Learn to Code

HTML, CSS and Bootstrap

Workbook 1-b

Version 5.0 Y

.

authors:
Dana L. Wyatt, PhD and Eric Schwartz

Table of Contents

Module 1

HTML: A Review

Section 1–1

HTML Syntax, Structure and Tags

HTML

- **HTML (Hyper Text Markup Language) is the markup language that provides us a structured way to create web pages**

- **HTML tags are a combination of keywords and symbols that the browser can identify and use to make sense of the content**

- **The HTML is interpreted by a browser to tell it how things should be represented visually**

Example HTML tags (demo1.html)

```
Why I like pizza!

Pizza has always had a special place in my heart. From pizza day in the school cafeteria to pizza parties at friends' houses, there was nothing more exciting than opening that cardboard to reveal eight delicious slices.
```

  - When the above example is rendered, the browser smashes it together

```
Why I like pizza! Pizza has always had a special place in my heart. From pizza day in the school cafeteria to pizza parties at friends' houses, there was nothing more exciting than opening that cardboard to reveal eight delicious slices.
```

- **Why does the browser do this?**
  - The browser's doesn't apply any preconceived structure to the text
    * It doesn't know that the first line is supposed to be a title
    * It doesn't know that the second line should be a paragraph

- **HTML tags allow us to explain this structure to the browser**

Example with HTML tags (demo2.html)

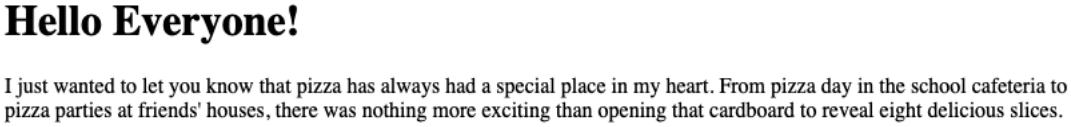
```
<h1>Hello Everyone!</h1>

<p>

  I just wanted to let you know that pizza has always had a special place in my heart. From pizza day in the school cafeteria to pizza parties at friends' houses, there was nothing more exciting than opening that cardboard to reveal eight delicious slices.

</p>
```

- The output looks closer to what we



  expected

HTML Tag Syntax

- **HTML tags are special keywords surrounded by angle brackets (< and > symbols)**

```
<tag>Hello World!</tag>
```

  - The tag is 'tag' (not an HTML tag that the browser would understand - it is just used as an example)
  - The content is the words "Hello Everyone!"

- **The `<p>` tag is a real HTML tag and tells the browser to display the contents in a "paragraph"**

```
<p>Hello World!</p>
```

- **Opening tags require 3 things**
  - An opening angle bracket `<`
  - The keyword that represents the tag you want to use
  - A closing angle bracket `>`

- **The closing tag is the same as the opening tag but also includes a forward slash `/` before the keyword**

- **"Most" HTML elements require an opening and closing tag**
  - NOTE: The term "HTML element" refers to the opening and closing tags and the content they wrap

Self-Closing HTML Tags

- **Some tags are self-closing**
  - They are also called empty elements or void elements
- **These tags don't have *content* between the opening and closing tags so the closing tag is omitted**
- **Void tags often require additional information from their *attributes* (more on that later)**

```
<br>
<hr>
<img src="http://fillmurray.com/200/300" alt="Ol' Bill">
<input type="text" id="username" name="username">
<link rel="stylesheet" href="style.css">
<meta charset="utf-8">
```

- **The void elements you'll use most often are `<br>`, `<hr>`, `<img>`, `<input>`, `<link>`, and `<meta>`**

HTML Attributes

- **Tags use *attributes* to add additional information**
  - We use key/value pairs in the opening tag

```
<pizza bake="well done" size="large">
    <topping>pineapple</topping>
    <topping>ham</topping>
    <topping extra="true">cheese</topping>
</pizza>
```

- **All attributes follow the same rules:**
  - Every attribute has one key (attribute name) and one value (attribute value)
  - Keys aren't quoted and values are wrapped in quotes `""`
  - An `=` sign separates the key and value
  - Attributes are separated by a space
  - Don't duplicate attributes on the same element

Element Specific Attributes

- **Some attributes can only be applied to one or a few HTML elements**
  - They are called *element-specific attributes*
  - Most attributes are element-specific

- **`<a>` tags require an `href` attribute**
  - The `href` attribute tells the browser where to navigate to when the link is clicked

Using attributes example

```
<a href="https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes">Click here for info
on attributes</a>
```

- **The `href` attribute is only used with the `<a>` tag**
  - It is *element-specific*

Global Attributes

- **Some attributes can be applied to *any* valid HTML tag**

- **The two most common global attributes are `class` and `id`**
  - `class` associates the HTML element with some CSS styling information
  - `id` provides a unique ID for the HTML element so that you can find it to apply CSS styling or find it to work with it from JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>List App</title>
  </head>
  <body class="wrapper" id="home">
    <header class="page_section" id="intro">
      <h1>This is a list app</h1>
      <p>Enjoy the list app and all its listy goodness.</p>
    </header>
    <div class="page_section" id="lists">
      <h2 class="section_title">Behold, a list of my fears</h2>
      <ul>
        <li>Bacon</li>
        <li>Umbrellas</li>
        <li>Space shuttle</li>
        <li>Small spaces between tall buildings</li>
        <li>A bad haircut</li>
        <li>Heights</li>
      </ul>
    </div>
    <footer class="page_section" id="related_content">
```

```
      <h2 class="section_title">Conclusion!</h2>
      <p>Just kidding you. I don't fear bacon... I love bacon.</p>
    </footer>
  </body>
</html>
```

Basic HTML Page Structure

- **Let's look at the structure of an HTML file**
  - What tags that are needed to have a well-formed page for display on the web?

HTML basic example

```
<!DOCTYPE html>
  <html>
  <head>
      <title></title>
  </head>
  <body>


  </body>
</html>
```

- **Let's take a closer look at these tags**
  - `<!DOCTYPE html>` - tells the browser we will be using HTML5 Doctype
    - The Doctype is special in that it's not really considered a tag but a declaration
    - It should first line in your HTML file.
  - `<html>` - represents the root (top-level element) of an HTML document
    - All other elements must be descendants of this element
    - All other tags other than the Doctype appear between the opening and closing `<html>` tag
  - `<head>` - contains descriptive information about your webpage like the title `<title>`
    - Would include things like `<meta>` tags that describe our site's content and `<link>` tags that allow us to include CSS files.

    * `<title>` - shown in a browser's title bar or on the webpage's tab
  - `<body>` - represents the content of an HTML document.

- **You will use the code above, or a version of it, as the starter for almost all web pages you build**

Section 1–2

Common HTML tags

Paragraphs `<p>`

- **The HTML `<p>` tag represents a paragraph of text**

Paragraph `<p>` example

```
    <p>

        Tart gingerbread ice cream sweet roll wafer chocolate bar bonbon marzipan pie.
Gummi bears dragée halvah toffee wafer jelly beans gummies. Lemon drops sesame snaps
caramels candy wafer jelly-o. Pie toffee sweet roll sugar plum gingerbread sugar plum.

    </p>
```

- **When you are laying out your web site, but waiting for your marketing department or user to provide you the text, you can use placeholder text**
  - There are several famous web sites that generates text used by typesetters.
    * You can specify how many words or paragraphs you need and it will generate them using letter combinations that, from afar, look like English text.
    * You use them to design placement until your content creator generates the actual text.

    `https://www.lipsum.com/`

    `https://baconipsum.com/`

    `https://www.shopify.com/partners/blog/79940998-15-funny-lorem-ipsum-generators-to-shake-up-your-design-mockups`

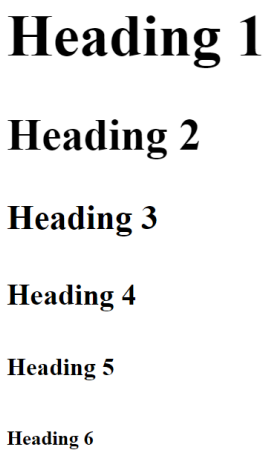
Headings <h1> – <h6>

- The HTML <h1> – <h6> tags represent six levels of section headings
  - <h1> is the highest section level and <h6> is the lowest

Headings <h1> – <h6> example

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```

- The code above would render as:



Lists `<ul>` (unordered) or `<ol>` (ordered)

- **HTML has the ability to display lists**
  - `<ol>` represents an ordered (numbered) list
  - `<ul>` represents an unordered (bulleted) list
- **Items in the list (list items) are represented by the `<li>` tag**

```
<ul>
    <li>Milk</li>
    <li>Eggs</li>
    <li>Frozen Pizza</li>
    <li>Dog Food</li>
</ul>
```

- **The output from above resembles:**

- Milk
- Eggs
- Frozen Pizza
- Dog Food

Hyperlinks `<a>`

- **Hyperlinks are represented by the anchor `<a>` tag**
  - When clicked, a hyperlink can take the user to other web pages, files, locations within the same page, email addresses, or any other URL

- **The text or element between the opening and closing `<a>` tag is what becomes the clickable link**

- **Element-specific attributes for an `<a>` tag include:**
  - **`href`** - The URL or destination hyperlink points to.
  - **`target`** - Specifies where to display the linked URL. Below are the most common values used.
    * **`_self`:** Load the URL into the same browser window as the current one. ***This is the default behavior if the target attribute is not used.***
    * **`_blank`:** Load the URL into a new browser window or tab based on how the user's browser is configured.

Simple anchor example

```
<!-- external link opens in the same window -->
<a href="http://www.google.com">google.com</a>
```

Opens another page on the same site example

```
<!-- link to local another file on the same site -->
<a href="my_html_file.html">view my_html_file</a>
```

Opens another page by clicking on an image example

```
<!-- External link that opens in a new window/tab using an image instead of text -->
<a href="https://developer.mozilla.org/en-US/" target="_blank">
  <img src="https://mdn.mozillademos.org/files/6851/mdn_logo.png"/>
</a>
```

Opens in a new window example

```
<!-- external hyperlink opens in a new window -->
<a href="http://www.google.com" target="_blank">google.com</a>
```

Opens in the default email client example

```
<!-- Email link that opens users default email client -->
<a href="mailto:user@example.com">Email User</a>
```

Links to another element on the page example

```
<a id="table_of_contents"></a>
...
!-- links to element on this page with id="page-section" -->
<a href="#table_of_contents">Go to Table of Contents</a>
```

Section 1–3

HTML - Images, Tables, Block & Inline Elements, and Validation

Images

- **Images are placed in our HTML document using the `<img>` tag**
  - The image tag is a ***self-closing tag*** and does not require a matching closing tag

- **There several attributes for an `<img>` tag, including:**
  - **`src`** - The image URL
    - This attribute is mandatory for the `<img>` element
  - **`alt`** - This attribute defines the description of the image
    - The browser will display this text if the image URL is wrong, the image type is not supported, or if the image is not yet downloaded
    - For accessibility purposes, you almost always provide this
  - **`height`** - The height of the image in pixels
  - **`width`** - The width of the image in pixels
    - Note: there are other ways to specify height and width (percent? auto?) that we will see in a bit

Basic `<img>` example

```
<!-- image with alt take providing a description -->
<img src="some_image.png" alt="MDN">
```

- **What happens if you don't specify a height or width?**
  - The HTML would be rendered when downloaded
  - Then, there would be a *layout shift* when the image is downloaded, the space needed to display is calculated, and the image is rendered

- **If you specify the height and width, the browser can calculate the image's position as the HTML is rendered**
  - This avoids the layout shift when the image is downloaded

```
<!-- image with width and height set -->
<img src="some_image.png" width="200" height="200">
```

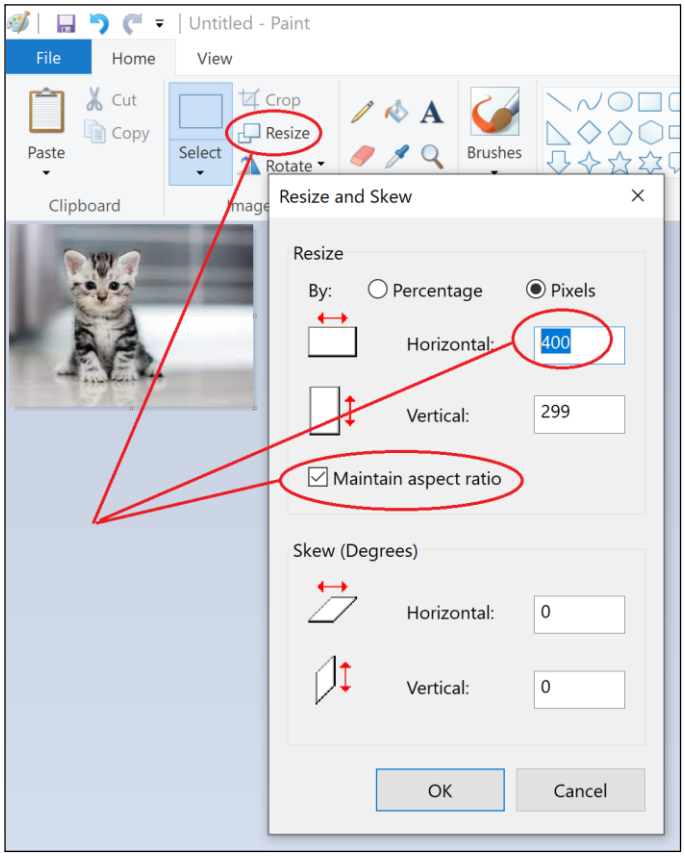
- **If you only specify the height OR width for an `<img>` element, most browsers seem to keep the proportions of the image**

- **You can also use an image as a link**

```
<!-- Image used as a link -->
<a href="https://developer.mozilla.org/en-US/" target="_blank">
  <img src="https://mdn.mozillademos.org/files/6851/mdn_logo.png"/>
</a>
```

Tricks for Finding and Sizing Images

- **You can use this special Google link to search images**
  `https://www.google.com/advanced_image_search`



- **You can use MS Paint to resize images**
  - Open the image file
  - Select Resize
  - Make sure "Maintain aspect ratio" is checked
  - Specify pixels and a horizontal width (or vertical height)
  - Click OK
  - Save your image file

Tables

- **The `<table>` element displays tabular data**
  - It allow you to create rows and columns to present data to the user

- **Tags to create rows and columns can be nested tags in `<table>` tag**
  - `<tr>` - creates a row
  - `<td>` - creates a column within a row
  - `<th>` - creates a special header column "header" that appears bolded
    * It should only be used in the first row

Basic `<table>` example

```
<table>
  <tr>
    <th>First name</th>
    <th>Last name</th>
  </tr>
  <tr>
    <td>John</td>
    <td>Doe</td>
  </tr>
  <tr>
    <td>Jane</td>
    <td>Doe</td>
  </tr>
</table>
```

- **This would render as:**

First name	Last name
John	Doe
Jane	Doe

Section 1–4

Block and Inline Elements

Divs

- **The `<div>` element is a generic container for content flow**
  - It is mainly used to group elements for styling and positioning

- **The `<div>` is a block-level element**
  - Browsers typically display the block-level element with a newline both before and after the element

Basic `<div>` example

```
<div>
  <p>Any kind of content here. Such as
  p or table tags. You name it!</p>
</div>
```

- **When we start adding more styling, you will see more advantage in using divs**
  - The code below uses inline styles using the `style` attribute
  - A better practice is to use CSS (which we will learn review)

Multiple `<div>` tags example

```
<div style="border: black 2px solid; color: blue; margin: 5px">

  <h1>HTML</h1>

  <p>HTML (HyperText Markup Language) is the standard markup language for documents
designed to be displayed in a web browser.  Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Etiam at mattis lacus. Curabitur non lobortis odio. Aenean dapibus
vulputate pulvinar. Duis ut.

  </p>

</div>

<div style="border: black 2px solid; color: tomato; margin: 5px">

  <h2>CSS</h2>

  <p>CSS (Cascading Style Sheets) is the language used to style an HTML document. CSS
describes how HTML elements should be displayed.  Lorem ipsum dolor sit amet, consectetur
adipiscing elit. In a orci eu lectus pharetra bibendum. Etiam id arcu tincidunt velit
porttitor consequat.

  </p>

</div>

<div style="border: black 2px solid; margin: 5px">

  <h1>JavaScript</h1>

  <p>JavaScript is the programming language that allows you to add dynamic behavior to a
web site.  For example, when the user enters their email address into a form and clicks a
button, JavaScript code can send the email to a server to look up an order status, then
JavaScript can generate HTML to display the returned order and its status.

  </p>

</div>
```

- **These three divs will be rendered as:**

HTML

HTML (HyperText Markup Language) is the standard markup language for documents designed to be displayed in a web browser. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam at mattis lacus. Curabitur non lobortis odio. Aenean dapibus vulputate pulvinar. Duis ut.

CSS

CSS (Cascading Style Sheets) is the language used to style an HTML document. CSS describes how HTML elements should be displayed. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In a orci eu lectus pharetra bibendum. Etiam id arcu tincidunt velit porttitor consequat.

JavaScript

JavaScript is the programming language that allows you to add dynamic behavior to a web site. For example, when the user enters their email address into a for and clicks a button, JavaScript code can send the email to a server to look up an order status, the JavaScript can generate HTML to display the order status.

Spans

- **The `<span>` element is a generic container for content flow**
  - It is mainly used for styling purposes and to provide the ability for JavaScript to access inline text

- **In some ways, it is similar to the `<div>`, except the `<span>` is an *inline* element**

- **An inline element occupies only the space bounded by the tags that define the inline element**
  - You can wrap it around a text and it won't add new lines above and below the span

Basic `<span>` example

```
  <p>Any kind of <span>content</span> here.</p>
```

Better `<span>` example

```
<script>
   let orderNo = 12345;  // the order no comes from somewhere else
   document.getElementById("orderNo").innerHTML = orderNo;
</script>
<!-- later in the page -->
<p>Your order number is <span id="orderNo"></span> here.</p>
```

Inline vs Block Level Elements

- **Every HTML element has a default display value:** *block* **or** *inline*

- **A block-level element:**
  - always starts on a new line
  - takes up the available full width
  - has a top and a bottom margin

<address>	<article>	<aside>	<blockquote>	<canvas>	<dd>	
<dl>	<dt>	<fieldset>	<figcaption>	<figure>	<footer>	<form>
<h1>-<h6>	<header>	<hr>		<main>	<nav>	<noscript>
	<p>	<pre>	<section>	<table>	<tfoot>	
<video>						

- **An inline-level element:**
  - does not start on a new line
  - takes up the only the width that the content needs

<a>	<abbr>	<acronym>	<b>	<bdo>	<big>	
<button>	<cite>	<code>	<dfn>		<i>	<img>
<input>	<kbd>	<label>	<map>	<object>	<output>	<q>
<samp>	<script>	<select>	<small>			<sub>
<sup>	<textarea>	<time>	<tt>	<var>		

- **Further reading on block-level elements:**

  `https://developer.mozilla.org/en-US/docs/Web/HTML/Block-level_elements`

- **Further reading on inline tags:**

  `https://developer.mozilla.org/en-US/docs/Web/HTML/Inline_elements`

Finding Errors in HTML Syntax

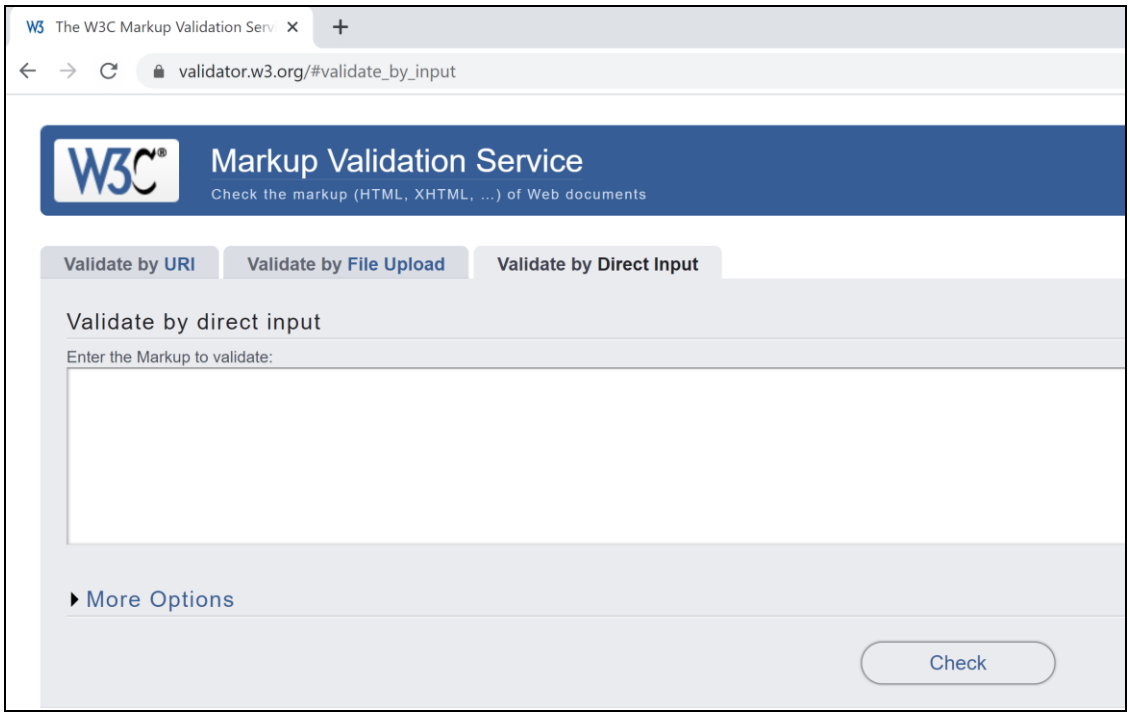
- **HTML can be notoriously hard to debug**
  - Some browsers (including Chrome) will try to "fix" broken HTML when a page is loaded
    - For example, Chrome may try to find a common-sense place to close the tag.
    - However, it might be right or it might be wrong!
  - A page that seems to load correctly in Chrome might not load correctly in an older browser

- **The W3C Markup Validation Service is a free tool that "validates" HTML**
  - You can specify a URL, upload the HTML file, or directly pasting the HTML into an input field
  - If the submitted HTML isn't valid, it will display a list of problems that you can address

- **You can find the W3C Markup Validation Service at :** `https://validator.w3.org/`
  - Go there now and add a bookmark to the site in your browser!

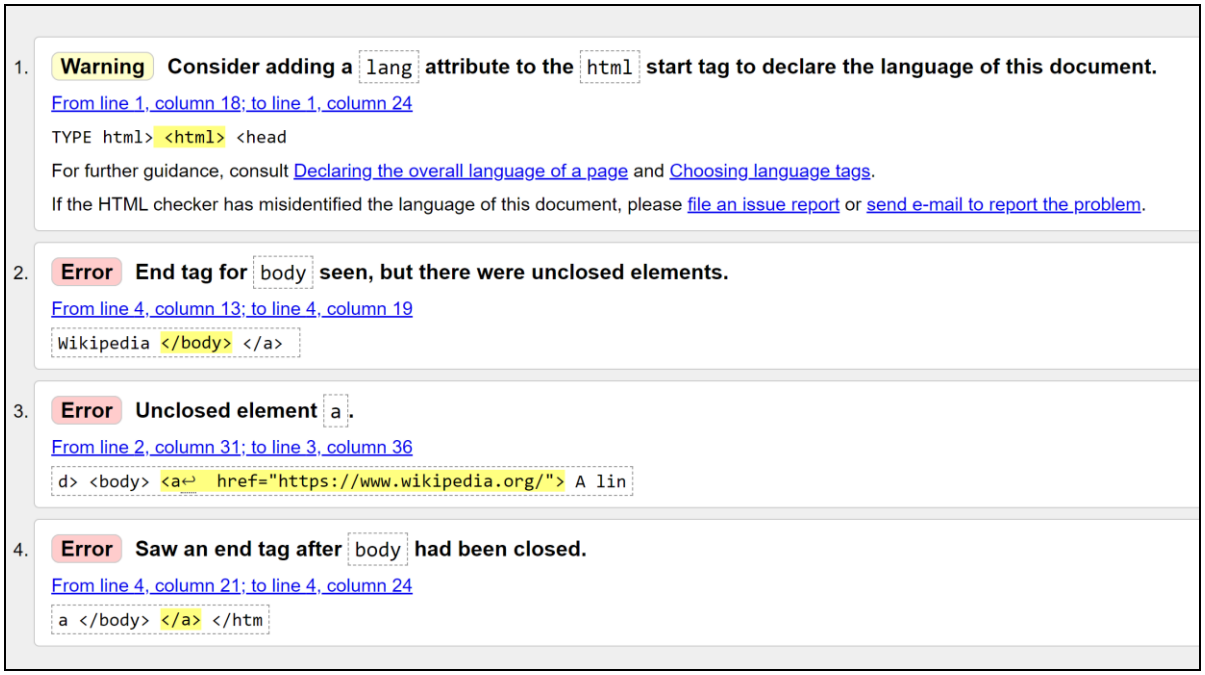
W3C Markup Validation Service

- **Select the tab that describes how you will submit the HTML you want to validate**



  - You can test this messy, invalid HTML

```
<!DOCTYPE html> <html> <head> <title>My first
HTML</title> </head> <body> <a
href="https://www.wikipedia.org/"> A link to
Wikipedia </body> </a> </html>
```



- **Once you know the errors, you can fix them and re-validate**
  - Keep at it until all of the errors are fixed!

- **If you indent your code well, you might have found the errors without the validator!**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>My first HTML</title>
  </head>
  <body>
    <a href="https://www.wikipedia.org/">A link to Wikipedia</a>
  </body>
</html>
```

Troubleshooting Tips

- **When using an HTML tag you aren't familiar with, make sure to check out documentation from a trusted website**
  - Mozilla Developer Network's HTML Element Reference has a list of every HTML element
  - W3schools is also a good place to go look
- **Double-check your HTML for typos**
  - Forgetting a closing tag and not correctly using triangle brackets and forward-slashes are just a few of the subtle typos you'll make from time to time
- **Nested HTML elements can end up closing and opening in odd places, causing a parent element to close itself inside of a child element, and vice-versa**
  - Keep an eye out for zig-zagged nesting
- **Remember to use the W3C Validator tool**
- **When you are really stumped, Google the question and look at the answers from `stackoverflow.com`**
  - Sometimes, there is more than one way to solve a problem
  - When there is more than one answer, users vote on the one that makes the best sense

Section 1–5

HTML - Semantic Elements

Semantic Elements

- **Semantic HTML makes web pages more descriptive when it comes to their content**
  - Plus, the semantic HTML5 tags help browsers and search engines interpret the content

- **Prior to HTML5, we used divs to group all content and gave them ids to help us understand their purpose**

Example: div vs header

```
<!-- using a div with an id of header-->
<div id="header">...</div>


<!-- using the Semantic HTML5 tag <header>-->
<header>...</header>
```

Common Semantic Tags for Page Layout: `<header>`, `<main>`, and `<footer>`

- **Common semantic tags you may see for HTML page layouts include `<header>`, `<main>`, and `<footer>`**

- **These tags are considered semantic because they are used to represent different sections on an HTML page**
  - They are more descriptive to search engines and browsers than div with `id` attributes
  - They make it easier for us as developers and for search engines to identify the sections of the page

Page Design: The Non-Semantic Way

```
<html>
    <head>
        <title>My Travel Site</title>
    </head>
    <body>
        <div id="header">
            My awesome logo and navigation would be here
        </div>
        <div id="nav">
            <a href="index.html">Home</a>
            <a href="contact.html">Contact Me</a>
        </div>
        <div id="main">
            My awesome content would be here
        </div>
        <div id="footer">
            My awesome footer information would be here
        </div>
    </body>
</html>
```

Page Design: The Semantic Way

```
<html>
    <head>
        <title>My Travel Site</title>
    </head>
    <body>
        <header>
            My awesome logo and navigation would be here
        </header>
        <nav>
            <a href="index.html">Home</a>
            <a href="contact.html">Contact Me</a>
        </nav>
        <main>
            My awesome content would be here
        </main>
        <footer>
            My awesome footer information would be here
        </footer>
    </body>
</html>
```

The `<nav>` tag

- The `<nav>` tag is used to wrap a site's navigation
- It is most common in the `header` and `footer` sections of a site and on its own
  - But it can be anywhere navigation elements make sense

Example of the `<nav>` within the `<header>` of our site

```
<header>
    <img src="logo.png" alt="Travel Site Logo"/>
    <nav>
        <a href="index.html">Home</a>
        <a href="contact.html">Contact Me</a>
    </nav>
</header>
```

The <article> tag

- **The `<article>` tag is typically used for things like a forum post, a magazine or newspaper article, or a blog entry**
  - They are independent items of content that stand on their own

Example of the `<article>` element

```
<main>
    <!-- possibly more here -->
    <article>
        <h1>Fun Facts about Spain, Italy, Aruba, and Alaska</h1>
        <table>
            <tr>
                <th>Destination</th>
                <th>Capital</th>
                <th>Official Language</th>
            </tr>
            <tr>
                <td>Spain</td>
                <td>Madrid</td>
                <td>Spanish</td>
            </tr>
            <tr>
                <td>Italy</td>
                <td>Rome</td>
                <td>Italian</td>
            </tr>
            <tr>
                <td>Aruba</td>
                <td>Oranjestad</td>
                <td>Dutch</td>
            </tr>
```

```
            <tr>
                <td>Alaska</td>
                <td>Juneau</td>
                <td>English</td>
            </tr>
        </table>
    </article>
    <!-- possibly more here -->
</main>
```

The <section> tag

- **One of the most common content indicators is the `<section>` tag**
  - Sections represent a generic section of your page and typically have a heading

- **Tags for content would typically appear in the `<main>` section of your page**

Example of the `<section>` element

```
<main>
    <section>
        <h2>Places I would like to visit</h2>
        <ul>
            <li>Spain</li>
            <li>Italy</li>
            <li>Aruba</li>
            <li>Alaska</li>
        </ul>
    </section>
</main>
```

The <aside> tag

- **The `<aside>` tag represents information that is related to other content**
  - Think of it as extra information or a call-out to related information

Example of the `<section>` element

```
<main>
    <section>
        <h2>Places I would like to visit</h2>
        <ul>
            <li>Spain</li>
            <li>Italy</li>
            <li>Aruba</li>
            <li>Alaska</li>
        </ul>
        <aside>
            <p>I am afraid fo bears so Alaska is last on my list</p>
        </aside>
    </section>
</main>
```

The <figure> tag

- The `<figure>` tag represents items like photos, charts, and other graphic content

- Your logo and images *related to the content of a page* can be wrapped in the `<figure>` tag
  - Images like ads should not be wrapped in the `<figure>` tag

Example of the `<figure>` element being used to wrap our logo

```
<header>
    <figure>
        <img src="logo.png" alt="Travel Site Logo"/>
    </figure>
    <nav>
        <a href="index.html">Home</a>
        <a href="contact.html">Contact Me</a>
    </nav>
</header>
```

Other HTML Elements with Meaning

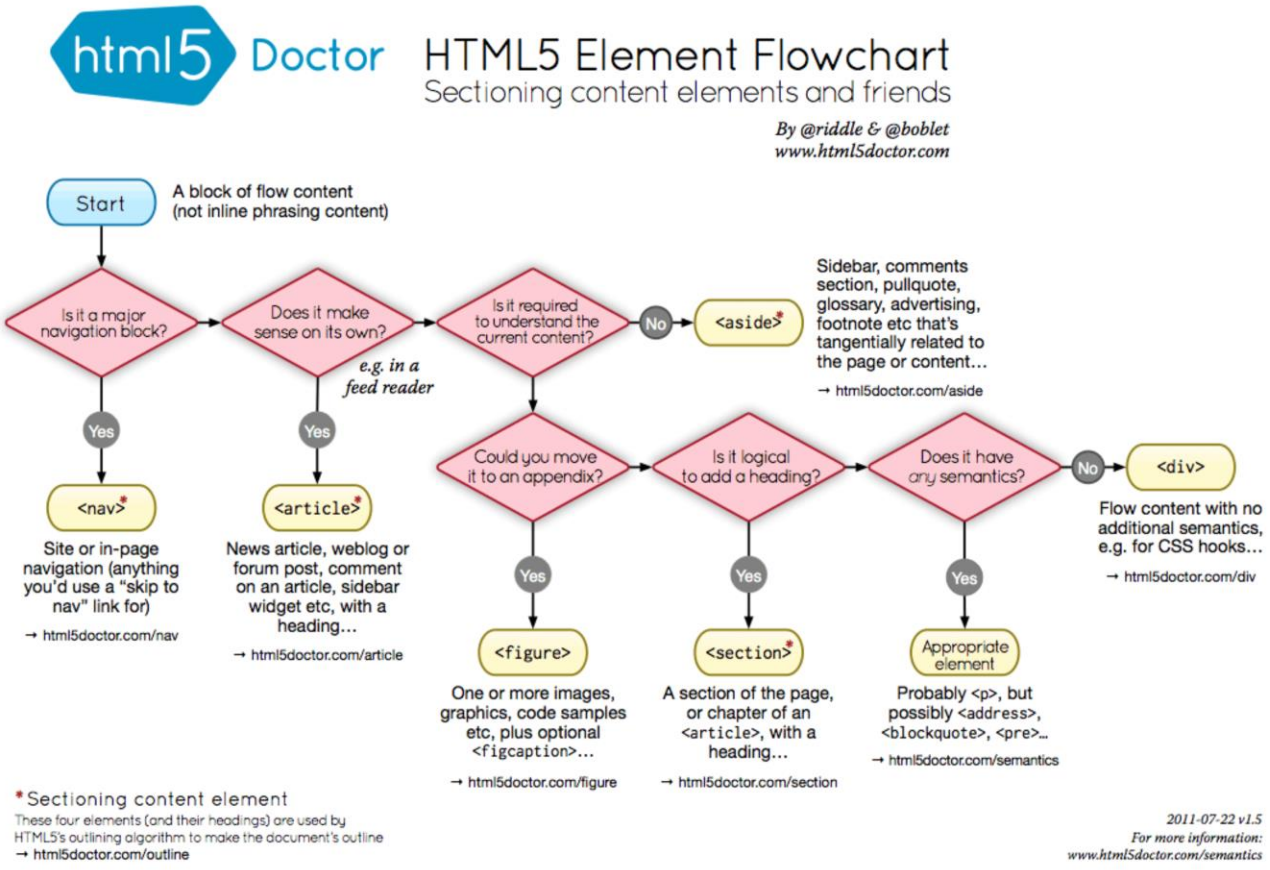
- **There are other HTML elements with very specific meanings, including:**
  - The HTML `<address>` element indicates that the enclosed HTML provides contact information for a person or people, or for an organization.
  - The HTML `<blockquote>` element indicates that the enclosed text is an extended quotation

    * It is usually rendered indented
  - The HTML `<code>` element displays its contents styled in a way the indicates that the text is a fragment of computer code

    * It is usually rendered in a non-proportional, monospaced font

How Do I Know What Tags to Use?

- **Many people like to use the HTML5 Doctor flowchart to help decide which semantic tags to use**



Exercise

In this exercise, you will create a Git repository on GitHub for a project called `simple-travel-site`. It will contain two pages of information about your favorite city.

Step 1. Create the repo

Create a Git repository on GitHub for a project called `simple-travel-site`. Check the box to include a `README.md` file.

Now, clone it to your local machine into the local `C:\LearnToCode\Workbook1` directory where you are keeping your HTML, CSS and Bootstrap projects.

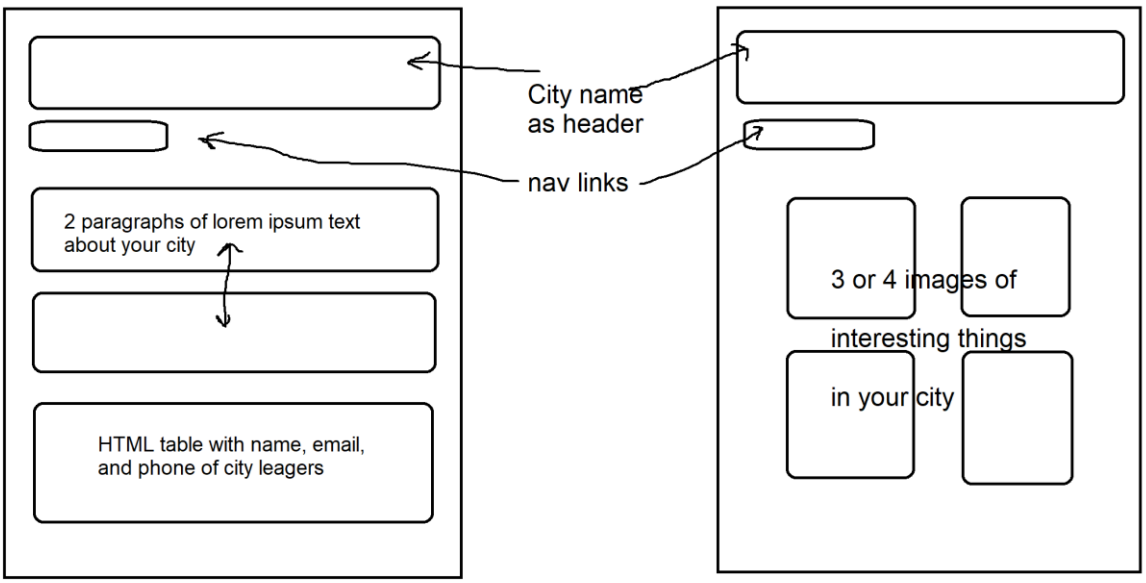
Step 2. Set up the project

Add folders named `css`, `images`, and `scripts`. Also add two pages named `index.html` and `cityimages.html`.

Now do a commit. Use the message "Added project structure". Push your changes to GitHub.

Step 3. Set up headers and navigation

You will create a two page web site with information about you (possibly fictional) favorite city. High-level page mockups are shown below.



Begin by designing your header and navigation links and include the same markup on each page. Make sure to use HTML5 semantic elements.

Test your appearance and the functionality of your links.

Now do a commit. Use the message "Added headers and navigation". Push your changes to GitHub. Go to GitHub and make sure you changed are there.

Step 4. Design the index page

Begin work on the `index.html` page content. Find a lorem ipsum generator and use text from there for your paragraphs.

Test your page. Then do a commit. Use the message "Flushed out index page". Push your changes to GitHub. Go to GitHub and make sure you changed are there.

Step 5. Design the images page

Create a new branch named ImagesPageBranch. Then, switch to that branch.

Find 3 or 4 images and add them to your images folder. Use a tool (MS Paint?) to resize the images so that they are all the same size.

Add the images to the `cityimages.html` page. See if you can get them to appear side-by-side or two to a row.

Test your page. Then do a commit. Use the message "Flushed out cityimages page".

Step 5. Merge and push

Switch back to the main branch.

Merge your ImagesPageBranch.

Check the status and then push your changes to GitHub. Now, go to GitHub and make sure you changed are there.

Module 2

CSS: A Review

Section 2–1

CSS Selectors and Common Properties

CSS: How to Make HTML Look Good

- **CSS is the language for describing the presentation of Web pages, including colors, layout, and fonts using a set of rules**
  - It is responsible for describing how elements should be displayed on a page by overriding your browser's default styling
  - CSS stands for Cascading Style Sheets

- **CSS also allows you to adapt the page's presentation to different types of devices**
  - For example, large screens, small screens, or printers.

- **CSS is independent of HTML and can be used with any XML-based markup language**
  - Having said that, we will look at its use with HTML and browsers

- **"Cascading" in the context of CSS describes the process by which a browser decides which rule applies to a specific piece of HTML**
  - This is need because more than one stylesheet rule could apply to a particular piece of HTML,

- **The CSS rule used is chosen by cascading down from the more general rules to the specific rule required**
  - The most specific rule is chosen

- **For example: if there was a rule that applied to all divs vs. a rule that applied to a div with a specific id, it would choose the ID rule**
  - It is the more specific rule

Inline Styling

- **HTML allows use to write style inline in our HTML tags**
  - This is done using the `style` attribute

- ***Although not the best practice*, it can come in handy when you are trying to test things out**
  - However, once you know what you want to do, you should move any inline styling to a separate style sheet before releasing your code

Inline Style Example

```
<h1 style="text-align: center;">Look Here!</h1>

<p style="height: 100px; color: red;">This is inline CSS</p>
```

Internal Styles using the `<style>` tag

- **Instead of using inline styles, you can place a style tag in the page's `<head>` tag**
  - This is better than using inline styles, but not as good as moving styles into a .css file
  - Each page would have to duplicate the styles they need and keeping them in sync would be tedious at best!

Internal Style Example

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      h1 {
          height: 200px;
          background-color: black;
          color: white;
      }
    </style>
  </head>
  <body>
    <!-- code not shown -->
  </body>
</html>
```

External Styles using the `<link>` tag

- **The `<link>` tag allows you to make an external style sheet available to your HTML file**

- **The `<link>` tag should be placed in the `<head>` tag of your site**
  - You can include more than one stylesheet
  - You should include your style sheets in order from generic to specific.

- **There several attributes for a `<link>` tag, including:**
  - **type** - The common use of this attribute is to define the type of style sheet linked and the most common current value is `text/css`, which indicates a Cascading Style Sheet format (this can usually be omitted)
  - **rel** - This lets the browser know what type of file we are linking. In this case, it will be a `stylesheet`
  - **href** - This attribute specifies the URL of the linked resource

`<link>` Example

```
<head>
  <!-- reference to your CSS page -->
  <link href="styles/sweet-styles.css" rel="stylesheet" />
</head>
```

- **Although we can write CSS in our HTML pages directly via the `<style>` tag or using inline styles, it is best practice to keep you CSS in its own file with the extension `.css`**
  - We can apply the same styling to multiple HTML pages
  - Keeps our site more organized. We know where to find our CSS quickly
  - Keeps you from writing the same code over and over again

CSS Syntax and Selectors

- **When coding CSS, you are really creating *rules***

- **Each rule consists of a selector and a declaration block**
  - The declaration block is wrapped in curly braces `{ }`
  - Each declaration and its value are separated by a colon (**:**) and end with a semi-colon (**;**)

CSS Rule Example

```
h1 {
  height: 200px;
  background-color: black;
  color: white;
}
```

  - **h1** - represents the selector
  - **height, background-color, and color** - represents the declarations

Simple Selectors

- **Simple selectors directly match one or more elements of a document and include:**
  - element selectors - selects all elements of a given tag type
  - class selectors - selects all elements that have a class attribute with that name
  - ID selectors - selects THE ONE element that has that ID
  - the universal selector - selects everything

Simple Selector Examples

```
/* targeting an element by it's name */
h1 {
  ...
}
/* targeting an element by a given class (class="myClass") */
.myClass {
  ...
}
/* targeting an element by a given id (id="myId") */
#myId {
  ...
}

/* The Universal Selector. Select all the things */
* {
  ...
}
```

Attribute Selectors

- **Attribute selectors match elements based on their *attributes* and *attribute values***

Attribute Selector Examples

```
/* All elements containing the attribute title are bolded */
[title] {
    font-weight: bold;
}
/* All elements with attribute "title" with the exact value "Buy American" are given a blue background color */
[title="Buy American"] {
    background-color: blue;
    color: white;
}
/* All elements with attribute "href" that contains the substring "city" given a red text color */
[href~="city"] {
    color: red;
}
/* anchors with a title (tooltip) */
a [title] {
  color: red;
}
/* img tag with an alt tag */
img [alt] {
    border: 5px solid red;
}
/* img tag with src that ends with .png */
img [src$=".png"] {
    border-radius: 25px;
}
```

Pseudo-class Selectors

- **CSS pseudo-class is a keyword that represents the *state* something may be in**
  - It is added to a selector separated by a colon (:)

- **It allows you to style selected elements only when they are in a certain state**

Common Pseudo-class Selector Examples

```
/* Change what a link should look like when hovered over with the mouse */
a:hover {
  color: darkred;
  text-decoration: none;
}

/* Change what a link should look like when its been visited */
a:visited {
  color: blue;
  text-decoration: none;
}
```

Combinators and Multiple Selectors

- **Combinators let you style elements based on their relationship with other elements**

- **Multiple selectors let you share settings in more than one situation**

Simple Combinators and Multiple Selectors examples

```
/* target all <p> tags within a <div> and make the text red */
div p {
  color: red;
}

/* target all <div> and <p> tags on the page and make their text blue */
div, p {
  color: blue;
}
```

Exercise

There are some great web sites where you can practice your CSS selectors without having to create a web page.

In this exercise, we will use the **CSS Diner** application at `https://flukeout.github.io/` and try to do "at least" the first 12 exercises in the time we have available.

Feel free to talk to student in your breakout room for help. Make sure everyone understands why each exercise works!

Section 2–2

CSS Properties

Understanding Units

- **Many of the CSS properties you will learn about expect units**
  - The width of an image
  - The amount of padding
  - The thickness of a border
  - The size of a font

- **Two types of units are supported:**
  - **absolute -** a repeatable, measurable size such as in, cm, mm or pixel based such as px or pt

    * *Not recommended because of the variety of screen sizes*
  - **relative** - a size relative to another property such as % or em

```
h1 {
  font-size: 2em;
  border: 3px solid black;
}
```

- **For more information, see:**
  `https://www.w3schools.com/cssref/css_units.asp`

Common Properties

- **Common CSS properties include:**
  - **background** - Control all things related to the background including but not limited to color and image
  - **color** - Controls the color of the text within an element.
    - Note: for a good CSS color reference, visit:
      `https://www.w3schools.com/cssref/css_colors.asp`
  - **font-weight** - controls the boldness of the font. Common values are normal and bold
  - **font-size** - specifies the size of the font. Usually in pixels or em units.
    - Note: for a good font-size reference, visit:
      `https://developer.mozilla.org/en-US/docs/Web/CSS/font-size`
  - **font-family** - specifies the font to use.
    - Note: For a list of web safe fonts, visit:
      `https://www.w3schools.com/cssref/css_websafe_fonts.asp`
  - **height** - controls the height of the content area of an element
  - **width** - controls the width of the content area of an element
  - **padding** - sets the padding space on all sides of an element.
    - Note: For a description of padding options, visit:
      `https://developer.mozilla.org/en-US/docs/Web/CSS/padding`
  - **margin** - sets the margin on all sides of an element.
    - Note: For a description of margin options, visit:
      `https://developer.mozilla.org/en-US/docs/Web/CSS/margin`
  - **border** - sets the border on all sides of an element.
    - Note: For a description of border options, visit:
      `https://developer.mozilla.org/en-US/docs/Web/CSS/border`

- **There a ton of CSS properties -- refer to the links for a deep dive into some of the properties and what they control**
  - To learn more about them, visit: `https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference`

PRO TIP

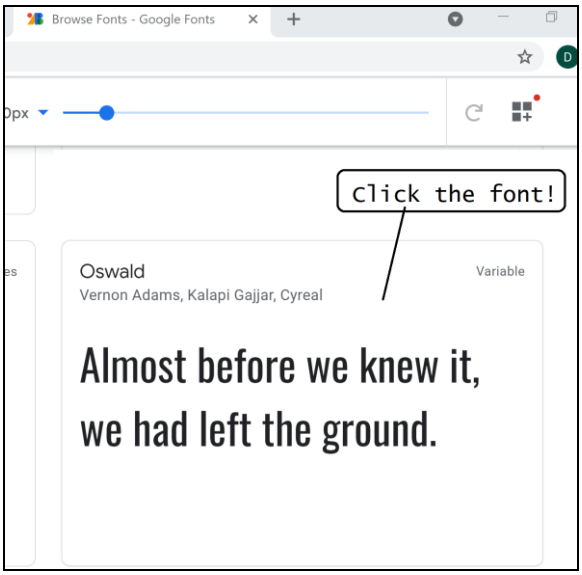
`https://mdn.io/YOUR_PROPERTY_HERE`

Replace YOUR_PROPERTY_HERE with something like 'color', 'background', or any other property you can think of should bring you to documentation on that specific property

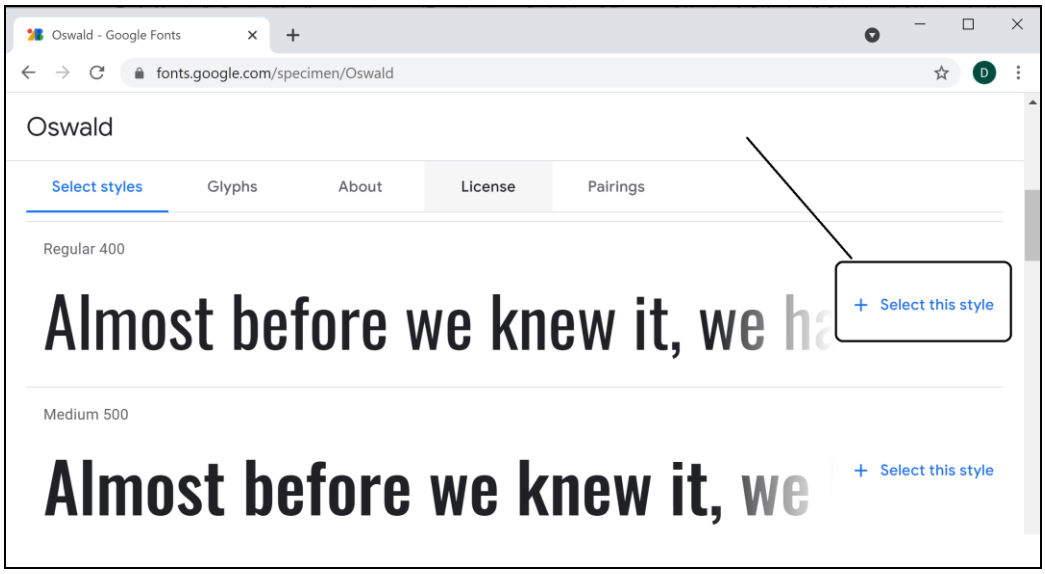
Custom Fonts

- Google has a large number of custom fonts available free for your use
  - `https://fonts.google.com/`

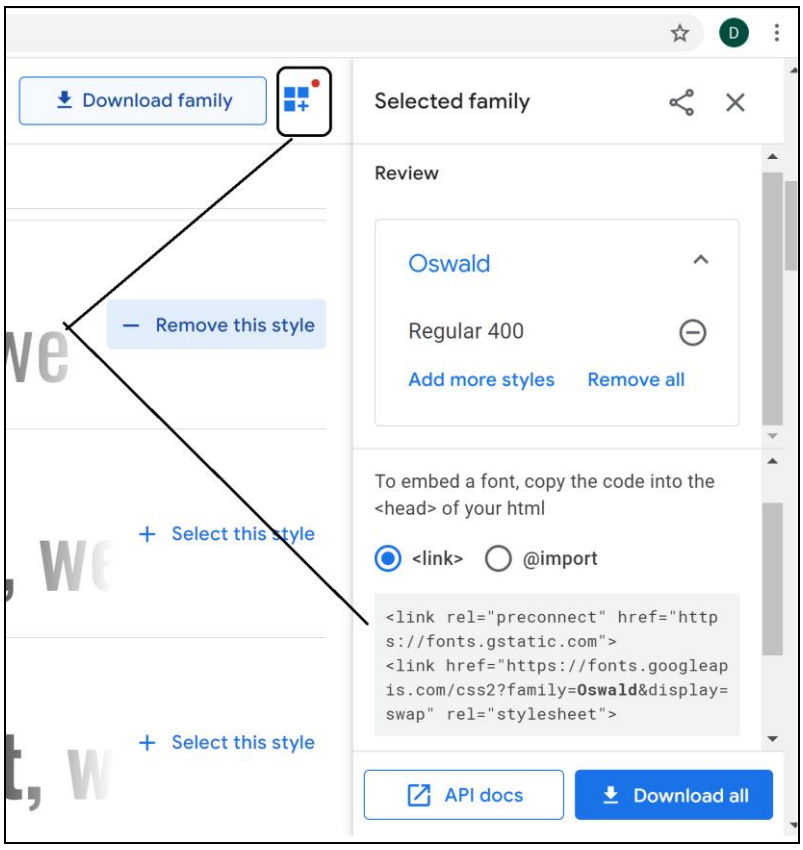
- To learn about the font, click the font



- Then select the font with the characteristics you want



- **Finally, view the tags to need to use to link the font in to your HTML**



- **The preconnect link tells the browser to connect to the server ahead of time to make the download go faster**
  - Note: Add `crossorigin` to the preconnect link

```
<head>
   <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
   <link href="https://fonts.googleapis.com/css2?family=Oswald&display=swap"
rel="stylesheet">
   <style>
      body {
        font-family: "Oswald", sans-serif;
      }
   </style>
</head>
```

Useful Resources

- **CSS at W3Schools -** https://www.w3schools.com/css/css_intro.asp
- **CSS at Mozilla -** https://developer.mozilla.org/en-US/docs/Web/CSS
- **CSS Selector Reference -** https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors

Exercise

Revisit your `simple-travel-site` project. We are going to add some styling rules to your site. We will do it in a new branch so that we can test out things before merging our changes back into our main branch. If we mess things up, we won't break our original code base.

Step 1: Make sure you have the most recent copy locally

Use the git pull command to get the most recent version repo from GitHub.

Step 2: Create a new working branch

Create a new branch off main named EnhancedIndexBranch. Then switch to it.

Step 3: Add the enhanced styling

If you don't have a `.css` file for your site, create one now and then link it to your `index.html` page.

Using an ***element selector***, create a CSS rule that applies to all paragraphs. In it, set the font-family to one of the ones from: `https://www.w3schools.com/cssref/css_websafe_fonts.asp.` Make sure to include the fallback font of either `serif` on `sans-serif`.

Using a ***class selector***, create a CSS rule that will be applied to the HTML table. Place a border around the HTML table and its cells. If you aren't sure how, refer to `https://www.w3schools.com/css/css_table.asp.` Apply that class to the HTML table on the index page.

Test your page.

When things work well, commit your changes.

Step 4: Merge your changes

Switch back to the main branch. Since you are happy that your changes didn't mess anything up, merge your EnhancedIndexBranch.

You can delete your EnhancedIndexBranch.

Run git status and if all is well, push your changes to GitHub.

Section 2–3

CSS Specificity, the Box Model, Positioning, Floats, and CSS Resets

CSS Specificity

- **CSS Specificity means that some selectors are going to be more *important* than others**

- **For example:**
  - Type selectors like `div` or `h1` are generic and will have a lower specificity
  - Selectors that reference things by id like `#myDiv` will be more *important* or have a higher specificity

```
/* Type selectors have the lowest specificity */
h1, div, a {
   .....
}

/* Class, attribute, and pseudo selectors have a higher specificity than Type selectors */
.nav-link, .contentDiv, .recipe, .albumCover {
   .....
}

/* ID selectors have a highest specificity */
#mainNav, #logo, #slideShow, .#contactForm {
   .....
}
```

Specificity Examples

- **To understand CSS specificity, you need to consider: when given two or more selectors that could apply to an element, which one "wins"?**

- **Consider the following case that has two class selectors that match**

```
/* CSS */
div {
  height: 200px;
  width: 200px;
}
.green {
  background-color: green;
}
.fail {
  background-color: red;
}
<!-- HTML -->
<div class="green fail"></div>
```

  - The div selector matches so the size of the div is 200 x 200, but what about the background color?
  - Because `.green` and `.fail` are both ***class selectors of equal importance*** `.fail` wins! But it's not because of the order of the CSS rules… it's because of the order in `class="green fail"`

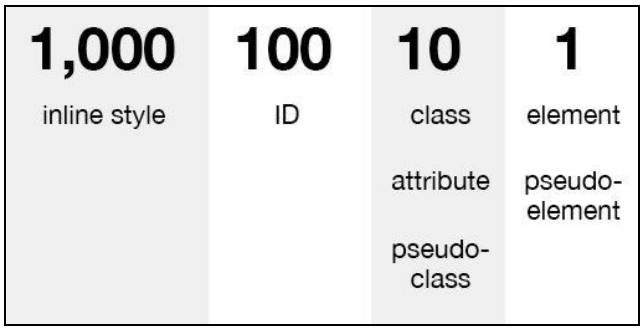
- **Consider the following case that has a class and an ID selector that match**

```
/* CSS */
div {
  height: 200px;
  width: 200px;
}
 #info {
  background-color: blue;
}
.green {
  background-color: green;
}
<!-- HTML -->
<div id="info" class="green"></div>
```

  - The ID selector wins! The div is 200 x 200 and the background is blue

What are the Specificity Rules?

- **There are simple rules that govern specificity based on a simple point system**



- **How do you apply it?**
  - Count the number of each type and create a number for it
  - Inline styles should have the highest specificity, so they aren't shown

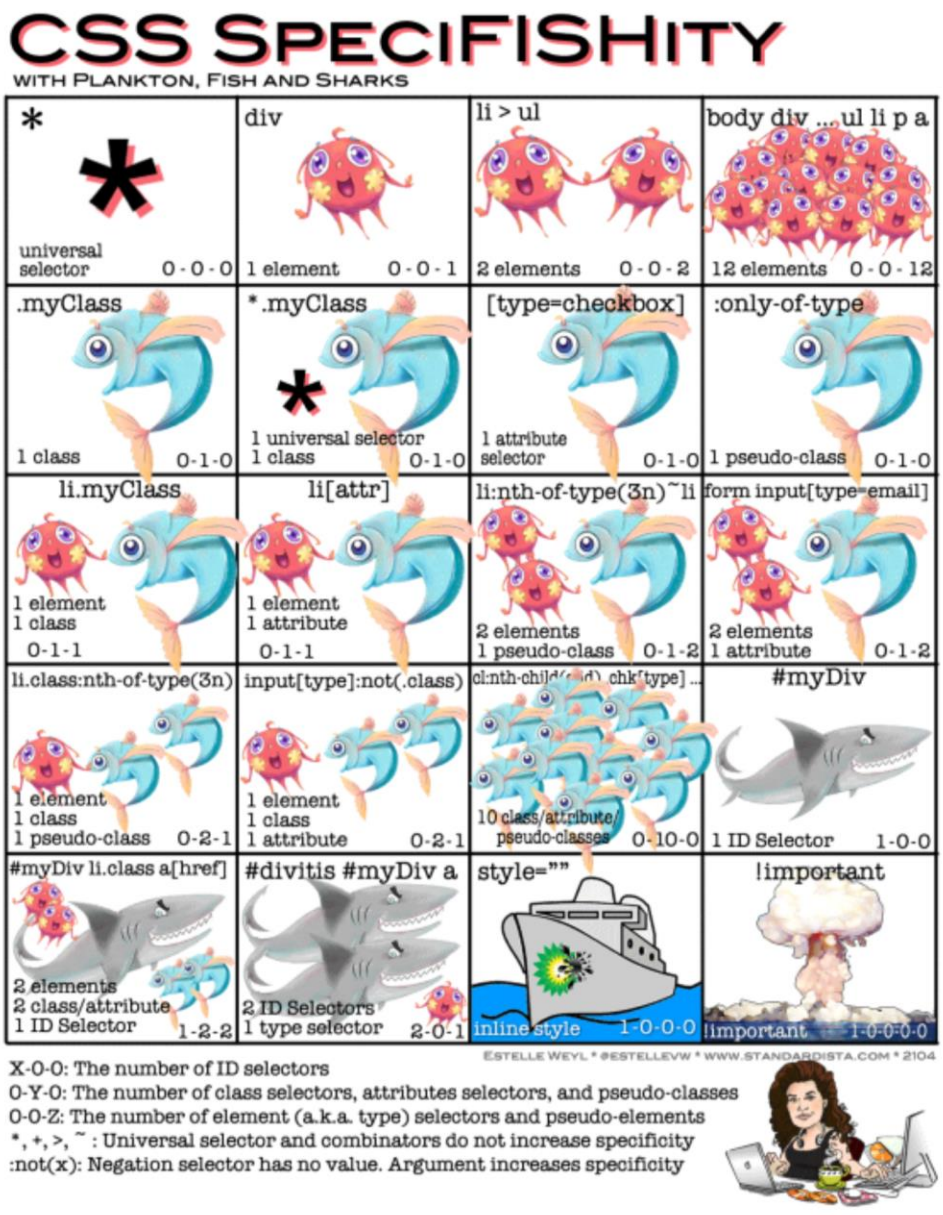
```
div             /* a=0 b=0 c=1 -> specificity =   1 */
div p           /* a=0 b=0 c=2 -> specificity =   2 */
div div a       /* a=0 b=0 c=3 -> specificity =   3 */
img [alt]       /* a=0 b=1 c=1 -> specificity =  11 */
img.red [alt]   /* a=0 b=2 c=1 -> specificity =  21 */
#logo           /* a=1 b=0 c=0 -> specificity = 100 */
#logo p         /* a=1 b=0 c=1 -> specificity = 101 */
```

- **There are a few special situations to keep in mind:**
  - The universal selector (`*`) is worth 0 points
  - Element selectors never beat a class selector
  - When two selectors have the same specificity, the last one wins
  - `!important` should win

- **You can compare two selectors at this specificity calculator:**
  `https://specificity.keegan.st/`

- **The official documentation for specificity is found at:**
  `https://www.w3.org/TR/selectors-3/#specificity`

Specifishity - Chart for reference



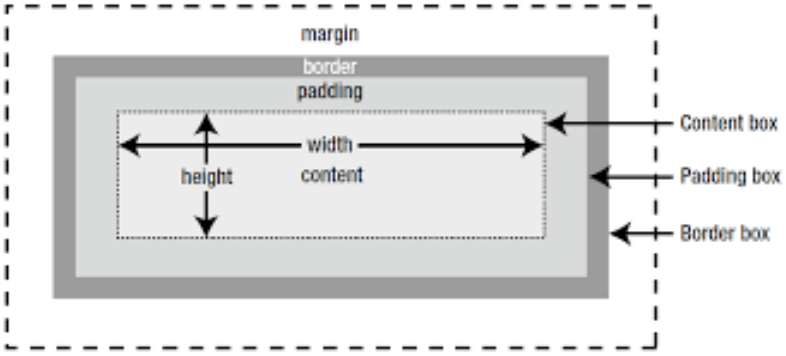
From: `https://specifishity.com/`

Section 2–4

The CSS Box Model

The CSS Box Model

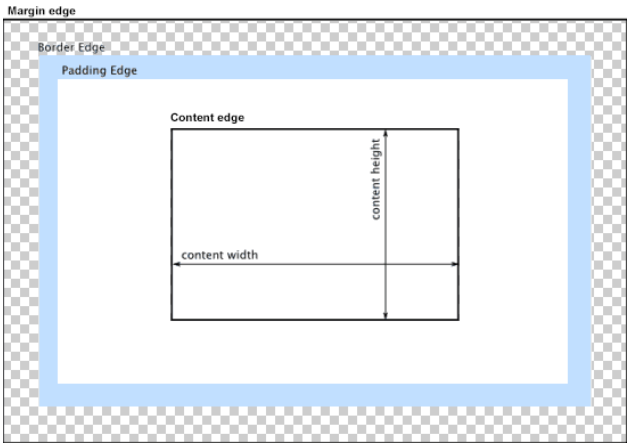
- **Every element is represented by a box (unless we indicate `display: none`) that contains:**
  - the content
  - the padding around the content
  - a border around the padding
  - a margin around the border



- **You can adjust the box using the CSS properties `padding`, `border`, and `margin`**

Calculating the Box Size

- **When you set the `width` of an element, you are typically specifying the width of the *content portion* of the box only**



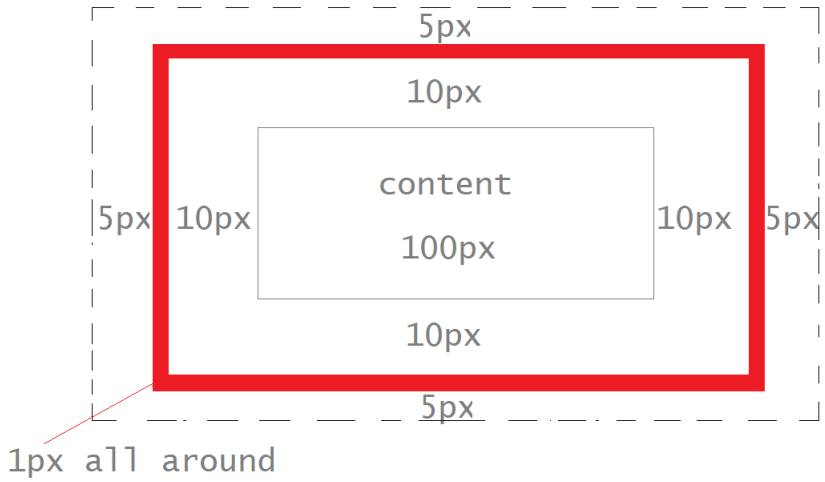
  - Margin is extra space *outside* of the border
  - Padding is the extra space *inside* of the border

- **To calculate the box size, you must add the padding, border, and margin to the width**

Example: Calculating the Box Size

- For example, if you had a div with the styles:

```
#myDiv {
  width: 100px;
  border: 1px solid red;
  padding: 10px;
  margin: 5px;
}
```



- The full width of the entire div would be `132px` because it has the margin, border and padding on each side of the content:

- To see shorthand ways to set margin, look here: https://www.w3schools.com/css/css_margin.asp

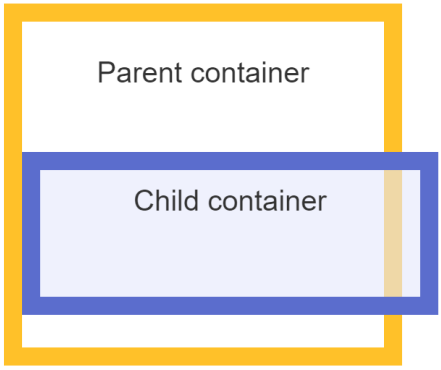
- To see shorthand ways to set padding, look here: https://www.w3schools.com/css/css_padding.asp

Managing the Way that `width` Works

- **You can change the way *`width`* works on boxes using the `box-sizing` CSS property**
  - The default value for this is `content-box` which means the `width` property is applied to the **content**
  - We just discussed this on the previous page!

- **Novices can end up surprised at appearances**

```
/* CSS */
.childContainer {
   width: 100%;
   border: 10px solid blue;
   padding: 5px;
}
```



- **But you can set `box-sizing` to `border-box`, which means the `width` property is applied to the content, padding, and border**

- **With box-sizing, 100% width on child elements has no surprises**

```
/* CSS */
.childContainer {
   box-sizing: border-box;
   width: 100%;
   border: 10px solid green;
   padding: 5px;
}
```

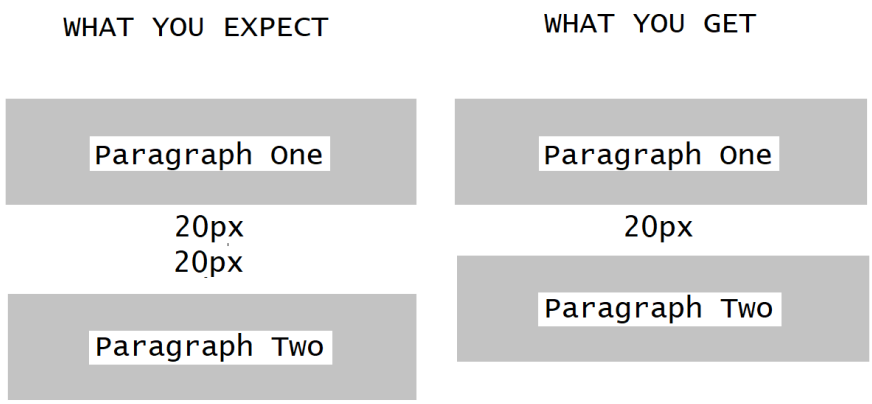
- **When you set `box-sizing` to `border-box`, the width ends up being the complete element width!**

```
/* CSS */
.box {
   box-sizing: border-box;
   width: 350px;
   border: 10px solid black;
}
<!-- HTML (div is exactly 350px wide (including the border) -->
<div class="box">
  <p>some content...</p>
</div>
```

Margin Collapse

- **The top and bottom margins of blocks are sometimes collapsed (combined) into a single margin**
  - When this happens, the margin is the size of the ***largest of the individual margins***

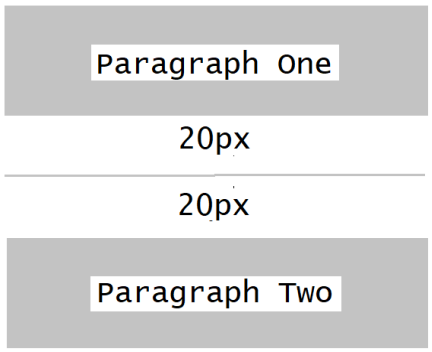
```
/* CSS */
p {
    margin-top: 20px;
    margin-bottom: 20px;
}
<!-- HTML -->
<p>Paragraph One</p>
<p>Paragraph Two</p>
```



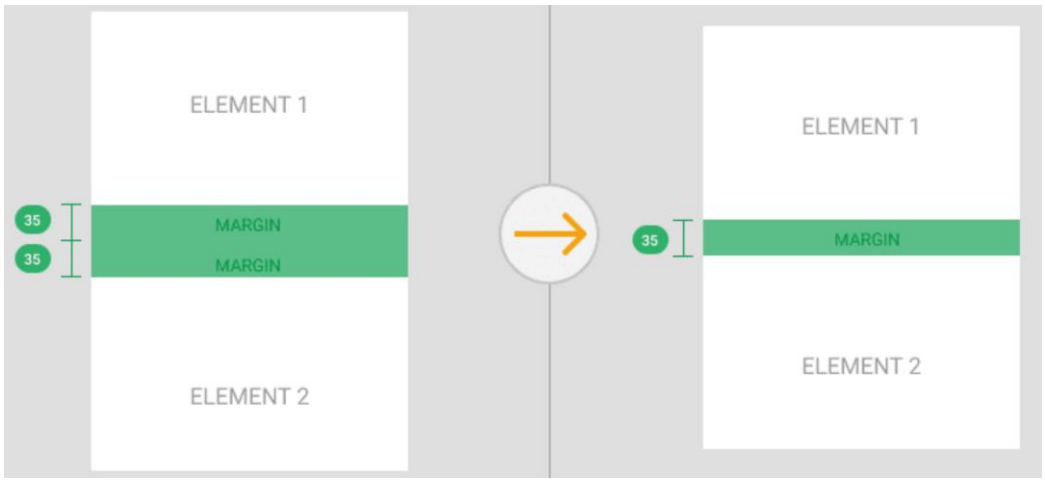
- **When does this happen?**
  - Well, first of all, only *vertical margins* collapse
    - You don't have to worry about horizontal margins
  - Secondly, only adjacent elements collapse

```
/* CSS */
p {
    margin-top: 20px;
    margin-bottom: 20px;
}
<!-- HTML -->
<p>Paragraph One</p>
<br>                          <!-- now the paragraphs aren't adjacent -->
<p>Paragraph Two</p>
```

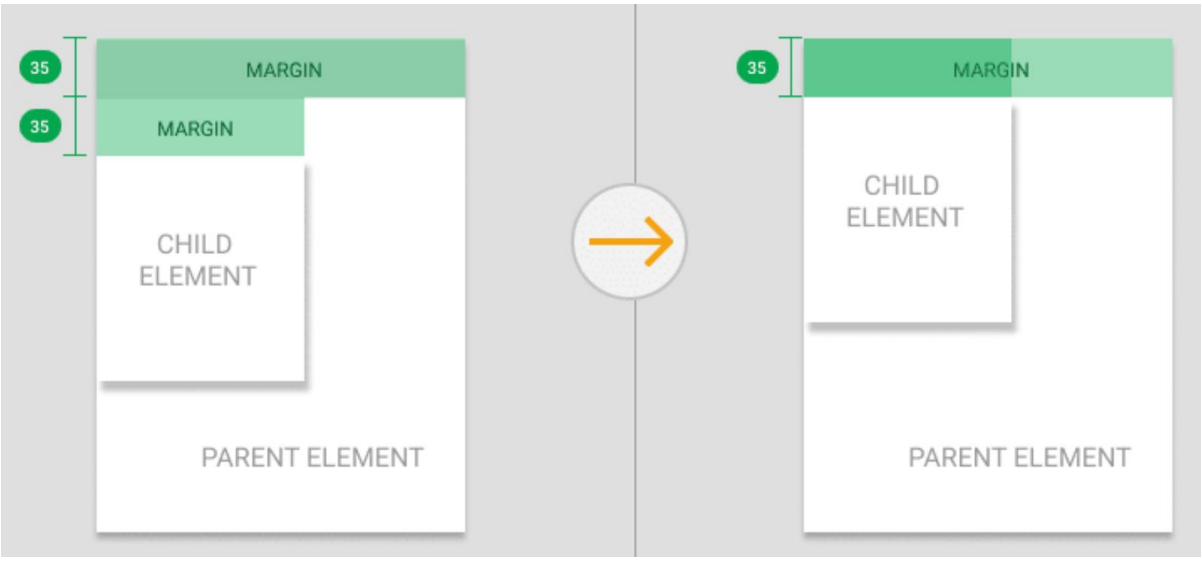
WHAT YOU GET



- **Collapsing margins are applied to adjacent siblings**
  - When there are adjacent siblings, the `margin-bottom` of the top element collapses with the `margin-top` of the bottom element



- **Collapsing margins may also be applied to parent and its first child element in some situations**
  - The parent element doesn't have a height set
  - The parent element doesn't have any padding or border along the relevant edge



  - The same happens with the last child of the parent!!

- **Read the following for a great description of collapsing margins and all of the special situations we haven't discussed when you encounter confusing layout**
  - `https://www.joshwcomeau.com/css/rules-of-margin-collapse/`

Section 2–5

Positioning

Positioning

- **Positioning refers to the method used to determine an element's place on the page**
  - The `position` property has several values that can be set

- **Once an element is positioned, there following properties mare available to adjust its position:**
  - `top`, `right`, `bottom`, `left` and `z-index`

- **Position choices include:**
  - **static** -- Default behavior. That is, it is laid out in its current position in the flow. The top, right, bottom, left and z-index properties do not apply.
  - **fixed** -- An element with a fixed position stays in the same place on the page, even while scrolling, but is removed from the normal flow of the document. Its position is relative to its parent element but no space is reserved for it in the flow of the document.
  - **relative** -- Elements are positioned relative to where they would have been. Adjusting top, right, bottom, and left will position the element without affecting elements around it.
  - **absolute** -- An element with an absolute position is removed from the normal flow of the document and positioned relative to its nearest positioned ancestor. If there is no positioned ancestor it is positioned relative to the document body.
  - **sticky** -- An element with a sticky position remains relative to where it would appear in the normal flow of the document even while scrolling. While scrolling it will remain sticky in its position on the screen

Fixed position items stay there while scrolling

```
/* CSS */
.fixed {
  border: 5px solid black;
  position: fixed;
  bottom: 0;
  left: 0;
  right: 0;
}
<!--  HTML -->
<div class="fixed">
This div stays fixed at the bottom of the page even when scrolling
</div>
```

Absolute position example

```
/* CSS */

.absolute {

  position: absolute;

  top: 100px;

  left: 100px;

  border: 3px solid blue;

}

<!-- HTML -->

<div class="absolute">

  <p>I am 100px down and over from the top left corner of the page.  I can scroll out of view.</p>

</div>
```

- **Good places to see examples of the position property include:**
  - https://www.freecodecamp.org/news/how-to-use-the-position-property-in-css-to-align-elements-d8f49c403a26/
  - https://blog.pragmatists.com/css-position-by-example-48e78f787b60
  - https://css-tricks.com/almanac/properties/p/position/

Floating

- **The `float` CSS property specifies that an element should be positioned along the left or right side of its container, *where text and inline elements will wrap around it***

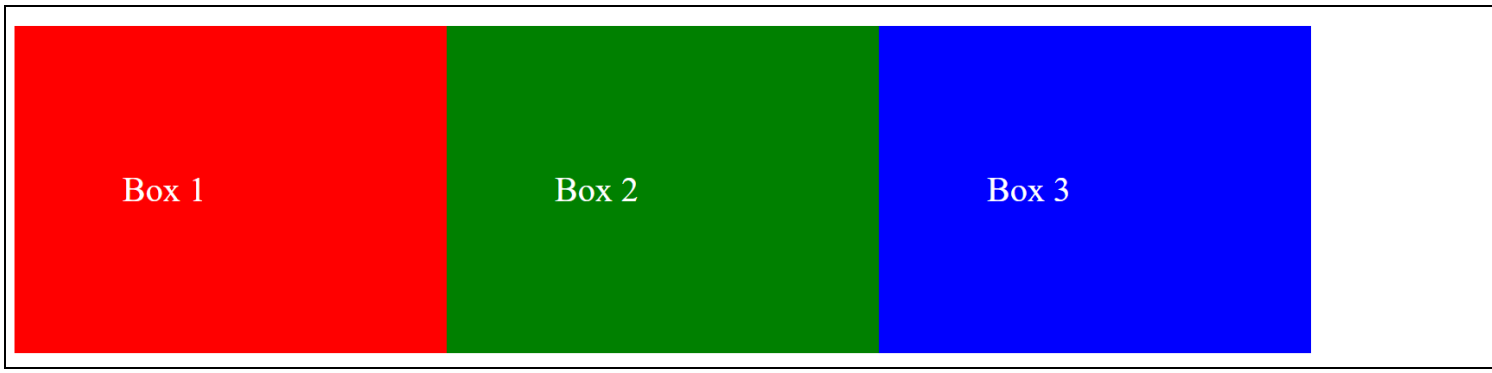
- **Settings include:**
  - **float: none** -- Default behavior. The element should remain where it is placed in the flow of the document
  - **float: left** -- Element should be positioned on the left side of its container
  - **float: right** -- Element should be positioned on the right side of its container

Example of float left

```
/* CSS */
* {
  box-sizing: border-box;
}
.box {
  float: left;                    /* floating left */
  width: 200px;
  padding: 25px;
  color: white;
}


<!-- HTML -->
<div>
  <div class="box" style="background-color:red ">
    <p>Box 1</p>
  </div>
  <div class="box" style="background-color:green">
    <p>Box 2</p>
```

```
  </div>
  <div class="box" style="background-color:blue">
   <p>Box 3</p>
  </div>
</div>
```

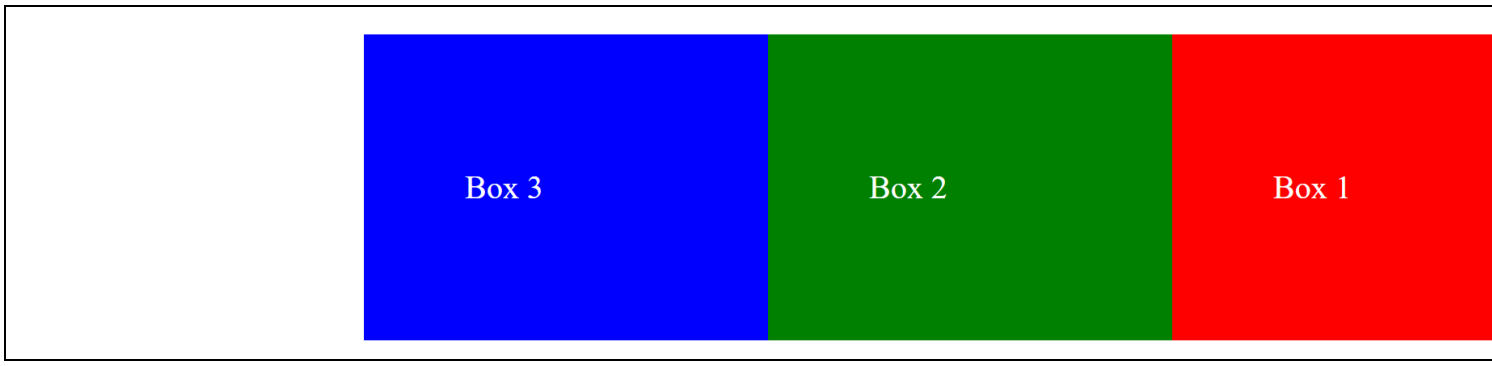


Example of float right

```
/* CSS */
* {
  box-sizing: border-box;
}
.box {
  float: right;                          /* floating right */
  width: 200px;
  padding: 25px;
  color: white;
}

<!-- HTML -->
<div>
  <div class="box" style="background-color:red ">
    <p>Box 1</p>
  </div>
  <div class="box" style="background-color:green">
    <p>Box 2</p>
```

```
  </div>
  <div class="box" style="background-color:blue">
   <p>Box 3</p>
  </div>
</div>
```



- **The only issue with float is that you have to stop floating at some point!**

Understanding `clear` and `clearfix`

- **The CSS float property specifies how an element should float**

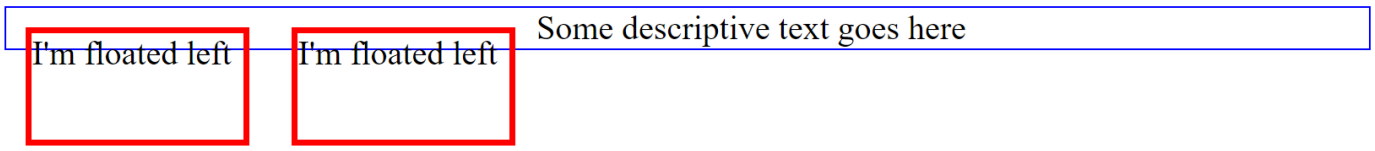
- **The CSS `clear` property specifies what elements can float beside the cleared element and on which side**
  - **`none`** - Allows floating elements on both sides. This is default
  - **`left`** - No floating elements allowed on the left side
  - **`right`**- No floating elements allowed on the right side
  - **`both`** - No floating elements allowed on either the left or the right side

- **Usually, we use the `clear` property after we have used a `float` property on an element**
  - Clearing almost always matches the float

- **For example, if an element is floated to the left, then you should clear the element below that doesn't float to the left**
  - Your floated element will continue to float, but the cleared element will appear below it on the page

Without clearing, float disrupts the flow of elements "below" them

```
/* CSS */
.left {
  float: left;
  width: 100px;
  height: 50px;
  margin: 10px;
  border: 3px solid red;
}
.message {
  border: 1px solid blue;
}
<!-- HTML -->
<div class="left">I'm floated left</div>
<div class="left">I'm floated left</div>
<div class="message">Some descriptive text goes here</div>
```



A cleared element adjusts for the flow

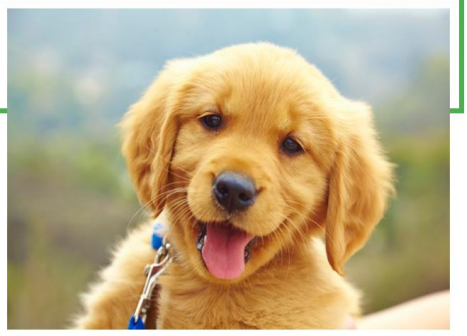
```
/* CSS */
.left {
  float: left;
  width: 100px;
  height: 50px;
  margin: 10px;
  border: 3px solid red;
}
.message {
  border: 1px solid blue;
  clear: left;
}
<!-- HTML -->
<div class="left">I'm floated left</div>
<div class="left">I'm floated left</div>
<div class="message">Some descriptive text goes here</div>
```

Overflow Issue

- **When a floated element is taller than its container element, it will "overflow" outside of its container**

```
/* CSS */
div {
  border: 3px solid green;
  padding: 5px;
}
.dog {
  float: right;
}
<!-- HTML -->
<div>
  <img class="dog" src="images/happy.jpg" alt="My Puppy" width="250" height="180">
  <p>My puppy is cute.<p>
</div>
```

My puppy is cute.



- **You can use a clearfix hack to fix this**
  - When we apply the **clearfix** to the container, we allow floats to happen without the container collapsing and causing our content shift with the container

```
/* CSS */
div {
  border: 3px solid green;
```

```
  padding: 5px;
}
.dog {
  float: right;
}
.clearfix::after {
  content: "";
  clear: both;
  display: block;
}

<!-- HTML -->
<div class="clearfix">
  <img class="dog" src="images/happy.jpg" alt="My Puppy" width="250" height="180">
  <p>My puppy is cute.<p>
</div>
```

My puppy is cute.

Just a Note

- **Managing the "floating" of elements used to be something we dreaded; it is tedious and prone to errors**
- **Two advances in CSS mean we rarely do what we just illustrated**
- **Instead, we manage layout with:**
  - Flexboxes
  - Bootstrap

CSS Resets and Normalize.css

- **CSS resets are used to 'normalize' the starting point for CSS across all browsers**

- **Several years ago, browsers deviated in their CSS defaults by a good bit**
  - They occasionally still do

- **This makes it hard to support older browsers**
  - The CSS defaults may not be what you want

- **You can include a CSS reset to clear most CSS defaults**
  - CSS resets usually aim to ***remove all built-in browser styling***
  - You're then supposed to add most decoration in your own CSS

- **A reset many people have used is found here:** `https://meyerweb.com/eric/tools/css/reset/`
  - But there are other resets, as described here: `https://www.webfx.com/blog/web-design/css-reset-stylesheets/`

```
<link rel="stylesheet" href="css/my-reset.css">
<link rel="stylesheet" href="css/styles.css">
```

- **Another approach is to use Normalize CSS**

- **Normalize.css makes browsers *render all elements more consistently and in line with modern standards***
  - Rather than clearing CSS settings, it standardizes (normalizes) them
    - Elements like H1-6 will appear bold, larger et cetera in a consistent way across browsers

- You're supposed to add ***only the* difference** in decoration your elements need

```
<link rel="stylesheet" href="css/normalize.css">

<link rel="stylesheet" href="css/styles.css">
```

- **You can also use Normalize from a Content Delivery Network (CDN)**
  - A CDN is a web-based server that hosts very common CSS and JavaScript frameworks that many sites use
  - The idea is that your browser may have already downloaded and cached the files so that your page doesn't have to explicitly download the files

```
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/normalize/8.0.1/normalize.css">

<link rel="stylesheet" href="css/styles.css">
```

Section 2–6

CSS Flexbox

Flexbox

- **CSS flexbox layout allows you to easily format HTML to align items vertically and horizontally using rows and columns**

- **It provides a more efficient way to layout, align and distribute space among items in a container that what we've learned so far**
  - This is true even when their size is unknown and/or dynamic (thus the word "flex").

- **Flexbox is intended to help with component layouts (pieces of the page) and not larger layouts (the entire page).**

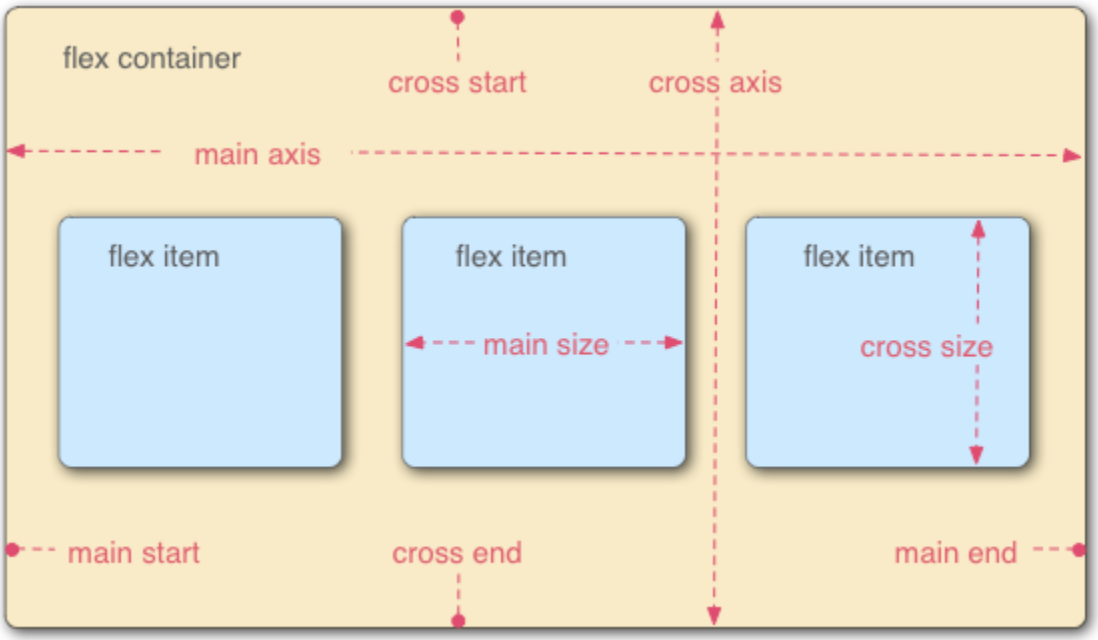
- **In a flexbox, items will "flex" to different sizes to fill the space**
  - They make responsive design easier

Flexbox Terminology

- **There are several new terms that are used with a flexbox**

- ***Flex container* is the element that contains the flex items**
  - A flex container is defined using the `flex` or `inline-flex` values of the **`display`** property

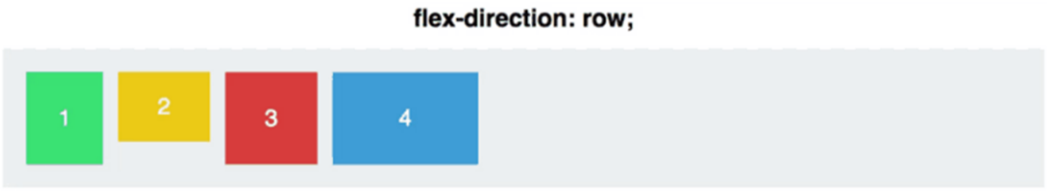
- ***Flex items* are the children of the flex container**



Common Flexbox CSS Properties for the Flex Container

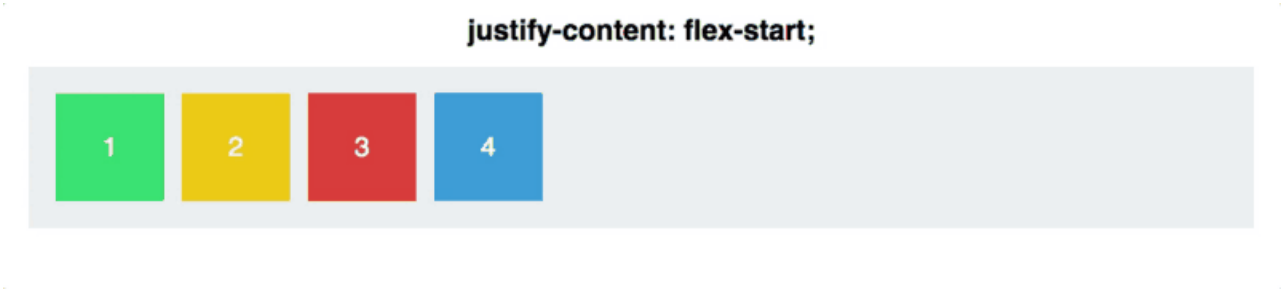
- **The `display` property can be set to `flex(box)` or `inline-flex` to turn the container into a flex container**
  - If the value is `flex` or `flexbox` for the `display` property, the element will become a *block-level* flex container
  - If the value is set to `inline-flex`, the element will become an *inline-level* flex container

- **The `flex-direction` property controls if items should be displayed in rows (inline) or columns (stacked).**
  - Valid options are `row`, `row-reverse`, `column`, and `column-reverse`



- **The `justify-content` property specifies how items are distributed on the main axis**
  - The main axis can change depending on the `flex-direction` property.
    * If `flex-direction` is set to `row` then the main axis is from left to right (horizontal)

    * If `flex-direction` is set to `column` then the main axis is from top to bottom (vertical
  - Options can include: `flex-start`, `flex-end`, `center`, `space-between`, and many others



- **The `align-items` property is similar to `justify-content` but controls how the items are distributed on the cross axis (opposite of the main axis)**
  - Options can include: `flex-start`, `flex-end`, `center`, `space-between`, and many others

- **The `flex-wrap` property specifies whether flex items are forced into a single line or can be wrapped onto multiple lines**
  - Options include: `nowrap`, `wrap`, and `wrap-reverse`

- **Rather than setting each of these properties separately, you can set the flex-flow property**
  - It is a shorthand for:

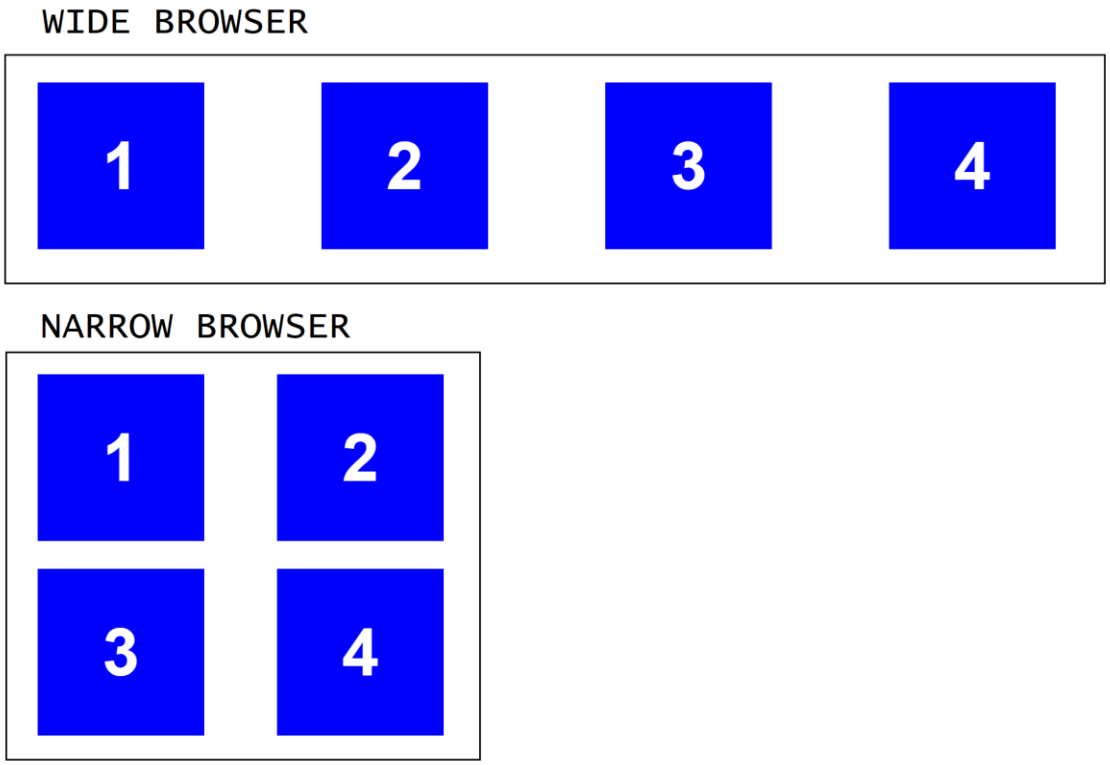
    * `flex-direction`

    * `flex-wrap`

Example: Flexbox Container with Items

```
/* CSS */
.flex-container {
  display: flex;                          /* make it a flexbox */
  flex-flow: row wrap;                    /* specify layout */
  justify-content: space-around;          /* decide what to do with empty space */
  margin: 0;
  padding: 10px;
  list-style: none;
}
.flex-item {
  width: 100px;
  height: 100px;
  margin: 10px;
  padding: 10px;
  background: blue;
  color: white;
  line-height: 100px;
  font-weight: bold;
  font-size: 3em;
  text-align: center;
}


<!-- HTML   (NOTE:  These do not have to be ul/li items -->
<ul class="flex-container">
  <li class="flex-item">1</li>
  <li class="flex-item">2</li>
  <li class="flex-item">3</li>
  <li class="flex-item">4</li>
</ul>
```



- **Now that we've seen some HTML/CSS for a flexbox, let's go watch some simple animations at:**

  `https://www.freecodecamp.org/news/flexbox-the-ultimate-css-flex-cheatsheet/`

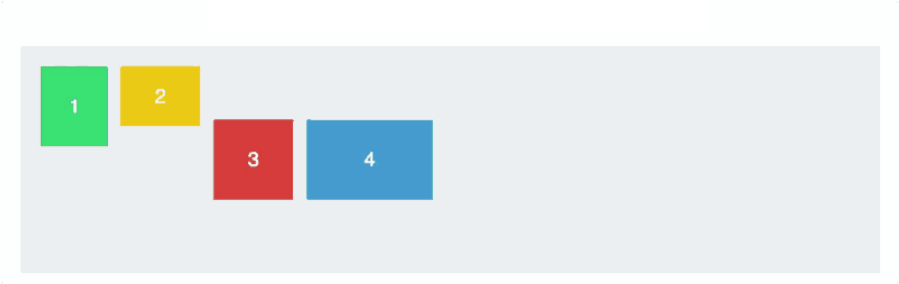
  – The show how a flexbox responds to different property values

Flexbox CSS Properties for the Flex Items

- **All of the properties we've looked at so far are settings for the flex containers**

- **The flex item also has a few properties you can set, including:**
  - The `align-self` property adjusts the alignment of a single flex item
    * `align-self` has all the same options as `align-items` that we would use on a container but only applies to a single flex item

```
1 and 2 have aligh-self: flex-start
3 and 4 have align-self: center
```



- **The `order` property allows you to specify the order in which the flexbox items should appear in the flex container**
  - It only affects the how the item is rendered; it doesn't affect how screen readers read the source code
  - Its default value is 0 and you can:
    * Use -1 to make one item the first
    * Use 1 to make one item the last
    * Specifically set each item's order (ex: 1, 2, 3)

```
/* CSS */
.alwaysFirst {
    order: -1;
}


<!-- HTML -->
<ul class="flex-container">
  <li class="flex-item">1</li>
  <li class="flex-item">2</li>
  <li class="flex-item">3</li>
  <li class="flex-item alwaysFirst">4</li>
</ul>
```

- **If your team doesn't use Bootstrap, you will become an expert at Flexbox in a short time!**

- **A good flexbox cheat sheet is:** https://www.freecodecamp.org/news/flexbox-the-ultimate-css-flex-cheatsheet/

Exercise

The easiest way to learn about the flexbox is to play an online game that lets you try out different flexbox properties.

In this exercise, we will go to `https://flexboxfroggy.com/` and try to complete at least the first 8 levels in the time we have available.

Feel free to talk to student in your breakout room for help. Make sure everyone understands why each exercise works!

Module 3

Bootstrap: A Review

Section 3–1

Responsive Design

Responsive Design

- **Responsive design refers to the act of designing a web page that work across devices of any size**

- **In the early days of phones with web browsing abilities, companies would create two versions of their site: `mysite.com` and `mobile.mysite.com`**
  - Do you remember browsing the web on a Motorola Razor flip phone?

- **These days, we want to create (mostly) a single code base that looks great on mobile devices, tablets, laptops, and desktops**

- **There are three core practices to make a site "responsive":**
  - Design around a *responsive grid* with percentage-based widths
  - Use *responsive images* that change `src` depending on the screen resolution
  - Use *media queries* to modify your CSS depending on the screen resolution

Responsive Breakpoints

- **Common breakpoints for different size device is usually:**
  - 320-480px: Phones
  - 481-768px: Tablets
  - 769-1024px: Laptops and small screens
  - 1025-1200px: Desktops and large screens
  - 1201px and larger: Extra large screens and TV

Responsive Images

- **When you specify a width for your images, make sure to do it in relative units and not absolute units**
  - This will make your images fluid can resize themselves appropriately based on screen size
  - Note: Avoiding fixed units is actually true for any type of HTML element in responsive design

```
.flagImage {
    width: 400px;     /* BAD IDEA - NOT RESPONSIVE */
}


.flagImage {
    width: 33%;       /* Responsive */
}
```

- **Another step to make an image responsive is to use the CSS the `max-width` property to 100%**
  - When set to 100%, the image will scale ***down*** if it has to, but never scale ***up*** to be bigger than its original size

Media Queries

- **Media queries let you create different CSS rules based on media types**
  - For example, you might design one set of rules for small screen widths and another set of rules for larger screen widths
  - You can also set CSS rules for when a page is going to be printed

- **We use a `@media` rule to create media queries and can check several things to define our rule, including:**
  - width and height of the viewport
  - width and height of the device
  - resolution
  - orientation (whether a device is in landscape or portrait mode)
  - media type (screen, sprint, or speech)
  - user agent

- **In responsive design, media queries can be used to specify different layouts for phones, tablets, laptops and desktops**

Example: Media Query

- **Sometimes images are simple decoration that might clutter the user experience on small screens**
  - In that case, you could use a media query to remove them

```
/* CSS RULE */
@media (max-width: 480px) {
   img.fluff {
     display: none;
   }
}


<!-- HTML -->
<img src="images/cutesyVines.png" alt="Vines" />
<img class="fluff" src="images/headquarters.png" alt="New Company Headquarters" />
```

- **The definitive reference for media queries is:**

https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries

Responsive Patterns

- **Responsive design is a while field of study on its own**

- **How should you approach the design of a web site that can be viewed by users on every type of imaginable device?**
  - This is especially complicated since well more than 50% of all network traffic is from mobile devices!

- **There are some common responsive patterns that regularly emerge in responsive sites, but the world is your oyster**
  - `https://developers.google.com/web/fundamentals/design-and-ux/responsive/patterns?hl=en`

- **As a web designer, the true test is -- don't just view your site with your browser maximized!**
  - Use browser tools (or just resize your browser) to see what your site looks like at different viewport widths!

- **Final note: don't be too intimidated about responsive design and the CSS you have to write**
  - Most people use a CSS library like Bootstrap and let it do much of the CSS work for you!

Section 3–2

Bootstrap Fundamentals

Bootstrap

- **Bootstrap is a CSS Framework that can (eventually) decrease our development efforts because it provides a ton of boiler-plate styles and components that are ready to use and are responsive to different screen sizes from the start**
  - Take a look at `https://getbootstrap.com/docs/5.2/getting-started/introduction/`
- **It was originally created at Twitter in 2010 and has become one of the most popular CSS Frameworks and open source projects in the world.**
- **There have been several versions of Bootstrap and it has changed sometimes significantly between versions**
  - This lesson will focus on Bootstrap 5.2
  - If you are interested in what Bootstrap 5 has to offer over Bootstrap 4, see: `https://www.sitepoint.com/bootstrap-5-new-features-examples/`
- **When you Google for help on this topic, make sure you are looking at the right version of Bootstrap**

Including Bootstrap in your Project

- **In order to use Bootstrap, you have to include it as well as Popper**
  - Popper is a JavaScript positioning engine that calculates the position of an element so that you can position another element relative to it

- **Some Bootstrap components that require Popper include:**
  - **Alerts** for dismissing
  - **Buttons** for toggling states and checkbox/radio functionality
  - **Carousel** for all slide behaviors, controls, and indicators
  - **Collapse** for toggling visibility of content
  - **Dropdowns** for displaying and positioning (requires Popper)
  - **Modals** for displaying, positioning, and scroll behavior
  - **Navbar** for extending our Collapse plugin to implement responsive behavior
  - **Tooltips** and popovers for displaying and positioning (requires Popper)

- **The two easiest and most common ways to include Bootstrap into your project are:**
  - Including it from a CDN
  - Downloading it from `getbootstrap.com`

Using the jsDelivr CDN (Content Delivery Network)

- **As a developer, the quickest way to start using Bootstrap is by including the .css and .js files into your project using the jsDelivr CDN (content delivery network)**
  - Bootstrap provides detailed instructions on how to include Bootstrap using the CDN in their quick start instructions

Your HTML file will look like the following example when including Bootstrap from the CDN

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <title>Bootstrap demo</title>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
0evHe/X+R7YkIZDRvuzKMRqM+OrBnVFBL6DOitfPri4tjfHxaWutUpFmBp4vmVo
r" crossorigin="anonymous">

  </head>
  <body>
    <h1>Hello, world!</h1>

    <!-- this includes both the Bootstrap and Popper JS -->

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/js/bootstrap.bundle.min.js" integrity="sha384-
pprn3073KE6tl6bjs2QrFaJGz5/SUsLqktiwsUTF55Jfv3qYSDhgCecCxMW52nD
2" crossorigin="anonymous"></script>

  </body>
</html>
```

- **There are other CDNs you will find that can also be used to include Bootstrap, jQuery, and Popper in your project**

Downloading Bootstrap

- **You can also download Bootstrap and manage its availability on your own site**

- **To download Bootstrap, follow these steps:**

  - Visit `https://getbootstrap.com/docs/5.2/getting-started/download/`

- **The main reason for downloading and hosting your own copy of Bootstrap is that your site will continue to work if the CDN is down**

- **In this class, however, you will use the CDN for your lab exercises unless your instructor says otherwise**

Mobile First Design

- **Bootstrap focuses on mobile-first responsive design**
  - Most components are completely responsive with very little effort

- **Bootstrap classes are used to control how our layouts respond to changes in the viewport size**
  - We can apply to our layout allowing our content to be responsive to viewport changes.

- **Media queries create sensible breakpoints ranging from `xs` - `xl` , where:**
  - `xs` (Portrait Phones) - Active when the viewport is >= 0
  - `sm` (Landscape Phones) - Active when the viewport width is >= 576px
  - `md` (Tablets) - Active when the viewport width is >= 768px
  - `lg` (Desktops) - Active when the viewport width is >= 992px
  - `xl` (Large Desktops) - Active when the viewport width is >= 1200px

- **You rely on these breakpoints to control the behavior of containers and the Grid System to ensure that your content always looks good regardless of screen size**

Exercise

Let's get ready to do some Bootstrap coding! In this exercise, you create a `bootstrap-examples` project and configure a home page to use Bootstrap. It is step 1 for other exercises in this module.

Step 1. Create a repo

Go to GitHub and create a repo named `bootstrap-examples`. Now clone it to your local computer.

Step 2. Build the site structure

Add the `css`, `images`, and `scripts` folder.

Add an `index.html` page.

In index.html, create the standard HTML layout including `DOCTYPE`, `html`, `head` and `body` elements.

Add the `meta` tags that Bootstrap expects. Then include the Bootstrap 4.6 framework's CSS and JS files ***from a CDN***.

Step 3. Commit your work

Save and then commit your repo using the message "Added site structure".

Push your repo to GitHub.

Section 3–3

Bootstrap Containers and the Grid System

Bootstrap Containers

- **Containers are layout elements in Bootstrap**
  - They group together elements as well as provide some default padding and margin
  - Containers are responsive to screen size as well

- **The commonly used container classes provided by bootstrap are `.container` and `.container-fluid`**
  - The `.container` class provides a responsive **fixed width container**
    * Its width (`max-width`) changes based on the screen size

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
max-width	100%	540px	720px	960px	1140px

  - The `.container-fluid` class provides a **full width container**, spanning the entire width of the viewport

```
<!-- The most common containers (.container and .container-fluid) -->
<div class="container">Size varies</div>
<div class="container-fluid">Always 100% wide</div>
```

The Bootstrap Grid

- **The Bootstrap grid system lets us use a table-like structure of rows and columns to create responsive layouts for our pages**
  - Unlike traditional tables, making the Bootstrap grid responds to changes in the viewport size is really easy

- **It allows us to make elements appear next to each other when viewing our site on a desktop and stack on top of each other when viewing our site on a mobile device**

Creating a Simple Grid using `row` and `col` Classes

- **The grid should be wrapped in a `container`**
  - In a grid layout, content must be placed within columns and only columns may be immediate children of rows
- **The container contains rows defined using the `row` class**
- **Each row has up to 12 columns**
  - Each column use the `col` class and has horizontal padding (called a gutter)
- **We can use breakpoint classes to make our grid responsive**

Example: Creating a Simple Grid

- **When creating a grid, start with a container that is the parent element for our responsive grid**
  - It controls how much of the viewport the grid takes up

- **Then, start adding rows and columns**

This is a simple grid example with 2 rows each containing 3 columns

```
<div class="container">
  <div class="row">
    <div class="col"> Column 1 </div>
    <div class="col"> Column 2 </div>
    <div class="col"> Column 3 </div>
  </div>
  <div class="row">
    <div class="col"> Column 1 </div>
    <div class="col"> Column 2 </div>
    <div class="col"> Column 3 </div>
  </div>
</div>
```

This would produce a grid similar to the following

Column 1	Column 2	Column 3
Column 1	Column 2	Column 3

Using Column Classes

- Column classes (ex: `col-4`, `col-6`) allow us to control how many columns a particular column can span

This is a simple grid example with 2 rows. The first row has 12 equally spaced columns. The second row uses column classes to make the center column span 6 columns out of the 12

```
<div class="container">
  <div class="row">
    <div class="col"> Column 1 </div>
    <div class="col"> Column 2 </div>
    <div class="col"> Column 3 </div>
    <div class="col"> Column 4 </div>
    <div class="col"> Column 5 </div>
    <div class="col"> Column 6 </div>
    <div class="col"> Column 7 </div>
    <div class="col"> Column 8 </div>
    <div class="col"> Column 9 </div>
    <div class="col"> Column 10 </div>
    <div class="col"> Column 11 </div>
    <div class="col"> Column 12 </div>
  </div>
  <div class="row">
    <div class="col"> Column 1 </div>
    <div class="col-6"> Column 2 </div>
    <div class="col"> Column 3 </div>
  </div>
</div>
```

This would produce a grid similar to the following

Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11	Column 12
Column 1			Column 2						Column 3		

- **Responsive column classes like `.col-sm-6` and `.col-md-4` are also available**
  - It allows us to design mobile pages differently that desktop pages
  - For example: our columns are stacked on mobile and side-by-side on wider screens

```
<div class="container">
  <div class="row">
    <div class="col-md-4"> Column 1 </div>
    <div class="col-md-4"> Column 2 </div>
    <div class="col-md-4"> Column 2 </div>
  </div>
</div>
```

On a medium (tablet) or above, the output would be:

Column 1	Column 2	Column 3

Since xs and sm and not defined, they default to full width and the output appears stacked

Column 1
Column 2
Column 3

- **To see more examples of the grid systems and the options you have, visit:**

  https://getbootstrap.com/docs/5.2/layout/grid/

  https://getbootstrap.com/docs/5.2/layout/grid/#responsive-classes

  https://getbootstrap.com/docs/5.2/layout/grid/#row-columns

Exercise

Let's create a responsive page using Bootstrap! In this exercise, you will create a home page that:

- displays "Bootstrap Examples" in your `<header>`
- uses an HTML5 `<nav>` element to display an anchor with the text "Home" that takes you here. We will add other anchors as time goes on.
- displays 3 paragraphs of lorem ipsum text in an HTML5 `<main>` element using different layouts depending on screen size

Step 1. Pull the most recent copy of the repo.

Use `git pull` to make sure you have the most recent copy of your `bootstrap-examples` repo.

Step 2. Design your responsive index page.

Display "Bootstrap Examples" in your `<header>`

Use an HTML5 `<nav>` element to display an anchor with the text "Home" that takes you here.

Create a `<main>` element. The main will contain 3 paragraphs of lorem ipsum text that should appear side-by-side on medium (md) devices and above. To do this, you will need to use the Bootstrap grid system.

Test your page and work out the kinks.

After your page looks okay in a full screen browser, resize the browser to be narrow and watch the paragraphs "stack" within a smaller display area.

Step 3: Save everything

Save and then commit your repo using the message "Designed index page".

Push your repo to GitHub.

Section 3–4

Bootstrap Utility Classes

Bootstrap Utility Classes

- **Bootstrap provides several categories of utility classes that can help us format our content, including:**
  - Colors
  - Borders
  - Spacing
  - Position

- **They help us with add borders to our elements, change colors of elements on our pages, control spacing through margin and padding, and even things like positioning our elements where we want them without having to write our own custom CSS**

- **In this section, we will cover some of the most common utility classes provided**
  - For a complete list of Bootstrap utility classes, visit: `https://getbootstrap.com/docs/5.2/utilities`

Color Utility Classes

- **We can use the color utility classes to change the color of various elements**
  - `.text-*` sets the text color and `.bg-*` sets the background color

- **Colors and the names of their classes available to us through bootstrap include:**

primary	secondary	success	danger
info	light	dark	white

Examples of the color utilities being used for backgrounds and text

```
<!-- Examples of the text color utility classes being used to change the color of the text
in a paragraph element -->

<p class="text-primary">.text-primary</p>

<p class="text-secondary">.text-secondary</p>

<p class="text-success">.text-success</p>

<p class="text-danger">.text-danger</p>

<p class="text-warning">.text-warning</p>

<p class="text-info">.text-info</p>


<!-- Examples of the background color utility classes being used to change the background
color of a div element -->

<div class="bg-primary">.bg-primary</div>

<div class="bg-secondary">.bg-secondary</div>

<div class="bg-success">.bg-success</div>

<div class="bg-danger">.bg-danger</div>

<div class="bg-warning ">.bg-warning</div>

<div class="bg-info">.bg-info</div>

<div class="bg-light">.bg-light</div>
```

```
<!-- text-white added as well so the text is visible in the div -->
<div class="bg-dark text-white">.bg-dark</div>
```

- **Some built in Bootstrap Components also leverage the color classes to change how they appear**

- **For more information on the Bootstrap color utility classes, visit:** `https://getbootstrap.com/docs/5.2/utilities/colors/`

Border Utility Classes

- **We can use the border utility classes to add or remove an element's borders**
  - `border` - Adds a 1px border to all sides of an element
  - `border-top` - Adds a 1px border to the top of an element
  - `border-right` - Adds a 1px border to the right side of an element
  - `border-bottom` - Adds a 1px border to the bottom of an element
  - `border-left` - Adds a 1px border to left side of an element

Examples of the border utilities in action

```
<!-- default use of border utility classes -->
<span class="border">Border on all sides</span>
<span class="border-top">Top border only</span>
<span class="border-right">Right border only</span>
<span class="border-bottom">Bottom border only</span>
<span class="border-left">Left border only</span>

<!-- combining border classes -->
<span class="border-left border-right">Left and Right border</span>

<!-- colored borders border classes using border-primary -->
<span class="border-left border-right border-primary">Left and Right border</span>
```

- **Other classes can be used to round the border's corners**

- **For more information on the Bootstrap border utility classes, visit:** `https://getbootstrap.com/docs/5.2/utilities/borders/`

Spacing Utility Classes

- **The spacing utility classes control the padding and margin for elements on your page**
  - `p` is used for padding and `m` is used for margin
  - They can be modified with what Bootstrap refers to as **sides** and **size**.

- **Sides is one of:**
  - `t` - for classes that set `margin-top` or `padding-top` `(ex: mt or pt)`
  - `b` - for classes that set `margin-bottom` or `padding-bottom` `(ex: mb or pb)`
  - `s` - for classes that set `margin-start` or `padding-start` (for English, start is left)
  - `e` - for classes that set `margin-end` or `padding-end` (for English, end is right)
  - `x` - for classes that set both `*-left` and `*-right` `(ex: mx or px)`
  - `y` - for classes that set both `*-top` and `*-bottom` `(ex: my or py)`
  - blank - for classes that set a `margin` or `padding` on all 4 sides of the element `(ex: m or p)`

- **Size is one of:**
  - `0` - for classes that eliminate the `margin` or `padding` by setting it to `0` `(ex: m-0 or pt-0)`
  - `1` - (by default) for classes that set the `margin` or `padding` to `1em * .25` `(ex: mb-1 or pl-2)`
  - `2` - (by default) for classes that set the `margin` or `padding` to `1em * .5`
  - `3` - (by default) for classes that set the `margin` or `padding` to `1em`
  - `4` - (by default) for classes that set the `margin` or `padding` to `1em * 1.5`
  - `5` - (by default) for classes that set the `margin` or `padding` to `1em * 3`

- `auto` - for classes that set the `margin` to auto

Examples of the spacing utilities in action

```
<!-- examples of padding utility classes combined with borders to help visualize the
differences -->

<span class="p-5 border">set the padding to 5 (1rem * 3) on all sides of the element</span>

<span class="px-3 border">Set the padding to 1rem on the left and right side of the
element</span>

<span class="pt-1 border">Set the padding to (1rem *.25) on the top of the element</span>


<!-- examples of margin utility classes combined with borders to help visualize the
differences -->

<span class="m-5 border">set the margin to 5 (1rem * 3) on all sides of the element</span>

<span class="my-3 border">Set the margin to 1rem on the top and bottom of the
element</span>

<span class="mb-1 border">Set the margin to (1rem *.25) on the bottom of the element</span>


<!-- combining padding and margin classes -->

<span class="m-5 p-5 border">set the margin and padding to 5 (1rem * 3) on all sides of the
element</span>

<span class="my-3 py-3 border">Set the margin and padding to 1rem on the top and bottom of
the element</span>

<span class="mb-1 pb-1 border">Set the margin and padding to (1rem *.25) on the bottom of
the element</span>
```

- **For more information,**
  visit: `https://getbootstrap.com/docs/5.2/utilities/spacing/`

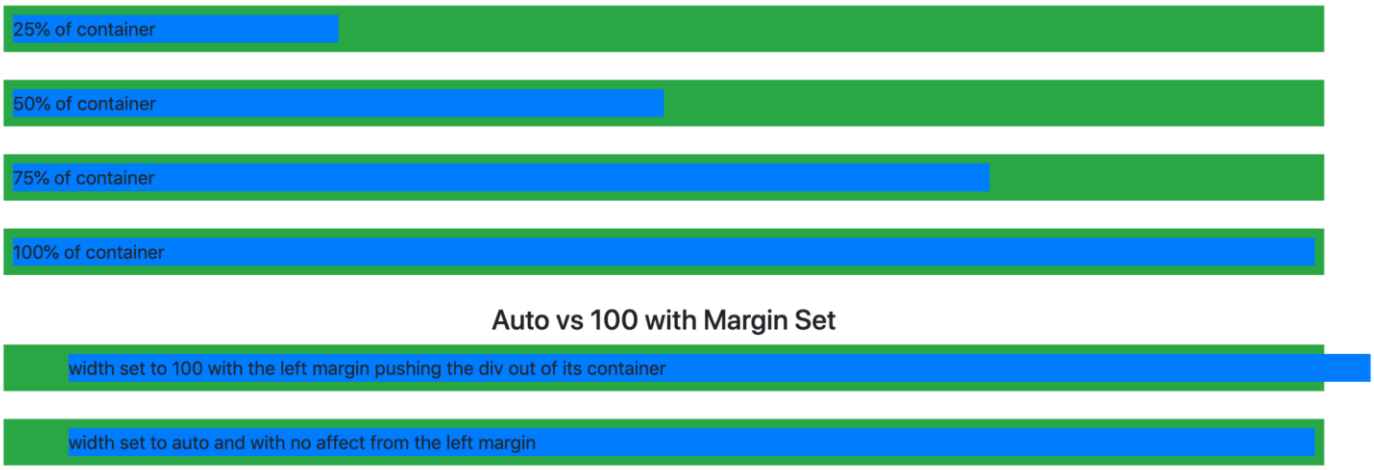
Sizing Utility Classes

- **The sizing utility classes control the width and height of elements on your page**
  - Sizing utility classes support `25%`, `50%`, `75%`, `100%`, and `auto` by default

- **Width utility classes are:**
  - `w-25` - 25% of the parent containers available horizontal space
  - `w-50` - 50% of the parent containers available horizontal space
  - `w-75` - 75% of the parent containers available horizontal space
  - `w-100` - 100% of the parent containers available horizontal space (can be affected by additional margin and appear to extend past its container)
  - `w-auto` - default behavior of a block-level element taking up all the available horizontal space in its container but not affected by an additional margin

- **Height utility classes are:**
  - `h-25` - 25% of the parent containers available vertical space
  - `h-50` - 50% of the parent containers available vertical space
  - `h-75` - 75% of the parent containers available vertical space
  - `h-100` - 100% of the parent containers available vertical space (can be affected by additional margin and appear to extend past its container)
  - `h-auto` - default behavior of a block level-element taking up all the available vertical space in its container but not affected by an additional margin

A visual representation of the `width` sizing classes highlighting `auto` vs `100`



- **For more information on the Bootstrap sizing utility classes, visit:** `https://getbootstrap.com/docs/5.2/utilities/sizing/`

Text Utility Classes

- **You can specify font weight and italics**

```
<p class="font-weight-bold">Bold text.</p>
<p class="font-weight-bolder">Bolder weight text (relative to the parent element).</p>
<p class="font-weight-normal">Normal weight text.</p>
<p class="font-weight-light">Light weight text.</p>
<p class="font-weight-lighter">Lighter weight text (relative to the parent element).</p>
<p class="font-italic">Italic text.</p>
```

Bold text.

Bolder weight text (relative to the parent element).

Normal weight text.

Light weight text.

Lighter weight text (relative to the parent element).

Italic text.

- **You can transform text**
  - `text-capitalize` only capitalizes the first letter of each word

```
<p class="text-lowercase">Lowercased text.</p>
<p class="text-uppercase">Uppercased text.</p>
<p class="text-capitalize">CapiTaliZed text.</p>
```

lowercased text.

UPPERCASED TEXT.

CapiTaliZed Text.

- **Text alignment classes let you justify text**

```
<p class="text-left">Left aligned text.</p>
<p class="text-center">Center aligned text.</p>
<p class="text-right">Right aligned text.</p>

<p class="text-sm-left">Left aligned text on viewports sized SM (small) or wider.</p>
<p class="text-md-left">Left aligned text on viewports sized MD (medium) or wider.</p>
<p class="text-lg-left">Left aligned text on viewports sized LG (large) or wider.</p>
<p class="text-xl-left">Left aligned text on viewports sized XL (extra-large) or wider.</p>
```

  - Remember that responsive design with Bootstrap means that you can include more than one CSS class

```
<p class="text-md-center text-sm-left">Usually centered, but left-aligned on small
viewports</p>
```

- **You can use the text classes to remove a text decoration from an `<a>` tag with the `.text-decoration-none` class**

```
<a href="https://getbootstrap.com" class="text-decoration-none">Bootstrap (not
underlined!)</a>
```

Other Useful Styling Classes

- **There are so many other useful Bootstrap styling classes**
  - You really learn Bootstrap by using it for many, many months (years?)
  - Once you understand Bootstrap basics, you Google

- **Need to style an HTML table?**
  - Visit Bootstrap Tables
    `https://getbootstrap.com/docs/5.2/content/tables/`

- **Need to style images?**
  - Visit Bootstrap Images
    `https://getbootstrap.com/docs/5.2/content/images/`

- **Need to style HTML headings and lists?**
  - Visit Bootstrap Typography
    `https://getbootstrap.com/docs/5.2/content/typography/`

- **Need to style an HTML form?**
  - Visit Bootstrap Forms
    `https://getbootstrap.com/docs/5.2/components/forms/`

- **Need a cool icon in your site like a list or a phone?**
  - Visit Bootstrap Icons
    `https://icons.getbootstrap.com/#install`

- **Need more cool (awesome) icons?**
  - Visit Fontawesome
    `https://fontawesome.com/`

Exercise - Styling

In this exercise, you will add some simple styling your `bootstrap-examples` project.

Step 1. Pull the most recent copy of the repo.

Use git pull to make sure you have the most recent copy of your `bootstrap-examples` repo.

Step 2. Add some styling.

Using Bootstrap classes, change the text color in your second paragraph to blue.

Also add a blue border to the second paragraph and set padding such that there is definitely some space between the text and the border.

Center the text in all three paragraphs on larger devices, but left align it on phones.

Test your page.

Step 3: Save everything

Save and then commit your repo using the message "Styled the index page".

Push your repo to GitHub.

Exercise - Bootstrap Forms

In this exercise, you will add a new page to the `bootstrap-examples` project that allows the user to login. (Note: there is currently NO login behavior yet)

Because this is a new feature, we will create the code in a new branch and then merge it with main branch when we get it working.

Step 1. Pull the most recent copy of the repo.

Use `git pull` to make sure you have the most recent copy of your `bootstrap-examples` repo.

Step 2. Create the new branch.

Use git status to make sure the main branch has no uncommitted changes. Then create a new branch named "LoginBranch". Switch to LoginBranch.

Step 3. Create the login page.

Create a new page named `login.html`. Make sure to add support for Bootstrap.

Now, do some Googling and find a login form example that is styled using Bootstrap. Copy the code to your page.

Make changes as needed to make the page "fit in" with your needs.

View the page.

Step 4. Provide a link to the page.

In `index.html`, add a second link near the Home link that allows a user to navigate to your login page.

Test the link.

Save and commit your changes.

Step 5. Merge your new feature.

Switch back to the main branch.

Merge your LoginBranch and remember to use the -m flag to specify a message.

Use git status to make sure everything was successful.

Now push your code to GitHub.

Section 3–5

Bootstrap Components

Working with Bootstrap Components

- **Bootstrap comes bundled with a variety of *components* that can help speed up the development process**
  - Components are ready to go design elements you can add to your webpage
- **Most components look great no matter the screen size or device used to view them**
  - They even come bundled with a lot of ready-made functionality
- **For a full list of available components, visit** `https://getbootstrap.com/docs/5.2/components`
  - ***Look at the list on the left and explore!***
- **Commonly used Bootstrap components include (but are not limited to):**
  - **Buttons** - `https://getbootstrap.com/docs/5.2/components/buttons/`
  - **Cards** - `https://getbootstrap.com/docs/5.2/components/card/`
  - **Carousel** - `https://getbootstrap.com/docs/5.2/components/carousel/`
  - **Forms** - `https://getbootstrap.com/docs/5.2/components/forms/`
  - **Navbar** - `https://getbootstrap.com/docs/5.2/components/navbar/`
  - **Modal** - `https://getbootstrap.com/docs/5.2/components/modal/`
- **Note: If you see a reference to a Bootstrap Jumbotron component when you Google, it was removed from Bootstrap in version 5**
  - You can recreate it in Bootstrap 5 using the styling shown here: `https://www.w3schools.com/bootstrap5/bootstrap_jumbotron.php`

Exercise - Bootstrap Carousel

In this exercise, you will add a page to your `bootstrap-examples` project that uses a Bootstrap carousel to display a slideshow of FOUR images on your page. You typically want all of the images to be of the same size.

Because this is a new feature, we will create the code in a new branch and then merge it with main branch when we get it working.

Step 1. Pull the most recent copy of the repo.

Use `git pull` to make sure you have the most recent copy of your `bootstrap-examples` repo.

Step 2. Create the new branch.

Use git status to make sure the main branch has no uncommitted changes. Then create a new branch named "CarouselBranch". Switch to CarouselBranch.

Step 3. Create the page and a link to it.

Begin by adding a new page to your project named `slideshow.html`. Then, on the home page, add a new link to this page. Test the link.

Step 4. Create the carousel.

Now find four images you want to add to your carousel and add them to your images folder. Make sure to size them so they are the same size.

To learn more about the carousel, go to:
`https://getbootstrap.com/docs/5.2/components/carousel/`

When we use Bootstrap components like this, we don't try to write them from scratch. Instead, we COPY the HTML example from `getbootstrap.com` or some other examples website and "tweak it" to work with our example.

In the code below, we would only change the contents of attributes whose value is "…"

```
<div id="carouselExampleSlidesOnly" class="carousel slide" data-
bs-ride="carousel">
  <div class="carousel-inner">
    <div class="carousel-item active">
      <img src="..." class="d-block w-100" alt="...">
```

```
    </div>

    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
  </div>
</div>
```

What would you do if you needed to display four images?

Before you commit to the code example above as your starting point, notice that `getbootstrap.com` tells you there are carousels that:

- have images only
- have controls
- have indicators

Once you figure out which type of carousel you want, create your `slideshow.html` page.

Test it.

Step 5. Save changes and merge the branch.

Commit your changes to the CarouselBranch.

Now, switch to the main branch and merge in CarouselBranch. Remember to use the -m flag to add a message.

Use git status to make sure everything is okay.

Finally, push your code to GitHub.

What Do I Do Now?

- **How do you get good with Bootstrap?**

- **You need to know:**
  - (1) WHAT types of things can you do with Bootstrap?
  - (2) HOW do you do those things?

- **The answer to both is Google and explore**
  - `https://getbootstrap.com/docs/5.2/examples/` shows examples of nice Bootstrap designs and you can right-click on them and choose Inspect to see the HTML/CSS
  - `https://startbootstrap.com/snippets` shows cool designs that you can use
  - There are really too many to list… just Google "examples of bootstrap 5" and make sure you look on pages 2 and 3 of the results too!

- **Basic tutorial-types examples are found at `getbootstrap.com` and other places, including:**
  - `https://www.w3schools.com/bootstrap/bootstrap_examples.asp` has more basic examples, shows the code, and explains it
  - `https://www.tutorialrepublic.com/twitter-bootstrap-examples.php` also has basic examples that make it easy to see what you need

Exercise - Bootstrap Card

In this exercise, you will add a set of Bootstrap cards to your `bootstrap-examples` project. A Bootstrap card is a container that includes options for headers, content, and footers, as well as contextual background colors and other display options.

Because this is a new feature, we will create the code in a new branch and then merge it with main branch when we get it working.

Step 1. Pull the most recent copy of the repo.

Use `git pull` to make sure you have the most recent copy of your `bootstrap-examples` repo.

Step 2. Create the new branch.

Use git status to make sure the main branch has no uncommitted changes. Then create a new branch named "AnimalCardsBranch". Switch to AnimalCardsBranch.

Step 3. Create the page and link to it.

Begin by adding a new page to your project named `cards.html`. Then, on the home page, add a new link to this page. Test the link.

Step 4. Design the page.

Your Bootstrap cards will show information about 4 different animals (think about something like a zoo or pet info card). You will need to find four small images of your animals, download them to your images folder, and make them the same size.

To learn more about Bootstrap cards, go to:
`https://getbootstrap.com/docs/5.2/components/card/`

Again, when we use Bootstrap components like this, we don't try to write them from scratch. Instead, we COPY the HTML example from `getbootstrap.com` or some other examples website and "tweak it" to work with our example.

Find the example on getbootstrap.com titled "Images". It should resemble:

```
<div class="card" style="width: 18rem;">
```

```
  <img src="..." class="card-img-top" alt="...">
  <div class="card-body">
    <p class="card-text">Some quick example text to build on the
card title and make up the bulk of the card's content.</p>
  </div>
</div>
```

Do you see where you would add your image attributes? What about where you would add your textual description?

Use this style of Bootstrap card to display pictures and information about two of your four animals. Test your page.

Now find the example on `getbootstrap.com` titled "Horizontal". It should resemble:

```
<div class="card mb-3" style="max-width: 540px;">
  <div class="row g-0">
    <div class="col-md-4">
      <img src="..." class="img-fluid rounded-start" alt="...">
    </div>
    <div class="col-md-8">
      <div class="card-body">
        <h5 class="card-title">Card title</h5>
        <p class="card-text">This is a wider card with supporting
text below as a natural lead-in to additional content. This
content is a little bit longer.</p>
        <p class="card-text"><small class="text-muted">Last
updated 3 mins ago</small></p>
      </div>
    </div>
  </div>
</div>
```

Note the card sets a max width of 540px. (This might have been better done using a CSS class.) The card's image takes up the first 1/3 of the row and the card's body takes up the last 2/3 of the row on medium (md) devices and higher.

Do you see where you would add your image attributes? What about where you would add a title and textual description?

Use this style of Bootstrap card to display information about your other two animals. Test your page.

Step 5. Save changes and merge the branch.

Commit your changes to the AnimalCardsBranch.

Now, switch to the main branch and merge in AnimalCardsBranch. Remember to use the -m flag to add a message.

Use git status to make sure everything is okay.

Finally, push your code to GitHub.

Exercise - Bootstrap Navbar

In this exercise, you will add a Bootstrap navbar to your `bootstrap-examples` project. A Bootstrap navbar is a responsive menu that appears full-width in larger devices and collapses into a "hamburger" icon and dropdown on smaller devices

Because this is a new feature, we will create the code in a new branch and then merge it with main branch when we get it working.

Step 1. Create the new branch.

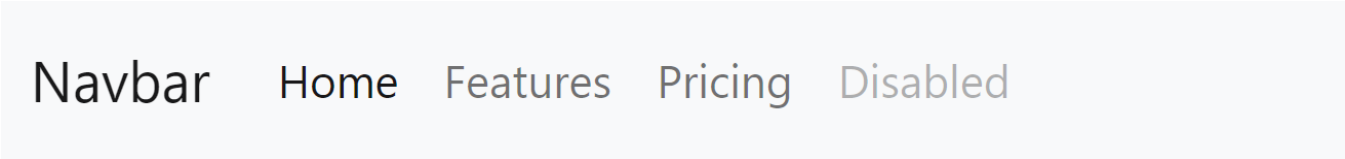
Use git status to make sure the main branch has no uncommitted changes. Then create a new branch named "NavbarBranch". Switch to NavbarBranch.

Step 2. Replace navigation with a Bootstrap navbar

`To learn more about Bootstrap navbars, go to:`
`https://getbootstrap.com/docs/5.2/components/card/`

For the last time, when we use Bootstrap components like this, we don't try to write them from scratch. Instead, we COPY the HTML example from `getbootstrap.com` or some other examples website and "tweak it" to work with our example.

The following is some code extracted from that website and what it looks like before it collapses.




```
<nav class="navbar navbar-expand-lg bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarNav" aria-
controls="navbarNav" aria-expanded="false" aria-label="Toggle
navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item">
```

```
          <a class="nav-link active" aria-current="page"
href="#">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Features</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Pricing</a>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled">Disabled</a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

Do you see where you would change the wording for your "brand" (in the example it is the word "Navbar" in an anchor? What about where you would add your specific links to other pages?

Replace the links on your index.html page with a Bootstrap navbar. Test your page.

Step3. Save changes and merge the branch.

Commit your changes to the NavbarBranch.

Now, switch to the main branch and merge in NavbarBranch. Remember to use the -m flag to add a message.

Use git status to make sure everything is okay.

Finally, push your code to GitHub.

(Challenge) Exercise - Media Queries

In this exercise, you will add a banner image to your home page below the navbar and above the paragraphs.

You can work directly in the main branch in this exercise.

Step 1: Add a banner

Find a very wide image that you can use as a banner on the index page below the header text and above the navigation area.

View the page on a maximized browser.

Now reside the browser to be the side of a phone. The image may be fluff that makes it difficult to see important content.

Step 2: Use a CSS3 media query to hide the banner on small devices

Write a CSS3 media query to hide the banner image on small devices.

View the page. Then resize the browser so that it is very narrow. Did the image go away?

Step 3: Save everything

Save and then commit your repo using the message "Hid banner on index page on small devices".

Push your repo to GitHub.