HTML is the language used to create the web pages you visit everyday. It provides a logical way to structure content for web pages.

Let's analyze the acronym "HTML", as it contains a lot of useful information. HTML stands for **H**yper**T**ext **M**arkup **L**anguage.

A markup language is a computer language that defines the structure and presentation of raw text. Markup languages work by surrounding raw text with information the computer can interpret, "marking it up" for processing.

HyperText is text displayed on a computer or device that provides access to other text through links, also known as “hyperlinks”. In fact, you probably clicked on many, many hyperlinks on your path to this Codecademy course!

In this course, you'll learn how to use the fundamentals of HTML to structure, present, and link content. You'll also learn how to use CSS, or Cascading Style Sheets, to style the HTML content you add to web pages.

Let's get started!

**1.**

In the code editor to the right, type your name in between <h1> and </h1>, then press Run.

A [web browser](https://en.wikipedia.org/wiki/Web_browser) must know what language a document is written in before they can process the contents of the document.

You can let web browsers know that you are using the HTML language by starting your HTML document with a *document type declaration*.

The declaration is the following:

**<!DOCTYPE html>**

This declaration is an instruction. It tells the browser what type of document to expect, along with what version of HTML is being used in the document.

<!DOCTYPE html>*must* be the first line of code in all of your HTML documents.

**Note:** If you don't use the doctype declaration, your HTML code will likely still work, however, it's risky. Right now, the browser will correctly assume that you are using HTML5, as HTML5 is the current standard. In the future, however, a new standard will override HTML5. Future browsers may assume you're using a different, newer standard, in which case your document will be interpreted incorrectly. To make sure your document is forever interpreted correctly, always include <!DOCTYPE html> at the very beginning of your HTML documents.

Instructions

**1.**

On line 1, add the <!DOCTYPE html> declaration.

Great! Browsers that read your code will know to expect HTML when they attempt to read your file.

The <!DOCTYPE html> declaration is only the beginning, however. It only tells the browser that you plan on using HTML in the document, it doesn't actually add any HTML structure or content.

To begin adding HTML structure and content, we *must* first add opening and closing <html> *tags*, like so:

**<!DOCTYPE html>**

**<html>**

**</html>**

Anything between <html> and </html> will be considered HTML code. Without these tags, it's possible that browsers could incorrectly interpret your HTML code and present HTML content in unexpected ways.

Instructions

**1.**

After your <!DOCTYPE html> declaration, add opening (<html>) and closing (</html>) tags.

Before we move forward, let's standardize some vocabulary we'll use as you learn HTML. Here are some of the terms you'll see used in this course:

1. Angle brackets - In HTML, the characters < and > are known as angle brackets.
2. HTML element (or simply, element) - HTML code that lives inside of angle brackets.
3. Opening tag - the first, or opening, HTML tag used to start an HTML element.
4. Closing tag - the second, or closing, HTML tag used to end an HTML element. Closing tags have a forward slash (/) inside of them.

With the exception of a few HTML elements, most elements consist of an opening and closing tag.

In the following example, there is one paragraph *element*, made up of one *opening tag* (<p>) and one *closing tag* (</p>):

**<p>Hello there!</p>**

So far, you've declared to the browser the type of document it can expect (an HTML document) and added the HTML element (<html>) that will contain the rest of your code. Let's also give the browser some information about the page. We can do this by adding a <head> element.

The <head> element will contain information about the page that isn't displayed directly on the actual web page (you'll see an example in the next exercise).

**<!DOCTYPE html>**

**<html>**

**<head>**

**</head>**

**</html>**

Instructions

**1.**

Add a <head> element to **index.html**.

What kind of information about the web page can the <head> element contain?

Well, if you look at the top of your browser (or at one of the tabs you may have open in this browser window), you'll notice the words Learn HTML & CSS: Part I | Codecademy, which is the *title* of this web page.

The browser displays the title of the page because the title can be specified directly inside of the <head> element, by using a <title> element.

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>My Coding Journal</title>**

**</head>**

**</html>**

If we were to open a file containing the HTML code in the example above, the browser would display the words My Coding Journal in the title bar (or in the tab's title).

Instructions

**1.**

Add a title to your web page using the <title> element. The title can be be anything you'd like.

We've added some HTML, but still haven't seen any results in the web browser to the right. Why is that?

Before we can add content that a browser will display, we have to add a *body* to the HTML file. Once the file has a body, many different types of content can be added within the body, like text, images, buttons, and much more.

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>I'm Learning To Code!</title>**

**</head>**

**<body>**

**</body>**

**</html>**

All of the code above demonstrates what is sometimes referred to as ["boilerplate code."](https://en.wikipedia.org/wiki/Boilerplate_code)

The term "boilerplate code" is used to describe the basic HTML code required to begin creating a web page. Without *all* of the elements in the boilerplate code, you'll risk starting without the minimum requirements considered to be best practice.

**Note:** The rest of the course will use code examples like the one above. To save space, however, code examples will avoid including common elements like the declaration, head, and so on. Unless otherwise specified, you can assume that the code in the example code blocks belongs directly within the HTML file's body.

Instructions

**1.**

Add a body to your web page using the <body> element.

Headings in HTML can be likened to headings in other types of media. For example, in newspapers, large headings are typically used to capture a reader's attention. Other times, headings are used to describe content, like the title of a movie or an educational article.

HTML follows a similar pattern. In HTML, there are six different *headings*, or *heading elements*. Headings can be used for a variety of purposes, like titling sections, articles, or other forms of content.

The following is the list of heading elements available in HTML. They are ordered from largest to smallest in size.

1. <h1> - used for main headings, all other smaller headings are used for subheadings.
2. <h2>
3. <h3>
4. <h4>
5. <h5>
6. <h6>

The following example code uses a headline intended to capture a reader's attention. It uses the largest heading available, the main heading element:

**<h1>BREAKING NEWS</h1>**

Often times, headings are meant to emphasize or enlarge only a few words.

If you want to add content in paragraph format, you can add a *paragraph* using the paragraph element <p>.

**<p>The Nile River is the longest river in the world, measuring over 6,850 kilometers long (approximately 4,260 miles). It flows through eleven countries, including Tanzania, Uganda, Rwanda, Burundi, Congo-Kinshasa, Kenya, Ethiopia, Eritrea, South Sudan, Sudan, and Egypt.</p>**

Paragraphs are great for expanding the amount of content (text) on your web page. As you begin to add more text to your web page, however, keep in mind that large amounts of text in paragraph format can overwhelm web page visitors. For example, if multiple paragraphs on your web page each contain large amounts of text, your web page could become difficult to consume.

**1.**

In **index.html**, after the <h1> heading, add a paragraph that contains the following text:

The brown bear (Ursus arctos) is native to parts of northern Eurasia and North America. Its conservation status is currently "Least Concern." There are many subspecies within the brown bear species, including the Atlas bear and the Himalayan brown bear.

Often times, it's better to display certain types of content in an easy-to-read list.

In HTML, you can use the *unordered list* for text you decide to format in bullet points. An unordered list outlines individual *list items* with a bullet point. You've probably used an unordered list when writing down a grocery list or school supplies list.

To create a unordered list using HTML, you can use the <ul> element. The <ul> element, however, cannot hold raw text and cannot automatically format raw text with bullet points. Individual list items must be added to the unordered list using the <li> element.

**<ul>**

**<li>Limes</li>**

**<li>Tortillas</li>**

**<li>Chicken</li>**

**</ul>**

In the example above, the list was created using the <ul> element and all individual list items were added using <li> elements.

Instructions

**1.**

In **index.html**, add another paragraph after the first paragraph.

Ordered lists are like unordered lists, except that each list item is numbered. You can create the ordered list with the <ol> element and then add individual list items to the list using <li> elements.

**<ol>**

**<li>Preheat the oven to 350 degrees.</li>**

**<li>Mix whole wheat flour, baking soda, and salt.</li>**

**<li>Cream the butter, sugar in separate bowl.</li>**

**<li>Add eggs and vanilla extract to bowl.</li>**

**</ol>**

Instructions

**1.**

In **index.html**, add another paragraph after the unordered list.

You're off to a great start! So far, you've learned how to add headings, paragraphs, and lists to a web page. We wouldn't be taking advantage of the full power of HTML (and the Internet), however, if we didn't *link* to other web pages.

You can add links to a web page by adding an anchor element <a> and including the text of the link in between the opening and closing tags.

<a>This Is A Link To Wikipedia</a>

Wait a minute! Technically, the link in the example above is incomplete. How exactly is the link above supposed to work if there is no [URL](https://www.codecademy.com/courses/learn-html-css/lessons/common-elements/exercises/links?action=lesson_resume) that will lead users to the actual Wikipedia page?

The anchor element in the example above is incomplete without the href *attribute*.

Attributes provide even more information about an element's content. They live directly inside of the opening tag of an element. Attributes are made up of the following two parts:

1. The *name* of the attribute.
2. The *value* of the attribute.

For anchor elements, the name of the attribute is href and its value must be set to the URL of the page you'd like the user to visit.

**<a href="https://www.wikipedia.org/">This Is A Link To Wikipedia</a>**

In the example above, the href attribute has been set to the value of the correct URL https://www.wikipedia.org/. The example now shows the correct use of an anchor element.

**Note:** When reading technical documentation, you may come across the term *hyperlink*. Not to worry, this is simply the technical term for link and, often times, these terms are used interchangeably.

**1.**

After the first paragraph, add a link that says: Learn More

Have you ever clicked on a link and observed the resulting web page open in a new browser window? If so, you can thank the anchor element's target attribute.

The target attribute specifies that a link should open in a new window. Why is it beneficial to open links in a new window?

It's possible that one or more links on your web page link to an entirely different website. In that case, you may want users to read the linked website, but hope that they return to your web page. This is exactly when the target attribute is useful!

For a link to open in a new window, the target attribute requires a value of \_blank. The target attribute can be added directly to the opening tag of the anchor element, just like the href attribute.

**<a href="https://en.wikipedia.org/wiki/Brown\_bear" target="\_blank">The Brown Bear</a>**

In the example above, the link would read The Brown Bear and open up the relevant Wikipedia page in a new window.

**Note:** In this exercise, we've used the terminology "open in a new window." It's highly likely that you are using a modern browser that opens up websites in new *tabs*, rather than new windows. Before the advent of browsers with tabs, additional browser windows had to be opened to view more websites. The *target* attribute, when used in modern browsers, will open new websites in a new tab.

Instructions

**1.**

In the link that reads Learn More, add the target attribute and set it equal to \_blank.

The <img> element lets you add images to a web page. This element is special because it does not have a closing tag, it only has an opening tag. This is because the <img> element is a *self-closing* element.

**<img src="https://www.example.com/picture.jpg" />**

Note that the <img> element has a required attribute called src, which is similar to the href attribute in links. In this case, the value of src must be the URL of the image. Also note that the end of the <img> element has a forward slash /. This is required for a self-closing element.

**1.**

In **index.html**, below the ordered list, add an image. Use the following as the image URL:

https://s3.amazonaws.com/codecademy-content/courses/web-101/web101-image\_brownbear.jpg

HTML helps support visually impaired users with the alt attribute.

The alt attribute is applied specifically to the <img> element. The value of alt should be a description of the image.

**<img src="#" alt="A field of yellow sunflowers" />**

The alt attributes also serves the following purposes:

1. If an image fails to load on a web page, a user can mouse over the area originally intended for the image and read a brief description of the image. This is made possible by the description you provide in the alt attribute.
2. Visually impaired users often browse the web with the aid of of screen reading software. When you include the alt attribute, the screen reading software can read the image's description outloud to the visually impaired user.

**Note**: If the image on the web page is not one that conveys any meaningful information to a user (visually impaired or otherwise), the alt attribute should not be used.

**1.**

In **index.html**, add the alt attribute to the image. Make sure the description accurately describes the image.

You saw how modifying the spacing between code in an HTML file doesn't affect the positioning of elements in the browser. If you *are* interested in modifying the spacing in the browser, you can use HTML's *line break* element: <br />.

The line break element is one self-closing tag. You can use it anywhere within your HTML code and a line break will be shown in the browser.

**Shall I compare thee to a summer's day?<br />Thou art more lovely and more temperate**

The code in the example above will result in an output that looks like the following:

Shall I compare thee to a summer's day?

Thou art more lovely and more temperate

**Note**: Line breaks are not the standard way of manipulating the positioning of HTML elements, but it's likely that you'll come across them every now and then. In later units, you'll learn more advanced techniques for positioning HTML elements.

**1.**

In **index.html**, add a line break after the word "populations" in the third paragraph of the page.

Comments begin with <!-- and end with -->. Any characters in between will be treated as a comment.

**<!-- This is a comment that the browser will not display. -->**

Including comments in your code is helpful for many reasons:

1. They help you (and others) understand your code if you decide to come back and review it at a much later date.
2. They allow you to experiment with new code, without having to delete old code.

<!-- Favorite Films Section -->

<p>The following is a list of my favorite films:</p>

In the example above, the comment is used to denote that the following text makes up a particular section of the page.

<!-- <a href="#" target="\_blank>Codecademy</a> -->

In the example above, a valid HTML element (an anchor element) has been "commented out." This practice is useful when you want to experiment with new code without having to delete old code.

**1.**

Add a comment anywhere in **index.html**.

So far, you've learned about the fundamentals of HTML, including the basic structure required to set up HTML files, as well as the common HTML elements used to add content to a web page.

Unfortunately, the HTML elements that we've used to add content to a web page have resulted in fairly bland results in the browser. For example, it appears that all content seems to be the same color, have the same font, and offer no direct control over the size of the font (apart from the six different heading options). How can we make our HTML more visually appealing?

CSS, or Cascading Style Sheets, is a language that web developers use to *style* the HTML content on a web page. If you're interested in modifying colors, font types, font sizes, shadows, images, element positioning, and more, CSS is the tool for the job!

In this unit, you'll first learn how to incorporate CSS so that you can style content. You'll also learn about CSS's basic structure and learn how to use its syntax. In later units, we'll explore in detail exactly how to change color, font options, and much more.

Let's begin!

Take a look at the code and the browser to the right. The code contains HTML elements that you've already learned to use, with a couple of exceptions. Don't worry about the details of the code just yet; instead, let's take a quick look at the power of CSS.

Copy the following line of code and paste it on line 5.

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

What happened?

Although CSS is a different language than HTML, it's possible to write CSS code directly within an HTML file. This is possible because of the <style> element.

The <style> element allows you to write CSS code between its opening and closing tags. To use the <style> element, it must be placed inside of the head.

<head>

<style>

</style>

</head>

Once <style> is placed in the web page's head, we can begin writing CSS code.

<head>

<style>

h2 {

font-family: Arial;

}

</style>

</head>

Don't worry about the CSS code in the example above just yet, you will learn more about the details of CSS code in later lessons.

**1.**

Add a <style> element in the head of **index.html**.

Although the <style> element allows you to write CSS code within HTML files, this mixture of HTML and CSS can result in code that is difficult to read and maintain.

It's common for developers to add substantial amounts of custom CSS styling to a web page. When all of that CSS code is placed within a <style> element in an HTML file, you risk the following two things:

1. Creating a large HTML file that is difficult to read and maintain (by you and other developers). Overall, this can result in an inefficient workflow.
2. Maintaining a clear distinction between web page structure (HTML) and web page styling (CSS).

Take a look at the code to the right. Note that **index.html** has evolved to include both HTML and CSS code.

Aside from an increase in length, this mixing of code creates an inefficient workflow for developers and can create confusion because structure (HTML) and aesthetics (CSS) are now combined.

Fortunately, the following solution will help you avoid creating large HTML files that mix in CSS code: a CSS file!

HTML files are meant to contain only HTML code. Similarly, CSS files are meant to contain only CSS code. You can create a CSS file by using the **.css** file name extension, like so: **style.css**

With a CSS file, you can write all the CSS code needed to style a page without having to sacrifice the readability and maintainability of your HTML file.

**1.**

Take a look at **index.html**. Cut the CSS code in between the opening and closing <style> tags and paste it directly in the new file called **style.css**.

Make sure to delete the remaining <style> element (now empty) from **index.html**.

When HTML and CSS code are in separate files, the HTML file must know exactly where the CSS code is kept, otherwise, the styling can't be applied the web page. In order to apply the styling to the web page, we'll have to *link* the HTML file and the CSS file together.

You can use the <link> element to link the HTML and CSS files together. The <link> element must be placed within the head of the HTML file. It is a self-closing tag and requires the following three attributes:

1. href - like the anchor element, the value of this attribute must be the address, or path, to the CSS file.
2. type - this attribute describes the type of document that you are linking to (in this case, a CSS file). The value of this attribute should be set to text/css.
3. rel - this attribute describes the relationship between the HTML file and the CSS file. Because you are linking to a stylesheet, the value should be set to stylesheet.

When linking an HTML file and a CSS file together, the <link> element will look like the following:

<link href="https://www.codecademy.com/stylesheets/style.css" type="text/css" rel="stylesheet">

Note that in the example above the path to the stylesheet is a URL:

https://www.codecademy.com/stylesheets/style.css

Specifying the path to the stylesheet using a URL is one way of linking a stylesheet.

If the CSS file is stored in the same [directory](https://en.wikipedia.org/wiki/Directory_%28computing%29) as your HTML file, then you can specify a [relative path](https://en.wikipedia.org/wiki/Path_%28computing%29#Absolute_and_relative_paths) instead of a URL, like so:

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

Using a relative path is very common way of linking a stylesheet.

**1.**

Let's link the stylesheet **style.css** to the HTML file **index.html**.

First, add a <link> element.

**2.**

Next, add the href attribute to the <link> element and set it equal to /style.css.

Take a look at the web page in the browser to the right. Do you notice any changes?

**3.**

Next, add the type attribute and set it to the correct value.

**4.**

Finally, add the rel attribute and set it to the correct value.

You've learned how to separate HTML and CSS into two files and link them together, but you haven't learned how CSS syntax is used to style elements on a web page.

In this lesson, you'll learn the basic structure and syntax of CSS so that you can start styling web page elements.

Let's begin!

Take a look at the code in **style.css**. You may not understand it right now, but after this lesson, you'll know how CSS syntax works. Later in the course, you'll learn even more about the details of CSS.

Explore the code in this file and try modifying something at random. What happens?

We've discussed styling HTML elements using CSS, but how is it actually done?

To style an HTML element using CSS, you must first *select* that element in the CSS file. For example, to style a <p> element, the syntax to select it using CSS is:

p {

}

In the example above, all paragraph elements are selected using a CSS *selector*. The selector (in this case) is p. Note that the CSS selector essentially matches the HTML tag for that element, but without the angle brackets.

**Note:** The p selector in the example above will select *all* <p> elements on the web page. Later in this course, you'll learn how to use more specific CSS selectors so that you can select any element you want.

Instructions

**1.**

In **style.css**, add a selector for the <h1> element.

**Note:** The content of the web page updates because we've already linked **index.html** and **style.css** for you.

It's not enough to simply select an HTML element in a CSS file. To actually style the element, you need to specify two things inside the body of the selector:

1. Property - the property you'd like to style of that element (i.e., size, color, etc.).
2. Value - the value of the property (i.e., 18px for size, Blue for color, etc.).

h1 {

color: Blue;

}

In the example above, the <h1> element has been selected. Inside of the selector's body, the heading's color property is set to a value of Blue. The heading will now appear the color blue in the browser.

The line color: Blue; is referred to CSS declaration. A CSS declaration consists of a property and a value. Note that a semicolon (;) ends all declarations.

Finally, the entire snippet of code in the example above is known as a CSS rule. A CSS rule consists of the selector and all declarations inside of the selector.

Now that you're familiar with the basics of CSS syntax, let's style some of the elements on the web page.

**1.**

In **style.css**, change the color of the heading to FireBrick.

**2.**

Add another selector that changes the font-size of the paragraph to 18px.

**Note:** All HTML elements have unique properties that can be styled using a variety of values. The purpose of this lesson is to become familiar with CSS syntax. Later in the course, we'll explore specific properties and values for HTML elements.

Styling with CSS would be very inefficient if you were forced to manually style the same property across many elements.

For example, what if you wanted to change the color of 10 different elements to Aquamarine in CSS?.

Fortunately, you can select multiple elements at once so that you can save time styling a shared property.

h1, h2, p {

color: Green;

}

In the example above, the <h1> heading, the <h2> heading, and the paragraph have all been styled to appear Green using a *multiple element selector*. A multiple element selector can save you time when you want to style the same property across many elements.

Instructions

**1.**

Use one selector to change the color of the heading and paragraph to DarkSlateGray.

There's a special selector that can instantly select every single element on the web page: the *universal selector*.

\* {

font-family: Arial;

}

In the example above, the universal selector, \*, is used to select every element on the page and set the font to Arial.

What makes the universal selector so special? When all elements on a web page require the same styling, it's often more efficient to set that styling using the universal selector. Afterwards, you can modify (or remove) that styling for specific elements that don't require it.

In a later lesson, you'll use the universal selector to help you understand how HTML elements are laid out within the browser and how to position those elements.

Instructions

**1.**

In **style.css**, add the universal selector. Inside of the selector's body, set the font-family to Georgia.

Just like HTML, CSS follows certain best practices for spacing and indentation.

h1 {

color: blue;

}

p {

color: red;

}

1. One space should be used between the selector and the opening curly brace ({).
2. No extra spaces should exist between opening and closing curly braces ({ and }) and CSS declarations (as in the example above).
3. Two spaces of indentation should be used for CSS declarations.
4. One line of spacing should exist between CSS rules. In the example above, there is one line of spacing between the CSS rule for the heading and the CSS rule for the paragraph.

CSS files can become lengthy as styling is added to a web page. Proper spacing and indentation helps keep CSS code maintainable and readable for you and other developers.

Just like HTML, you can also leave *comments* in your CSS file. CSS comments begin with /\* and end with \*/, like so:

/\* This is a comment in CSS! \*/

Including comments in your code is helpful for many reasons:

1. They help you (and others) understand your code if you decide to come back and review it at a much later date.
2. They allow you to experiment with new code, without having to delete old code.

/\* Paragraph Styling \*/

p {

color: Blue;

}

In the example above, a comment is used to specify CSS styling for paragraphs.

/\*

h1 {

color: Red;

}

\*/

In the example above, a valid CSS rule has been "commented out." This practice is useful when you want to experiment with new code without having to delete old code.

Instructions

**1.**

Add a comment in **style.css**. The comment can say anything you'd like.

Great work! You've learned the basics of CSS structure and syntax. We'll continue to build on these basics as you learn more about CSS.

Let's review what you've learned so far:

1. A CSS selector targets an HTML element.

2. CSS declarations style HTML elements. Declarations must contain the following two things:

* property - the property you want to style.
* value - the value for the property you are styling.

3. CSS declarations must end in a semicolon (;)

4. A CSS rule consists of a CSS selector and the declarations inside of the selector.

5. Multiple element selectors can be used to style multiple elements at once.

6. Comments keep code easy to read and allow you to experiment with new code without having to remove old code.

7. CSS follows certain best practices for spacing and indentation:

* One line of spacing between a selector and the opening curly brace.
* No spacing between CSS declarations and the opening and closing curly braces of the CSS rule.
* Two spaces of indentation for CSS declarations.
* One line of spacing between CSS rules.

Instructions

If you're up for it, take some time to experiment with your new CSS syntax knowledge in **style.css**.

We've cleared your previous CSS rules to give you a fresh start. You'll also note that **index.html** and **style.css** have already been linked for you. When you're done experimenting, you can proceed to the next unit.

Before discussing the specifics of color, it's important to make two distinctions about color. Color can affect the following design aspects:

1. The foreground color
2. The background color

Foreground color is the color that an element appears in. For example, when a heading is styled to appear green, the *foreground color* of the heading has been styled.

Conversely, when a heading is styled so that its background appears yellow, the *background color* of the heading has been styled

In CSS, these two design aspects can be styled with the following two properties:

1. color - this property styles an element's foreground color.
2. background-color - this property styles an element's background color.

h1 {

color: Red;

background-color: Blue;

}

In the example above, the text of the heading will appear in red, and the background of the heading will appear blue.

Instructions

**1.**

In **style.css**, set the foreground color of the main heading to MidnightBlue.

**2.**

Next, set the background color of the heading to Aqua.

Notice that the background-color property did not change the background color of the entire page. Instead, only the background area behind heading changed. We'll learn why this occurs in more detail later in the course.

**Note:** The background color of the heading is blocking the image of the coffee, so let's remove it. Remove the background color of the main heading.

Over the past few exercises, you've seen CSS examples that use colors like Red, Blue, or Cyan. In CSS, these colors are technically known as named colors. There are a total of [147 named colors](http://www.colors.commutercreative.com/grid/).

At this point, you might be wondering if you are expected to memorize the list of 147 named colors that CSS offers.

Fortunately, you don't have to! There are plenty of available resources on the Internet that list all of the named colors in CSS, like the one we linked you to above. If you're ever in need of a named color, a quick Google search will yield the results you are looking for.

Let's try some new named colors out!

Instructions

**1.**

Set the color of the main heading to Crimson.

**2.**

Set the background color of the subheadings to DarkMagenta

Although named colors provide 147 different options, this is a small amount when you consider the flexibility of CSS. To take advantage of the full spectrum of colors that CSS supports, you have the option of using RGB colors.

RGB (Red, Green, Blue) colors offer the option of 16,777,216 possible colors. How is that possible?

RGB colors work by mixing together different amounts of red (R), green (G), and blue (B). Each color (R, G, or B) can take on 1 of a possible 256 values (between 0 and 255). This results in 16,777,216 possible colors.

To use RGB colors, you can use the rgb() value when styling a color.

h1 {

color: rgb(123, 20, 233);

background-color: rgb(99, 21, 127);

}

In the example above, the value of color is set to rgb(). The three numbers in the parentheses represent the values for R, G, and B, in that order. Note that you can use rgb() for background colors as well.

How can you tell what color the RGB values in the example above will result in? Are you expected to memorize 16,777,216 possibilities? Thankfully, no!

There are many resources on the Internet known as "[color pickers](https://color.adobe.com/create/color-wheel/)" that allow you to view the result of different RGB values before you decide to use a certain color. If you're ever in need of a color picker resource, a quick Google search will yield the results you are looking for.

Instructions

**1.**

In **style.css**, remove the named color for the main heading and set it to an RGB color with the following values: 54, 74, 101.

**2.**

Next, remove the named color for the subheading's background and set it to an RGB color with the following values: 23, 108, 224.

There's an additional way to specify colors in CSS: hexadecimal color codes, often referred to as "hex color codes" for short.

Hex color codes also offer 16,777,216 color options, but they follow a different syntax.

When specifying an RGB color mixture, the values are in [base 10](https://en.wikipedia.org/wiki/Decimal). Hex color codes, however, use [base 16](https://en.wikipedia.org/wiki/Hexadecimal), or hexadecimal base (hence the name), to specify color mixtures.

h1 {

color: #09AA34;

}

When read from left to right, each group of two characters responds to a value for red, green and blue, respectively. In the example above, 09 refers to the value for red, AA refers to the value for green, and 34 refers to the value for blue. All hex color codes begin with a # character.

Is there a difference between RGB values and hex color codes?

Not really. RGB values and hex color codes are different ways to represent the same thing: color. It's possible to convert back and forth between RGB values and hex color codes (color pickers often help with this conversion).

**Note:** When a hex color code is composed of entirely of the same characters, the hex color can be abbreviated, like so:

h1 {

color: #FFFFFF;

color: #FFF; /\* This is the same color as above \*/

}

h2 {

color: #AA33BB;

color: #A3B; /\* This is the same color as above \*/

}

Instructions

**1.**

In **style.css**, change the color of the main heading to #452F73.

**2.**

Next, change the background color of the subheadings to #AA8EB5.

The current revision of CSS, CSS3 (at the time of this writing), introduces a new way to specify colors using *HSL colors*.

HSL stands for **H**ue, **S**aturation, and **L**ightness. Specifically, this is what each means:

1. Hue - the technical term that describes what we understand as "color." In HSL, hue is represented on a color wheel. It can take on values between 0 and 360.
2. Saturation - the amount of gray in a given color. In HSL, saturation is specified using a percentage between 0% and 100%. The percentage 0% represents a shade of gray, whereas 100% represents full saturation.
3. Lightness - the amount of white in a given color. Similar to saturation, lightness is specified using a percentage between 0% and 100%. The percentage 0% represents black, whereas 100% represents white. 50% is normal.

You can use HSL colors in your CSS like this:

h1 {

color: hsl(182, 20%, 50%);

}

Notice that using HSL is very similar to using RGB.

**Note:** Because HSL is a part of CSS3, older browsers may not support it. In a later exercise, you'll learn how to work around support issues for colors.

Instructions

**1.**

In **style.css**, change the color of the main heading and set it to an HSL color with the values 350, 58.8%, 42.0%

You learned that RGB and hex color codes are two different methods of representing the same thing: color. However, there is one feature that RGB colors support that hex color codes do not: opacity.

Opacity is a measure of how transparent a color is. To modify opacity in RGB colors, CSS offers the rgba() value. Note the slight difference in rgb() and rgba().

The extra a character in the rgba() value is known as the *alpha value*. It represents the opacity of a color. The alpha value can be a number between 0 or 1, inclusive.

h1 {

color: rgba(123, 88, 9, 0.5);

}

In the example above, the alpha value has been set to 0.5. This indicates that the color of the heading will be set to 50% of its normal opacity.

**Note:** The alpha value can also be used for HSL colors, using hsla():

h1 {

color: hsla(239, 45%, 22%, 0.4);

}

Instructions

**1.**

In **style.css**, change the color of the main heading and set it to an RGBa color with the values 2, 93, 140 and an opacity of 0.6.

**2.**

Notice how the main heading is now transparent. Let's make it more opaque to make it easier to read for users. Keep the same color, but change the alpha value to 0.9.

RGB colors, hex color codes, and HSL colors offer web developers an extraordinary amount of color customization options. As these properties become more advanced, however, it's important to keep in mind that not all users browse the Internet with the same browser, let alone the same version of a given browser.

How does this affect web development? Newer revisions of HTML and CSS affect older browsers. Older browsers, over time, will become dated (possibly obsolete) and not be able to support newer CSS features. For example, many older browsers do not support RGBa, HSL, or HSLa.

Because of this, we must include redundant color options in our CSS code that can cater to a wide audience of different browsers.

Specifically, we can add multiple CSS color declarations, just in case a user's browser can't support a certain declaration.

h1 {

color: rgb(22, 34, 88);

color: rgba(22, 34, 88, 0.4);

}

In CSS, the latter of multiple declarations takes priority. In the example above, if the user's browser supports rgba(), then that color will be applied to the heading. If it does not, then CSS will use the first rgb() color declaration, as a backup.

Using redundant declarations allow you to support as many users as possible across multiple versions of different Internet browsers.

Instructions

**1.**

In **style.css**, set the color of the main heading to an RGB color with the values 3, 150, 100.

**2.**

On the next line, set the main heading's color to an RGBa color with the values 3, 101, 100, 0.75.

Which color was applied to the heading?

Great job! You've learned how to style an important aspect of the user experience: color.

Let's review what you've learned so far:

1. Foreground color refers to the actual color of an element, styled with the color property.
2. Background color refers to the color behind an element, styled with the background-color property.
3. There are 147 named colors available.
4. RGB and hexadecimal colors offer over 16 million color possibilities.
5. HSL is a new way of defining colors in CSS3.
6. You can modify the opacity of a color with RGBa or HSLa colors.
7. Not all browsers support newer CSS features, like opacity or HSL, so additional declarations should be made to support a wide audience of users.
8. There are many color picker resources available on the Internet to help you select specific colors, as well as provide colors in different formats.

If you've ever used a formatted word processor, chances are that you probably also used a feature that allowed you change the "type of font" you were typing in. The phrase "type of font" refers to the technical term [typeface](https://en.wikipedia.org/wiki/Typeface), or font family.

To change the typeface of text on your web page, you can use the font-family property.

h1 {

font-family: Garamond;

}

In the example above, the font family for all main heading elements has been set to Garamond.

When setting typefaces on a web page, keep the following points in mind:

1. The font specified in a stylesheet must be installed on a user's computer in order for that font to display when a user visit the web page. We'll learn how to work around this issue in a later exercise.
2. You've probably noticed that we haven't been specifying a typeface in previous exercises of this course. How exactly does the browser know what typeface to use when displaying the web page? The default typeface for all HTML elements is Times New Roman. You may be familiar with this typeface if you have ever used a formatted word processor.
3. It's a good practice to limit the number of typefaces used on a web page to 2 or 3.
4. When the name of a typeface consists of more than one word, it must be enclosed in double quotes (otherwise it will not be recognized), like so:

h1 {

font-family: "Courier New";

}

Instructions

**1.**

In **style.css**, change the font family of the main heading and subheading to Georgia.

**2.**

Next, change the font family of the paragraph to Helvetica.

The practice of [typography](https://en.wikipedia.org/wiki/Typography) has been around for centuries! Over time, typographers have refined their craft and have developed many different typefaces, which has allowed them, in some cases, to classify them as one of the following two types: serif fonts and sans-serif fonts.

1. Serif - the letters in these fonts have extra details on the ends of each letter. Examples include fonts like Times New Roman or Georgia, among others.
2. Sans-Serif - the letters in these fonts do not have extra details on the ends of each letter. Instead, letters have straight, flat edges. Some examples include Arial or Helvetica.

The majority of fonts that we'll study in this lesson will be either serif or sans-serif fonts.

Notice the difference between serif fonts and sans-serif fonts in the diagram to the right. When you're done, proceed to the next exercise.

Earlier, you learned that users *must* have the fonts specified in the stylesheet installed on their computer in order for their browser to display that font. What happens when a font is not installed on a user's computer?

Most computers have a small set of typefaces pre-installed. This small set includes serif fonts and sans-serif fonts, like Times New Roman and Arial, respectively.

When the stylesheet specifies a font not installed on a user's computer, the pre-installed fonts can be used as *fallback fonts* for users.

To use fallback fonts, the following syntax is required:

h1 {

font-family: Garamond, Times, serif;

}

The CSS rule above says: "Use the Garamond font for all <h1> elements on the web page. If that font is not available, use the Times font. If both of those fonts are not available, use any serif font pre-installed on the user's computer." The fonts specified after Garamond are the fallback fonts.

Fallback fonts help ensure a consistent experience for the diverse audience of users that visit a site.

Instructions

**1.**

In **style.css**, add Garamond as the first *fallback* font for the main heading and subheading, and add serif as the final fallback font.

The first fallback font should be Garamond, and the final fallback font should be serif, like so:

h1 {

font-family: Georgia, Garamond, serif;

}

Make sure to also set this for the subheading.

**2.**

Next, set the first fallback font of the paragraph to Arial, and set the final fallback font to sans-serif.

New fonts are constantly being developed. Because there are so many new fonts available, it would be unrealistic to expect users to have all of them installed on their computers.

Fortunately, you don't have to predict which fonts are installed on a user's computer. Many (but not all) of the new fonts that emerge on a daily basis are being centralized in directories made available for public use.

For example, Google offers [Google Fonts](https://fonts.google.com/), a directory of thousands of open-source fonts that are free to use by anyone.

To use these fonts, you can link to a specific Google Font in your HTML code and use it immediately in your stylesheet. Because the HTML file links directly to the Google Font, a user's browser can display the typeface you specify. This avoids having to determine whether or not that font is installed on a user's computer.

To use a Google Font, you can use a <link> element, just like you did for a CSS stylesheet:

<head>

<link href="https://fonts.googleapis.com/css?family=Raleway" type="text/css" rel="stylesheet" >

</head>

In the example above, the href attribute is set to the following URL, which was retrieved from Google Fonts:

https://fonts.googleapis.com/css?family=Raleway

The URL in the example above specifies the Raleway typeface from Google Fonts.

You can use the new font just as you would use any other font:

h1 {

font-family: Raleway, Georgia, serif;

}

You now have access to thousands of new, modern, free-to-use fonts!

Instructions

**1.**

In **index.html**, add a link to the Google Font Roboto. Use the following URL:

https://fonts.googleapis.com/css?family=Roboto:100

The 100 in the URL specifies an optional font weight for Roboto (don't worry, you'll learn about font weight in this lesson).

**2.**

In **style.css**, specify Roboto as the font family for the subheading and the paragraph, followed by sans-serif as the fallback font

Notice how the fonts changed!

**3.**

In **index.html**, add an additional link to the Google Font Playfair Display. Use the following URL:

https://fonts.googleapis.com/css?family=Playfair+Display

**4.**

In **style.css**, specify Playfair Display as the font family for the main heading, followed by serif as the fallback font.

Changing the typeface isn't the only way to customize text. Often times, different sections of a web page and are highlighted by modifying the *font size*.

To change the size of text on your web page, you can use the font-size property.

p {

font-size: 18px;

}

In the example above, the font-size of all paragraphs was set to 18px. What exactly does px mean?

Measurements require units in order for them to be useful (for example kilograms for weight, or miles for distance). Font size also requires a unit of measurement.

In the next exercise, we'll explore in detail the units of measurement available for font size.

Instructions

**1.**

In **style.css**, set the size of the paragraph to 20 pixels.

There are three units of measurement for font size:

1. px - Represents the unit of *pixels*. The display of a computer monitor can be measured in pixels. A pixel is a small point on the display. How small? Most computer monitors have a resolution of 72 pixels per inch, so a pixel represents about 1/72nd of an inch. Pixels are sometimes also referred to as *points*. Pixels are used to set the exact size of an element, in this case, text.

p {

font-size: 18px;

}

2. ems - Pronounced just as it looks, "em." An em is equal to the width of the letter "m". Ems are a relative unit of measurement. They change the size of text relative to the parent element's size of text.

p {

font-size: 1.3em;

}

3. % - Percentages are also a relative unit of measurement. The default size of text in web browsers is 16 pixels, or 16px. When percentages are used, they set the size of text relative to this default size. For example, setting the font size to 200% would be equivalent to setting it to 32px.

p {

font-size: 150%;

}

Text on a web page must also be easy to read. When text is styled to appear larger, the vertical spacing between lines of text can decrease, creating text that is difficult to read, particularly in paragraphs.

To avoid this problem, you can modify the spacing between lines of text with the line-height property.

p {

line-height: 1.5em;

}

When the line height of an element is modified, you are manipulating the *leading* (pronounced "ledding") of the font. When the line height is increased, the spacing between lines of text increases, which can make text easier to read.

The line height can be modified using pixels or ems, but the unit of ems is preferred, since ems offer a spacing relative to the size of the text on the page.

Instructions

**1.**

In **style.css**, set the line height of the paragraph element to 1.7em.

When the line-height property of an element is modified, the leading is increased, resulting in an increase of the vertical spacing between lines of text.

You can also increase the spacing between words in a body of text, technically known as *word spacing*.

To do so, you can use the word-spacing property:

h1 {

word-spacing: 0.3em;

}

The default amount of space between words is usually 0.25em. In the example above, the word spacing is set to 0.3em, which represents an increase of only .05em in word spacing.

It's not common to increase the spacing between words, but it may help enhance the readability of bolded or enlarged text. Note, again, that the preferred unit is ems.

Instructions

**1.**

In **style.css**, set the word spacing of the subheading to 0.05em.

You've learned how to increase the spacing between lines of text and words, but it's possible to get even more detailed: increasing the spacing between individual letters.

The technical term for adjusting the spacing between letters is called "kerning". Kerning can be adjusted with the letter-spacing property in CSS.

h1 {

letter-spacing: 0.3em;

}

Like word spacing, it's not common to increase the kerning in text, but sometimes it enhances the readability of uppercase text.

Instructions

**1.**

In **style.css**, set the letter spacing of the subheading to 0.02em.

You've probably noticed **bolded** text across many different web sites. It's common to bold important headings or keywords.

In CSS, the font-weight property turns bold on or off.

p {

font-weight: bold;

}

In the example above, all paragraphs on the web page would appear bolded.

The font-weight property has a second value: normal. Why does it exist?

If we wanted *all* text on a web page to appear bolded, we could select all text elements and change their font weight to bold. If a certain section of text was required to appear normal, however, we could set the font weight of that particular element to normal, essentially "shutting off" bold for that element.

In later units, you'll learn how to be more selective about what parts of your site you'd like to style.

Instructions

**1.**

In **style.css**, set the font weight of the paragraph to bold. It'll be difficult to notice, but the letters in the paragraphs will thicken a bit.

**2.**

Earlier, you added the following line of code to **index.html**:

<link href="https://fonts.googleapis.com/css?family=Roboto:100" rel="stylesheet">

Notice the 100 in the URL. This specifies that you'd like to use a font weight of 100 when linking to the Roboto font. It's possible to specify a different font weight, or even multiple within the same URL.

Font weights can be set to 100 (thin), 200, 300, 400 (normal weight), 500, 600, 700, 800, or 900 (bold).

In **style.css**, set the font weight of the paragraph to 100.

You can also *italicize* words with the font-style property.

h3 {

font-style: italic;

}

The italic value causes text to appear in italics. The font-style property also has a normal value, for the same reasons discussed in the previous exercise.

Instructions

**1.**

In **style.css**, set the font style of the subheading to italic.

Text can also be styled to appear in either all uppercase or lowercase with the text-transform property.

h1 {

text-transform: uppercase;

}

The code in the example above formats all <h1> elements to appear in uppercase, regardless of the case used for the heading within the HTML code. Alternatively, the lowercase value could be used to format text in all lowercase.

Since text can be directly typed in all uppercase or lowercase within an HTML file, what is the point of a CSS rule that allows you to format [letter case](https://en.wikipedia.org/wiki/Letter_case)?

Depending on the type of content a web page displays, it may make sense to always style a specific element in all uppercase or lowercase letters. For example, a website that reports breaking news may decide to format all <h1> heading elements such that they always appear in all uppercase, as in the example above. It would also avoid uppercase text in the HTML file, which could make code difficult to read.

Instructions

**1.**

In **style.css**, transform the text in the main heading to appear uppercase.

No matter how much styling is applied to text (typeface, size, weight, etc.), text always appears on the left side of the browser.

To move, or align, text, we can use the text-align property.

h1 {

text-align: right;

}

The text-align property can be set to one of the following three values:

1. left - aligns text to the left hand side of the browser.
2. center - centers text.
3. right - aligns text to the right hand side of the browser.

Later in the course, you'll learn exactly how the browser positions HTML elements by default, which will help you understand how the browser "aligns" text, since "align" is a relative term. For now, it's enough to know that text can be moved to the left, center, or right side of the web page.

Instructions

**1.**

In **style.css**, set the text-align property of the main heading so that it appears in the center.

**2.**

On second thought, the heading looks better on the left. Reset the heading so that it is aligned to the left.

Great job! You learned how to style an important aspect of the user experience: fonts!

Let's review what you've learned so far:

1. The font-family property changes the typeface of text.
2. Serif fonts have extra details on the ends of each letter. Sans-Serif fonts do not.
3. Fallback fonts are used when a certain font is not installed on a user's computer.
4. Google Fonts provides free fonts that can be used in an HTML file with the <link> element.
5. Font size can be specified using pixels, ems, or percentages.
6. The vertical spacing between lines of text can be modified with the line-spacing property.
7. The horizontal spacing between words can be modified with the word-spacing property.
8. The spacing between letters, the kernel, can be modified with the letter-spacing property.
9. Text can appear bold with the font-weight property.
10. Text can appear in italics with the font-style property.
11. Text can appear in all uppercase or all lowercase with the text-transform property.
12. Text can be aligned with the text-align property.

So far, you've learned how to style color and fonts. Note, however, that all of the styling you've used is always applied to all elements of one type on a web page. For example, a p selector in the stylesheet targets *all* <p> elements on the web page. How can we be more selective about the elements we'd like to style?

In the next unit, you'll learn how to organize and label your HTML code so that you can be more selective with your CSS styling.

With the proper labels, we can style individual HTML elements! Specifically, we can label HTML elements with a unique identifier, or *ID*. We can then style that specific element in the stylesheet.

To label an element with an ID, we can use the id attribute on an HTML element.

<h1 id="botswana">Botswana</h1>

In the example above, the heading is labeled with an id of botswana.

What purpose do IDs serve? IDs are intended to label unique elements in an HTML file. No two HTML elements should ever share the same ID — that would defeat the purpose of a unique identifier!

Instructions

**1.**

In **index.html**, add an ID to the <div> nested inside of the first <div> element. The ID should be set to header-text.

**Note**: Don't focus on understanding the <div> element just yet, we'll cover it later in this lesson.

Now that you know how to label HTML elements with an ID, we can style that specific element in the stylesheet.

In a previous lesson, you learned how to target HTML elements with element selectors, like so:

p {

font-size: 10px;

font-color: #A091DD;

}

Remember, however, that an element selector like the one in the example above would targets *all* paragraphs on the web page. In the example above, all paragraphs are styled to have a font size of 10 pixels and a font color of #A091DD.

To style a specific element labeled with an ID, you can use an *ID selector* in the stylesheet.

#botswana {

background-color: #56ABFF;

}

In the example above, the HTML element with an ID of botswana is targeted with the ID selector #botswana.

All ID selectors begin with the octothorpe character: #. The value of the ID immediately follows the octothorpe. Once you've correctly targeted the element, you can proceed to style it as usual.

Instructions

**1.**

In **style.css**, add an ID selector for the ID header-text.

IDs are great for labeling unique elements, but IDs have a limitation. Because unique IDs should not be used across multiple HTML elements, they are insufficient for quickly styling elements that should all share a specific style.

CSS offers a solution to this limitation. For multiple elements that should share the same styling, *classes* can be used to label them.

To label an element with a class, we can use the class attribute on an HTML element.

<h1 class="science">Scientist Discovers Important Cure</h1>

<h1 class="science">New Study Reveals The Importance of Sleep</h1>

In the example above, there are two headings with a class of science. Why?

HTML elements are often labeled with class names that reflect the content they represent on the web page. In the example above, perhaps a news company decided that all news headlines about science should be labeled with a class of science.

Later, in the stylesheet, all science headlines can be quickly styled to have the same styling.

Instructions

**1.**

In **index.html**, give the second <div> a class of partner.

Now that you know how to label HTML elements with a class, we can style elements belonging to the same class at once. How exactly do you select them in CSS, though?

To style elements of the same class, you can use a *class selector* in the stylesheet.

.science {

font-family: Georgia, Times, serif;

color: #A3B4C5;

text-transform: uppercase;

}

Class selectors begin with a period (.) and are immediately followed by the name of the class. In the example above, all elements with a class of science will have their typeface, color, and letter case modified.

As you learn more about web development, you'll notice that it's very common to style groups of elements using classes. In fact, you're more likely to see classes more often than you see IDs. While IDs still play a critical role when you move into languages like JavaScript, classes are by far the most commonly used for styling groups of elements.

Instructions

**1.**

Scroll through **style.css** and add a class selector for the class partner.

The class selector targets *all* elements of a particular class. It's possible, however, for multiple elements on a web page to share a specific styling, but for one of those elements to differ slightly.

For example, a heading and a paragraph (both with a class of breaking) may need to share the same typeface, but the paragraph may require a styling better suited for paragraphs, as in the following example.

.breaking {

font-family: Georgia, Times, serif;

}

p.breaking {

line-height: 1.3em;

}

The example above uses a new selector: p.breaking. What's the difference between the .breaking and p.breaking selectors?

The .breaking selector targets *all* elements with a class of breaking. The p.breaking selector targets *only* <p> elements with a class of breaking. This type of selector allows you to be even more specific about a particular element's styling, even when that element must share some styling with other elements.

Unless otherwise specified, the rest of this course will use the element.class selector syntax.

CSS does not limit you to selectors that target a single element or class.

In a previous exercise, you learned how to use a multiple element selector to style multiple elements at once.

h1, p {

font-family: Garamond, serif;

}

The same syntax can be used to style multiple classes at once.

.first, .last {

font-size: 20px;

}

Using a multiple class selector is an efficient way of styling multiple HTML elements.

Instructions

**1.**

Scroll down in **style.css** and take a look at the .prop and .partner CSS rules. They both share the same text-align setting. Delete this property from *both* CSS rules.

Next, create a CSS rule that uses two multiple class selectors at once (.partner, .prop). Inside of the rule, set the text-align property to center.

This is one example of using a selector that targets multiple CSS classes at once.

It's also possible to label HTML elements with more than one class. How is this functionality useful?

You learned that elements that share a styling (e.g. typeface) are labeled with the same class. When those same elements must also be differentiated, however, labeling with an additional class is helpful.

<h1 class="book domestic">The Way of the Deep</h1>

<h1 class="book foreign">A Night in the Sky</h1>

.book {

font-family: Georgia, serif;

}

.domestic {

font-color: #0902CC;

}

.foreign {

font-color: #B097DD;

}

In the example above, the HTML code uses main headings for two different book titles. In this example, all book titles must share the same typeface, so the headings are labeled with a class of book and are then styled with a typeface of Georgia.

The books, however, must be differentiated based on their country of origin. To differentiate the book titles based on this information, two additional classes, domestic and foreign, are applied to the respective headings, which style the color of the heading in the CSS code.

To label an HTML element with an additional class, simply include the class within the double quotes, immediately after the first class. HTML elements can be labeled with many classes, but whenever possible, it's best to limit an element to four classes at most.

Instructions

In this web page, no HTMl element needs to have multiple classes. However, be aware that using multiple classes for an element is extremely common in web development. Keeping this exercise as reference will come in handy later.

Classes and IDs are useful for labeling elements, but a disorganized, overuse of classes and IDs can result in an HTML file that is difficult to read. We need a way of organizing the contents of the HTML file. This will help:

1. Keep HTML code easy to read.
2. Group elements that belong together.

HTML offers an element that is the backbone of code organization: the *div*, represented by <div> in HTML.

You can think of the div as a box, or container, that groups elements that belong together.

For example, a <div> can be used to group together all of the elements that make up the navigation for a web page, or any other section of a page.

<div>

<h1>Alice In Wonderland</h1>

<p> ... </p>

</div>

In the example above, a heading for "Alice In Wonderland" and a brief paragraph description of the book are contained within a single <div>. If more books were to appear on the page, additional divs could be used to organize them. This would keep the code easier to read, maintain, and style.

Instructions

**1.**

Scroll down in **index.html**. You'll notice this bit of code sitting by itself:

<img src="https://s3.amazonaws.com/codecademy-content/courses/web-101/unit-5/htmlcss1-img\_diamond.png" width="60px">

<h2>Luxury</h2>

<p>We understand that you expect the world of your vacation - Jetsetter can provide once in a lifetime experiences at top of the line quality.</p>

Enclose this bit of code with a div.

Now that you know how to organize code into sections using divs, we can start labeling divs with classes instead, rather than labeling individual elements with classes.

This does not mean that labeling individual elements with classes is no longer useful. Even when divs are labeled with classes, there will be many other times when an individual element will need to be labeled with a class.

<div class="container">

<h1 class="title">Alice In Wonderland</h1>

<p> ... </p>

</div>

div.container {

background-color: rgb(252, 255, 205);

font-family: Roboto, Helvetica, sans-serif;

}

h1.title {

color: #0D1A2F;

}

In the example above, a heading and paragraph are contained within a div that has a class of container. In the stylesheet, the background color and typeface of the div have been styled.

Divs are intended to contain other HTML elements, not raw text. Does it make sense to style a div directly then?

When a div is styled, *all elements inside* of the div will *inherit* the styling applied to the div. This example illustrates how easy it is to style sections of a web page using div.

**Note:** The example above uses a class called container. Chances are that you'll frequently see this class name as you learn more about web development.

**1.**

Give the div that you added in the last exercise a class of prop.

**2.**

Notice that there are three divs in **index.html** with a class of prop. Let's organize these three divs into a larger, containing div.

Enclose these three divs with a div that has a class called value-props.

**3.**

Finally, let's contain the value-props div into an even larger div.

Enclose the entire value-props div with a div that has a class of main.

**Note**: Be sure to also enclose the <h3> and <p> elements above the value-props div.

The div is one of the most commonly used elements in all of HTML. Modern web pages make extensive use of the div, and learning how to use divs for organization and styling is a critical skill for all web developers.

Moving forward, this course will make more use of the div so that you can master it and become more familiar with this important element.

Instructions

Take a look at the code in **index.html**. You're in a much better position now to understand the code and edit it.

Note that the HTML code is organized neatly with div sections. Note, also, that the stylesheet targets divs heavily in order to make styling more efficient.

Great job! You learned how to organize HTML with classes, IDs, and divs.

Let's review what you've learned so far:

1. Code is a lot more readable when it is organized using IDs, classes, and divs.
2. IDs label HTML elements that are unique to the web page (an element that appears *only once*).
3. Classes label elements that will share the same styling. They make styling more efficient.
4. The <div> groups elements together. It makes the HTML file easier to read by organizing the web page into logical sections.
5. HTML elements can be labeled with multiple classes.
6. Divs are one of the most commonly used HTML elements. Understanding how they are used is a critical skill for web developers.

In this unit, you learned how to organize code with IDs, classes, and divs. The remainder of this course will use these concepts extensively, so it's important to have a strong understanding of them.

Instructions

Take some time to experiment with your new knowledge of IDs, classes, and divs in **index.html** and **style.css**. When you're done, you can proceed to the next unit.

In the last unit, you learned how to organize HTML code with IDs, classes, and divs. All of these concepts are necessary to understand advanced CSS styling techniques. An understanding of CSS is not complete, however, without a study of the *box model*.

In some of the past exercises, you've unknowingly seen aspects of the box model. For example, when you set the background color of an element, you may have noticed that the background color was applied not only to the area directly behind the element, but also to the area to the right of the element. In another exercise, you learned how to align the text. How did the browser know how to align the text?

All HTML elements live within a box. Elements on a web page are understood by the browser as "living" inside of a container, or a box. This is what is meant by the *box model*.

When you changed the background color of an element, you changed the background color of its entire box. When you aligned text, the text was aligned relative to the element's entire box. To truly create custom websites, you'll have to understand the box model.

In this unit, you'll learn about the following aspects of the box model:

1. The dimensions of an element's box
2. The borders of an element's box
3. The content within an element's box
4. The area around all four sides of an element's box

Let's begin!

Instructions

Where are the boxes that supposedly contain all HTML elements? They're invisible, so we'll have to reveal them!

In **style.css**, add the following code:

\* {

border: 1px solid rgba(0, 0, 0, 0.3);

}

The code above selects *all* elements on the page (using the universal selector you learned about earlier) and reveals the borders of their box. Surprise!

Don't worry about the details of the border property. You'll learn about it in this unit.

An element's box has two dimensions: a height and a width. In HTML, all boxes have default dimensions. These default dimensions are automatically set to hold the raw contents of the box.

To modify the default dimensions an element's box in CSS, you can use the width and height properties.

These two properties can be specified with the following units of measurement:

1. Pixels - You learned about pixels when you learned about fonts. This unit lets you set the exact size of an element's box.
2. Ems - This unit sets the dimensions of the box relative to the size of the text within the box.
3. Percentages - This unit sets the dimensions of the box relative to the size of the box that it is encased in. For example, consider an element (a box) of class blue set to a height of 200 pixels and a width of 200 pixels. Inside of blue, consider another box of class red, set to a height of 37% and a width of 45%. The resulting dimensions of the red box would be a height of 74 pixels and a width of 90 pixels.

Developers often use ems or percentages to set box dimensions. Because many web pages are now accessed on mobile devices and other displays of differing sizes, ems and percentages allow boxes to scale depending on the size of a user's display.

Instructions

**1.**

In **style.css**, set the height of the content-header div to 320 pixels.

**2.**

Next, set the width of the navigation div to 100%.

**Note:** The browser to the right is, of course, a smaller version of a typical browser you might use. In order for the web page to look presentable in this smaller viewport, we'll continue to use percentages for box dimensions. In a later course, you'll learn more about [responsive web design](https://en.wikipedia.org/wiki/Responsive_web_design).