The Flex Container Properties

In the previous lesson, I established some fundamental principles. What flex-containers and flex-items are, and how to initiate the Flexbox model.

Now is a good time to put all of that to good use.

Having set a parent element as a flex container, a couple of alignment properties are made available to be used on the flex container.

Just like you'd define the width property on a block element as `width: 200px`, there are 6 different properties the flex container can take on.

The good news is that defining these properties doesn't require a different approach from what you're already used to.

1. Flex-direction

The `Flex-direction` property controls the direction in which the flex-items are laid along the **main axis**.

It may take any of four values.

```
/*where ul represents a flex container*/
ul {
  flex-direction: row || column || row-reverse || column-reverse;
  }
```

In layman's terms, the `flex-direction` property let's you decide how the flex items are laid out. Either *horizontally*, *vertically* or *reversed* in both directions.

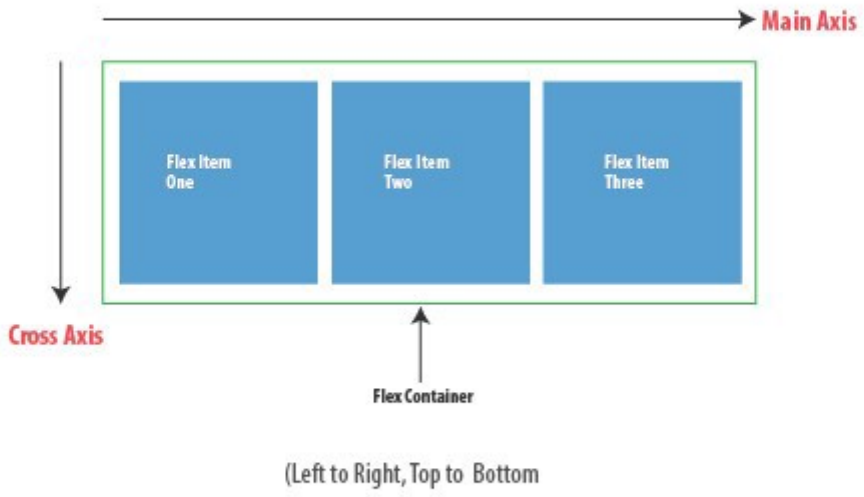
Technically, *"horizontal"* and *"vertical"* isn't what the directions are called in the "flex world".

These are described as **main-axis** and **cross axis**. The defaults are shown below.

In layman's terms again, the main-axis' default direction feels like *"horizontal."*

From left to right.

The cross-axis feels like “vertical.” From top to bottom.



By default, the `flex-direction` property is set to `row` and it aligns the flex-item(s) along the main axis. This explains what happened with the unordered list at the start of this article.

Even though the `flex-direction` property wasn’t explicitly set, it took on the default value of `row`.

The flex items were then laid across the main-axis, stacking horizontally from left to right

If the `flex-direction` property is changed to `column`, the flex-items will be aligned along the cross axis.

along the cross axis.

They would stack from top to bottom, not from left to right any longer.

2. Flex-wrap

The flex wrap property can take on any of three values:

```
//where ul represents a flex container
ul {
  flex-wrap: wrap || no-wrap || wrap-reverse;
  }
```

I'll explain how the `flex-wrap` property works by walking you through an example.

Try sticking a lot more list items into the unordered list.

What do you think? Will the flex container resize to accommodate more, or will it break up the list items unto another line?

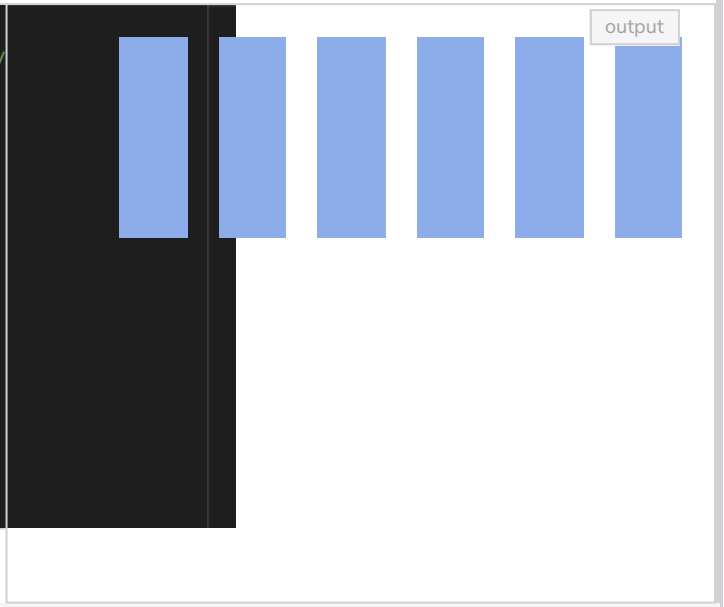
```
/*adding 3 more li elements*/
<ul> <!--parent element-->
  <li></li> <!--first child element-->
  <li></li> <!--second child element-->
  <li></li> <!--third child element-->
  <li></li>
  <li></li>
  <li></li>
</ul>
```

Fortunately, the flex-container adapts to accommodate the new flex-items

HTML CSS Output

```
ul {
    display: flex; /*or inline-flex*/
    list-style: none;
  }

li {
    width: 100px;
    height: 100px;
    background-color: #8cacea;
    margin: 8px;
  }
```



Go a bit further.

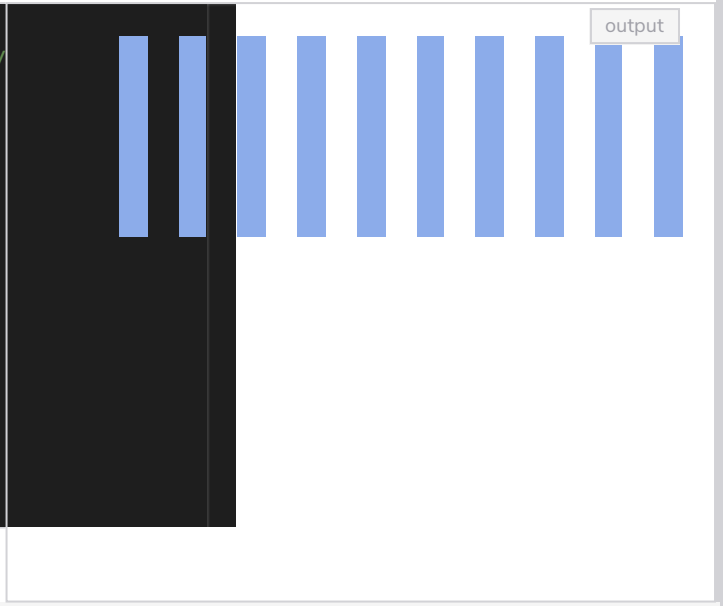
Add a ridiculous amount of flex-items to the parent element. Make it a total of 10 items.

What happens?

HTML CSS Output

```
ul {
    display: flex; /*or inline-flex*/
    list-style: none;
  }

li {
    width: 100px;
    height: 100px;
    background-color: #8cacea;
    margin: 8px;
  }
```



Again, the flex container adapts to fit all children in, even if the browser needs to be scrolled horizontally.

This is the default behavior of every flex container. A flex container will keep on accommodating more flex items on a single line.

This is because the `flex-wrap` property defaults to `nowrap`. This causes the flex container to NOT wrap.

```
ul {
        flex-wrap: no-wrap; /*Keep on taking more flex items without breaking (wrapping)*/
}
```

The `no-wrap` isn't a iron-clad value. It can be changed.

With that number of flex-items, you certainly want the flex-items to *"wrap"* within the flex-container.

"Wrap" is a fancy word to say, "when the available space within the flex-container can no longer house the flex-items in their default widths, break unto multiple lines.

This is possible with the `wrap` value.

```
ul {
        flex-wrap: wrap;
}
```

With this, the flex-items now break up into multiple lines when needed.

In this case, when a single line can no longer contain all the list items in their default width, they break up into multiple lines. Even on resizing the browser.

Here's what that looks like.

Note that the flex items are now displayed in their default widths. There's no need to force multiple flex items unto one line.

HTML CSS Output

```
ul {
        display: flex; /*or inline-flex*/
        list-style: none;
      flex-wrap: wrap;
      }

li {
        width: 100px;
        height: 100px;
        background-color: #8cacea;
        margin: 8px;
```

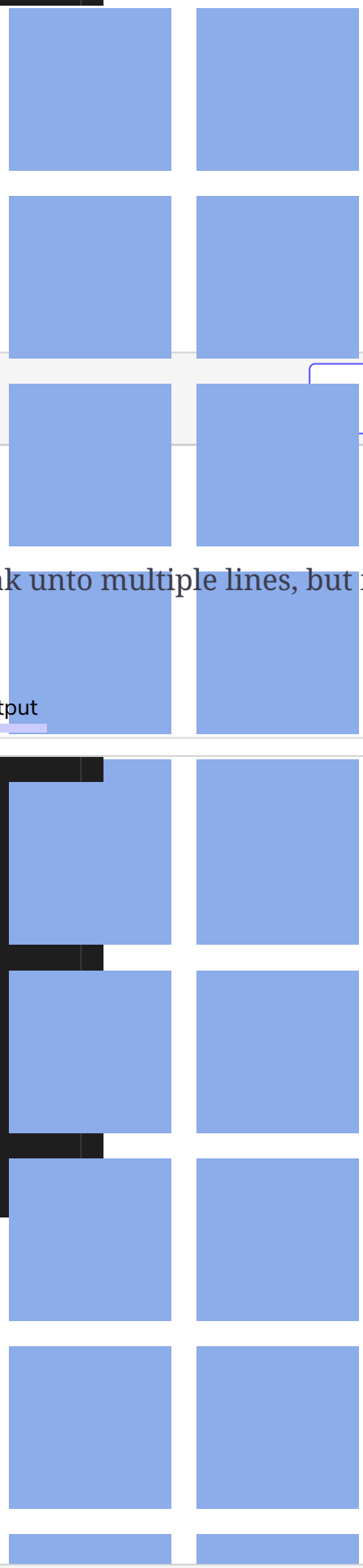
```
12      }
```

There's one more value, `wrap-reverse`.

Yes, you guessed right. It lets the flex items break unto multiple lines, but in the reverse direction.

HTML CSS Output

```
1  ul {
2      display: flex; /*or inline-flex*/
3      list-style: none;
4     flex-wrap: wrap-reverse;
5      }
6
7  li {
8      width: 100px;
9      height: 100px;
10     background-color: #8cacea;
11     margin: 8px;
12     }
```



3. Flex-flow

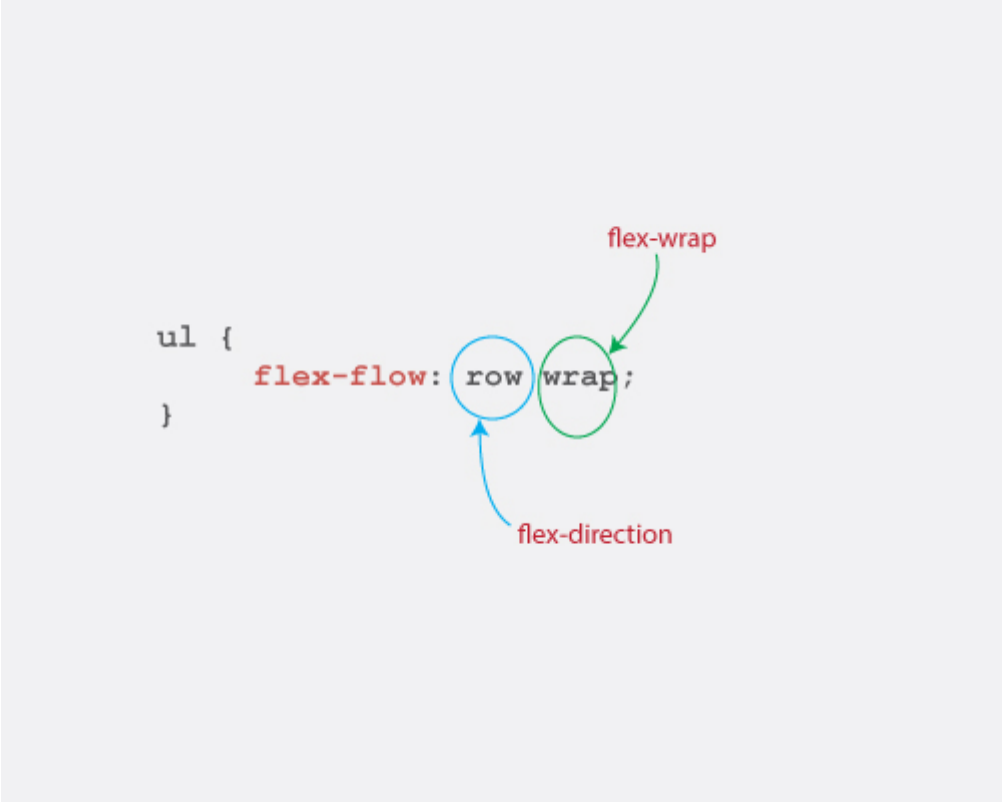
The `flex-flow` is a shorthand property which takes flex-direction and `Flex-wrap` values.

Ever used the `border` shorthand property? `border: 1px solid red`.

It's the same concept here. Multiple values declared in one line.

See the example below.

```
ul {
        flex-flow: row wrap; /*direction 'row' and yes, please wrap the items.*/
}
```



Try out the other combinations this could take. `flex-flow: row nowrap, flex-flow: column wrap, flex-flow: column nowrap`

The results produced are not different from what you've seen with the flex-direction and flex-wrap values.

I'm sure you understand what those would produce.

Give them a try.

4. Justify-content

Life's really good with the Flexbox model. If you still doubt that, the `justify-content` property may convince you.

The `justify-content` property takes on any of the 5 values below.

```
ul {
        justify-content: flex-start || flex-end || center || space-between || space-around
}
```

And what exactly does the `justify-content` property bring to the table?

Well, It may remind you of the text-align property.

The justify content property defines how flex items are laid out on the *main axis*.

A quick example. Consider the simple unordered list below.

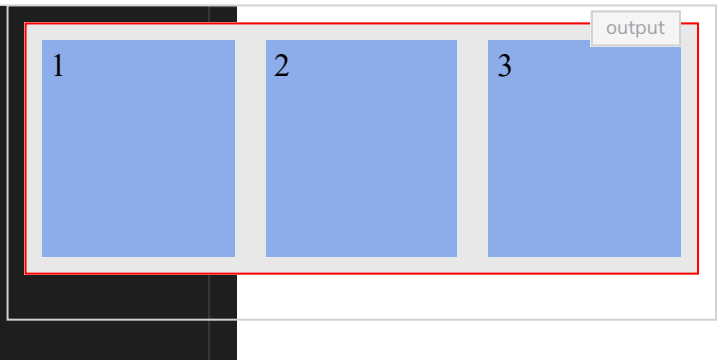
```
<ul>
  <li>1</li>
  <li>2</li>
  <li>3</li>
</ul>
```

Adding up some basic styling...

HTML CSS Output

```
ul {
    display: flex;
    border: 1px solid red;
    padding: 0;
    list-style: none;
    background-color: #e8e8e9;
  }

li {
      background-color: #8cacea;
      width: 100px;
      height: 100px;
      margin: 8px;
```



```
14        padding: 4px;
15    }
```

With the `justify-content` property, the three flex-items may be aligned across the main-axis in whatever way you desire.

Here's the breakdown of what's possible.

(i) Flex-start

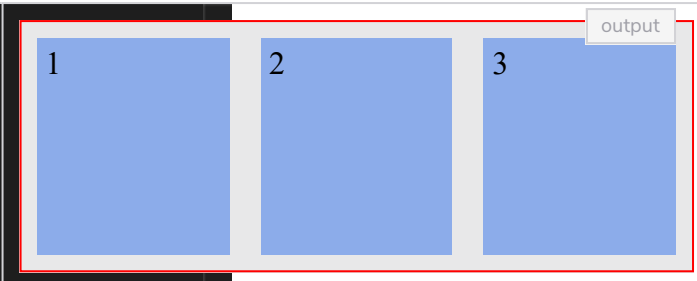
The default value is `flex-start.`

`flex-start` groups all flex-items to the *start* of the main axis

```
ul {
  justify-content: flex-start;
}
```

HTML CSS Output

```
1   ul {
2       display: flex;
3     justify-content: flex-start;
4       border: 1px solid red;
5       padding: 0;
6       list-style: none;
7       background-color: #e8e8e9;
8     }
9
10  li {
11        background-color: #8cacea;
12        width: 100px;
13        height: 100px;
14        margin: 8px;
15        padding: 4px;
16    }
```



(ii) Flex-end

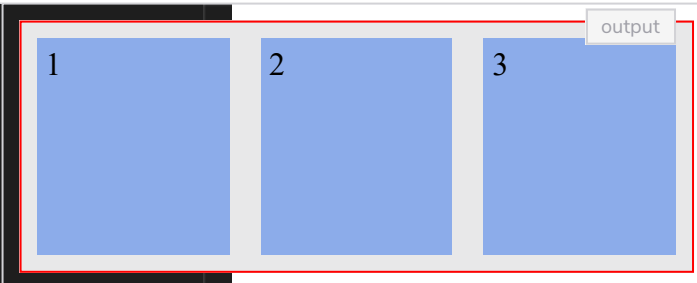
`flex-end` groups the flex-items to the **end** of the main axis.

```
ul {
  justify-content: flex-end;
}
```

HTML CSS Output

```
ul {
    display: flex;
  justify-content: flex-end;
    border: 1px solid red;
    padding: 0;
    list-style: none;
    background-color: #e8e8e9;
  }

li {
      background-color: #8cacea;
      width: 100px;
      height: 100px;
      margin: 8px;
      padding: 4px;
  }
```



(iii) Center

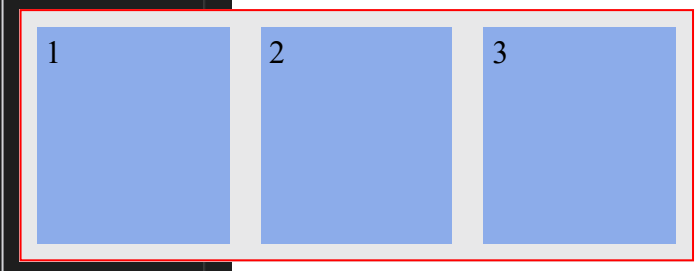
`Center` does just what you'd expect. It centers the flex items along the main axis.

```
ul {
  justify-content: center;
}
```

HTML CSS Output

```
ul {
    display: flex;
  justify-content: center;
    border: 1px solid red;
    padding: 0;
    list-style: none;
    background-color: #e8e8e9;
  }

li {
      background-color: #8cacea;
      width: 100px;
      height: 100px;
      margin: 8px;
      padding: 4px;
  }
```



(iv) Space-between

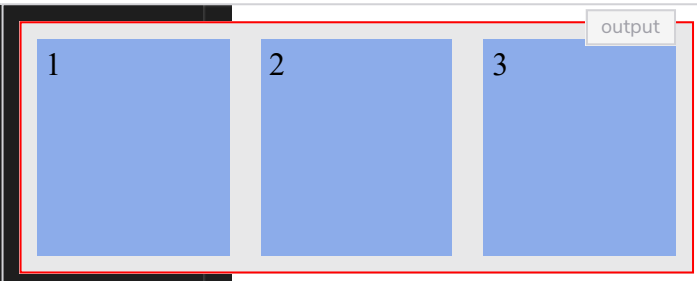
`Space-between` keeps the same space between each flex item.

```
ul {
  justify-content: space-between;
}
```

HTML CSS Output

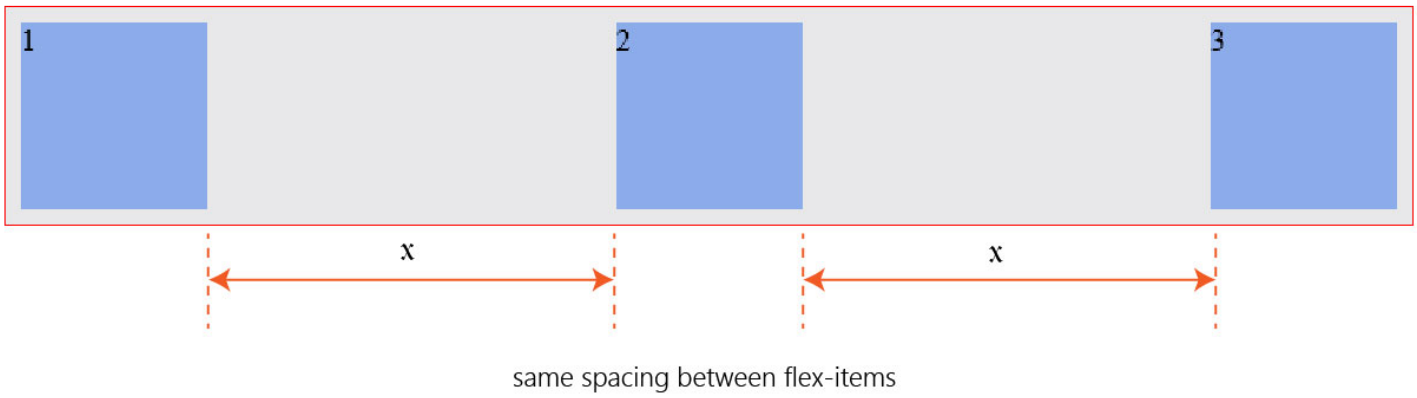
```
ul {
    display: flex;
  justify-content: space-between;
    border: 1px solid red;
    padding: 0;
    list-style: none;
    background-color: #e8e8e9;
  }

li {
      background-color: #8cacea;
      width: 100px;
      height: 100px;
      margin: 8px;
      padding: 4px;
  }
```



Um, did you notice anything different here?

Take a look at the descriptive image below.



(v) Space-around

Finally, `space-around` keeps the same spacing around flex items.

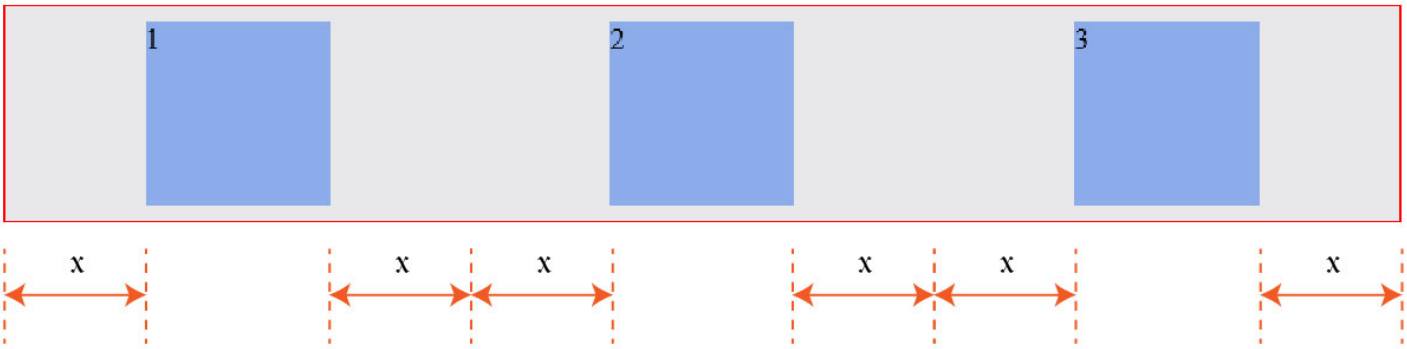
```
ul {
  justify-content: space-around;
}
```

HTML CSS Output

```
ul {
    display: flex;
  justify-content: space-around;
    border: 1px solid red;
    padding: 0;
    list-style: none;
    background-color: #e8e8e9;
  }

li {
      background-color: #8cacea;
      width: 100px;
      height: 100px;
      margin: 8px;
      padding: 4px;
  }
```

A second look doesn't hurt:



same spacing **around** flex-items

Don't worry if these seem like too much to get a hold of. With a bit of practice you will get very comfortable with the syntax.

Be sure to understand how they affect the display of flex items along the main axis.

5. Align-items

The `align-items` property is somewhat similar to the `justify-content` property.

Having understood the `justify-content` property, this should be easier to take in.

`Align-items` can be set to any of these values: `flex-start || flex-end || center || stretch || baseline`

```
/*ul represents any flex container*/
ul {
        align-items: flex-start || flex-end || center || stretch || baseline
}
```

It defines how flex-items are laid out on the *cross axis*. This is the difference

between the `align-items` property and `justify-content`.

Below is how the different values affect flex items.

Do not forget the direction being affected by these properties. The cross-axis.

(i) Stretch

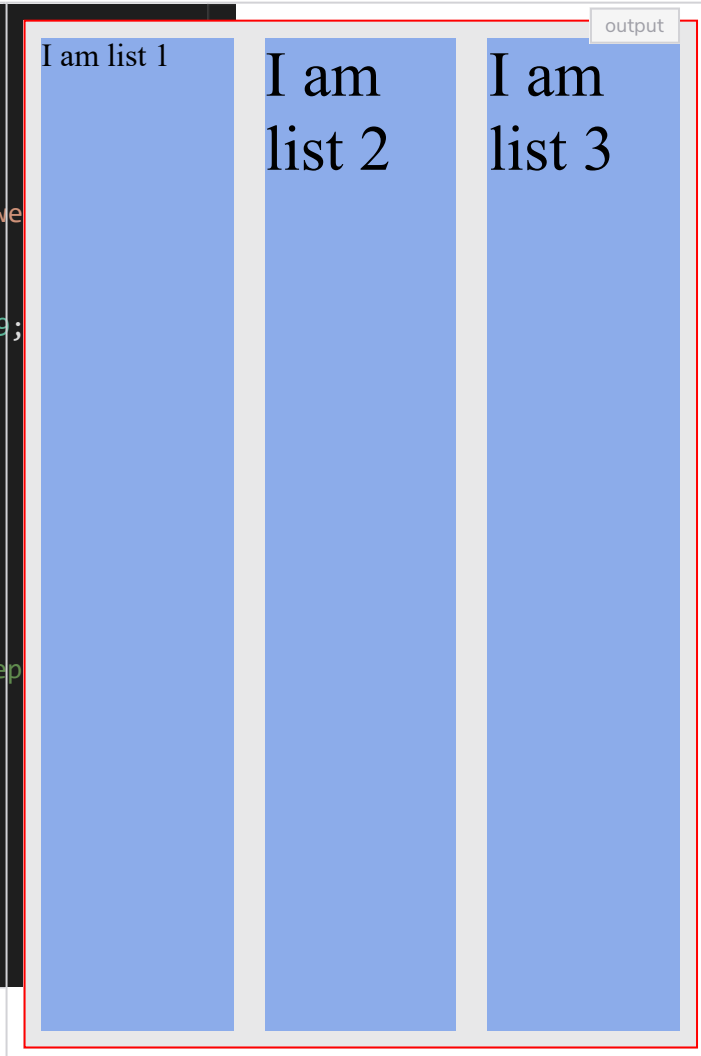
The default value is `stretch`. This will *"stretch"* the flex-items so they fill the entire height of the flex container.

HTML CSS Output

```
ul {
        display: flex;
        border: 1px solid red;
        padding: 0;
        list-style: none;
        justify-content: space-betwe
        align-items: stretch;
        height: 100%;
        background-color: #e8e8e9;
    }

    li {
      width: 100px;
      background-color: #8cacea;
      margin: 8px;
    }

    /* apply to all list items excep
    li:not(:first-child) {
        font-size: 2rem;
    }
```



(ii) Flex-start

The `flex-start` does what you expect. It groups the flex items to the start of the

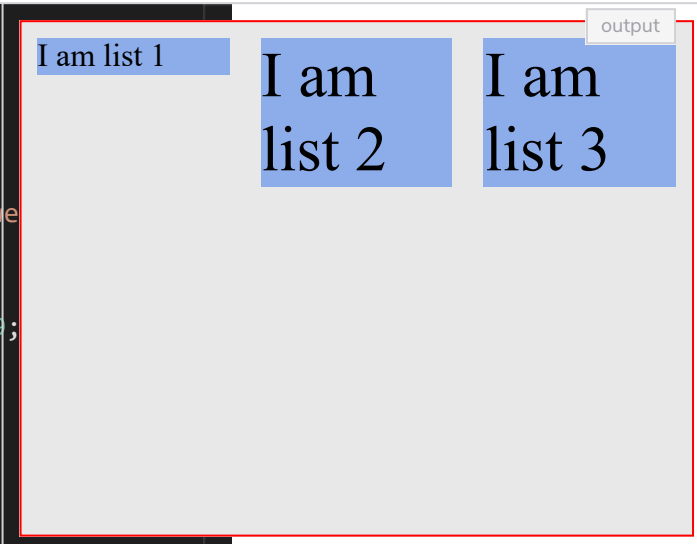
The `flex-start` does what you expect. It groups the flex items to the start of the cross-axis.

HTML CSS Output

```
ul {
        display: flex;
        border: 1px solid red;
        padding: 0;
        list-style: none;
        justify-content: space-betwe
        align-items: flex-start;
        min-height: 50%;
        background-color: #e8e8e9;
    }

    li {
      width: 100px;
      background-color: #8cacea;
      margin: 8px;
    }

    /* apply to all list items except the first on
    li:not(:first-child) {
        font-size: 2rem;
    }
```



(iii) Flex-end

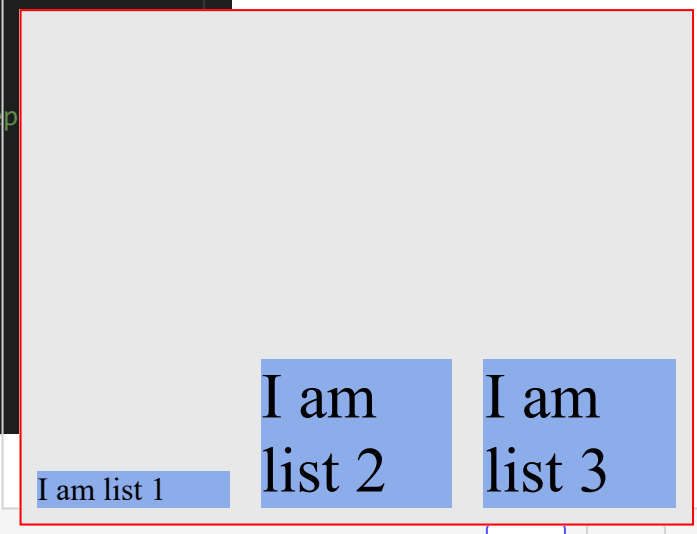
As expected, `flex-end` groups the flex items to the end of the cross-axis.

HTML CSS Output

```
ul {
        display: flex;
        border: 1px solid red;
        padding: 0;
        list-style: none;
        justify-content: space-between;
        align-items: flex-end;
        min-height: 50%;
        background-color: #e8e8e9;
    }
```

output

```
12        li {
13          width: 100px;
14          background-color: #8cacea;
15          margin: 8px;
16        }
17
18        /* apply to all list items excep
19        li:not(:first-child) {
20            font-size: 2rem;
21        }
```

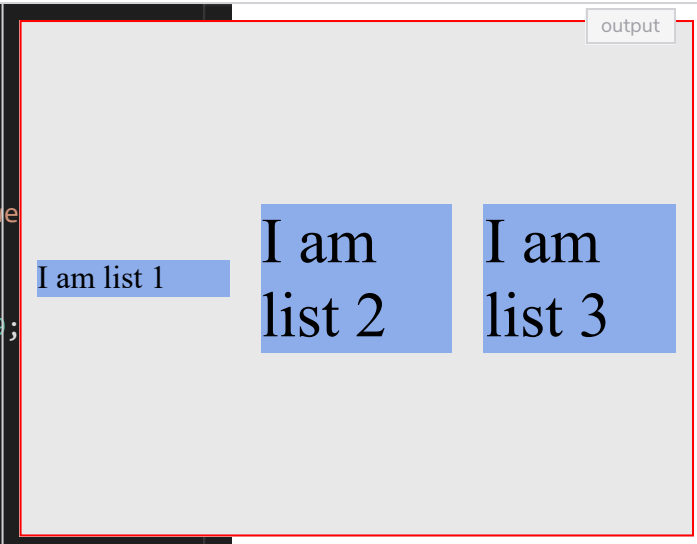


(iv) Center

The `center` value is equally predictable. It aligns the flex items to the center of the flex-container.

HTML CSS Output

```
1   ul {
2           display: flex;
3           border: 1px solid red;
4           padding: 0;
5           list-style: none;
6           justify-content: space-betwe
7           align-items: center;
8           min-height: 50%;
9           background-color: #e8e8e9;
10        }
11
12        li {
13          width: 100px;
14          background-color: #8cacea;
15          margin: 8px;
16        }
17
18        /* apply to all list items except the first on
19        li:not(:first-child) {
20            font-size: 2rem;
21        }
```



(v) Baseline

And the baseline value?

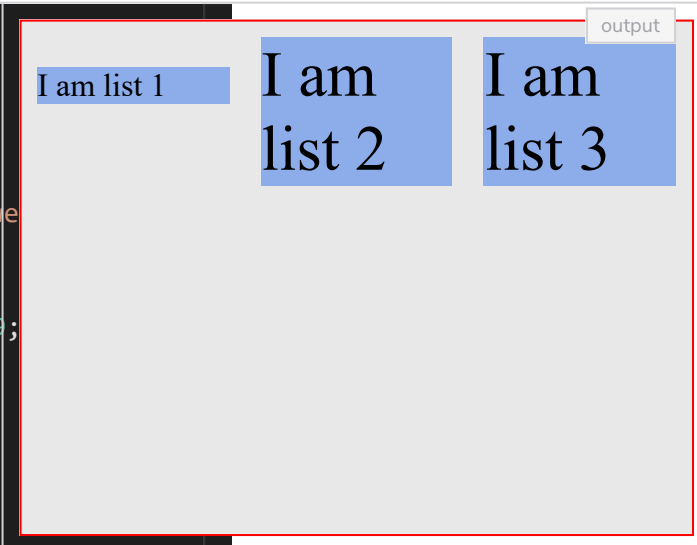
It aligns flex-items along their *baselines*.

HTML CSS Output

```
ul {
        display: flex;
        border: 1px solid red;
        padding: 0;
        list-style: none;
        justify-content: space-betwe
        align-items: baseline;
        min-height: 50%;
        background-color: #e8e8e9;
    }

    li {
      width: 100px;
      background-color: #8cacea;
      margin: 8px;
    }

    /* apply to all list items except the first on
    li:not(:first-child) {
        font-size: 2rem;
    }
```

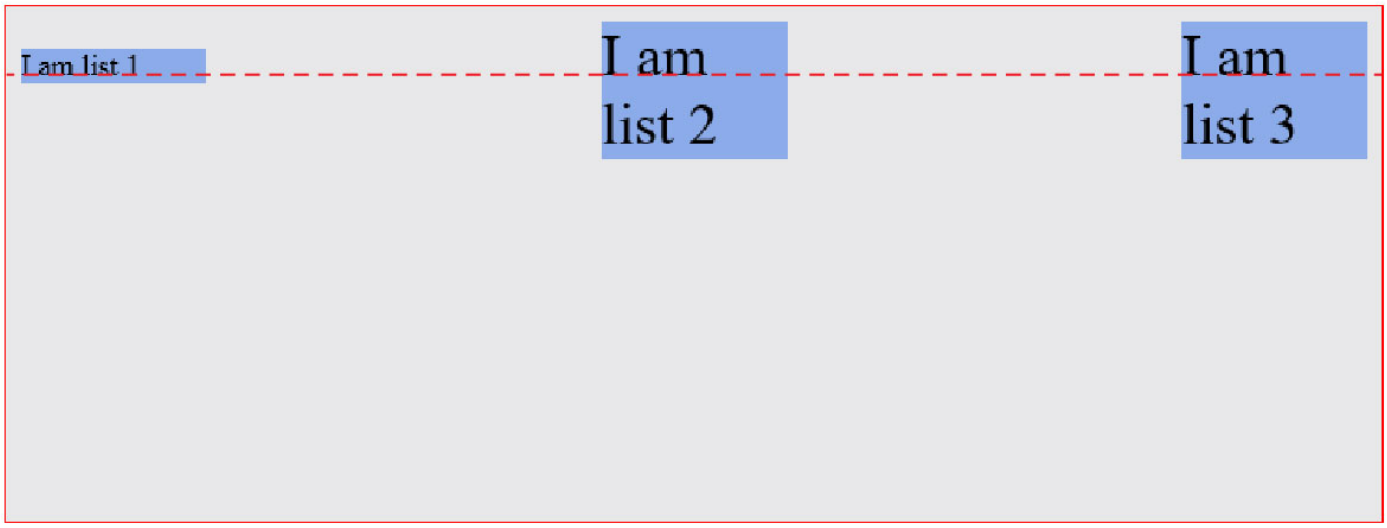


"Baseline" really sounds fancy.

The result appears to look just like `flex-start` but it is subtly different.

What the heck is "baseline"?

The image below should help.



Notice how all the flex-items are aligned to have their content seat on the "baseline"?

6. Align-content

While discussing the `wrap` property, do you remember what happened when you added more flex-items to the flex-container?

You got a *multi-line* flex container.

The `align-content` property is used on *multi-line* flex-containers.

It takes the same values as `align-items` apart from `baseline.`

By definition, it controls how the flex-items are aligned in a multi-line flex container.

Just like `align-items`, the default value is also `stretch`

These are values you should now be familiar with. So, here's how they affect a *multi-line* flex-container with 10 flex-items.

(i) Stretch

With `stretch`, the flex items are “stretched” to fit the available space along the cross-axis.

The spacing you see between the flex items below is owing to the `margin` set on the items.

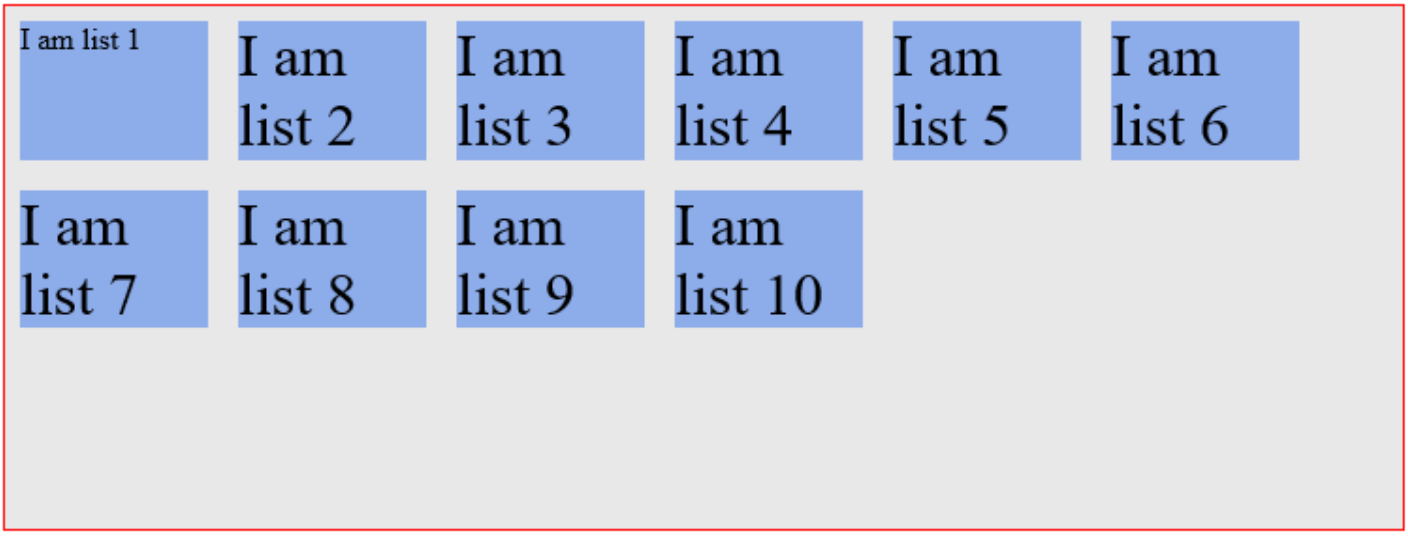
(ii) Flex-start

You’ve seen the `flex-start` value before.

This time it aligns the items in the *multi-line* container to the **start** of the cross-axis.

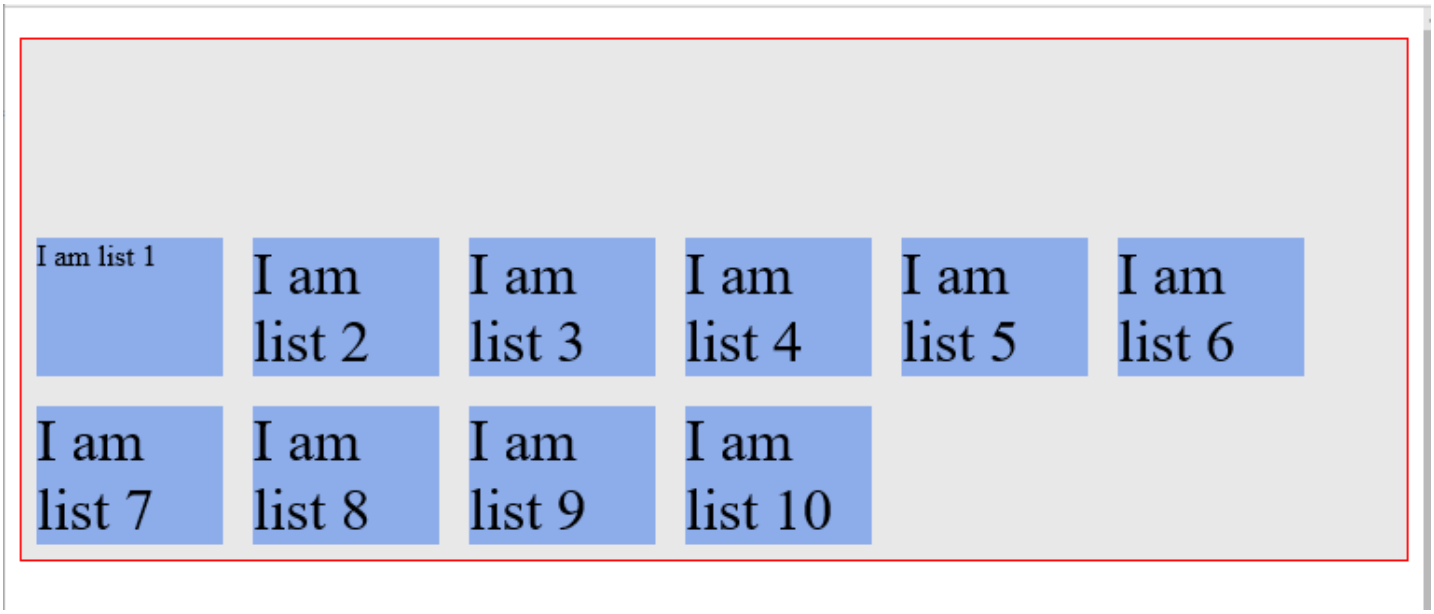
Remember the default cross axis is from top-to-down.

Thus, the flex items are aligned to the top of the flex container.



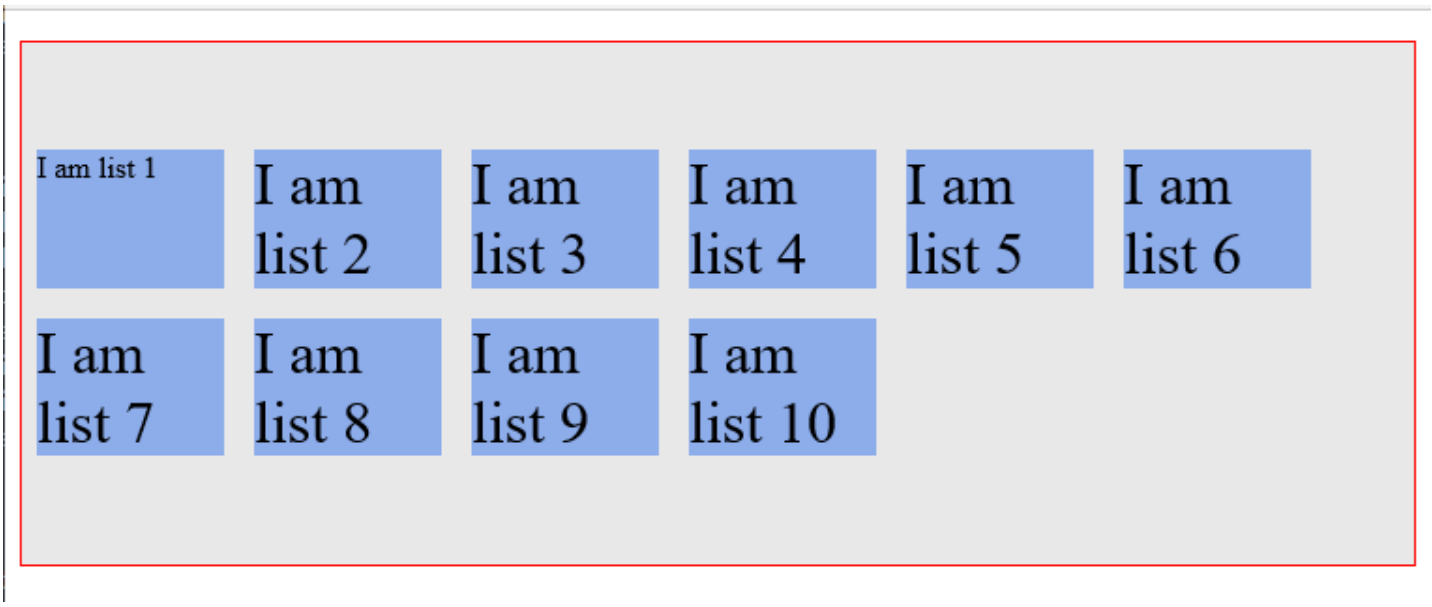
(iii) Flex-end

The `flex-end` value aligns the flex items to the end of the cross-axis.



(iv) Center

Like you may have guessed, `center` aligns the flex-items to the *center* of the cross-axis.



That's the last of the flex-container properties.

You now understand how to use the various flex-container properties. You'll use these to work through the practical sections coming up in the lessons that come.