

Advanced CSS

Advanced CSS

In this lesson you will learn:

- How to utilize the browser's developer tools to aide your development process
- The importance of CSS pseudo-classes and pseudo-elements
- Supercharged CSS using the Sass preprocessor

Introduction

Last week, you learned the basics of how to style your web pages to begin to make them look good.

Maybe you feel like an absolute rock star with CSS, or maybe you've hit your head against the wall more than a few times because the styling doesn't turn out quite like you expected... and if this is the case, you're not alone!



>

CSS can be tricky at first, to say the least. Don't worry, though! After a while, with continued practice, you'll arrive at a level of comfort and familiarity with CSS and what may be frustrating you now will become second-nature.

Over the course of this week's lessons, you'll begin to add a little more to your CSS arsenal, while also learning about various tools that will help make your life easier. Let's get started!

DevTools: A Frontend Developer's Best Friend

What are Browser DevTools?

Much like the name suggests, browser DevTools are a set of tools for web developers provided by the browser to make your life easier while building web pages.

There is a lot you can do with DevTools, but for now we'll primarily be focusing on how we can use them to inspect our HTML and CSS, as well as debug any issues that we may encounter.

>

How to Open DevTools

>

Most of the methods to open DevTools will be similar for Firefox as well; we'll briefly touch on one noteworthy difference in a moment.

Chrome DevTools provides a variety of keyboard shortcuts to help you quickly access specific panels. Some examples include:

Inspect an element's attributes or CSS in the Elements panel

- Pressing Command + Option + C on a Mac (or Ctrl + Shift + C on Windows or Linux)

Opening console directly

- Pressing Command + Option + J on a Mac (or Ctrl + Shift + J on Windows or Linux)

Close and reopen the same panel

- Pressing Command + Option + I on a Mac (or Ctrl + Shift + I on Windows or Linux), this can be useful for refreshing logs or network activity.

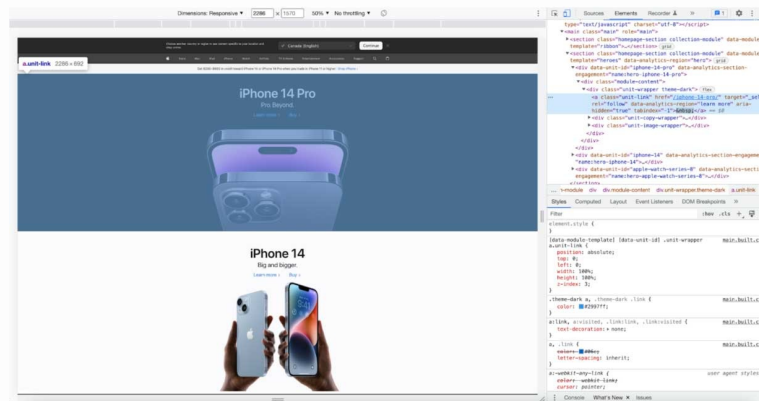
Right-clicking on elements

Another way to open Chrome DevTools is by right-clicking on an element in the browser and selecting "Inspect."



alt text

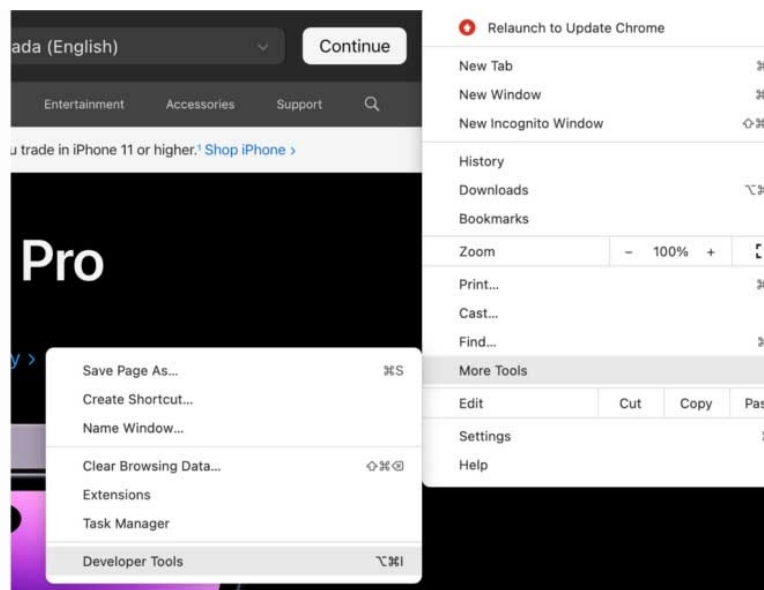
This will open the Elements panel with that element selected, allowing you to view and modify its attributes and styles.



alt text

UI tool on the top right

You can also access Chrome DevTools through the UI tool on the top right of the Chrome window. Click the three dots, then select “More Tools” and “Developer Tools.” This will open DevTools in a new window.



alt text

F12 key

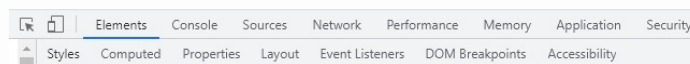
The F12 key is another classic way to open Chrome DevTools. Pressing F12 on your keyboard will open DevTools in a new window. Note that some laptops may require you to press the “FN” key as well.

KNOWLEDGE CHECK

In the Firefox browser, pretty much all the ways mentioned in the above video will translate over except the shortcut to open the console panel directly (CTRL + SHIFT + J). Instead, this shortcut in Firefox is: **CTRL + SHIFT + K** / **CMD + OPTION + K**.

Now that we’ve seen the different ways that we can open DevTools, let’s move on to what we can actually do with them.

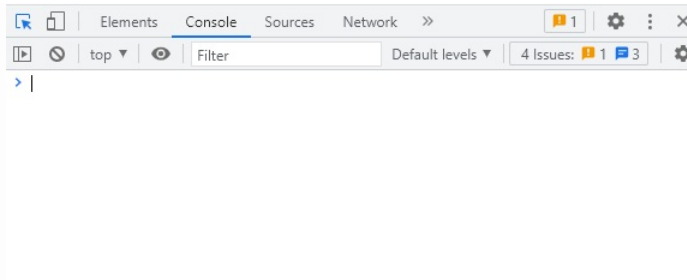
The Different DevTools Components



devtool toolbar

More than likely, as a frontend developer, you’ll be using the console panel, the elements panel, and the styles pane most often, along with ‘Device’ mode to view your webpage at different screen sizes, and the network panel occasionally. Let’s take a quick look at each of these.

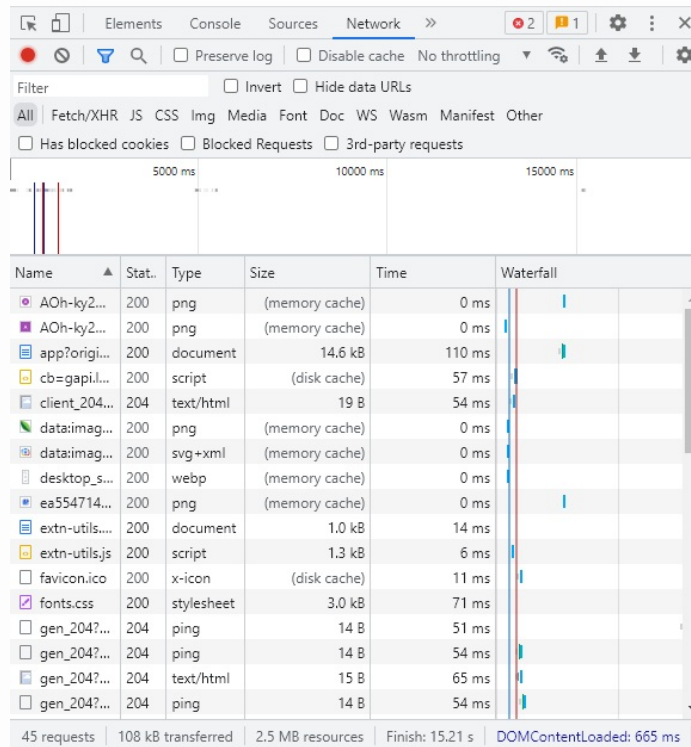
CONSOLE PANEL



console panel

The console is used for debugging JavaScript. You can also write JavaScript here directly.

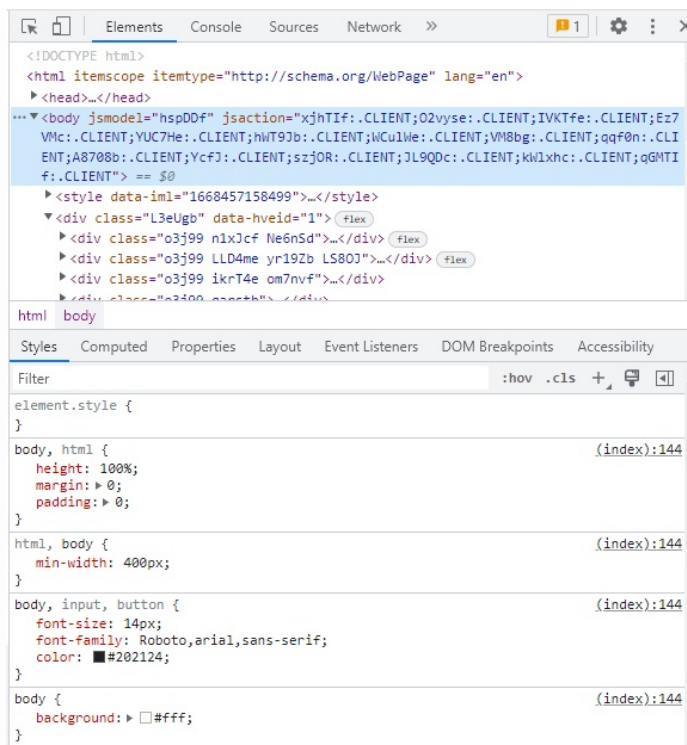
NETWORK PANEL



network panel

The network panel is useful primarily to inspect the status and response time of the server to requests for your web page's resources (HTML, CSS, JavaScript, and other included files).

ELEMENTS PANEL

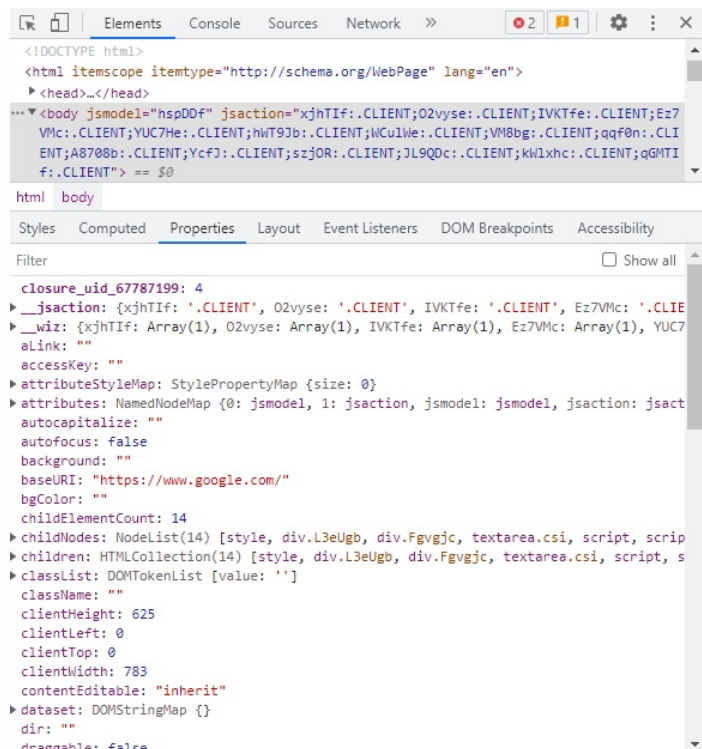


elements panel

The **Elements panel** is where to go to inspect HTML (shown in the top section of the image) and CSS (shown in the bottom section).

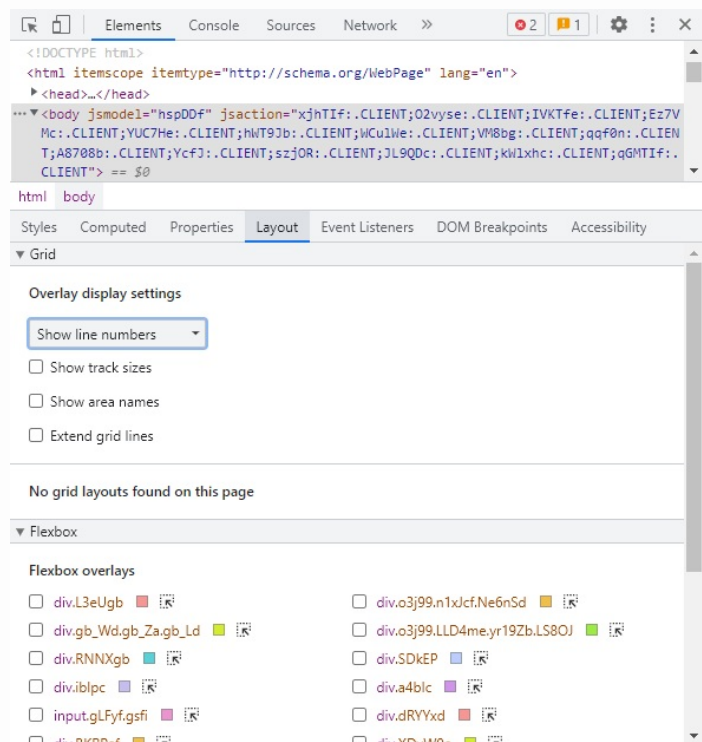
Within the Elements panel, there are a number of different useful panes:

- the **Styles pane**, where you can see all the different styles being applied to a given element, and also where you can make temporary changes (already pictured above)
- the **Properties pane**, which lists all the different properties for a given element



properties pane

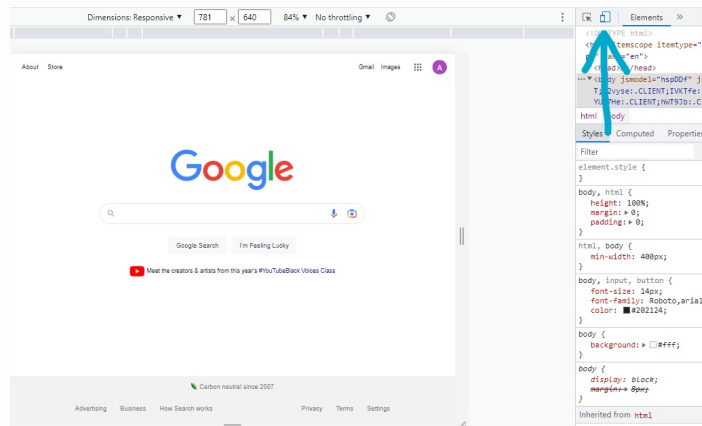
- The **Layout** pane, where you can inspect Grid and Flexbox containers and items



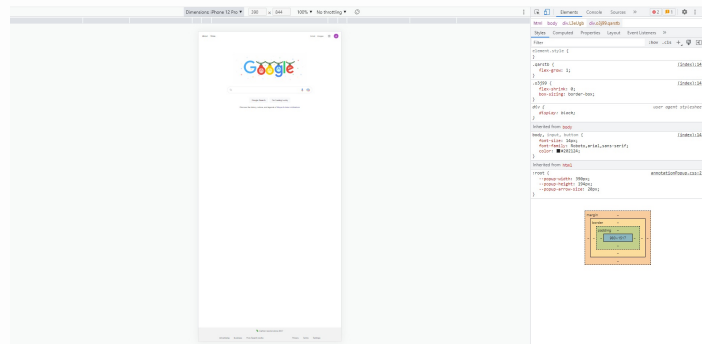
layout pane

- **Device mode**, where you can preview your webpage on different

screen sizes



device mode



device mode2

If you want to dive deeper into all you can do with DevTools be sure to have a look at the documentation. A link is provided in the Resources section.

KNOWLEDGE CHECK

Resources and Links

Documentation for Chrome's developer tools:
<https://developer.chrome.com/docs/devtools/>

Documentation for Firefox's developer tools: <https://firefox-source-docs.mozilla.org/devtools-user/>

Pseudo-classes and Pseudo-elements

What the heck are these pseudo things?

According to the MDN (Mozilla Developer Network) web docs (a link is in the Resources section), a pseudo-class is a “keyword added to a selector that specifies a special state of the selected element(s)” (e.g. - whether the element is being hovered over, or is currently in focus). These are written by adding a single colon to the selector, followed by the name of the pseudo-class.

A pseudo-element, on the other hand, according to MDN web docs, is a “keyword added to a selector that lets you style a specific part of the element”, and is written by adding a double colon to the selector followed by the name of the pseudo-element.

Let's look at some examples of both of these.

Pseudo-class & Pseudo Elements: Examples and Uses

First let's look at the basic syntax of pseudo-classes and pseudo-elements.

PSEUDO-CLASS

```
/* PSEUDO-CLASS SYNTAX */  
.someSelector:pseudo-class {  
  /* styles */  
}
```

pseudo-class

A pseudo-class is composed of the selector name, followed by a colon and then the name of the pseudo-class you want to target.

PSEUDO-ELEMENT

```
/* PSEUDO-ELEMENT SYNTAX */  
.someSelector::pseudo-element {  
  /* styles */  
}
```

pseudo-element

A pseudo-element is defined almost exactly the same way, except the selector and pseudo-element name are separated by a double-colon.

Here is a list of all pseudo-classes from W3schools:

https://www.w3schools.com/css/css_pseudo_classes.asp

Pseudo-classes

CSS pseudo-classes are a way to style elements based on their state changes. For example, when a user hovers over a button in the browser, it automatically adds the “hover” keyword to the selector. There are many different pseudo-classes available in CSS, and they can be very useful for targeting specific elements and creating dynamic styles.

Hover Pseudo-class:

The hover pseudo-class is used to style an element when the user hovers over it. For example, to change the background color of a button when it is hovered over, you would use the following CSS code:

```
button:hover {  
  background-color: blue;  
}
```

alt text

Focus Pseudo-class:

The focus pseudo-class is used to style an element when it has focus. This is particularly useful for form inputs, such as text fields and checkboxes. For example, to change the border color of a text field when it has focus, you would use the following CSS code:

```
input:focus {  
  border-color: red;  
}
```

alt text

First Child Pseudo-class:

The first child pseudo-class is used to target the first child element of a parent element. This can be useful for creating styles for the first element in a list or menu. For example, to change the font weight of the first list item in an unordered list, you would use the following CSS code:

```
ul li:first-child {  
  font-weight: bold;  
}
```

alt text

Last Child Pseudo-class:

The last child pseudo-class is used to target the last child element of a parent element. This can be useful for creating styles for the last element in a list or menu. For example, to change the color of the last link in a navigation menu, you would use the following CSS code:

```
nav a:last-child {  
  color: purple;  
}
```

alt text

Nth Child Pseudo-class:

The nth child pseudo-class is used to target a specific child element of a parent element based on its position. This can be useful for creating styles for every nth element in a list or menu. For example, to change the background color of every other list item in an unordered list, you would use the following CSS code:

```
ul li:nth-child(even) {  
  background-color: lightgray;  
}
```

alt text

Not Pseudo-class:

The not pseudo-class is used to target all elements except the one specified. This can be useful for creating styles for all elements in a list or menu except for the first or last element. For example, to change the font color of all links in a navigation menu except for the first link, you would use the following CSS code:

```
nav a:not(:first-child) {  
  color: green;  
}
```

alt text

Conclusion

CSS pseudo-classes are a powerful tool for creating dynamic styles based on the state of elements. They can be used to target specific elements, create styles for every nth element, and more. By mastering these techniques, students can become proficient in using CSS to create engaging and responsive web pages.

Pseudo-elements

CSS pseudo-elements are a way to target and style specific parts of a website that don't exist in the HTML markup. They can be used to create more efficient and maintainable code, as well as add advanced animations.

Here is a list of all pseudo-elements from W3schools:

https://www.w3schools.com/css/css_pseudo_elements.asp

Selector	Example
----------	---------

::after	p::after Insert something after the content of each < p > element
::before	p::before Insert something before the content of each < p > element
::first-letter	p::first-letter Selects the first letter of each < p > element
::first-line	p::first-line Selects the first line of each < p > element
::marker	p::marker Selects the markers of list items
::selection	p::selection Selects the portion of an element that is selected by a user

Example 1: Styling Placeholder Text

HTML

```
<form>
  <input type="text" placeholder="Enter your name">
</form>
```

CSS

```
input::placeholder {
  color: red;
  font-size: 18px;
}
```

Example 2: Styling First Letter of a Paragraph

HTML

```
<p>This is example default text</p>
```

CSS

```
p::first-letter {
  font-size: 24px;
  color: blue;
  font-weight: bold;
}
```

Example 3: Adding Content Before/After an Element

HTML

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
```

CSS

```
li::before {
  content: "🌟";
  margin-right: 5px;
}
```

In this example, the `::before` pseudo-element is used to add a star emoji before each list item.

Note that these are just basic examples to demonstrate the use of CSS pseudo-elements. There are many more possibilities and ways to use them depending on the desired effect.

Now let's see a couple more examples in code. Look over the HTML, as well as the CSS (beginning at the EXAMPLE 1 comment) in the editor below, then open the browser preview.

Note that the html and css are in the same window; you can click over to look at the html, but we don't need to because we are only working with the css for this example.

>

Try turning off the pseudo-classes and pseudo-elements to get a feel for how they work.

Exercise

EXERCISE:

Make the button change color when hovered over. Try this exercise on your own, but if you get stuck, you can look at the preceding example.

****Note:** since the button currently has a dark background color with white text, choose another somewhat dark color for the hover state. Or, optionally, you can also change the button's text color to a dark color if you prefer to use a lighter color for the button's background.

Hint: Give the button a class, and look at the code on the last page for how to create a pseudo-class to hover.

There are many more pseudo-classes and pseudo-elements than those covered here. To learn more about them, check out the MDN Web Docs.

Resources and Links

MDN Web Docs: <https://developer.mozilla.org/>

CSS-Tricks - Pseudo Class Selectors: <https://css-tricks.com/pseudo-class-selectors/>

Sass up your Styles: The Awesome CSS Preprocessor

Sass up your Styles: The Awesome CSS Preprocessor



tbht sassy gif

What is Sass (and why it's awesome)

Today we're going to dive into the world of Sass - a language that extends CSS with additional features to simplify your CSS code and make it more efficient.

At its core, Sass is designed to make the process of writing CSS easier and more streamlined. It allows you to write less code while still achieving the same results.

Sass has two syntaxes:

- **indented syntax** - The indented syntax uses whitespace and indentation instead of semicolons and curly braces to define blocks of code.
- **SCSS (superset syntax)** - The SCSS syntax is a superset of CSS which means all valid CSS is also valid SCSS and is fully compatible with it.

To demonstrate the difference between Sass and CSS, let's take a look at a sample code snippet.

```
// CSS code snippet

body {
  background-color: #1abc9c;
}

// Equivalent SCSS code snippet

$primary-color: #1abc9c;

body {
  background-color: $primary-color;
}
```

As you can see, Sass allows you to use variables to store commonly used values, which can be referenced later in the code. In this case, we've defined a variable `$primary-color` with the hex color value `#1abc9c` and used it to set the background color of the body element.

This means that if we need to change the primary color later on, we only need to update the variable, and the change will be reflected throughout the code.

Variables

Variables are a powerful feature of Sass that allow you to define named values that can be reused throughout your code. They make it easy to update styles consistently and ensure that your code is easy to maintain.

In Sass, variables are declared using the dollar sign symbol followed by the variable name and a value. For example, `$primary-color: #00ff00;` would declare a variable named `primary-color` with the value `#00ff00`.

To use a variable in your code, you simply call its name (without the dollar sign symbol) in place of a value. For example, `color: $primary-color;` would set the text color to the value of the `primary-color` variable.

Let's take a look at an example to see how variables can be used in practice:

```
// Declare a variable for the primary color

$primary-color: #1abc9c;

// Use the variable to set the background color of the body

body {
  background-color: $primary-color;
}

// Use the variable to set the color of headings

h1, h2, h3 {
  color: $primary-color;
}
```

In this example, we've declared a variable for the primary color and used it to set the background color of the body element and the color of all headings. This means that if we want to change the primary color later on, we only need to update the variable, and the change will be reflected throughout the code.

Nesting

Nesting is another powerful feature of Sass that can help simplify CSS code. It allows styles to be written within the scope of a parent selector, which eliminates the need for repeating the parent selector multiple times.

This means that if you have a set of styles that apply only to a specific element within another element, you can nest the styles inside the parent element, and they will only apply to that specific child element.

Nesting can also be used to group related styles together and make code more readable. For example, you can nest hover styles within a parent selector or nest styles for a navigation menu.

Let's take a look at an example. In plain CSS, you might have something like this:

```
CSS
.nav {
  background-color: #fff;
}

.nav ul {
  list-style: none;
}

.nav ul li {
  display: inline-block;
}

.nav ul li a {
  color: #000;
}

.nav ul li a:hover {
  color: #fff;
  background-color: #000;
}
```

In Sass, we can use nesting to simplify this code:

```
CSS
.nav {
  background-color: #fff;

  ul {
    list-style: none;

    li {
      display: inline-block;

      a {
        color: #000;

        &:hover {
          color: #fff;
          background-color: #000;
        }
      }
    }
  }
}
```

As you can see, the styles are nested within their parent selectors, which makes the code much more readable and easier to follow.

You can nest styles for any selector, including hover states, pseudo-selectors, and media queries. This can greatly simplify your code and make it easier to maintain.

Mixins

Mixins are a way to group styles together and apply them to multiple selectors without repeating code. You can create a mixin by giving it a name and listing the styles it includes. Then you can apply that mixin to selectors using the `@include` directive.

This can be helpful for reusing common styles like gradients or drop shadows. Here's an example of a mixin that sets a gradient background:

```
less
@mixin gradient-background($start-color, $end-color) {
  background: $start-color;
  background: linear-gradient(to bottom, $start-color, $end-color);
}

.button {
  @include gradient-background(#FFA07A, #FF4500);
}
```

In this example, we're creating a mixin called `gradient-background` that takes two arguments: `start-color` and `end-color`. The mixin sets the background color to `start-color` and then applies a linear gradient background from `start-color` to `end-color`.

To use the mixin, we simply call `@include` followed by the mixin name and any arguments. In this case, we're applying the `gradient-background` mixin to a `.button` selector and passing in the start and end colors as arguments. This will set the background of the `.button` to a gradient from `#FFA07A` to `#FF4500`.

Functions

Functions are operations that can change the values of colors, numbers, or strings in Sass code. Sass has built-in functions like `lighten()`, `darken()`, `rgba()`, and `percentage()` that can perform calculations, modify colors, or generate random values. For example, `lighten()` and `darken()` can be used to make a color brighter or darker, and `percentage()` can be used to calculate the width of an element as a percentage of its parent container.

To use a built-in function in Sass, the function name is written followed by parentheses, and the value to be operated on is passed as an argument. For example, to lighten a color by 10%, the `lighten()` function can be used as follows:

```
CSS
$primary-color: #3b5998;

background-color: lighten($primary-color, 10%);
```

This code will make the background color of an element 10% lighter than the primary-color.

This code will make the background color of an element 10% lighter than the primary-color.

Now onto an example:

Sassy Example

Now, to see Sass in action, as well as the CSS output after Sass compiles it, take a look at the example below, which has a refactored version of the code from the Pseudo-class and Pseudo-elements section – this time taking advantage of some of Sass’s features!

>

Note: You will make your changes in the sass file, `.scss`. However, you need a `.css` file for the styling. You will have to use the terminal for that. To use the terminal, press `shift-alt-t`.

This blockquoted section is only necessary on your own machine. The environment here should already have sass installed and ready to run. You can skip to the paragraph after the blockquote. Once in the terminal, you will want to install sass. You will do this with the command `npm install sass`. npm is a software registry for node projects. Node is used for javascript projects, which is primarily web development. We’ll see more on this later in the course.

Now, type `npm install` which will install the scripts in the `package.json` file.

Once you’ve done that, sass should be set up for you.

When you edit sass and want to see your changes, you have to go back to the terminal and type `npm run sass` (this will update the `.css` file) and then refresh your preview!

What's happening here? You should see a file called `package.json`. This contains a piece of code that will update (or create if it wasn't there) a file called `style3.css` from the sass file called `style.scss`. When you edit the `.scss` file, it doesn't do anything unless you tell it to create the `.css` file that the html file is using for styling!

You see line 8 in the file `index3.html`? It is: `<link rel="stylesheet" href="style3.css">`. You use the `.scss` sass file to create the `.css` file, and then the `.html` file uses that for styling.

Note: the `.scss` file and the `.html` file are sharing a window. You can switch back and forth between them.

Exercise

EXERCISE:

Try creating a mixin on your own which includes the following styles:

```
display: block;
margin: 0 auto;
padding: 1.5rem;
```

If you get stuck, reference the material above or you can check out the finished example at the end of this section.

How did you do? If all went well, your code should look something like the following:

```
@mixin myAwesomeMixin {
  display: block;
  margin: 0 auto;
  padding: 1.5rem;
}
```

mixin styles completed

Resources and Links

Sass: <https://sass-lang.com/>

Sass - Install: <https://sass-lang.com/install>

Sass - Basics: <https://sass-lang.com/guide>

Conclusion & Takeaways

Conclusion & Takeaways

DevTools are an extremely helpful tool for frontend development. They allow you to get a closer look at what's going on with your code so you can make the right changes faster to get everything working and looking good.

Pseudo-elements and pseudo-classes, while potentially confusing at first, give us the ability to add some additional flair to our styles. We saw how using the **:hover** pseudo-class allows us to create a nice change-color effect when the user moves their mouse over it.

In addition, in this section we learned that Sass gives us the ability to do many powerful things we can't do in CSS (i.e. - mixins and selector nesting) and can help make styles easier and more fun to write. You saw some code examples of variables, mixins, nesting, as well as a couple built-in functions included by Sass, and also got some practice creating your own mixin.

Attribution

1. CSS gif. (n.d.). GIPHY. <https://giphy.com/gifs/frustrated-annoyed-programming-yYSSBtDgbbRzq>
2. Chrome DevTools Documentation. (2016, March 28). Overview - Chrome Developers. <https://developer.chrome.com/docs/devtools/overview/>
3. (2021, May 18). Sassy Big Bang GIF by Wisconsin Sportscenter. GIPHY. <https://giphy.com/gifs/WisconsinSportscenter-howard-oh-no-he-didnt-she-w7XSKW9twsbMNBnZDE>

Responsive Design

Goals

By the end of this lesson you will:

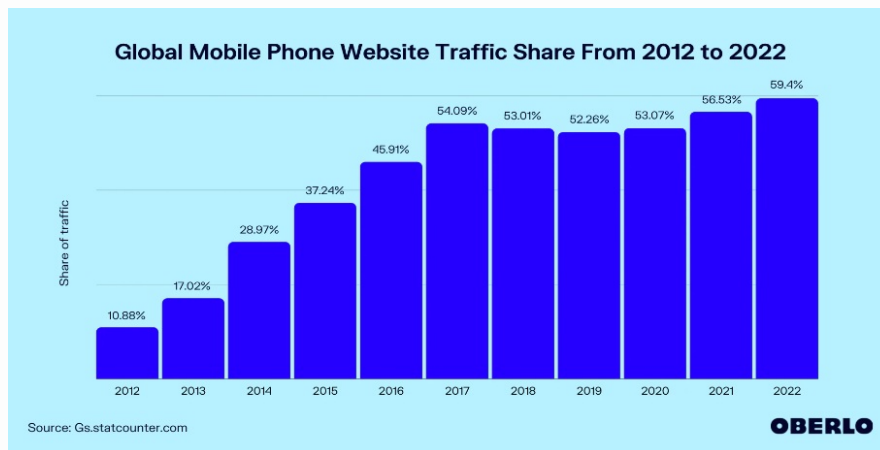
- Understand responsive design and its importance
- Be familiar with the two approaches: desktop-first and mobile-first
- Be able to apply responsive design to your own projects

>

Introduction

Responsive web design (RWD), a term first used by web designer Ethan Marcotte in an [article](#) in 2010, is a critical foundation of being a frontend developer in today's world where people are interacting with the web not only on computers anymore, but also (and even primarily) on phones or tablets.

>



>

In fact, the most recent statistic as of November 2022, according to [Oberlo](#), is that a staggering 60.28% of websites are visited from mobile phones.

>

In this lesson, you will learn some of the foundations and key concepts of responsive design and how to apply them.

>

Philosophy of Responsive Design

>

So what is responsive design? Very simply put, it's a way of building web pages so that they will look good no matter what device is being used to view them.

>

Watch the following video which will give a brief intro to responsive design and some of its important concepts, then complete the Knowledge Check.

>

[Responsive Web Design | 10 Basics \(6:37\)](#)

>

The Viewport & Other Considerations

The crux of responsive design is the **viewport** which, according to the [MDN Web Docs](#), is defined as “...the part of the document you’re viewing which is currently visible in its window (or the screen, if the document is being viewed in full screen mode).”

With that in mind, HTML provides us with a meta tag that is very important to include in the head of our HTML document. Responsiveness begins with this tag!

If you're using an editor like the one here, or VS Code, you can make use of one of the Emmet abbreviations to generate the HTML boilerplate (typing either `"!"` or `"html:5"`, then hitting tab), and you'll see that this tag is already included for you.

In case you need to include it manually, here is what it looks like:

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

If you want more info about this tag, check out the MDN web docs page [here](#).

Let's look at an example to better illustrate all of this. Say we have some container on our page and a considerable amount of text in it:

HTML Innate Responsiveness

[illegible]

responsiveness example

The text is hard to read in this image, but that's okay... it's just placeholder text and isn't important for the purposes of this example.

Right now, the only styles applied are the following:

>

```
body {
  padding: 0 4rem;
}

h1 {
  text-align: center;
  margin-bottom: 5rem;
  font-family: 'Courier New', Courier, monospace;
}

.container {
  border: 2px solid #000;
  padding: 1.5rem;
  margin: 0 auto;
}
```

>

The code for this example is included in the editor below:

>

Try commenting out (**CTRL** + / (Windows/Linux) or **CMD** + / (Mac)) the **padding** rule on the body element, and then adding different **width** and **height** values to the **container** class and open the page in the browser to see the result. Then, try setting a **max-width** of something like **25rem** and a **min-height** of **40vh** (viewport height units) to see the difference.

>

Just a few import tips to keep in mind when you're writing your own responsive styles.

>

Another important thing to keep in mind is that HTML in many ways already tries to be responsive on its own, and oftentimes if we're not careful when writing CSS we can actually make things less responsive!

>

A good rule-of-thumb in this regard: on most HTML elements, it's generally better to avoid explicitly setting the **height** property (and be cautious setting fixed widths as well) unless there's a good reason to do so.

>

Setting fixed widths can sometimes cause sideways scrolling, which you don't want in modern web design. Setting explicit heights can be even worse, causing the content to break out of the container element!

>

Most of the time, if you have to set widths and/or heights, it's better to use **min-height**, **min-width** or **max-width**, or even **padding** in certain instances (a navbar, for example), in order to achieve the desired height of an element while still preserving some of the baked-in responsiveness.

>

Now, let's move on to the bread and butter of responsive design: **media queries**.

Media Queries

Take a look at this video to get a good basic grasp of media queries – what they are, and how to use them.

[Tutorial: Learn how to use CSS Media Queries in less than 5 minutes](#)
(4:29)

The basic syntax for writing a media query is as follows:

```
@media (/* WIDTH VALUE */) {  
  /* STYLES TO APPLY AT THIS VIEWPORT WIDTH */  
}
```

What this is saying is: “Apply these styles at this particular viewport width”. For the width value, you can use either **min-width** or **max-width**.

The difference is that using **min-width** here means “apply these styles only when the viewport is equal to or greater than the width specified”, whereas using **max-width** means “apply these styles only when the viewport width is equal to or less than the width specified”.

Which one you should use depends on whether you’re designing your page starting from the smaller screen sizes and working your way up (mobile-first), in which case you would likely use **min-width** most often. If you’re designing the desktop version of your page first and working your way down you would use **max-width** most often. You can also use a range of **min-width** and **max-width**.

Here’s an example of what all of those would look like:

```
/* MIN-WIDTH */  
@media (min-width: 600px) {  
  /* STYLES TO APPLY */  
}  
  
/* MAX-WIDTH */  
@media (max-width: 800px) {  
  /* STYLES TO APPLY */  
}  
  
/* RANGE WIDTH */  
@media (min-width: 600px) and (max-width: 900px) {  
  /* STYLES TO APPLY */  
}
```

When using media queries, you need to figure out at what specific viewport sizes the styles should change so that your page still looks good – in other words, where your design begins to break. These viewport sizes, then, are in this context referred to as **breakpoints**.

There are many opinions regarding which breakpoints are best to use. A standard set used frequently are those utilized by Bootstrap (an immensely popular CSS toolkit which we will learn about in a later lesson):

Available breakpoints

Bootstrap includes six default breakpoints, sometimes referred to as *grid tiers*, for building responsively. These breakpoints can be customized if you're using our source Sass files.

Breakpoint	Class infix	Dimensions
Extra small	<i>None</i>	<576px
Small	<i>sm</i>	≥576px
Medium	<i>md</i>	≥768px
Large	<i>lg</i>	≥992px
Extra large	<i>xl</i>	≥1200px
Extra extra large	<i>xxl</i>	≥1400px

min and max width

Here's a link to Bootstrap's breakpoint documentation: [Breakpoints · Bootstrap v5.2](#)

Another school of thought, however, takes a different approach:

The 100% correct way to do CSS breakpoints

The author of this article suggests breakpoints of **600px**, **900px**, **1200px**, and finally **1800px** “if you plan on giving the giant-monitor people something special”.

A perhaps less structured, but equally as valid, approach is to simply use the Device mode in DevTools to figure out where your page starts to not look good anymore, and create a media query at that viewport width.

So which approach should you use? That's entirely up to you. Whatever works for your project and makes sense to you.

Example

Let's look at an example to get some practice with media queries. In the code editor below, try adding a media query at the end of the CSS file modifying the grid at **max-width: 750px** so that inside the **.grid-container** selector, **grid-template-columns** is now **1fr** (fraction unit). If you remember from last week, *fr* units are a special unit used in CSS Grid.

Give this a try on your own, but if you get stuck, you can reference the finished code on the next page.

Example and Conclusion

Conclusion & Takeaways

Given the vast number of devices on which people are accessing the web these days, responsive design is critical in modern frontend development. There are a number of best practices you can follow, including the use of media queries with sensible breakpoints, as well as avoiding the use of explicit widths and heights except where absolutely necessary, so that your web pages will look great no matter where they're viewed.

>

Attribution

>

1. Marcotte, Ethan (May 25, 2010). Responsive Web design. A List Apart.
1. Gaubys, J. (n.d.). What Percentage of Internet Traffic Is Mobile? [Dec '22 Upd]. <https://www.oberlo.ca/statistics/mobile-internet-traffic>

CSS Animations

Goals

By the end of this lesson you will:

- Be familiar with basic transitions and animations in CSS
- Know how to begin using transitions and animations effectively
- Understand accessibility concerns regarding animations and how to address them

Introduction

In the world of CSS, there are many ways to add flair to your projects.



pic

Among them are **transitions** and **animations**. Transitions are a more basic form of animation, and are great for simple effects where you want one or a couple properties of an element to transition from one state to another.

CSS Animations, on the other hand, while functioning in a similar way give you a little more firepower to create more complex visual effects on a web page.

According to the MDN web docs, CSS Animations are composed of two components: “a style describing the CSS animation and a set of keyframes that indicate the start and end states of the animation’s style, as well as possible intermediate waypoints”.

Let’s dive in and see how we can begin using these “pieces of flair”, as it were.

CSS Animation: The Building Blocks

CSS Animation: The Building Blocks

Watch the following video to get a basic overview of transitions and animations.

[CSS Animation in 100 Seconds](#) (2:04)

Transitions are most commonly defined using the single line transition shorthand:

```
div {  
  transition: <property> <duration> <timing-function> <delay>;  
}
```

This is the [basic definition example](#) on MDN. The last item in the definition (delay) you might not use that often, but this is how you would include it if you wanted there your transition not to begin immediately.

Let's look at an example with values plugged in for all these sub-properties:

```
button {  
  width: 10rem;  
  border: 3px solid #000;  
  border-radius: 5px;  
  padding: 0.5rem 1.5rem;  
  background-color: #fff;  
  cursor: pointer;  
}  
  
button#transition {  
  scale: none;  
  
  /* ADD THE TRANSITION SHORTHAND */  
  transition: scale 200ms ease-in-out;  
}  
  
button#transition:active {  
  scale: 0.95;  
  
  /* Optionally, you can add the transition here as well */  
  transition: scale 200ms ease-in-out;  
}
```

What this is doing is scaling the button down when clicked, using the **:active** pseudo-class. The transition being applied is on the **scale** property, with a duration of **200 milliseconds** and is using the **ease-in-out** timing function. The details of these timing functions can get a bit complex, so we won't go into detail here. If you want to read more about them go to the following link:

[animation-timing-function - CSS: Cascading Style Sheets | MDN](#)

Now let's look at an example of an animation, which, as mentioned previously, consists of two elements: a definition (as shown below), and some keyframes.

```
div {  
  animation: <animation-name> <animation-duration> <animation-fill-mode>;  
}
```

This example is only using three of the animation sub-properties, but you could include more.

Keyframes are defined as follows:

```
@keyframes my-awesome-animation {  
  /* STARTING STYLES */  
  0% {  
    /* ... */  
  }  
  
  /* ENDING STYLES */  
  100% {  
    /* ... */  
  }  
}
```

Let's see what this looks like with values plugged in:

```
button#animation:focus {  
  scale: 0.95;  
  animation: my-awesome-animation 1s forwards;  
  animation-delay: 700ms;  
}  
  
@keyframes my-awesome-animation {  
  0% {  
    rotate: 0;  
    scale: 0.95;  
  }  
  
  100% {  
    rotate: 360deg;  
    scale: 1;  
  }  
}
```

This animation is **scaling down** the button when pressed, waiting **700 milliseconds**, then rotating the button by **360 degrees** and finishing at the button's original scale. Since we added **forwards** to this animation, it will only run once per focus event.

One important note here is that not all properties in CSS can have transitions or animations applied to them. See this [list from MDN](#) of those properties which can be transitioned/animated.

There will be links to more resources in the Resources section. For now, let's see some more examples and get some practice with basic animations.

We'll start with a simple example using transitions. Check out the code below, then open the browser preview.

Example

Now let's see an example of a basic CSS Animation:

Exercise

Now try on your own to create a simple fade-in animation. There is only one property you need to worry about here: **opacity**.

>

We will add this to our starter code:

```
.fade-in {  
  animation: fade-in 3s forwards;  
}  
  
@keyframes fade-in {  
  from {  
    opacity: 0;  
  }  
  
  to {  
    opacity: 1;  
  }  
}
```

Best Practices & Accessibility

Best Practices & Accessibility

One very important note and best practice is not to overuse transitions and animations. There are a couple reasons for this:

1. Too many animations become too much, visually, and can actually distract users from the content
2. For certain users, animations can physically affect them in an adverse way

>

On this second point, we want to be mindful of users with vestibular motion disorders who will likely set a preference in their browser for reduced animation. It's a good idea when writing your styles to add a media query for the feature called "prefers-reduced-motion". It would look like this:

```
@media (prefers-reduced-motion) {  
  /* styles */  
}
```

One way of going about things for **prefers-reduced-motion** is to simply apply the styles of the finished state of the animation, or those which most effectively express the styles you want to convey.

Conclusion & Takeaways

Conclusion & Takeaways

Transitions and animations are great to create beautiful smooth visual effects to make your web page look dynamic and professional. However, as mentioned, be careful not to overuse them, or it may have the opposite effect.

Attribution

- Office Space - 37 pieces of flair. (n.d.). Memesmonkey.
<https://www.memesmonkey.com/images/memesmonkey/e1/e11e886bb1fd58e016fd770ce09fefc3.jpeg>
- Using CSS transitions - CSS: Cascading Style Sheets | MDN. (2022, October 21). https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions
- Using CSS animations - CSS: Cascading Style Sheets | MDN. (2022, September 28). https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations

Advanced CSS Grid

Goals

By the end of this lesson you will:

- Have brushed up on the basics of CSS Grid
- Be familiar with some of the more advanced ways to use CSS Grid

Introduction

CSS Grid is an immensely powerful tool at our disposal for laying out our content. You learned the basics last week (which we will review again briefly), and then you'll see some of the more advanced ways you can use Grid.

Brief review of Grid basics

Let's briefly recap the basics of CSS Grid.

First you set up a parent container, set its display property to 'grid', then you need to begin setting up columns and/or rows. Take a look at the code below:

Here you can see the basic setup, and for now we're just setting **grid-template-columns** to '1fr 1fr 1fr' to get a simple 3 column layout. Now, we can start defining rows and how much space we want each item to occupy.

As you learned last week, you can use different values when setting columns and rows: **px**, **em**, **rem**, **%**, **fr**, or a combination of them. Then, if you want to exercise more control on the grid items, you can begin specifying how many rows or columns that item should take up, using **grid-column** or **grid-row** on the child elements of the grid container.

Example

The following code shows, while not the most practical of examples, nonetheless how these features make controlling a layout much easier.

```
>
```

Advanced Grid

Now that we've reviewed the basics, let's go over some of the more advanced things you can do with Grid which, almost counter-intuitively, end up making Grid a bit easier to work with than explicitly setting columns and rows.

grid-auto-flow

First up is **grid-auto-flow**. This lets us define what we want the flow of the grid to be (i.e. - **row** or **column**). Let's go back to where we left off in the Grid basics review.

>

Now, let's comment out grid items 6 through 10 in the HTML, the **grid-template-columns** and **grid-template-rows** lines in **.grid-container** selector in our CSS, as well as everything below the **.grid-item styles**. Back in **.grid-container**, on a new line just below the commented-out lines, write **grid-auto-flow: column** then open the browser preview and see the result.

...and voila! All the grid items are still in columns, but with an important difference: this is more dynamic. Now if we add more grid items, the grid will auto-adjust everything to accommodate the new content. Try uncommenting some of the grid items in the HTML to see how this works.

Using repeat() with auto-fill and minmax()

A very useful way to define **grid-template-columns**, especially in terms of responsiveness, is using **auto-fill** and the **minmax()** CSS function inside the **repeat()** function. The value **auto-fill** (or the similar **auto-fit**) refers to how the grid items will be distributed in the grid container. The **minmax()** function will allow us to specify minimum and maximum widths of each column in the grid.

Let's look at an example in code to see the difference.

Uncomment all the grid items in the HTML. Comment out the line in CSS where we set grid-auto-flow. Now, add the following just below it:

```
grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));  
grid-template-rows: auto;
```

>

This has essentially created a fluid grid. Try opening the browser preview and resizing the window to see the result.

Using grid-template-areas

Another way of defining a grid in CSS is using **grid-template-areas**. This method is sort of like creating a map of where you want all of your web page's components to live, and involves two steps to implement:

- Setting the **grid-template-areas** property in a grid container using the following syntax:

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

>

- Setting grid items to a specific area of the template using the **grid-area** property:

```
.header {  
  grid-area: header;  
}
```

>

This feature of CSS Grid is very powerful and affords more precise control over a layout. Let's see how it can be used to create a variation of the so-called "Holy Grail" layout:

Exercise

Exercise:

Try recreating the “Holy Grail” layout on your own. If you get stuck, you can of course refer to the completed example above.

>

Some starter code has been provided.

>

Conclusion and Takeaways

CSS Grid is a fantastic tool at our disposal for gaining control over web page layouts. While it can be somewhat complicated to learn, mastering it will definitely pay off. You can use it in simple ways or very complex ways, depending on the needs of your project.

Introduction to Flexbox

Goals

By the end of this case you will:

- Be familiar with flex containers and flex items, and the different properties relevant to each
- Understand how Flexbox helps with responsive design
- Begin using Flexbox (on its own, or with Grid) in your own projects

Introduction

You've already seen the power of using CSS grid for your overall page layout. CSS also provides us with another extremely powerful layout tool called Flexbox. In this lesson, we're going to first dive into the basics of using Flexbox on its own then see how we can use it with CSS Grid.

Layout purpose: Choosing Box, Flexbox or Grid?

So what does Flexbox do exactly, and what makes it different from Grid? Both are layout tools included in CSS, but whereas Grid operates on two dimensions at once (row and column), Flexbox operates on one dimension at a time (row or column).

As mentioned in the Introduction, you can use Flexbox in conjunction with Grid. Or you can use Grid only... or Flexbox only. Or you can use neither and simply rely on the natural flow of the box model. So, how do you decide? Watch the following video, which tackles this very question.

[Flexbox vs. CSS Grid — Which is Better?](#)(4:17)

Flex containers, flex items, and their properties

So what are the different components of Flexbox, and how do we get started with it? Before looking at an example in code, take a look at the following diagram which shows the different properties which apply to Flex containers and Flex items respectively:

Flex Container Properties	Flex Item Properties
Display Flex-direction Flex-wrap Gap Align-items Justify-content	Flex-basis Flex-shrink Flex-grow Flex (shorthand for flex-basis, flex-shrink, and flex-grow)

flex container and item properties

Have a look at the following code and open the browser preview to see an overview of these properties.

Flex direction and aligning content (the two axes)

Flexbox, like Grid, is a great tool for aligning or centering content. When thinking about alignment in Flexbox, it's important to know that there are two axes on which it operates and this changes depending on the **flex-direction**. The two axes are commonly referred to as

1. the main axis
2. the cross axis

With **flex-direction: column**, the main axis is the vertical axis and the cross axis is the horizontal axis, whereas with **flex-direction: row**, it's the opposite. To get a clearer picture of this let's look at another example showing both flex-directions:

Try changing up the **align-items** and **justify-content** values for each flex-direction to see the result.

Flexbox and responsive design and Conclusions

A simple way you can use Flexbox for responsive design is to change the **flex-direction** property at a certain breakpoint. For example, changing from **flex-direction: row** to **flex-direction: column**.

We'll see an example of this in the next section when we work through an exercise.

Conclusion & Takeaways

Whatever you end up choosing as your go-to layout tool (be it Grid, Flexbox, or even just using the box model with floats and clears), it's always a good idea to add more tools to your toolbelt because every project is different and knowing a variety of ways to solve a problem is an essential skill for a developer.

Exercise: Apply what you've learned

Exercise:

Apply what you've learned.

>

Goals

By the end of this exercise you will:

- Have used animations and media queries to a common UI scenario
- Have used Flexbox to create a responsive design

>

Introduction

We've covered a lot today. We went over responsive design, media queries, animations, we reviewed Grid and learned some new aspects of it, and also began to look at Flexbox. To wrap things up for this module, we'll now apply what we've learned to a UI component common on many web pages: a hero section. A hero section usually included in the landing page of a website. It tells the visitor what you offer, who you are, why to work with you/use your product, and what to do next.

>

The finished product should look like this:

Flexbox - Hero Section (Exercise)

Discover your inner foodie.

Lorem ipsum dolor sit amet consectetur, adipiscing elit. Officiis architecto dolore fuga dignissimos deleniti magni inventore et eius vero laborum.

[READ MORE](#)

[ORDER NOW](#)



>

Getting setup with Flexbox

Let's start by adding Flexbox to the layout.

>

In the CSS file, add the following to the .hero-container selector:

>

```
.hero__container {  
  display: flex;  
  align-items: center;  
  justify-content: space-evenly;  
  width: min(100% - 2rem, 1300px);  
  min-height: 70vh;  
  margin-inline: auto;  
  padding: 2rem;  
}
```

This will set the element up as a flex container, center everything, and apply a dynamic width as well as some padding.

Now, to start styling the text, let's add some styles to ****.hero__heading**** and ****.hero__subheading****:

>

```
.hero__heading {  
  font-size: 4rem;  
  max-width: 25ch;  
  margin-bottom: 2rem;  
}  
  
.hero__subheading {  
  font-size: 1.75rem;  
  max-width: 50ch;  
}
```

Now we want to make our page interactive. A lot of the time you'll go back and forth between your html and css files while you're working. Here, we want to add two buttons. Add the following code just below the div in your with the "Lorem ipsum" text to create two buttons:

```
<div class="hero__buttons">  
  <button class="read-more-btn">READ MORE</button>  
  <button class="order-now-btn">ORDER NOW</button>  
</div>
```

Those buttons could definitely look better. Let's add some margin on the top, some padding inside, and a gap between them.

>

```
.hero__buttons {  
  margin-top: 3.5rem;  
  display: flex;  
  gap: 1.5rem;  
}  
  
.hero__buttons > button {  
  padding: 0.7rem 2rem;  
  border-radius: 7px;  
  border: none;  
  cursor: pointer;  
}
```

But maybe we want to style each of these buttons slightly differently, so that one stands out over the other. For example, we could make one a solid color with light text so it really pops, and then have the other button transparent with a darker border and text color:

```
.hero__buttons > .read-more-btn {  
  background: transparent;  
  border: 3px solid darkslategray;  
  color: darkslategray;  
  font-weight: 600;  
  transition: all 300ms ease-in-out;  
}  
  
.hero__buttons > .order-now-btn {  
  background-color: darkslategray;  
  font-weight: 600;  
  color: #fff;  
  transition: all 300ms ease-in-out;  
}
```

Okay, so far so good. Next let's work on adding some transitions and animation.

Adding transitions and a fade-in-right animation

To add some sizzle to our design, let's add the following transition to make the buttons scale slightly larger when they're hovered over:

```
.hero__buttons > .read-more-btn:hover {  
  transform: scale(1.07);  
  font-weight: 600;  
  transition: all 300ms ease-in-out;  
}  
  
.hero__buttons > .order-now-btn:hover {  
  transform: scale(1.07);  
  transition: all 300ms ease-in-out;  
}
```

Now we could take things one step further and add a subtle but slick **fade-in-right** animation.

In the ****.hero__container**** selector, add the following:

```
animation: fade-in-right 500ms forwards;
```

This specifies that our animation name is “fade-in-slide-up”, we want it to take 500 milliseconds for the animation to complete, and by adding “forwards” we only want the animation to run once.

Now, we need to define the keyframes for the animation. At the end of our CSS, add the following:

```
@keyframes fade-in-right {  
  0% {  
    transform: translateX(-10%);  
    opacity: 0;  
  }  
  
  100% {  
    transform: translateX(0);  
    opacity: 1;  
  }  
}
```

The code above says that we want to start the `.**hero__container**` out 10 percent to the left of its original position and completely transparent (invisible), and finish by moving the container back to its original position in the document flow (0) and be completely visible.

Once you hit 'Save', if everything was done correctly you should see the hero text and image fade in and slide right.

Great!

Adding media queries

Last, but not least, let's add a couple media queries so that our hero section will look good (and eventually stack vertically) on smaller screens. First we'll add one removing all the animations for the folks who prefer reduced motion.

At the very end of our styles, below the **@keyframes**, add the following:

```
@media (prefers-reduced-motion) {  
  .hero__container {  
    animation: none;  
  }  
  
  .hero__buttons > .read-more-btn:hover {  
    transform: none;  
    transition: none;  
  }  
  
  .hero__buttons > .order-now-btn:hover {  
    transform: none;  
    transition: none;  
  }  
}  
  
@media (max-width: 830px) {  
  .hero__subheading {  
    max-width: 25ch;  
  }  
}  
  
@media (max-width: 770px) {  
  .hero__container {  
    flex-direction: column-reverse;  
    align-items: center;  
    gap: 1.5rem;  
  }  
  
  .hero__text {  
    text-align: center;  
  }  
  
  .hero__buttons {  
    justify-content: center;  
  }  
}
```

The breakpoints above (**770px** and **830px**) were arrived at using Device mode in DevTools to see where the design started to look not so great, but you can use different ones if you choose so long as the design still resizes smoothly and looks good on mobile.

If we did everything right, when we resize the screen to less than 770 pixels we should now see all the content of our hero section stacking vertically. The image should now be on top of the hero text, and all the text should be center-aligned.

If needed, here is the finished code for reference:

Conclusion & Takeaways

Here is our final code! It is nicely formatted and looks much more professional.

In this section, you got some valuable practice building a version of a very common real-world design. You added some transitions, animations, and media queries so that your project would look good on various screen sizes and would account for those who prefer reduced motion and animation. You've accomplished a lot the past two days. Take a moment and enjoy it!

Responsive Websites with Bootstrap



bootstrap image

Goals

By the end of this lesson you will:

- Understand what Bootstrap is, and why it's helpful for frontend development
- Know how to get Bootstrap setup in a project
- Be familiar with Bootstrap's grid system and how to use it for responsive design

Introduction

In the world of frontend development, there are several tools that aim to remove some of the tedium that comes with writing custom styles for many common web page components (navbars, buttons, cards, etc.).

One of the most popular of these tools is **Bootstrap**. Throughout today's lessons and project, you will be introduced to Bootstrap and many of its features.

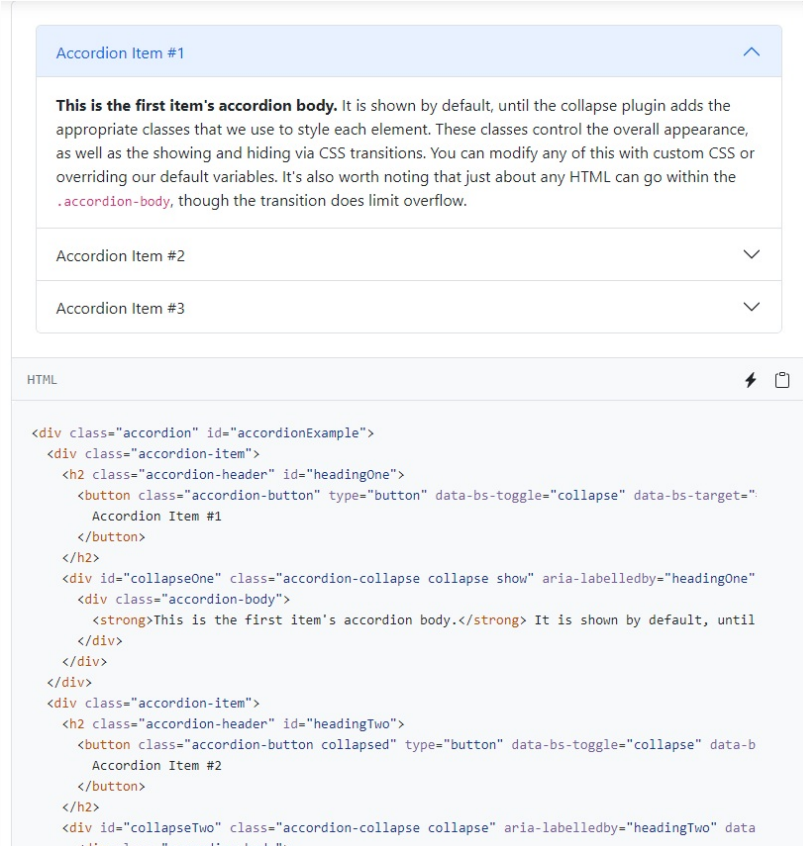
In this lesson in particular, we'll look at what it is and how to set it up so you can begin using it in your own projects.

What is Bootstrap?

What is Bootstrap?

Bootstrap, created by former Twitter employees, is a CSS framework composed of many predefined classes you can add to HTML elements in order to quickly and easily build an elegant user interface (UI). In the documentation, there are even ready-made examples of various UI components which you can copy and paste and then modify with your own content.

All of the components are detailed in the docs which shows an image of the component with the code just below it:



The screenshot displays the Bootstrap documentation for the Accordion Component. It is divided into two main sections: a visual example and the underlying HTML code.

Visual Example: The top section shows a light blue header bar with the text "Accordion Item #1" and a small upward-pointing chevron icon. Below this is a white box containing the text: "This is the first item's accordion body. It is shown by default, until the collapse plugin adds the appropriate classes that we use to style each element. These classes control the overall appearance, as well as the showing and hiding via CSS transitions. You can modify any of this with custom CSS or overriding our default variables. It's also worth noting that just about any HTML can go within the .accordion-body, though the transition does limit overflow." Below this box are two more items, "Accordion Item #2" and "Accordion Item #3", each with a downward-pointing chevron icon.

HTML Code: The bottom section, titled "HTML", shows the code for the accordion. It uses the `collapse` plugin. The first item is expanded by default, indicated by the `show` class on the `collapseOne` element. The code includes `data-bs-toggle="collapse"` and `data-bs-target="#collapseOne"` attributes on the buttons.

```
<div class="accordion" id="accordionExample">
  <div class="accordion-item">
    <h2 class="accordion-header" id="headingOne">
      <button class="accordion-button" type="button" data-bs-toggle="collapse" data-bs-target="#collapseOne" data-bs-expanded="true">
        Accordion Item #1
      </button>
    </h2>
    <div id="collapseOne" class="accordion-collapse collapse show" aria-labelledby="headingOne">
      <div class="accordion-body">
        <strong>This is the first item's accordion body.</strong> It is shown by default, until the collapse plugin adds the appropriate classes that we use to style each element. These classes control the overall appearance, as well as the showing and hiding via CSS transitions. You can modify any of this with custom CSS or overriding our default variables. It's also worth noting that just about any HTML can go within the .accordion-body, though the transition does limit overflow.
      </div>
    </div>
  </div>
  <div class="accordion-item">
    <h2 class="accordion-header" id="headingTwo">
      <button class="accordion-button collapsed" type="button" data-bs-toggle="collapse" data-bs-target="#collapseTwo">
        Accordion Item #2
      </button>
    </h2>
    <div id="collapseTwo" class="accordion-collapse collapse" aria-labelledby="headingTwo" data-bs-show="false">
      <div class="accordion-body">
        ...
      </div>
    </div>
  </div>
  <div class="accordion-item">
    <h2 class="accordion-header" id="headingThree">
      <button class="accordion-button collapsed" type="button" data-bs-toggle="collapse" data-bs-target="#collapseThree">
        Accordion Item #3
      </button>
    </h2>
    <div id="collapseThree" class="accordion-collapse collapse" aria-labelledby="headingThree" data-bs-show="false">
      <div class="accordion-body">
        ...
      </div>
    </div>
  </div>
</div>
```

Pictured above: [Accordion Component](#)

Watch the following two videos for a quick intro to Bootstrap, and an overview of how to add it to a project, then complete the knowledge check.

Bootstrap overview:

[Bootstrap 5 Crash Course Tutorial #1 - Intro & Setup](#) (1:23)

Install and Setup:

[Bootstrap 5 Crash Course Tutorial #1 - Intro & Setup](#) (0:56)

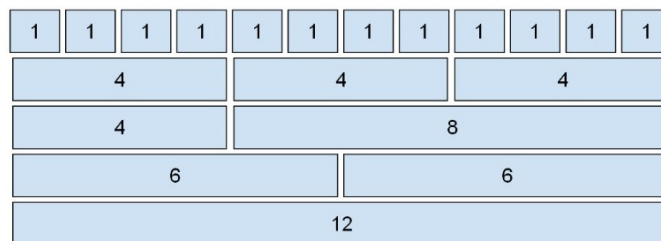
Exercise:

Try setting up Bootstrap yourself

First, visit the following link to Bootstrap's Getting Started page:
[Get started with Bootstrap · Bootstrap v5.2](#)

Copy the CDN links and put them in the appropriate areas of the HTML document in the code editor.

If you've added the CDN links correctly, you should already see a distinct difference in the way things look. That's because the classes included on the HTML elements are all from Bootstrap. If these classes don't make a lot of sense at the moment, don't worry – we'll delve more into them throughout the day today.



bootstrap columns and grid system

Image from article "[Bootstrap 5 Columns and Grid System | Explained](#)"

Example

Let's take a look at Bootstrap's responsive grid system.

Bootstrap organizes its grid into a 12-column structure, meaning the grid container gets divided into 12 hypothetical slices by default with the child elements taking up equal space across those 12 columns. To make more sense of how this works, we will look at an example:

You'll notice that the grid container is defined by adding Bootstrap's **row** class to the section element. Inside that section element are three divs with a class of **col**. See how each of them takes up an equal amount of space?

Try removing one of the divs to see the result:

...both remaining divs still take up an equal amount of space!

Now, add the third div back in and we'll take a look at the baked-in responsiveness. Modify the class attribute for each of the three divs to read as follows:

```
<div class="col-12 col-md-3">...</div>
```

Click 'View Webpage' again, open DevTools, and click the Device mode icon at the top. Try resizing the window and see how the grid changes from horizontal to stacked once the viewport is less than the breakpoint specified by **md** in **col-md-3** (less than or equal to 768px wide).

Responsiveness with two short class names and no media queries... it's a beautiful thing! What these class names are saying is that on all viewport sizes up to the medium breakpoint (**768px**), this div should take up all of the available space (all 12 columns in the grid system), and on every viewport size larger than that it should take up one quarter of the space (4 columns).

Browser Compatibility

A great thing about Bootstrap is that it is pretty much compatible with most browsers. According to the documentation, it “supports the latest, stable releases of all major browsers and platforms”.

	Chrome	Firefox	Safari	Android Browser & WebView
Android	Supported	Supported	—	v6.0+
iOS	Supported	Supported	Supported	—

From Bootstrap [docs](#) - Mobile devices

Desktop browsers

Similarly, the latest versions of most desktop browsers are supported.

	Chrome	Firefox	Microsoft Edge	Opera	Safari
Mac	Supported	Supported	Supported	Supported	Supported
Windows	Supported	Supported	Supported	Supported	—

From Bootstrap [docs](#) - Desktop browsers

For more information, check out the full browser compatibility page in the docs:

[Browsers and devices · Bootstrap v5.2](#)

Advantages and Disadvantages

As with any framework, there are always trade-offs.

One major advantage of using Bootstrap, as mentioned previously, is that you don't have to code common user interface components like cards, navbars, modals, etc., from scratch. Bootstrap has plug-n-play examples where all you need to do is add in your own text content. It also comes with a number of utility classes for things like margin, padding, shadows, position, colors, and many more.

One downside with using Bootstrap, however, is that unless you do some work customizing it (which is possible, and a link will be provided in the Resources section to learn more), the resulting webpage could look very cookie-cutter.

The nice thing is that Bootstrap is not an all-or-nothing framework, meaning you can use as much or as little of it as you want. If you only want to use the Grid system, that's perfectly okay!

KNOWLEDGE CHECK

Answer the following two questions about Bootstrap.

Conclusion & Takeaways

Bootstrap is a great CSS framework that can help us build UIs quickly. The advantage of using Bootstrap is that we don't have to reinvent the wheel to create common UI components. A downside, however, is that if you use Bootstrap on every project (unless you customize the color-theme and some other styles) every web page you make will look the same.

In this lesson, you got some practice setting up Bootstrap in a project by adding the CDN links, then you got a brief overview of the grid system and got to see an example of how the responsive class names work. Then we went over how to change your grid and viewed the outcomes.

Bootstrap: A Frontend Toolkit



>

Goals

By the end of this lesson you will:

- Have a foundational understanding of Bootstrap as a CSS framework
- Be familiar with several of Bootstrap's components and utilities

>

Introduction

Now that you have had an introduction to Bootstrap, we're going to take things a step further and try using many of Bootstrap's UI components, as well as some of its utilities and helper classes (including those pertaining to media like images and video).

>

We'll see how all these components come together to help build (or prototype) a website's user interface relatively quickly.

>

Containers

One of the most common things in modern websites is containers which restrict the overall width of a page's content on larger screens. This helps to make content more easily digestible.

>

A pattern you will often see is a container which on large screens restricts content to roughly 70 to 80% of the viewport width and keeps everything in the center, then on small screens ups this percentage to nearly the entire viewport while still leaving a bit of margin on either side.

>

Let's see an example of Bootstrap's container component.

>

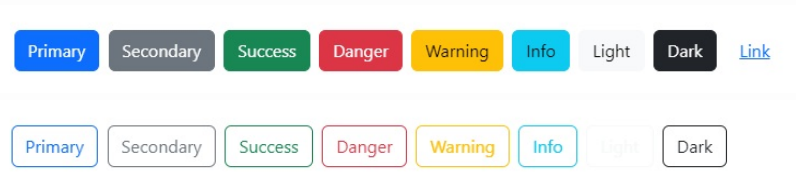
Now, add **container** to the class attribute of the **div** just inside the **body** element to see what the container does. That div tag should now look like `<div class="container">`.

UI Components: Buttons, Cards and Modals, Oh My!

In this section we'll look at several UI components Bootstrap offers. To see the full details about any one of these, check out the [documentation](#).

Buttons

Bootstrap has a variety of button styles to choose from. Here are some samples from the docs:



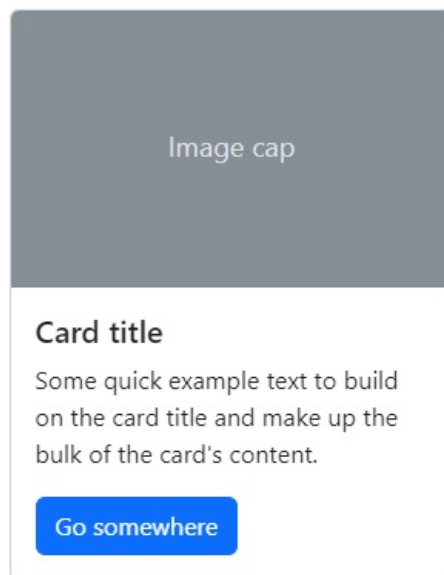
They also come in different size variations, using the classes **btn-sm** for small buttons and **btn-lg** for large buttons.

Normally, you would use the button classes with the HTML button element, but you can also use them with `<a>` elements and `<input>` elements.

>

Cards

Cards are another very common UI component. Below is a screenshot from the docs depicting a basic card:



>

Bootstrap's **card** component, by default, doesn't have a predefined size and

will automatically take up the width of its parent element. There are a number of options available to size cards:

>

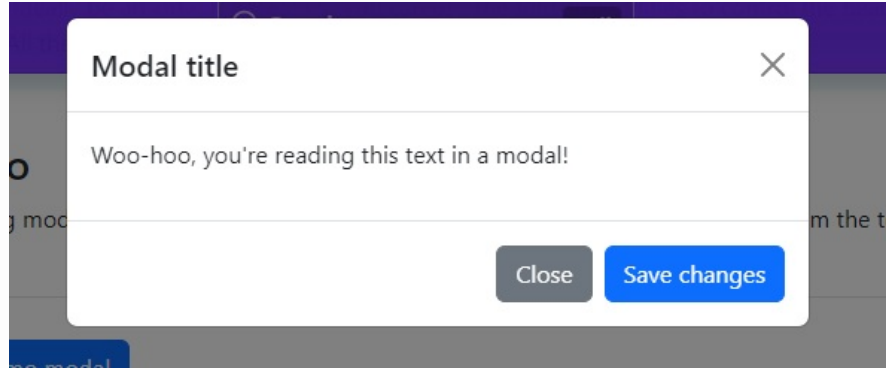
1. You can use a fixed width for the card either in your own stylesheet or using the style attribute

1. You can use the grid system

1. You can use Bootstrap's [size utilities](#)

>

Modals



>

Creating modals from scratch can often be tedious. Fortunately, Bootstrap has you covered.

>

Forms

Adding visually appealing forms with Bootstrap is also a breeze:

>

This example shows form inputs with “floating” labels – another common pattern in modern websites.

>

Breadcrumbs

Breadcrumbs are related to navigation, and offer a nice visual to help users know where they are on your site and the path they took to get there.. You’ve probably seen these before, particularly in online storefronts:

[Home](#) / [Library](#) / Data

>

Images & Video

Bootstrap also comes with utility classes for styling images and video, and also making them responsive. Here are links to the respective pages in the docs that go over these, so you can read more about them:

>

VIDEO EMBEDS

[Ratios · Bootstrap v5.2](#)

>

IMAGES

[Images · Bootstrap v5.2](#)

>

Look at the code examples for all these components taken from the Bootstrap docs in the IDE and preview.

>

Conclusion & Takeaways

Bootstrap boasts tons of components with code examples in the docs that you can basically just insert into your HTML. We have reviewed several of them here in order to help you get started using Bootstrap. For further information, and a guide on how to customize it, head over to the [docs](#).

Bootstrap makes it easy to format. Review the features here, and pay particular attention to any features you see here that you might want to add to your personal page!

Attribution

1. ajay_suresh. (2020, April 18). Toolkit image. FLICKR. Retrieved November 23, 2022, from <https://flickr.com/photos/83136374@N05/49790892077>
2. Otto, M. J. T. (n.d.-a). Bootstrap 5.2 Documentation - Buttons. <https://getbootstrap.com/docs/5.2/components/buttons/>
3. Otto, M. J. T. (n.d.-f). Bootstrap 5.2 Documentation - Outline Buttons. <https://getbootstrap.com/docs/5.2/components/buttons/#outline-button>
4. Otto, M. J. T. (n.d.-b). Bootstrap 5.2 Documentation - Cards. <https://getbootstrap.com/docs/5.2/components/card/>
5. Otto, M. J. T. (n.d.-c). Bootstrap 5.2 Documentation - Floating labels. <https://getbootstrap.com/docs/5.2/forms/floating-labels/>
6. Otto, M. J. T. (n.d.-d). Bootstrap 5.2 Documentation - Modal. <https://getbootstrap.com/docs/5.2/components/modal/>
7. Otto, M. J. T. (n.d.-e). Bootstrap 5.2 Documentation - Navbar. <https://getbootstrap.com/docs/5.2/components/navbar/>
8. Otto, M. J. T. (n.d.-a). Bootstrap 5.2 Documentation - Breadcrumb. <https://getbootstrap.com/docs/5.2/components/breadcrumb/>
9. Otto, M. J. T. (n.d.-e). Bootstrap 5.2 Documentation - Images. <https://getbootstrap.com/docs/5.2/content/images/>
10. Otto, M. J. T. (n.d.-i). Bootstrap 5.2 Documentation - Ratios. <https://getbootstrap.com/docs/5.2/helpers/ratio/>

Project

DevBlog Home Page

Goals

By the end of this project you will:

- Have experience reading documentation in order to implement a solution in code
- Have built a completed UI using a variety of Bootstrap features, the grid system, as well as your own styles

Introduction

If you want to get good at frontend development, as with many things in life, it takes practice!

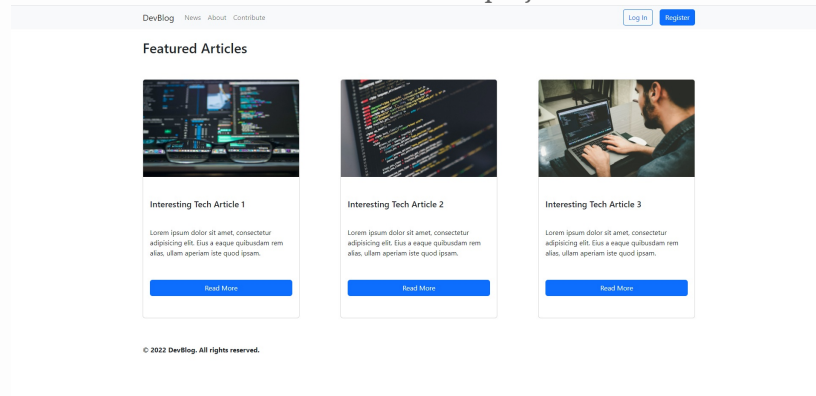


Now that you've spent this week learning about advanced CSS, responsive design, and the Bootstrap framework, it's time to put all that newly-acquired knowledge to use.

This assignment works as a pair-coding exercise. Time permitting, your instructor can arrange teams. One person can act as navigator and one as driver, working together to figure out the documentation and layout formatting.

Instructions

Here's a screenshot of what the finished project should look like:



1. TIP: Take a look at the starter code for the project (some of the HTML will get overwritten with the components from Bootstrap).

- TIP: Use Bootstrap's documentation for more info about implementing the necessary components: **Navbar**, **Button**, **Outline Button**, **Card**, and **Modal**.
Step 1: Use CSS Grid (with grid-template-areas) to achieve the overall layout of the page.
Step 2: Modify the **nav** element to use the basic Navbar from Bootstrap.
Step 3: Modify the **navbar-brand** text to be: **DevBlog**.
Step 4: Replace the **li** elements in the **ul** with the class **navbar-nav** to the following:

```
<li class="nav-item">
  <a class="nav-link" aria-current="page" href="#">News</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="#">About</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="#">Contribute</a>
</li>
```

Step 5: There should be two Bootstrap button elements in the nav; the first should have the text “Log In”, the second “Register”.
Modify the class attribute of the “Log In” button to be:

```
class="btn btn-outline-primary mx-3"
```

Make sure the attributes of the “Register” button are:

```
type="button" class="btn btn-primary" data-bs-toggle="modal"
data-bs-target="#exampleModal"
```

Step 6: Add the markup for the Modal element from Bootstrap right below the “Register” button.

Step 7: Modify the **div** with the class **modal-dialog** so that the **class** attribute reads as follows:

```
class="modal-dialog modal-dialog-centered"
```

Step 8: Add the following inside the **div** with the class **modal-body**:

```
<div class="form-floating mb-3">
  <input
    type="text"
    class="form-control"
    id="floatingInput"
    placeholder="Full name"
  >
  <label for="floatingInput">Full name</label>
</div>
<div class="form-floating mb-3">
  <input
    type="email"
    class="form-control"
    id="floatingInput"
    placeholder="name@example.com"
  >
  <label for="floatingInput">Email address</label>
</div>
```

Step 9: Modify the **button** elements in the **modal-footer**, so the second one has the text “Subscribe”, and the class attribute:
`data-bs-dismiss="modal"`

Step 10: Add the **container** class to the **main** element.

Step 11: Add the classes **mt-4** and **mb-3** to the **h2** element containing the text “Featured Articles”.

Step 12: Add three of the basic example **Card** from Bootstrap, then change the **src** attribute on the **img** element for each one (Order: **image-1.jpg**, **image-2.jpg**, and **image-3.jpg**)

Step 13: In **style.css**, add grid-template-areas to the **.wrapper** selector:

```
grid-template-areas:
  "header header header"
  "main main main"
  "main main main"
  "footer footer footer"
;
```

Step 14: Add the appropriate **grid-area** to the **.header**, **.main**, and **.footer** selectors.

Step 15: Add the following right after the **.main** selector's styles:

```
.featured-articles {
  display: flex;
  flex-direction: column;
  justify-content: space-around;
}

.article-cards {
  min-height: calc(100vh - 300px);
  display: flex;
  align-items: center;
  justify-content: space-between;
  gap: 0.5rem;
}
```

Step 16: Add selectors for **.card**, **.card-body**, and **#img1**, **#img2**, **#img3** to apply some additional styles:

```
.card {
  min-height: 35rem;
  max-width: 23rem;
}

.card-body {
  display: flex;
  flex-direction: column;
  justify-content: space-evenly;
}

#img1, #img2, #img3 {
  height: 228.42px;
}
```

Step 17: Lastly, add a **.footer__content** selector with the following styles:

>

```
.footer__content {  
  width: 100%;  
  padding-block: 1.5rem;  
  font-weight: bold;  
}
```

And that's a wrap! On the next page, we will take a quick look at a finished example of the code!

Example and Conclusion

Finished Example

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>DevBlog - Home</title>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/boot
strap.min.css">
  <link rel="stylesheet" href="./style.css">
</head>
<body class="wrapper">
  <header class="header">
    <nav class="navbar navbar-expand-lg bg-light">
      <div class="container container-fluid">
        <a class="navbar-brand" href=".">DevBlog</a>

        <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse"
id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link" aria-current="page"
href="#">News</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">About</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">Contribute</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
  </header>
</body>
</html>
```

```

        </li>
    </ul>
    <button type="button" class="btn btn-outline-primary
mx-3">
        Log In
    </button>
    <button type="button" class="btn btn-primary" data-bs-
toggle="modal" data-bs-target="#exampleModal">
        Register
    </button>

    <div class="modal fade" id="exampleModal"
tabindex="-1" aria-labelledby="exampleModallabel" aria-
hidden="true">
        <div class="modal-dialog modal-dialog-centered">
            <div class="modal-content">
                <div class="modal-header">
                    <h1 class="modal-title fs-5"
id="exampleModallabel">Subscribe to DevBlog</h1>
                    <button type="button" class="btn-close" data-
bs-dismiss="modal" aria-label="Close"></button>
                </div>
                <div class="modal-body">
                    <div class="form-floating mb-3">
                        <input
                            type="text"
                            class="form-control"
                            id="floatingInput"
                            placeholder="Full name"
                        >
                        <label for="floatingInput">Full name</label>
                    </div>
                    <div class="form-floating mb-3">
                        <input
                            type="email"
                            class="form-control"
                            id="floatingInput"
                            placeholder="name@example.com"
                        >
                        <label for="floatingInput">Email
address</label>
                    </div>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-
secondary" data-bs-dismiss="modal">Close</button>
                    <button type="button" class="btn btn-primary"
data-bs-dismiss="modal">Subscribe</button>
                </div>
            </div>
        </div>
    </div>

```

```

        </div>
    </div>
</div>
</div>
</div>
</nav>
</header>

<main class="main container">
    <section class="featured-articles">
        <h2 class="mt-4 mb-3">Featured Articles</h2>
        <div class="article-cards">
            
            <div class="card-body">
                <h5 class="card-title">Interesting Tech Article
1</Article></h5>
                <p class="card-text">Lorem ipsum dolor sit amet,
consectetur adipisicing elit. Eius a eaque quibusdam rem alias,
ullam aperiam iste quod ipsam.</p>
                <a href="#" class="btn btn-primary">Read More</a>
            </div>
        </div>
        <div class="card" >
            
            <div class="card-body">
                <h5 class="card-title">Interesting Tech Article
2</h5>
                <p class="card-text">Lorem ipsum dolor sit amet,
consectetur adipisicing elit. Eius a eaque quibusdam rem alias,
ullam aperiam iste quod ipsam.</p>
                <a href="#" class="btn btn-primary">Read More</a>
            </div>
        </div>
        <div class="card">
            
            <div class="card-body">
                <h5 class="card-title">Interesting Tech Article
3</h5>
                <p class="card-text">Lorem ipsum dolor sit amet,
consectetur adipisicing elit. Eius a eaque quibusdam rem alias,
ullam aperiam iste quod ipsam.</p>
                <a href="#" class="btn btn-primary">Read More</a>
            </div>
        </div>
    </div>

```

```

        </div>
    </section>
</main>

<footer class="footer container">
    <div class="footer__content">
        &copy; 2022 DevBlog. All rights reserved.
    </div>
</footer>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootst
rap.bundle.min.js"></script>
</body>
</html>

```

CSS:

```

.wrapper {
    display: grid;
    grid-template-areas:
        "header header header"
        "main main main"
        "main main main"
        "footer footer footer"
    ;
}

.header {
    grid-area: header;
}

.main {
    grid-area: main;
}

.featured-articles {
    display: flex;
    flex-direction: column;
    justify-content: space-around;
}

.article-cards {
    min-height: calc(100vh - 300px);
    display: flex;
    align-items: center;
    justify-content: space-between;
}

```

```

    gap: 0.5rem;
  }

  .card {
    min-height: 35rem;
    max-width: 23rem;
  }

  .card-body {
    display: flex;
    flex-direction: column;
    justify-content: space-evenly;
  }

  #img1, #img2, #img3 {
    height: 228.42px;
  }

  .footer {
    grid-area: footer;
  }

  .footer__content {
    width: 100%;
    padding-block: 1.5rem;
    font-weight: bold;
  }

```

Bear in mind, this is only an example! If your code looks slightly different from this, no worries! As long as your finished project looks pretty close to the screenshot at the beginning of the instructions, you are good to go!

Attribution

NOTE: All the Bootstrap components are taken from the docs, with some modifications such as custom text, images from Unsplash, or other changes. Links to each page will be given below.

1. (2019, August 28). Practice GIF. GIPHY. <https://giphy.com/gifs/hockey-skateboard-balance-Y2c1ZjXVHRdSr1YojF>
2. Otto, M. J. T. (n.d.-a). Bootstrap 5.2 Documentation - Buttons. <https://getbootstrap.com/docs/5.2/components/buttons/>
3. Otto, M. J. T. (n.d.-f). Bootstrap 5.2 Documentation - Outline Buttons. <https://getbootstrap.com/docs/5.2/components/buttons/#outline-button>
4. Otto, M. J. T. (n.d.-b). Bootstrap 5.2 Documentation - Cards. <https://getbootstrap.com/docs/5.2/components/card/>
5. Otto, M. J. T. (n.d.-c). Bootstrap 5.2 Documentation - Floating labels. <https://getbootstrap.com/docs/5.2/forms/floating-labels/>

6. Otto, M. J. T. (n.d.-d). Bootstrap 5.2 Documentation - Modal.
<https://getbootstrap.com/docs/5.2/components/modal/>
7. Otto, M. J. T. (n.d.-e). Bootstrap 5.2 Documentation - Navbar.
<https://getbootstrap.com/docs/5.2/components/navbar/>

>

Resources

To get more practice building designs, a really tremendous resource is [Frontend Mentor](#). You can sign up for free and be on your way to frontend mastery through solving their challenges!

Project: Your “About Me” Page

Week 4 Project

For this assignment, your task is to add advanced features to your “About Me” page. You should notice that your code is looking increasingly professional, and you are getting closer to having a potential employer look at what you’ve created to showcase your skills.

For this project, you will be adding advanced css features to your page. You should make frequent use of pseudo-class and pseudo-element items, and include at least two or three of the following in your design:

- a sass .scss sheet to produce your css file
- media queries
- css animations
- advanced grid
- flexbox
- bootstrap

When you’ve updated your page, share the code here:

HTML:

CSS or sass:

What a great week of writing code!

“On two occasions, I have been asked [by members of Parliament], ‘Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?’ I am not able to rightly apprehend the kind of confusion of ideas that could provoke such a question.”

— Charles Babbage