display: flex

Any element can be a flex container. Flex containers are helpful tools for creating websites that respond to changes in screen sizes. Child elements of flex containers will change size and location in response to the size and position of their parent container.

For an element to become a flex container, its display property must be set to flex.

div.container {  
  display: flex;  
}

In the example above, all divs with the class container are flex containers. If they have children, the children are flex items. A div with the declaration display: flex; will remain block level — no other elements will appear on the same line as it.

However, it will change the behavior of its child elements. Child elements will not begin on new lines. In the exercises that follow, we will cover how the flex display property impacts the positioning of child elements.

inline-flex

In the previous exercise, you might have observed that when we gave a div — a block level element — the display value of flex that it remained a block level element. What if we want multiple flex containers to display inline with each other?

If we didn’t want div elements to be block-level elements, we would use display: inline. Flexbox, however, provides the inline-flex value for the display property, which allows us to create flex containers that are also inline elements.

<div class='container'>  
  <p>I’m inside of a flex container!</p>  
  <p>A flex container’s children are flex items!</p>  
</div>  
<div class='container'>  
  <p>I’m also a flex item!</p>  
  <p>Me too!</p>  
</div>

.container {  
  width: 200px;  
  height: 200px;  
  display: inline-flex;  
}

In the example above, there are two container divs. Without a width, each div would stretch the entire width of the page. The paragraphs within each div would also display on top of each other because paragraphs are block-level elements.

When we change the value of the display property to inline-flex, the divs will display inline with each other if the page is wide enough. As we progress through this lesson, we will cover in more detail how flex items are displayed.

Notice that in the example above, the size of the flex container is set. Currently, the size of the parent container will override the size of its child elements. If the parent element is too small, the flex items will shrink to accommodate the parent container’s size. We’ll explain why in a later exercise.

<div class='container'>  
  <div class='child'>  
    <h1>1</h1>  
  </div>  
  <div class='child'>  
    <h1>2</h1>  
  </div>  
</div>

.container {  
  width: 200px;  
}  
   
.child {  
  display: inline-flex;  
  width: 150px;  
  height: auto;  
}

In the example above, the .child divs will take up more width (300 pixels) than the container div allows (200 pixels). The .child divs will shrink to accommodate the container’s size. In later exercises, we will explore several ways to handle this.

**Instructions**

**1.**

In the .container ruleset, set the display property to inline-flex.

justify-content

In previous exercises, when we changed the display value of parent containers to flex or inline-flex, all of the child elements (flex items) moved toward the upper left corner of the parent container. This is the default behavior of flex containers and their children. We can specify how flex items spread out from left to right, along the main axis. We will learn more about axes in a later exercise.

To position the items from left to right, we use a property called justify-content.

.container {  
  display: flex;  
  justify-content: flex-end;  
}

In the example above, we set the value of justify-content to flex-end. This will cause all of the flex items to shift to the right side of the flex container.

Below are five commonly used values for the justify-content property:

* flex-start — all items will be positioned in order, starting from the left of the parent container, with no extra space between or before them.
* flex-end — all items will be positioned in order, with the last item starting on the right side of the parent container, with no extra space between or after them.
* center — all items will be positioned in order, in the center of the parent container with no extra space before, between, or after them.
* space-around — items will be positioned with equal space before and after each item, resulting in double the space between elements.
* space-between — items will be positioned with equal space between them, but no extra space before the first or after the last elements.

In the definitions above, “no extra space” means that margins and borders will be respected, but no more space (than is specified in the style rule for the particular element) will be added between elements. The size of each individual flex item is not changed by this property.

### Instructions

**1.**

Assign the <div> element with the id flexstart a justify-content property with a value of flex-start.

You won’t see any changes since flex-start is the default value;

align-items

In the previous exercise, you learned how to justify the content of a flex container from left to right across the page. It is also possible to align flex items vertically within the container. The align-items property makes it possible to space flex items vertically.

.container {  
  align-items: baseline;  
}

In the example above, the align-items property is set to baseline. This means that the baseline of the content of each item will be aligned.

Below are five commonly used values for the align-items property:

* flex-start — all elements will be positioned at the top of the parent container.
* flex-end — all elements will be positioned at the bottom of the parent container.
* center — the center of all elements will be positioned halfway between the top and bottom of the parent container.
* baseline — the bottom of the content of all items will be aligned with each other.
* stretch — if possible, the items will stretch from top to bottom of the container (this is the default value; elements with a specified height will not stretch; elements with a minimum height or no height specified will stretch).

These five values tell the elements how to behave along the *cross axis* of the parent container. In these examples, the cross axis stretches from top to bottom of the container. We’ll learn more about this in a future exercise.

You might be unfamiliar with the min-height and max-height properties, but you have used height and width before. min-height, max-height, min-width, and max-width are properties that ensure an element is at least a certain size or at most a certain size. You’ll see how these become useful as you move throughout this lesson.

Now you’re going to see each of the five values above in action!

flex-grow

In Exercise 3, we learned that all flex items shrink proportionally when the flex container is too small. However, if the parent container is larger than necessary then the flex items will not stretch by default. The flex-grow property allows us to specify if items should grow to fill a container and also which items should grow proportionally more or less than others.

<div class='container'>  
  <div class='side'>  
    <h1>I’m on the side of the flex container!</h1>  
  </div>  
  <div class='center'>  
    <h1>I'm in the center of the flex container!</h1>  
  </div>  
  <div class='side'>  
    <h1>I'm on the other side of the flex container!</h1>  
  </div>  
</div>

.container {  
  display: flex;  
}  
   
.side {  
  width: 100px;  
  flex-grow: 1;  
}  
   
.center {  
  width: 100px;  
  flex-grow: 2;  
}

In the example above, the .container div has a display value of flex, so its three child divs will be positioned next to each other. If there is additional space in the .container div (in this case, if it is wider than 300 pixels), the flex items will grow to fill it. The .center div will stretch twice as much as the .side divs. For example, if there were 60 additional pixels of space, the center div would absorb 30 pixels and the side divs would absorb 15 pixels each.

If a max-width is set for an element, it will not grow larger than that even if there is more space for it to absorb.

All of the previous properties we have learned are declared on flex containers, or the parent elements. This property — flex-grow — is the first we have learned that is declared on flex items.