CSS Flexbox: Planning a Webpage with a Grid-Based Layout

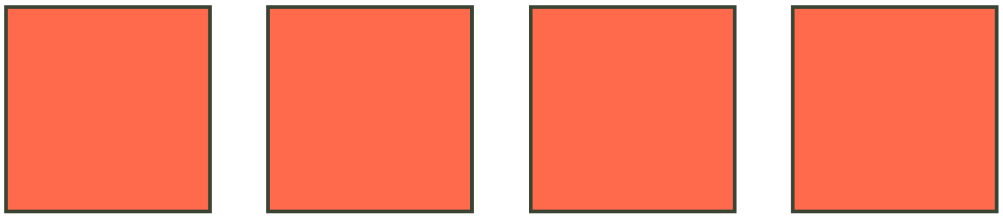
CSS Flexbox Module (or Flexbox for short) defines a CSS box model that is optimized for user interface design. Flexbox is used to layout items on a webpage into rows or columns in one dimension at a time. With this model, items on a page automatically "flex" their sizes by expanding to fill additional space or contracting to fit into smaller spaces. With Flexbox, the alignment of items can be rearranged both horizontally and vertically. When items can flex in this way, this allows websites to adapt to devices of different sizes — including mobile ones — without affecting the user's experience. Therefore, Flexbox allows designers to control the appearance of a webpage and create responsive layouts without the need for positioning, float, or margin offset calculations.

Example: Planning a Grid-Layout with Flexbox

Flexbox is a powerful tool that offers a variety of properties to build responsive websites. An in-depth look at all of Flexbox's capabilities goes beyond the scope of this course. Here, we will focus on a few Flexbox properties to introduce that can be used to design simple responsive layouts.

Let's try creating a grid-based system for a 4-column layout with Flexbox. A 4-column system provides several layout possibilities, while at the same time keeping the math easy.

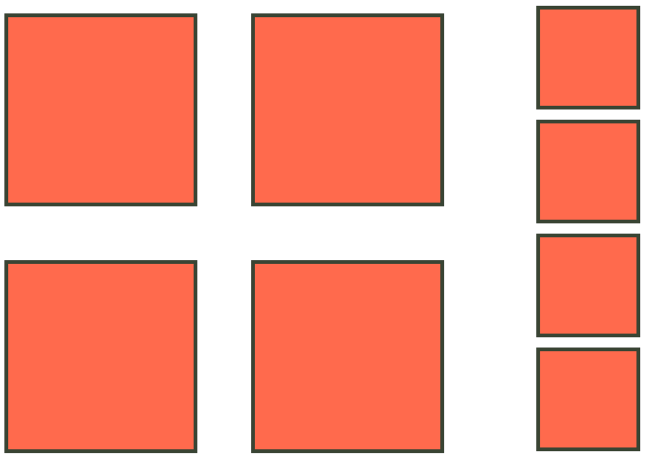
First, let's conceptualize the possible layouts we might have in a 4-column system. To start, consider a single row of 4 boxes on a desktop webpage as illustrated below.



Desktop

How will these boxes display when the screen gets narrower?

For a tablet dimension screen, we'll change to the layout display to the configuration on the left, and for a mobile device, we'll change the layout display to the configuration on the right. Notice how the boxes in the layout shift and rearrange depending on the screen size.



Tablet Device **Mobile Device**

Calculating the Flexbox Code

Now that we have conceptualized possible layouts for our 4-column grid system, we next need to assign some widths to these boxes with the **flex-basis property**.

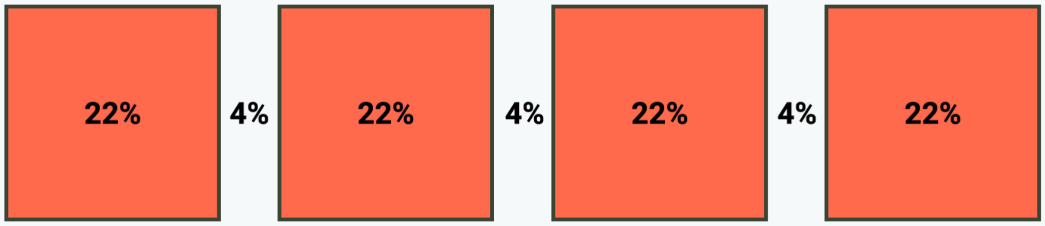
Calculations for a Desktop Display

Let's start with calculating a 4-column design for a desktop display first. Remember, the display is equal to 100%, which is as wide as the screen in use. Therefore, in a 4-column layout, there are four boxes, so each box could be 25% wide (100 divided by 4 equals 25).

However, we also need to figure in a bit of margin (or space) between the boxes. In a 4-column layout, there are three spaces between the four boxes. Therefore, in addition to the four boxes, we also need to account for these three spaces, and all of these elements together must add up to 100%.

So, let's assume that for this example, each box is 22% wide. To calculate how much space the four boxes use, we multiply 22% by four boxes, which equals 88%. Therefore, 88% of the space is occupied by the boxes, with 12% leftover for margin. If we divide the space for margin evenly between the three spaces between boxes, we get 4% per margin.

The graphic below illustrates the 4-column layout with labels that define the space that each element occupies.



$$22\% + 4\% + 22\% + 4\% + 22\% + 4\% + 22 = 100\%$$

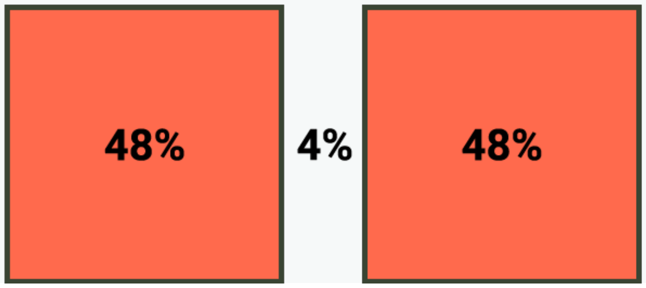
Calculations for a Tablet Display

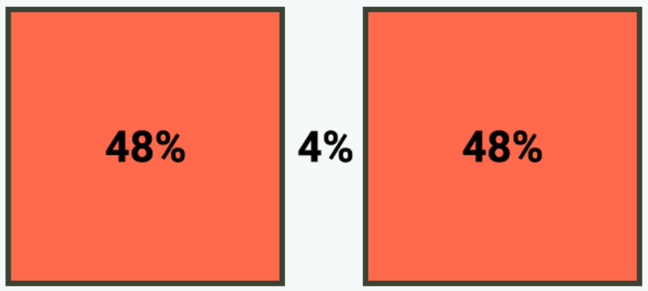
With our desktop calculations complete, let's move to the tablet layout. Remember that each row must still add to 100%, and the space between columns should still be 4%, so everything will align with the previous version of the layout.

Remember, due to a tablet's different screen dimensions, the display has two rows, with two boxes per row. Each box covers two columns. How does this change our Flexbox code calculations?

For the first box in the first row, what is its width? Many will say 44%, because it's the width of the two boxes added together, 22% + 22%. However, you need to *also add the space between the two boxes*. Therefore, the correct calculation is 22% + 22% + 4% = 48%.

The new configuration looks like this:

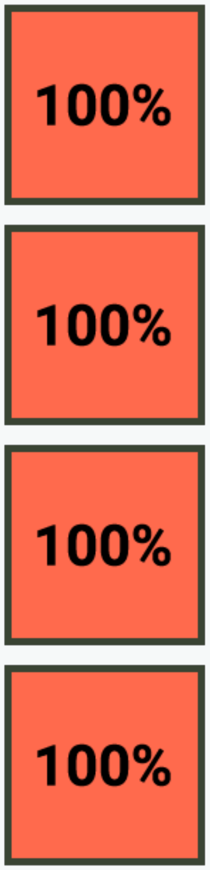




$$48\% + 4\% + 48\% = 100\%$$

Calculations for a Mobile Display

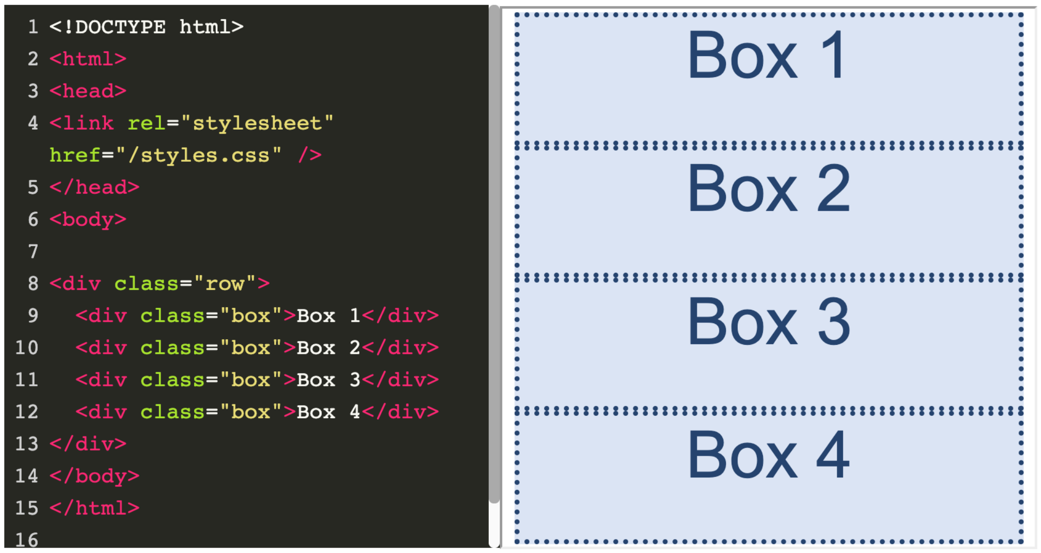
And finally, for the mobile layout, each of the boxes can stack on top of each other, with each box occupying 100% of the width of the row.



Example: Applying Calculations to a Desktop Webpage

Now that we have an understanding of how the math calculated, let's configure a 4-column display using Flexbox. We'll start by setting up a desktop layout in Flexbox. The other layouts will depend on media queries, which will be discussed in a later assignment.

Below is an example of four boxes on a page.



```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet"
href="/styles.css" />
</head>
<body>

<div class="row">
  <div class="box">Box 1</div>
  <div class="box">Box 2</div>
  <div class="box">Box 3</div>
  <div class="box">Box 4</div>
</div>
</body>
</html>

```

Set the Display Property

The first step is to layout the boxes with Flexbox into a row. This is accomplished by setting the display property (**display: flex**) on the flex container (in this example .row) in the CSS stylesheet.

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

Add the Flex Container Properties

For this example, we'll use the Flexbox container properties **flex-flow: row wrap;** and the **justify-content: space-between;** to set up the direction of the boxes and some behaviors.

The flex-flow: row wrap; property defines that the flexible items should wrap as rows. The justify-content: space-between; property defines that the space should be distributed evenly between the boxes, and manages the space in between the boxes, pushing the first and last box to the edge of the row.

```
.row {
display: flex;
flex-flow: row wrap;
justify-content: space-between;
   }
```

Add the Flex Item Properties

Next, we want to set the width of the columns in our layout to 22%. To do this, we use the **flex-basis** property. The flex-basis property defines the initial width of a flexible item.

Therefore, the final code should look like this:

```
.row {
display: flex;
flex-flow: row wrap;
justify-content: space-between;
   }
.box {
flex-basis: 22%;
   }
```

Wait! Why didn't we set a 4% margin? We don't need it. Flexbox will manage the space between the boxes by itself.

Give it a Try!

Instructions:

1. Click the 'run code' button in the code tool below and take note of the browser display
2. Copy the CSS .row properties containing the Flexbox code from above, and paste it into the CSS stylesheet in the code tool below
3. Click the 'run code' button again and notice how the display changes

Remember to be mindful of flex containers (parents) and flex items (children) when applying your properties!

Run Code Save Export Reset

index.html styles.css

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="/styles.css" />
  <style> <!-- code below this line is not involved in web page layout; presentationa
only -->
    .box {
  font-family: Arial, sans-serif;
  height: 90px;
  border: 4px dotted #24436e;
  background-color: #dae4f3;
  font-size: 3rem;
  text-align: center;
  color: #24436e;
}
  </style>
```

Theme Select:

rubyblue

Exercise: 3-Column Grid Layout with CSS Flexbox

Let's write the Flexbox properties for a 3-column grid layout. For this exercise, we'll use the **flex-flow: row wrap;** and the **justify-content: space-between;** properties to lay out our boxes. For this layout, we want a 5% margin.

Instructions:

- Click 'run code' and take note of the browser display
- Calculate the Flexbox code based on a 5% margin

- Click on the css.styles tab and add the appropriate css and Flexbox properties for this 3-column grid layout to the stylesheet.
- Click 'run code' when finished

Run Code Save Export Reset

index.html styles.css

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="/styles.css" />
  <style> <!-- code below this line is not involved in web page layout; presentational only -->
    .box {
  font-family: Arial, sans-serif;
  height: 200px;
  border: 4px dotted #0e786a;
  background-color: #8bfced;
  font-size: 3rem;
  text-align: center;
  color: #04332d;
}
  </style>
```

Theme Select:

rubyblue

Click on the following tab to check your answer.

Check Your Answer

First, we must lay out the boxes with Flexbox into a row by adding the flex properties to the CSS stylesheet.

```
.row {
display: flex;
flex-flow: row wrap;
justify-content: space-between;
   }
```

Then, we must calculate the flex-basis for this 3-column grid layout. The Flexbox calculations are:

The width of the entire screen is 100%.

To account for the margin we subtract 5% from 100% (100 minus 5 equals 90).

Therefore, the space the two boxes can occupy is equal to 90% of the screen width. So the space each box can occupy is 30% (90 divided by 3 equals 30).

Therefore, for this layout the flex-basis property is: **flex-basis: 30%;**

** Remember that Flexbox inherently accounts for the 5% margin

Therefore the complete code that needs to be added to the stylesheet is:

```
.row {
display: flex;
flex-flow: row wrap;
justify-content: space-between;
   }
.box {
flex-basis: 30%;
   }
```

Click 'run code' to see the code render in the browser window of the code tool below.

Run Code Save Export Reset

index.html styles.css

Theme Select:

rubyblue