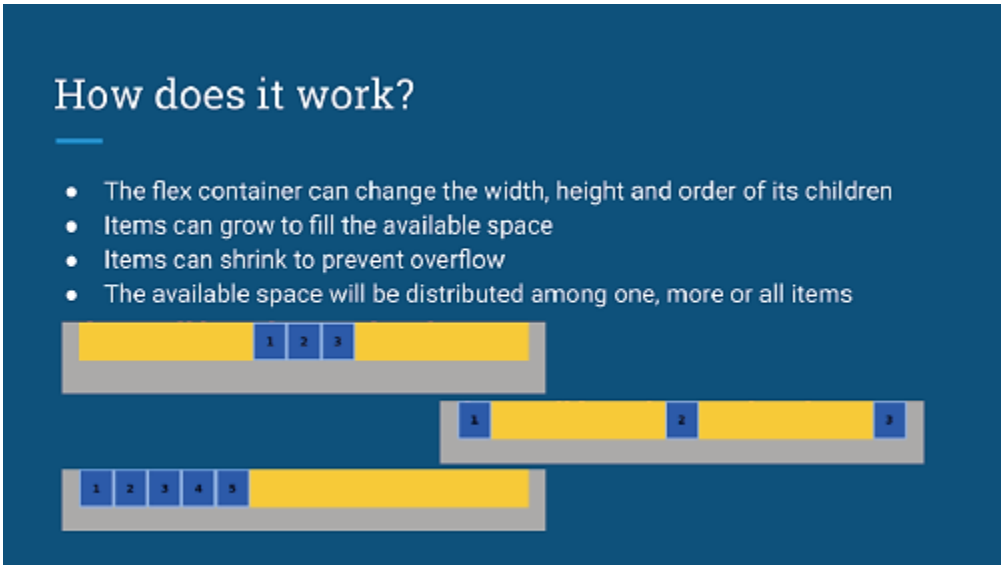
FLEXBOX

Flexbox is a CSS module that gives you great flexibility when creating layouts.
Flexbox layouts can be organized top to bottom, bottom to top, left to right or right to left that most other layout tools are not able to.
Flexbox layouts have items (flex-items) inside a container (flex-container). These items can grow or shrink according to the available container space. The items “flex” to fit the parent container.
You can create a Flexbox layout in context by declaring a container using this property:
{ display : flex }
Flexbox uses a parent-child relationship so when you declare a flex container, the direct children of this container will automatically turn into flex items.
Flex items can be laid out in all directions across their particular axis: left to right, right to left, top to bottom or bottom to top.
Flex items can grow to fill the available space inside a container, or they can shrink to prevent overflow.

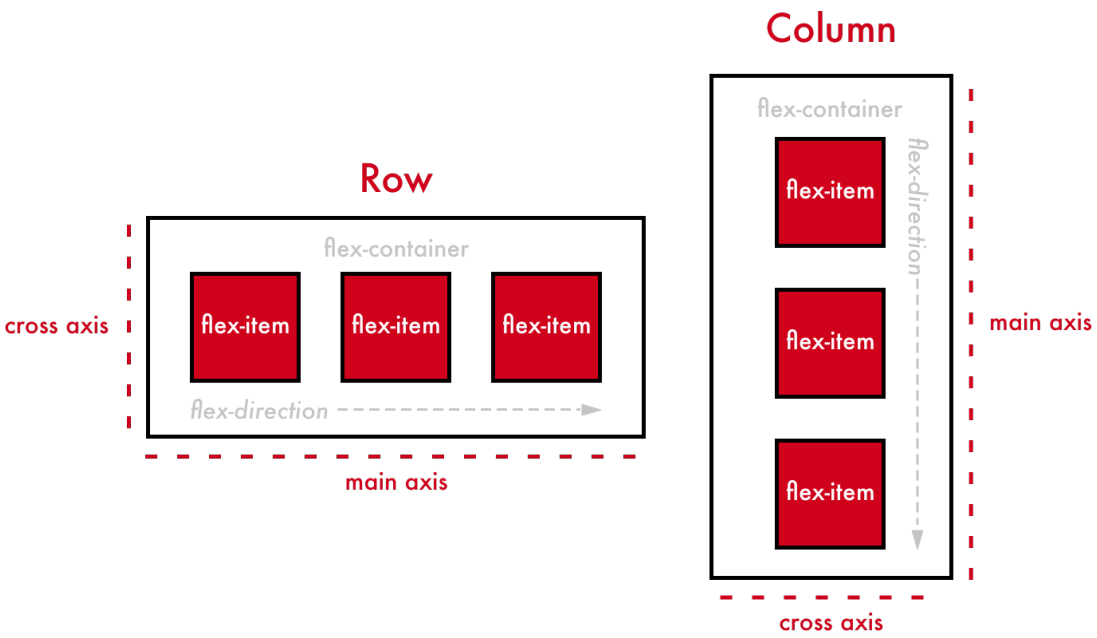


THE FLEXBOX AXES

A flexbox container has **main** axis and **cross** axis. The main axis is determined by the value of the flex-direction property. The main axis can go in these directions:

- left to right: flex-direction: row
- right to left: flex-direction: row-reverse
- top to bottom: flex-direction: column
- bottom to top: flex-direction: column-reverse

The cross axis on the other side will always be perpendicular to the main axis.


Row
flex-container
flex-item
flex-item
flex-item
flex-direction
cross axis
main axis
Column
flex-container
flex-item
flex-item
flex-item
flex-direction
main axis
cross axis

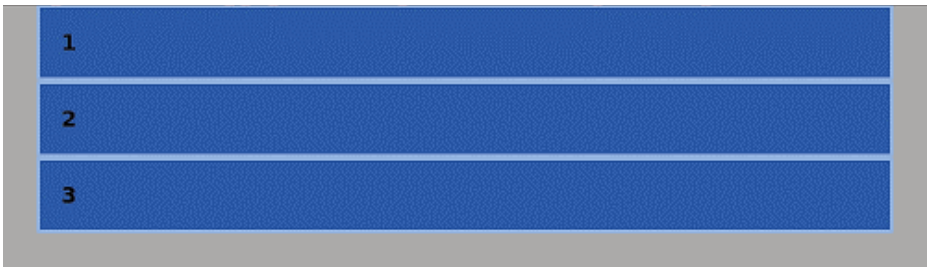
Lab

CREATING A FLEXBOX LAYOUT

Create the following html and css files

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Flexbox Basics</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <div class="item item1">1</div>
        <div class="item item2">2</div>
        <div class="item item3">3</div>
    </div>
</body>
</html>
```

```css
/* GLOBAL STYLES */

* {
    box-sizing: border-box;
}

body {
    background-color: #AAA;
    margin: 0px 50px 50px;
}

/* Each item in the grid contains numbers */
.item {
    padding: 2rem;
    border: 5px solid #87b5ff;
    border-radius: 3px;
    font-size: 2em;
    font-family: sans-serif;
    font-weight: bold;
    background-color: #1c57b5;
}
```



Add css

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

Now change css:

```
.container {
    display: flex;
    background-color: #f5ca3c;
}
```

Add to css:

```
.container {
    display: flex;
    background-color: #f5ca3c;
    flex-direction: row-reverse;
}
```

CHANGING FLEXBOX ROWS TO COLUMNS

Now let's learn how to change the direction of our layout. The default direction of a flex-container is row-based. However, you can change this behavior with the flex-direction property.
Edit the CSS code:

```
.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: row;
}
```

After updating your code, you will see no change because flex-direction: row is the default value.

Edit the CSS code:

```
.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: column;
}
```

Now, the direction of the flex-container is based on the block axis (column). The flex-items are aligned from top to bottom, and each one of these items takes the full width of its parent container. So, they behave like block elements.
The layout now looks exactly the same as the very first layout in this tutorial that is true, but we have more control on the new block. For example, you can invert the direction of flex-items with the row-reverse and column-reverse properties.

Edit the CSS code:

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

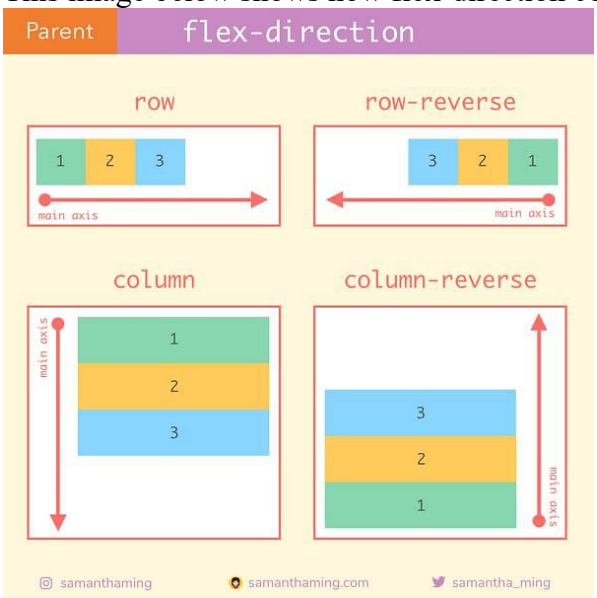
```
  background-color: #f5ca3c;
  flex-direction: column-reverse;
}
```

Now let's try

```
.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: row-reverse;
}
```

THE JUSTIFY-CONTENT PROPERTY

The main axis of the flex-container is determined by the value of the flex-direction property. This image below shows how flex-direction controls the main axis and cross axis:



Understanding the main axis is important because the justify-content property specifies how flex-items are distributed along the main axis.

This justify-content property has five different possible values:

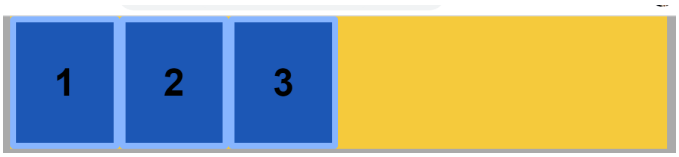
1. flex-start(default)
2. flex-end
3. center
4. space-between
5. space-around

That we will try. So, change CSS

```css
* {
  box-sizing: border-box;
}

body {
  background-color: #aaa;
  margin: 0px 50px 50px;
}

.item {
  padding: 2rem;
  border: 5px solid #87b5ff;
  border-radius: 3px;
  font-size: 2em;
  font-family: sans-serif;
  font-weight: bold;
  background-color: #1c57b5;
}

.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: column;
  justify-content: space-around;
}
```

Edit your CSS code:

```
.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: row;
  /* and then add the following line*/
  justify-content: flex-start;
}
```



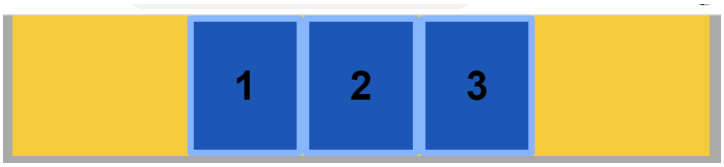
AS you see no changes because default values for flex-direction and justify-content are row and flex-start respectively.

Edit the CSS code:

```
.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: row;
  justify-content: flex-end;
}
```

Edit the CSS code:

```
.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: row;
  justify-content: center;
}
```




```
.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: row;
  justify-content: space-between;
}
```


```
.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: row;
  justify-content: space-around;
}
```

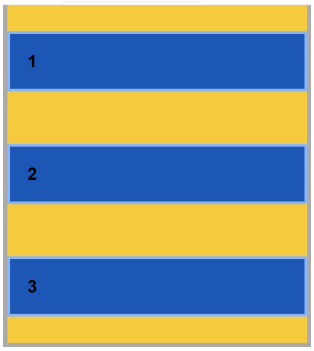
CHANGING FROM THE INLINE TO THE BLOCK AXIS

The flex-direction property determines whether the flex-items are distributed along the inline (row) or block (column) axis.
So far, we've only see flex-items distributed along the inline (row) axis.
To see the behavior of the items on the block axis, we'll make a few changes to the code.

```
container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: column;
  justify-content: space-around;
}
```

The flex-items behave now like block elements. That means they are as high as the content inside them. The same applies to the flex-container. In order to see how the justify-content property works on the block axis, you have to declare a height value for the flex-container.

```
.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: column;
  justify-content: space-around;
  height: 90vh;
}
```



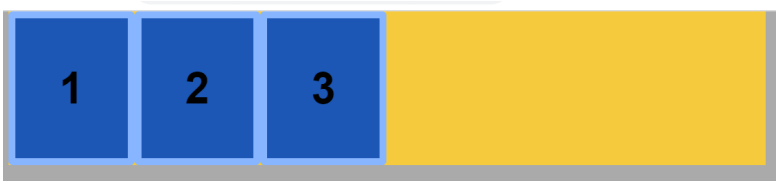
THE ALIGN-ITEMS PROPERTY

Earlier, we learned that a flex-container has two axes:

- main axis
- cross axis

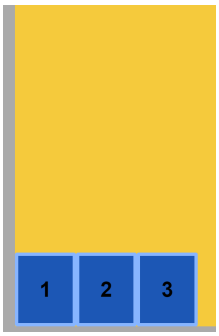
We've also seen that the justify-content property controls how the flex-items are distributed along the main axis.
Let's look at the align-items property, which controls how the flex-items are distributed along the cross axis.

```
* {
  box-sizing: border-box;
}

body {
  background-color: #aaa;
  margin: 0px 50px 50px;
}

.item {
  padding: 2rem;
  border: 5px solid #87b5ff;
  border-radius: 3px;
  font-size: 2em;
  font-family: sans-serif;
  font-weight: bold;
  background-color: #1c57b5;
}

.container {
    display: flex;
    background-color: #f5ca3c;
    align-items: flex-end;
}
```

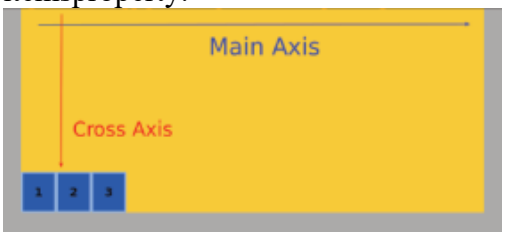


As you can see, the flex-container is only as high as the items inside it, just like a regular block container.

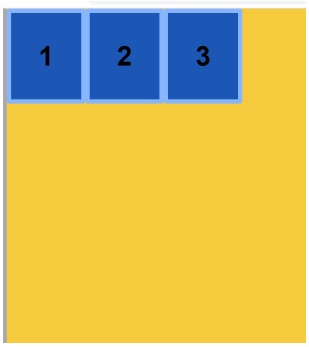
```
.container {
    display: flex;
    background-color: #f5ca3c;
    height: 90vh;
    align-items: flex-end;
}
```



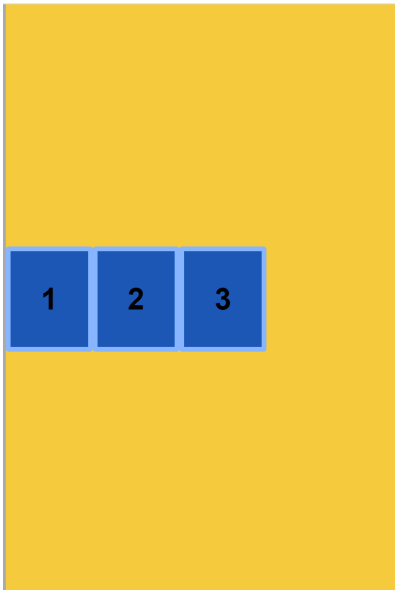
see that the flex-items have been displaced to the end of the cross axis, thanks to the align-itemsproperty.



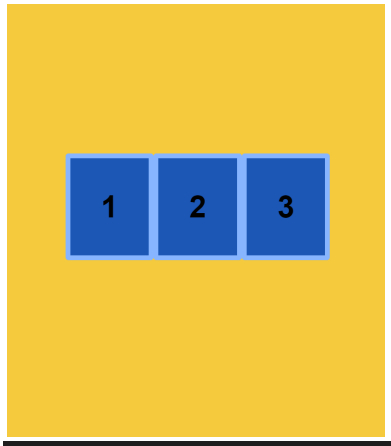

```
.container {
    display: flex;
    background-color: #f5ca3c;
    height: 90vh;
    align-items: flex-start;
}
```



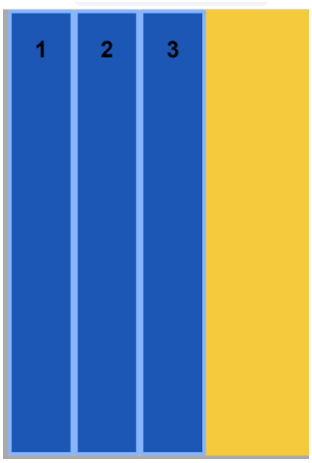

```
.container {
    display: flex;
    background-color: #f5ca3c;
    height: 90vh;
    align-items: center;
}
```




```
.container {
    display: flex;
    background-color: #f5ca3c;
    height: 90vh;
    align-items: center;
    justify-content: center;
}
```



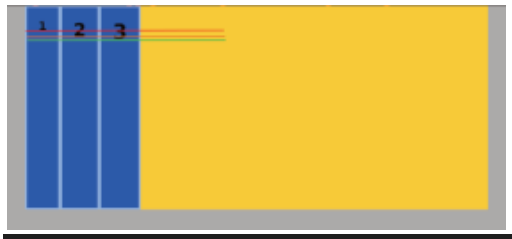

```
.container {
  display: flex;
  background-color: #f5ca3c;
  height: 90vh;
}
```



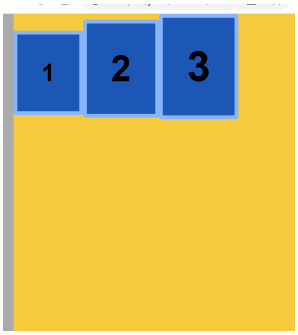
Remember that the default value for the align-items property is stretch.Remember also that if you edit the code and add this default value property, you will see no difference when compared to not including the property.

```
.container {
  display: flex;
  background-color: #f5ca3c;
  height: 90vh;
  align-items: stretch;
}

.item2 {
  font-size: 3em;
}

.item3 {
  font-size: 3.5em;
}
```

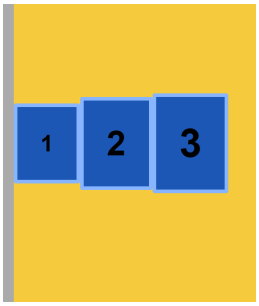
The image below shows what you will see in your browser. Brown and green lines are added to make it easier to see the bottom of each flex-item.



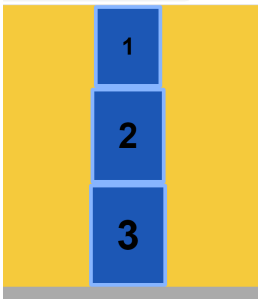

```
.container {
  display: flex;
  background-color: #f5ca3c;
  height: 90vh;
  align-items: baseline;
}
```



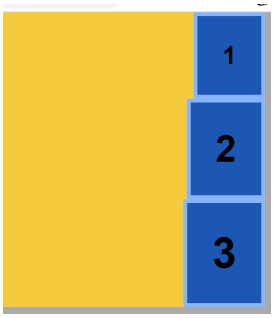

```
.container {
  display: flex;
  background-color: #f5ca3c;
  height: 90vh;
  align-items: center;
}
```




```
.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: column;
  align-items: center;
}
```




```
.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: column;
  align-items: flex-end;
}
```




```
.container {
  display: flex;
  background-color: #f5ca3c;
  flex-direction: column;
  align-items: flex-start;
}
```

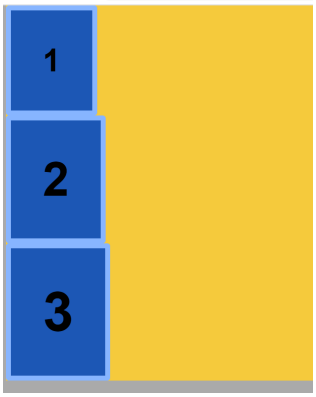
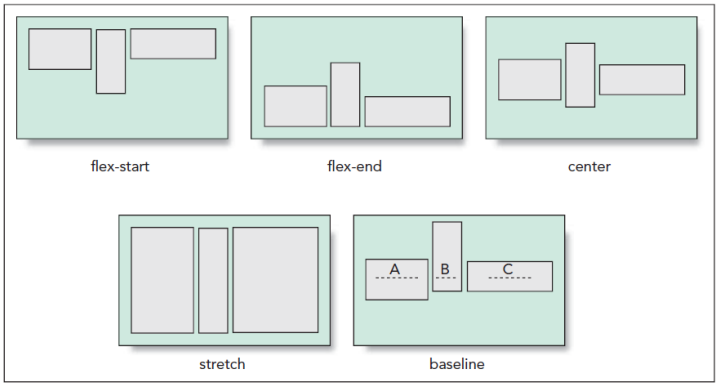


Figure 5–40 Values of the align-items property



THE FLEX-GROW PROPERTY

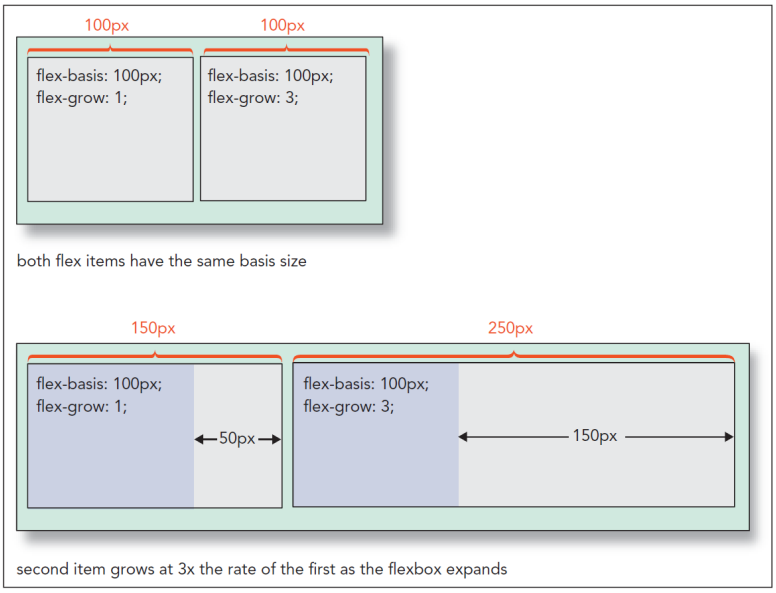
We learned so far:

- the justify-content property controls how the flex-items are distributed along the main axis.
- the align-items property which controls how the flex-items are distributed along the cross axis.

The same is true for:

- The flex-grow property specifies how items will grow along the main axis.
- The flex-shrink property specifies how items will shrink if there's not enough space available.

Figure 5–29 Growing flex items beyond their basis size



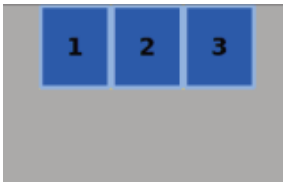
The flex-items automatically position themselves into the flex container, as if they were inline elements. However, this doesn't fit every use-case. For example, you will often want to use all the available space in the flex container. You might also want to set the different items to be different sizes.

In our first flex-grow example, let's suppose that the second item has to be twice as wide as the first and third items.

```
.item1 {
  flex-grow: 1;
}
.item2 {
  flex-grow: 2;
}
.item3 {
  flex-grow: 1;
}
```

If you resize your browser window, you'll notice that the second item will keep this proportion, if there's enough space available considering the width of the viewport. The image below shows the same layout but at a width of 768px.

The image below shows the same layout at a width of 375px, the second item does not have enough space in the container to be twice as wide as the other two items, considering the content inside them. So the available space is distributed equally between all three items.

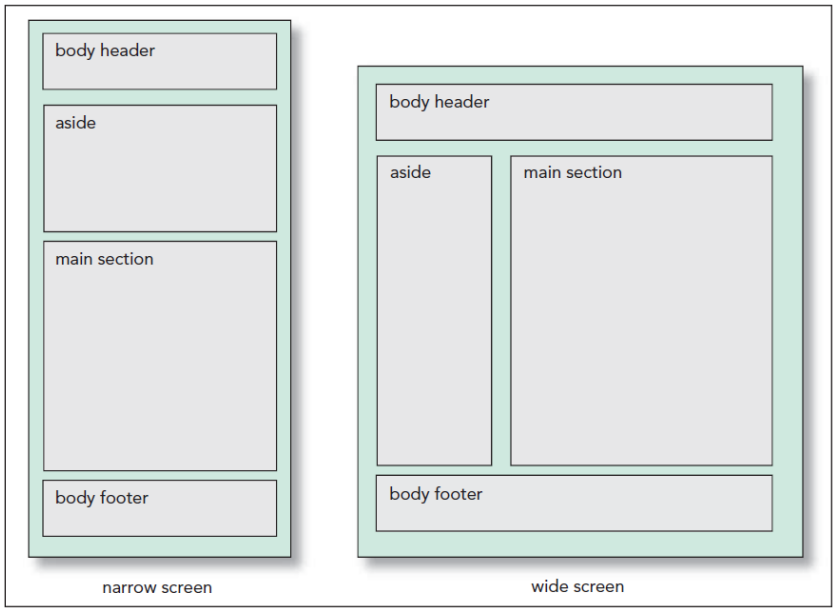


THE FLEX-BASIS PROPERTY

Here, we look at the flex-basis property which sets the initial length of the flex-items inside a flex-container.

You can think of it as an improved version of the width or height values. The flex-basis has always prevalence over width or height.

Figure 5–31 Proposed flex layout for the Pre-K page



Using the techniques of the first session, this would require media queries with one grid layout for narrow screens and a second grid layout for wide screens. However, you can accomplish the same effect with a single flex layout. First, you set the width of the body header and footer to 100% because they will always occupy their own row:

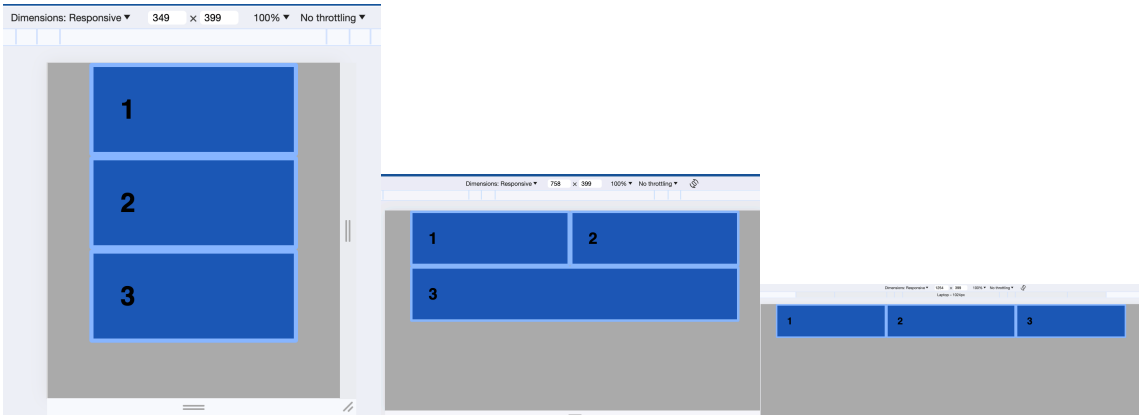
```
header, footer {
   width: 100%;
}
```

Then, you set the basis size of the `aside` and `section` elements to 120 and 361 pixels respectively. As long as the screen width is 481 pixels or greater, these two elements will be displayed side-by-side; however, once the screen width drops below 481 pixels, the elements will wrap to separate rows as illustrated in the narrow screen image in Figure 5–31. Because you want the main `section` element to grow at a rate three times faster than the `aside` element (in order to maintain the 3:1 ratio in their sizes), you set the `flex-growth` values to 1 and 3 respectively. The flex style rules are

```
aside {
   flex: 1 1 120px;
}

section#main {
   flex: 3 1 361px;
}
```

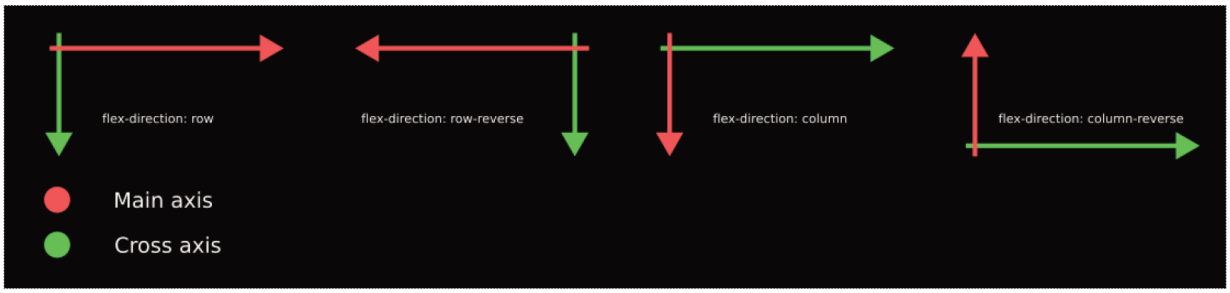
```
* {
    box-sizing: border-box;
}

body {
```

```
    background-color: #AAA;
    margin: 0px 50px 50px;
}

/* Each item in the grid contains numbers */
.item {
    padding: 2rem;
    border: 5px solid #87b5ff;
    border-radius: 3px;
    font-size: 2em;
    font-family: sans-serif;
    font-weight: bold;
    background-color: #1c57b5;
}

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

.item1 {
    flex-basis: 300px;
    flex-grow: 1;
  }
  .item2 {
    flex-basis: 300px;
    flex-grow: 2;
  }
  .item3 {
    flex-basis: 300px;
    flex-grow: 1;
  }
```



THE ORDER PROPERTY

Flexbox doesn't only provide us ways to control the direction and size of items in our layout. Flexbox also enables us to control the order of items, thanks to the order property.
If you want to place flex-items in a particular sequence inside their flex-container, independently of how they are placed in the HTML code, you use the order property.
We saw how to invert the order of the flex-items using row-reverse on the inline axis.

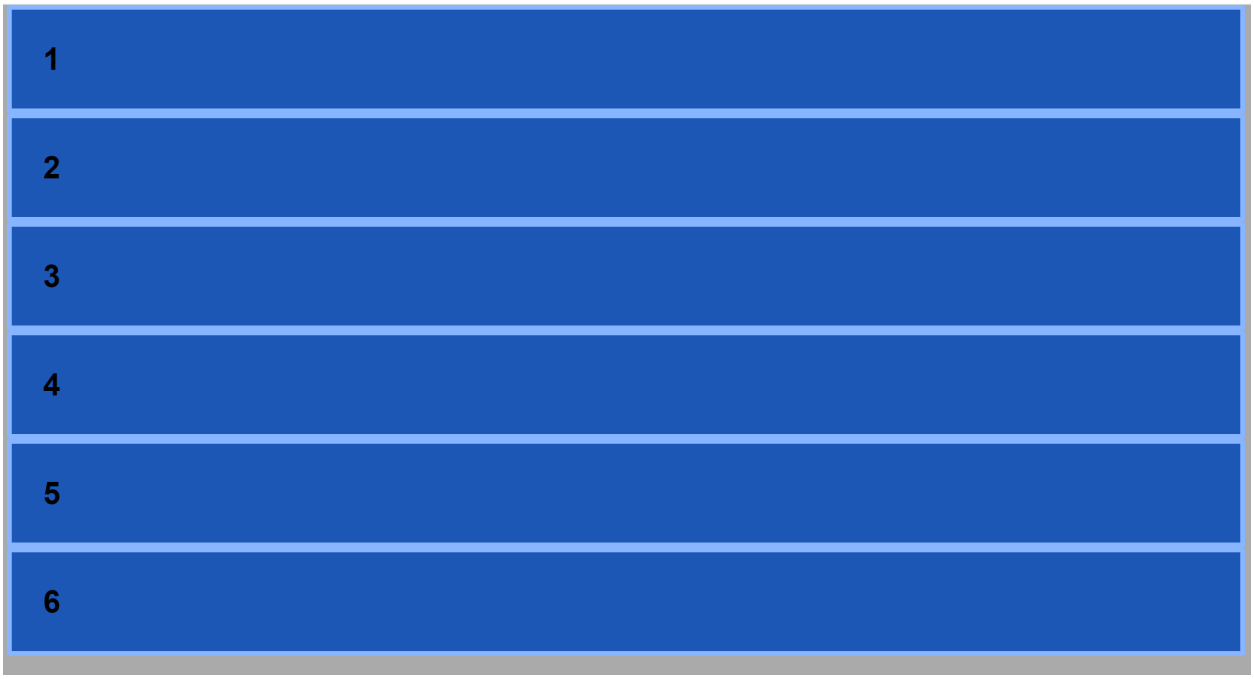


The order property gives you much more flexibility because it allows you to visually change the order of each item and still keep the source order in the HTML code.
CREATE THE HTML AND CSS

```
  <body>
    <div class="container">
      <div class="item item1">1</div>
      <div class="item item2">2</div>
      <div class="item item3">3</div>
      <div class="item item4">4</div>
      <div class="item item5">5</div>
      <div class="item item6">6</div>
    </div>
  </body>
```

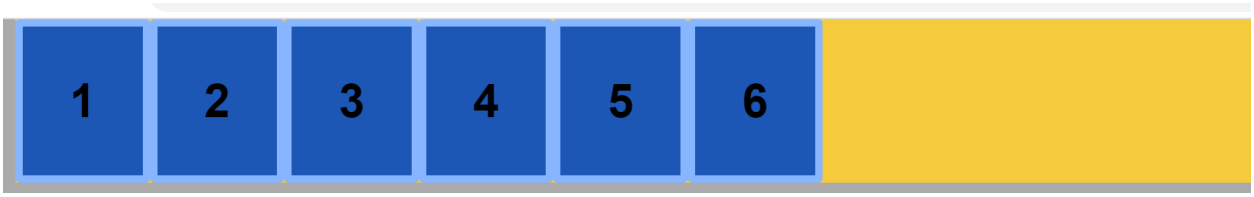
```
/* GLOBAL STYLES */
* {
  box-sizing: border-box;
}
body {
  background-color: #aaa;
  margin: 0px 50px 50px;
}
.item {
  padding: 2rem;
  border: 5px solid #87b5ff;
  border-radius: 3px;
  font-size: 2em;
  font-family: sans-serif;
  font-weight: bold;
```

```
  background-color: #1c57b5;
}
```



THE CSS FLEXBOX STYLES

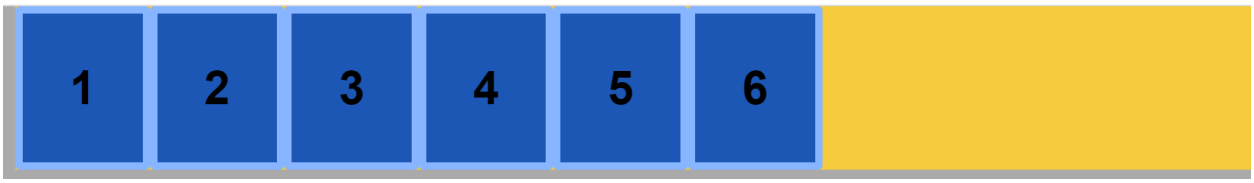
```
.container {
  display: flex;
  background-color: #f5ca3c;
}
```



The default value for the order property is 0.

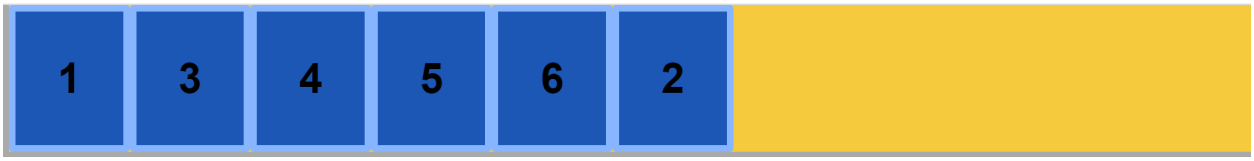
Try

```
.item2 {
    order:0;
}
```



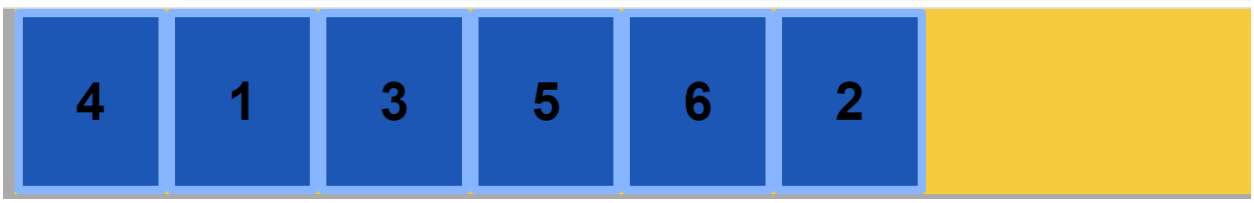
Now try:

```
.item2 {
    order: 1;
}
```



Now try:

```
.item4 {
  order: -1;
}
```



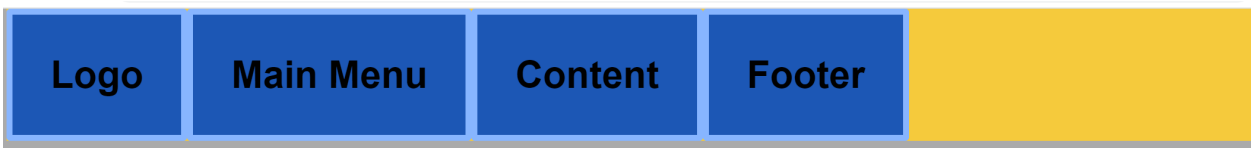
THE FLEX-WRAP PROPERTY

With the flex-wrap property, it is possible to make flex-items wrap over to the next line.

CREATE THE HTML AND CSS:

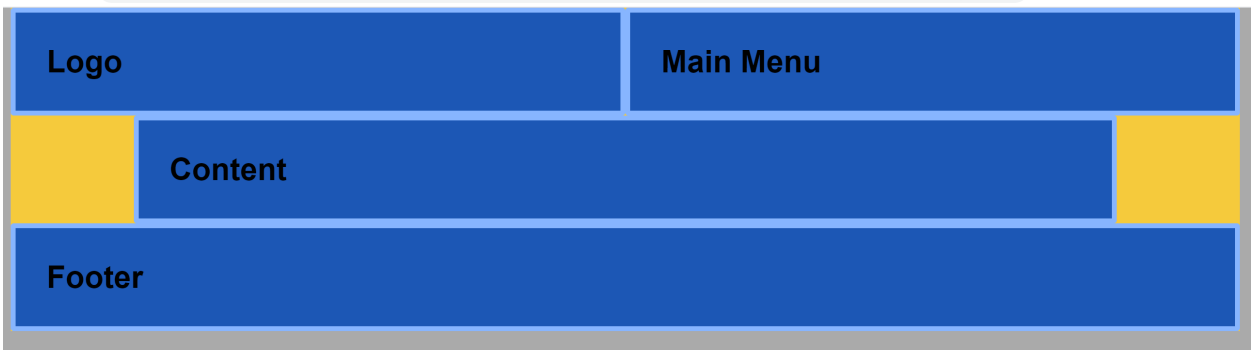
```
    <div class="container">
      <div class="item item1">Logo</div>
      <div class="item item2">Main Menu</div>
      <div class="item item3">Content</div>
      <div class="item item4">Footer</div>
    </div>
```

```
/* GLOBAL STYLES */

* {
    box-sizing: border-box;
 }

 body {
    background-color: #AAA;
    margin: 0px 50px 50px;
 }

 .item {
    padding: 2rem;
    border: 5px solid #87b5ff;
    border-radius: 3px;
    font-size: 2em;
    font-family: sans-serif;
    font-weight: bold;
    background-color: #1c57b5;
 }

.container {
    display: flex;
    background-color: #f5ca3c;
}
```



Now try: pay attention to flex-wrap property

```css
.container {
    display: flex;
    background-color: #f5ca3c;
    flex-wrap: wrap;
}

.item1 {
    flex-basis: 50%;
}

.item2 {
    flex-basis: 50%;
}

.item3 {
    flex-basis: 80%;
    margin: auto;
}

.item4 {
    flex-basis: 100%;
}
```



The flex-wrap property has two other possible values:

- nowrap which is the default value.
- wrap-reverse which inverts the order of start and end on the cross axis.

Try:

```
.container {
    display: flex;
    background-color: #f5ca3c;
    flex-wrap: wrap-reverse;
}
```

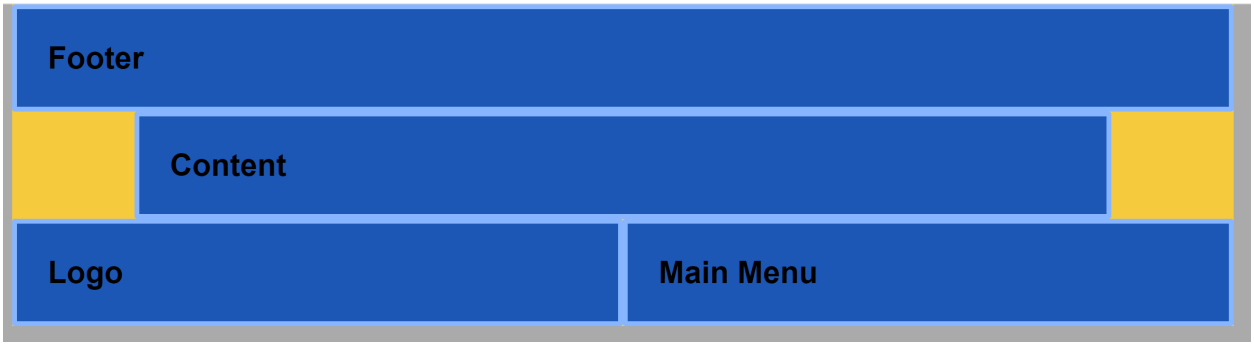
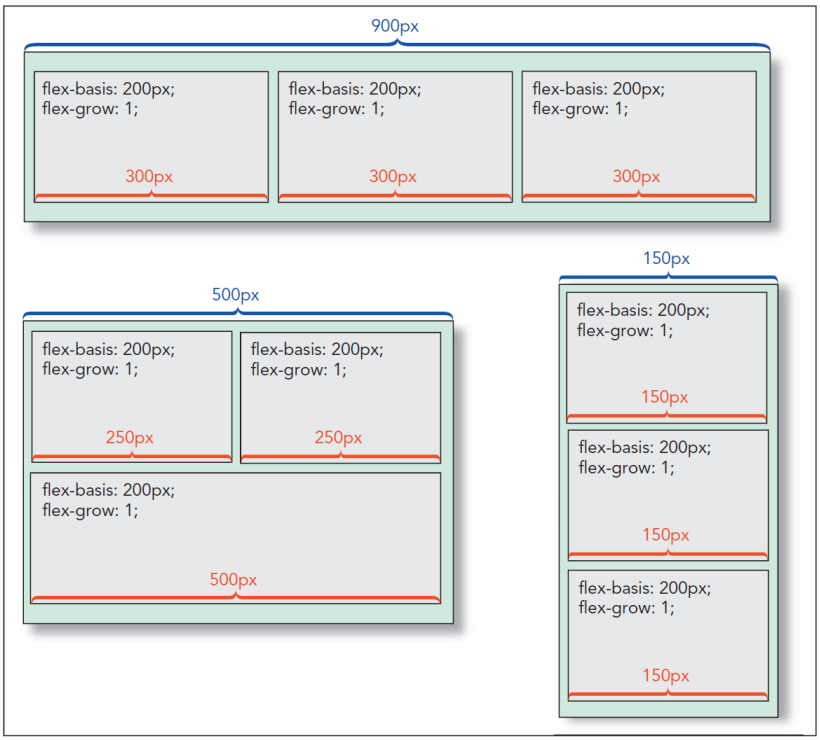


Figure 5–30 Shrinking flex items smaller than their basis size



THE FLEX SHORTHAND PROPERTY

We learned:

- flex-grow specifies how items expand to fill up the available remaining space within the flex-container.
- flex-shrink controls how items shrink if there is not enough space available within the flex-container.
- flex-basis assigns a fixed, ideal width or height to flex-items.

All of these properties can be summed up in one shorthand property, the flex property.
HTML and CSS:

```
    <div class="container">
      <div class="item item1">1</div>
      <div class="item item2">2</div>
      <div class="item item3">3</div>
      <div class="item item4">4</div>
    </div>
```

The flex Shorthand Property

The three properties assigned to the .item class can be shortened in one line using the flex shorthand property. You have to enter the values in this order:

1. flex-grow
2. flex-shrink
3. flex-basis

```
.item {
  flex: 1 1 150px;
}
```

Css:

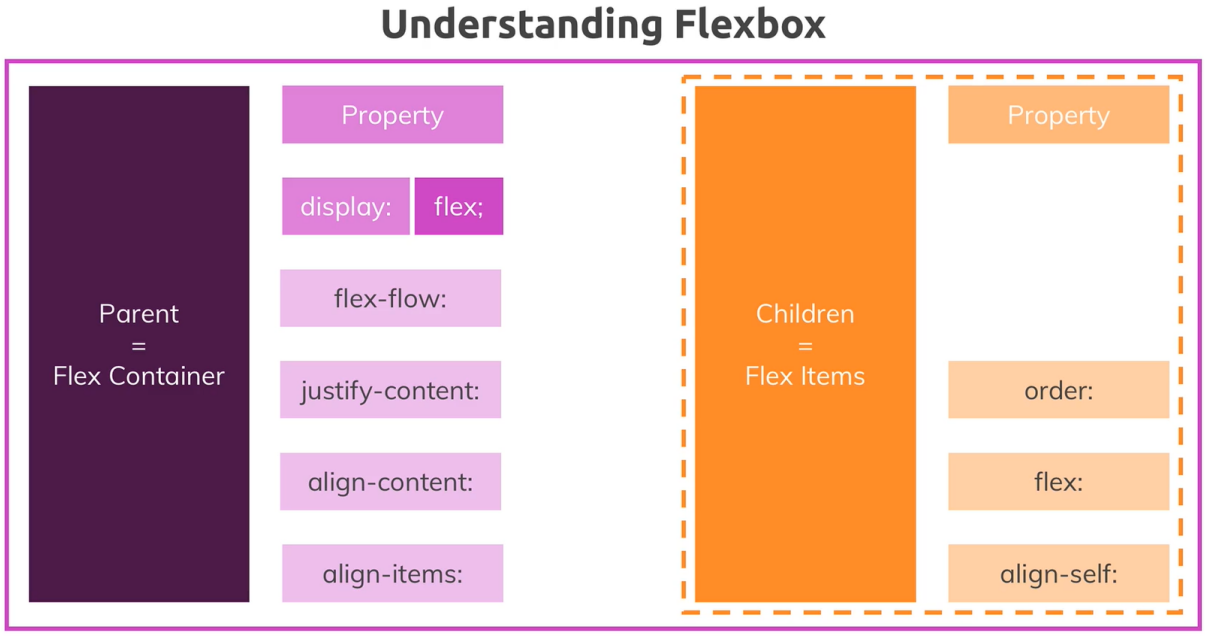
```
/* GLOBAL STYLES */

* {
  box-sizing: border-box;
}

body {
  background-color: #aaa;
  margin: 0px 50px 50px;
}

.item {
  padding: 2rem;
```

```
  border: 5px solid #87b5ff;
  border-radius: 3px;
  font-size: 2em;
  font-family: sans-serif;
  font-weight: bold;
  background-color: #1c57b5;
}

.container {
  display: flex;
  background-color: #f5ca3c;
}

.item {
  flex: 1 1 150px;
}

.item1 {
  flex-shrink: 3;
}
```

Resize your browser window to test the code. You will see no difference in the layout. Your code on the other side seems much cleaner.

THE FLEXBOX PROPERTIES SUMMARY

https://flexbox.malven.co/



Flex Box Chart

Property	Value(s)
#1 display	flex
#2 flex-direction	row||column||column-reverse||row-reverse
#3 justify-content	flex-start || flex-end || center || space-between || space-around || space-evenly
#4 align-items	flex-start || flex-end || center || stretch || baseline
#5 align-content	flex-start || flex-end || center || stretch||space-between||space-around
#6 align-self	auto || flex-start || flex-end || center || baseline || stretch
#7 Order	/* Any positive Value */
#8 flex-grow	/* Any positive Value */
#9 flex-shrink	/* Any positive Value */
#10flex-wrap	nowrap || wrap || wrap-reverse

/ Joy Shaheb

THE ALIGN-CONTENT PROPERTY

The align-content property specifies how the lines inside the container will be distributed once you have applied the flex-wrap property.
The align-content property accepts six possible values:

- stretch (default)
- center
- flex-start
- flex-end
- space-between
- space-around

Figure 5–38 Values of the justify-content property

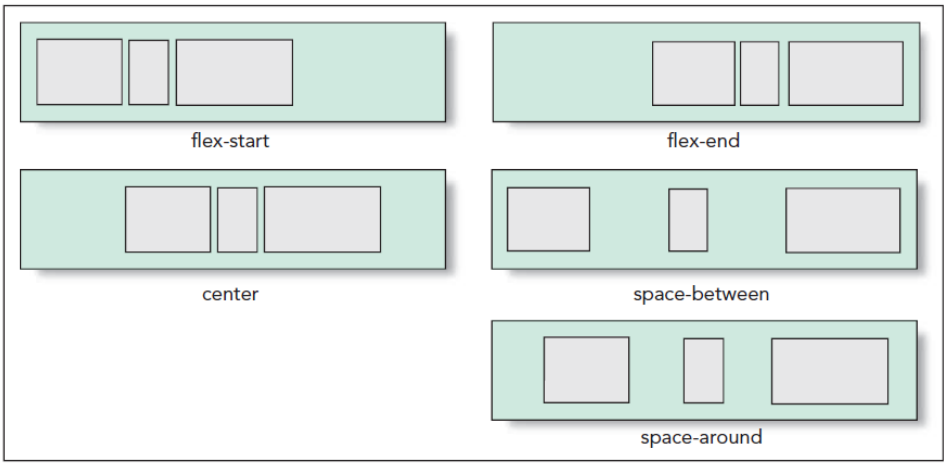


Figure 5–39 Values of the align-content property