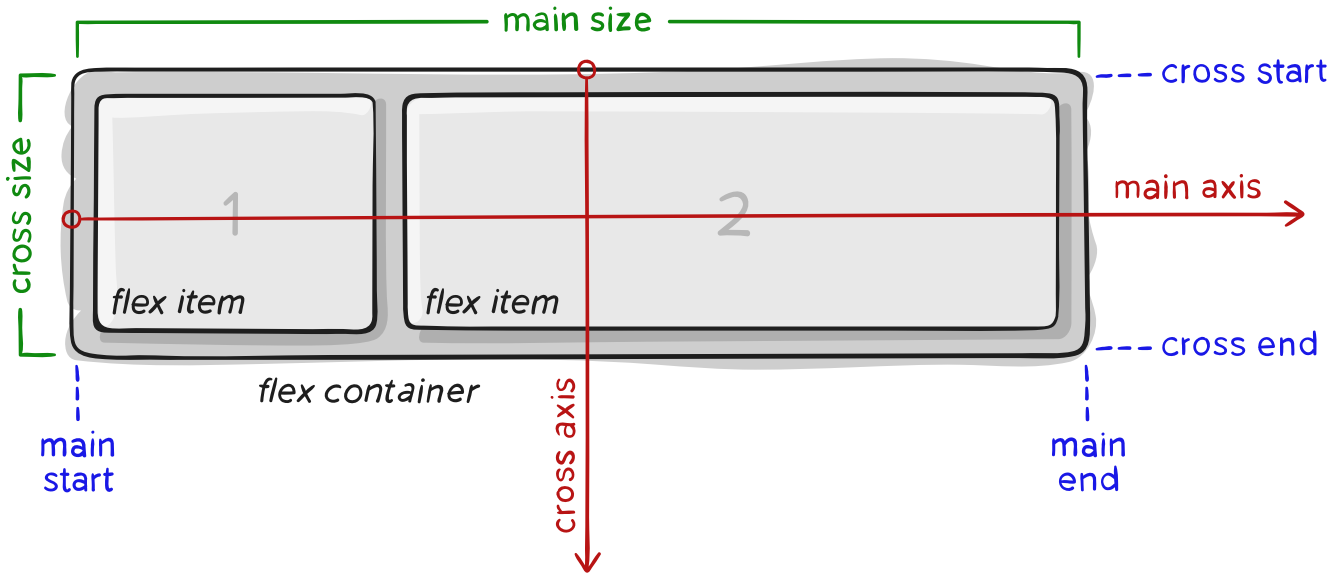
Background

The main idea behind the flex layout is to give the container the ability to alter its' items width/height (and order) to best fill the available space (mostly to accommodate all kinds of display devices and screen sizes).

The flexbox layout is direction-agnostic as opposed to the regular layouts (block which is vertically-based and inline which is horizontally-based).

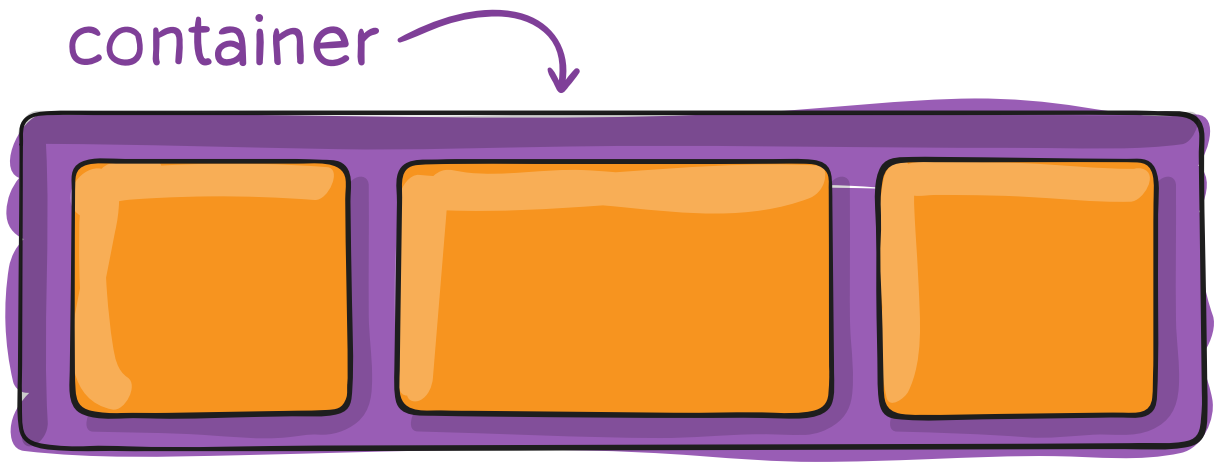
Flexbox layout is most appropriate to the components of an application, and small-scale layouts, while the Grid layout is intended for larger scale layouts.

Basics and terminology



Flexbox properties

Properties for the Parent (flex container)

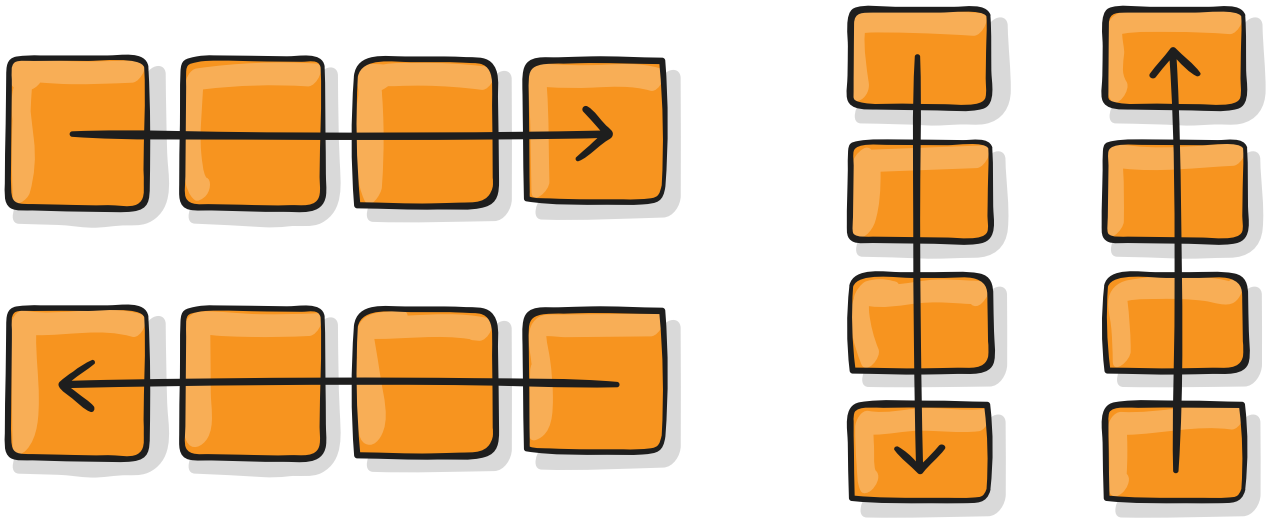


display

This defines a flex container; inline or block depending on the given value.It enables a flex context for all its direct children.

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

flex-direction

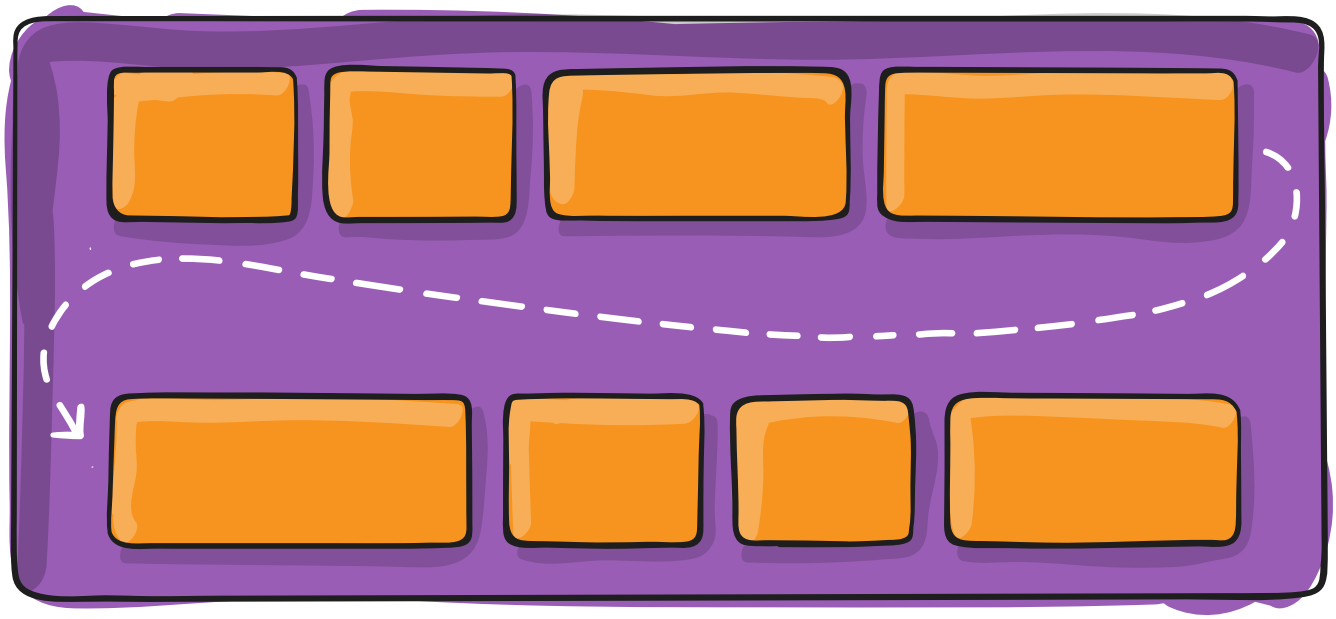


This establishes the main-axis. Flexbox is (aside from optional wrapping) a single-direction layout concept.

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

Has reverse options.

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 mulitple lines, from top to bottom
- `wrap-reverse`: flex items will wrap onto multiple lines from bottom to top

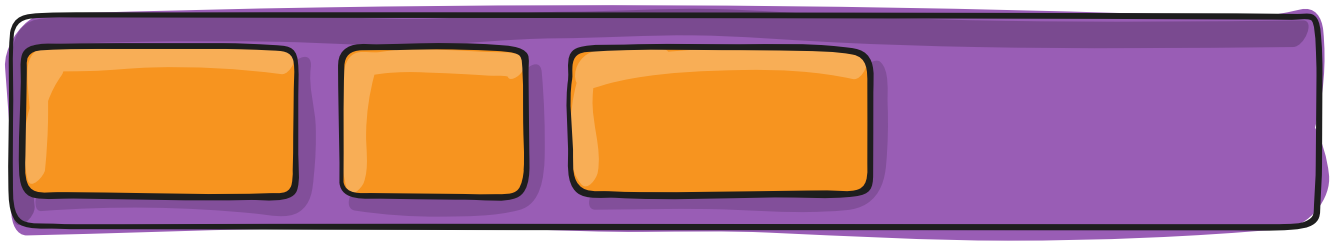
flex-flow

This is a shorthand for the `flex-direction` and `flex-wrap` properties. The default valu is `row nowrap`.

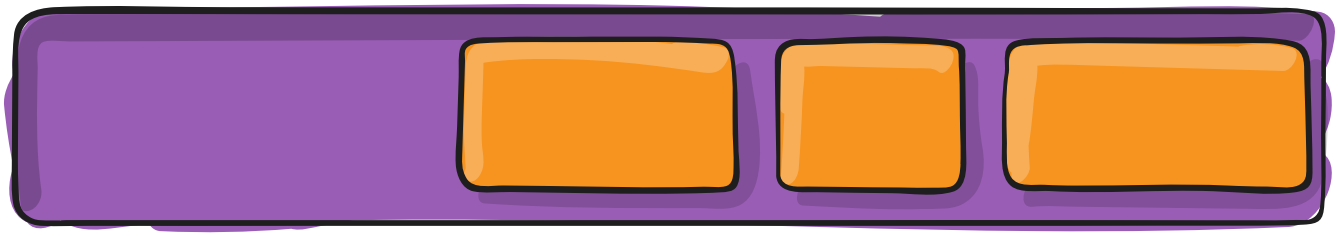
```css
.container {
  flex-flow: column wrap;
}
```

justify-content

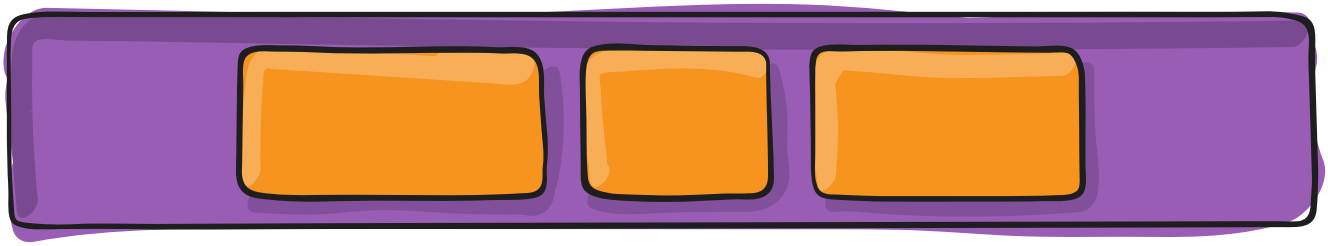
flex-start



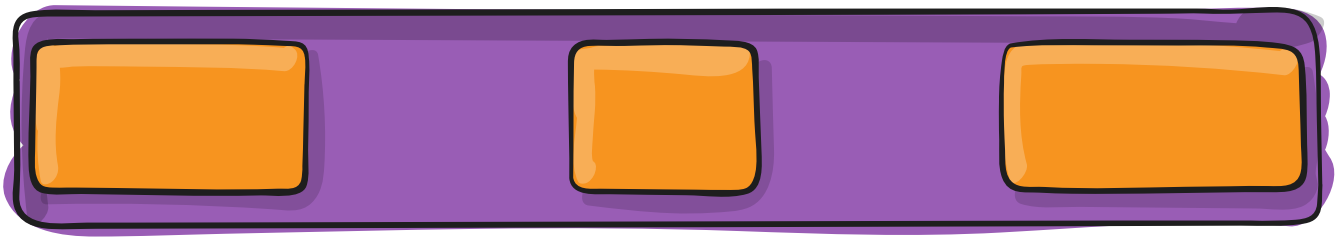
flex-end



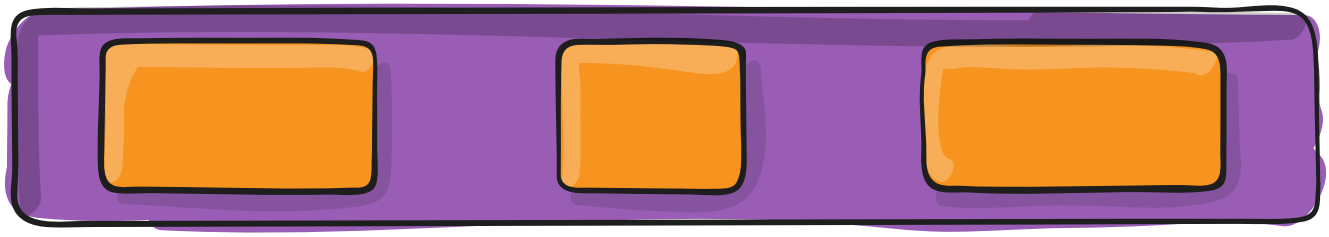
center



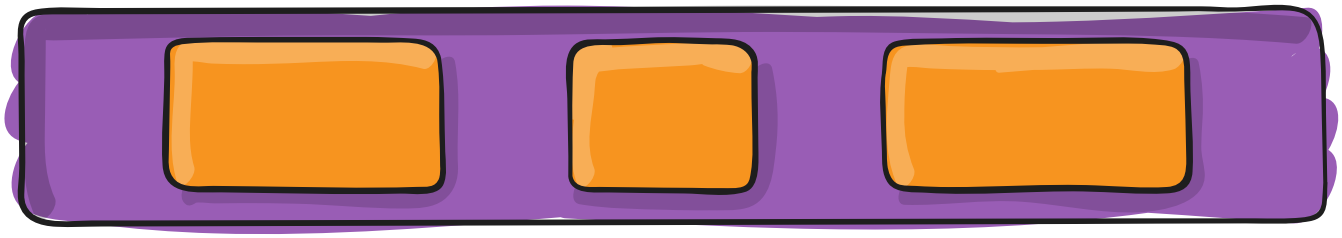
space-between



space-around



space-evenly



This defines alignment along the main axis. It helps distribute extra free space leftover when either all the flex items on a line are inflexible, or ar 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 | spac
}
```

```
| center | space-between | space-around | space-evenly | st
```

```
evenly | start | end | left | right ... + safe | unsafe;
```

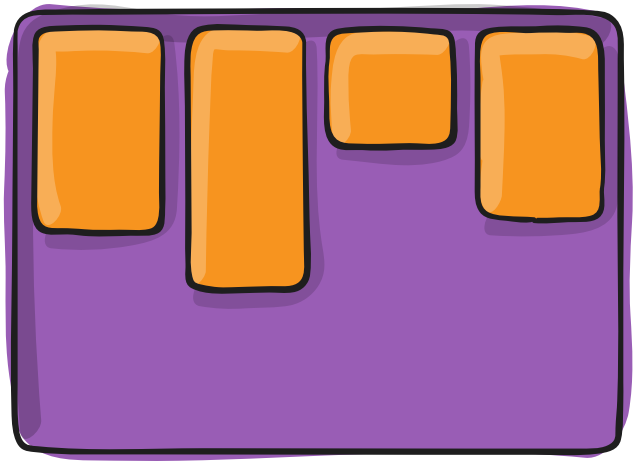
- `flex-start` (default): items are packed toward the start of the flex-direction.
- `start`: items are packed toward the start of the `writing-mode` direction
- `left`: items are packed toward the left edge of the container, unless that doesn't make sense with the `flex-direction`, then it behaves like `start`.
- `space-between`: items are evenly distributed in the line; first item is on the start line, last item on the end line
- `space-around`: items are evenly distributed in the line with equal space around them. Note that visually the spaces aren't equal, sincec all the items have equal space on both sides. The first item will have one unit of space against the container edge, but two units of space between the next item because that next item has its own spacing that applies

- `space-evenly`: items are distributed so that the spacing between any two items (and the space to the edges) is equal

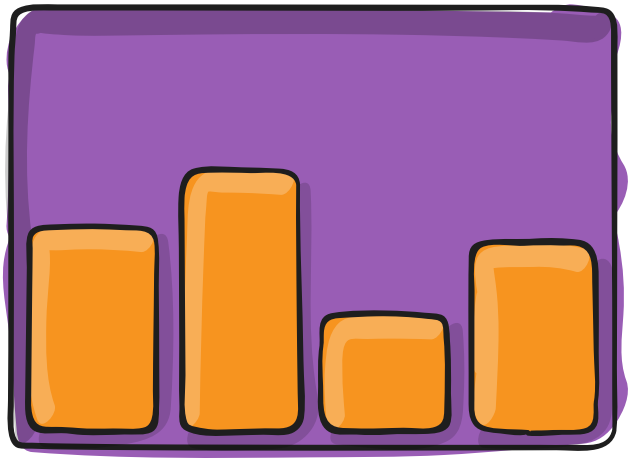
There are also two additional keywords you can pair with these values: `safe` and `unsafe`. Using `safe` ensures that however you do this type of positioning, you can't push an element such that it renders off-screen in such a way that the content can't be scrolled to ("data loss")

align-items

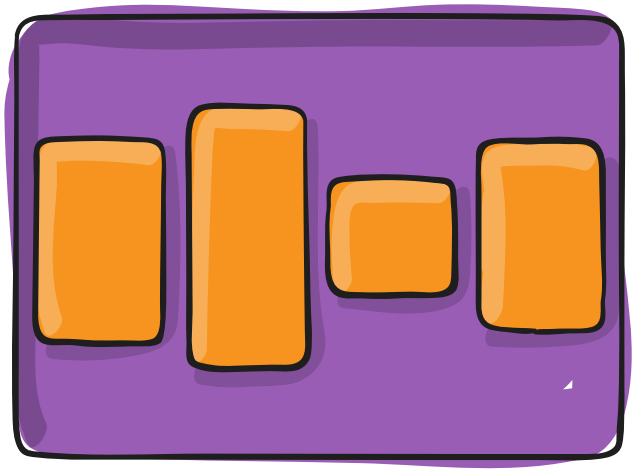
flex-start



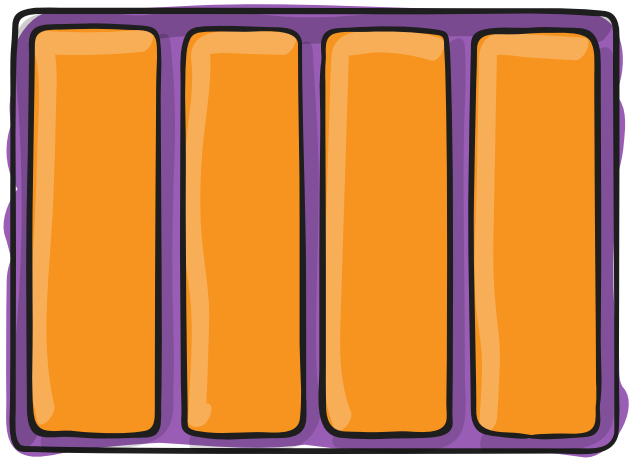
flex-end



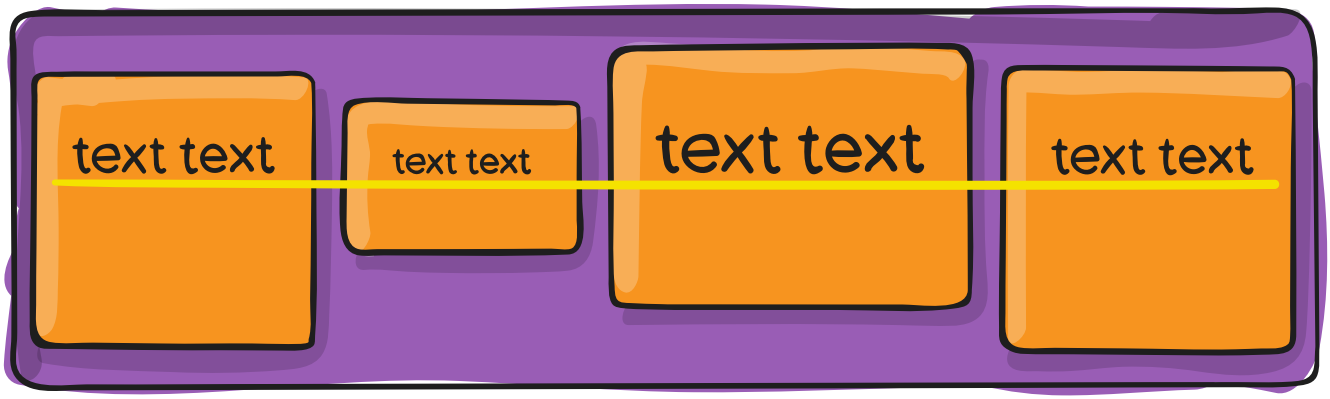
center



stretch

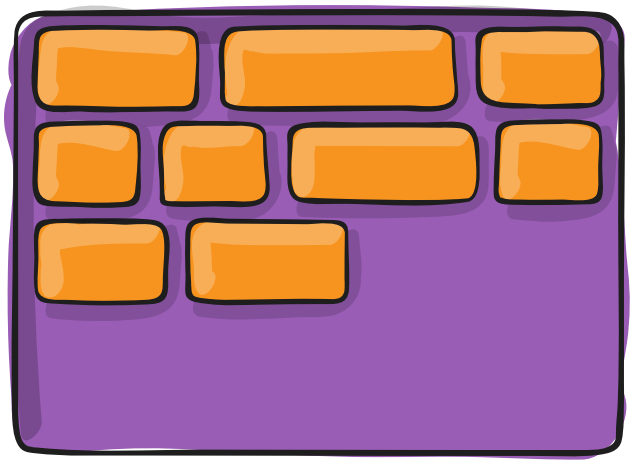


baseline

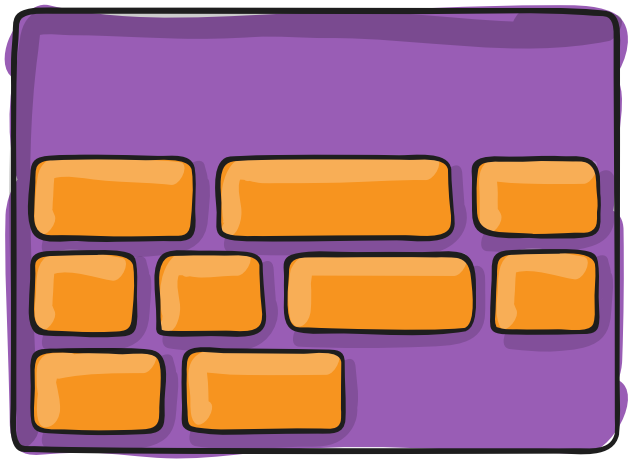


align-content

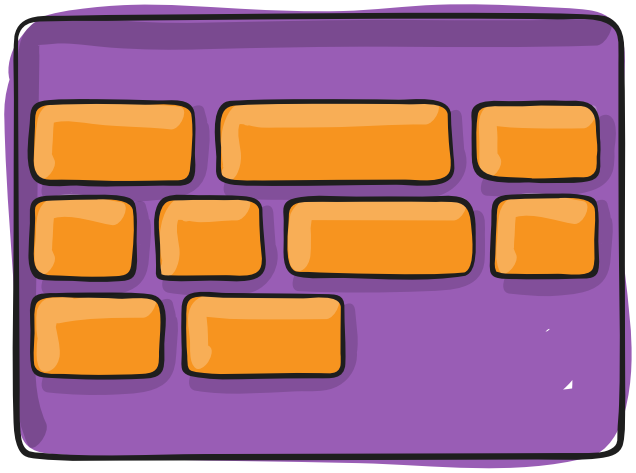
flex-start



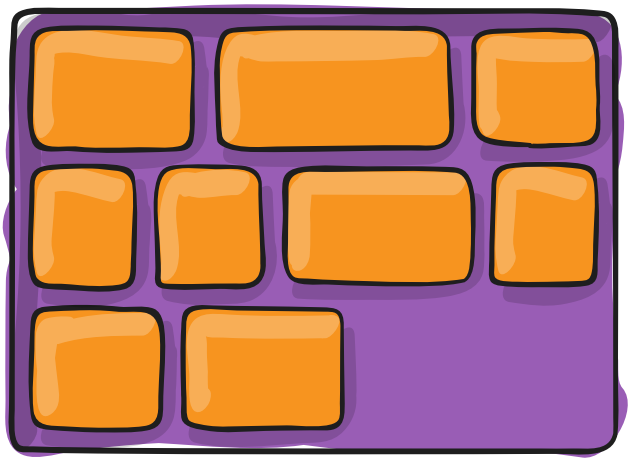
flex-end



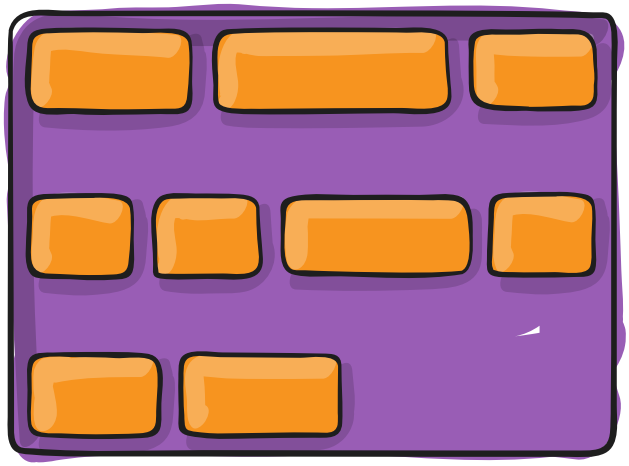
center



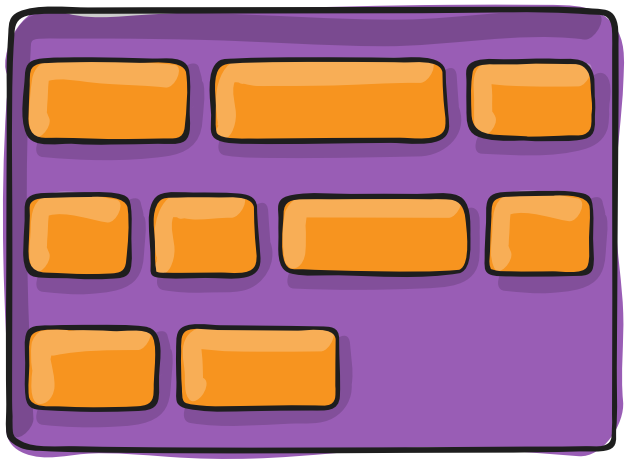
stretch



space-between



space-around

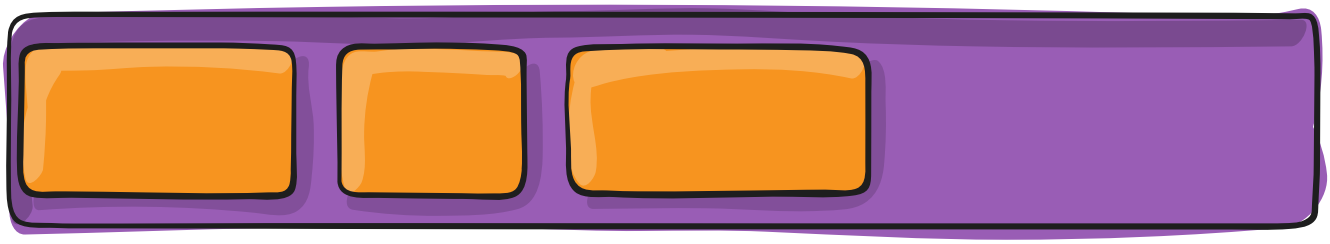


This aligns a flex container's lines when there is extra space in the cross-axis, similar to how justify-content aligns individual items within the main-axis.

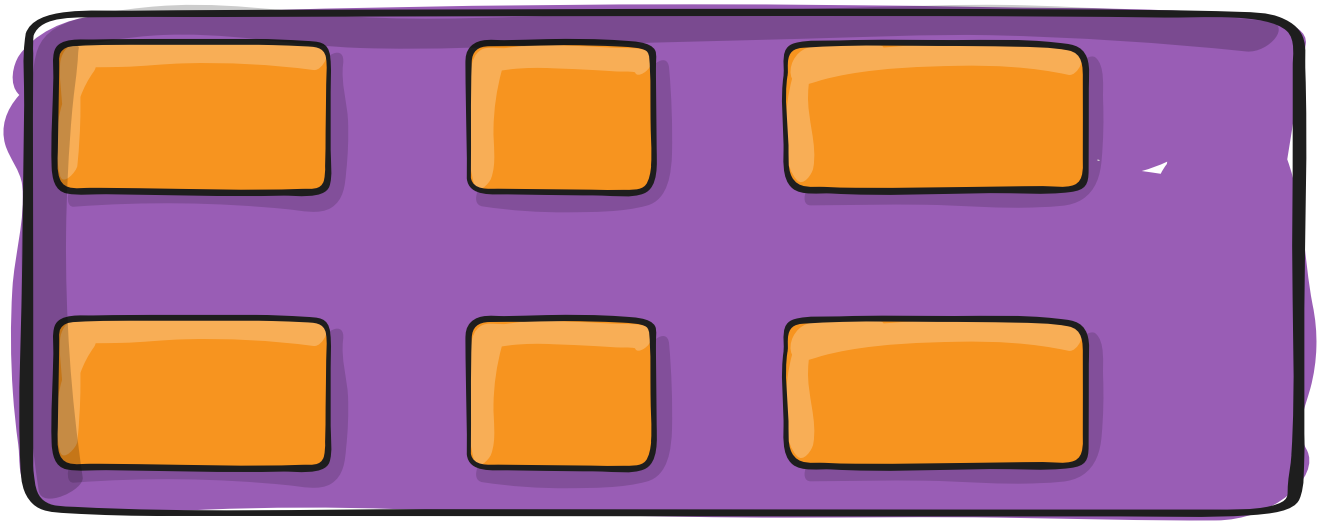
This property only takes effect on multi-line flexible containers, where `flex-wrap` is set to either `wrap` or `wrap-reverse`). A siongle-line flexible container (i.e. where `flex-wrap` is set to its default value, `no-wrap`) will not reflect align-content.

gap, row-gap, column-gap

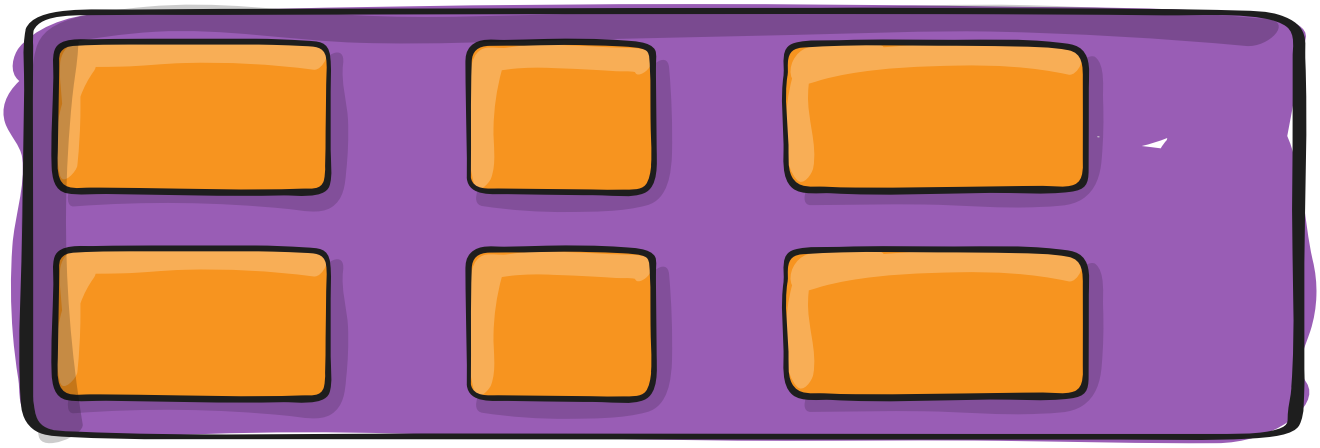
gap: 10px



gap: 30px



gap: 10px 30px



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

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

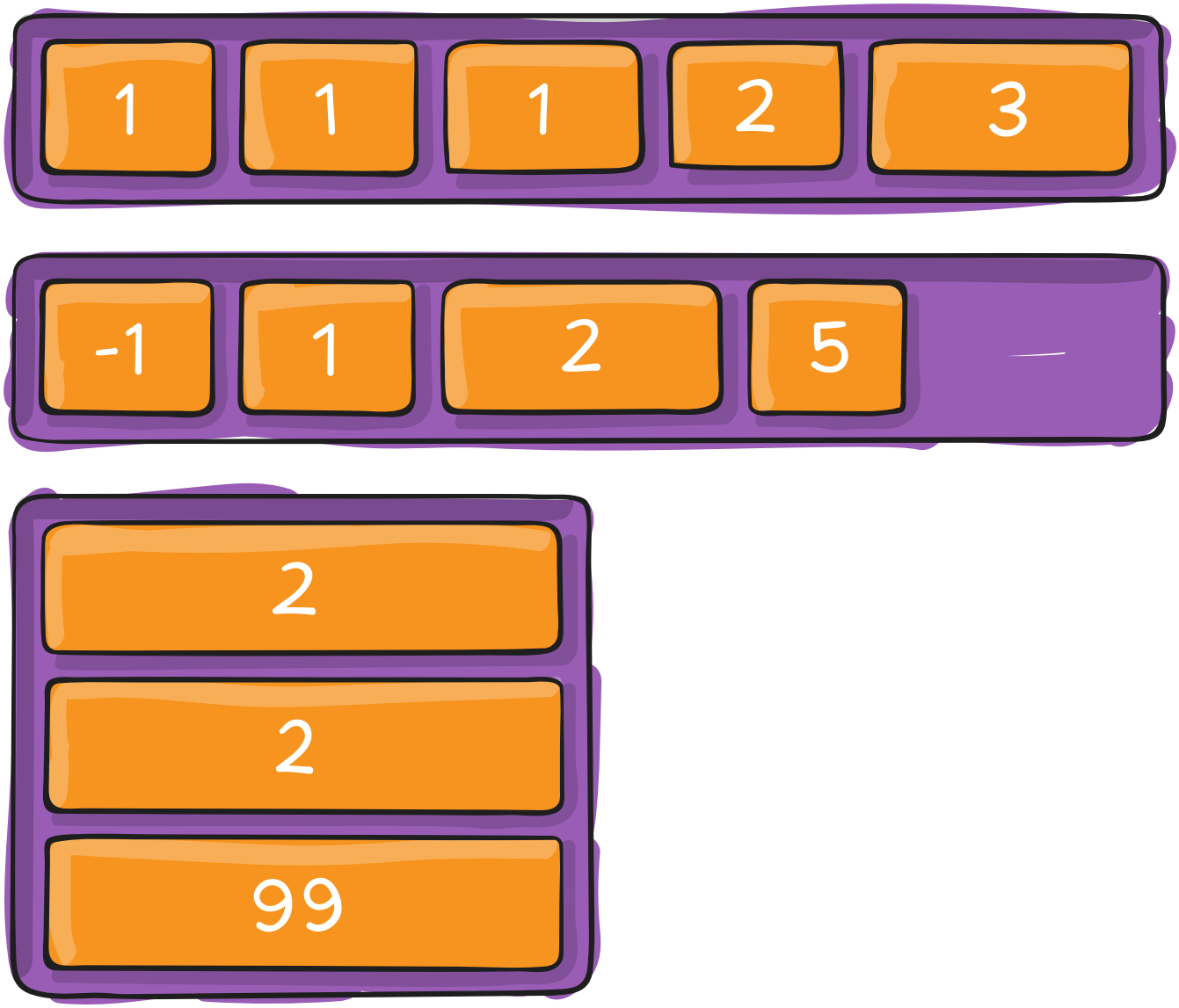
The gap property explicitly controls the space between flex items. It applies that spacing *only between items* not on the outer edges.

If there is 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 mutli-column layout as well.

Properties for the Children (flex items)

order

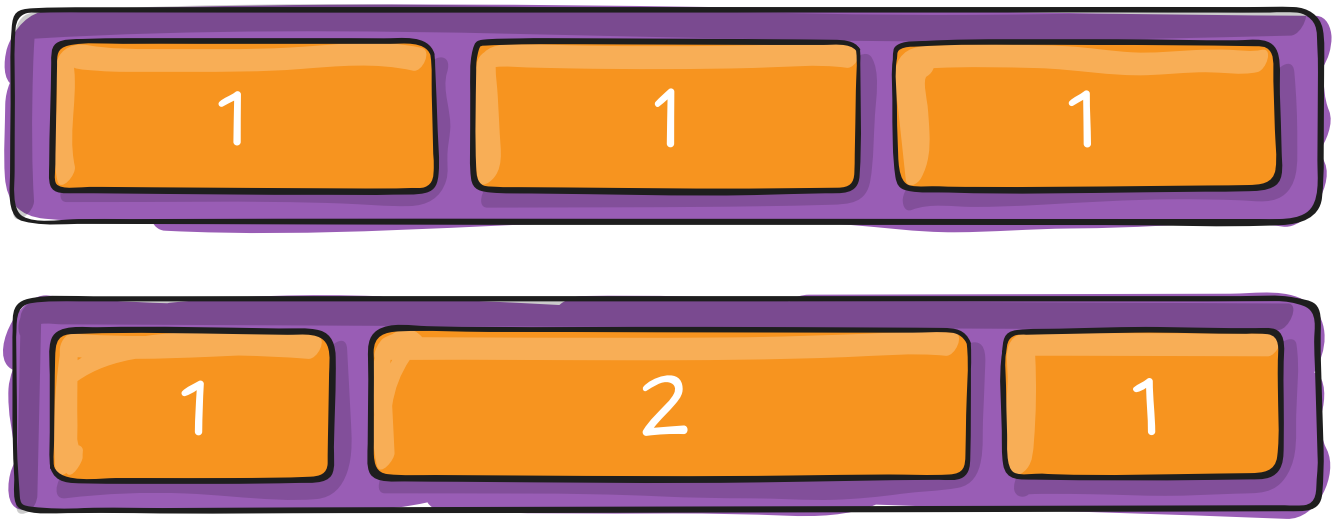


By default, flex items are laid out in source order. However, the order property controls the order in which they appear in the flex container.

```
.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.

If one of the children has a value double the rest, that child would take up twice as much of the space as any of the others (or it will try, at least).

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

flex-shrink

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

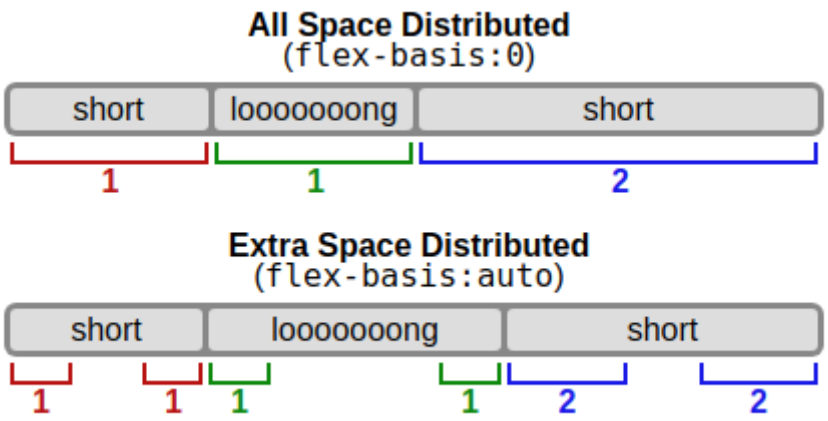
flex-basis

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".

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

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".

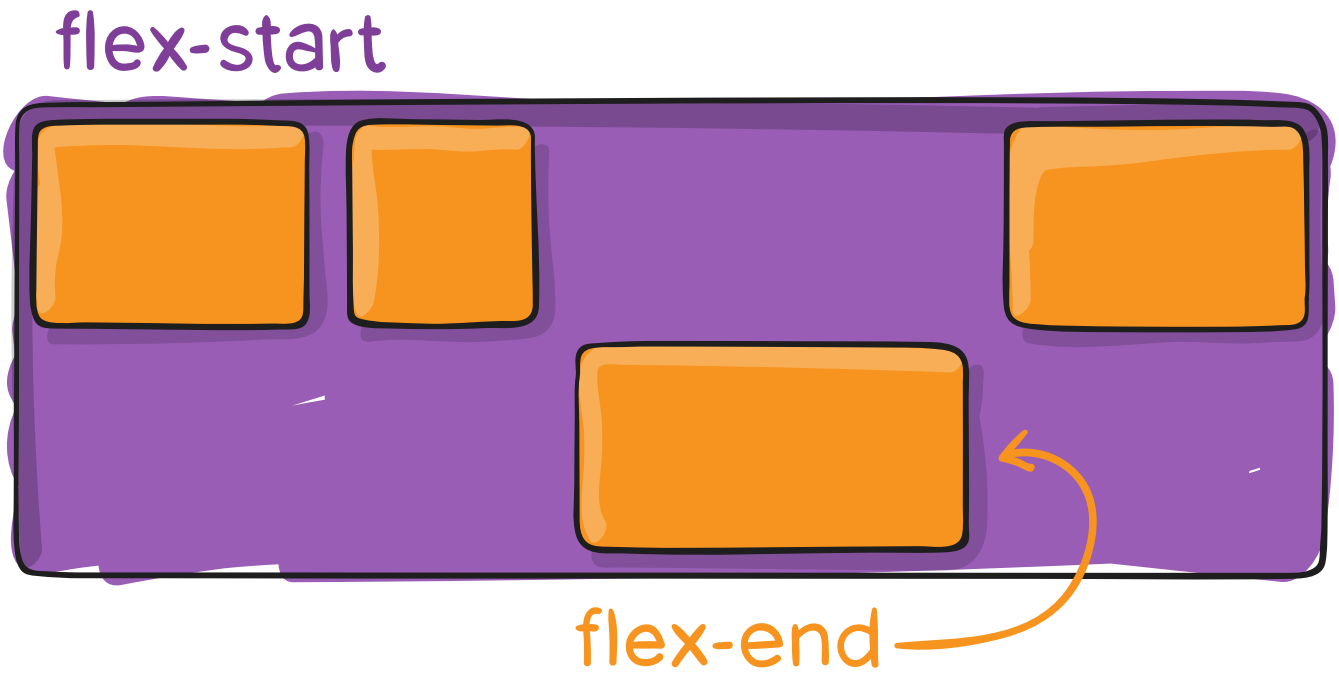
See this graphic:



flex

It is recommended that you use this shorthand property rather than set the individual properties; the shorthand sets the other values usually to what you want.

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 | ba
}
```

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

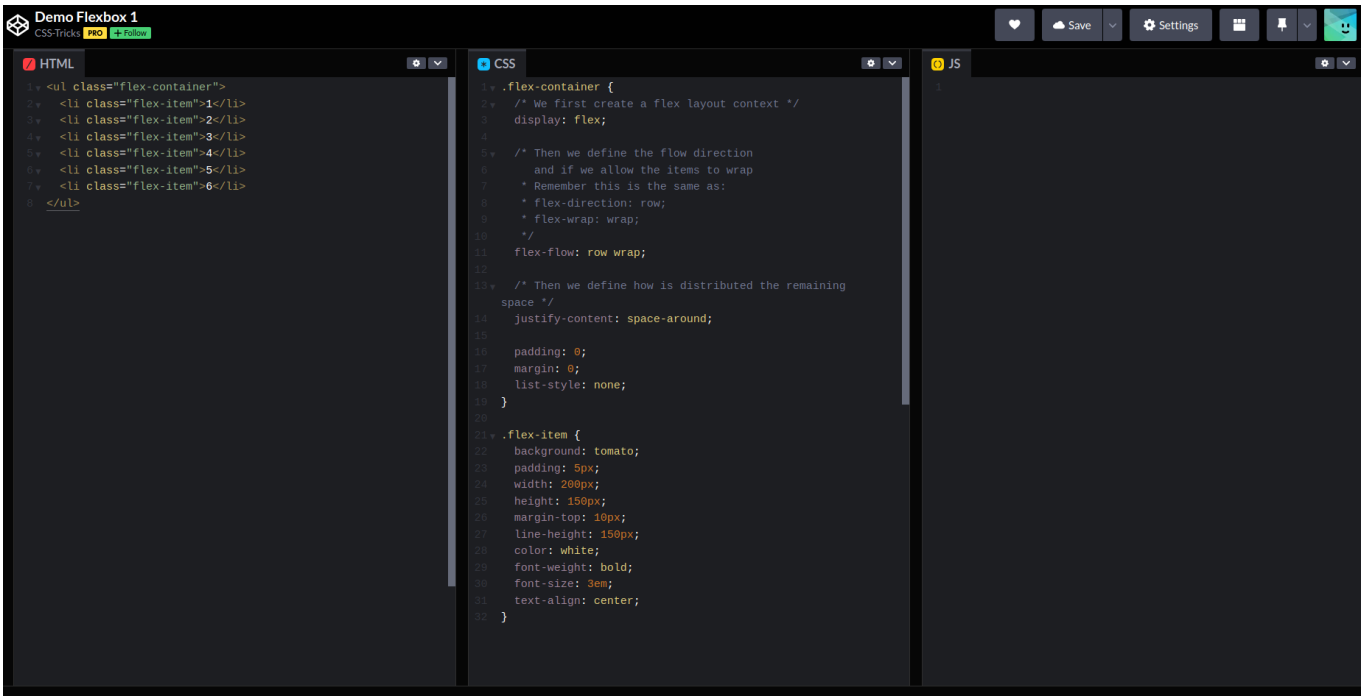
Examples

Perfect centering:

```
.parent {
  display: flex;
  height: 300px; /* Or whatever */
}

.child {
  width: 100px;  /* Or whatever */
  height: 100px; /* Or whatever */
  margin: auto;  /* Magic! */
}
```

Nice distribution of items:



Demo Flexbox 1

CSS-Tricks PRO + Follow

Save Settings

HTML

```html
<ul class="flex-container">
  <li class="flex-item">1</li>
  <li class="flex-item">2</li>
  <li class="flex-item">3</li>
  <li class="flex-item">4</li>
  <li class="flex-item">5</li>
  <li class="flex-item">6</li>
</ul>
```

CSS

```css
.flex-container {
  /* We first create a flex layout context */
  display: flex;

  /* Then we define the flow direction
     and if we allow the items to wrap
   * Remember this is the same as:
   * flex-direction: row;
   * flex-wrap: wrap;
   */
  flex-flow: row wrap;
```

JS

1 2 3 4 5 6