LEARN CSS FLEXBOX BY BUILDING A PHOTO GALLERY

Introduction

Flexbox helps you design your webpage so that it looks good on any screen size. In this course, you'll use Flexbox to build a responsive photo gallery webpage.

Step 1:

Begin with your standard HTML boilerplate. Add a `DOCTYPE` declaration, an `html` element, a `head` element, and a `body` element.

Add the `lang` attribute to the opening `<html>` tag with `en` set as the value.

Step 2:

Within your `head` element, add a `meta` tag with the `name` set to `viewport` and the `content` set to `width=device-width, initial-scale=1`.

Also add a `meta` tag with the `charset` set to `UTF-8`.

Step 3:

Within your `head` element, add a `title` element with the text set to `Photo Gallery`, and a `link` element to add your `styles.css` file to the page.

Step 4:

Add a `header` element within the `body` element and assign a class of `header` to it. Inside the `header`, create an `h1` with `css flexbox photo gallery` as the text.

Step 5:

Below your `.header` element, create a new `div` element and assign it a `class` of `gallery`. This `div` will act as a container for the gallery images.

Inside that `.gallery` element, create nine `img` elements.

Step 6:

Next, give each `img` a `src` attribute according to its order in the document. The first `img` should have a `src` of `https://cdn.freecodecamp.org/curriculum/css-photo-gallery/1.jpg`. The rest should be the same, except replace the `1` with the number the `img` is in the document.

Step 7:

In order to better visualize how your elements are sized, adding a border can be helpful.

Give your `.gallery` element a `width` of `50%` and a `border` set to `5px solid red`.

Then give your `img` elements a `width` of `100%`, `padding` set to `5px`, and a `border` set to `5px solid blue`.

Step 8:

Notice how the blue image border extends beyond the red gallery border. This is due to the way browsers calculate the size of container elements.

The `box-sizing` property is used to set this behavior. By default, the `content-box` model is used. With this model, when an element has a specific width, that width is calculated based only on the element's content. Padding and border values get added to the total width, so the element grows to accommodate these values.

Try setting `box-sizing` to `content-box` explicitly, with the global `*` selector. At this point, you will not see any changes, because you are using the default value.

Step 9:

The `border-box` sizing model does the opposite of `content-box`. The total width of the element, including padding and border, will be the explicit width set. The content of the element will shrink to make room for the padding and border.

Change the `box-sizing` property to `border-box`. Notice how your blue image borders now fit within your red gallery border.

Step 10:

Now that you have figured out your `box-sizing` approach, you can clean up the CSS you added to see the changes.

Remove your `.gallery` and `img` selectors, and all rules within.

Step 11:

Now your images are too big.

Create a `.gallery img` selector to target them. Give them all a `width` of `100%` and a `max-width` of `350px`.

Also set the `height` property to `300px` to keep your images a uniform size.

Step 12:

Remove the margin from your body element, set the `font-family` to `sans-serif`, and give it a `background-color` of `#f5f6f7` as the value.

Step 13:

Align your `.header` text in the center. Make the text uppercase using the `text-transform` property with `uppercase` as the value.

Give it a padding of `32px` on all sides. Set the background to `#0a0a23` and the text to `#fff` as the color values.

Add a `border-bottom` with `4px solid #fdb347` as the value.

Step 14:

Flexbox is a one-dimensional CSS layout that can control the way items are spaced out and aligned within a container.

To use it, give an element a `display` property of `flex`. This will make the element a *flex container*. Any direct children of a flex container are called *flex items*.

Create a `.gallery` selector and make it a flex container.

Step 15:

Flexbox has a main and cross axis. The main axis is defined by the flex-direction property, which has four possible values:

- `row (default)`: horizontal axis with flex items from left to right
- `row-reverse`: horizontal axis with flex items from right to left
- `column:` vertical axis with flex items from top to bottom
- `column-reverse`: vertical axis with flex items from bottom to top

Note: The axes and directions will be different depending on the text direction. The values shown are for a left-to-right text direction.

Try the different values to see how they affect the layout.

When you are done, set an explicit `flex-direction` of `row` on the `.gallery` element.

Step 16:

The `flex-wrap` property determines how your flex items behave when the flex container is too small. Setting it to `wrap` will allow the items to wrap to the next row or column. `nowrap` (default) will prevent your items from wrapping and shrink them if needed.

Make it so your flex items wrap to the next row when they run out of space.

Step 17:

The `justify-content` property determines how the items inside a flex container are positioned along the main axis, affecting their position and the space around them.

Give your `.gallery` selector a `justify-content` property with `center` as the value.

Step 18:

The `align-items` property positions the flex content along the cross axis. In this case, with your `flex-direction` set to `row`, your cross axis would be vertical.

To vertically center your images, give your `.gallery` selector an `align-items` property with `center` as the value.

Step 19:

Give your `.gallery` selector a `padding` property set to `20px 10px` to create some space around the container.

Then, give it a `max-width` of `1400px` and add a `margin` of `0 auto` to center it.

Step 20:

Notice how some of your images have become distorted. This is because the images have different aspect ratios. Rather than setting each aspect ratio individually, you can use the `object-fit` property to determine how images should behave.

Give your `.gallery img` selector the `object-fit` property and set it to `cover`. This will tell the image to fill the `img` container while maintaining aspect ratio, resulting in cropping to fit.

Step 21:

Your images need some space between them.

The `gap` CSS shorthand property sets the gaps, also known as gutters, between rows and columns. The `gap` property and its `row-gap` and `column-gap` sub-properties provide this functionality for flex, grid, and multi-column layout. You apply the property to the container element.

Give your `.gallery` flex container a `gap` property with `16px` as the value.

Step 22:

Smooth out your images a bit by giving the `.gallery img` selector a `border-radius` property with `10px` set as the value.

Step 23:

The `::after` pseudo-element creates an element that is the last child of the selected element. You can use it to add an empty element after the last image. If you give it the same `width` as the images it will push the last image to the left when the gallery is in a two-column layout. Right now, it is in the center because you set `justify-content: center` on the flex container.

Example Code:

```
.container::after {

  content: "";

  width: 860px;

}
```

Create a new selector using an `::after` pseudo-element on the `.gallery` element. Add a `content` property set to an empty string `""` and `350px` set for the `width` property.

Step 24:

The `alt` image attribute should describe the image content. Screen readers announce the alternative text in place of images. If the image can't be loaded, this text is presented in place of the image.

To complete the project, add an `alt` attribute to all nine of your cat images to describe them. Use a value at least five characters long for each.