HTML stands for Hyper Text Markup Language. It is used to give websites structure with text, links, images, and other fundamental elements.

CSS stands for Cascading Style Sheets. It is used to change the appearance of HTML elements.

# Site Structure

<h1>You're Building a Website!</h1>

1. All HTML elements begin with an *opening tag*. In this case, the opening tag is <h1>.
2. Most elements require a *closing tag*, denoted by a /. In this case, the closing tag is </h1>.
3. The website user only sees the content between the opening and closing tags.

Headings are a frequently used HTML element. You can think of them like headlines in a newspaper. Your eyes may notice headings first because the words are large and bold compared to other text on the webpage.

There are six heading elements: h1, h2, h3, h4, h5, and h6. h1 is the largest heading and h6 is the smallest.

The HTML paragraph element, p, is used to hold one or more sentences, just like paragraphs in an essay or a book.

<p>Paragraph text here</p>

Let's use a paragraph element to add the company description.

What if you wanted to link users to a different webpage? The HTML anchorelement makes it possible to do this with a single click.

<a href="http://google.com"> Click here for Google!</a>

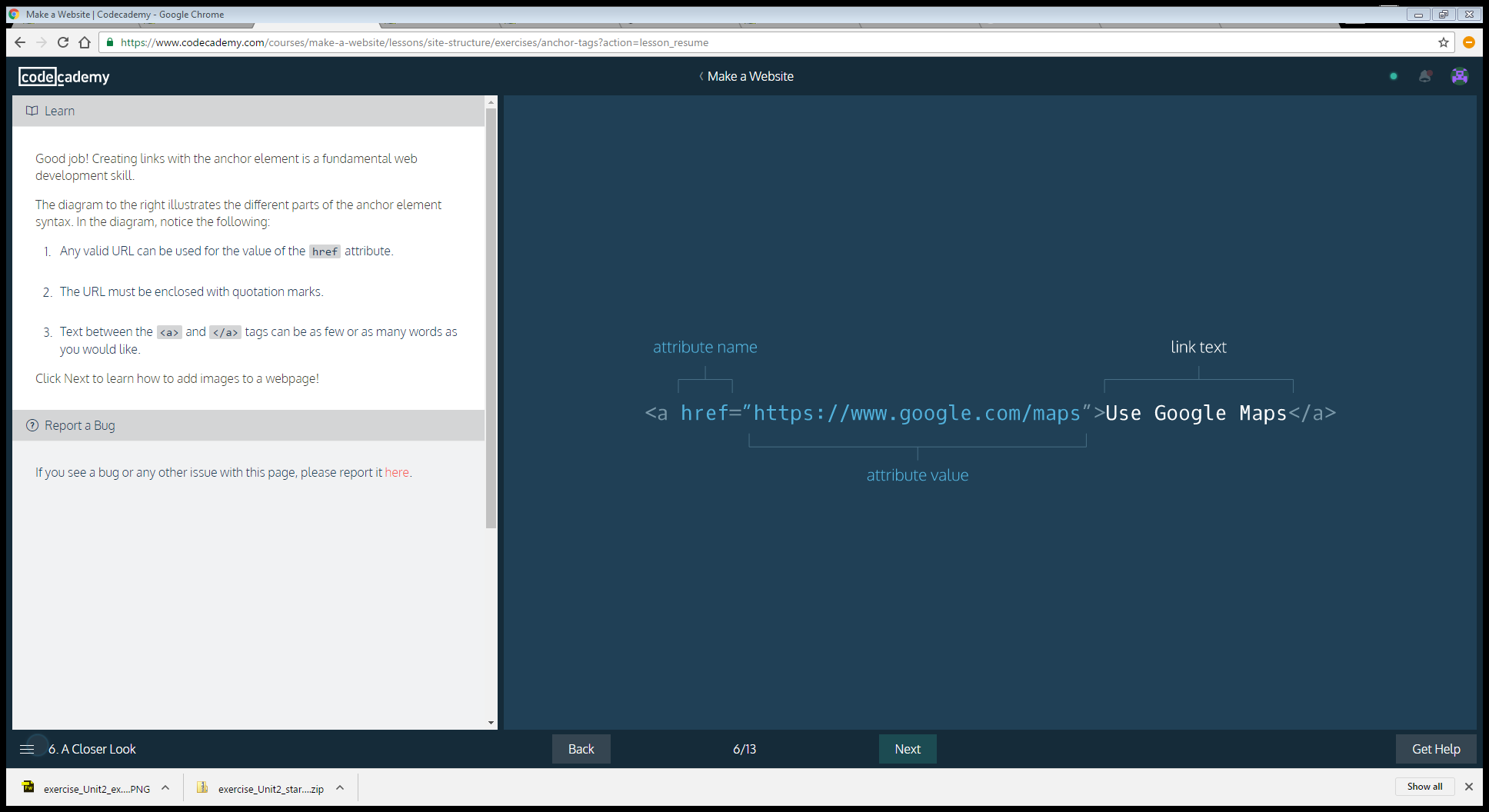
Anchor elements use an attribute to link users to websites. Attributes customize the behavior or appearance of HTML elements. Anchor elements use the href attribute, which specifies the webpage to link to. In the example above, the text "Click here for Google!" links to http://google.com.

**IMPORTANT**: Web addresses, such as http://google.com, have a technical name: URL.

Good job! Creating links with the anchor element is a fundamental web development skill.

The diagram to the right illustrates the different parts of the anchor element syntax. In the diagram, notice the following:

1. Any valid URL can be used for the value of the href attribute.
2. The URL must be enclosed with quotation marks.
3. Text between the <a> and </a> tags can be as few or as many words as you would like.



To add images to a webpage, use the HTML image element:

<img src="https://s3.amazonaws.com/codecademy-content/projects/make-a-website/lesson-1/bikes1.jpg"/>

Just like websites have URLs, images on the web also have URLs. Image URLs typically end with the .jpg or .png file extension. The src attribute sets the source for an image element.

Image elements are self-closing, which means they do not need a closing tag.

The photo makes the webpage much cooler. What's cooler than a photo? A video!

The HTML *video* element can add video to a webpage.

<video width="320" height="240" controls>

<source src="video-url.mp4" type="video/mp4">

</video>

The video element uses a number of attributes. Let's take a look at them:

1. width and height: Set the size of the screen that displays the video.
2. controls: Adds play, pause and volume control.
3. source src: Sets the URL of the video to play.
4. type: Specifies different video formats.

Another essential HTML element is the *unordered list*. Items in an unordered list are referred to as *list items*. Each item is bulleted, not numbered. For example:

* A list item
* A second list item
* A third list item

The HTML code for the list above:

<ul>

<li>A list item</li>

<li>A second list item</li>

<li>A third list item</li>

</ul>

About unordered lists:

1. ul tags create the unordered list.
2. li tags contain each list item.

Unordered list elements can be used to organize content on a webpage in a number of ways. Below we will use one to organize our website's navigation menu, sometimes called a *navbar*.

With the video and unordered list elements, you may have noticed something interesting: these HTML elements had other HTML elements nested inside of them.

For example, in unordered lists, li elements are nested inside the ul.

<ul>

<li>First item</li>

<li>Second item</li>

</ul>

Web developers refer to the enclosing element as the parent element and the enclosed elements as children.

Referring to HTML elements as parents and children may sound funny, but it's a core web development concept. The web browser also knows about these parent/child relationships, which will be important as we explore CSS in the next lesson.

Now that we know about HTML element nesting and parent/child relationships, let's see another way these concepts are applied on a real-life webpage.

Div elements divide your page by enclosing other elements. These enclosed groups of elements can then be organized, moved and styled independently from one another.

Div elements are often used with the class attribute. Here's an example:

<div class="main">

...

</div>

**Note**: The ... indicates where other HTML elements would normally be enclosed by the div.

Below, we will use divs to divide our page into three separate parts.

The last HTML elements we will explore are involved in *metadata* processes. You can think of these elements as the "brains" of a webpage because they communicate vital information to the web browser, but are not visible to a webpage visitor.

1. <!DOCTYPE html>: Tells the web browser to expect an HTML document.
2. <html>...</html>: The root of the HTML document and parent of all other HTML elements on the webpage.
3. <head>...</head>: Enclose other metadata about the site, such as its title.
4. <title>...</title>: Contains the site's title, which is one way users can find your site through a search engine, like Google.
5. <meta charset="utf-8"/>: Tells the web browser which character set to use. In this case, the character set is "utf-8".

# A Closer Look at CSS

Take a look at the web browser. Technically, this is the same site you saw in the previous exercise. It contains the exact same HTML elements and content as before. What's missing? The CSS that was used to previously style the site has been *unlinked*.

The HTML *link* element links a CSS file to an HTML file so that CSS styling can be applied. Here's an example of the link element:

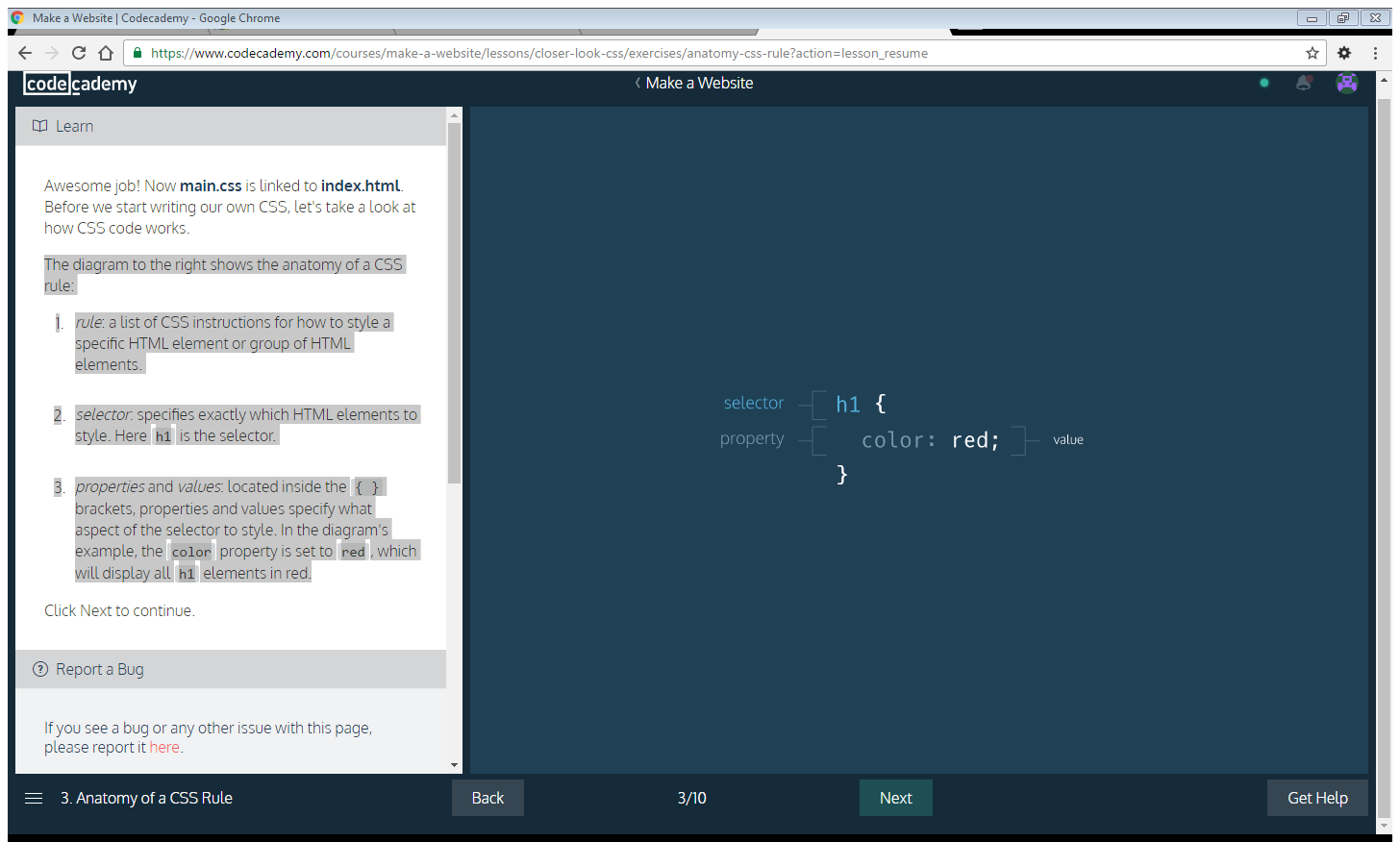
<link rel="stylesheet" type="text/css" href="main.css"/>

About *link*:

1. The link element uses three attributes:  
     
   -rel: Specifies the *relationship* between the current file and the file being linked to: in this case, the rel attribute is "stylesheet".  
     
   -type: Specifies the type of file linked to.  
     
   -href: Provides the URL of the file being linked to.
2. Like the HTML image element, link is self-closing. It does not need a closing tag.
3. In the example above, **main.css** is an external style sheet. Using external stylesheets is one of the most popular ways to write CSS. Inline CSS is another method.

The diagram to the right shows the anatomy of a CSS rule:

1. *rule*: a list of CSS instructions for how to style a specific HTML element or group of HTML elements.
2. *selector*: specifies exactly which HTML elements to style. Here h1 is the selector.
3. *properties* and *values*: located inside the { } brackets, properties and values specify what aspect of the selector to style. In the diagram's example, the color property is set to red, which will display all h1elements in red.



One of the most effective ways to enhance the look and feel of a website is by changing the font. In CSS, font is managed using the font-family property:

h1 { font-family: Georgia, serif; }

Above, the font-family property of the h1 selector is set to the value of Georgia, with serif as a fallback font. Fallback fonts are included in case a visitor's web browser does not support the first font. Sometimes, more than one fallback font is included.

The new font looks great! However, black is not your only option when it comes to font color.

In CSS, the *color* property sets the color of a CSS selector's font:

h1 { color: red; }

CSS comes equipped with 140 *named colors*, such as red, used above. For many situations, these named colors will suffice. However, web developers who want to get even more exact with their color choices can use *hexadecimal*and *RGB* color values.

1. *Hexadecimal color (#RRGGBB)*: Hexadecimal values that represent mixtures of red, green and blue. For example, red can be expressed with the hexadecimal value of #FF0000: the value ff represents red, 00represents green, and 00 represents blue.
2. *RGB (Red, Green, Blue) colors*: Color created by three numbers representing red, green, and blue. When mixed together, the three values create a specific color. For example: purple can be represented asrgb(128,0,128).

Up to this point, we've used CSS to select an HTML element by its tag name only. However, we can use class selectors to target classes of HTML elements.

Consider the HTML below:

<div class="header"> <h2>Heading</h2> <p>Paragraph</p> </div>

Here, the div is the parent element and the h2 and p are children. CSS styles applied to the header class selector will automatically apply to the h2and the p.

[Here's a refresher](https://www.codecademy.com/en/courses/make-a-website/lessons/site-structure/exercises/parent-child) on parent and child elements.

In CSS, class selectors can be identified by a dot . followed by the class name, as seen below:

.header { color: #ffffff; }

As a result of this code, child elements of divs with the header class will have a font color of #ffffff (white).

Below, we will use a CSS class selector to style more than one HTML element at once.

The size of text has an impact on how users experience a website. The *font-size*property sets the size of an HTML element's text.

h1 { font-size: 60px; }

In the CSS above, font-size is set to 60px. In CSS, size can be assigned in*pixels* (px), *rems*, or *ems*.

1. *pixel (px)*: Standard unit of measurement for sizing fonts and other HTML elements.
2. *rem*: Represents the default font size for the web browser. Rems can be used to ensure that HTML elements scale in proportion to each other on various web browsers and screen sizes. On most web browsers, 1rem is equivalent to 16px, 2rem is equivalent to 32px (a doubling), 3rem is equivalent to 48px (a tripling) and so on.
3. *em*: A relative value that changes in proportion to the size of the parent element. For example, if a parent element has font-size: 20px;, child elements with font-size: 1em; would be equivalent to 20px. Child elements with font-size: 0.5em; would be equivalent to 10px (a halving) and so on.

In CSS, the background-image property sets a background image of your choice for a given selector, as seen below.

.hero { background-image: url("https://s3.amazonaws.com/codecademy-content/projects/make-a-website/lesson-2/bg.jpg"); }

The CSS rule above assigns the image hosted at

https://s3.amazonaws.com/codecademy-content/projects/make-a-website/lesson-2/bg.jpg

to elements that have a class attribute set to hero.

To control the size of the chosen background image, use the propertybackground-size as seen below:

.hero { background-image: url("https://s3.amazonaws.com/codecademy-content/projects/make-a-website/lesson-2/bg.jpg"); background-size: cover; }

Here, we have specified that we want the image to completely cover elements with the .hero class.

A previous exercise introduced CSS class selectors to style HTML elements of specific classes. What if a webpage design calls for an individual page element to be styled uniquely, but all other elements of that kind to be styled a different way?

For example, to style one anchor element differently than all the others on a webpage, you could use the HTML *id attribute*:

<a id="learn-code" href="https://www.codecademy.com">Click here to learn to code!</a>

Then in the CSS file, you would create a rule for the id selector, using a #symbol:

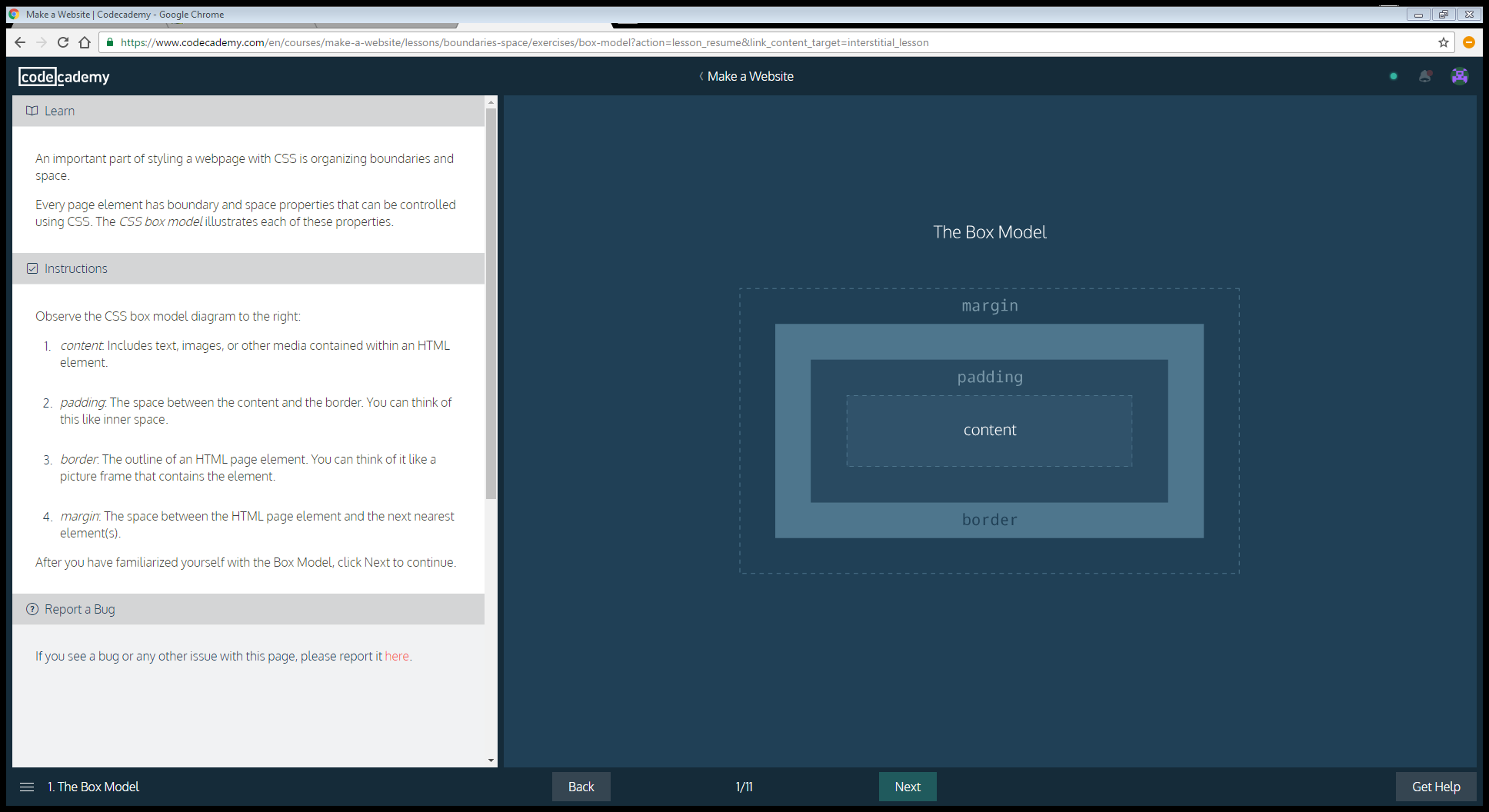
#learn-code { color: #2e69a3; }

About using id:

1. An id attribute can only be used once because the id is unique to the element it is assigned to.
2. Ids have greater specificity than classes. If an HTML element is using both id and class attributes, the CSS rule for the id will take precedence over that of the class.

Below we will use an id selector to style a single HTML element differently than others of that kind on the webpage.

# Boundaries and Space



Observe the CSS box model diagram to the right:

1. *content*: Includes text, images, or other media contained within an HTML element.
2. *padding*: The space between the content and the border. You can think of this like inner space.
3. *border*: The outline of an HTML page element. You can think of it like a picture frame that contains the element.
4. *margin*: The space between the HTML page element and the next nearest element(s).

After you have familiarized yourself with the Box Model, click Next to continue.

The *border* property can be used to visually define a page element's outer edge.

In CSS, the *border* property's value requires three parts:

1. *thickness*: Sets the thickness of the border, using pixels, ems, or rems.
2. *type*: Sets the border type. Common options are solid, dotted, anddashed. There are many others.
3. *color*: sets the border's color, using named colors, HEX, or RGB values.

The CSS below gives a paragraph element a solid black border that is 2 pixels thick:

p { border: 2px solid black; }

The CSS padding property controls the empty space between the page element's content and its border. Increasing a page element's padding makes the space around the content more spacious, while decreasing it makes the space more compact.

p { padding: 20px; }

The CSS above would give paragraph elements a padding of 20px.

The CSS margin property controls the space between different HTML elements on a webpage. Use margin to bring page elements closer together or to move them further apart.

The CSS below ensures 2rems of space between elements with the class answer and surrounding page elements.

.answer { margin: 2rem; }

The margin property creates space on all sides of a page element. It's also possible to set separate margin spacings on each side of an element.

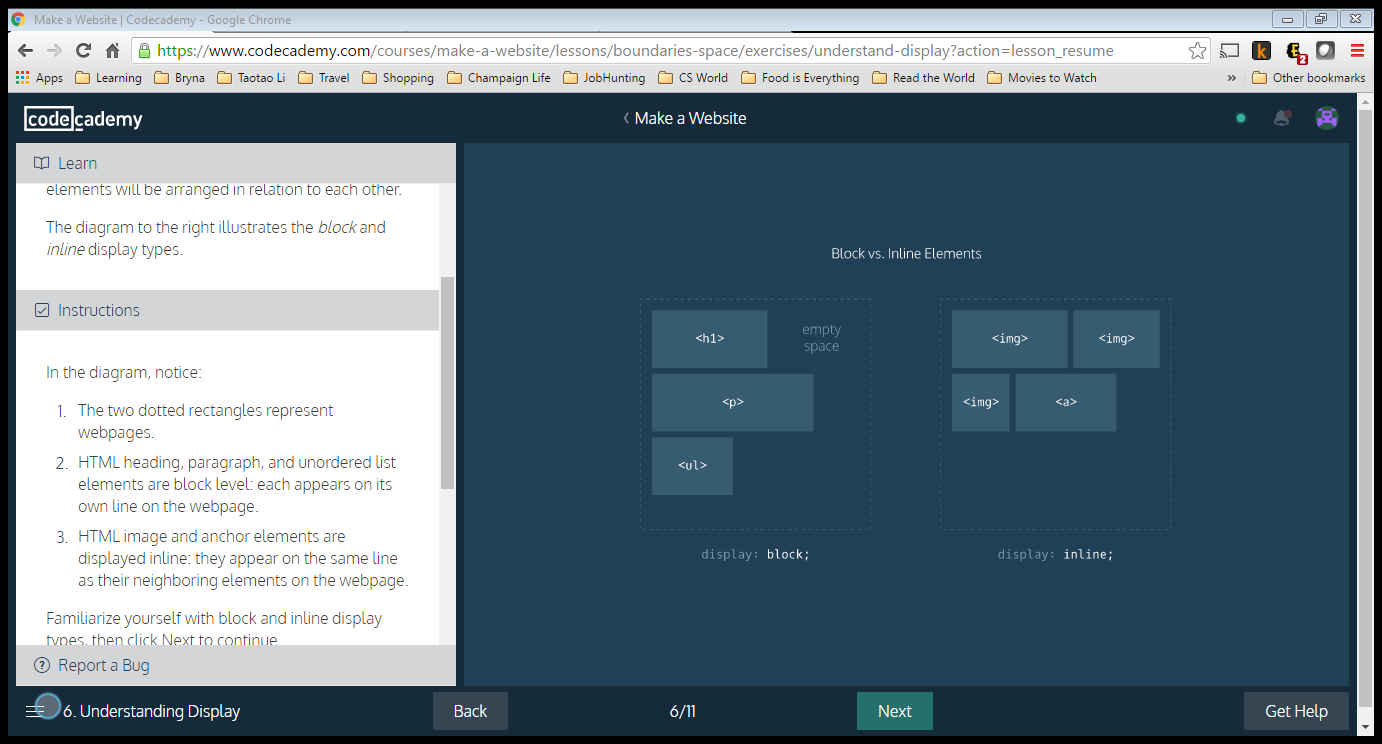
Additional margin properties:

1. *margin-top*: Sets the top margin.
2. *margin-bottom*: Sets the bottom margin.
3. *margin-left*: Sets the left margin.
4. *margin-right*: Sets the right margin.

**Note**: Below we will change margin properties for a div that encloses HTML *figure* elements. Figures are used to organize visuals, such as photos and diagrams.

Using borders, padding, and margins allows us to control boundaries and space for individual HTML elements.

But what CSS properties are available to move elements around on the page and create unique page layouts? The CSS display and position properties help accomplish this.



DISPLAY

Not all HTML elements are displayed on a page in the same way. Display types determine how HTML elements will be arranged in relation to each other.

The diagram to the right illustrates the block and inline display types.

In the diagram, notice:

1. The two dotted rectangles represent webpages.
2. HTML heading, paragraph, and unordered list elements are block level: each appears on its own line on the webpage.
3. HTML image and anchor elements are displayed inline: they appear on the same line as their neighboring elements on the webpage.

Familiarize yourself with block and inline display types, then click Next to continue.

Display types can be overwritten in CSS by using the *display* property.

For example, we can make list items appear on the same line by setting display to inline:

li { display: inline; }

**Note**: Below, we will encounter an HTML *nav* element. Navs are used to organize site navigation menus on a webpage.

The navbar is starting to come together nicely. It would be even better if we could get the Contact button to fill in the empty corner on the bottom right.

To achieve this, we can use the CSS *float* property. By using float, we have the option of floating elements left or right.

Consider the example code below. The class selector, .logo, floats left, and the id selector #search-bar floats right:

.logo {

float: left;

}

#search-bar {

float: right;

}

Let's arrange the Contact button using float.

The CSS display value that arranged the images,*flex*, has been removed. In addition to other capabilities, flex can be used to easily align multiple page elements horizontally.

In the example code below, there is a div with class parent:

<div class="parent">

...

</div>

To make children of the div align horizontally on the webpage, in CSS we can use:

.parent {

display: flex;

}

Children elements of the div with class parent will now align horizontally. We can make sure no child element moves off the page by using flex-wrap: wrap;:

.parent {

display: flex;

flex-wrap: wrap;

}

Finally, to center rows of children elements, we can use justify-content: center;:

.parent {

display: flex;

flex-wrap: wrap;

justify-content: center;

}

The CSS *position* property enables you to position HTML elements in exact locations on a webpage. One useful value for this property is *relative*. This value positions page elements on a webpage relative to where they would normally appear.

By first setting position: relative;, you can then use the CSS properties top, left, bottom, and right to shift an element away from where it would have normally appeared on the page.

The code snippet below moves a div with the class container 10px away from the up and 20px away from the left side of the page.

.container {

position: relative;

top: 10px;

left: 20px;

}

Ever click a button on a webpage that seemed to move down and then back up like a real-life button? We can implement this trick using the position property.

# Bootstrap

Bootstrap is a popular CSS framework withprewritten CSS rules designed to help you build webpages faster.

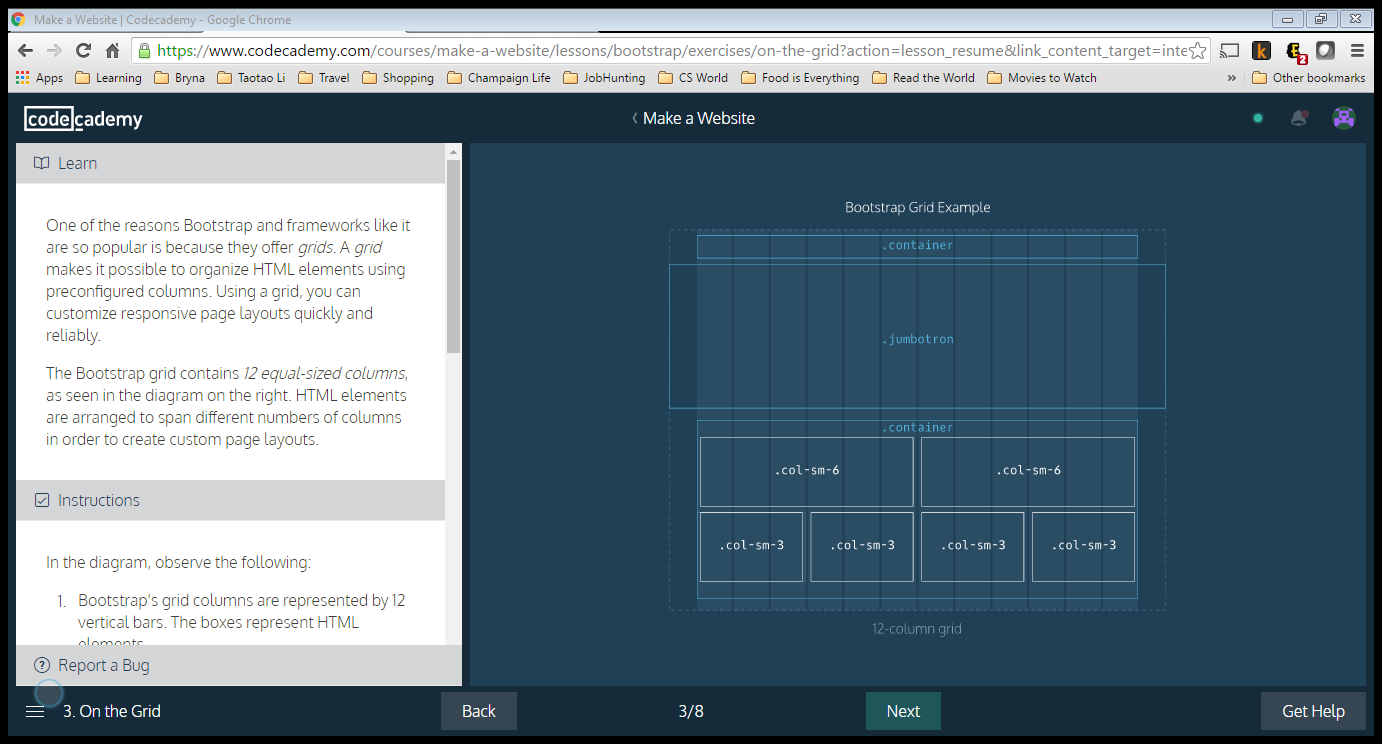
Also in this lesson, we will encounter three new HTML elements:

1. *header*: Used to organize headers on a webpage.
2. *footer*: Used to organize footers on a webpage.
3. *section*: Defines sections on a webpage.

Before Bootstrap can work for us, we need to link to it.

In earlier lessons, we linked only to **main.css**. Now, in addition to **main.css**, we will link to a URL that hosts Bootstrap.

The HTML link element makes Bootstrap available to us. Remember that the link element is typically contained within HTML head tags.



One of the reasons Bootstrap and frameworks like it are so popular is because they offer *grids*. A *grid* makes it possible to organize HTML elements using preconfigured columns. Using a grid, you can customize responsive page layouts quickly and reliably.

The Bootstrap grid contains *12 equal-sized columns*, as seen in the diagram on the right. HTML elements are arranged to span different numbers of columns in order to create custom page layouts.

In the diagram, observe the following:

1. Bootstrap's grid columns are represented by 12 vertical bars. The boxes represent HTML elements.
2. The words "container", "jumbotron", "col-sm-6" and "col-sm-3" refer to Bootstrap classes.
3. The element with class "jumbotron" spans the entire width of the webpage, beyond the borders of the grid.
4. Elements inside the second "container", such as "col-sm-6" and "col-sm-3" are contained within the grid columns.
5. Elements labeled "col-sm-3" take up three grid columns; elements labeled "col-sm-6" take up six grid columns.

In addition to a header/navigation, many websites have a large showcase area featuring important content. Bootstrap refers to this as a *jumbotron*. Below is an example implementation of a jumbotron.

First, a new *section* element is created and assigned the jumbotron class:

<section class="jumbotron"> </section>

Next a div with the Bootstrap class container is used:

<section class="jumbotron"> <div class="container"> ... </div> </section>

A div with the Bootstrap class row text-centercan center subsequent child elements which will contain text:

<section class="jumbotron"> <div class="container"> <div class="row text-center"> ... </div> </div> </section>

The ... is a placeholder for HTML elements containing text, such as h2, p or anchor elements.

Many websites have a supporting content area. Supporting content can be arranged using Bootstrap's grid. Below is an example implementation of a supporting content area.

First, an HTML section element with the containerclass is used:

<section class="container"> </section>

Next, div elements with the row class are added:

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

Finally, the rows are divided by using divs with Bootstrap's col-sm-... class.

<section class="container"> <div class="row"> <div class="col-sm-6"> ... </div> <div class="col-sm-6"> ... </div> </div> <div class="row"> <div class="col-sm-6"> ... </div> <div class="col-sm-6"> ... </div> </div> </section>

Above, two rows are divided into two equal parts. Each part takes up 6 of bootstrap's 12 columns. Using the col-sm-6 class ensures that this layout will appear when the user's screen is the width of a tablet device(768 pixels). On narrower screens, such as an iPhone, only one image per row will appear.

Footers display copyright information, social media links and sometimes site navigation.

Below is one possible implementation for a footer section using Bootstrap:

First, a footer element with Bootstrap's containerclass is used:

<footer class="container"> </footer>

Inside the footer, a div with Bootstrap's row class is added to hold footer content:

<footer class="container"> <div class="row"> ... </div> </footer>

Finally, the row is divided into parts using Bootstrap's grid. Here is one suggestion:

<footer class="container"> <div class="row"> <p class="col-sm-4">...</p> <ul class="col-sm-8"> <li class="col-sm-1">...</li> <li class="col-sm-1">...</li> <li class="col-sm-1">...</li> </ul> </div> </footer>

Above, the row is broken into three parts: a pelement that takes up four columns, a ul which takes up 8 columns, and li items which take up 1 column each. The lis could hold navigation menu items or social media icons.

Again, because the col-sm-... class is used, this layout will appear on tablet-width screens and wider. Screen sizes smaller than tablet-width (768 pixels), will not maintain this layout.