch`. Add the property `align-content` to the flex container to change that default behavior.

```
.container {
  align-content: center;
}
```

Try this out in the demo. The example has wrapped lines of flex items, and the container has a `block-size` in order that we have some spare space.

{% Codepen { user: 'web-dot-dev', id: 'poREawo' } %}

The `place-content` shorthand

To set both `justify-content` and `align-content` you can use `place-content` with one or two values. A single value will be used for both axes, if you specify both the first is used for `align-content` and the second for `justify-content`.

```
.container {
  place-content: space-between;
  /* sets both to space-between */
}

.container {
  place-content: center flex-end;
  /* wrapped lines on the cross axis are centered,
  on the main axis items are aligned to the end of the flex container */
}
```

Aligning items on the cross-axis

On the cross axis you can also align your items within the flex line using `align-items` and `align-self`. The space available for this alignment will depend on the height of the flex container, or flex line in the case of a wrapped set of items.

The initial value of `align-self` is `stretch`, which is why flex items in a row stretch to the height of the tallest item by default. To change this, add the `align-self` property to any of your flex items.

```
.container {
  display: flex;
}

.item1 {
  align-self: flex-start;
}
```

Use any of the following values to align the item:

- `flex-start`
- `flex-end`
- `center`
- `stretch`
- `baseline`

See the full list of values on MDN.

The next demo has a single line of flex items with `flex-direction: row`. The last item defines the height of the flex container. The first item has the `align-self` property with a value of `flex-start`. Try changing the value on that property to see how it moves within its space on the cross axis.

{% Codepen { user: 'web-dot-dev', id: 'RwKGQee', height: 600 } %}

The `align-self` property is applied to individual items. The `align-items` property can be applied to the flex container to set all of the individual `align-self` properties as a group.

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

In this next demo try changing the value of `align-items` to align all of the items on the cross axis as a group.

{% Codepen { user: 'web-dot-dev', id: 'QWdKmby', height: 600 } %}

Why is there no justify-self in flexbox?

Flex items act as a group on the main axis. So there is no concept of splitting an individual item out of that group.

In grid layout the `justify-self` and `justify-items` properties work on the inline axis to do alignment of items on that axis within their grid area. Due to the way that flex layouts treat items as a group, these properties are not implemented in a flex context.

It is worth knowing that flexbox does work very nicely with auto margins. If you come across a need to split off one item from a group, or separate the group into two groups you can apply a margin to do this. In the example below the last item has a left margin of `auto`. The auto margin absorbs all space in the direction it is applied. This means that it pushes the item over to the right, thus splitting the groups.

{% Codepen { user: 'web-dot-dev', id: 'poRELbR' } %}

How to center an item vertically and horizontally

The alignment properties can be used to center an item inside another box. The `justify-content` property aligns the item on the main axis, which is row. The `align-items` property on the cross axis.

```
.container {
  width: 400px;
  height: 300px;
  display: flex;
  justify-content: center;
  align-items: center;
}
```

{% Aside %} In the future we may be able to do this alignment without needing to make the parent a flex container. The alignment properties are specified for block and inline layout. At present no browser has implemented these. However, switching into a flex formatting context gives you access to the properties. If you need to align something it's a great way to do it. {%

endAside %}

{% Assessment 'conclusion' %}

Resources

- MDN CSS Flexible Box Layout includes a series of detailed guides with examples.
- CSS Tricks Guide to Flexbox
- What Happens When You Create a Flexbox Flex Container
- Everything You Need To Know About Alignment In Flexbox
- How Big Is That Flexible Box?
- Use Cases For Flexbox
- Inspect and debug CSS Flexbox layouts in Chrome DevTools

□□□ #2
□ □□□ □□□ 2026-10-31 22:45:34 UTC
□ □□□ □□□ 2026-10-31 22:45:36 UTC