CSS Flexbox exercise

This exercise is divided into two distinct parts.

- *Part One* is a collection of exercises that test your knowledge of flex container properties and flex items properties. There is a series of individual "mini-exercises" for you to complete.
- *Part Two* gives you the chance to test your Flexbox knowledge by building a complete page, using CSS Flexbox to style a photography portfolio.

Learning objectives

After completing this exercise, you'll understand:

- How to use CSS Flexbox to layout items on a page.
- How changes to specific flex properties affect the look of your page.

Evaluation criteria and functional requirements

- Your page displays using Live Server within Visual Studio Code.
- Your page displays flex items in a way similar to the page images included in this document.
- All tests pass as expected.

Getting started

1. Open the `exercise` folder in VS Code.

2. If this is the first time you have viewed this exercise, in a VS Code terminal window, type:

```
npm install
```

This prepares the testing software (Cypress) to run, and it may take a few minutes to complete.

3. Run the tests at any time using one of the following commands in a terminal window:

- `npm run test` - Opens the Cypress UI, which allows you to run tests and view results interactively. Note that Cypress is slow to open - it may take a minute or so before you see the Cypress user interface.

- `npm run test-headless` - Doesn't open the Cypress UI, but runs the tests and displays results in the terminal window. This is a quicker way to run tests.

4. Follow the steps for each set of exercises in the instructions that follow. After each part, be sure to run the tests to see your progress. You can rerun tests at any time.
5. Repeat until all tests are passing.

Part One: Practice with flexbox properties

This is a collection of exercises that test your knowledge of flex container and flex item properties.

Getting started

1. In VS Code, navigate to the `exercise-part-1` folder.
2. Right-click `index.html` and select **Open with Live Server** to view your page. This page links to the pages you edit for all the following steps.

Set One: Adjust header layout

In the header exercises, you'll align several navigation items. This type of construct is often seen near the top of HTML files to help the user navigate the site.

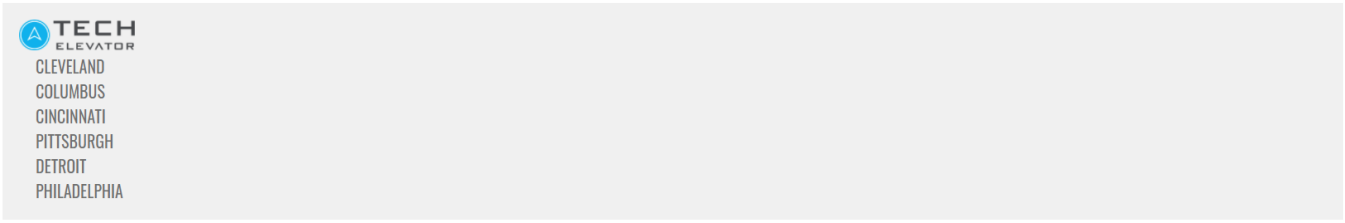
In VS Code, look in the `set-1-headers` subfolder. In this folder, there are five subfolders, `step-1` through `step-5`. Each subfolder contains files `index.html` and `style.css`.

You'll make changes in the corresponding `style.css` file in each step. *Do not change the HTML files*.

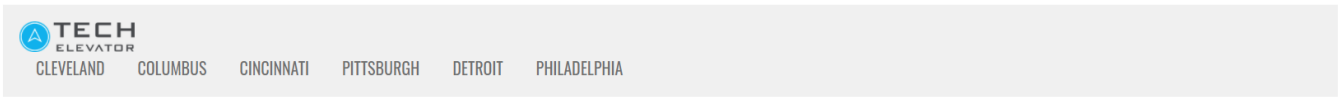
> **Note**: All of your work for this exercise must be in each `style.css`. You must not modify the contents of any HTML file to complete this exercise.

Step One: Align navigation items

Initially, this page displays all items listed down the page, like this:



Your task is to use flexbox to display the navigation items (the locations) in a row across the page, like this:

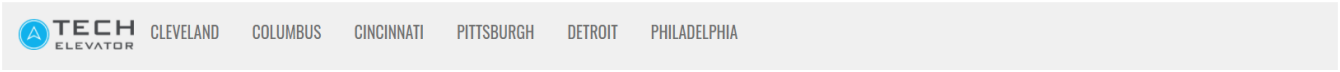


Look at the structure of the HTML in `index.html`, and decide which element needs to be changed to a flex container. Remember that the *direct children* of the flex container are the flex items. You want each of the list items (`<li>`) to be flex items, so you want to change the parent element to a flex container.

- **Action**: Edit `step-1\style.css` to apply the appropriate styling.

Step Two: Align the logo and navigation items on the same row

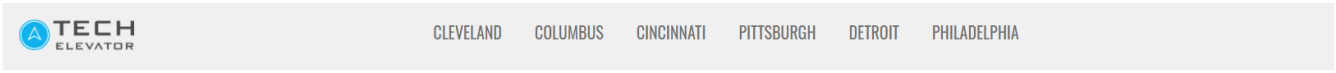
Notice that your change in step one didn't include the logo. Your task in this step is to make all items appear on one line, so add another flex container, incorporating the logo and the `<nav>` as flex items.



- **Action**: Edit `step-2\style.css` to apply the styling you applied in step one.
- **Action**: Add another flex container to put all items on one row.

Step Three: Center the navigation items in the space after the logo

In this step, you must keep the logo on the left side, but in the remaining horizontal space, center the navigation items, like this:



- **Action**: Edit `step-3\style.css` to apply the styling you applied in the previous steps.
- **Action**: Apply the styling needed to center the navigation items.

  Hint: Make sure the `<nav>` grows to take up the full row before you align its items.

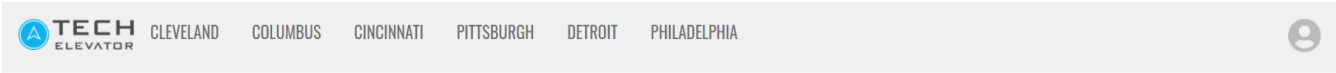
Step 4: Right-align the navigation items

In this step, the logo must stay on the left side, and the navigation items must align themselves from the right edge of the page, like this:

- **Action**: Edit `step-4\style.css` to apply the styling you applied in the previous steps.
- **Action**: Modify one CSS value to get the styling you need.

Step 5: Right-align the user icon

On some sites, you may have seen a user icon on the right edge of the screen, allowing you to manage your user account. Your task in this step is to align the navigation items on the left side, next to the logo, and the user icon on the right side, like this:



Notice that the HTML included in this step has two additional elements which are children of the `header` element—`div.spacer` and `a.my-account`. The latter of these is the element that must appear on the right edge of the screen.

- **Action**: Edit `step-5\style.css` to apply the styling you applied in step two.
- **Action**: Right-align the user icon by setting `.spacer` to take up the rest of the available space.

After you complete this step, the tests in **Part One: Practice with flexbox properties > Set One: Adjust header layout** pass.

Set Two: Align flex items within a row

In this set of exercises, you'll align several flex items within a row. In VS Code, look in the `set-2-align-content-row` subfolder.

In this folder, there are two subfolders, `step-1` and `step-2`. Each subfolder contains files `index.html` and `style.css`.

You'll make changes in the corresponding `style.css` file in each step. *Do not change the HTML files*.

Step One: Align items with equal space on all sides

Initially, this page displays all items listed down the page, like this:

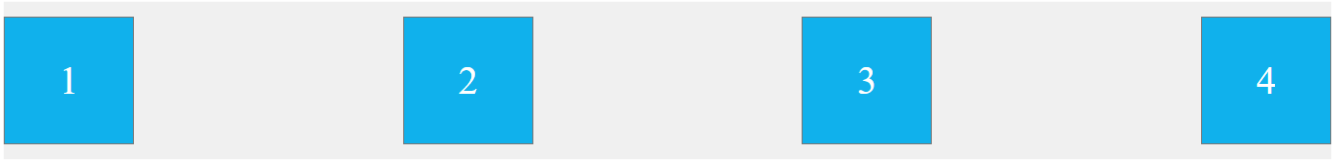
Your task is to use flexbox to display the boxes in a row with equal space on each side of the boxes, like this:

Look at the structure of the HTML in `index.html`, and decide which element needs to be changed to a flex container to make the boxes (`div.box`) flex items. Then use the appropriate flex properties to space the boxes as in the diagram.

- **Action**: Edit `step-1\style.css` to apply the appropriate styling.

Step Two: Align items with equal space between

In this step, your task is to use flexbox to display the boxes in a row with the first and last boxes at the edges, and equal space between the boxes, like this:



- **Action**: Edit `step-2\style.css` to apply the appropriate styling.

After you complete this step, the tests in **Part One: Practice with flexbox properties > Set Two: Align flex items within a row** pass.

Set Three: Align flex items vertically

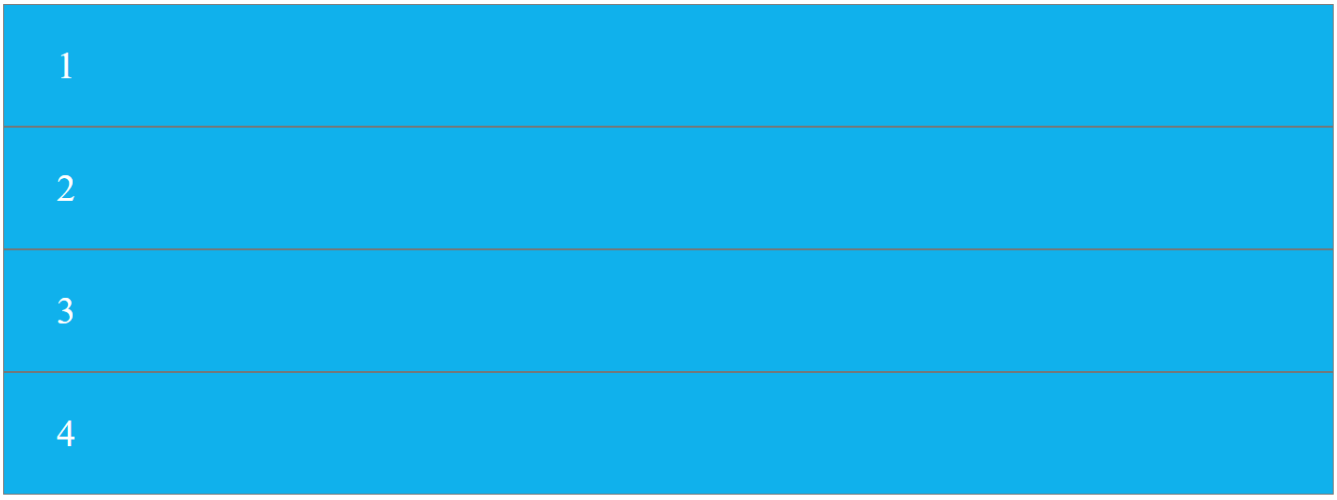
In this set of exercises, you'll align a row of flex items in a vertical orientation.

In VS Code, look in the `set-3-align-vertically` subfolder. In this folder, there are four subfolders, `step-1` through `step-4`. Each subfolder contains files `index.html` and `style.css`.

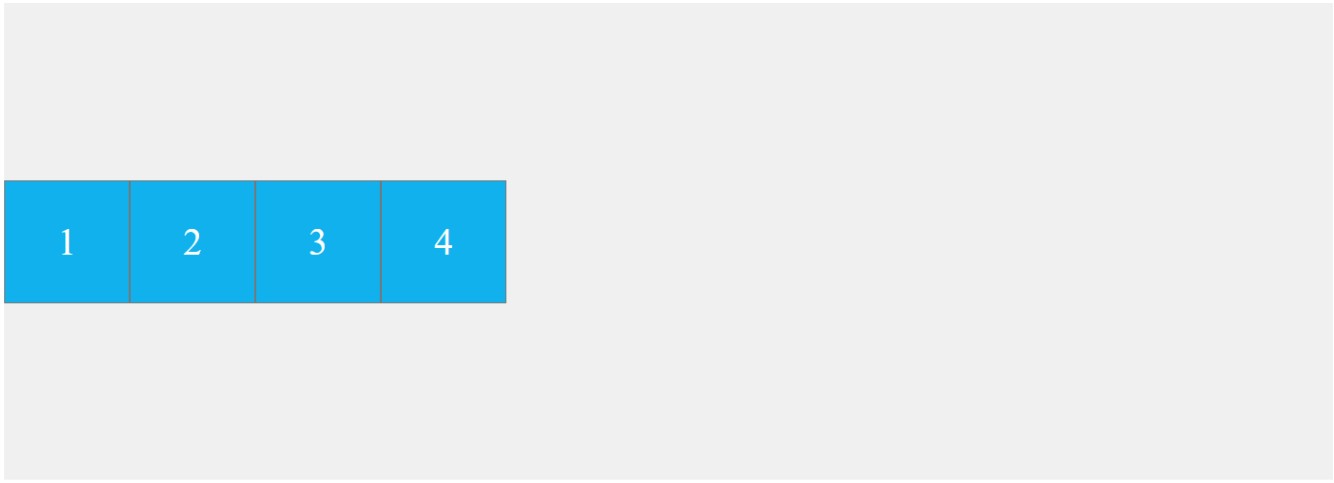
You'll make changes in the corresponding `style.css` file in each step. *Do not change the HTML files*.

Step One: Center the row of items vertically

Initially, this page doesn't have a flexbox. `div` elements (the boxes) take up the full row, so items display listed down the page, like this:



The container height is set to 500 pixels. Your task is to use flexbox to display the boxes in a row, and *vertically center* that row within the height of the container, like this:

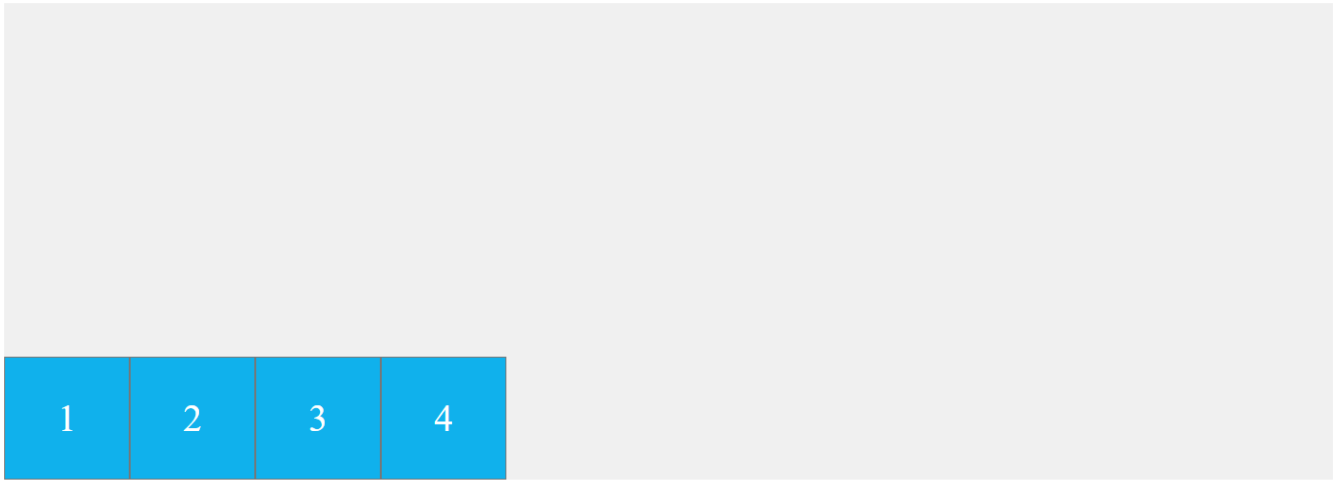


Look at the structure of the HTML in `index.html`, and decide which element needs to be changed to a flex container to make the boxes (`div.box`) flex items. Then use the appropriate flex properties to align the boxes as in the diagram.

- **Action**: Edit `step-1\style.css` to make the container of the boxes into a row-oriented flexbox.
- **Action**: Apply flexbox properties to vertically center the row of boxes within the container.

Step Two: Place the row at the bottom of the container

In this step, your task is to use a flexbox property to display the row of boxes *along the bottom edge* of the container, like this:



Look at the structure of the HTML in `index.html`, and decide which element needs to be changed to a flex container to make the boxes (`div.box`) flex items. Then use the appropriate flex properties to align the boxes as in the diagram.

- **Action**: Edit `step-2\style.css` to make the container of the boxes into a row-oriented flexbox.
- **Action**: Apply flexbox properties to display the row of boxes along the bottom edge of the container.

Step Three: Place the row at the top of the container

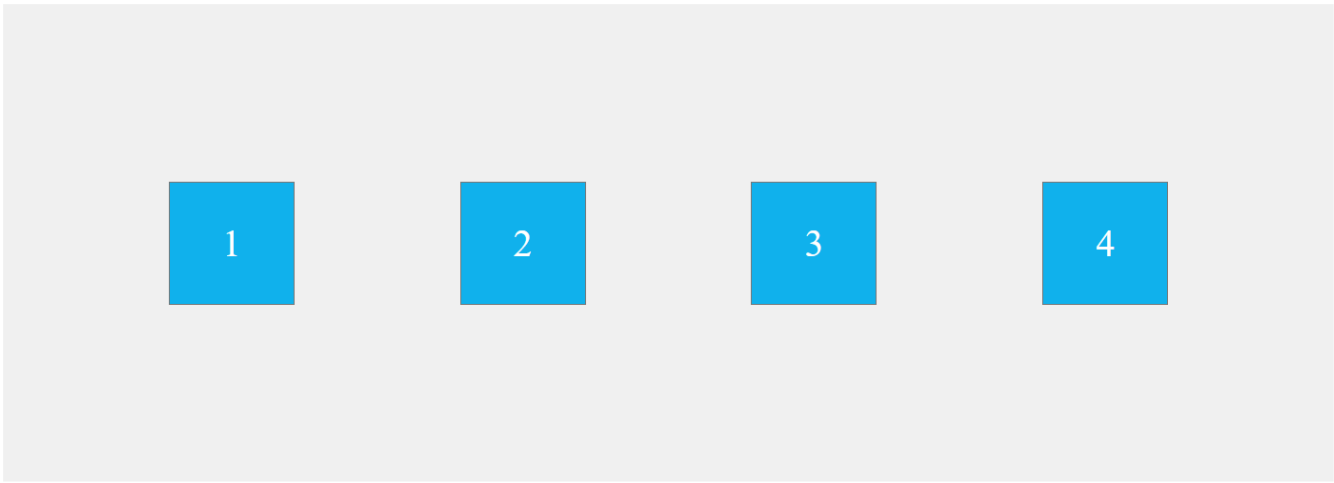
In this step, your task is to use a flexbox property to display the row of boxes *along the top edge* of the container, like this:

Look at the structure of the HTML in `index.html`, and decide which element needs to be changed to a flex container to make the boxes (`div.box`) flex items. Then use the appropriate flex properties to align the boxes as in the diagram.

- **Action**: Edit `step-3\style.css` to make the container of the boxes into a row-oriented flexbox.
- **Action**: Apply flexbox properties to display the row of boxes along the top edge of the container.

Step Four: Set vertical and horizontal alignment

In this step, your task is to use flexbox properties to display the row of boxes *vertically in the center* of the container, and spread across the row so that there is equal space on all sides, like this:



Look at the structure of the HTML in `index.html`, and decide which element needs to be changed to a flex container to make the boxes (`div.box`) flex items. Then use the appropriate flex properties to align the boxes as in the diagram.

- **Action**: Edit `step-4\style.css` to make the container of the boxes into a row-oriented flexbox.
- **Action**: Apply flexbox properties to vertically center the row of boxes within the container, and spread across the row so that there is equal space on all sides.

After you complete this step, the tests in **Part One: Practice with flexbox properties > Set Three: Align flex items vertically** pass.

Set Four: Size flex items

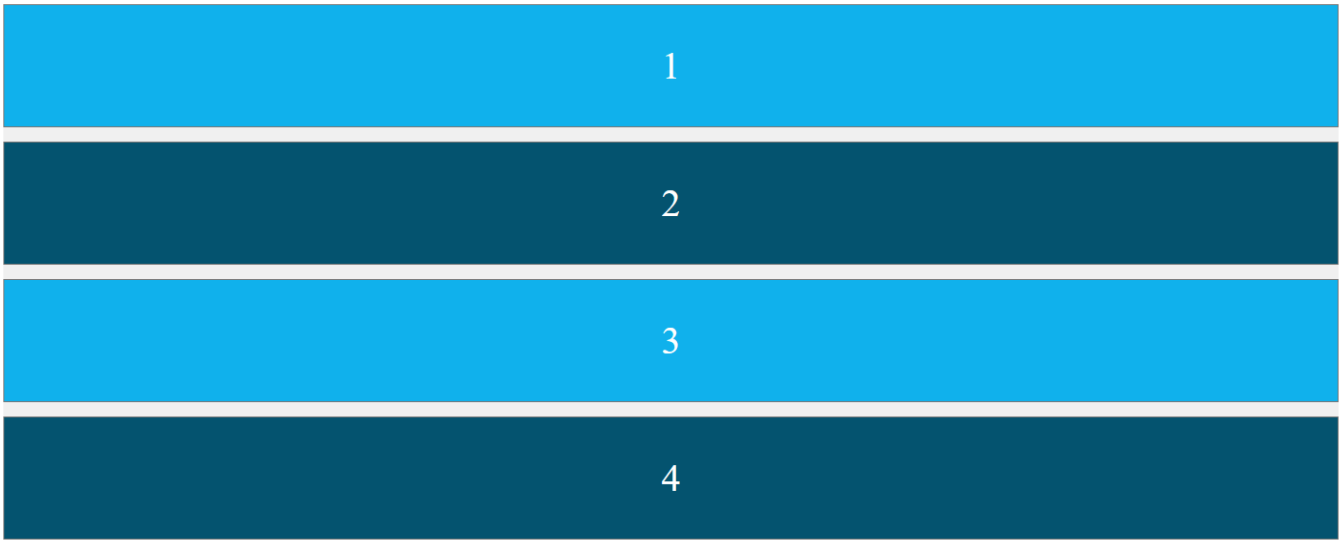
In this set of exercises, you'll size several flex items with the `flex-grow` property.

In VS Code, look in the `set-4-flex-grow` subfolder. In this folder, there are two subfolders, `step-1` and `step-2`. Each subfolder contains files `index.html` and `style.css`.

You'll make changes in the corresponding `style.css` file in each step. *Do not change the HTML files*.

Step One: Grow items to fill the row

Initially, this page doesn't have a flexbox. `div` elements (the boxes) take up the full row, so items display listed down the page, like this:



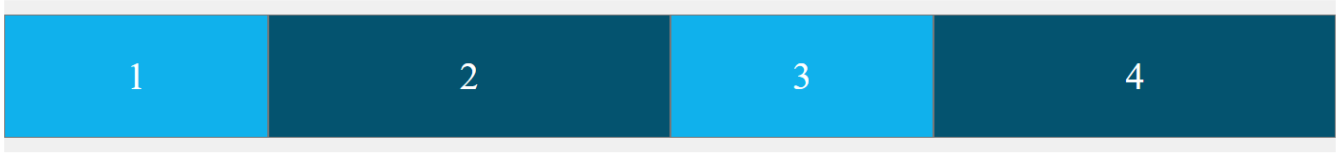
Your task is to use flexbox to display the boxes in a row, such that each item grows an equal amount to consume the entire width of the container, like this:

Look at the structure of the HTML in `index.html`, and decide which element needs to be changed to a flex container to make the boxes (`div.box`) flex items. Then use the appropriate flex properties to grow the boxes as in the diagram.

- **Action**: Edit `step-1\style.css` to make the container of the boxes into a row-oriented flexbox.
- **Action**: Apply flexbox properties to make every item grow an equal amount.

Step Two: Grow items two and four even more

In this step, your task is to use a flexbox property to allow boxes two and four to grow twice as much as boxes one and three. All four boxes grow to fill the width of the container, but boxes two and four grow at a faster rate. The result looks like this:



Look at the structure of the HTML in `index.html`, and decide which element needs to be changed to a flex container to make the boxes (`div.box`) flex items. Then use the appropriate flex properties to grow the boxes as in the diagram.

- **Action**: Edit `step-2\style.css` to make the container of the boxes into a row-oriented flexbox.
- **Action**: Apply flexbox properties to make items two and four grow twice as much as items one and three.

After you complete this step, the tests in **Part One: Practice with flexbox properties > Set Four: Size flex items** pass.

Set Five: Wrap flex items

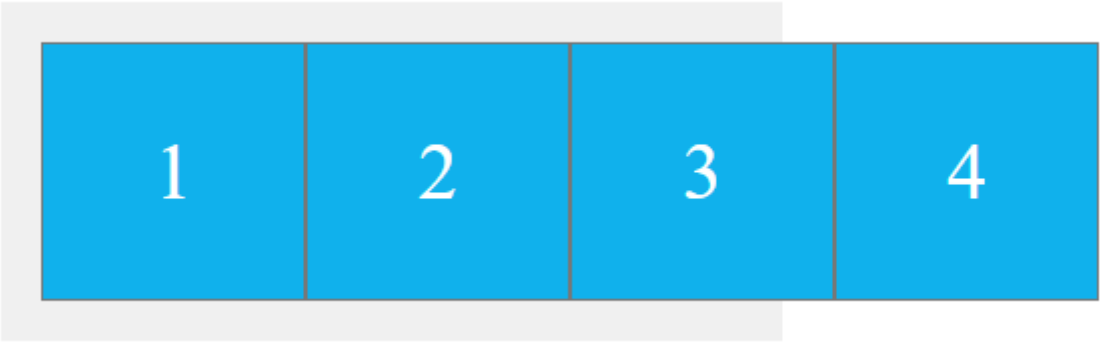
In this set of exercises, you'll wrap flex items when their width exceeds that of their container.

In VS Code, look in the `set-5-flex-wrap` subfolder. In this folder, there is a single subfolder, `step-1`, which contains files `index.html` and `style.css`.

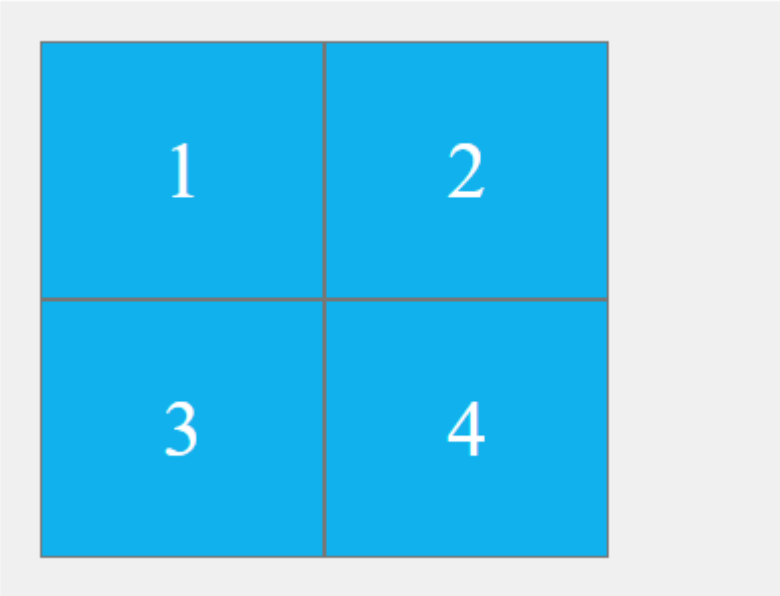
You'll make changes in the corresponding `style.css` file in each step. *Do not change the HTML files*.

Step One: Wrap flex items

Initially, this page is set up to have a flexbox. By default, all flex items are forced onto one line, and the natural width boxes causes them to "spill" out of the container, like this:



Your task is to use a flexbox so the items wrap once they hit the edge of the container, like this:



- **Action**: In `step-1/style.css`, apply a flexbox property so the items wrap once they hit the edge of the container.

After you complete this step, the test in **Part One: Practice with flexbox properties > Set Five: Wrap flex items** passes.

Set Six: Order flex items

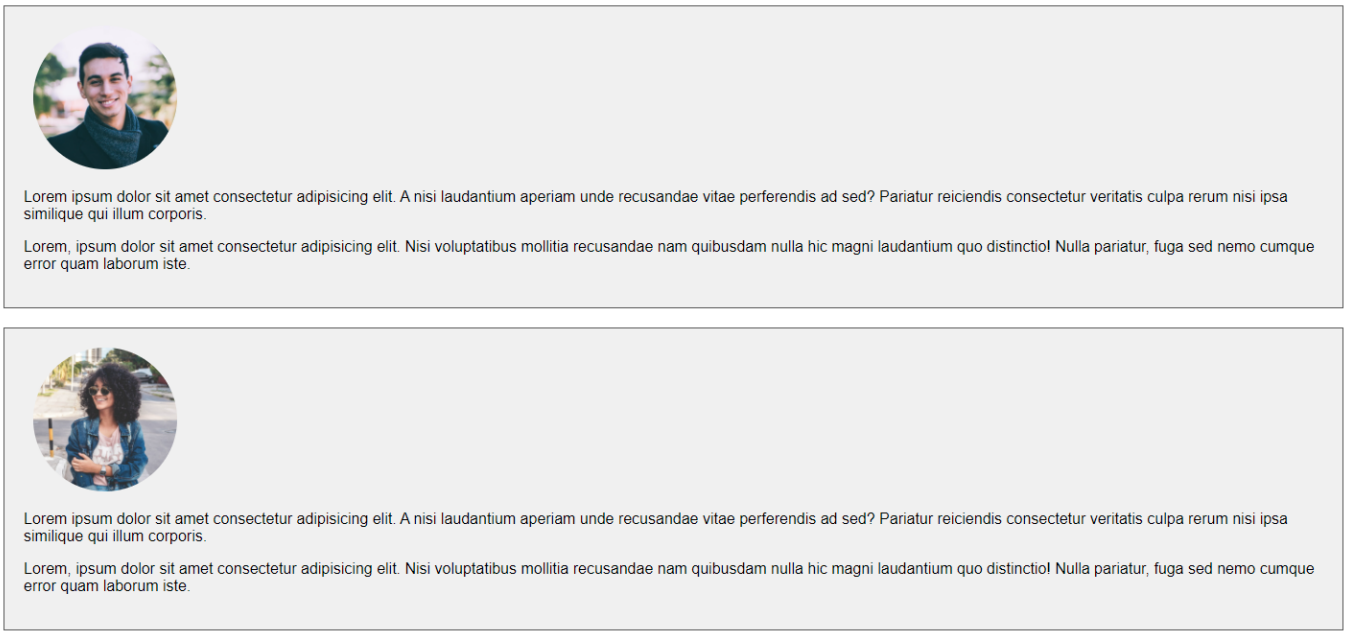
In this set of exercises, you'll reorder elements on a page without changing the HTML markup.

In VS Code, look in the `set-6-flex-order` subfolder. In this folder, there is a single subfolder, `step-1`, which contains files `index.html` and `style.css`.

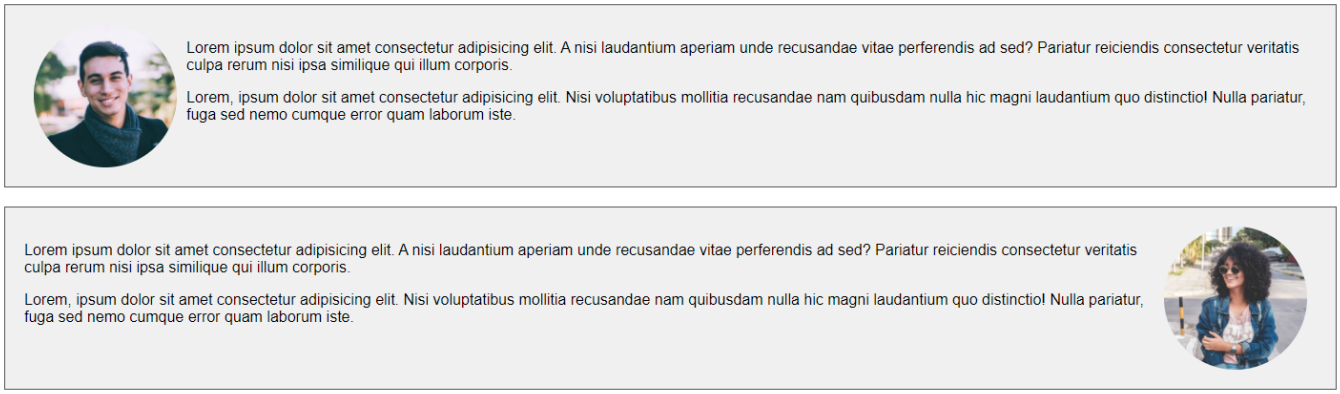
You'll make changes in the corresponding `style.css` file in each step. *Do not change the HTML files*.

Step One: Order flex items

Initially, this page doesn't have a flexbox, so block elements take up the full row, like this:



Your task is to find the appropriate container to change into a flexbox, so the image and text are displayed next to one another. Then for the *second item*, use a flexbox property to change the order the items are display, so the image appears to the right of the text. The result looks like this:



- **Action**: Edit `step-1\style.css` determine the elements to make into a row-oriented flexbox.
- **Action**: Apply flexbox properties to make the image appear after the text in the second group.

After you complete this step, the tests in **Part One: Practice with flexbox properties > Set Six: Order flex items** pass.

Part Two: Build a portfolio page with flexbox

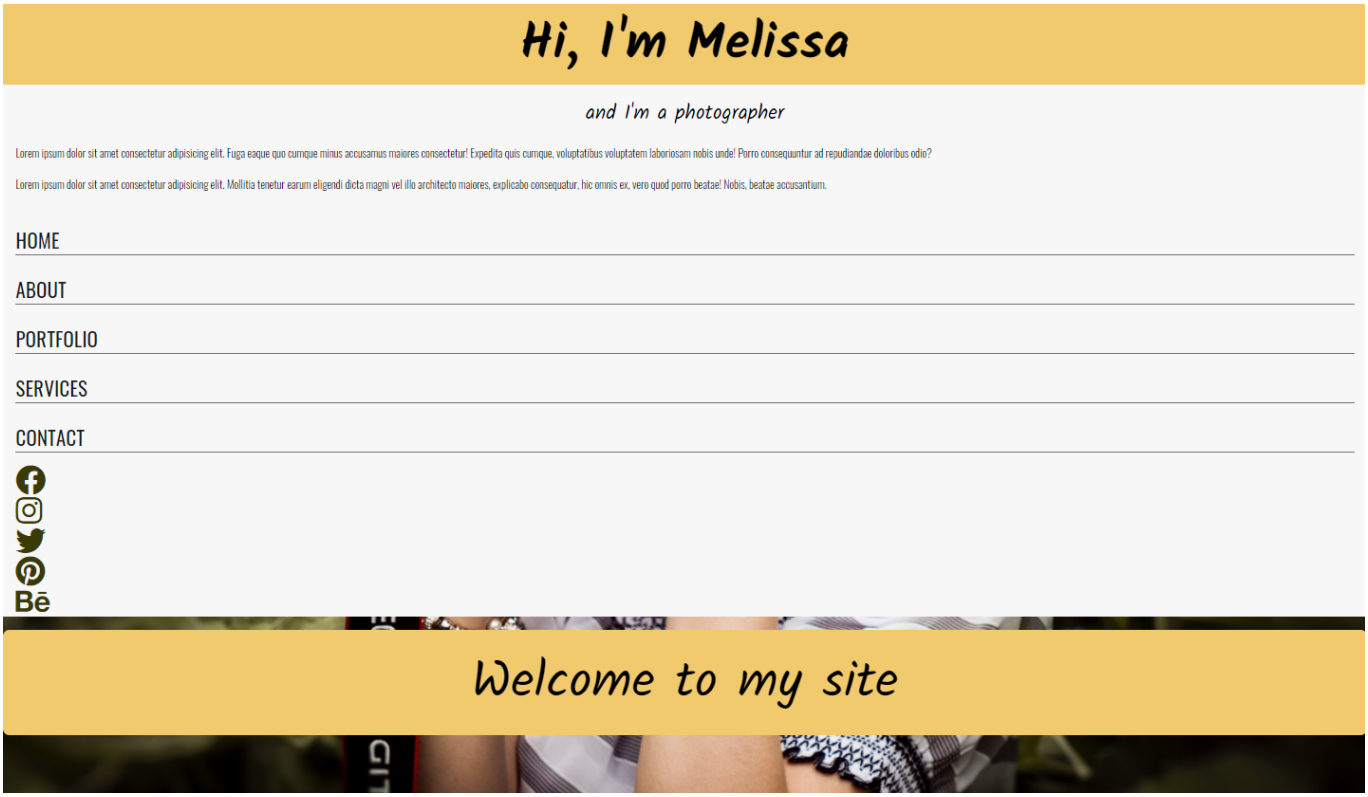
This exercise gives you the chance to test your Flexbox knowledge by building a photography portfolio.

Getting started

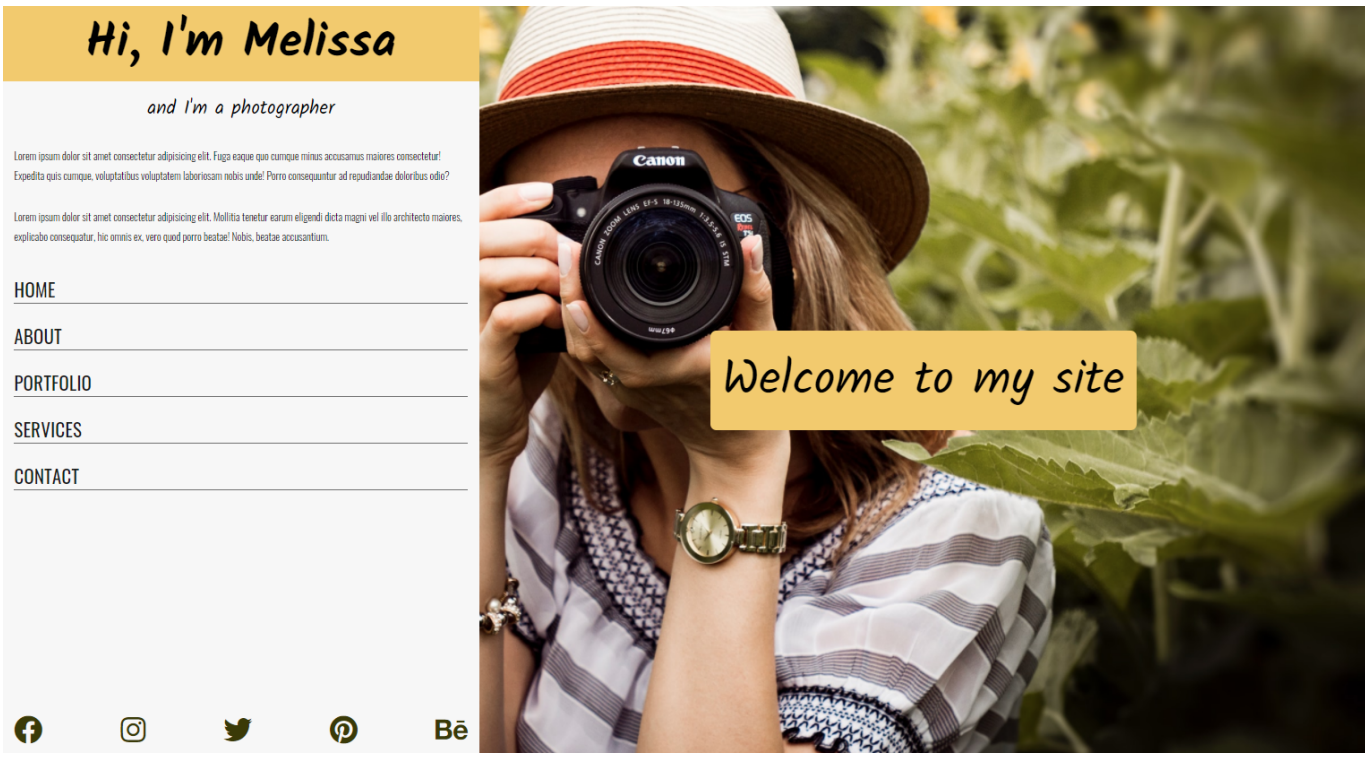
1. In VS Code, navigate to the `exercise-part-2` folder.
2. Right-click `index.html` and select **Open with Live Server** to view your page. This displays your portfolio page.
3. Change the CSS in `css/style.css`.

> **Note**: All of your work for this exercise must be in `style.css`. You must not modify the contents of `index.html` to complete this exercise.

When you first run the project, it looks like this:



If you complete all of the steps in this exercise, your solution looks like this:

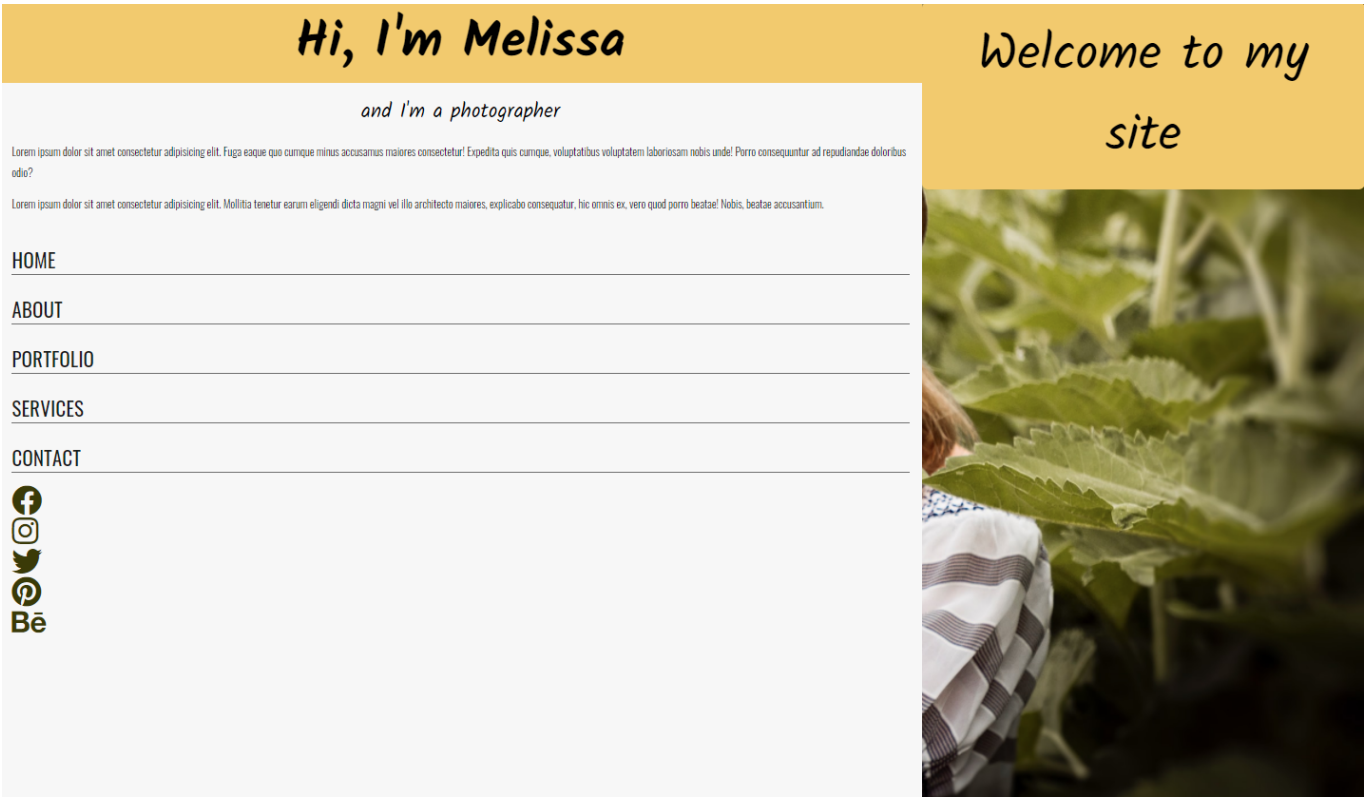


Step One: Define the overall layout

The layout for this page consists of two panels: a left panel and a right panel. The left panel contains a bio, navigation, and social media links. The right panel displays an image and a welcome banner. The body style

pushes the layout to full screen using `min-height: 100vh`.

You need to place `section#left-panel` and `section#right-panel` next to one another in the same row. What property do you need to add to their parent element?

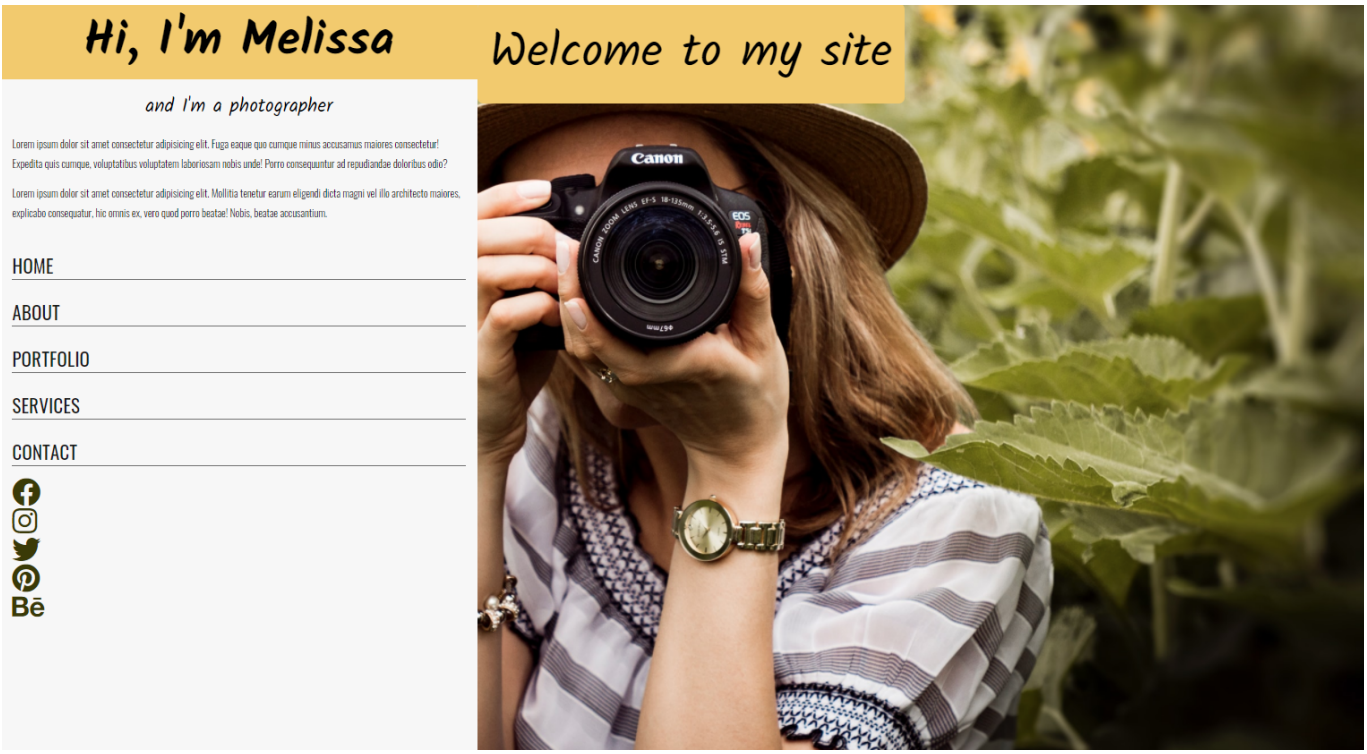


- **Action**: Edit `css/style.css` and add the appropriate CSS declaration to make the panels appear next to one another.

Step 2: Add flex layout to the left panel

The left-panel is wide because of its text. You want the left panel restricted to 35% of the width of the page. You also must make the left-panel into a column-oriented flexbox so that you have control over placement of its child elements (`header`, `h2`, `p`, `nav` , and `footer`). The result looks like this:

Add a flex layout to the left panel so the items stack on top of each other and take up a width (basis) of 35%:

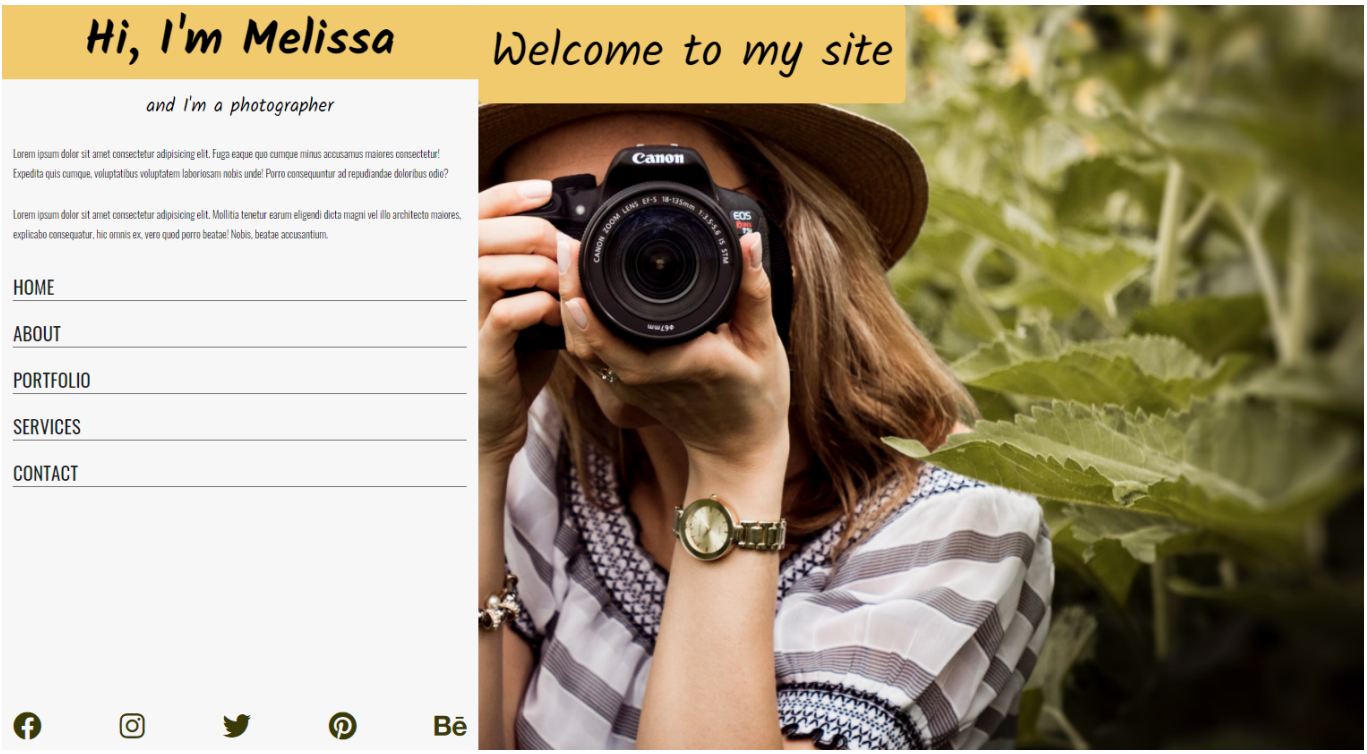


- **Action**: Add a property to the left panel so it takes up a width (basis) of 35%.
- **Action**: Add CSS to make the `section#left-panel` into a column-oriented flexbox.

Step 3: Add styles to social icons

Next, you need to add styles to the element that contains all the social icons, so the icons line up on the same row and stretch across the width of the left-panel and have the same amount of space between each of them.

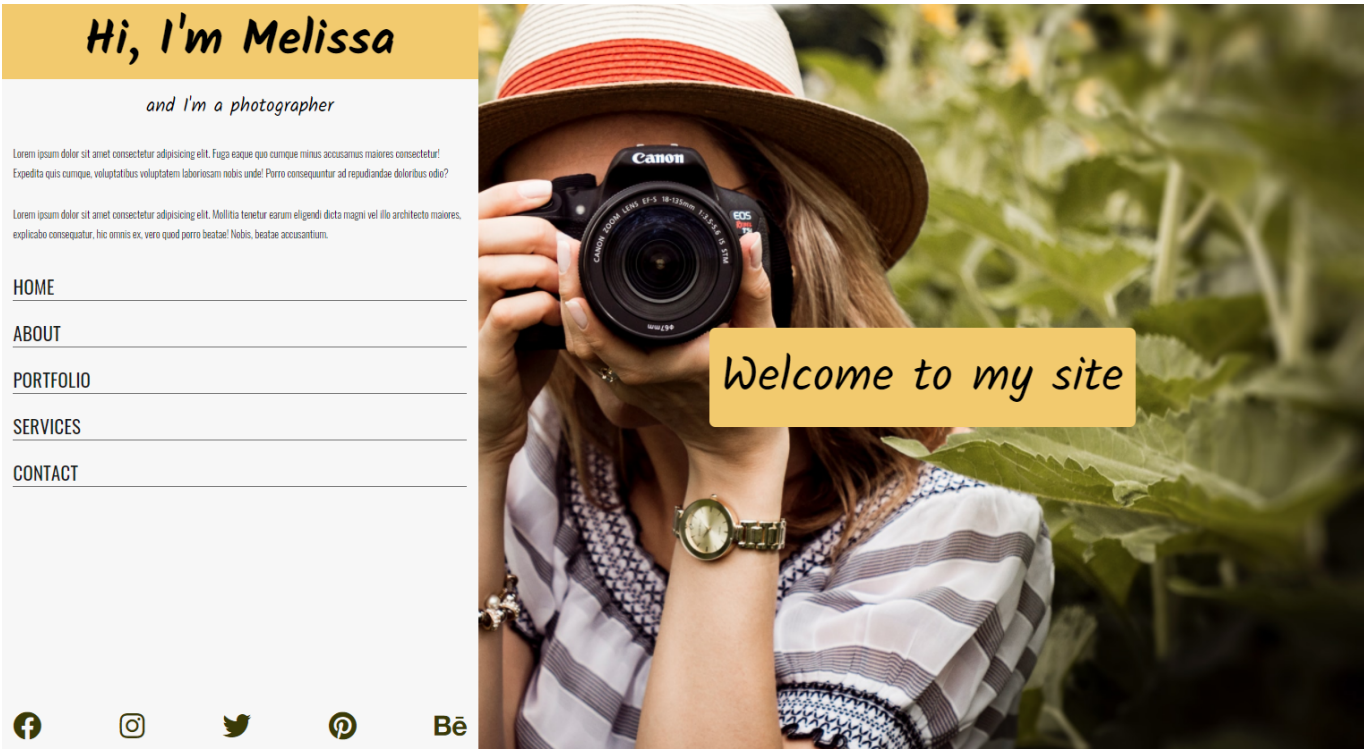
To push the social media icons to the bottom of the left panel, you can set the `nav` to **grow** to take up remaining space. After these steps, here's how your page should look:



- **Action**: Add CSS to the parent of the social icons to create a row-oriented flexbox whose items (the icons) are spread across the row with equal space between them.
- **Action**: Add CSS to the `nav` item to allow it to grow to take remaining vertical space.

Step 4: Center welcome banner

Finally, you need to center the welcome banner in the middle of the right panel. You can do this by making its parent (`section#right-panel`) into a flexbox and then aligning the items (the banner) both horizontally and vertically. It takes four style declarations to achieve the desired effect:



- **Action**: Use a flexbox in the right panel to vertically and horizontally center the banner within that panel.

After you complete this step, the tests in **Part Two: Build a portfolio page with flexbox** pass.