Flexbox, Grid, & Bootstrap

CSS Layout & Bootstrap

Learn by doing step by step exercises.

Includes downloadable class files that work on Mac & PC.

EDITION 2

noble desktop

Published by:
Noble Desktop
185 Madison Ave, 3rd Floor
New York, NY 10016
nobledesktop.com

Publish Date: 7-19-2021

The publisher makes no representations or warranties with respect to the accuracy or completeness of the contents of this work, and specifically disclaims any warranties. Noble Desktop shall not have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer software and hardware products described in it. Further, readers should be aware that software updates can make some of the instructions obsolete, and that websites listed in this work may have changed or disappeared since publication.

Table of Contents

SETUP & INTRODUCTION

INFO & EXERCISES

SECTION 1

SECTION 2

Table of Contents

SECTION 3

SECTION 4

Table of Contents

BONUS MATERIAL

REFERENCE MATERIAL

Thank You for Purchasing a Noble Desktop Course Workbook!

These instructions tell you how to install the class files you'll need to go through the exercises in this workbook.

Downloading & Installing Class Files

1. Navigate to the **Desktop**.
2. Create a **new folder** called **Class Files** (this is where you'll put the files after they have been downloaded).
3. Go to **nobledesktop.com/download**
4. Enter the code **fg-2107-19**
5. If you haven't already, click **Start Download**.
6. After the **.zip** file has finished downloading, be sure to unzip the file if it hasn't been done for you. You should end up with a **Flexbox Grid Class** folder.
7. Drag the downloaded folder into the **Class Files** folder you just made. These are the files you will use while going through the workbook.
8. If you still have the downloaded .zip file, you can delete that. That's it! Enjoy.

If You've Done Other Noble Desktop Coding Books

If you've setup Visual Studio Code in other Noble Desktop workbooks, there's nothing new for you to do so you're all set up.

Choosing a Code Editor to Work In

You probably already have a preferred code editor, such as **Visual Studio Code**, **Sublime Text**, etc. You can use whatever code editor you want, but we highly recommend Visual Studio Code.

Installing Visual Studio Code

Visual Studio Code is a great code free editor for Mac and Windows.

1. Visit **code.visualstudio.com** and download **Visual Studio Code**.
2. To install the app:
   - Mac users: Drag the downloaded app into your **Applications** folder.
   - Windows users: Run the downloaded installer.
     During installation check on the **Add "Open with Code" ...** options.

Setting Up Visual Studio Code

There are a few things we can do to make Visual Studio Code more efficient.

Setting Preferences

1. Launch **Visual Studio Code**.
2. In Visual Studio Code do the following:
   - Mac users: Go into the **Code** menu and choose **Preferences > Settings**.
   - Windows users: Go into the **File** menu and choose **Preferences > Settings**.
3. In the search field type: **wrap**
4. Find **Editor: Word Wrap** and change its setting from **off** to **on**

   NOTE: This ensures you'll be able to see all the code without having to scroll to the right (because it will wrap long lines to fit to your screen size).

Setting Up Open in Browser

Visual Studio Code does not include a quick way to preview HTML files in a web browser (which you'll do a lot). We can add that feature by installing an extension:

1. On the left side of the Visual Studio Code window, click on the **Extensions** icon ⊞.
2. In the search field type: **open in browser**
3. Under the **open in browser** click **Install**.

Defining a Keystroke for Wrapping Code

Visual Studio Code has a feature to wrap a selection with some code. This is very useful so we'll want to use it frequently. There's no keystroke for it by default, so let's define one:

1. In Visual Studio Code do the following:
   - Mac users: Go into the **Code** menu and choose **Preferences > Keyboard Shortcuts**.
   - Windows users: Go into the **File** menu and choose **Preferences > Keyboard Shortcuts**.
2. In the search field type: **wrap with**
3. Double–click on **Emmet: Wrap with Abbreviation** and then:
   - Mac users: Hold **Option** and press **W** then hit **Return**.
   - Windows users: Hold **Alt** and press **W** then hit **Enter**.
4. You're done, so close the **Keyboard Shortcuts** tab.

Creating Your Own Copy of Class Files

Throughout this workbook you will be editing class files that we have prepared for you. Instead of editing the originals, we'll have you make a copy just for yourself to edit.

1. If you have any windows open, minimize or hide them so you can see the Desktop.
2. Open the **Class Files** folder.
3. Follow the appropriate Mac or Windows instructions below:

Mac:

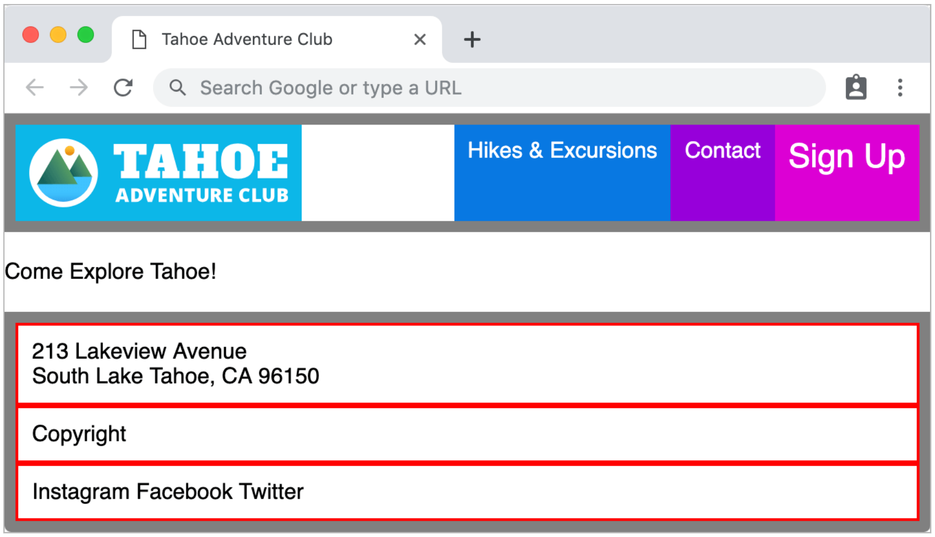
- Click once on the **Flexbox Grid Class** folder to select it.
- Press **Cmd–D** to duplicate it.
- Rename the duplicate folder **yourname-Flexbox Grid Class**.

Windows:

- Click once on the **Flexbox Grid Class** folder to select it.
- Press **Ctrl–C** to copy it.
- Press **Ctrl–V** to paste it.
- The new copy may be at the bottom of the list of folders. Rename it **yourname-Flexbox Grid Class**.

4. You now have your own set of class files to use throughout the class. Have fun!

Exercise Preview



Come Explore Tahoe!

213 Lakeview Avenue
South Lake Tahoe, CA 96150

Copyright

Instagram Facebook Twitter

Exercise Overview

In this exercise you'll get started with CSS **flexbox**, which is a way to lay out and align content.

Getting Started

1. Launch your code editor (**Visual Studio Code**, **Sublime Text**, etc.). If you are in a Noble Desktop class, launch **Visual Studio Code**.
2. We'll be working with the **Flexbox Intro** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class > Flexbox Intro**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Flexbox Intro** folder.
4. Notice the following:
   - There is a **ul** tag for a **navbar**.
   - Below that is one line of text.
   - There is a **footer** with 3 nested divs: address, copyright, and social.
   - We kept this page simple with only a little content so it will be easier to understand what's going on, and keep the CSS code that we'll be adding simpler, so you can keep your focus on flexbox.
   - In the head we're linked to a **main.css** file where we've applied a little styling to make the page visually easier to understand.

5. Preview **index.html** in Firefox. We'll be using Firefox because (at the time of this writing) Firefox has better DevTools for inspecting flexbox.
6. Notice the following:
    - The **navbar ul** and **footer** both have a gray border to help you see their size.
    - We gave each list item in the **navbar ul** a different color to help you see their size.
    - We gave the 3 **div** tags in the **footer** a red border to help you see their size.
7. Leave **index.html** open in Firefox as you work, so you can reload the page to see the code changes you'll make.

Display Flex

The CSS **display** property has values such as **block**, **inline**, **inline-block**, etc. One of the newer values is **flex** which we'll now explore.

1. Switch back to your code editor.
2. Open **main.css** from the **css** folder (in the **Flexbox Intro** folder).
3. In the **.navbar** rule add **`display: flex;`** as shown below in bold:

```
.navbar {
   display: flex;
   border: 8px solid gray;
   CODE OMITTED TO SAVE SPACE
}
```

4. Save the file, and reload the page in Firefox.
    - Notice the navbar list is now horizontal (it was a vertical stack before).
    - The width of the list items has collapsed to the width of the content inside them.
    - The gray border that we applied to the **navbar ul** tag extends the full width of the page, indicating that the list didn't collapse, just the elements inside.

Setting **display** to **flex** on an element makes it a **flex container**. The child elements automatically become **flex items**.

5. **Ctrl–click** (Mac) or **Right–click** (Windows) on anything inside the navbar and choose **Inspect Element**.
6. In the DevTools you should see a **`<ul class="navbar">`** element with nested **li** tags. Click on the **`<ul class="navbar">`**
7. In the DevTools **Flex Container** area you notice the list of **Flex Items** (the items within this flex container).

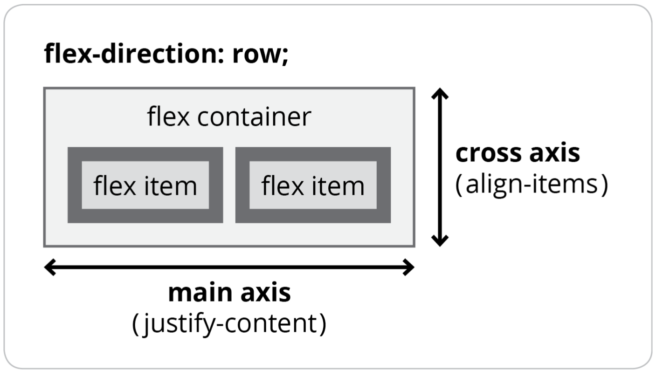
8. In the DevTools styles area you should see a **.navbar** style.
9. For the **display** property, click on **flex** so it becomes editable.
10. Change **flex** to **inline-flex**
11. Make the window wide enough to see that the navbar's gray border has collapsed to the width of the content, and the **Come Explore Tahoe!** now can fit to the right of the navbar. If you need to close the DevTools to make see this, go ahead and do so.

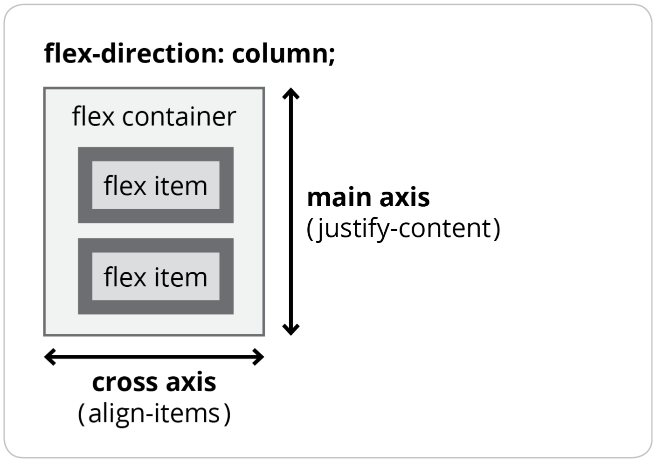
    Like block elements, **flex** elements fill the width of their parent element, whereas **inline-flex** elements do not (they are the width of their content). We don't want the navbar to be inline for this page, but it's good to know **inline-flex** exists.

12. Reload the page to get rid of this temporary change.

Alignment & Distribution on Main Axis & Cross Axis

Flexbox enables us to align or distribute flex items in multiple ways. To do this flexbox has a **main axis** and a **cross axis**. They are not called horizontal and vertical, because the direction of flexbox can be changed from the default **row**, to **column** (as you'll soon see). Here's an illustration of how they work:





1. Switch back to **main.css** in your code editor.
2. In the **.navbar** rule add the code shown below in bold:

```
.navbar {
   display: flex;
   justify-content: center;
   border: 8px solid gray;

   CODE OMITTED TO SAVE SPACE
}
```

3. Save the file, and reload the page in Firefox.
   - Notice the navbar items are centered horizontally.
4. **Ctrl–click** (Mac) or **Right–click** (Windows) on anything inside the navbar and choose **Inspect Element**.
5. In the DevTools click on the **`<ul class="navbar">`**
6. In the DevTools styles area, in **.navbar** style there's a **justify-content** property. Click on **center** so it becomes editable and change it to the following values (noticing how each changes the navbar in the page):
   - **flex-start** (the default value)
   - **flex-end**
   - **center**
   - **space-between**
   - **space-around**
   - **space-evenly**

   For a description/illustration of each, visit **tinyurl.com/csstricks-justify-content**
7. Switch back to **main.css** in your code editor.
8. In the **.navbar** rule add the code shown below in bold:

```
.navbar {
   display: flex;
   justify-content: center;
   align-items: center;
   border: 8px solid gray;

   CODE OMITTED TO SAVE SPACE
}
```

9. Save the file, and reload the page in Firefox.
   - Notice the navbar items are now centered vertically.

10. **Ctrl–click** (Mac) or **Right–click** (Windows) on anything inside the navbar and choose **Inspect Element**.

11. In the DevTools click on the **`<ul class="navbar">`**

12. In the DevTools styles area, in **.navbar** style there's a **align-items** property. Click on **center** so it becomes editable and change it to the following values (noticing how each changes the navbar in the page):

    - **stretch** (the default value)
    - **flex-start**
    - **flex-end**
    - **center**
    - **baseline**

    For a description/illustration of each, visit **tinyurl.com/csstricks-align-items**

Remembering Justify-Content vs. Align-Items

Here's a way to remember the difference:

- **justify-content** is a longer word, so main axis.
  (horizontal for row, vertical for column)
- **align-items** is a a shorter word, so cross axis.
  (vertical for row, horizontal for column)

Marcus Herrmann @_marcusherrmann remembers it in a more amusing way: "I memorize Michael Jackson = MJ = Main Axis -> justify-content"

Flex Direction (Row & Column)

By default, flex items are displayed as a row, but we make them a column.

1. Switch back to your code editor.

2. In the **.navbar** rule add the code shown below in bold:

```
.navbar {
   display: flex;
   flex-direction: column;
   justify-content: center;
   align-items: center;

   CODE OMITTED TO SAVE SPACE
}
```

3. Save the file, and reload the page in Firefox.
   - Notice the navbar is now displayed as a column.
   - The align-items center is centering horizontally now.
   - The height of the **navbar ul** is the height of the content inside, so justify-content center is not doing anything. We must set a larger height to see it work.
4. Switch back to your code editor.
5. In the **.navbar** rule add the code shown below in bold:

```
.navbar {
   display: flex;
   flex-direction: column;
   min-height: 100vh;
   justify-content: center;
   align-items: center;

   CODE OMITTED TO SAVE SPACE
}
```

   NOTE: To set the height, we can't use 100% like you might expect. Instead we can use viewport units, which are relative to the size of the viewport (the visible portion of the page). Similar to percentages, 1 viewport unit = 1/100th of the viewport.

6. Save the file, and reload the page in Firefox.
   - Notice the navbar fills the screen now, because we set the height to 100/100 of the visible height of the viewport.
   - Now the justify-content works, so the navbar content is vertically and horizontally centered in the window. It's awesome that we can create this layout with a few lines of code on a single element!
7. **Ctrl–click** (Mac) or **Right–click** (Windows) on anything inside the navbar and choose **Inspect Element**.
8. In the DevTools click on the **`<ul class="navbar">`**

9. In the DevTools styles area, in **.navbar** style experiment with different values for **justify-content** and **align-items**. As a reminder, here are the possible values for each:

Justify-Content	Align-Items
flex-start (default)	stretch (default)
flex-end	flex-start
center	flex-end
space-between	center
space-around	baseline
space-evenly	

NOTE: For **flex-direction** you can also use **row-reverse** and **column-reverse** if you want to reverse the order of content.

How Auto Margins Are Useful

1. Switch back to your code editor.
2. In the **.navbar** rule remove the properties you added (except for display) so you end up with:

```
.navbar {
   display: flex;
   border: 8px solid gray;

   CODE OMITTED TO SAVE SPACE
}
```

3. In the **.navbar .signup** rule add the bold code shown below:

```
.navbar .signup {
   margin-left: auto;
   background: #dc00d4;
   font-size: 1.5em;
}
```

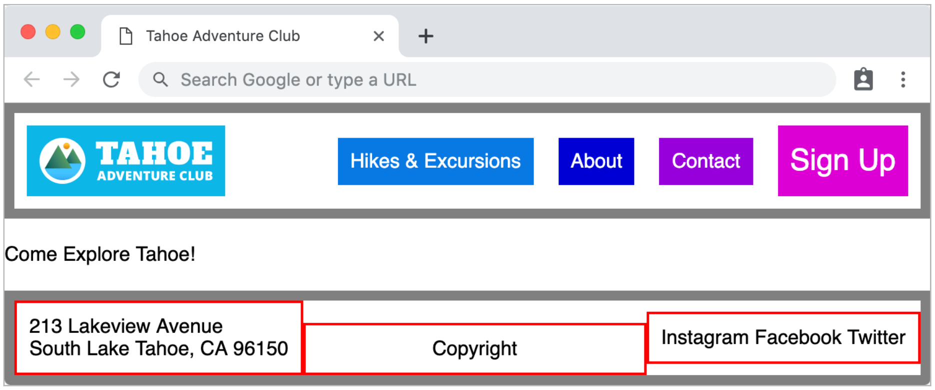
4. Save the file, and reload the page in Firefox.
   - Notice that **Sign Up** has moved all the way to the right of the page.
5. Switch back to **main.css** in your code editor.
6. In the **.navbar .logo** rule add the bold code shown below:

```
.navbar .logo {
   margin-right: auto;
   background: #0db8e8;
}
```

7. Save the file, and reload the page in Firefox.
   - Notice that the 2 middle nav items have an equal space around them, centered between the logo on the far left and **Sign Up** on the far right.
8. Switch back to your code editor.
9. In the **.navbar .signup** rule delete the margin code.
10. Save the file, and reload the page in Firefox.
    - Notice that all the nav items to the right of the logo have moved all the way to the right of the page.

Flexbox: Sizing & Alignment

Exercise Preview



Come Explore Tahoe!

213 Lakeview Avenue
South Lake Tahoe, CA 96150

Copyright

Instagram Facebook Twitter

Exercise Overview

In this exercise you'll learn how to size an align flex items (which are children of a flex container).

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Flexbox Size Alignment** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Flexbox Size Alignment** folder.
4. Preview **index.html** in a browser.
   - This page is the same as the one you finished in the previous exercise.
   - Notice the colored items in the nav do not fill the entire width of the navbar. Let's make them fill the entire area.
   - Notice the red outlined divs in the footer are stacked on top of each other (because they are block elements). We want to put them into a row and adjust their size and alignment.
5. Leave **index.html** open in the browser as you work, so you can reload the page to see the code changes you'll make.

Controlling Size with Flex-Grow

Let's see how to change the size of flex items (elements inside a flex container) with the **flex-grow** property. The default value for **flex-grow** is **0** which means do not grow. We can set it to a positive number to control how flex items grow.

1. Switch back to your code editor.
2. Open **main.css** from the **css** folder (in the **Flexbox Size Alignment** folder).
3. In the **.navbar li** rule, add the following bold code:

```
.navbar li {
   flex-grow: 1;
   color: white;
   padding: 10px;
}
```

4. Save the file, and reload the page in your browser.
   - Notice the colored items in the navbar now fill up the entire width.
   - The way flexbox works, is the browser takes any leftover space and divides it up between elements based on their flew-grow amounts. Because all the flex items share the same flex-grow (1), the leftover space is divided up and equally added to all the elements.
5. Switch back to your code editor.
6. Switch to **index.html**.
7. Add a new list item to the navbar, as shown below in bold:

```
<li class="hikes">Hikes & Excursions</li>
<li class="about">About</li>
<li class="contact">Contact</li>
```

8. Save the file, and reload the page in your browser.
   - We already had a style for this in our CSS, which is why the **About** has a color.
   - Notice that flexbox handles the new item well. There's less extra space now that we have another item. Each nav items gets a bit less space, but the spacing is evenly distributed.
   - What if we wanted the **About** element to be slightly bigger than the others? Let's see another feature of flex-grow.
9. Switch back to your code editor.
10. Switch to **main.css**.

11. In the **.navbar .about** rule add the following bold code:

```
.navbar .about {
   flex-grow: 2;
   background: #0000d5;
}
```

12. Save the file, and reload the page in your browser.

    - Notice how the **About** element is now slightly bigger than it was, and proportionally is slightly bigger than the other navbar elements.

How Flex-Grow Is Calculated

Flex-grow controls the distribution of free space. In order to know how wide to make the flex items, the browser sizes the elements and then looks at any leftover space. It divides the amount of free space by the total of all the flex-grow values and then adds that amount to elements based on their flex-grow value.

As an example, let's say there is 500px of leftover space and a total flex-grow of 5 (across all the flex items). 500px/5 means each flex-grow equals 100px. So an element with a flex-grow of 1 gets 100px bigger, an element with a flex-grow of 2 gets 200px, and so on.

Visit **css-tricks.com/flex-grow-is-weird** to see a great illustration of this.

13. Switch back to your code editor.

14. Remove the **flex-grow: 2;** from the **.navbar .about** rule.

15. Let's move onto the footer. In the **footer** rule, make it a flex container with the following bold code:

```
footer {
   display: flex;
   border: 8px solid gray;
   margin-top: 20px;
}
```

16. Save the file, and reload the page in your browser.

    - Notice the red outlined divs in the footer are now are in a row.
    - Notice the width of the red outlined divs has collapsed to the width of the content, and there's empty space to the right.

17. Switch back to your code editor.

18. In the **footer div** rule, set them to grow with the following bold code:

```
footer div {
   flex-grow: 1;
   border: 2px solid red;
   padding: 10px;
}
```

19. Save the file, and reload the page in your browser.

    - Notice the red outlined divs now fill the entire width of the footer.
    - The extra space has been evenly distributed between these 3 flex items, but what if we didn't want the social div to grow? Let's prevent that from changing size.

20. Switch back to your code editor.

21. Below the **footer div** rule, add the following new rule:

```
footer .social {
   flex-grow: 0;
}
```

22. Save the file, and reload the page in your browser.

    - Notice the width of the footer's social networks div has collapsed to the width of the text inside.
    - We really only want the copyright div to grow, so we should not have set a grow on all the divs in the footer. Let's go back and adjust.

23. Delete the **`flex-grow: 1;`** from the **footer div** rule.

24. Change the name of the **footer .social** rule to **footer .copyright** as shown below:

```
footer .copyright {
   flex-grow: 0;
}
```

25. In the **footer .copyright** rule, change **flex-grow** to **1** as shown below:

```
footer .copyright {
   flex-grow: 1;
}
```

26. While we're here, let's also center the text in the copyright div by adding the following bold code:

```
footer .copyright {
   flex-grow: 1;
   text-align: center;
}
```

27. Save the file, and reload the page in your browser.

    - Notice the footer's address and social divs are the width of their content, while the copyright div fills all the space between (with its text centered in the area).

Aligning All vs. Specific Flex Items

As we saw in the previous exercise, we can align items on the main axis or cross axis. We can align all items in a flex container, or individually align each flex item.

1. Switch back to your code editor.
2. Let's vertically center the footer divs. The current (and default) flex-direction is row, so the main axis is horizontal. That means we'll use the cross axis for vertical alignment. In the **footer** rule add the following bold code:

```
footer {
   display: flex;
   align-items: center;
   border: 8px solid gray;
   margin-top: 20px;
}
```

3. Save the file, and reload the page in your browser.

    - Notice the footer's red outlined divs are all vertically centered (and their height has collapsed to the content inside).
    - Let's align the copyright to the bottom of the footer.

4. Switch back to your code editor.
5. In the **footer** rule add the following bold code:

```
footer .copyright {
   flex-grow: 1;
   align-self: flex-end;
   text-align: center;
}
```

6. Save the file, and reload the page in your browser.

    - Notice the copyright should now be aligned to the bottom of the footer.

NOTE: It's important to keep in mind that some flexbox properties are applied to the flex container (the parent), while others are applied to flex items (the children). CSS Trick's **Complete Guide to Flexbox** tinyurl.com/flexref is the definitive flexbox reference with great illustrations and explanations of each property and whether they apply to the the flex container or flex item.

Nesting Flexbox

1. Switch back to your code editor.

2. We no longer want the navbar items to grow, so in the **.navbar li** rule, delete the **flex-grow: 1;**

3. Let's vertically center the items in the navbar. In the **.navbar** rule add the code shown below in bold:

```
.navbar {
   display: flex;
   align-items: center;
   border: 8px solid gray;
   CODE OMITTED TO SAVE SPACE
}
```

4. Save the file, and reload the page in your browser.

   - In the navbar, the logo should be on the left with the nav text on the right.

   - Let's make the **Sign Up** stretch the entire height of the navbar so it stands out.

5. Switch back to your code editor.

6. In the **.navbar .signup** rule add the following bold code:

```
.navbar .signup {
   align-self: stretch;
   background: #dc00d4;
   font-size: 1.5em;
}
```

7. Save the file, and reload the page in your browser.

   - Notice that **Sign Up** stretches to fill the entire height of the navbar, but the text is no longer centered.

   - How do we get **Sign Up** to stretch and have its text vertically centered?
     Flex containers (the parent element) contain flex items (the child element).
     Flex items can themselves also be turned into flex containers. This is called nesting.
     By setting **Sign Up** (currently a flex item) to also be flex container, we can control the alignment of elements inside it.

8. Switch back to your code editor.

9. The **.signup** div is currently a flex item, but it's not yet a flex container. Add the following bold code to make it a flex container and align its children along the cross axis:

```
.navbar .signup {
   align-self: stretch;
   display: flex;
   align-items: center;
   background: #dc00d4;
   font-size: 1.5em;
}
```

10. Save the file, and reload the page in your browser.

    - Notice that **Sign Up** text is now vertically centered within the color that stretches to fill the entire height of the navbar.

11. Switch back to your code editor.

12. Space apart all the nav items by adding the following bold code to **.navbar li**:

```
.navbar li {
   margin: 10px;
   color: white;
   padding: 10px;
}
```

13. Save the file, and reload the page in your browser.

    - Notice everything in the navbar has whitespace around it.
    - Resize the window narrower, to notice that the logo shrinks down because its container is shrinking. Let's stop that.
    - The **Sign Up** text breaks onto two lines when the window is too narrow, which we'd also like to prevent.

Controlling Size with Flex-Shrink

To control whether flex items get smaller we use **flex-shrink**. Its default value is **1**, which means items get smaller. This works like flex-grow, but removing space instead of adding it. Flexbox looks at the space that needs to be removed, and distributes that based on the flex-shrink value. By default all items have flex-shrink of 1, so equal amounts of space are removed from each item.

1. Switch back to your code editor.

2. To prevent the **Sign Up** button from shrinking, add the following bold code to the **.navbar .signup** rule:

```
.navbar .signup {
   align-self: stretch;
   flex-shrink: 0;
   display: flex;
   align-items: center;
   background: #dc00d4;
   font-size: 1.5em;
}
```

3. Do the same thing in the **.navbar .logo** rule:

```
.navbar .logo {
   flex-shrink: 0;
   margin-right: auto;
   background: #0db8e8;
}
```

4. Save the file, and reload the page in your browser.

   - Resize the window from wide to narrow, noticing that the logo and **Sign Up** no longer shrink.

Sizing with Flex-Basis

To set the width or height of a flex item (depending on the flex-direction), we use **flex-basis**. Let's see how it works by making the logo div less wide.

1. Switch back to your code editor.

2.
```
.navbar .logo {
   flex-shrink: 0;
   flex-basis: 160px;
   margin-right: auto;
   background: #0db8e8;
}
```

NOTE: In the provided CSS we set **`max-width: 100%;`** on all images, so by shrinking the **.logo** div, the image inside will automatically shrink. If the content could not shrink, the column would not reduce in size. It would be as wide as the content unless you set overflow to hidden (which would cut off the excess content).

3. Save the file, and reload the page in your browser.

   - Notice the width of the logo div (and therefore the image inside) has reduced.

   NOTE: Flex-basis sets the initial size an element should be, **before** it grows or shrinks. Setting a flex-basis doesn't guarantee the element will be that size, unless you disable flex-grow and flex-shrink.

Flex Basis vs Width

"In case both **flex-basis** (other than **auto**) and **width** (or **height** in case of **flex-direction: column**) are set for an element, **flex-basis** has priority."
— Mozilla's MDN **tinyurl.com/fba-vs-w**

Why not just use width? Keep in mind that Flexbox lets you change directions, and flex-basis works on the main axis. That means it works like **width** for **flex-direction: row** or **height** for **flex-direction: column**. Also, when using the **flex** shorthand property you can combine all 3 properties (flex-grow, flex-shrink and flex-basis) into one line of code.

Flex Shorthand

Instead of writing out **flex-grow**, **flex-shrink**, and **flex-basis**, we can combine all 3 properties into one shorthand property: **flex**.

- The syntax is **`flex: flex-grow flex-shrink flex-basis;`**
- The default value is **`flex: 0 1 auto;`**

1. Switch back to your code editor.

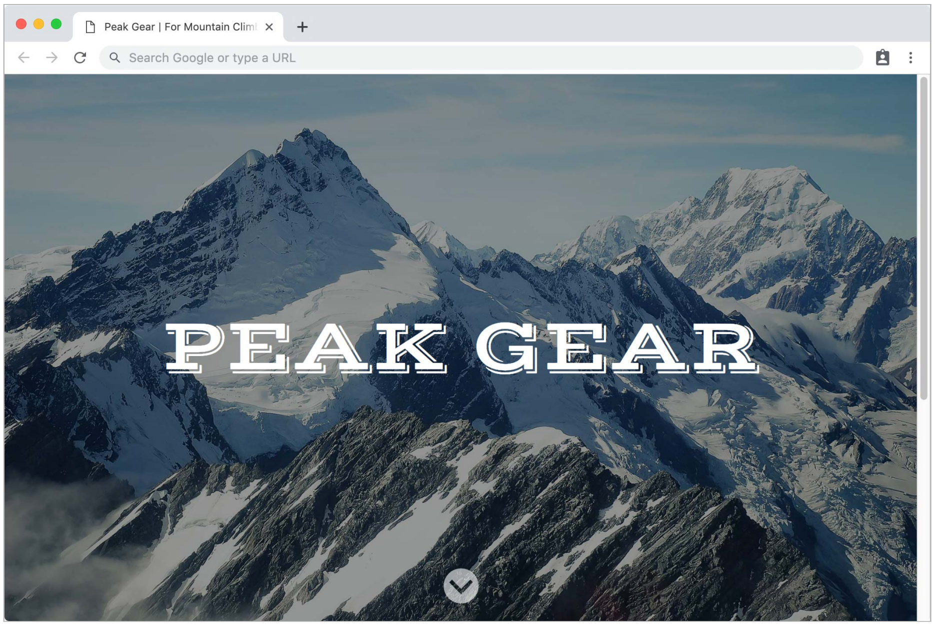
2. In the **.navbar .logo** rule merge the **flex-shrink** and **flex-basis** properties into one **flex** property as shown below:

```
.navbar .logo {
   flex: 0 0 160px;
   margin-right: auto;
   background: #0db8e8;
}
```

3. Save the file, and reload the page in your browser.

   - The page should look the same, but you now had more concise code.

Exercise Preview



Exercise Overview

In this exercise you'll use flexbox to layout a page header that fills the entire screen with an image, and a heading that is centered in the screen.

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Flexbox Vertical Centering** folder located in **Desktop** > **Class Files** > **yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Flexbox Vertical Centering** folder.
4. Preview **index.html** in a browser.
   - Currently the header has a photo background that is sized to the content, but we want it to fill the screen.
5. Keep the page open in your browser, and switch back to **index.html** in your code editor.
6. Find the **header** tag.
7. Inside that, notice the heading and anchor tags. These are the elements we will be targeting.

8. In your code editor, open **main.css** from the **css** folder (in the **Flexbox Vertical Centering** folder).

Creating a Full Screen Background

We want the header to take up the full screen. Background images fill their container, so we need to make the header element taller.

1. Switch back to **main.css** in your code editor.
2. Add the following bold code:

```
header {
   color: #fff;
   background: url(../img/mountains@2x.jpg) center;
   background-size: cover;
   min-height: 100vh;
}
```

3. Save the file.
4. Switch back to the browser and reload. The header fills the entire height of the screen now.
5. Resize the browser window to see how the header area resizes with the window. The background image scales to cover the area.

Darkening the Background Image

The text can be hard to read on the brighter parts of the background image. We can add a transparent dark overlay on top of the background image to help this.

1. Return to your code editor.
2. Add the following bold code and don't miss the comma!

```
header {
   color: #fff;
   background:
      linear-gradient(),
      url(../img/mountains@2x.jpg) center;
   background-size: cover;
   min-height: 100vh;
}
```

3. Add the following bold code:

```
header {
   color: #fff;
   background:
      linear-gradient( rgba(), rgba() ),
      url(../img/mountains@2x.jpg) center;
   background-size: cover;
   min-height: 100vh;
}
```

4. Add the following bold code:

```
header {
   color: #fff;
   background:
      linear-gradient( rgba(0,0,0, .4), rgba(0,0,0, .4) ),
      url(../img/mountains@2x.jpg) center;
   background-size: cover;
   min-height: 100vh;
}
```

5. Save the file.

6. Switch back to the browser and reload to see the darkened background image!

Vertically Centering the Header Content

Now that the header fills the screen, we want to vertically center the text inside it.

1. Add the following bold code:

```
header {

   CODE OMITTED TO SAVE SPACE

   min-height: 100vh;
   display: flex;
}
```

2. Save the file, and reload the page in your browser.

   - Notice the scroll down arrow has moved to the right of **Peak Gear** because the default flex direction is row. Let's stack these vertically by changing that direction.

3. Return to your code editor.

4. Add the following bold code:

```
header {
   CODE OMITTED TO SAVE SPACE
   display: flex;
   flex-direction: column;
}
```

5. Save the file, and reload the page in your browser.

   - Notice the scroll down arrow is below **Peak Gear**, which is what we want. Now we can center the content in both directions: vertically (along the main axis) and horizontally (along the cross axis).

6. Return to your code editor.

7. Add the following bold code:

```
header {
   CODE OMITTED TO SAVE SPACE
   display: flex;
   flex-direction: column;
   justify-content: center;
   align-items: center;
}
```

8. Save the file, and reload the page in your browser.

   - On large screens the content is nicely centered and looking good!
   - Resize the window narrower, to see that the heading text is left aligned (on mobile screens), because the default text alignment is left. Let's center the text.

9. Switch back to your code editor.

10. In the **h1** rule, add the following bold code:

```
h1 {
   CODE OMITTED TO SAVE SPACE
   margin: 0;
   text-align: center;
}
```

11. Save the file, and reload the page in your browser.

    - On narrow screens (as well as wide) the text should now be centered.

Moving the Scroll Down Arrow to the Bottom

1. Switch back to your code editor.
2. In the **h1** rule, change the **margin** from **0** to **auto** as shown below:

   ```
   h1 {
      CODE OMITTED TO SAVE SPACE
      margin: auto;
      text-align: center;
   }
   ```

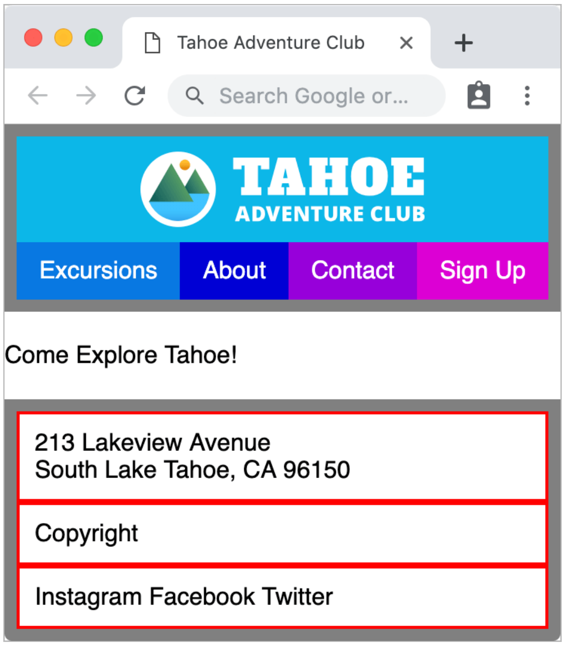
3. Save the file, and reload the page in your browser.
   - Notice the scroll down arrow has moved down to the bottom, which is where we want it!
   - The scroll down arrow needs some space below so it doesn't touch the edge.
   - The heading is now vertically centered between the top of the page and the scroll down arrow. That means the heading is not fully centered in the screen (it's a little too high). We think it looks good, but let's see how to vertically center it perfectly in case you wanted to.
4. Return to your code editor.
5. In the **header** rule add the following bold code to adjust the spacing:

   ```
   header {
      CODE OMITTED TO SAVE SPACE
      align-items: center;
      padding: 60px 0 20px;
   }
   ```

   NOTE: Why these amounts? We're adding 20px of padding to the bottom (below the arrow). The height of the arrow (40px) plus the 20px equals 60px. Therefore we're adding 60px of top padding to match the space occupied by the arrow to balance things out.

6. Save the file, and reload the page in your browser.
   - Resize the window from mobile through desktop size to check out your full screen, responsive background!

Exercise Preview



Come Explore Tahoe!

213 Lakeview Avenue
South Lake Tahoe, CA 96150

Copyright

Instagram Facebook Twitter

Exercise Overview

In this exercise you'll learn how to make flex items can wrap onto multiple lines.

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Flexbox Wrapping** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Flexbox Wrapping** folder.
4. Preview **index.html** in Firefox (we'll be using its DevTools).
   - This is page very similar to previous exercises, but this time we're going to style the navigation a bit differently.
5. Leave **index.html** open in Firefox as you work, so you can reload the page to see the code changes you'll make.

Setting Display Flex & Wrapping

1. Switch back to your code editor.

2. Open **main.css** from the **css** folder (in the **Flexbox Wrapping** folder).

3. In the **.navbar** rule add the following bold code:

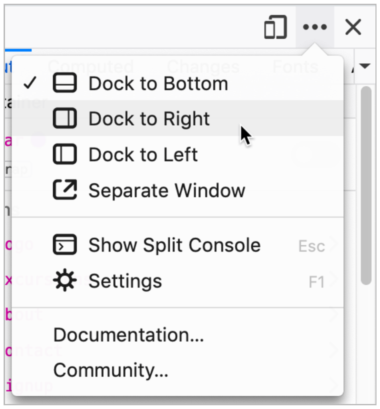
```
.navbar {
   display: flex;
   border: 8px solid gray;

   CODE OMITTED TO SAVE SPACE
}
```

4. Save the file, and reload the page in Firefox.

   - Notice the items in the nav now display as a row (they were stacking).

5. **Ctrl–click** (Mac) or **Right–click** (Windows) anywhere on the apge and choose **Inspect Element**.

6. If the DevTools are not on the right side of the window, at the top right of the DevTools panel, click the ••• button and choose **Dock to Right**.



   Now you'll be able to resize the webpage area smaller than you could if the DevTools were not open.

7. Resize the webpage area narrower until you can see the nav items remain on one line, even when it forces the content wider than the width of the window (which requires horizontal scrolling).

8. Keep the DevTools open.

9. Return to your code editor.

10. In the **.navbar** rule add the following bold code:

```
.navbar {
   display: flex;
   flex-wrap: wrap;
   border: 8px solid gray;

   CODE OMITTED TO SAVE SPACE
}
```

11. Save the file, and reload the page in Firefox.
    - Resize the webpage area and notice the items now wrap when they do not fit on a line.

Sizing the Flex Items

1. Return to your code editor.
2. Above the **min-width: 571px** media query, add the following new media for smaller screens:

```
@media (max-width: 570px) {
   .navbar .logo {
      flex-basis: 100%;
   }
}
```

3. Save the file, and reload the page in Firefox.
   - The logo should fill the width, with the other items wrapping to the line(s) below.
   - Because the remaining items don't scale up to fill the available space, there's empty white space on the right. Let's make them grow.
4. Return to your code editor.
5. In the **max-width: 570px** media query, below the **.navbar .logo** rule, add the following new rule:

```
@media (max-width: 570px) {
   .navbar .logo {
      flex-basis: 100%;
   }
   .navbar li {
      flex-grow: 1;
   }
}
```

6. Save the file, and reload the page in Firefox.
   - Now that the items are bigger, let's center the text inside them.
7. Return to your code editor.

8. In **.navbar li** rule add the following bold code:

```
@media (max-width: 570px) {
   .navbar .logo {
      flex-basis: 100%;
   }
   .navbar li {
      flex-grow: 1;
      text-align: center;
   }
}
```

9. Save the file, and reload the page in Firefox.

   - That looks better with the text centered for the nav items.
   - Resize the webpage area so the 4 nav "links" become a grid of 2 by 2.
   - It would look nicer if they were 50% wide so the widths would match across both rows.

10. Return to your code editor.

11. In the **max-width: 570px** media query's **.navbar li** rule, add the following bold code:

```
.navbar li {
   flex-grow: 1;
   flex-basis: 50%;
   text-align: center;
}
```

12. Save the file, and reload the page in Firefox.

   - That's looking better on small screens!
   - Close the DevTools.
   - Make the browser window wide enough so you see the navbar on one line again.
   - Notice that all the extra space is on the right of the nav. We'd like the nav links to move over to the right (while the logo remains on the left).

13. Return to your code editor.

14. Before we adjust the spacing, let's make our code more efficient by changing it to the flex shorthand.

    In the **max-width: 570px** media query's **.navbar li** rule, replace the two flex properties into one as shown below in bold:

    ```
    .navbar li {
       flex: 1 0 50%;
       text-align: center;
    }
    ```

    NOTE: Remember that the syntax is **flex: flex-grow flex-shrink flex-basis;**

Adjusting the Layout on Larger Screens

1. In the **min-width: 571px** media query, above the **footer** rule add the following new rule:

   ```
   .navbar .logo {
      margin-right: auto;
   }
   ```

2. Save the file, and reload the page in Firefox.

   - The nav should now look better on larger screens, with the logo on the left and the nav links on the right.

Optional Bonus: Refining the Layout at Another Screen Size

The 2 by 2 grid looks good on small screens, but we you get a little bigger, all 4 nav "links" could all fit on a single line below the logo. Let's add a new media query for these size screens.

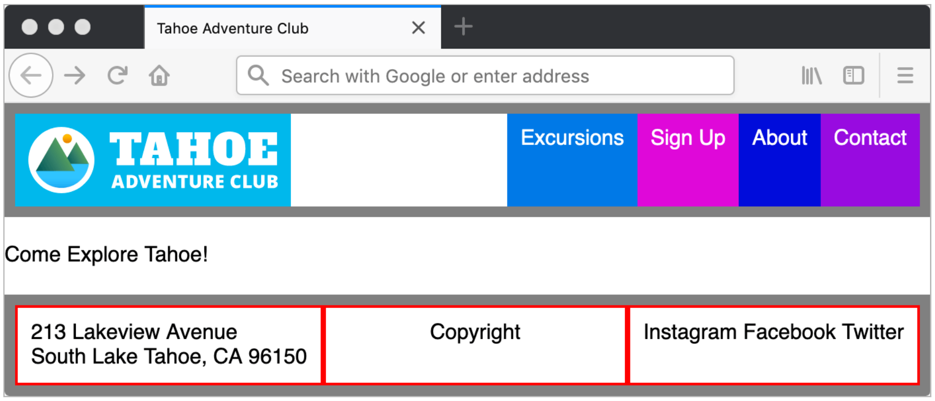
1. Above the **min-width: 571px** media query, add the following media and new rule:

   ```
   @media (min-width: 415px) and (max-width: 570px) {
      .navbar li {
         flex-basis: 25%;
      }
   }
   ```

2. Save the file, and reload the page in Firefox.

   - Resize the window from narrow to wide, so you can see the nav links should be a 2 by 2 grid on small screens, then changing to a single row (below the logo), and finally changing to logo on the left with the nav links on the right.

Exercise Preview



Tahoe Adventure Club

Search with Google or enter address

TAHOE ADVENTURE CLUB

Excursions Sign Up About Contact

Come Explore Tahoe!

213 Lakeview Avenue
South Lake Tahoe, CA 96150

Copyright

Instagram Facebook Twitter

Exercise Overview

In this exercise you'll learn now to change the order of flex items, which can change across various screens sizes!

Ordering Content in HTML

When order content in HTML, you must decide if you'll order things according to the mobile or desktop layout.

Mobile first develop is popular for multiple reasons, such as:

- It's typically more efficient to code mobile first. That's because on mobile devices we often stack content on top of each other, which is the browser's default appearance. Then on larger screens we can make columns as needed.
- Many websites have more mobile users than desktop users, so it makes sense to develop first for the larger audience.

We think the order of the items in HTML should be in the logical, ideal order (which typically is what you want for mobile).

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Flexbox Reorder Content** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Flexbox Reorder Content** folder.

4. Preview **index.html** in a browser.
   - Resize the window to be narrow, so you see the navbar and footer items stack (instead of being in a single row).
   - The order of items in the navbar and footer are different than in previous exercises. We changed the order to be the ideal order, which is correct on mobile.
   - Make the window wider, so you see the navbar and footer items as a single row.
   - We're going to change the order of some of the elements on wider screens.
5. Leave **index.html** open in the browser as you work, so you can reload the page to see the code changes you'll make.

Reordering Content in the Footer (Positive vs. Negative Order Values)

Before we start, keep in mind that the default **order** of all flex items is **0**.

1. Switch back to your code editor.
2. Open **main.css** from the **css** folder (in the **Flexbox Reorder Content** folder).
3. In the **min-width: 571px** media query, below the **footer** rule add the following new rule:

```
footer .social {
   order: -1;
}
```

   NOTE: The **order** property does not work on all elements. The reason it works here, is because the **.social** div is a flex item (because we set the footer to display: flex).

4. Save the file, and reload the page in your browser.
   - The social div has moved to the start, because -1 is less than the default 0.
5. Return to your code editor.
6. In the **min-width: 571px** media query's **footer .social** rule, change the order to 1:

```
footer .social {
   order: 1;
}
```

7. Save the file, and reload the page in your browser.
   - Now the social div has moved to the end, because 1 is greater than 0.

Reordering Content in the Navbar

1. Return to your code editor.

2. In the **min-width: 571px** media query, below the **.navbar .logo** rule add the following new rule:

```
.navbar .signup {
   order: 1;
}
```

3. Save the file, and reload the page in your browser.

   - **Sign Up** should be all the way on the right, because 1 is greater than the default 0.
   - While this position is where we want it, what if we wanted it to be between **Excursions** and **About**? The logo would be the first item, **Excursions** would be second, but setting **Sign Up** to 3 won't make it third in the current group, just like setting order to 1 didn't make it first. That's because the default order of flex items is 0. Let's change that for the nav.

4. Return to your code editor.

5. In the **min-width: 571px** media query, above the **.navbar .logo** rule add the following new rule:

```
.navbar li {
   order: 5;
}
```

   NOTE: Giving it a higher number ensures we have enough low numbers to order the first items the way we want.

6. In the **min-width: 571px** media query's **.navbar .logo** rule, add the following bold code:

```
.navbar .logo {
   order: 1;
   margin-right: auto;
}
```

7. In the **min-width: 571px** media query, below the **.navbar .logo** rule add the following new rule:

```
.navbar .excursions {
   order: 2;
}
```

8. In the **min-width: 571px** media query, in the **.navbar .signup** rule change the order to 3:

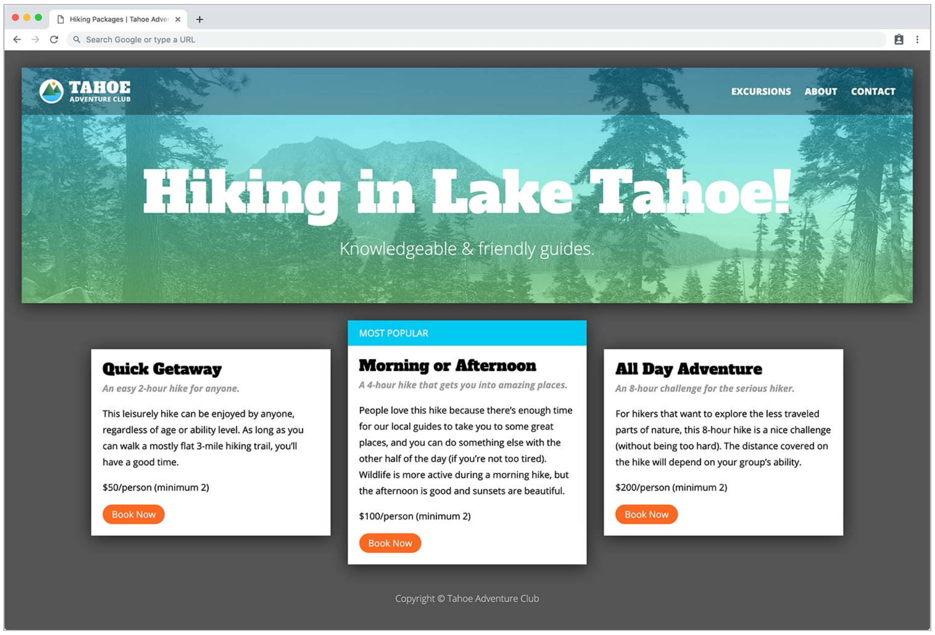
```
.navbar .signup {
   order: 3;
}
```

9. Save the file, and reload the page in your browser.

   - The order on wider screens should now be **Excursions, Sign Up, About, Contact**.
   - Resize the browser window so you see that all the order changes only apply to the desktop size (because we put them in a media query) and the mobile still retains the original order.

Flexbox: Creating a Responsive Pricing Grid

Exercise Preview


Hiking Packages | Tahoe Adve
Search Google or type a URL
TAHOE
ADVENTURE CLUB
EXCURSIONS
ABOUT
CONTACT
Hiking in Lake Tahoe!
Knowledgeable & friendly guides.
MOST POPULAR
Book Now
Copyright © Tahoe Adventure Club

Quick Getaway

An easy 2-hour hike for anyone.

This leisurely hike can be enjoyed by anyone, regardless of age or ability level. As long as you can walk a mostly flat 3-mile hiking trail, you'll have a good time.

$50/person (minimum 2)

Morning or Afternoon

A 4-hour hike that gets you into amazing places.

People love this hike because there's enough time for our local guides to take you to some great places, and you can do something else with the other half of the day (if you're not too tired). Wildlife is more active during a morning hike, but the afternoon is good and sunsets are beautiful.

$100/person (minimum 2)

All Day Adventure

An 8-hour challenge for the serious hiker.

For hikers that want to explore the less traveled parts of nature, this 8-hour hike is a nice challenge (without being too hard). The distance covered on the hike will depend on your group's ability.

$200/person (minimum 2)

Exercise Overview

In this exercise you'll use your new flexbox knowledge to lay out a pricing grid. You'll also learn how nesting flexbox is useful.

Getting Started

1. In your code editor, close any files you may have open.

2. For this exercise we'll be working with the **Flexbox Pricing Grid** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).

3. Open **hikes.html** from the **Flexbox Pricing Grid** folder.

4. Preview **hikes.html** in Firefox (we'll be using its DevTools).

   - The layout looks good on smaller screens, but on desktops we want the 3 options (**Morning or Afternoon**, **Quick Getaway**, and **All Day Adventure**) to be columns.

5. Leave **hikes.html** open in Firefox as you work, so you can reload the page to see the code changes you'll make.

Using Flexbox to Create Column

1. Switch back to your code editor.

2. In **hikes.html** notice that inside the **main** tag we have 3 **section** tags which we want to turn into columns.

3. Open **main.css** from the **css** folder (in the **Flexbox Pricing Grid** folder).

4. In **min-width: 1000px** media query:

```
main {
   display: flex;
   max-width: 1310px;
   margin: 30px auto;
}
```

5. Save the file, and reload the page in Firefox.

   - Resize the browser window wide enough so you see the 3 columns.
   - Notice the width of 3 columns are not equal. Let's fix that.
   - Also notice the columns on the far left and far right do not line up with the header above. We'll need to adjust the margins.

6. Return to your code editor.

7. In **min-width: 1000px** media query, below the **main** rule add this new rule:

```
section {
   flex-basis: 33.33%;
   margin: 0;
}
```

8. Save the file, and reload the page in Firefox.

   - The width of the 3 columns are now equal.
   - We'll need to add margins between the columns, but notice the columns on the far left and far right now line up with the header above.

9. Return to your code editor.

10. In **min-width: 1000px** media query, below the **section** rule add this new rule:

```
section:not(:last-child) {
   margin-right: 30px;
}
```

NOTE: This selector targets all sections that are not the last child. This will only add right margin to the first and second columns, not the third.

11. Save the file, and reload the page in Firefox.

    - The column spacing now looks good.
    - If you're used to how width works, you may be surprised that the 3 columns still fit on a line now that we've added margin. If each column truly was 33.33%, adding margins would have pushed the third column onto the next line. We don't have flex wrap on, so flexbox fits the elements into a single line. Therefore flex-basis is considered an ideal size, but not guaranteed! It is subject to growing or shrinking. To learn more about flex-basis and width, read **tinyurl.com/fb-width**
    - We want to move the **Most Popular** column to be the middle column.

Reordering the Columns

1. Return to your code editor.
2. In **min-width: 1000px** media query, below the **section:not(:last-child)** rule add this new rule:

```
section.secondary {
   order: -1;
}
```

3. Save the file, and reload the page in Firefox.

    - The **Most Popular** column should now be the middle column.
    - Let's move the price and **Book Now** buttons to the bottom of the columns, so they are aligned with each other. This is much easier to do now with flexbox.

4. **Ctrl–click** (Mac) or **Right–click** (Windows) on the **Quick Getaway** heading (in the left column) and choose **Inspect Element**.
5. In the DevTools **`<h2>Quick Getaway</h2>`** should be selected. Above that click on the containing **`<div class="text-wrapper">`**
6. As your cursor hovers over **`<div class="text-wrapper">`** (you may need to move the cursor a bit) in the page you should see the section highlighted.

    - Notice the height of this **text-wrapper** does not extend down to the bottom of the column.
    - If we're going to move the price and **Book Now** button down, this div needs to stretch to fill the container div. Flexbox items can grow, so we can make the **text-wrapper**'s parent tag (the **section** tag) a flex container so this will become a flex child that can grow.
    - Keep the DevTools open so we can return to them in a moment.

Vertically Aligning Buttons to the Bottom (Nesting Flexbox)

1. Return to your code editor.
2. In **min-width: 1000px** media query's **section** rule, add the following bold code:

```
section {
   flex-basis: 33.3333%;
   margin: 0;
   display: flex;
}
```

3. Save the file, and reload the page in Firefox.
   - The **Most Popular** label has changed to vertical. That's because this section contains the **Most Popular** paragraph and a **.text-wrapper** div (which contains everything else). These 2 items have become flex items in the default direction of row. We'll need to change the direction to column.
   - The outer columns only have one flex item so we don't see the issue with them.
4. Return to your code editor.
5. In **min-width: 1000px** media query's **section** rule, add the following bold code:

```
section {
   flex-basis: 33.3333%;
   margin: 0;
   display: flex;
   flex-direction: column;
}
```

6. Save the file, and reload the page in Firefox.
   - **Most Popular** should be back on top where it belongs.
   - In the DevTools hover over `<div class="text-wrapper">` again and notice it's still not the height of the containing div. We'll need to tell this flex item to grow.
7. Return to your code editor.
8. In **min-width: 1000px** media query, below the **section.secondary** rule add this new rule:

```
.text-wrapper {
   flex-grow: 1;
}
```

9. Save the file, and reload the page in Firefox.
   - In the DevTools hover over **`<div class="text-wrapper">`** again to see the height now fills the entire column!
   - We're getting closer, but for us to move the price down, those items also need to be flex items. That means the **.text-wrapper** also needs to become a flex container.
   - If you're having trouble remembering the structure, here's an overview:
     main (flex container)
     --> **section** (flex item, and flex container)
        --> **.text-wrapper** (flex item, needs to be a flex container)
           --> **items in .text-wrapper** (need to be flex items)
10. Return to your code editor.
11. In **min-width: 1000px** media query's **.text-wrapper** rule, add the following bold code:

```
.text-wrapper {
   flex-grow: 1;
   display: flex;
}
```

12. Save the file, and reload the page in Firefox.
    - Wow, the default direction of row has messed up this layout! Not to worry, we can change it to column.
13. Return to your code editor.
14. In **min-width: 1000px** media query's **.text-wrapper** rule, add the following bold code:

```
.text-wrapper {
   flex-grow: 1;
   display: flex;
   flex-direction: column;
}
```

15. Save the file, and reload the page in Firefox.
    - That's better!
    - You may notice that the paragraphs have more space between them than they previously did. Normally vertical margins between elements like headings and paragraphs collapse into each other (so you only get the larger of the two amounts). Margins work differently on flex items, because they stack (you get both the bottom margin from one element and the top margin from the next). We're not going to worry about it for this layout, but that's important to know.
    - Now we can finally move the price and **Book Now** buttons down!

16. Return to your code editor.

17. In **min-width: 1000px** media query, below the **.text-wrapper** rule add this new rule:

```
.text-wrapper .price {
   margin-top: auto;
}
```

18. Save the file, and reload the page in Firefox.

    - Nice! The price and **Book Now** buttons look good aligned with each other.
    - Let's move the top of the left and right columns down to draw more attention to the **Most Popular** line.

19. Return to your code editor.

20. In **min-width: 1000px** media query, below the **section.secondary** rule add this new rule:

```
section:not(:first-child) {
   margin-top: 44px;
}
```

    NOTE: You may try to use nth-child(odd) because we want to style the first and third columns, but that wouldn't work. Remember that the primary section is the first item in the code, even though on desktops we're rearranged the order. CSS refers to the code order, not the visual order.

21. Save the file, and reload the page in Firefox.

    - That looks good, but let's try another layout.

Vertically Centering the Sections

1. Return to your code editor.

2. In **min-width: 1000px** media query, delete the following rules because they're no longer needed (they should all be next to each other):

    - **section:not(:first-child)**
    - **.text-wrapper**
    - **.text-wrapper .price**

3. In the **section** rule (inside **min-width: 1000px** media query) delete the **display** and **flex-direction** properties so you end up with the following:

```
section {
   flex-basis: 33.3333%;
   margin: 0;
}
```

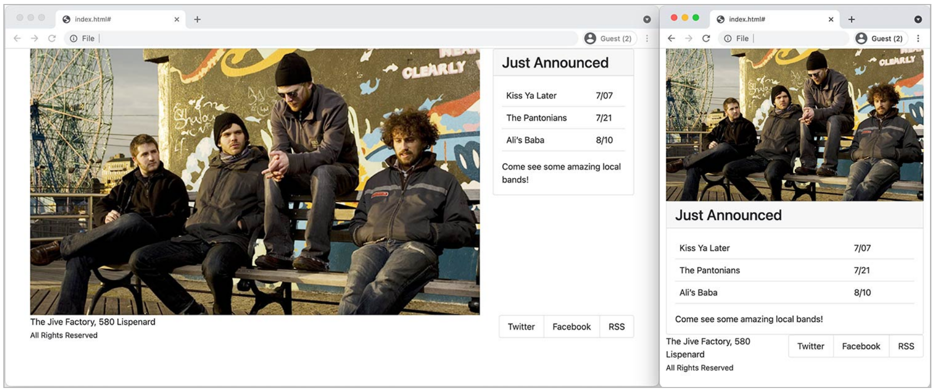
4. In **min-width: 1000px** media query's **main** rule, add the following bold code:

```
main {
   display: flex;
   align-items: center;
   max-width: 1310px;
   margin: 30px auto;
}
```

5. Save the file, and reload the page in Firefox.

   - The left and right columns should be vertically centered compared to the center column.
   - Resize the window to see the layout looks nice on small screens and large.

Exercise Preview



Just Announced

Kiss Ya Later	7/07
The Pantonians	7/21
Ali's Baba	8/10

Come see some amazing local bands!

Twitter Facebook RSS

The Jive Factory, 580 Lispenard
All Rights Reserved

Exercise Overview

In this exercise, you'll start learning about **Bootstrap**… a popular front-end component library that can make web development faster and easier. Its core features are a grid system for creating rows/columns, and components (such as navbars, buttons, carousels, and more). We'll be using Bootstrap version 5.0.2.

Getting Started

1. Switch to the **Desktop** (don't be in your code editor).
2. Navigate into **Class Files > yourname-Flexbox Grid Class > Bootstrap Starter**.
   - We went to **getbootstrap.com** and downloaded the files for you. The download included **css** and **js** folders.

     In those folders we deleted all but the 2 required files (one CSS and one JS file). The full download includes many files you typically don't need. For example **.map** files aren't needed unless you're using the Sass version of Bootstrap. In the CSS and JS files we also deleted a **sourceMappingURL** comment on the last line. It's not needed and can be confusing in some DevTools when you're not using Sass.
   - We added a standard **index.html** file, **img** and **snippets** folders.
3. For this exercise we'll be working with this **Bootstrap Starter** folder. Open the **Bootstrap Starter** folder in your code editor if it allows you to.
4. In your code editor, open **index.html**.

5. We need to link to Bootstrap's CSS file, which will provide us with the grid and components. Below the **title** tag, add the following link:

```
<title></title>
<link rel="stylesheet" href="css/bootstrap.min.css">
<link rel="stylesheet" href="css/main.css">
```

NOTE: We link to Bootstrap's CSS before our own, so we can override theirs as needed. We use the minified version because it's a smaller file. If you would only be using Bootstrap's grid (none of its components) link to **bootstrap-grid.min.css** (which in the full download) because it's much smaller.

Bootstrap's Grid: Containers, Rows, & Columns

Let's start with Bootstrap's grid using their pre-made classes, which are explained at getbootstrap.com/docs/5.0/layout/grid

Bootstrap's grid requires a specific nested structure: **container > row > column**.

1. Let's start by adding a **.container** div. Inside the **body** tag, add a **container** (as shown below in bold):

```
<body>
   <div class="container"></div>
</body>
```

2. Add a **row** inside the **container** div:

```
<div class="container">
   <div class="row"></div>
</div>
```

3. Add 2 **col** divs inside the **row**:

```
<div class="container">
   <div class="row">
      <div class="col"></div>
      <div class="col"></div>
   </div>
</div>
```

4. So that we can see our 2-column layout, add the same image into both columns:

```
<div class="container">
   <div class="row">
      <div class="col">
         <img src="img/large-low-lustre.jpg">
      </div>
      <div class="col">
         <img src="img/large-low-lustre.jpg">
      </div>
   </div>
</div>
```

5. Save the file and preview the page in your browser.
   - Whoa, the images are huge because it's a HiDPI image that has not been scaled down to fit the container. Let's fix that.
6. Switch back to your code editor.
7. Open **main.css** from the **css** folder (inside the **Bootstrap Starter** folder).
8. At the top of the file, add the following rule:

```
img {
   max-width: 100%;
}
```

9. Save the file and reload the page in your browser.
   - You should now see the 2 images next to each other, each filling out the size of their column (which are equally sized).
   - Resize the window and notice that as the columns get bigger or smaller, the images adjust to fill their column.
   - The container is a fixed-width, where the max-width changes at each breakpoint (screen size). There's also a fluid container, so let's see how that works next.

   NOTE: Bootstrap has an **img-fluid** class which makes an image responsive. Adding that class to all images is a lot more work compared to adding this one CSS rule.

10. Switch back to **index.html** in your code editor and add **-fluid** to the container:

```
<body>
   <div class="container-fluid">
```

11. Save the file and reload the page in your browser.
    - Resize the window to see the container fills 100% the screen, although the content has some space on the left and right sides because of column gutters. We'll look at spacing in a later exercise.
12. Switch back to your code editor.

13. We don’t want the fluid container, so remove **-fluid** as shown below:

```
<body>
   <div class="container">
```

Adjusting Column Widths

Bootstrap’s grid uses flexbox. Each **row** is a flex container. Each **col** is a flex item with **`flex-grow: 1;`** which means the columns are automatically sized to fill the space (as we saw when learning flexbox). Bootstrap has a 12-column grid that we can optionally use, so let’s see how that works.

1. Let’s make the left column take up 8 columns and the right column take up 4 columns (8+4=12). Add a **-8** to the **col** class for the first column:

```
<div class="container">
   <div class="row">
      <div class="col-8">
         <img src="img/large-low-lustre.jpg">
```

2. Save the file and reload the page in your browser.

- Notice the left column is now larger than the right. 12 columns, minus the 8 we just set equals 4 left over columns. The right column will automatically become 4 columns without us having to set it. If however the content did not fit naturally, flex wrapping would cause it to move down to the next line. In that case setting **col-4** on the right column would ensure it fits.

3. Resize the window to see the layout always has columns, even on small screens. We’d prefer the images stack on smaller screens.

Bootstrap uses a mobile first approach. To stack things on small screens, and get columns on larger screens, we specify the smallest screen size that should have columns. We get columns at that screen size and larger. Here are Bootstrap’s sizes.

Code	Size	Breakpoint
xs	Extra Small	Devices Smaller than 576 px
sm	Small	Devices 576 px & Larger
md	Medium	Devices 768 px & Larger
lg	Large	Devices 992 px & Larger
xl	Extra Large	Devices 1200 px & Larger
xxl	Extra Extra Large	Devices 1400 px & Larger

4. Switch back to your code editor and add a **-md** to the col class:

```
<div class="container">
   <div class="row">
      <div class="col-md-8">
         <img src="img/large-low-lustre.jpg">
```

5. Save the file and reload the page in your browser.

   - Resize the window from narrow to wide. When the window is less than 768px wide, the images will stack vertically. At 768px or wider the columns appear.
   - We can change the size of columns for any screen size we want. Let's see how.

6. Switch back to your code editor.

7. Let's make the left column even bigger, but only on the largest (xl) screen size. Add a **col-xl-10** class to make it 10 columns wide on extra large screens:

```
<div class="container">
   <div class="row">
      <div class="col-md-8 col-xl-10">
         <img src="img/large-low-lustre.jpg">
```

8. Save the file and reload the page in your browser. Resize the window from small to large and notice:

   - When the window is less than 768px wide, the images will stack vertically.
   - At 768px or wider the columns appear as 8 and 4 columns wide.
   - At 1200px or wider the columns appear as 10 and 2 columns wide.

9. Switch back to your code editor.

10. Change the sizes to the following, which will look good with the content we'll be adding next:

```
<div class="container">
   <div class="row">
      <div class="col-lg-8 col-xl-9">
         <img src="img/large-low-lustre.jpg">
```

Using the Bootstrap "Card" Component

Bootstrap has many components (navbars, buttons, etc.) which you can learn about at getbootstrap.com/docs/5.0/components In our right column we want to replace the image with text. Let's see how the **card** component will give us a bordered section with a title. Normally you'd copy some sample code from the Bootstrap website and edit it to your liking. Just in case the website changes, we copied the sample code from the Bootstrap website and pasted it into a file for you to use.

1. Open **card.html** from the **snippets** folder (inside the **Bootstrap Starter** folder).

2. Select and copy all the code.

3. Close the file.

4. In the second column, replace the image with the **card** code as shown below in bold:

```
<div class="row">
   <div class="col-lg-8 col-xl-9">
      <img src="img/large-low-lustre.jpg">
   </div>
   <div class="col">
      <div class="card">
         <div class="card-header">
            Featured
         </div>
         <div class="card-body">

            CODE OMITTED TO SAVE SPACE

          </div>
       </div>
   </div>
</div>
```

5. Save the file and reload the page in your browser.

   The classes on the various parts of the sample code (from the Bootstrap website) link up with Bootstrap's CSS to give us a nicely styled section adding a light gray background, a border, etc. Let's change the content to be appropriate for our page.

6. Switch back to your code editor.

7. Change the **card-header** div to an **h2** as shown below in bold:

```
<div class="col">
   <div class="card">
      <h2 class="card-header">
         Featured
      </h2>
      <div class="card-body">
```

   NOTE: We'll add an h1 somewhere else (later), so we're using an h2 here.

8. Change the text inside the **card-header** as shown below in bold:

```
<h2 class="card-header">
   Just Announced
</h2>
```

9. To save you some time we've already written the HTML code for the main content of this section. Open **table.html** from the **snippets** folder (inside the **Bootstrap Starter** folder).

10. Select and copy all the code.

11. Close the file.

12. You should be back in **index.html**. Delete the content inside **card-body** (the **h5**, **p**, and **a**) and paste in the new content as shown below:

```
<div class="card">
   <h2 class="card-header">
      Just Announced
   </h2>
   <div class="card-body">
      <table class="table">

         CODE OMITTED TO SAVE SPACE

      </table>
      Come see some amazing local bands!
   </div>
</div>
```

13. In the content you just pasted, notice that we've already added Bootstrap's **table** class to the table tag. This applies some nice formatting to the table.

14. Save the file and reload the page in your browser.
    - Now we have content appropriate for our page.
    - The lines between the rows of the table are from Bootstrap's CSS.
    - The heading is a bit too big, so let's make it smaller.

15. Switch back to your code editor.

16. Open **main.css** from the **css** folder (inside the **Bootstrap Starter** folder).

17. Below the **img** rule, add the following rule:

```
h2 {
   font-size: 1.7rem;
}
```

18. Save the file and reload the page in your browser.
    - The smaller heading fits better.
    - Resize the window to see how the "card" is below the photo on small screens, and in the right column on larger screens.

Adding a Footer

1. Switch back to **index.html** in your code editor.
2. Let's add second row for some footer content that will include the copyright and social links. A **row** class can be used on any tag, so in this case we'll use a **footer** tag. Near the bottom of the page, below the **row**, add the following bold code:

```
      </div>
      <footer class="row">

      </footer>
   </div>
</body>
```

NOTE: If you have a hard time keeping track of what each closing div tag refers to (a row, col, etc.), try installing the Visual Studio Code extension called **HTML End Tag Labels** which adds a comment after each closing tag telling you what it's closing.

3. Add 2 columns inside the footer:

```
<footer class="row">
   <div class="col"></div>
   <div class="col"></div>
</footer>
```

4. Add the following text inside the first column in the footer:

```
<footer class="row">
   <div class="col">
      <p>
         The Jive Factory, 580 Lispenard<br>
         <small>All Rights Reserved</small>
      </p>
   </div>
   <div class="col"></div>
</footer>
```

5. For the second column we'll use Bootstrap's list group. We copied Bootstrap's sample code from their website into a file for you (normally you'd get it straight from their website). In your code editor, open **list-group.html** from the **snippets** folder.
6. Select and copy all the code.
7. Close the file.

8. You should be back in **index.html**. In the footer's second column, paste the code as shown below.

```
<footer class="row">
   <div class="col">
      <p>
         The Jive Factory, 580 Lispenard<br>
         <small>All Rights Reserved</small>
      </p>
   </div>
   <div class="col">
      <div class="list-group">
         <a href="#" class="list-group-item list-group-item-action active"
aria-current="true">
            The current link item
         </a>
         CODE OMITTED TO SAVE SPACE
         <a href="#" class="list-group-item list-group-item-action disabled"
tabindex="-1" aria-disabled="true">A disabled link item</a>
      </div>
   </div>
</footer>
```

9. Inside the **list-group** we only want to keep the second **list-group-item**, so delete the others. You should end up with the following:

```
<div class="list-group">
   <a href="#" class="list-group-item list-group-item-action">A second link
item</a>
</div>
```

10. Replace the sample text with **Twitter**:

```
<a href="#" class="list-group-item list-group-item-action">Twitter</a>
```

11. Make 2 more links by copying/pasting the line so you get the following:

```
<a href="#" class="list-group-item list-group-item-action">Twitter</a>
<a href="#" class="list-group-item list-group-item-action">Twitter</a>
<a href="#" class="list-group-item list-group-item-action">Twitter</a>
```

12. Edit the content as shown below in bold:

```
<a href="#" class="list-group-item list-group-item-action">Twitter</a>
<a href="#" class="list-group-item list-group-item-action">Facebook</a>
<a href="#" class="list-group-item list-group-item-action">RSS</a></li>
```

13. Save the file and reload the page in your browser. Notice the following:

    - The two footer columns are equal width (half of the container).
    - The social links are a vertical list, but we want them to be horizontal.

14. Switch back to your code editor.

15. Add a **list-group-horizontal** to the **list-group** as shown below in bold:

    ```
    <div class="list-group list-group-horizontal">
    ```

16. Save the file and reload the page in your browser.

    - The social links would look better if they were aligned to the right of the page and not filling up half the width of the container.

17. The column the social links are in is a flex item. By default Bootstrap sets **flex-grow** to **1** on columns. We can change it to **0** so the column width won't grow. Bootstrap has a class to do this, so you don't have to write the CSS.

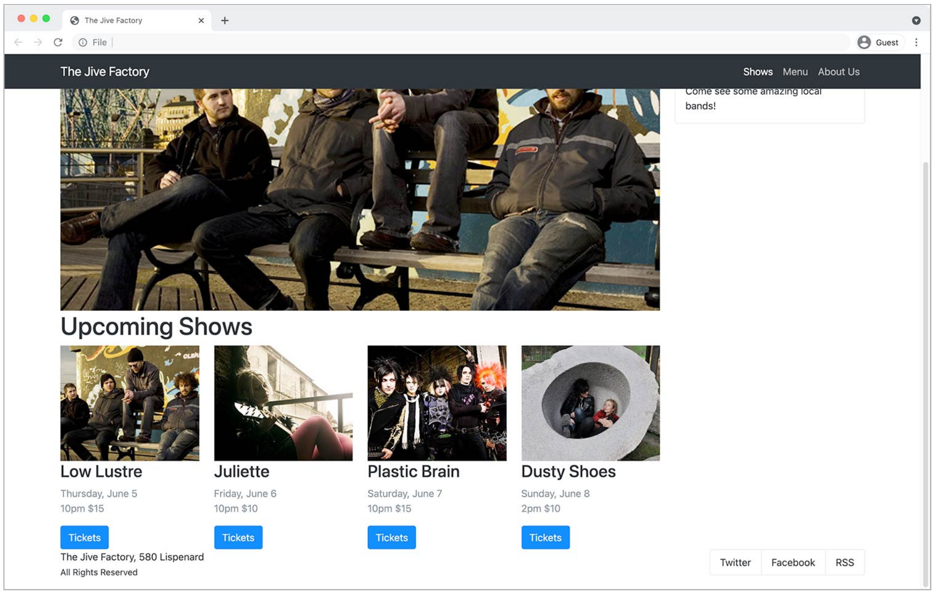
    Add a **flex-grow-0** class to the column that contains the social media list:

    ```
    <div class="col flex-grow-0">
       <div class="list-group list-group-horizontal">

          CODE OMITTED TO SAVE SPACE

       </div>
    </div>
    ```

18. Save the file and reload the page in your browser.

    - Now the social links should be aligned to the right and not streched out, which is how we want them.
    - The page needs some better spacing, and the footer does not look great on smaller screens, but we'll fix those issues in the next exercises.

Exercise Preview



Come see some amazing local bands!

Upcoming Shows

Low Lustre
Thursday, June 5
10pm $15

Juliette
Friday, June 6
10pm $10

Plastic Brain
Saturday, June 7
10pm $15

Dusty Shoes
Sunday, June 8
2pm $10

The Jive Factory, 580 Lispenard
All Rights Reserved

Photos courtesy of istockphoto, Hakan Çaglav, Image #14393929, Renee Keith, Image #7827478.

Exercise Overview

In this exercise, you'll learn more about Bootstrap's grid system and components.

Getting Started

The files for this exercise pick up where the previous exercise left off, but we'll start with a new folder in case you experimented with anything in that exercise.

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Bootstrap More Grids** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to.
3. Open **index.html** from the **Bootstrap More Grids** folder.
4. Preview **index.html** in a browser.
   - This page is the same as what you created in the previous exercise.
   - Below the photo, we want to add a heading and a row of additional photos.
5. Leave the page open in the browser and return to your code editor.

Nesting Grids

1. Switch to your code editor.

2. Below the image in the left column, add a heading:

```
<div class="row">
   <div class="col-lg-8 col-xl-9">
      <img src="img/large-low-lustre.jpg">
      <h1>Upcoming Shows</h1>
   </div>
```

3. Below the h1, add a row:

```
<div class="row">
   <div class="col-lg-8 col-xl-9">
      <img src="img/large-low-lustre.jpg">
      <h1>Upcoming Shows</h1>
      <div class="row"></div>
   </div>
```

4. Below the heading we want to add multiple columns, so let's create our first column. Inside the new **row** div, add our first column with a placeholder image:

```
<h1>Upcoming Shows</h1>
<div class="row">
   <div class="col">
      <img src="img/small-plastic-brain.jpg">
   </div>
</div>
```

5. Copy and paste that column so you have a total of 3 columns:

```
<h1>Upcoming Shows</h1>
<div class="row">
   <div class="col">
      <img src="img/small-plastic-brain.jpg">
   </div>
   <div class="col">
      <img src="img/small-plastic-brain.jpg">
   </div>
   <div class="col">
      <img src="img/small-plastic-brain.jpg">
   </div>
</div>
```

6. Save the file and reload the page in your browser.

   - Notice that the 3 images (below the **Upcoming Shows** heading) fit on a single line.

7. Switch back to **index.html** in your code editor.

8. Create a fourth column and image, by pasting one more column:

```
<div class="row">
   <div class="col">
      <img src="img/small-plastic-brain.jpg">
   </div>
   <div class="col">
      <img src="img/small-plastic-brain.jpg">
   </div>
   <div class="col">
      <img src="img/small-plastic-brain.jpg">
   </div>
   <div class="col">
      <img src="img/small-plastic-brain.jpg">
   </div>
</div>
```

9. Save the file and reload the page in your browser.

   - Notice how it automatically adjusts to 4 columns because the layout is using flexbox. Sweet!

   - We have more info we need to add for the upcoming shows (and a different image for each). To save you typing, we've prepared the HTML for you.

10. Switch back to your code editor.

11. Open **upcoming-shows.html** from the **snippets** folder.

12. Select and copy all the code.

13. Close the file.

14. You should be back in **index.html**. Select the 4 column divs containing the upcoming shows:

```
<div class="col">
   <img src="img/small-plastic-brain.jpg">
</div>
```

CODE OMITTED TO SAVE SPACE

```
<div class="col">
   <img src="img/small-plastic-brain.jpg">
</div>
```

15. Paste to replace it with the updated code.

16. Save the file and reload the page in your browser.

    - You should now have text below each of the 4 different band images. Notice the show date/time/price is a bit lighter than the other text, because we added a Bootstrap class of **text-muted** for you.
    - We don't want 4 columns on small screens. One column would be good on mobile, switching to 2 columns on slightly larger screens, and finally 4 columns on larger screens. To make 2 columns, each column will need to span 6 Bootstrap columns (6+6=12).

17. Switch back to your code editor.

18. Set the following sizes on all 4 columns under the Upcoming Shows:

```
<div class="col-sm-6 col-xl-3">
   <img src="img/small-low-lustre.jpg">

   CODE OMITTED TO SAVE SPACE

<div class="col-sm-6 col-xl-3">
   <img src="img/small-juliette.jpg">

   CODE OMITTED TO SAVE SPACE

<div class="col-sm-6 col-xl-3">
   <img src="img/small-plastic-brain.jpg">

   CODE OMITTED TO SAVE SPACE

<div class="col-sm-6 col-xl-3">
   <img src="img/small-dusty-shoes.jpg">
```

19. Save the file, reload the page in your browser, and notice:

    - By starting columns on **sm** (small) screens, **xs** (extra small) screens will not be columns and will therefore stack. On **sm** (small) screens and wider we have 2 columns. On **xl** (extra large) and wider we have 4 columns.
    - The images on mobile would look better if they scaled up to fill the entire width. We could do this via CSS, but Bootstrap has a class to do this as well.

20. Switch back to your code editor.

21. Add the following class on all 4 images:

```
<div class="col-sm-6 col-xl-3">
   <img src="img/small-low-lustre.jpg" class="w-100">
```

CODE OMITTED TO SAVE SPACE

```
<div class="col-sm-6 col-xl-3">
   <img src="img/small-juliette.jpg" class="w-100">
```

CODE OMITTED TO SAVE SPACE

```
<div class="col-sm-6 col-xl-3">
   <img src="img/small-plastic-brain.jpg" class="w-100">
```

CODE OMITTED TO SAVE SPACE

```
<div class="col-sm-6 col-xl-3">
   <img src="img/small-dusty-shoes.jpg" class="w-100">
```

22. Save the file and reload the page in your browser.

- Resize the window to see that when the upcoming shows stack, the images are now the full width of the page. That looks much better.
- The **Tickets** links would be nicer if they looked like buttons. Luckily, Bootstrap has classes we can use.

23. Switch back to your code editor.

24. Add the following classes to each of the 4 **Tickets** links:

```
<a href="#" class="btn btn-primary">Tickets</a>
```

NOTE: There are other button styles (btn-secondary, btn-warning, btn-danger, and more) which you can see at getbootstrap.com/docs/5.0/components/buttons.

25. Save the file and reload the page in your browser to see that the Tickets links now look like buttons.

Showing & Hiding an Element at Specific Sizes

1. Switch back to your code editor.

2. For small screens, hide the large image that's above the **Upcoming Shows** h1 by adding the following classes.

```
<img class="d-none d-lg-block" src="img/large-low-lustre.jpg">
<h1>Upcoming Shows</h1>
```

NOTE: The **d-none** sets display to none on all screens. The **d-lg-block** changes display to block on **lg** and **xl** screens.

3. Save the file and reload the page in your browser.

   - Notice the large image above the **Upcoming Shows** heading is hidden on smaller screens, and visible on larger screens.

Adding a Navbar

Let's add the website's navigation at the top of the page. To start, we'll use the navbar code we copied from the Bootstrap website.

1. Switch back to your code editor and open **navbar.html** from the **snippets** folder.

   NOTE: This (and all other Bootstrap code) is found on Bootstrap's website. The navbar code can be found at **getbootstrap.com/docs/5.0/components/navbar**

2. Select and copy all the code.

3. Close the file.

4. You should be back in **index.html**.

5. Paste the code after the start of the body tag, as shown below:

```
<body>
   <nav class="navbar navbar-expand-lg navbar-light bg-light">

      CODE OMITTED TO SAVE SPACE

   </nav>
   <div class="container">
      <div class="row">
```

6. This navbar collapses on smaller sized screens, creating a menu icon that expands when you click on it. This effect uses JavaScript, so we need to add the Bootstrap JS file to our page.

   At the bottom of the page, add the following link:

```
   </div>
   <script src="js/bootstrap.min.js"></script>
</body>
```

7. Save the file and reload the page in your browser.

   - Resize the window to see that on smaller screens the navbar will collapse to a 3-lined menu button (often called a hamburger icon because it's like a hamburger patty in a bun) at the top right of the page.
   - Click the hamburger icon a couple times to expand and collapse it.

8. Switch back to your code editor.

9. Let's start replacing the placeholder content with real text, and removing things we don't need.

   Replace the **Navbar** placeholder title with **The Jive Factory** as shown below:

   ```
   <div class="container-fluid">
      <a class="navbar-brand" href="#">The Jive Factory</a>
   ```

10. As shown below, replace **Home** and **Link**, with **Shows** and **Menu**:

    ```
    <ul class="navbar-nav me-auto mb-2 mb-lg-0">
       <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Shows</a>
       </li>
       <li class="nav-item">
          <a class="nav-link" href="#">Menu</a>
       </li>
    ```

11. The **ul** tag **navbar-nav** you're working in has 4 **li** tags.

    - Keep the first 2 **li** tags (which contain the **Shows** and **Menu** links we just made).
    - Delete the next 2 **li** tags (the **nav-item dropdown** and the last **li** tag which contains a **disabled** link).
    - Your list should end up with only 2 **li** tags, as shown below:

    ```
    <ul class="navbar-nav me-auto mb-2 mb-lg-0">
       <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Shows</a>
       </li>
       <li class="nav-item">
          <a class="nav-link" href="#">Menu</a>
       </li>
    </ul>
    ```

12. Let's make one more link. Copy and paste the **Menu** list item:

    ```
    <ul class="navbar-nav me-auto mb-2 mb-lg-0">
       <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Shows</a>
       </li>
       <li class="nav-item">
          <a class="nav-link" href="#">Menu</a>
       </li>
       <li class="nav-item">
          <a class="nav-link" href="#">Menu</a>
       </li>
    </ul>
    ```

13. In the second **Menu** list item, change the text to **About Us**:

```
<li class="nav-item">
   <a class="nav-link" href="#">Menu</a>
</li>
<li class="nav-item">
   <a class="nav-link" href="#">About Us</a>
</li>
```

14. Below the list, we don't need the search form, so delete the entire **form** tag.

15. Save the file and reload the page in your browser.

    - Resize the browser window and notice the hamburger menu appears at a screen size that is still capable of showing the menu items we have. We can change this.

16. On the **nav** tag, change the **navbar-expand** size from **lg** to **sm** as shown below:

```
<body>
   <nav class="navbar navbar-expand-sm navbar-light bg-light">
```

17. Save the file and reload the page in your browser.

    - Resize the window to see that the hamburger icon now only displays at smaller screens, so we see the 3 nav links at more screen sizes.

18. Switch back to your code editor.

19. Switch to a dark navbar by changing **light** to **dark** (in 2 places):

```
<nav class="navbar navbar-expand-sm navbar-dark bg-dark">
```

20. Save the file and reload the page in your browser.

    - The navbar should now be a dark gray.
    - On wide screens we'd like the 3 nav links to be on the right side.

21. Switch back to your code editor.

22. On the **navbar-nav** list, change **me** (which stands for margin end… like flex end versus start) to **ms** (which stands for margin start) and shown below:

```
<ul class="navbar-nav ms-auto mb-2 mb-lg-0">
```

23. Delete the **mb** (margin bottom) classes so you end up with the following:

```
<ul class="navbar-nav ms-auto">
```

24. Save the file and reload the page in your browser.

    - The 3 nav links should now be on the right side of wide screens.

25. Switch back to your code editor.

26. Make the navbar stick to the top of the screen by adding **sticky-top** to the **nav** tag:

```
<nav class="navbar navbar-expand-sm navbar-dark bg-dark sticky-top">
```

27. Save the file and reload the page in your browser.
    - Make the page short enough to scroll.
    - As you scroll notice the navbar remains fixed to the top of the screen.
    - Moving onto a different issue… make the window wide enough so you can see how the navbar's content extends to the far left/right and does not align with our columns of content. You may like that, but let's see how to make it align if that's what you prefer.

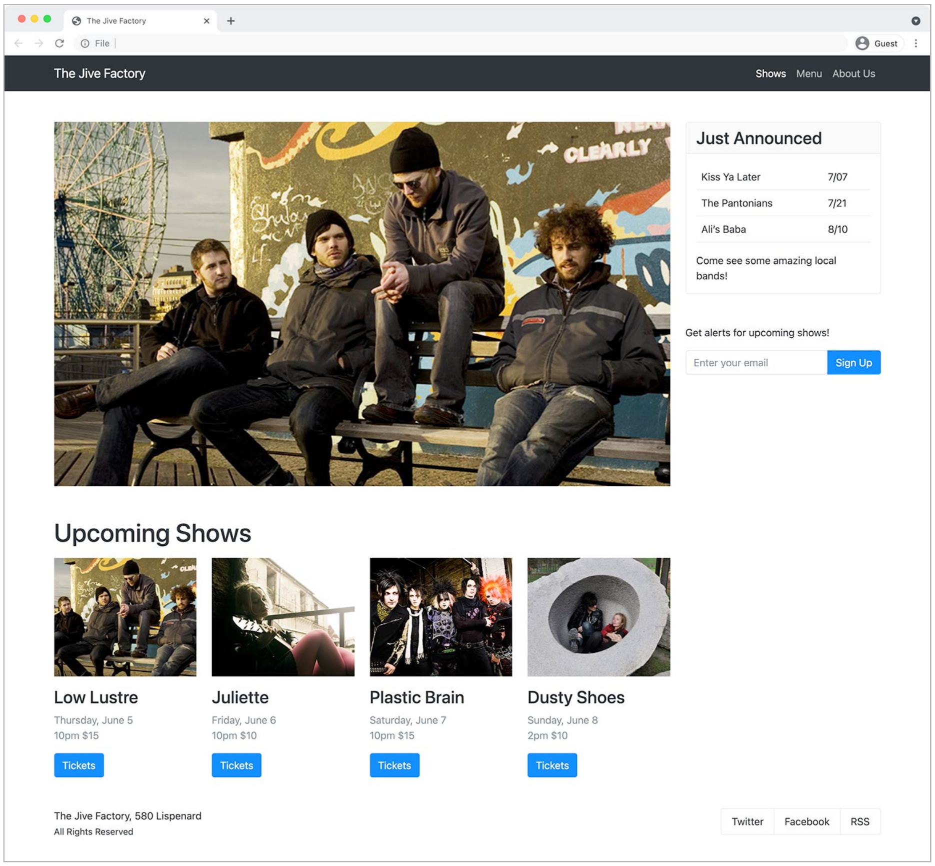
28. Switch back to your code editor.

29. To constrain the navbar's content to the same width as the page's content, change **container-fluid** to **container** as shown below:

```
<nav class="navbar navbar-expand-sm navbar-dark bg-dark sticky-top">
   <div class="container">
```

30. Save the file and reload the page in your browser.
    - The navbar's content should now align with the columns on the page. How you want to style your navbars is up to you, so this change is optional. We're just showing you some options.
    - Isn't it cool what we could do using only Bootstrap's provided CSS and JS? So far we've done everything by only writing HTML (except for 2 basic CSS rules)!

Exercise Preview



The Jive Factory

Shows Menu About Us

Just Announced

Kiss Ya Later	7/07
The Pantonians	7/21
Ali's Baba	8/10

Come see some amazing local bands!

Get alerts for upcoming shows!

Enter your email | Sign Up

Upcoming Shows

Low Lustre
Thursday, June 5
10pm $15
Tickets

Juliette
Friday, June 6
10pm $10
Tickets

Plastic Brain
Saturday, June 7
10pm $15
Tickets

Dusty Shoes
Sunday, June 8
2pm $10
Tickets

The Jive Factory, 580 Lispenard
All Rights Reserved

Twitter | Facebook | RSS

Photos courtesy of istockphoto, Hakan Çaglav, Image #14393929, Renee Keith, Image #7827478.

Exercise Overview

In this exercise you'll look at adjusting spacing, changing the layout across screens, and more Bootstrap components.

Getting Started

The files for this exercise pick up where the previous exercise left off, but we'll start with a new folder in case you experimented with anything in that exercise.

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Bootstrap Spacing and Layout** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to.
3. Open **index.html** from the **Bootstrap Spacing and Layout** folder.

4. Preview **index.html** in a browser.
   - This page is the same as what you created in the previous exercise.
5. Leave the page open in the browser and return to your code editor.

Adding an Email Signup Form

We need to add an email signup form in the right column and can use some Boostrap components to make it look better.

1. Switch to your code editor.
2. We've typed up some HTML to save you time (and included some sample code from the Bootstrap website). In your code editor, open **input-group.html** from the **snippets** folder.
3. Select and copy all the code.
4. Close the file.
5. You should be back in **index.html**.
6. Below the **card** (inside the same column as the card), paste the code:

```
            Come see some amazing local bands!
         </div>
      </div>
      <form action="">
         <p>Get alerts for upcoming shows!</p>
         <div class="input-group mb-3">
            <input type="text" class="form-control" placeholder="Recipient's
username" aria-label="Recipient's username" aria-describedby="button-addon2">
            <button class="btn btn-outline-secondary" type="button" id="button-
addon2">Button</button>
         </div>
      </form>
   </div>
</div>
<footer class="row">
```

7. Save the file and reload the page in your browser.
   - In the right column (on wide screens) there should be a nice looking text field and a button.
   - Click into the text field and notice it now glows with a subtle CSS transition.
   - We need to change the text to suit our needs, and make the button's appearance stand out more.

8. Let's change the button's appearance by changing its class. Return to your code editor and change the following bold code:

   ```
   <button class="btn btn-primary" type="button" id="button-addon2">Button</
   button>
   ```

9. Save the file and reload the page in your browser.

   - The button is now colored so it stands out more.

10. Return to your code editor.

11. Change the content to be appropriate for an email sign up form, by making the edits shown below in bold:

    ```
    <input type="email" class="form-control" placeholder="Enter your email" aria-
    label="Enter your email" aria-describedby="button-addon2">
    <button class="btn btn-primary" type="button" id="button-addon2">Sign Up</
    button>
    ```

12. Save the file and reload the page in your browser.

    - The text field should now say **Enter your email** and have a **Sign Up** button.

Margins

Now let's improve the spacing between elements. (Refer to getbootstrap.com/docs/5.0/utilities/spacing for all the details on margin and padding.)

1. Return to your code editor.

2. Add margin space below the navbar by adding the following classes to it:

   ```
   <nav class="navbar navbar-expand-sm navbar-dark bg-dark mb-3 mb-sm-5 sticky-
   top">
   ```

   NOTE: The **mb** stands for **margin-bottom**. The lack of a size on **mb-3** means it applies to all screen sizes. The **mb-sm-5** increases the amount of space starting on small screens and larger (which will override the **mb-3** at those screen sizes).

3. Save the file and reload the page in your browser.

   - Notice the improved spacing below the navbar.
   - The **Upcoming Shows** heading could use some space around it.

4. Return to your code editor.

5. Add space above and below the **h1** by adding the following classes:

   ```
   <h1 class="mt-lg-5 mb-3">Upcoming Shows</h1>
   ```

6. Save the file and reload the page in your browser.

   - Notice the improved spacing around the **Upcoming Shows** heading.

   - The 4 images below the **Upcoming Shows** heading could use some space below them.

7. Return to your code editor.

8. Add bottom margin to all 4 images by adding the **mb-3** class to each:

```
<div class="col-sm-6 col-xl-3">
   <img src="img/small-low-lustre.jpg" class="w-100 mb-3">

   CODE OMITTED TO SAVE SPACE

</div>
<div class="col-sm-6 col-xl-3">
   <img src="img/small-juliette.jpg" class="w-100 mb-3">

   CODE OMITTED TO SAVE SPACE

</div>
<div class="col-sm-6 col-xl-3">
   <img src="img/small-plastic-brain.jpg" class="w-100 mb-3">

   CODE OMITTED TO SAVE SPACE

</div>
<div class="col-sm-6 col-xl-3">
   <img src="img/small-dusty-shoes.jpg" class="w-100 mb-3">
```

9. Save the file and reload the page in your browser.

   - Notice the improved space below the 4 images.

   - We need some more space above and below the Sign Up form.

10. Return to your code editor.

11. Instead of adding both margin top (**mt**) and margin bottom (**mb**), we can use **my**, which is margin on the **y**-axis (both top and bottom). Add the following class to the form:

```
<form class="my-5" action="">
```

12. Save the file and reload the page in your browser.

    - Notice the form now has space above and below it (the space below is needed on small screens where the form has the footer below it).

Gutters

While you can adjust the spacing of specific elements with margin and padding, each row can specify a gutter amount for the columns inside it. You specify horizontal gutters with a **gx** class, vertical with a **gy** class or both horizontal and vertical with a **g** class. To learn more visit **getbootstrap.com/docs/5.0/layout/gutters**

1. Return to your code editor.
2. Increase the gutters around all 4 images by adding the **mb-3** class to each:

```
<h1 class="mt-lg-5 mb-3">Upcoming Shows</h1>
<div class="row g-5">
```

3. Save the file and reload the page in your browser.
   - Notice the increased space between the columns/rows of 4 images below Upcoming Shows.
   - The space looks good on small screens when the columns stack, but is a bit too much on larger screens. Let's reduce it a bit.
4. Add the **g-md-4** class as shown below in bold:

```
<h1 class="mt-lg-5 mb-3">Upcoming Shows</h1>
<div class="row g-5 g-sm-4">
```

5. Save the file and reload the page in your browser.
   - The gutters change from a 5 gutter amount (large gutters) on the xs (extra small) screens, to a 4 gutter (slightly smaller gutters) on sm (small) and wider screens.
   - Make the window narrow so you can see there's a horizontal scrollbar that allows us to scroll left/right just a few pixels. Bootstrap's website says "The .container or .container-fluid parent may need to be adjusted if larger gutters are used too to avoid unwanted overflow".
6. Return to your code editor.
7. As instructed by Bootstrap's website, add an **overflow-hidden** class to the container:

```
</nav>
<div class="container overflow-hidden">
    <div class="row">
```

8. Save the file and reload the page in your browser.
   - The horizontal scrollbar should now be gone.
   - Make the window narrow enough so you can see the footer (which includes the social links) doesn't look good when it's 2 columns on extra small screens.

Changing the Footer Layout Across Screen Sizes

Let's make the footer look better on mobile.

1. Return to your code editor.
2. We don't want the footer to have columns on smaller screens. On both of the footer's 2 columns, add **-md** as shown below in bold:

```
<footer class="row">
   <div class="col-md">

      CODE OMITTED TO SAVE SPACE

   </div>
   <div class="col-md flex-grow-0">

      CODE OMITTED TO SAVE SPACE

   </div>
</footer>
```

3. Save the file and reload the page in your browser.
   - The footer is now a single column on smaller screens (the address and social items now stack).
   - The social media buttons have grown to fill the entire width and are too wide.
   - It would look nicer if the footer content was centered.
4. Return to your code editor.
5. Center the footer's text on smaller screens by adding the following classes:

```
<footer class="row text-center text-md-start">
```

6. Save the file and reload the page in your browser.
   - The footer's text is now centered on small screens, but the buttons are still too wide.
7. Return to your code editor.
8. Flex elements (like block elements) fill up the entire width of their container. Alternatively inline-flex elements (like inline elements) collapse to the width of their content (and they follow their parent element's text alignment). Style the social network list-group to as inline-flex by adding the following class:

```
<div class="list-group list-group-horizontal d-inline-flex">
   <a href="#" class="list-group-item list-group-item-action">Twitter</a>
```

NOTE: The **d-** stands for **display**.

9. Save the file and reload the page in your browser.
   - The footer layout on small screens is looking much better!
   - What if we wanted **The Jive Factory, 580 Lispenard** to be below the social networks? We can use flexbox to reorder the elements.
10. Return to your code editor.
11. Add the following classes to change the order:

```
<footer class="row text-center text-md-start">
   <div class="col-md order-1 order-md-0">
```

12. Save the file and reload the page in your browser.
   - On mobile, notice **The Jive Factory, 580 Lispenard** is below the social networks.
   - Now we need some space below the social networks.
   - Make the window wider to see that the footer also needs space above.
13. Return to your code editor.
14. Add the following classes to adjust the gutters and add more margin space above:

```
<footer class="row text-center text-md-start g-4 mt-4">
   <div class="col-md order-1 order-md-0>
```

15. Save the file and reload the page in your browser.
   - Resize the window to see that the footer's margins look across all screen sizes.

Optional Bonus: Revisiting the Fluid Container

1. Return to your code editor.
2. Below the **nav** tag, find the container and add **-fluid** as shown below:

```
</nav>
<div class="container-fluid">
```

3. Save the file and reload the page in your browser.
   - Resize the browser to see how the page behaves.
4. Return to your code editor.

5. We prefer a non-fluid container for this page, so let's switch back. Remove the **-fluid** you just added, so you end up with:

   ```
   </nav>
   <div class="container">
   ```

   NOTE: Which style you like is personal preference and how it works with the site's content. We prefer the regular container for this site. Also, there's no limit to how wide the page can get, but most likely you'd want to add a limit. We'd suggesting adding the following rule in **main.css** and adjusting the size to whatever you desire:

   ```
   .container-fluid {
      max-width: 1400px;
   }
   ```

Optional Bonus: Borders & Rounded Corners

1. Try adding the classes **border border-secondary rounded** to all the images to add a gray rounded border.
2. Save the file and reload the page in your browser.

   We've only scratched the surface of the components available in Bootstrap. You can learn about them all (with code examples) at **getbootstrap.com**

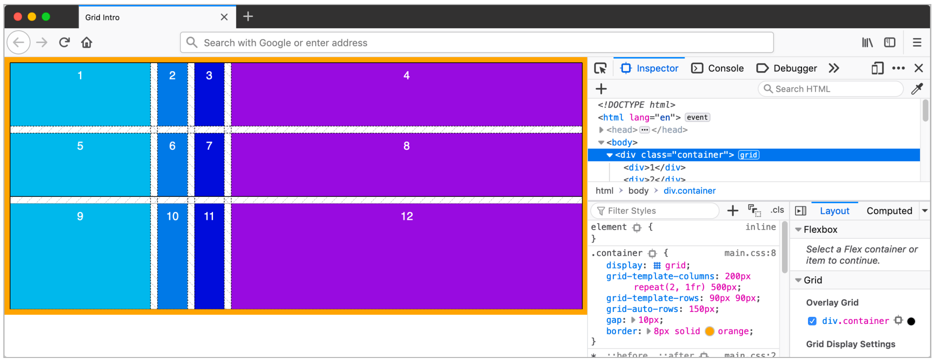
   Even if you don't use the pre-made components, and only use Bootstrap for a responsive grid, it can be useful.

Optional Bonus Challenge: Finishing the Margin Spacing

Use what you've learned to add space (margin) above the **Just Announced** card only when it moves under the other content (and not on wider screens when it's in the right column).

Intro to Grid

Exercise Preview



```
<!DOCTYPE html>
<html lang="en"> event
<head>…</head>
<body>
  <div class="container"> grid
    <div>1</div>
    <div>2</div>
```

```
element {                          inline
}
.container {                     main.css:8
  display: grid;
  grid-template-columns: 200px
      repeat(2, 1fr) 500px;
  grid-template-rows: 90px 90px;
  grid-auto-rows: 150px;
  gap: 10px;
  border: 8px solid orange;
}
```

Exercise Overview

In this exercise you'll get started with CSS **grid**, which is a way to lay out content in columns and rows. It's very different than other CSS layout techniques and completely changes how we approach laying out a webpage.

When to Use Flexbox & Grid

Flexbox and grid are different approaches to laying out content, and can be used together. While grid is newer, it does not replace flexbox. There are scenarios in which you could use both, so both are useful to know.

On the CSS Tricks article **What's the Difference Between Flexbox and Grid?** Beverly @aszenitha commented "Use grid when you already have the layout structure in mind, and flex when you just want everything to fit. Layout first vs content first." You can read that article at **tinyurl.com/flex-grid-dif**

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Grid Intro** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Grid Intro** folder.
   - Notice there's a **.container** div that wraps a bunch of divs with numbers.

4. Preview **index.html** in Firefox. We'll be using Firefox because (at the time of this writing) Firefox has better DevTools for inspecting CSS grid.
   - Notice the **.container** div has an orange border (so you can more easily see its size) and we've given different background colors to the numbered divs inside.
   - Notice the numbered divs currently take up the entire width of their parent container div.
5. Leave **index.html** open in Firefox as you work, so you can reload the page to see the code changes you'll make.

Display Grid

1. Switch back to your code editor.
2. Open **main.css** from the **css** folder (in the **Grid Intro** folder).
3. In the **.container** rule add the following bold code:

```
.container {
   display: grid;
   border: 8px solid orange;
}
```

4. Save the file, and reload the page in Firefox.
   - Notice that nothing changed. That's because we have not told grid how many columns we want.
5. Switch back to your code editor.
6. In the **.container** rule add the following bold code:

```
.container {
   display: grid;
   grid-template-columns: 200px 200px;
   border: 8px solid orange;
}
```

   NOTE: Can also use other measurements such as rem, em, %, etc. Later we'll cover the **fr** unit which is often a better option than percentages.

7. Save the file, and reload the page in Firefox.
   - You should now see the numbered divs have formed 2 columns.
   - Notice the container div's orange border still fills the entire width of the window, but the divs inside do not fill their container (leaving white space on the right).
8. Switch back to your code editor.

9. In the **.container** rule add the following bold code:

```
.container {
   display: grid;
   grid-template-columns: 200px 200px;
   gap: 10px;
   border: 8px solid orange;
}
```

NOTE: Early versions of grid used the **grid-gap** property, which was later replaced by the **gap** property. The **gap** property is shorthand for **row-gap** and **column-gap**.

- If one value is given, it will be used for both row and column gaps.
- If two values are given, the first is for row-gap and the second is for column-gap. So **`gap: 10px 30px;`** would give 10px row-gap and 30px column-gap.

10. Save the file, and reload the page in Firefox.

- The numbered divs should now have 10px of white space between them.

11. Switch back to your code editor.

12. In the **.container** rule add the following bold code:

```
.container {
   display: grid;
   grid-template-columns: 200px auto 200px;
   gap: 10px;
   border: 8px solid orange;
}
```

13. Save the file, and reload the page in Firefox.

- You should now have 3 columns.
- The left and right columns should be a fixed 200px wide, with the middle column filling the remaining space between.

14. Switch back to your code editor.

15. In the **.container** rule add the following bold code:

```
.container {
   display: grid;
   grid-template-columns: 200px auto 200px;
   grid-template-rows: 90px 90px;
   gap: 10px;
   border: 8px solid orange;
}
```

16. Save the file, and reload the page in Firefox.

- We defined 2 rows, so the first 2 rows are both 90px tall. The other rows have not been changed.
- The columns and rows we define are called the explicit grid.
- The columns and rows that are automatically created are called the implicit grid.
- Currently all our columns are explicit. The first 2 rows are explicit, and the rest are implicit.

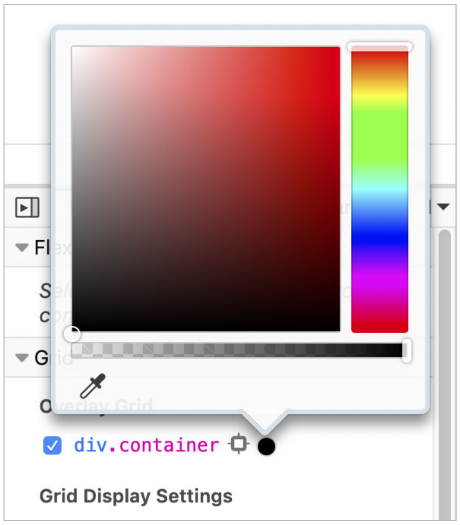
17. **Ctrl–click** (Mac) or **Right–click** (Windows) anywhere in the grid and choose **Inspect Element**.

18. In the DevTools, to the right of **`<div class="container">`** click the **grid** button to show a grid overlay.

19. Click the **grid** button again to turn off the grid overlay.

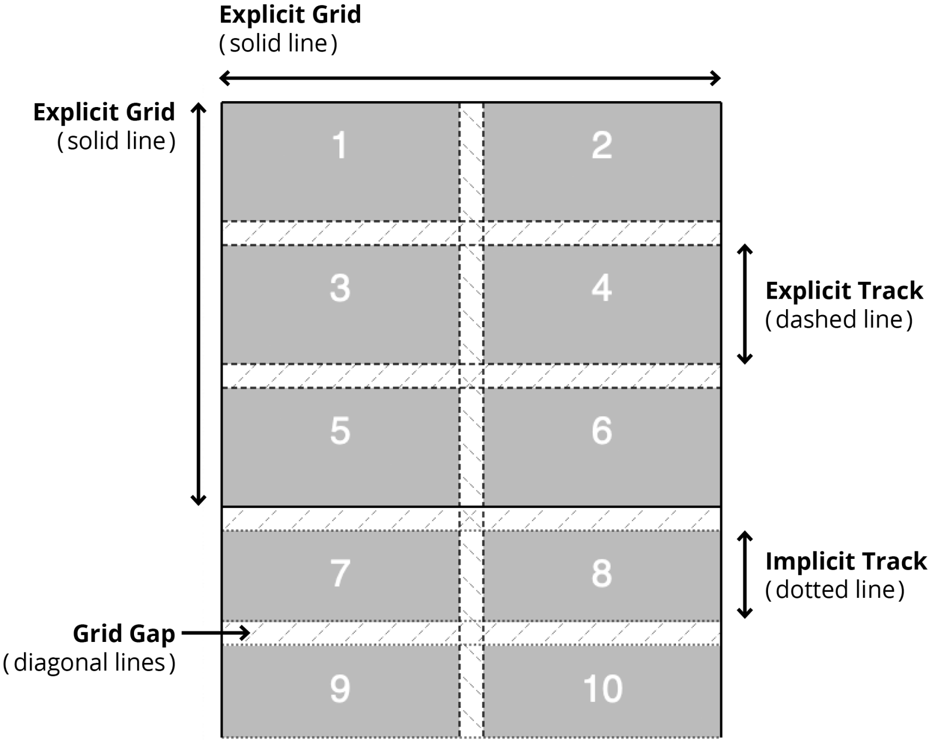
20. In the **Grid** area of the DevTools, it lists all the grid containers in the page.

- You should see **div.container** with a checkbox.
- Check the box for **div.container** to turn on the overlay (this is the same as clicking the **grid** button.)
- To the right of **div.container** is a colored dot. As shown below, click that dot to choose the color of the overlay and choose black (or whatever color makes them easiest for you to see).



The Explicit vs. Implicit Grid

Here's what the various lines in Firefox's grid overlay mean:



1. Switch back to your code editor.
2. Let's set the size for implicit rows. In the **.container** rule add the following bold code:

```
.container {
   display: grid;
   grid-template-columns: 200px auto 200px;
   grid-template-rows: 90px 90px;
   grid-auto-rows: 150px;
   gap: 10px;
   border: 8px solid orange;
}
```

3. Save the file, and reload the page in Firefox.
   - Notice the first 2 rows remain the same (because they are explicit), but the other row implicit rows are now taller (150px each).
4. Switch back to your code editor.

5. In the **.container** rule, change the size of the columns as follows:

```
.container {
   display: grid;
   grid-template-columns: 200px 1fr;

   CODE OMITTED TO SAVE SPACE
}
```

NOTE: The **fr** unit is a **fr**actional unit. According to the spec **tinyurl.com/fr-unit** an **fr** unit "represents a fraction of the leftover space in the grid container". So another way to think of **fr** is that it stands for "free space". First the browser calculates the size of grid items with specific widths (such as pixels), then takes the leftover free space and distributes it between the grid tracks with fr units.

6. Save the file, and reload the page in Firefox.
   - You should now have 2 columns.
   - The left column should be 200px wide.
   - The right column should fill the remaining space.
7. Switch back to your code editor.
8. In the **.container** rule, change the size of the columns as follows:

```
.container {
   display: grid;
   grid-template-columns: 200px 1fr 1fr;

   CODE OMITTED TO SAVE SPACE
}
```

9. Save the file, and reload the page in Firefox.
   - You should now have 3 columns.
   - The left column should be 200px wide.
   - The other 2 columns should equally fill the remaining space.
10. Switch back to your code editor.
11. In the **.container** rule, change the size of the columns as follows:

```
.container {
   display: grid;
   grid-template-columns: 200px 2fr 1fr;

   CODE OMITTED TO SAVE SPACE
}
```

NOTE: 2fr gets twice the amount of free space as 1fr.

12. Save the file, and reload the page in Firefox.
    - You should still have 3 columns.
    - The left column should be 200px wide.
    - The middle column should be larger than the right column. The free space is divided into 3 units, with the middle column getting 2 units and the right column getting 1 unit.
13. Switch back to your code editor.
14. In the **.container** rule, change the size of the columns as follows:

```
.container {
   display: grid;
   grid-template-columns: auto 1fr;
   CODE OMITTED TO SAVE SPACE
}
```

    NOTE: In this case, auto collapses to width of content.

15. Save the file, and reload the page in Firefox.
    - You should now have 2 columns.
    - The left column should be the width of content.
    - The right column should fill the remaining space.
16. Switch back to your code editor.
17. In the **.container** rule, change the size of the columns as follows:

```
.container {
   display: grid;
   grid-template-columns: 1fr 1fr 1fr;
   CODE OMITTED TO SAVE SPACE
}
```

18. Save the file, and reload the page in Firefox.
    - You should now have 3 equally sized columns.
19. Switch back to your code editor.
20. In the **.container** rule, change the size of the columns as follows:

```
.container {
   display: grid;
   grid-template-columns: repeat(3, 1fr);
   CODE OMITTED TO SAVE SPACE
}
```

21. Save the file, and reload the page in Firefox.
    - Nothing should have changed, this was just an alternate way of coding.
22. Switch back to your code editor.
23. In the **.container** rule, change the size of the columns as follows:

```
.container {
   display: grid;
   grid-template-columns: repeat(3, 1fr 50px);
   CODE OMITTED TO SAVE SPACE
}
```

24. Save the file, and reload the page in Firefox.
    - You should now have 6 columns.
    - The **1fr 50px** repeats 3 times, so the columns should alternate between the flexible 1fr size and the fixed 50px size.
25. Switch back to your code editor.
26. In the **.container** rule, change the size of the columns as follows:

```
.container {
   display: grid;
   grid-template-columns: 100px repeat(2, 1fr) 500px;
   CODE OMITTED TO SAVE SPACE
}
```

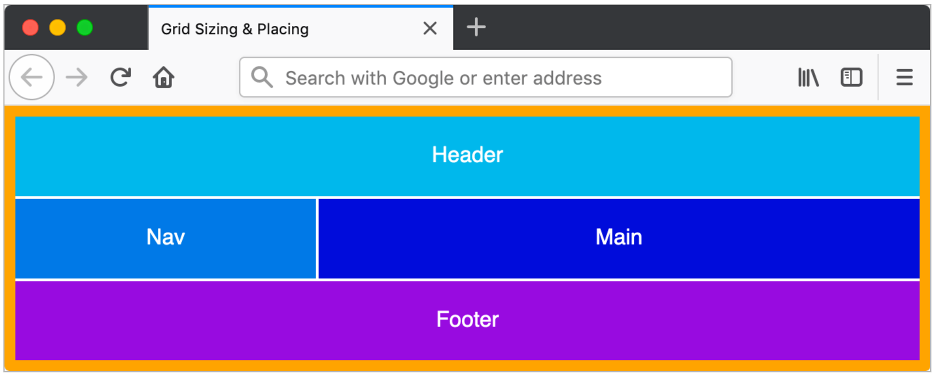
27. Save the file, and reload the page in Firefox.
    - You should now have 4 columns.
    - The left column is 100px wide.
    - The right column is 500px wide.
    - The 2 columns between are flexible and equally share the available space.

Grid Auto Flow

Similar to flexbox, grid has a direction. By default, **grid-auto-flow** is set to **row**. As you've seen in this exercise, grid items wrap onto multiple lines creating rows.

You can use **`grid-auto-flow: column;`** to make grid add new **columns** instead of new **rows**. You can then use **grid-auto-columns** to set a size for the implicit columns (like we did with grid-auto-rows to set a size for the implicit rows).

Exercise Preview



Exercise Overview

In this exercise you'll learn about placing elements within the grid and sizing them.

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Grid Sizing and Placing** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Grid Sizing and Placing** folder.
   - Notice a **.container** div wraps 4 elements: **header**, **nav**, **main**, and **footer**.
4. Preview **index.html** in Firefox.
   - Notice the **.container** div has an orange border (so you can more easily see its size) and we've given different background colors to the elements inside.
   - Notice the **header**, **nav**, **main**, and **footer** currently take up the entire width of their parent container div.
5. Leave **index.html** open in Firefox as you work, so you can reload the page to see the code changes you'll make.

Starting the Grid

1. Switch back to your code editor.
2. Open **main.css** from the **css** folder (in the **Grid Sizing and Placing** folder).

3. In the **.container** rule add the following bold code:

```
.container {
   display: grid;
   grid-template-columns: 1fr 2fr;
   grid-template-rows: 90px 90px 90px;
   gap: 2px;
   border: 8px solid orange;
}
```

4. Save the file, and reload the page in Firefox.

   - You should have 2 columns (the right column is wider than the left).
   - You should have 3 rows, with the last row being empty (we'll fill it in a moment).

Spanning Columns

1. Switch back to your code editor.
2. In the **header** rule add the following bold code:

```
header {
   grid-column: span 2;
   background: #0db8e8;
}
```

3. Save the file, and reload the page in Firefox.

   - The header should now span both columns.
   - Notice the footer has been pushed down into the third row, which had been empty.

4. Switch back to your code editor.
5. In the **footer** rule add the following bold code:

```
footer {
   grid-column: span 2;
   background: #9700da;
}
```

6. Save the file, and reload the page in Firefox.

   - The footer should now span both columns.

7. **Ctrl–click** (Mac) or **Right–click** (Windows) anywhere in the grid and choose **Inspect Element**.
8. In the DevTools, to the right of **`<div class="container">`** click the **grid** button to show a grid overlay.

9. Leave the DevTools open with the grid overlay on.
10. Switch back to your code editor.
11. In the **header** rule add the following bold code:

```
header {
   grid-column: span 4;
   background: #0db8e8;
}
```

12. Save the file, and reload the page in Firefox.
    - Even though we didn't explicitly define 4 columns, because we said to span 4 columns it had to add 2 more columns (as you can see in the overlay).
13. Switch back to your code editor.
14. Undo the change to go back to a span of 2:

```
header {
   grid-column: span 2;
   background: #0db8e8;
}
```

Spanning Rows

1. In the **nav** rule add the following bold code:

```
nav {
   grid-row: span 2;
   background: #0978e2;
}
```

2. Save the file, and reload the page in Firefox.
    - Now that the nav is spanning 2 rows, the footer can't fit beside it. Therefore a new 4th row was created. It an implicit row so it doesn't get the same height we explicitly set for our 3 rows.
    - You do not have to fill all grid spaces! Later we'll tuck the **Footer** into the empty space below **Main**, but it's OK to leave them empty if you want.

3. In the **Grid** area of the Firefox DevTools, under **Grid Display Settings** check on **Display line numbers**.

   - Notice that each line has a number.
   - Notice how the lines are counted with **positive** numbers from left to right (and top to bottom).
   - Notice how the lines are counted with **negative** numbers in the opposite direction: from right to left (and bottom to top).

Placing & Sizing Using Numbered Grid Lines

1. Switch back to your code editor.
2. In the **header** rule, replace **grid-column** with the following bold code:

   ```
   header {
      grid-column-start: 1;
      grid-column-end: 2;
      background: #0db8e8;
   }
   ```

   NOTE: These numbers refer to the column line numbers (not column numbers like you might guess). So 1 is the first line, and 2 is the second line.

3. Save the file, and reload the page in Firefox.

   - The header should now take occupy only one column, because it starts at line 1 (the leftmost line) and end at line 2.

4. Switch back to your code editor.
5. In the **header** rule, change the start and end values as shown below:

   ```
   header {
      grid-column-start: 2;
      grid-column-end: 3;
      background: #0db8e8;
   }
   ```

6. Save the file, and reload the page in Firefox.

   - The header should now be in the right column (because it starts at line 2 and ends on line 3).

7. Switch back to your code editor.

8. In the **header** rule, change the start and end values as shown below:

```
header {
   grid-column-start: 1;
   grid-column-end: -1;
   background: #0db8e8;
}
```

NOTE: Grid lines are counted as negative numbers from right to left, so -1 is the rightmost line.

9. Save the file, and reload the page in Firefox.

- The header should now span the entire width of the grid (both columns).

10. Switch back to your code editor.

11. In the **header** rule, combine the two properties into a single shorthand grid-column property as shown below in bold:

```
header {
   grid-column: 1 / -1;
   background: #0db8e8;
}
```

NOTE: The syntax is **grid-column: start / end;**
You can also use values such as:

- grid-column: span 2 / 5; (span 2 ending at 5)
- grid-column: 2 / span 3; (start at 2 and span 3)

12. Save the file, and reload the page in Firefox.

- The header should look the same, this was just a more efficient way to code.

13. Switch back to your code editor.

14. In the **nav** rule, update the grid-row value as follows:

```
nav {
   grid-row: 2 / -1;
   background: #0978e2;
}
```

15. Save the file, and reload the page in Firefox.
    - With the -1 ending line number, you may have expected the nav to end at the bottom of the grid, but it does not. Why?
    - -1 is the bottom-most **explicit** row line. We've only explicitly set 3 rows. When there are not enough explicit rows for all the content, the other elements will end up below the -1 line. They are put into the **implicit** rows, which end up below the -1 line.
    - Next we'll move the footer into the empty grid space, so this will work in a moment.
16. Switch back to your code editor.
17. In the **footer** rule change grid-column so it fits into the empty space:

```
footer {
   grid-column: 2 / -1;
   background: #9700da;
}
```

18. Save the file, and reload the page in Firefox.
    - The footer should now fill the empty space that was below main, so we are back to our 3 explicit rows.
19. Switch back to your code editor.
20. In the **.container** rule add the following bold code:

```
.container {
   display: grid;
   grid-template-columns: 1fr 2fr;
   grid-template-rows: auto auto auto;
   gap: 2px;
   border: 8px solid orange;
}
```

NOTE: It's typically best to avoid fixed pixel heights. We were only using them temporarily, so now we're switching to auto (which will adapt to the height of the content in each element).

21. Save the file, and reload the page in Firefox.
    - The row heights now only take up the height of the content.

Naming Grid Lines

Instead of referring to lines by numbers, we can name them.

1. Switch back to your code editor.

2. We'll start by naming the column lines. In the **.container** rule add the following bold code:

```
.container {
   display: grid;
   grid-template-columns: [container-start] 1fr 2fr [container-end];
   CODE OMITTED TO SAVE SPACE
}
```

3. Each line can have multiple names. In the **.container** rule's **grid-template-columns** property, add the following bold code:

```
grid-template-columns: [container-start nav-start] 1fr [nav-end] 2fr
[container-end];
```

4. In the **.container** rule's **grid-template-columns** property, add the following bold code:

```
grid-template-columns: [container-start nav-start] 1fr [nav-end main-start]
2fr [main-end container-end];
```

5. Now we'll name the row lines. In the **.container** rule break the grid-template-rows value onto multiple lines as shown below in bold:

```
.container {
   display: grid;
   grid-template-columns: [container-start nav-start] 1fr [nav-end main-start]
2fr [main-end container-end];
   grid-template-rows:
      auto
      auto
      auto
      ;
   gap: 2px;
   CODE OMITTED TO SAVE SPACE
}
```

6. In the **.container** rule add names for the outer container lines as shown below in bold:

```
.container {
   CODE OMITTED TO SAVE SPACE
   grid-template-rows:
      [container-start]
      auto
      auto
      auto
      [container-end];
   gap: 2px;
   CODE OMITTED TO SAVE SPACE
}
```

7. In the **.container** rule add names for the main section's lines as shown below in bold:

```
grid-template-rows:
   [container-start]
   auto
   [main-start]
   auto
   [main-end]
   auto
   [container-end];
```

8. In the **.container** rule add names for the nav's lines as shown below in bold:

```
grid-template-rows:
   [container-start]
   auto
   [main-start nav-start]
   auto
   [main-end nav-end]
   auto
   [container-end];
```

9. Lastly, in the **.container** rule add names for the footer's lines as shown below in bold:

```
grid-template-rows:
   [container-start]
   auto
   [main-start nav-start]
   auto
   [main-end nav-end footer-start]
   auto
   [container-end footer-end];
```

10. Now that we've named the lines, we can change from using line numbers to line names when positioning our grid items. Edit the **header** rule as shown below in bold:

    ```
    header {
        grid-column: container-start / container-end;
        background: #0db8e8;
    }
    ```

11. Save the file, and reload the page in Firefox.

    - The page should look the same, but the code is clearer.

12. Switch back to your code editor.

13. Because we named the start and end lines, we can shorten our code even more. Edit the **header** rule as shown below in bold:

    ```
    header {
        grid-column: container;
        background: #0db8e8;
    }
    ```

    NOTE: Using a name with -start and -end is special. Grid understands the first part is a name!

14. Save the file, and reload the page in Firefox.

    - The page should look the same, but the code is even clearer.

15. Switch back to your code editor.

16. We can update the footer the same way. Edit the **footer** rule as shown below in bold:

    ```
    footer {
        grid-column: container;
        background: #9700da;
    }
    ```

17. In the **nav** rule, replace **grid-row** with **grid-column** as shown below in bold:

    ```
    nav {
        grid-column: nav;
        background: #0978e2;
    }
    ```

18. Save the file, and reload the page in Firefox.

    - Once again the page should still look the same, but the code is clearer. If the layout is messed up, you probably missing changing grid-**row** to grid-**column** in the **nav** rule. Go back and check your code.

Optional Bonus: Setting a *Max-Width* & Centering

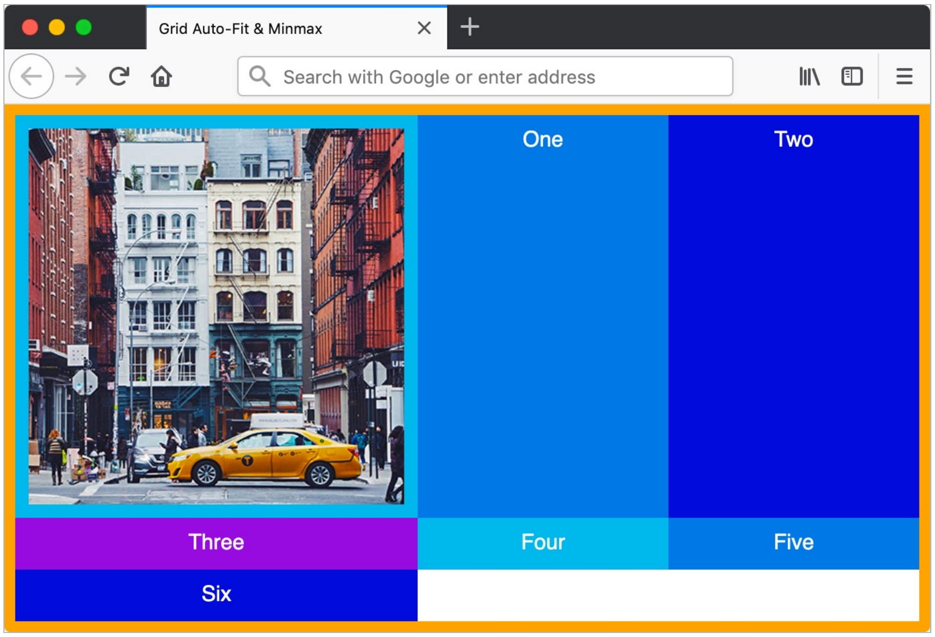
The grid container div can be sized and centered like any normal div, so let's try it.

1. Switch back to your code editor.
2. In the **.container** rule add the following bold code:

```
.container {
   max-width: 1000px;
   margin: auto;
   display: grid;

   CODE OMITTED TO SAVE SPACE
}
```

3. Save the file, and reload the page in Firefox.
   - Notice the width of the grid container can't get wider than 1000px, and is centered on wider screens.

Exercise Preview



Exercise Overview

In this exercise you'll learn about sizing grid items with minmax, auto-fit, auto-fill, max-content, and min-content.

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Grid Minmax** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Grid Minmax** folder.
   - As you've seen in previous exercises, there's a **.container** div that wraps a bunch of divs with numbers (and an image in the first div).
   - There are some commented out divs so we can quickly add more content later.
4. Preview **index.html** in Firefox.
   - As you've seen in previous exercises, the **.container** div has an orange border with multi-colored divs inside.
5. Leave **index.html** open in Firefox as you work, so you can reload the page to see the code changes you'll make.

Intro to Minmax

1. Switch back to your code editor.
2. Open **main.css** from the **css** folder (in the **Grid Minmax** folder).
3. In the **.container** rule add the following bold code:

```
.container {
   display: grid;
   grid-template-columns: repeat(5, 1fr);
   border: 8px solid orange;
}
```

4. Save the file, and reload the page in Firefox.
   - You should have 5 columns with flexible widths that fill the container.
   - Resize the window narrower to see how small the columns can get.
5. Switch back to your code editor.
6. In the **.container** rule, edit the column size as shown below in bold:

```
.container {
   display: grid;
   grid-template-columns: repeat(5, 100px);
   border: 8px solid orange;
}
```

7. Save the file, and reload the page in Firefox.
   - The 5 columns now have fixed widths so they no longer fill the container.
8. Switch back to your code editor.
9. In the **.container** rule, edit the column size as shown below in bold:

```
.container {
   display: grid;
   grid-template-columns: repeat(5, minmax(100px, 1fr));
   border: 8px solid orange;
}
```

10. Save the file, and reload the page in Firefox.
    - Resize the window and notice that at wider screen sizes the columns are the flexible 1fr size (their max size), but have a minimum size of 100px that prevents them from getting too small.
11. **Ctrl–click** (Mac) or **Right–click** (Windows) anywhere in the grid and choose **Inspect Element**.

12. In the DevTools, to the right of **`<div class="container">`** click the **grid** button to show a grid overlay.
13. In the **Grid** area of the Firefox DevTools, under **Grid Display Settings** check on **Display line numbers**.
14. Leave the DevTools open with the grid overlay on.
15. Switch back to your code editor.
16. Switch to **index.html**.
17. Uncomment **`<div>Five</div>`**
18. Save the file, and reload the page in Firefox.
    - Our grid has a repeat of 5, so we have 5 columns. We just added a sixth column (don't forget to count the image column), which wraps onto a new row.

Auto-Fit vs. Auto-Fill

1. Switch back to your code editor.
2. Switch to **main.css**.
3. In the **.container** rule, edit the column size as shown below in bold:

```
.container {
   display: grid;
   grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
   border: 8px solid orange;
}
```

4. Save the file, and reload the page in Firefox.
    - Resize the window from small to wide to see how this works.
    - The columns have a minimum size of 100px. Any columns that don't fit wrap onto rows, and the others scale up as needed to fill the width (their max width is 1fr).
    - Once all columns are in a single row, as you keep resizing the window wider, notice the right-most column number keeps increasing but you don't see any extra columns. Even though additional 100px wide columns would fit, the auto-fit option collapses those columns to allow our actual columns to fill the entire width of the container.
5. Switch back to **index.html** in your code editor.
6. Uncomment **`<div>Six</div>`**

7. Save the file, and reload the page in Firefox.
   - Resize the window and see how things behave.
   - It's cool how with this technique we don't need to know how many columns we'll have (the layout adjusts to fit them).
8. Switch back to **main.css** in your code editor.
9. In the **.container** rule, edit the column size as shown below in bold:

```
.container {
   display: grid;
   grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
   border: 8px solid orange;
}
```

10. Save the file, and reload the page in Firefox.
    - Resize the window from small to wide to see how this works.
    - On small screens this acts like auto-fit, but on wider screens once all columns are in a single row you'll see it acts differently. As you keep resizing the window wider this technique adds empty columns.

To review:

- **auto-fill**: Fills the available space with columns (which may be empty) by adding or removing columns as needed.
- **auto-fit**: Fits the current number of columns to the width of the grid.

More About Minmax

1. Switch back to your code editor.
2. In the **.container** rule, edit the column size as shown below in bold:

```
.container {
   display: grid;
   grid-template-columns: minmax(200px, 300px) 1fr 1fr;
   border: 8px solid orange;
}
```

3. Save the file, and reload the page in Firefox.
   - Resize the window and notice the first column will be the max 300px wide, unless it can't fit (then it will scale down until it hits the minimum 200px width).
4. Switch back to your code editor.

5. In the **.container** rule, edit the column size as shown below in bold:

```
.container {
   display: grid;
   grid-template-columns: minmax(200px, max-content) 1fr 1fr;
   border: 8px solid orange;
}
```

6. Save the file, and reload the page in Firefox.

   - Resize the window too see the maximum width for the first column is now based on the widest thing in that column (the photo).

7. Switch back to your code editor.

8. In the **.container** rule, edit the column size as shown below in bold:

```
.container {
   display: grid;
   grid-template-columns: minmax(min-content, 300px) 1fr 1fr;
   border: 8px solid orange;
}
```

   NOTE: While **minmax(min-content, max-content)** works, it's the same as **minmax(auto, auto)** or even more simply, just **auto** which is a lot less code!

9. Save the file, and reload the page in Firefox.

   - Resize the window to see the minimum size of the first column is the width of its narrowest content (in a way that doesn't cause overflow). In this case that's **Three**.

Fit-Content vs. Minmax

Similar to **minmax**, another sizing option is called **fit-content**. Below is a comparison and explanation of the slight differences between them. To help you compare them, imagine a scenario where the content does not fill the entire column width.

fit-content(800px)

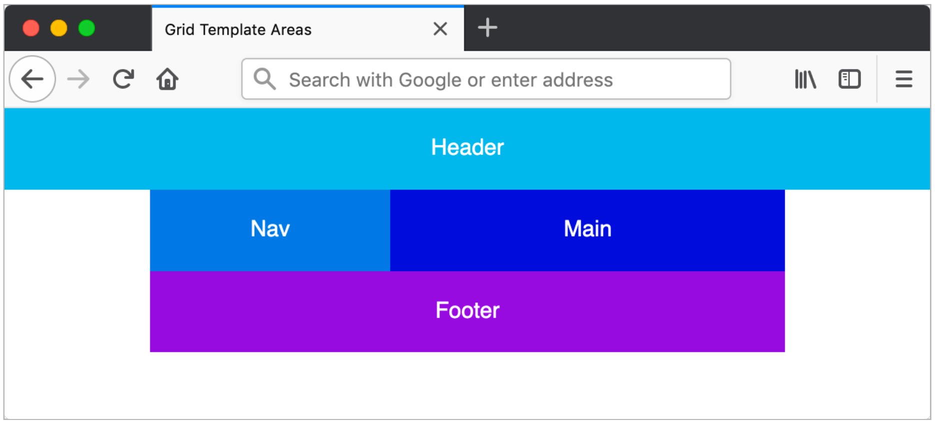
- If a column's content is less than 800px wide, the column will be the width of the content (even if there's enough space to make the column 800px wide).
- If a column's content is wider than 800px, the column will be limited to 800px wide.
- In other words: fit-content wants to shrink down to fit the content, but has a max width.

minmax(auto, 800px)

- If there is enough space to make the column 800px wide, the column will be 800px (even if the column's content is less than 800px wide).
- If there's not enough space to make the column 800px, it will be narrower (down to a minimum width of the content).
- In other words: minmax wants to hit the max (regardless of the content inside) but will be smaller if it can't be the max width.

Summary: Fit-content if more content centric and therefore wants to shrink down to the content when possible, whereas minmax wants to scale up to fill the space.

Exercise Preview



Exercise Overview

In this exercise you'll learn how position and size grid items using grid template areas, which can be an intuitive way to lay out a grid.

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Grid Template Areas** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Grid Template Areas** folder.
   - As you've seen in previous exercises, there's a **.container** div that wraps 4 sections.
4. Preview **index.html** in Firefox.
   - We didn't add the orange **.container** border that we had in previous exercises, because we won't be needing it and wanted a cleaner layout.
   - This stacking layout looks good on mobile devices, but we want to create a multi-column grid layout on larger screens.
5. Leave **index.html** open in Firefox as you work, so you can reload the page to see the code changes you'll make.

Setting Up Grid Template Areas

1. Switch back to your code editor.
2. Open **main.css** from the **css** folder (in the **Grid Template Areas** folder).

3. We only want our grid to appear on larger screens, so below all the other rules, create the following media query:

   ```
   @media (min-width: 600px) {

   }
   ```

4. In the **min-width: 600px** media query add the following bold code:

   ```
   @media (min-width: 600px) {
      .container {
         display: grid;
         grid-template-areas:
            "nav header"
            "nav main"
            "nav footer";
      }
   }
   ```

   This defines named areas into which we can place grid items. The cool thing is you can visualize the grid simply by looking at these labels!

   - We want a nav column on the left that's 3 rows tall.
   - We want a right column that contains a header, main, and footer (each on a different row).

5. Now we need to place the grid items into the named areas we defined. Add the following bold code:

   ```
   header {
      grid-area: header;
      background: #0db8e8;
   }
   nav {
      grid-area: nav;
      background: #0978e2;
   }
   main {
      grid-area: main;
      background: #0000d5;
   }
   footer {
      grid-area: footer;
      background: #9700da;
   }
   ```

6. Save the file, and reload the page in Firefox.
    - Make the window wider than 600px.
    - You should see a 2 column layout, with nav in the left column and the header, main, and footer in the right column.
7. Switch back to your code editor.
8. In the **.container** rule, change the areas as shown below in bold:

```
.container {
   display: grid;
   grid-template-areas:
      "header header"
      "nav main"
      "footer footer";
}
```

NOTE: The areas you create must be rectangles.

9. Save the file, and reload the page in Firefox.
    - Wow, that was so much easier to totally change the layout!
    - Now the header should span both columns on the top and the footer should span across both columns on the bottom.

Creating Empty Grid Areas

Sometimes you don't want anything in a section of the grid layout. We can use a period (.) to create an empty space.

1. Switch back to your code editor.
2. In the **.container** rule, change the areas as shown below in bold:

```
.container {
   display: grid;
   grid-template-areas:
      "header header header header header"
      ". nav main main ."
      ". footer footer footer .";
}
```

NOTE: If you want to line things up better, you can add extra spaces. You can also use multiple periods instead of one (as long as the periods for a single grid area don't contain spaces). So you could use . or …..

3. Save the file, and reload the page in Firefox.
   - If the layout is broken, double-check your code to make sure that you have the same number of columns in all 3 rows (we have 5 columns).
   - The header should extend across the entire width of the page, but the other 2 rows should have an empty column on both the left and right sides.

Viewing Grid Template Area Names In Firefox's DevTools

1. **Ctrl–click** (Mac) or **Right–click** (Windows) anywhere in the grid and choose **Inspect Element**.
2. In the DevTools, to the right of **`<div class="container">`** click the **grid** button to show the grid overlay.
3. In the **Grid** area of the Firefox DevTools, under **Grid Display Settings** check on **Display area names**.
4. Make sure the window is wide enough to see the grid layout, and you'll see the grid template area names we wrote in our CSS!
5. Close the DevTools.

Sizing the Columns

As we've done before, we use **grid-template-columns** to set the width of columns.

1. Switch back to your code editor.
2. In the **.container** rule, add the following bold code:

```
.container {
   display: grid;
   grid-template-columns: 75px 200px 1fr 1fr 75px;
   grid-template-areas:
   CODE OMITTED TO SAVE SPACE
}
```

NOTE: We created 5 columns with **grid-template-areas** so we're defining 5 column widths with **grid-template-columns**.

3. Save the file, and reload the page in Firefox.
   - The outer empty white columns should be smaller, at 75px wide.
   - The nav column should be a fixed 200px wide.
   - The rest should be flexible to fill the remaining space.

Using Automatically Created Named Lines

In an earlier exercise we saw how to refer to grid lines by numbers, or by names we define. A cool thing about defining names with **grid-template-areas**, is that the grid lines are automatically named for you! You simply add **-start** and **-end** to your template area names.

Let's add an **overlay** on top of the **nav, main, and footer** areas (but not over the header). We can do this using the named lines.

1. Switch back to your code editor.
2. We'll want the overlay to work on all screens, but currently the grid layout is only for larger screens. Let's add a simple grid layout for mobile devices. Below the **body** rule (outside of any media query), the following new rule:

```
.container {
   display: grid;
   grid-template-areas:
      "header"
      "nav"
      "main"
      "footer";
}
```

3. In the **min-width: 600px** media query's **.container** rule, remove the now redundant **display: grid;** so you end up with:

```
@media (min-width: 600px) {
   .container {
      grid-template-columns: 75px 200px 1fr 1fr 75px;
      grid-template-areas:
         "header header header header header"
         ". nav main main ."
         ". footer footer footer .";
   }
}
```

4. Save the file, and reload the page in Firefox.
   - The layout should be stacked in a single column on smaller screens, and the same multi-column layout on wider screens.
5. Switch back to your code editor.
6. Switch to **index.html**.

7. Below the footer add an overlay div as shown below in bold:

```
   <footer>Footer</footer>
   <div class="overlay">Overlay</div>
</div>
```

8. Save the file.

9. Switch to **main.css**.

10. Below the **footer** rule (outside of the media query), add the following new rule:

```
.overlay {
   background: rgba(0,0,0, .6);
   grid-column: nav-start / main-end;
   grid-row: nav-start / footer-end;
}
```

NOTE: Notice how we're using the nav, main, and footer names we defined in the grid-template-areas, adding **-start** or **-end** as needed.

11. Save the file, and reload the page in Firefox.

- You should see a darkened overlay on top of the **nav, main, and footer** areas.

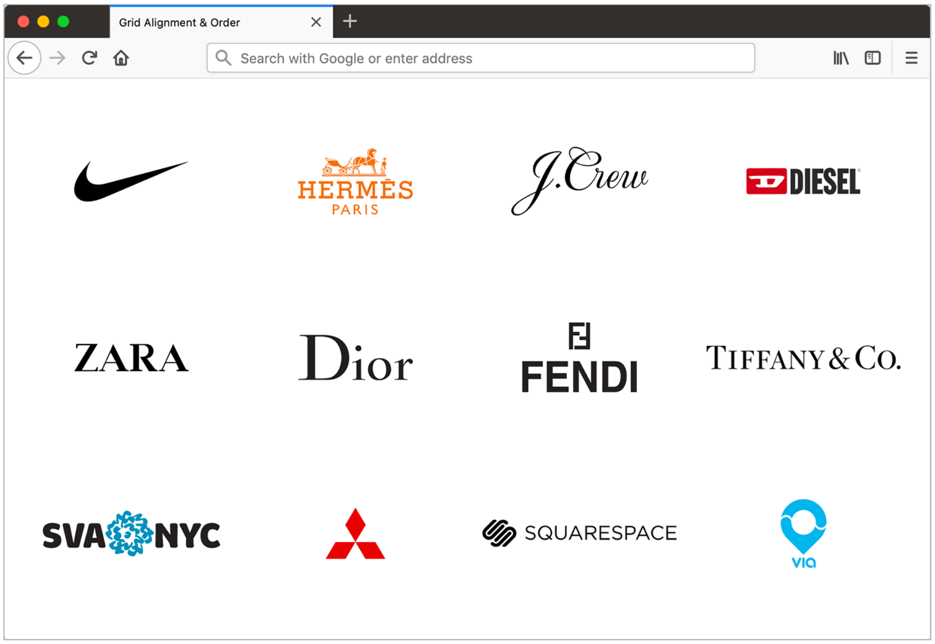
- Isn't is cool that multiple elements can occupy a grid area?

TIP: The overlay is on top because it comes after the other elements in the HTML. To change the stacking order you could use `z-index: -1;` to send it behind the other elements. If you did that in this page you wouldn't see the overlay because the other elements have background colors. If they didn't have backgrounds you'd see through to it.

Grid: Alignment, Centering, & Reordering Content

4B

Exercise Preview



Exercise Overview

In this exercise you'll learn about aligning grid items and ordering content (which are both similar to flexbox).

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Grid Alignment and Order** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Grid Alignment and Order** folder.
   - Notice we have a **.gallery** div that contains images.
   - Notice the first image is wrapped in **.logo-wrapper** div. In the css file we've styled this div with a background color so you'll be able to see some things about sizing easier than if all these items were only images.

4. Preview **index.html** in Firefox.

   - The **.gallery** div has an orange border (so you can more easily see its size).
   - The J.Crew logo is in a div with a background color, so it's currently filling up the entire width of the gallery.
   - The remaining logos are displayed inline like images normally are.

5. Leave **index.html** open in Firefox as you work, so you can reload the page to see the code changes you'll make.

Starting the Grid

1. Switch back to your code editor.
2. Open **main.css** from the **css** folder (in the **Grid Alignment and Order** folder).
3. In the **.gallery** rule add the following bold code:

```
gallery {
   border: 8px solid orange;
   display: grid;
   gap: 30px;
   grid-template-columns: repeat(4, 1fr);
}
```

4. Save the file, and reload the page in Firefox.

   - The logos should now be in a 4 by 4 grid.
   - The logos would look better aligned to the center (horizontally and vertically).

5. **Ctrl–click** (Mac) or **Right–click** (Windows) anywhere in the grid and choose **Inspect Element**.
6. In the DevTools, to the right of **`<div class="gallery">`** click the **grid** button to show the grid overlay.
7. Notice the colored div that contains the J.Crew logo fills its grid area.
8. Leave the DevTools open with the grid overlay on.

Aligning Grid Items

1. Switch back to your code editor.

2. In the **.gallery** rule add the following bold code:

```
gallery {
   CODE OMITTED TO SAVE SPACE
   grid-template-columns: repeat(4, 1fr);
   justify-items: center;
}
```

3. Save the file, and reload the page in Firefox.
   - Notice the logos are now horizontally centered within their grid areas.
   - Notice the colored div that contains the J.Crew logo has collapsed to be the size of its content (it previously had stretched to fill the grid area).
4. Switch back to your code editor.
5. In the **.gallery** rule add the following bold code:

```
gallery {
   CODE OMITTED TO SAVE SPACE
   justify-items: center;
   align-items: center;
}
```

6. Save the file, and reload the page in Firefox.
   - Notice the logos are also vertically centered within their grid areas.

Aligning Within the Grid Container

1. Switch back to your code editor.
2. In the **.gallery** rule change **1fr** to **175px** as shown below in bold:

```
gallery {
   border: 8px solid orange;
   display: grid;
   gap: 30px;
   grid-template-columns: repeat(4, 175px);
   justify-items: center;
   align-items: center;
}
```

3. Save the file, and reload the page in Firefox.
   - Resize the window so it's wide enough that when the logos do not fill the gallery div, you can see there's now empty white space to the right of the logos (inside the orange border).

4. Switch back to your code editor.

5. In the **.gallery** rule add the following bold code:

```
gallery {
    CODE OMITTED TO SAVE SPACE
    justify-items: center;
    align-items: center;
    justify-content: center;
}
```

6. Save the file, and reload the page in Firefox.

   - Notice the entire section of grid columns is now centered (so the empty space is evenly split between the left and right sides).

7. In DevTools find the **.gallery** rule's **justify-content** property.

   - Try out the following values: **start**, **end**, **center**, **space-around**, **space-between**, and **space-evenly** (**stretch** is the default).

8. Switch back to your code editor.

9. In the **.gallery** rule add the following bold code:

```
gallery {
    CODE OMITTED TO SAVE SPACE
    justify-content: center;
    height: 700px;
}
```

10. Save the file, and reload the page in Firefox.

    - The grid is now taller.
    - We needed to set a height to see how align-content works.

11. Switch back to your code editor.

12. In the **.gallery** rule add the following bold code:

```
gallery {
    CODE OMITTED TO SAVE SPACE
    justify-items: center;
    align-items: center;
    justify-content: center;
    align-content: center;
    height: 700px;
}
```

13. Save the file, and reload the page in Firefox.
    - Notice the entire section of grid rows is now vertically centered (so the empty space is evenly split between the top and bottom).
14. In DevTools find the **.gallery** rule's **align-content** property.
    - Try out the following values: **start**, **end**, **center**, **space-around**, **space-between**, and **space-evenly** (**stretch** is the default).
    - These work the same as justify-content, but vertically instead of horizontally.
15. Reload the page to get rid of any changes you were experimenting with.

Aligning Individual Grid Items

In this layout we don't need to align individual elements, but let's take a look at how that works for when you will need it.

1. Switch back to your code editor.
2. Below the **.gallery** rule add the following new rule:

```
.gallery .zara {
   justify-self: end;
   align-self: start;
}
```

3. Save the file, and reload the page in Firefox.
    - Notice the **Zara** logo is now at the top right of its grid area.
    - **Ctrl–click** (Mac) or **Right–click** (Windows) on the **Zara** logo and choose **Inspect Element**.
    - In DevTools find the **.gallery .zara** rule and experiment with changing **justify-self** and **align-self** to any of the following values: **start**, **end**, and **center** (**stretch** is the default, which is appropriate for divs, but not images).
4. Switch back to your code editor.
5. Delete the **.gallery .zara** rule (we were only seeing how this works, and didn't actually want to change it).

Ordering Grid Items

Ordering works the same in grid and flexbox.

1. Below the **.gallery** rule add the following new rule:

```
.gallery .nike {
   order: -1;
}
```

2. Save the file, and reload the page in Firefox.

   - The Nike logo should now be first (because the default order is 0).

3. Switch back to your code editor.

4. In the **.gallery .nike** rule change the order as shown below:

```
.gallery .nike {
   order: 1;
}
```

5. Save the file, and reload the page in Firefox.

   - The Nike logo should now be last.

6. Switch back to your code editor.

7. Switch to **index.html**.

8. Delete the **.logo-wrapper** div that wraps the first image (be sure to delete the div's start and end tags).

9. Save the file.

10. Switch back to **main.css**.

11. Above the **.gallery .nike** rule add the following new rule:

```
.gallery img {
   order: 3;
}
```

12. Below the **.gallery .nike** rule add the following new rule:

```
.gallery .hermes {
   order: 2;
}
```

13. Save the file, and reload the page in Firefox.

   - The Nike logo should be first, followed by Hermes, J.Crew, and then the rest of the logos.

Optional Bonus: Cleaning Up the Appearance

1. Switch back to your code editor.

2. Remove the **height** from the **.gallery** rule.

3. Remove the **border** from the **.gallery** rule.

4. Save the file, and reload the page in Firefox.

   - In the DevTools, to the right of **`<div class="gallery">`** click the **grid** button to hide the grid overlay.

   - That's a clean looking grid of logos!

Aligning in CSS Grid

Here's a quick comparison of grid's alignment properties:

Justify vs. Align

- **justify-** aligns along the **row** axis (**x** or **inline** axis)
- **align-** aligns along the **column** axis (**y** or **block** axis)
- Unlike Flexbox, the axes don't switch.

Items vs. Content

- **-items** aligns what is inside each grid item/cell.
- **-content** defines the alignment/spacing of grid items within the grid container (assuming the grid is smaller than the size of the grid container).

How to Remember Items vs. Content

- **-items**: Items is plural. There are many items in the grid, so justify-items and align-items apply individually to every one of the items.
- **-content**: Content is singular. This refers to the grid container's content as a whole (the grid's content).

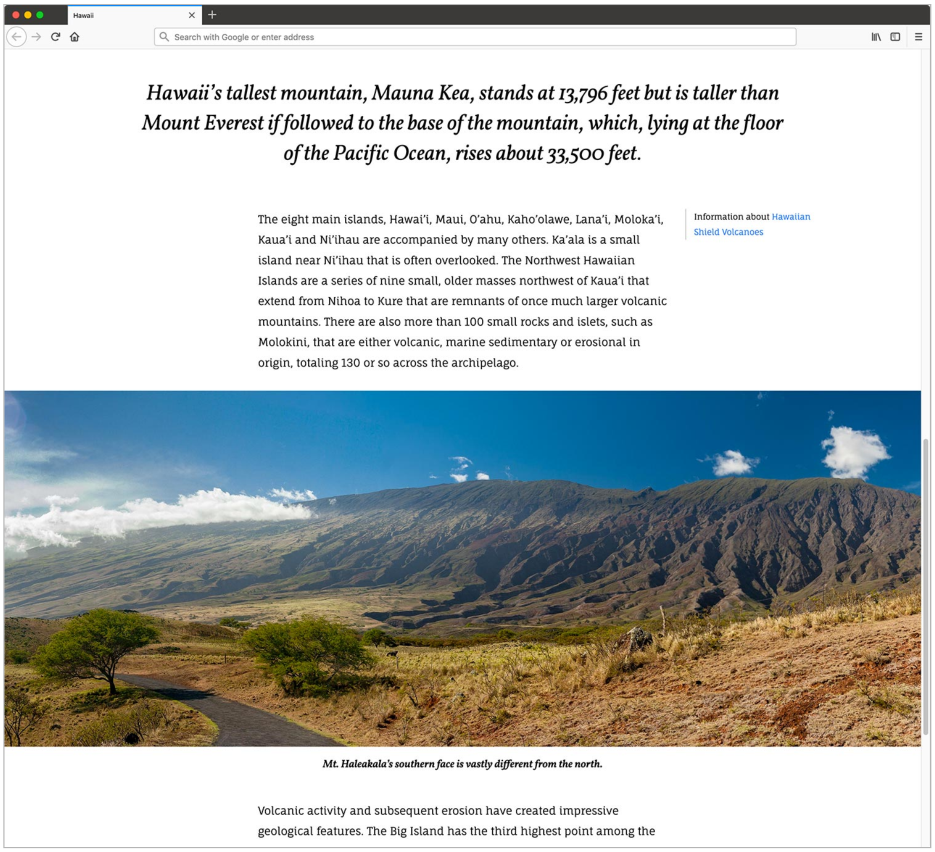
For a detailed look at grid's various properties, CSS Trick's **Complete Guide to Grid** **tinyurl.com/csst-grid** is a great reference with visuals. If you ever forget something, or are confused by the terminology, this should help.

Place Items Shorthand

The shorthand for **justify-items** and **align-items** is **place-items** but at the time of this writing does not have good browser support. You can check browser compatibility and learn more about this property on the MDN Web Docs at **tinyurl.com/grid-pi**

Grid: Laying out an Article

Exercise Preview



Hawaii's tallest mountain, Mauna Kea, stands at 13,796 feet but is taller than Mount Everest if followed to the base of the mountain, which, lying at the floor of the Pacific Ocean, rises about 33,500 feet.

The eight main islands, Hawai'i, Maui, O'ahu, Kaho'olawe, Lana'i, Moloka'i, Kaua'i and Ni'ihau are accompanied by many others. Ka'ala is a small island near Ni'ihau that is often overlooked. The Northwest Hawaiian Islands are a series of nine small, older masses northwest of Kaua'i that extend from Nihoa to Kure that are remnants of once much larger volcanic mountains. There are also more than 100 small rocks and islets, such as Molokini, that are either volcanic, marine sedimentary or erosional in origin, totaling 130 or so across the archipelago.

Information about Hawaiian Shield Volcanoes

Mt. Haleakala's southern face is vastly different from the north.

Volcanic activity and subsequent erosion have created impressive geological features. The Big Island has the third highest point among the

Exercise Overview

In this exercise you'll use CSS grid to lay out an article that looks good on all screen sizes. With full page-width images and notes in the margins, this kind of layout would be harder to create if we weren't using grid.

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Grid Article** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Grid Article** folder.
   - Notice this page is an article tag that contains a heading, paragraphs, figures, blockquotes, and asides.

4. Preview **index.html** in Firefox.
   - Scroll down the page to look at the content.
   - For our layout we'll want to make the text column narrower, but to keep the images and pull quotes to extend beyond the width of the text column.
5. Leave **index.html** open in Firefox as you work, so you can reload the page to see the code changes you'll make.

Starting the Grid

We'll start by making a mobile phone appropriate grid (with one column, and an appropriate sized gap).

1. Switch back to your code editor.
2. Open **main.css** from the **css** folder (in the **Grid Article** folder).
3. Below the **a:hover** rule add the following bold code:

```
article {
   display: grid;
   gap: 30px;
   grid-template-columns: 1fr;
}
```

4. Save the file, and reload the page in Firefox.
   - Now that we're using grid gap to space things out, we'll need to remove the paragraph margins so we don't have too much space.
5. Switch back to your code editor.
6. Above the **a** rule add the following bold code:

```
p {
   margin: 0;
}
```

7. Save the file, and reload the page in Firefox.
   - Now the paragraphs have a more appropriate amount of space between them.
   - Now let's set up the desktop grid with multiple columns.
8. Switch back to your code editor.

9. In the **min-width: 700px** media query at the bottom of the file, below the **h1** rule add the following new rule:

```
article {
   grid-template-columns: 1fr 4fr 1fr;
}
```

10. Save the file, and reload the page in Firefox.

    - Resize the window to make sure you're seeing the 3 column layout.
    - That gave us 3 columns, but things are not where we want them! Don't worry, that will be quick to fix.

11. Switch back to your code editor.

12. In the **min-width: 700px** media query, below the **article** rule add the following new rule:

```
article > * {
   grid-column: 2 / -2;
}
```

13. Save the file, and reload the page in Firefox.

    - That looks much better already!
    - Resize the window narrower, and notice the page needs some space on the left and right so the content does not touch the edge.

14. Switch back to your code editor.

15. Above the **min-width: 700px** media query, add the following new media query:

```
@media (max-width: 699px) {
   article > * {
      margin: 0 20px;
   }
}
```

16. Save the file, and reload the page in Firefox.

    - On small screens, the spacing now looks better.
    - We want the images to be the full window width on all screens. It's getting the 20px on small screens, but figures have default margin so the large screens also have margins. Let's remove the margin on all screens, and make it full width on large screens.

Placing the Content in the Grid

1. Switch back to your code editor.

2. Below the **article** rule (the one NOT in a media query) add the following bold code:

```
article > figure {
   margin: 0;
   grid-column: 1 / -1;
}
```

3. Save the file, and reload the page in Firefox.

   - Resize the window from small to large. The full-width images look nice! Layouts like this were not this easy to create without CSS grid.

   - Let's also make the pull quote the full browser width as well.

4. Switch back to your code editor.

5. In the **article > blockquote** rule, add the following bold code:

```
article > blockquote {
    grid-column: 1 / -1;
    font-family: Vollkorn, serif;
   CODE OMITTED TO SAVE SPACE
}
```

6. Save the file, and reload the page in Firefox.

   - Notice the quote is the full width.

   - That's nice on medium width screens, but on wide screens the text can too wide. Also, the main text column can get too wide on large screens as well. Let's see how to limit the width of both.

Adding Max Widths for the Text

1. Switch back to your code editor.

2. Below the **min-width: 700px** media query, add the following new media query:

```
@media (min-width: 950px) {
   article {
      grid-template-columns: 1fr 58ch 1fr;
   }
}
```

   NOTE: 58ch means 58 characters wide. We're using this as a max width so we don't have too many characters per line.

3. Save the file, and reload the page in Firefox.

   - Resize the window from narrow to wide, noticing how the text column now has a max width so it can't get too wide. That's much better.

   - Next let's limit the width of the pull quote.

4. Switch back to your code editor.

5. Let's prevent the pull quote from getting too wide. In the **article > blockquote** rule, add the following bold code:

```
article > blockquote {
    grid-column: 1 / -1;
    max-width: 50ch;
    margin: auto;
    padding: 20px;
    font-family: Vollkorn, serif;
```

   NOTE: The auto margins will center the element when the element's width does not fill the screen.

6. Save the file, and reload the page in Firefox.

   - Notice the width of the pull quote is limited so it can't get too wide.

   - Scroll down to above the second photo. Notice there's a paragraph that is a bit smaller than the other text and has a thin gray line on the left. This is in an aside tag, and there's a second one farther down the page. We'd like to move these in the left or right columns (on wide screens).

   - We've already added a class or **left** or **right** to each aside, so we only need to add the CSS to style them.

Putting Notes in the *Margins* on Large Screens

1. Switch back to your code editor.

2. In the **min-width: 700px** media query, below the **article >** * rule add the following new rule:

```
article > aside.right {
   grid-column: span 1 / -1;
}
```

   NOTE: This says to span 1 column that ends at -1 (the right-most column).

3. Save the file, and reload the page in Firefox.

   - Above the second photo, you should now see **Information about Hawaiian Shield Volcanoes** in the right column!

   - The gray line on its left takes up the entire height of the paragraph it's next too. Let's align this to the top instead of having it stretch (which is the default).

4. Switch back to your code editor.

5. In the **min-width: 700px** media query's **article > aside.right** rule add the following bold code:

```
article > aside.right {
   grid-column: span 1 / -1;
   align-self: start;
}
```

6. Save the file, and reload the page in Firefox.

   - The line no longer stretches to fill the entire grid area, and has collapsed to the height of **Information about Hawaiian Shield Volcanoes** (which is what we want).

7. Switch back to your code editor.

8. In the **min-width: 700px** media query, below the **article > aside.right** rule add the following new rule:

```
article > aside.left {
   grid-column: 1 / span 1;
   align-self: start;
}
```

9. Save the file, and reload the page in Firefox.

   - Scroll to the bottom of the page and notice that **Learn more about Haleakalā and Hawaii Volcanoes National Parks** is in the correct position in the left column.

   - We'll need to move the line to the right, and we'd like to right align this text as well.

10. Switch back to your code editor.

11. In the **min-width: 700px** media query's **article > aside.left** rule add the following bold code:

```
article > aside.left {
   grid-column: 1 / span 1;
   align-self: start;
   text-align: right;
   border-left-width: 0;
   border-right-width: 1px;
}
```

12. Save the file, and reload the page in Firefox.

    - The line is now on the right and the text is now right aligned. Much better.
    - On wide screens the asides can get too wide. Let's limit their width as well.

13. In the **min-width: 700px** media query, below the **article >** * rule add the following new rule:

    ```
    article > aside {
       max-width: 30ch;
    }
    ```

14. Save the file, and reload the page in Firefox.

    - The limited width looks good on the aside that's in the right column, but the aside that's in the left column needs to be moved over to the right side of the column.

15. Switch back to your code editor.

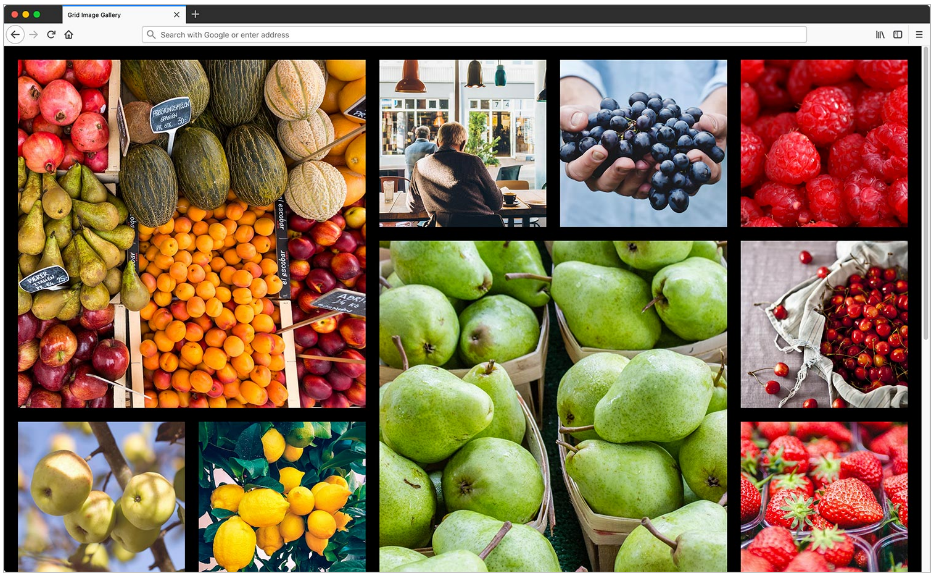
16. In the **min-width: 700px** media query's **article > aside.left** rule add the following bold code:

    ```
    article > aside.left {
       grid-column: 1 / span 1;
       align-self: start;
       justify-self: end;
       text-align: right;
       border-left-width: 0;
       border-right-width: 1px;
    }
    ```

17. Save the file, and reload the page in Firefox.

    - Now the aside in the left column looks good too.
    - Scroll through the page to admire the final layout.
    - Resize the window to see it looks good across all screen sizes.

Exercise Preview



Exercise Overview

In this exercise you'll create an image grid that looks good on all screen sizes.

Layouts with equally sized items are much easier to create than ones with different sized items. Getting unequally sized elements to fit nicely together (like Pinterest) has usually required JavaScript. CSS grid is the first CSS technology which can give us this kind of layout.

Many years ago David DeSandro created a popular JavaScript library which he named **Masonry**, and that is what people now call this kind of layout.

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Grid Image Gallery** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Grid Image Gallery** folder.
   - Notice we have a **.gallery** div that contains images.
4. Preview **index.html** in Firefox.
   - Scroll through to see the images we have on the page.
   - Let's put them into grid.

5. Leave **index.html** open in Firefox as you work, so you can reload the page to see the code changes you'll make.

Setting Up the Grid for Small Screens

1. Switch back to your code editor.
2. Open **main.css** from the **css** folder (in the **Grid Image Gallery** folder).
3. Below the **body** rule add the following new rule:

```
.gallery {
   display: grid;
   gap: 15px;
   grid-template-columns: 1fr 1fr;
}
```

4. Save the file, and reload the page in Firefox.
   - You should have a 2 column image grid.
   - Resize the window to see it looks nice on mobile, but the images get a bit too big on wider screens (where we have space to see more columns).
   - Notice we have gaps between columns, but we don't have the same space round the outside of the grid. Let's add that.
5. Switch back to your code editor.
6. In the **.gallery** rule, add the following bold code:

```
.gallery {
   display: grid;
   gap: 15px;
   padding: 15px;
   grid-template-columns: 1fr 1fr;
}
```

7. Save the file, and reload the page in Firefox.
   - Notice there's now space on the outside of the grid.

Adjusting the Grid for Larger Screens

1. Switch back to your code editor.

2. Below the **.gallery** rule add the following new media query:

```
@media (min-width: 600px) {
   .gallery {
      grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
   }
}
```

3. Save the file, and reload the page in Firefox.

   - Resize the window from narrow to wide.
   - On narrow screens 2 columns fit. On wider screens you get more and more columns, depending on how many fit. Nice!
   - On larger screens would could have more space between the photos, so let's adjust that.

4. Switch back to your code editor.

5. In the **min-width: 600px** media query's **.gallery** rule, add the following bold code:

```
@media (min-width: 600px) {
   .gallery {
      gap: 25px;
      padding: 25px;
      grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
   }
}
```

6. Save the file, and reload the page in Firefox.

   - Resize the window from narrow to wide and notice the margin and gap gets a little bigger when the window is 600px or wider.

7. Switch back to your code editor.

Enlarging Some Photos

1. Switch back to your code editor.
2. Switch to **index.html**.
3. Notice that some of the photos have a **big** class.
4. Switch back **to main.css**.

5. Below the **.gallery** rule add the following new rule:

```
.big {
   grid-column: span 2;
   grid-row: span 2;
}
```

6. Save the file, and reload the page in Firefox.

   - Some of the photos are now bigger, but that has created a few gaps at certain screen sizes (resize your browser and scroll through the page to find them).

   - Luckily grid has a way to help fill in these gaps.

7. Switch back to your code editor.

8. In the **min-width: 600px** media query's **.gallery** rule, add the following bold code:

```
@media (min-width: 600px) {
   .gallery {
      gap: 25px;
      padding: 25px;
      grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
      grid-auto-flow: dense;
   }
}
```

9. Save the file, and reload the page in Firefox.

   - Notice the gaps should now be filled in.

   - Keep in mind the limitations of masonry layout. The bottom may not align perfectly (there can still be gaps at the bottom) because there may not be the right number of elements to perfectly fill the grid.

Limiting the Width

We don't want the photo grid to get too wide so let's add a max-width.

1. In the **min-width: 600px** media query's **.gallery** rule, add the following bold code:

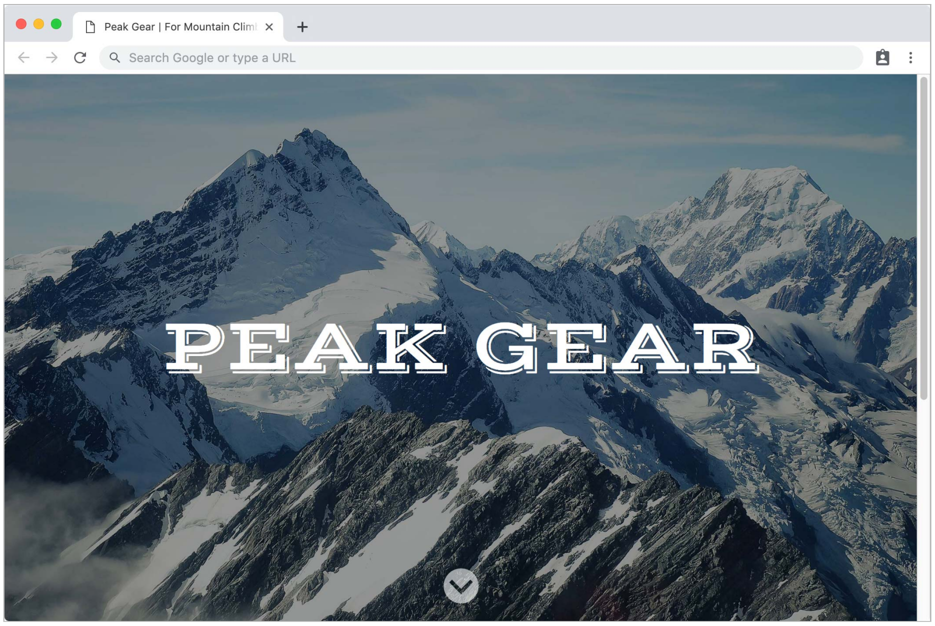
```
@media (min-width: 600px) {
   .gallery {

      CODE OMITTED TO SAVE SPACE

      grid-auto-flow: dense;
      max-width: 1900px;
      margin: auto;
   }
}
```

2. Save the file, and reload the page in Firefox.

   - If you have a wide enough screen, resize the window to see the photo grid does not get wider than 1900px.
   - If you do not have a wide enough screen, zoom out by holding **Cmd** (Mac) or **Ctrl** (Window) and hitting the **Minus (-)** key a few times until you can see that the width is limited.
   - Zoom the page back to normal size by holding **Cmd** (Mac) or **Ctrl** (Window) and hitting the **Zero (0)** key.

Grid: Vertical Centering on a Full Screen Background

Exercise Preview



Exercise Overview

In this exercise you'll use grid to center content vertically (and horizontally) in the screen. This is the accomplishing same thing we did in a earlier exercise with flexbox, but now you'll see how to do it with grid.

Getting Started

1. In your code editor, close any files you may have open.
2. For this exercise we'll be working with the **Grid Vertical Centering** folder located in **Desktop > Class Files > yourname-Flexbox Grid Class**. You may want to open that folder in your code editor if it allows you to (like Visual Studio Code does).
3. Open **index.html** from the **Grid Vertical Centering** folder.
4. Preview **index.html** in a browser.
   - The photo background covers the full width and height of the window, but we want to center the content inside it.
5. Leave **index.html** open in the browser as you work, so you can reload the page to see the code changes you'll make.
6. Switch back to your code editor.
7. Find the **header** tag.

8. Inside that, notice the heading and anchor tags. These are the elements we will be targeting.

Vertically Centering the Header Content

1. Open **main.css** from the **css** folder (from the **Grid Vertical Centering** folder).
2. In the **header** rule, add the following bold code:

```
header {

   CODE OMITTED TO SAVE SPACE

   min-height: 100vh;
   display: grid;
   grid-template-rows: auto 60px;
   justify-items: center;
   align-items: center;
}
```

NOTE: We need 2 rows: one for the heading (the auto height) and another for the scroll down arrow (the 60px height). We made the arrow row 60px high because the arrow is 40px tall and we wanted 20px of space below it.

3. Save the file, and reload the page in your browser.
   - That's looking good already, but a few tweaks are needed to perfect it.
   - Currently the arrow is vertically centered in the 60px row, so it only has 10px below it. We want to align it to the top of the row so all 20px is below it.
   - Make the window narrow, so you can see the heading text is not centered on mobile screens (because the default text alignment is left). Let's center the text.
4. Switch back to your code editor.
5. In the **h1** rule, add the following bold code:

```
h1 {

   CODE OMITTED TO SAVE SPACE

   margin: 0;
   text-align: center;
}
```

6. Save the file, and reload the page in your browser.
   - The heading is now centered on mobile screens.
7. Switch back to your code editor.

8. In the **.scroll-down** rule, add the following bold code:

```
.scroll-down {

   CODE OMITTED TO SAVE SPACE

   opacity: .6;
   align-self: start;
}
```

9. Save the file, and reload the page in your browser.

   - Now the arrow is at the top of the 60px row, so the 40px arrow now has 20px of space below it. That's better.

   - The heading is now vertically centered between the top of the page and the scroll down arrow. That means the heading is not fully centered in the screen (it's a little too high). It looks OK, but let's see how to vertically center it perfectly in case that's what you prefer.

10. Return to your code editor.

11. In the **header** rule add the following bold code to adjust the spacing:

```
header {

   CODE OMITTED TO SAVE SPACE

   align-items: center;
   padding-top: 60px;
}
```

12. Save the file, and reload the page in your browser.

    - Perfect! Resize the window from mobile through desktop size to check out your full screen background with centered content!

Web Development Level 1
Building
Flexbox, Grid, & Bootstrap
Sketch
Design the UI & UX of Websites & Apps
Learn by doing step by step exercises.
Includes downloadable class files that work on Mac.

Useful Websites, Online Tools, & More

Because there are so many wonderful online tools, blog writeups, and more, we decided to make a webpage with all the links instead of printing them in this book. Not only does this save you some typing, but it will also allow us to keep it more up-to-date. The list includes sites we talked about earlier in this book as well as some additional ones.

Check them out at: **nobledesktop.com/flexbox-grid-links**