Learn Flexbox

@April 7, 2023

Written by Ramzi @slayingthedragon

YouTube link - https://youtu.be/phWxA89Dy94

Codepen link - https://codepen.io/ramzibach-the-styleful/pen/GRXLZEq?editors=1100

https://youtu.be/phWxA89Dy94

During the CSS stone-age developers were creating layouts with floats and positioning until one fateful day Flexbox would be introduced and the world would never be the same.

Flexbox

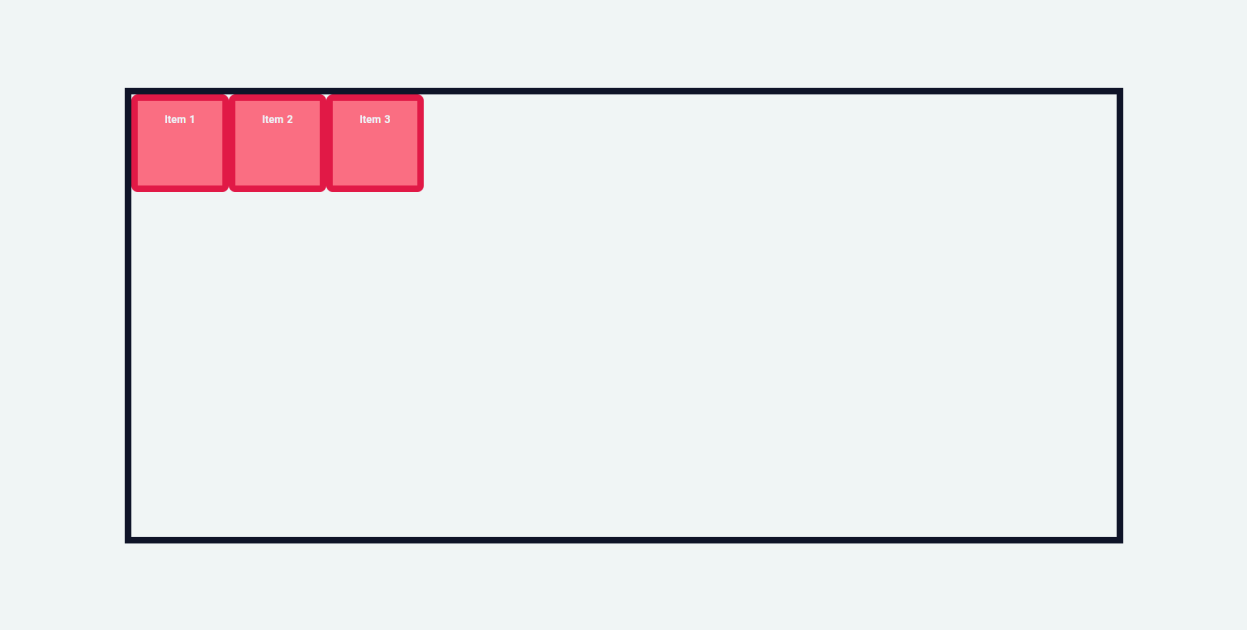
To use flexbox - we first need a container and some children in our HTML.

```
<div class="container">
  <div class="item-1">Item 1</div>
  <div class="item-2">Item 2</div>
```

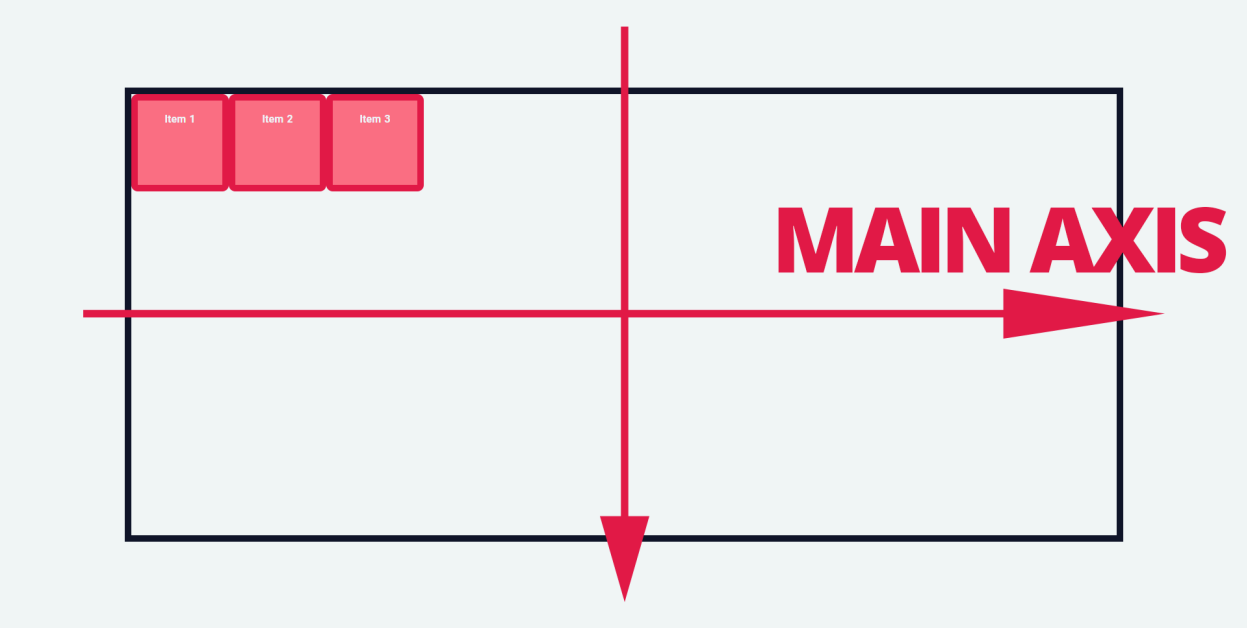
```
    <div class="item-3">Item 3</div>
</div>
```

Then, in our CSS, we can give the container the display of flex.

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



This created 2 invisible axes, a main axis and a cross axis.



Our items are no longer stacked on top of one another because display flex is positioning our items on the main axis and by default the main axis is horizontal.

flex-direction

If we want - we can change the main axis to be vertical with the flex-direction property.

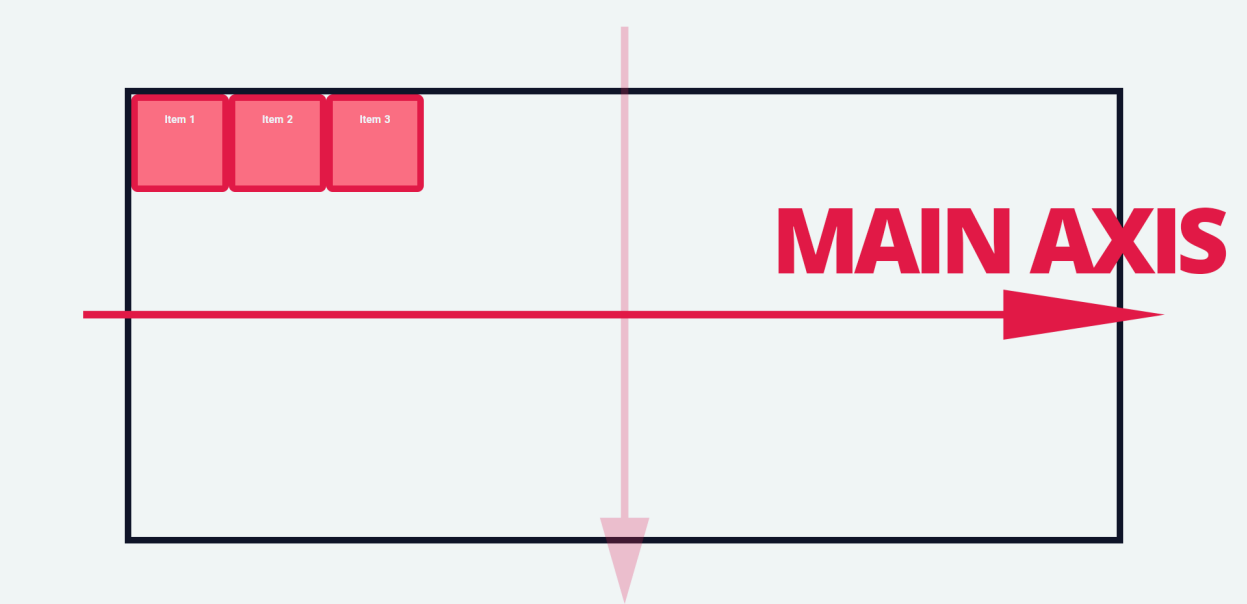
```
.container {
   display: flex;
   flex-direction: ;
}
```

If we set the flex-direction to column - the main axis will become vertical.

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

If we set it to row the main axis will be horizontal.

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



We actually want our main axis to be horizontal so we set flex-direction to row **or we could remove it** entirely because **flex-direction is set to row by default**.

```
.container {
  display: flex;
  /* flex-direction: column; */
}
```

justify-content

Now that we know the direction of our main axis, we can align our items along the main axis with the justify-content property.

```
.container {
    display: flex;
    /* flex-direction: column; */
    justify-content: ;
}
```

The default is **flex-start**, see nothing happens.

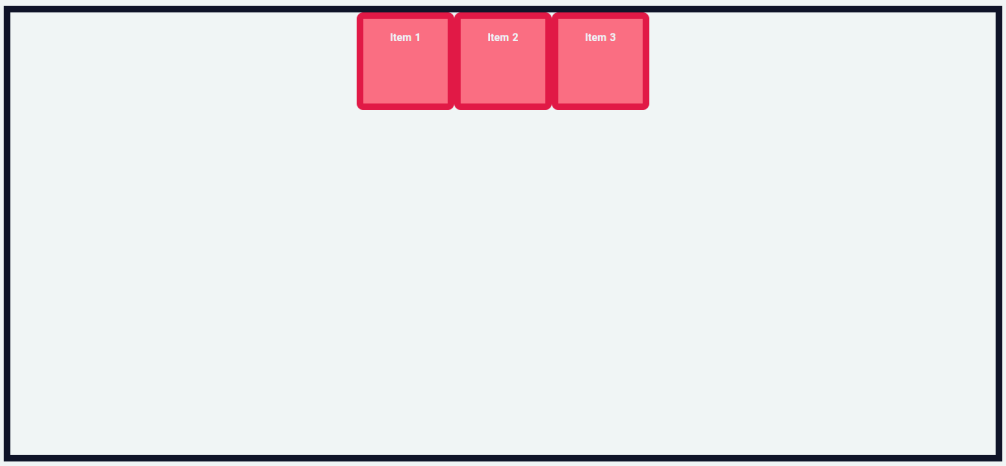
```
.container {
    display: flex;
    /* flex-direction: column; */
    justify-content: flex-start;
}
```

But if we set it to **flex-end -** our items will be pushed to the end of the main axis.

```
.container {
  display: flex;
  /* flex-direction: column; */
  justify-content: flex-end;
}
```

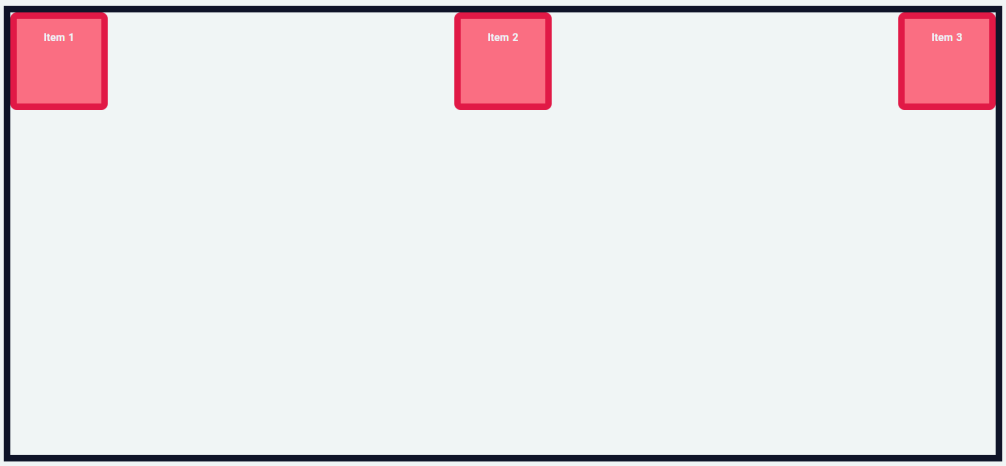
center will center our items in the middle of the main axis.

```
.container {
  display: flex;
  /* flex-direction: column; */
  justify-content: center;
}
```



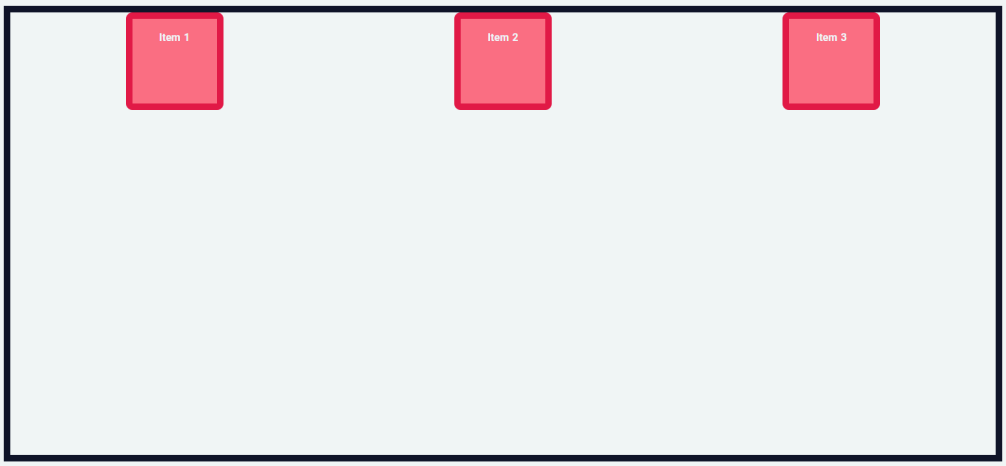
space-between will evenly distribute the items in between the first and last item.

```
.container {
    display: flex;
    /* flex-direction: column; */
    justify-content: space-between;
}
```



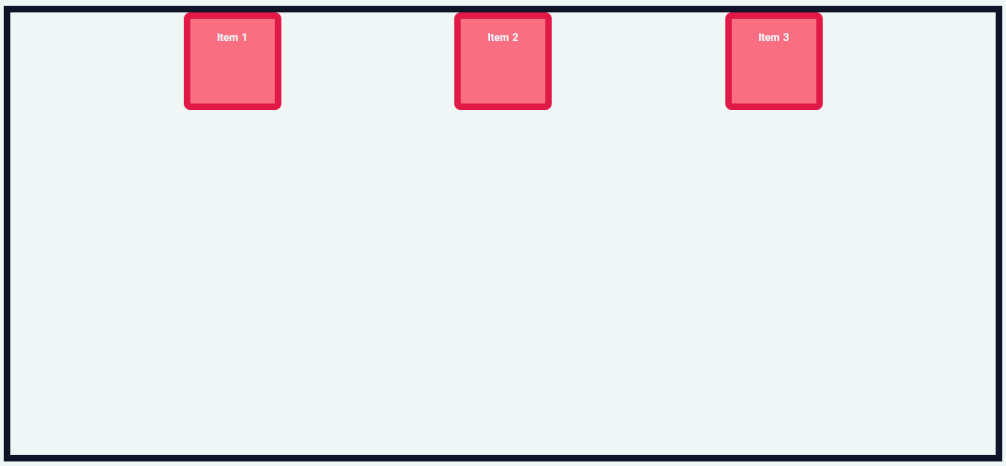
space-around is similar to space-between but now the edges will also have some spacing to them.

```
.container {
    display: flex;
    /* flex-direction: column; */
    justify-content: space-around;
}
```



And finally, **space-evenly** will evenly distribute space in between all the items.

```
.container {
    display: flex;
    /* flex-direction: column; */
    justify-content: space-evenly;
}
```



align-items

Justify-content is for aligning items on the main axis, **but we can also align items along the cross axis with the align-items property.**

```
.container {
    display: flex;
    /* flex-direction: column; */
    /* justify-content: space-evenly; */
    align-items: ;
}
```

flex-start places items at the start of the cross axis.

```
.container {
    display: flex;
    /* flex-direction: column; */
    /* justify-content: space-evenly; */
```

```
    align-items: flex-start;
}
```

flex-end places items at the end of the cross axis.

```
.container {
    display: flex;
    /* flex-direction: column; */
    /* justify-content: space-evenly; */
    align-items: flex-end;
}
```

center will center our items in the middle of the cross axis.

```
.container {
    display: flex;
    /* flex-direction: column; */
    /* justify-content: space-evenly; */
    align-items: center;
}
```

And **baseline** will align items so that the baseline text of each item is aligned.

You can see this more clearly when I increase the font-size on item-1.

The font-size on the first item is larger than the second and third but with align-items set to baseline the baseline of all 3 texts are aligned.

```
.container {
    display: flex;
    /* flex-direction: column; */
    /* justify-content: space-evenly; */
    align-items: baseline;
}

.item-1 {
    font-size: 1.5rem;
}
```

JUSTIFY-CONTENT WILL ALIGN ITEMS ON THE MAIN AXIS.

ALIGN-ITEMS WILL ALIGN ITEMS ON THE CROSS AXIS.

When the flex-direction is set to column:

```
.container {
   display: flex;
   flex-direction: column;
   /* justify-content: space-evenly; */
   /*  align-items: baseline; */
}
```

justify-content will still align items on the main axis but now that we defined the flex-direction to be column, the main axis is now vertical. This is the key to understanding flexbox.

```
.container {
   display: flex;
   flex-direction: column;
```

```
  justify-content: space-evenly;
    /*  align-items: baseline; */
}
```

flex-wrap

By default, flex items will all try to fit into one line.

When I add more items to the container the items will start crushing each-other..

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

```
  <div class="item item-3">Item 3</div>
  <div class="item item-1">Item 1</div>
  <div class="item item-2">Item 2</div>
  <div class="item item-3">Item 3</div>
  <div class="item item-1">Item 1</div>
  <div class="item item-2">Item 2</div>
  <div class="item item-3">Item 3</div>
</div>
```

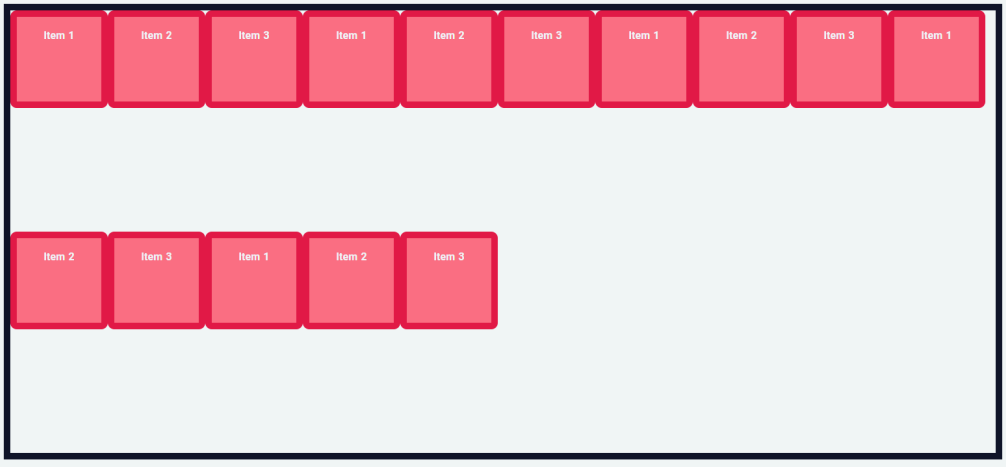
```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
}
```

To fix this we can make our items wrap with the **flex-wrap property**.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
    flex-wrap: ;
}
```

By default this is set to nowrap, but we can set it to **wrap** and now our items are no longer trying to fit into one line but instead are allowed to wrap when there's no more space available.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
    flex-wrap: wrap;
}
```



align-content

When we have flex-wrap set to wrap, a new property is unlocked, the align-content property, not to be confused with align-items, the align-content property only works when we have flex-wrap set to wrap and have items wrapping.

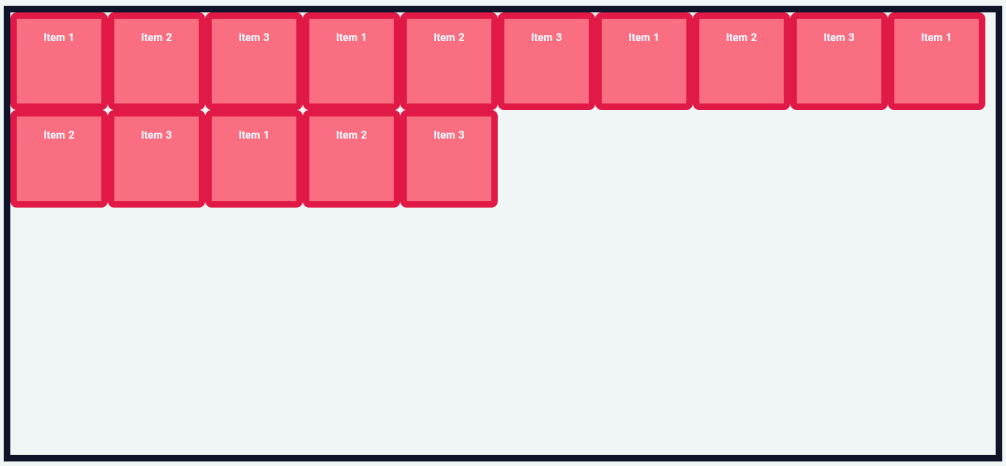
This property allows us to align everything on the cross-axis.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
    flex-wrap: wrap;
    align-content: ;
}
```

The possible values that go here are pretty much the same as with the justify-content.

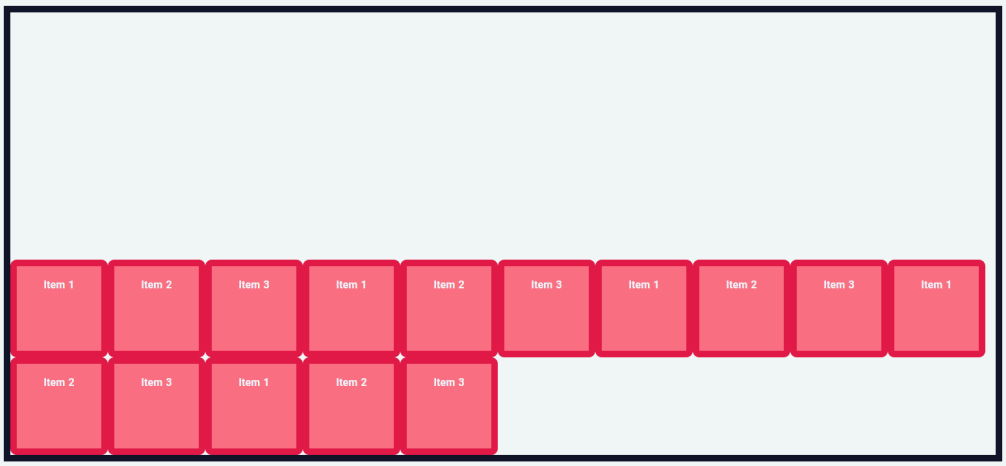
flex-start

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
    flex-wrap: wrap;
    align-content: flex-start;
}
```



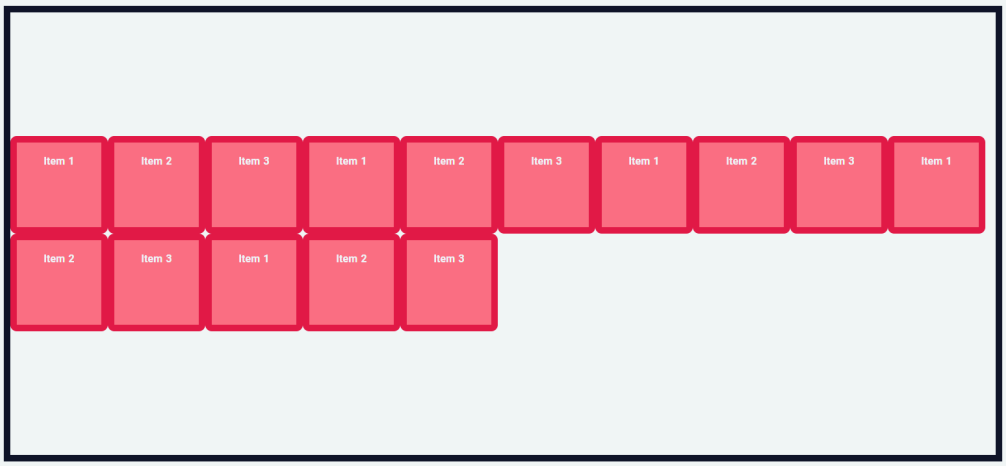
flex-end

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
    flex-wrap: wrap;
    align-content: flex-end;
}
```



center

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
    flex-wrap: wrap;
    align-content: center;
}
```

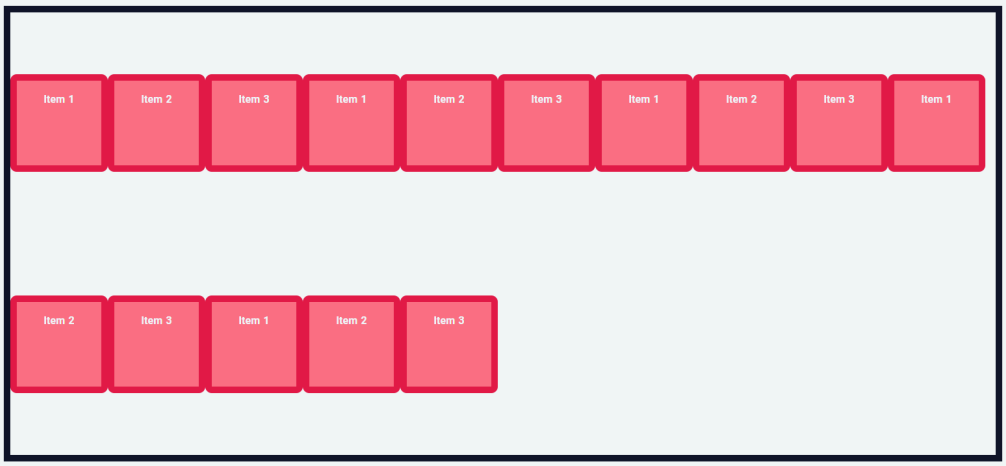


space-between

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
    flex-wrap: wrap;
    align-content: space-between;
}
```

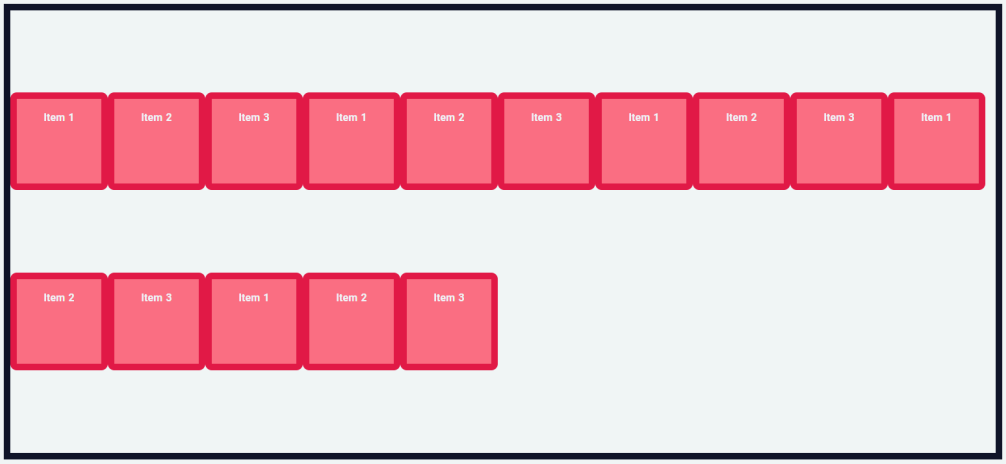
space-around

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
    flex-wrap: wrap;
    align-content: space-around;
}
```



space-evenly

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
    flex-wrap: wrap;
    align-content: space-evenly;
}
```

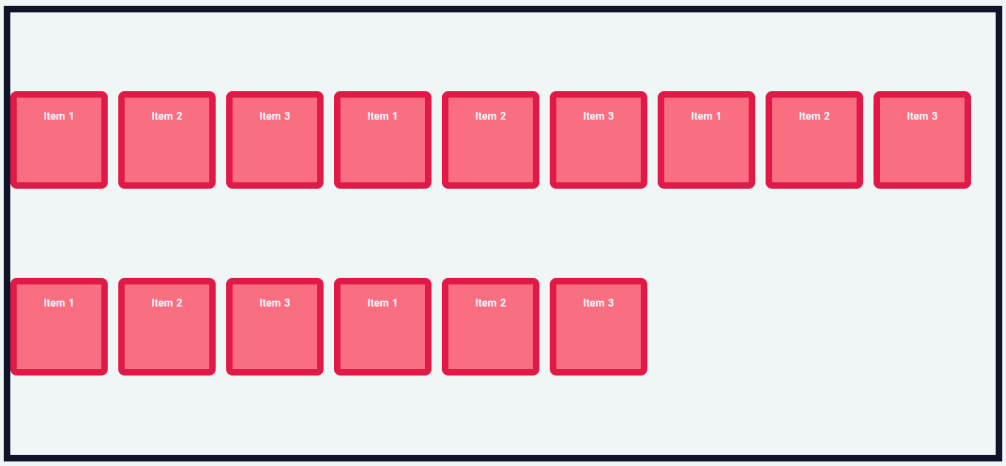


gap

You can also add gaps in between items.

Currently all our items are touching each-other, we can use the **gap** property on the container to add gaps in-between each item, like this.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
    flex-wrap: wrap;
    align-content: space-evenly;
    gap: 1em;
}
```



All the flexbox properties I've presented so far - belong inside the container.

However, there's also a few other flexbox properties but that belong in the direct children of the container.

flex-grow

flex-grow is one of them but before I get into it, in the HTML I'm going to remove most of the items and leave just the first 3. Then back in our CSS we can remove the flex-wrap and the align-content because we don't need them anymore.

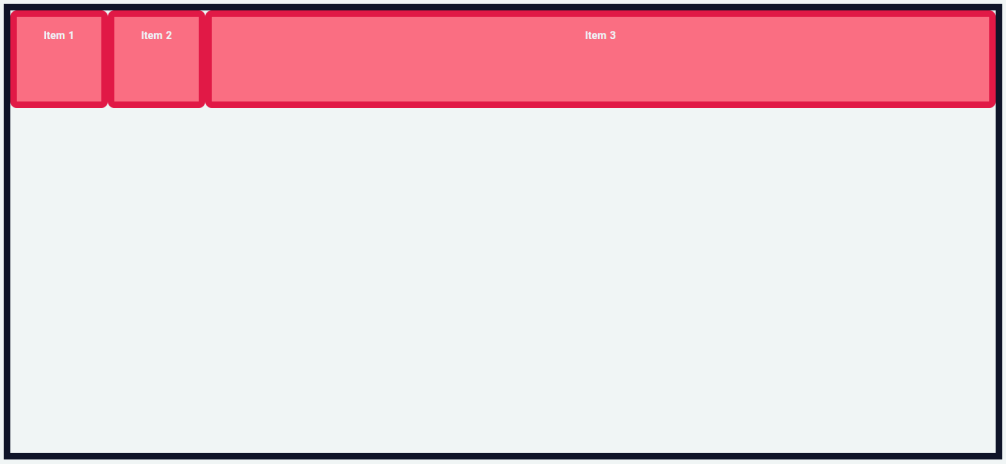
```
<div class="container">
  <div class="item item-1">Item 1</div>
  <div class="item item-2">Item 2</div>
  <div class="item item-3">Item 3</div>
</div>
```

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

```
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}
```

Then under our container we can select the third item by it's class name, item-3 and give it the flex-grow of 1.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-3 {
    flex-grow: 1;
}
```

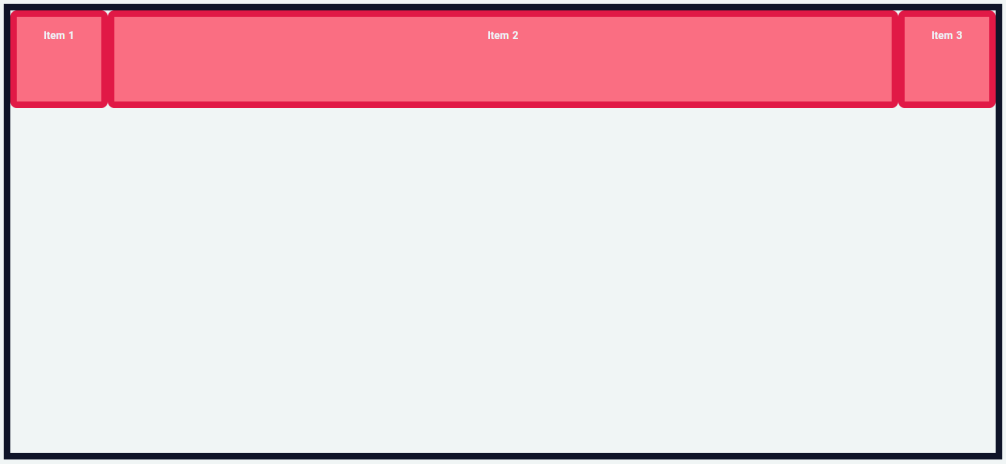


flex-grow takes a unitless value that serves as a proportion and what it does is allows the item to grow if there is enough space to do so.

We see our third item grew to fill out the remaining space..

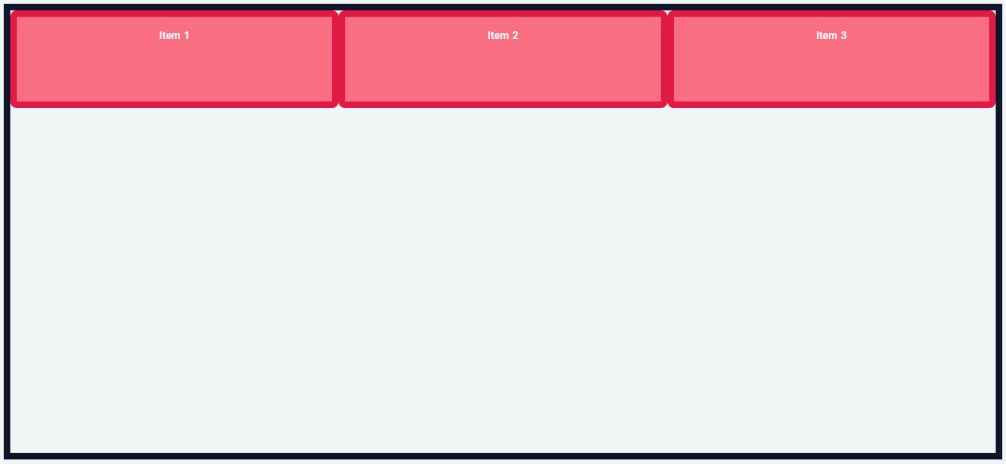
The same will happen if instead I apply this rule on any of the other items.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-2 {
    flex-grow: 1;
}
```



If they all have flex-grow set to 1 then the remaining space will be distributed equally to all children.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-1,
.item-2,
.item-3 {
    flex-grow: 1;
}
```

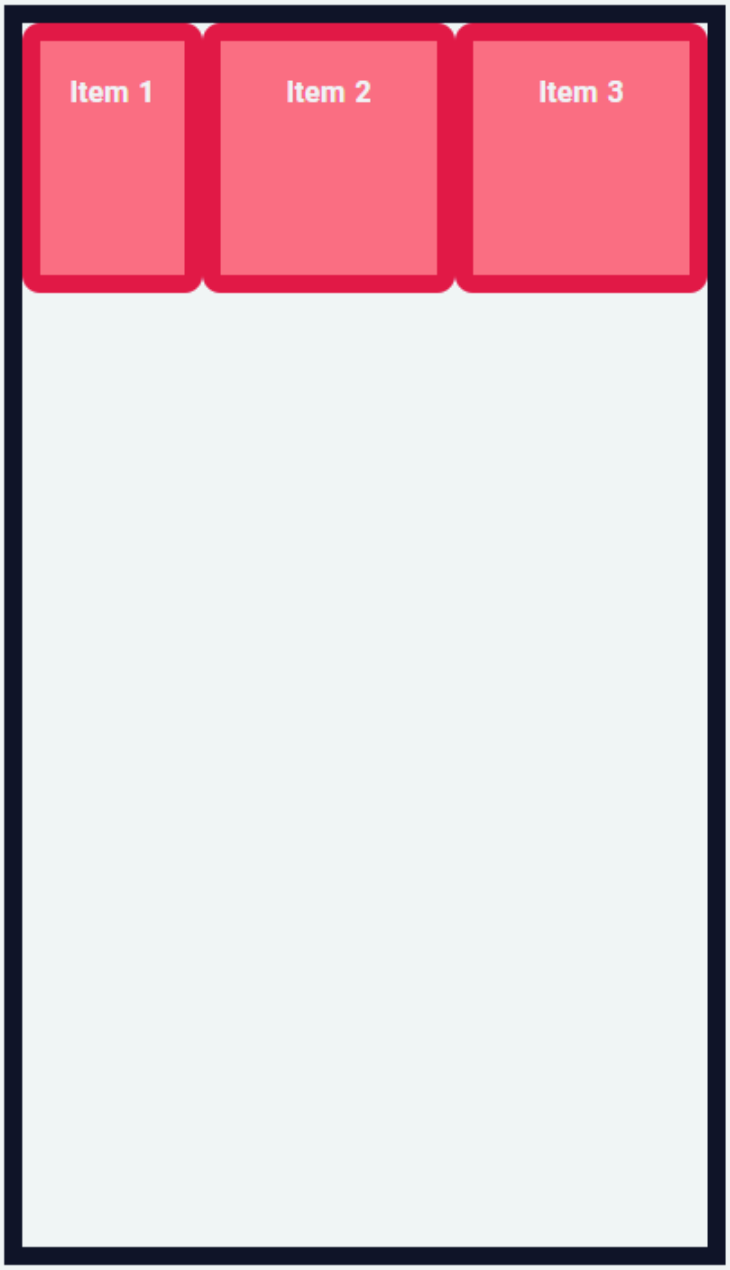


flex-shrink

Next is **flex-shrink**, flex-shrink also takes a unitless value, this property though defines how fast one item shrinks in comparison to the others, if I set item-1 to 5 for example, the item 1 will shrink much faster when I resize.

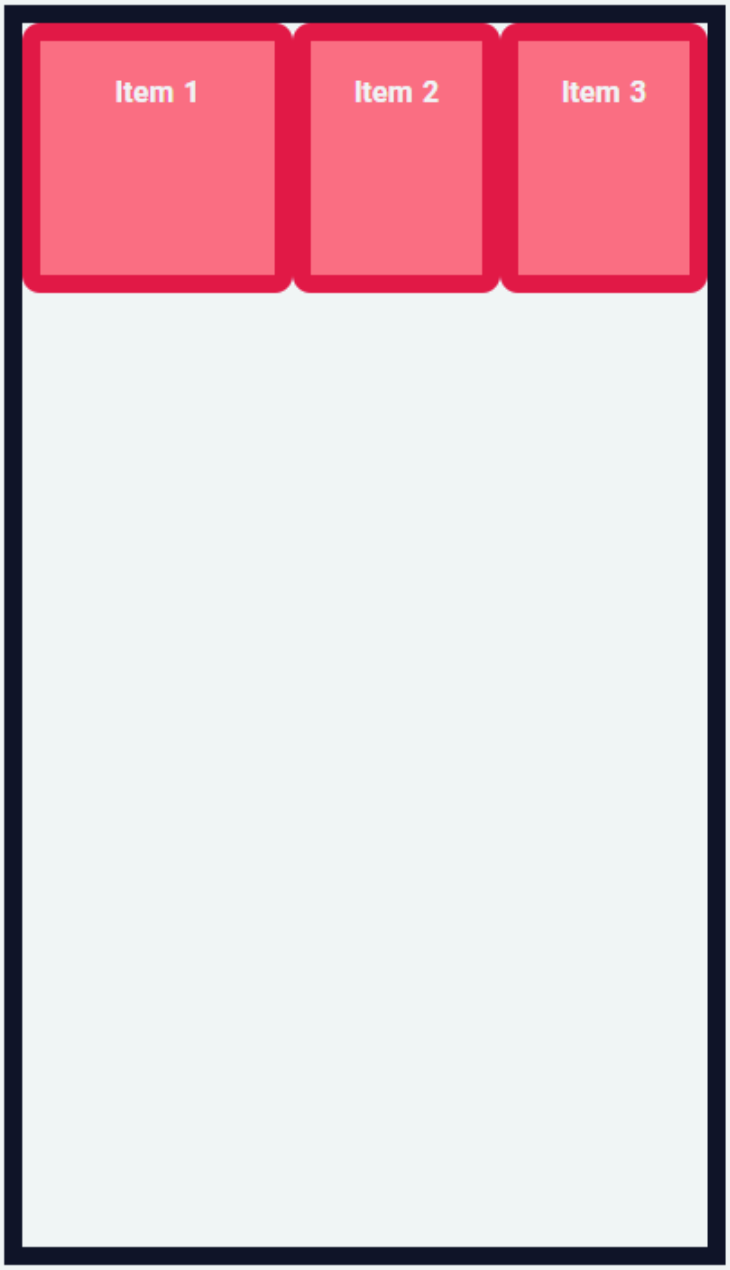
```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-1 {
  /* flex-grow: 1; */
```

```
  flex-shrink: 5;
}
```


Item 1
Item 2
Item 3

If I don’t want an item to shrink at all I can set flex-shrink to 0. Now the item is refusing to shrink.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-1 {
  /* flex-grow: 1; */
  flex-shrink: 0;
}
```


Item 1
Item 2
Item 3

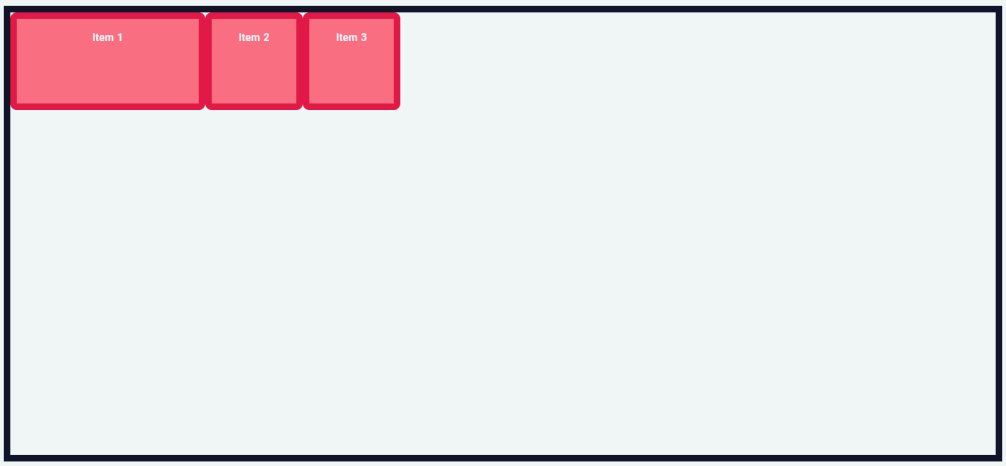
flex-basis

flex-basis defines the size of an item before the remaining space is distributed.

Basically, if your item already has a size like a width but you want to overwrite that size with something else than you use flex-basis.

For example, my item has a width of 150px, I can overwrite that by setting the flex-basis to something else like 300px. Now my item has a width of 300px.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-1 {
  /* flex-grow: 1; */
  /* flex-shrink: 0; */
  flex-basis: 300px;
}
```



If set to 0, you’re basically shrinking it to the max.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-1 {
  /* flex-grow: 1; */
  /* flex-shrink: 0; */
  flex-basis: 0;
}
```

flex shorthand

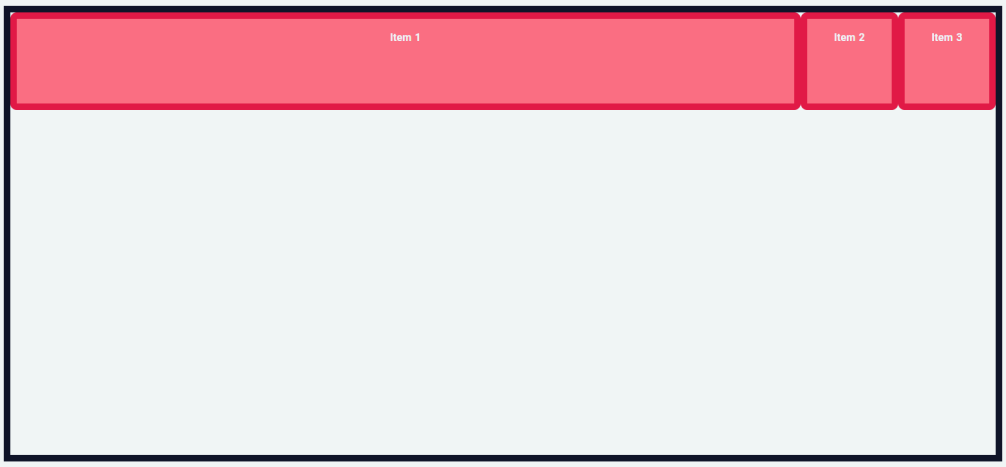
Personally - I almost never (except in a few niche exceptions) use either the flex-grow, flex-shrink or flex-basis properties, but instead, I use the shorthand **flex** property.

flex is the shorthand property for the flex-grow, flex-shrink and flex-basis properties combined, but the second and third parameters are optional.

When you set flex shorthand property to only one value like flex: 1, the other 2 optional values are set automatically and intelligently for you.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}
```

```
.item-1 {
  /* flex-grow: 1; */
  /* flex-shrink: 0; */
  /* flex-basis: 0; */
  flex: 1;
}
```



align-self

Another item property is the **align-self** property.

This one will overwrite the value that you set in the align-items property on the container but for an individual item.

So for example, if I have align-items set to flex-start on the container but want the first item to be aligned center on the cross axis, I can use the align-self property to overwrite what I defined on the container and have this individual item be placed at the **center**.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-1 {
  /* flex-grow: 1; */
  /* flex-shrink: 0; */
  /* flex-basis: 0; */
  /* flex: 1; */
  align-self: center;
}
```

Other than center you also have **flex-start**

```css
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-1 {
  /* flex-grow: 1; */
  /* flex-shrink: 0; */
  /* flex-basis: 0; */
  /* flex: 1; */
  align-self: flex-start;
}
```

flex-end

```css
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-1 {
  /* flex-grow: 1; */
  /* flex-shrink: 0; */
  /* flex-basis: 0; */
  /* flex: 1; */
  align-self: flex-end;
}
```

and **baseline.**

```css
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-1 {
  /* flex-grow: 1; */
  /* flex-shrink: 0; */
  /* flex-basis: 0; */
  /* flex: 1; */
  align-self: baseline;
}
```

order

The last property is the **order** property.

By default our items are laid out in the order of our HTML, item 1, item 2 and item 3.]

However we can use the order property to change the order in which items appear.

If I want the last item to appear first I can give it the order property and set it to -1.

-1 because the default value for this property is 0, that means all item have an order of 0 so for us to have the third item appear first I need to set the third item to have an order of -1.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-1 {
  /* flex-grow: 1; */
  /* flex-shrink: 0; */
  /* flex-basis: 0; */
  /* flex: 1; */
  /* align-self: baseline; */
}

.item-3 {
  order: -1;
}
```

If I want the first item to appear last than I give it the order or 1.

```
.container {
  display: flex;
  /* flex-direction: column; */
  /* justify-content: space-evenly; */
  /* align-items: baseline; */
  /* flex-wrap: wrap; */
  /* align-content: space-evenly; */
  /* gap: 1em; */
}

.item-1 {
  /* flex-grow: 1; */
  /* flex-shrink: 0; */
  /* flex-basis: 0; */
  /* flex: 1; */
  /* align-self: baseline; */
    order: 1;
}
```

```css
.item-3 {
  order: -1;
}
```

This property btw really shouldn’t be something you use unless you absolutely have to because it messes with the semantics and the accessibility of your HTML.