1.

Webmaster day 1

What is HTML?

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

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" to be processed.
- In HTML, the computer can interpret raw text that is wrapped in HTML *elements*. These elements are often nested inside one another, with each containing information about the type and structure of the information to be displayed in the browser.
- *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!

INTRODUCTION TO HTML

!DOCTYPE

Whether you realize it or not, when you read text, your brain must first identify the text's language. If you can understand that language, then your brain immediately begins to interpret the text. This same process happens whether you're reading a street sign, a book, or a name tag.

Web browsers work in a similar way. They must know what language a document is written in before they can process its contents.

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

The declaration looks like this: `<!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. If you don't use the declaration, your HTML code will likely still work, however, it's risky. For now, the browser will correctly assume that the `html` in `<!DOCTYPE html>` is referring to HTML5, as it 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.

Preparing for HTML

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 indicates to the browser that you will use HTML in the document, but it doesn't actually add any HTML structure or content.

To create HTML structure and content, we must add opening and closing `<html>` *tags*, like so:

```
<!DOCTYPE html>
<html>

</html>
```

Anything between the opening `<html>` and closing `</html>`tags will be interpreted as HTML code. Without these tags, it's possible that browsers could incorrectly interpret your HTML code.

Instructions

After the `<!DOCTYPE html>` declaration, notice the opening (`<html>`) and closing (`</html>`) HTML tags.

HTML Anatomy

Before we move forward, it's important that we discuss how HTML elements are structured. The diagram to the right displays an HTML paragraph element.

In this example, the paragraph element is made up of one opening tag (`<p>`), the “Hello world!” text, and a closing tag (`</p>`):

Let's quickly review each part of the tag pictured:

1. HTML Tag - The element name, surrounded by an opening (**<**) and closing (**>**) angle bracket.
2. HTML element (or simply, element) - a unit of content in an HTML document formed by HTML tags and the text or media it contains.
3. Opening tag - the first HTML tag used to start an HTML element. The tag type is surrounded by opening and closing angle brackets.
4. Element content - The information (text or other elements) contained between the opening and closing tags of an HTML element.
5. Closing tag - the second HTML tag used to end an HTML element. Closing tags have a forward slash (/) inside of them, directly after the left angle bracket.

Most elements require both opening and closing tags, but some call for a single self-closing tag. We'll encounter examples of both element types in the next few exercises.

Instructions

Study the diagram to the right to learn about the anatomy of HTML syntax. When you're done, continue to the next exercise.

The Head

So far you've done two things:

1. Declared to the browser that your code is HTML.
2. 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 contains the *metadata* for a web page. Metadata is information about the page that isn't displayed directly on the web page. You'll see an example of this in the next exercise.

The opening and closing head tags (`<head></head>`) typically appear as the first item after your first HTML tag.

Instructions

The empty `<head>` element is placed within the opening and closing `<html>` tags.

Page Titles

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

If you navigate to the Codecademy catalog and look at the top of your browser (or at the tab you have open), you'll notice the words `All Courses | Learn to code interactively | Codecademy`, which is the *title* of the 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>` tag.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Coding Journal</title>
  </head>
</html>
```

Where Does the Title Appear?

Good work! If the browser within the environment had a title bar, you'd see the title of the web page you added where it appears in the image to the right.

The Body

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. Only content inside the opening and closing body tags can be displayed to the screen.

Once the file has a body, many different types of content – including text, images, and buttons – can be added to the body.

```
<!DOCTYPE html>
<html>
  <head>
    <title>I'm Learning To Code!</title>
  </head>
  <body>

  </body>
</html>
```

In the example above, the opening body tag (`<body>`) is placed directly below the closing head tag (`</head>`), and the closing body tag (`</body>`) is placed directly above the closing html tag (`</html>`).

Instructions

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

Self-closing Tag

Thus far we have only seen HTML elements with an opening and a closing tag. A few types of elements, however, require only one tag.

Self-closing elements contain all the information the browser needs to render the element inside a single tag. Also, because they are single tags, they cannot wrap around raw text or other elements.

The line break element `<br />` is one example of a self-closing tag. You can use it anywhere within your HTML code. The result is a line break in the browser.

```
<p>line one<br />line two</p>
```

In the example above, the paragraph tags (`<p>`) enclose two phrases, split by a break tag (`<br />`). Note that single tags, unlike elements with two tags, can't wrap around raw text or other elements.

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

```
line one
line two
```

Without the break tag, the browser would render `line one`and `line two` on the same line.

Instructions

Add a self-closing `<br />` tag after the question mark `?`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Coding Journey</title>
  </head>
  <body>
```

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

HTML Structure

The rest of this lesson will focus on how HTML is structured and some tools developers use to make code easier to interpret.

HTML documents are organized as a collection of parent-child relationships. When an element is contained inside another element, it is considered the child of that element. The child element is said to be *nested* inside of the parent element.

```
<body>
  <p>Paragraph</p>
</body>
```

In the example above, the `<p>` element is nested inside the `<body>` element. The `<p>` element is considered a child of the `<body>` element, the parent.

Since there can be multiple levels of nesting, this analogy can be extended to grandchildren, great-grandchildren and beyond. Let's consider a more complicated example:

```
<body>
  <div>
    <h1>Student</h1>
    <p>Get Started</p>
  </div>
</body>
```

In this example, the `<body>` element is the parent of the `<div>` element. Both the `<h1>` and `<p>` elements are children of the `<div>` element. Because the `<h1>` and `<p>`elements are in the same level, they are considered siblings, and are both grandchildren of the `<body>` element.

Understanding this hierarchy is important, because child elements can inherit attributes from their parent element.

Instructions

Add the paragraph below as a child of the `div`

```
<!DOCTYPE html>
<html>

  <head>
      <title>Hello World</title>
```

```
  </head>

  <body>
        <h1>Hello World</h1>
  <p>This paragraph is a child of the body element</p>
        <div>
    <p>This paragraph is a child of the div element</p>
        </div>

  </body>
</html>
```

Whitespace

As the code in an HTML file grows, it becomes increasingly difficult to keep track of how elements are related. Programmers use two tools to visualize the relationship between elements: *whitespace* and *indentation*.

Both tools take advantage of the fact that the position of elements in a browser is independent of the amount of whitespace or indentation in the **index.html** file.

For example, if you wanted to increase the space between two paragraphs on your web page, you would *not* be able to accomplish this by adding space between the paragraph elements in the **index.html** file. The browser ignores *whitespace* in HTML files when it renders a web page, so it can be used as a tool to make code easier to read and follow.

What makes the example below difficult to read?

```
<body><p>Paragraph 1</p><p>Paragraph 2</p></body>
```

You have to read the entire line to know what elements are present. Compare the example above to this:

```
<body>
<p>Paragraph 1</p>
<p>Paragraph 2</p>
</body>
```

This example is easier to read, because each element is on its own line. While the first example required you to read the entire line of code to identify the elements, this example makes it easy to identify the body tag and two paragraphs.

A browser renders both examples the same way:

```
Paragraph 1
Paragraph 2
```

In the next exercise you will learn how to use indentation to help visualize nested elements.

Instructions

Use whitespace to make the code more readable by putting each element on its own line.

```
<!DOCTYPE html>
<html>
<body>
<h1>Whitespace</h1>
<p>Whitespace and indentation make html documents easier to read.</p>
</body>
</html>
```

Indentation

The second tool web developers use to make the structure of code easier to read is *indentation*.

The World Wide Web Consortium, or W3C, is responsible for maintaining the style standards of HTML. At the time of writing, the W3C recommends 2 spaces of indentation when writing HTML code. Although your code will work without exactly two spaces, this standard is followed by the majority of professional web developers. Indentation is used to easily visualize which elements are nested within other elements.

```
<body>
  <p>Paragraph 1</p>
  <div>
    <p>Paragraph 2</p>
  </div>
</body>
```

In the example above, `Paragraph 1` and the `<div>` tag are nested inside of the `<body>` tag, so they are indented two spaces. The `Paragraph 2` element is nested inside of the `<div>` tag, so it is indented an additional two spaces.

The spaces are inserted using the spacebar on your keyboard.

Instructions

Indent the code in **index.html** to match the W3C standards.

```
<body>
 <h1>Whitespace</h1>
 <div>
  <p>Whitespace and indentation make html documents easier to read.</p>
 </div>
</body>
```

Comments

HTML files also allow you to add comments to your code.

Comments begin with `<!--` and end with `-->`. Any characters in between will be ignored by your browser.

```
<!-- 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 this example, the comment is used to denote that the following text makes up a particular section of the page.

```
<!-- <p> Test Code </p> -->
```

In the example above, a valid HTML element (a paragraph element) has been "commented out." This practice is useful when there is code you want to experiment with, or return to, in the future.

Instructions

Add a comment to **index.html** explaining the purpose of the code.

Comment out the paragraph:

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

```
<!-- <p> Test Code </p> -->
<!DOCTYPE html>
<html>
  <head>
    <title>My Coding Journey</title>
  </head>
  <body>
<!-- <p> Practice Test Code </p> -->
    <p>Shall I compare thee to a summer's day?<br /> Thou art more lovely and more temperate</p>
  </body>
</html>
```

Review

Congratulations on completing the first lesson of HTML & CSS! You are well on your way to becoming a skilled web developer.

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

1. **HTML** stands for **H**yper**T**ext **M**arkup **L**anguage and is used to create the structure and content of a webpage.
2. Most HTML elements contain opening and closing tags with raw text or other HTML tags between them.
3. Single-closing tags cannot enclose raw text or other elements.
4. Comments are written in HTML using the following syntax: `<!-- comment -->`.
5. HTML elements can be nested inside other elements. The enclosed element is the child of the enclosing parent element.
6. Whitespace between HTML elements helps make code easier to read while not changing how elements appear in the browser.
7. Indentation also helps make code easier to read. It makes parent-child relationships visible.
8. The `<!DOCTYPE html>` declaration should always be the first line of code in your HTML files.
9. The `<html>` element will contain all of your HTML code.
10. Information about the web page, like the title, belongs within the `<head>` of the page.
11. You can add a title to your web page by using the `<title>` element, inside of the head.
12. A webpage's title appears in a browser's tab.
13. Code for visible HTML content is placed inside of the `<body>` element.

What you learned in this lesson constitutes the required setup for all HTML files. The rest of the course will teach you more about how to add content using HTML and how to style that content using CSS!

Instructions

Add a body to the web page.

Copy and paste the following line of code within the body of the **index.html** file:

```
<h1>Hello World!</h1>
```

```
<!DOCTYPE html>
<html>
  <head>
        <title>My Coding Journey</title>
  </head>
  <body>
    <h1>Hello World!</h1>
      </body>
</html>
```

Day 2 **HTML Tags**

make lists, embed images, and link out to other resources.

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>
```

Below you'll add the headings you saw in Exercise 1. This is how we'll begin to recreate the website we previewed in the opening exercise!

Instructions

In **index.html**, add an `<h1>` heading under the opening body tag. The heading should say: `The Brown Bear`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Brown Bears</title>
  </head>
  <body>
    <h1>The Brown Bear</h1>
    <h2>About Brown Bears</h2>
    <h3>Species</h3>
    <h3>Features</h3>
```

```
    <h2>Habitat</h2>
    <h3>Countries with Large Brown Bear Populations</h3>
    <h3>Countries with Small Brown Bear Populations</h3>
    <h2>Media</h2>
  </body>
</html>
```

Text Content Tags

Headings are meant to emphasize or enlarge only a few words.

If you want to add blocks of text in HTML, you can use a *paragraph*, *div*, or *span*:

1. *Paragraphs* (`<p>`) simply contain a block of plain text.
2. `<div>`s can contain any text or other HTML elements. They are primarily used to *divide* HTML documents into sections.
3. `<span>`s contain short pieces of text or other HTML. They are primarily used to wrap small pieces of content that are on the same line as other content and do not break text into different sections.

Take a look at each of these elements in action below:

```
<div>
    <p>Technology</p>
</div>
<div>
  <p><span>Self-driving cars</span> are anticipated to replace up to 2 million jobs over the next two decades.</p>
</div>
```

In the example above, there are two different `<div>`s that each contain `<p>` elements. The second `<div>` contains a `<p>` with `<span>Self-driving cars</span>`. This `<span>`element separates `Self-driving cars` from the rest of the text in the paragraph.

As we noted above, the `<div>`s divided *blocks* of code, while the `<span>` divides *inline text*, or text that is on the same line as other text.

Below, we're going to add several `<div>`s with `id`s to organize our text into sections. Later, we will use this structure to create links that allow a user to quickly navigate the content of our page.

Note: We will explain the purpose of the `id`s that you will see in a later exercise. An `id` only needs to be in the opening tag of an element.

Instructions

Below the `<h1>` element that says `The Brown Bear`, add this opening `<div>` tag: `<div id="introduction">`.

Add the closing `</div>` tag after the `<h3>` element that says `Features`.

Always add two spaces of indentation when you nest elements inside of `<div>`s.

==========

```
<!DOCTYPE html>
<html>
  <head>
    <title>Brown Bears</title>
  </head>
  <body>
    <h1>The Brown Bear</h1>
    <h2>About Brown Bears</h2>
    <h3>Species</h3>
    <h3>Features</h3>
    <h2>Habitat</h2>
    <h3>Countries with Large Brown Bear Populations</h3>
    <h3>Countries with Small Brown Bear Populations</h3>
    <h2>Media</h2>
  </body>
</html>
```

==========

```
<!DOCTYPE html>
<html>
  <head>
    <title>Brown Bears</title>
  </head>
  <body>
    <h1>The Brown Bear</h1>
    <div id="introduction">
      <h2>About Brown Bears</h2>
      <p>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.</p>
```

```
    <h3>Species</h3>
    <h3>Features</h3>
    <p>Brown bears are not always completely brown. Some can be reddish or yellowish. They have very large, curved claws and huge paws. Male brown bears are often 30% larger than female brown bears. They can range from 5 feet to 9 feet from head to toe.</p>
  </div>
  <div id="habitat">
  <h2>Habitat</h2>
  <h3>Countries with Large Brown Bear Populations</h3>
  <h3>Countries with Small Brown Bear Populations</h3>
    <p>Some countries with smaller brown bear populations include Armenia, Belarus, Bulgaria, China, Finland, France, Greece, India, Japan, Nepal, Poland, Romania, Slovenia, Turkmenistan, and Uzbekistan.</p>
  </div>
  <div id="media">
  <h2>Media</h2>
  </div>
 </body>
</html>
```

==========

Text Style Tags

Tags provided by HTML exist to organize and describe the content of web pages. Two of these HTML tags are `<em>` and `<strong>`. They are used to signal that the text within them should be "emphasized" or "strong."

Later, when you begin to style websites you will decide how you want browsers to display content within `<em>` and `<strong>` tags. However, browsers have built-in style sheets that will generally style these tags in this manner:

1. The `<em>` tag will generally render as *italic* emphasis.
2. The `<strong>` will generally render as **bold** emphasis.

Take a look at each emphasis in action:

```
<p><strong>The Nile River</strong> is the <em>longest</em> river in the world, measuring over 6,850 kilometers long (approximately 4,260 miles).</p>
```

In this example, the `<strong>` and `<em>` tags are used to emphasize the text to produce the following:

The Nile River is the *longest* river in the world, measuring over 6,850 kilometers long (approximately 4,260 miles).

Notice, `The Nile River` is bolded and `longest` is in italics.

Now, it's your turn to get some practice.

==========

```
<!DOCTYPE html>
<html>
  <head>
    <title>Brown Bears</title>
  </head>
  <body>
    <h1>The Brown Bear</h1>
    <div id="introduction">
      <h2>About Brown Bears</h2>
      <p>The brown bear (<em>Ursus arctos</em>) is native to parts of northern Eurasia and North America. Its conservation status is currently <strong>Least Concern</strong>. There are many subspecies within the brown bear species, including the Atlas bear and the Himalayan brown bear.</p>
      <h3>Species</h3>
      <h3>Features</h3>
      <p>Brown bears are not always completely brown. Some can be reddish or yellowish. They have very large, curved claws and huge paws. Male brown bears are often 30% larger than female brown bears. They can range from 5 feet to 9 feet from head to toe.</p>
    </div>
    <div id="habitat">
      <h2>Habitat</h2>
      <h3>Countries with Large Brown Bear Populations</h3>
      <h3>Countries with Small Brown Bear Populations</h3>
      <p>Some countries with smaller brown bear populations include Armenia, Belarus, Bulgaria, China, Finland, France, Greece, India, Japan, Nepal, Poland, Romania, Slovenia, Turkmenistan, and Uzbekistan.</p>
    </div>
```

```
    <div id="media">
      <h2>Media</h2>
    </div>
  </body>
</html>
```

==========

https://www.codecademy.com/courses/freelance-one-unit-1/lessons/common-elements/exercises/line-breaks?action=lesson_resume&program_content_id=d926adb83008389f400dc19128f02799&program_id=d3ed66f81f36bd54a853f65dd2afca4c

5/15

Line Breaks

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

```
<p>The Nile River is the longest river <br /> in the world, measuring over 6,850<br /> kilometers long (approximately 4,260 <br /> miles).</p>
```

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

```
The Nile River is the longest river
in the world, measuring over 6,850
kilometers long (approximately 4,260
miles).
```

You may see line breaks written as `<br />` or `<br>`. Both are valid break tags.

Note: Line breaks are not the standard way of manipulating the position of HTML elements, but it's likely that you'll come across them every now and then.

Unordered Lists

So far, all text has been in paragraph form. What if you want to display content in an easy-to-read list? In HTML, you can use an *unordered list* tag (`<ul>`) to create a list of items in no particular order. An unordered list outlines individual *list items* with a bullet point.

The `<ul>` element cannot hold raw text and cannot automatically format raw text into an unordered list of items. Individual list items must be added to the unordered list using the `<li>` tag. The `<li>` or list item tag is used to describe an item in a list.

```
<ul>
  <li>Limes</li>
```

```
  <li>Tortillas</li>
  <li>Chicken</li>
</ul>
```

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

Ordered Lists

Great job! Some lists, however, will require a bit more structure. HTML provides the *ordered list* for when you need the extra ordering that unordered lists don't provide.

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

```
<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>
```

Images

All of the elements you've learned about so far (headings, paragraphs, lists, and spans) share one thing in common: they're composed entirely of text! What if you want to add content to your web page that isn't composed of text, like images?

The `<img>` tag allows you to add an image to a web page. This is another example of a self-closing tag.

```
<img src="image-location.com" />
```

The `<img>` tag has a required attribute called `src`. The `src`attribute must be set to the image's *source*, or the location of the image. In this case, the value of `src` must be the *uniform resource locator* (URL) of the image. A URL is the web address or local address where a file is stored. Note that the end of the `<img>` tag has a forward slash `/`. Self-closing tags may include or omit the final slash — both will render properly.

This example is our first mention of an attribute. However, the `id`s that we added to our `div` tags earlier are also attributes!

Attributes provide 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

Instructions

Under the `Media` heading, add an image. Use the following URL as the source (`src`) for the image:
`https://s3.amazonaws.com/codecademy-content/courses/web-101/web101-image_brownbear.jpg`

Videos

In addition to images, HTML also supports displaying videos. Like the `<img>` tag, the `<video>` tag requires a `src` attribute with a link to the video source. Unlike the `<img>` tag however, the `<video>` element requires an opening and a closing tag.

```
<video src="myVideo.mp4" width="320" height="240" controls>
  Video not supported
</video>
```

In this example, the video source (`src=`) is "myVideo.mp4." The source must link to a video file, not to a video on another site. After the `src` attribute, the `width` and `height`attributes are used to set the size of the video displayed in the browser. The `controls` attribute instructs the browser to include basic video controls: pause, play and skip. The text, `Video not supported`, between the opening and closing video tags will only be displayed if the browser is unable to load the video.

Linking Out

You're off to a great start! So far you've learned how to add headings, paragraphs, lists, images, and videos 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 that will lead users to the actual Wikipedia page?

The anchor element in the example above is incomplete without the `href` *attribute*. This attribute stands for *hyperlink reference* and is used to link to a *file path*, or the address to where a file is located (whether it is on your computer or another location).

```
<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 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. These terms are often used interchangeably.

=========

<!DOCTYPE html>

<html>

```
<head>
  <title>Brown Bears</title>
</head>

<body>
  <h1>The Brown Bear</h1>
  <div id="introduction">
    <h2>About Brown Bears</h2>
    <p>The brown bear (<em>Ursus arctos</em>) is native to parts of northern Eurasia and North America. Its conservation status is currently <strong>Least Concern</strong>.<br /><br /> There are many subspecies within the brown bear species, including the Atlas bear and the Himalayan brown bear.</p>
    <a href="https://en.wikipedia.org/wiki/Brown_bear">Learn More</a>
    <h3>Species</h3>
    <ol>
      <li>Arctos</li>
      <li>Collarus</li>
      <li>Horribilis</li>
      <li>Nelsoni (extinct)</li>
    </ol>
    <h3>Features</h3>
    <p>Brown bears are not always completely brown. Some can be reddish or yellowish. They have very large, curved claws and huge paws. Male brown bears are often 30% larger than female brown bears. They can range from 5 feet to 9 feet from head to toe.</p>
  </div>
  <div id="habitat">
    <h2>Habitat</h2>
    <h3>Countries with Large Brown Bear Populations</h3>
    <ol>
      <li>Russia</li>
      <li>United States</li>
      <li>Canada</li>
    </ol>
    <h3>Countries with Small Brown Bear Populations</h3>
```

```
    <p>Some countries with smaller brown bear populations include Armenia, Belarus, Bulgaria, China, Finland, France, Greece, India, Japan, Nepal, Poland, Romania, Slovenia, Turkmenistan, and Uzbekistan.</p>
  </div>
  <div id="media">
    <h2>Media</h2>
    <img src="https://s3.amazonaws.com/codecademy-content/courses/web-101/web101-image_brownbear.jpg" />
    <video src="https://s3.amazonaws.com/codecademy-content/courses/freelance-1/unit-1/lesson-2/htmlcss1-vid_brown-bear.mp4" height="240" width="320" controls>Video not supported</video>
  </div>
</body>

</html>
```

==========

New Page

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 `<a>` 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, setting the `target` attribute to `"_blank"` instructs the browser to open 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.

Relative

Thus far you have learned how to link to external web pages. Many sites also link to internal web pages like `Home`, `About`, and `Contact`.

Before we learn how to link between internal pages, let's establish where our files are stored. When making multi-page static websites, web developers often store HTML files in the *root directory*, or a main folder where all the files for the project are stored. As the size of the projects you create grows, you may use additional folders within the main project folder to organize your code.

```
about.html
contact.html
index.html
```

The example above shows three different files — **about.html**, **contact.html**, and **index.html** in one folder.

If the browser is currently displaying **index.html**, it knows that **about.html** and **contact.html** are in the same folder as **index.html**, also referred to as the *current* folder. Since the browser knows the current folder, other files in the folder can be linked using a *relative path*.

A relative path is a filename that shows the path to a *local file*(a file on the same website, such as `./index.html`) versus an absolute path (a full url, like `www.codecademy.com/learn/ruby` which is stored in a different folder). The `./` in `./index.html` tells the browser to look for the file in the current folder.

```
<a href="./contact.html">Contact</a>
```

In this example, the `<a>` tag is used with a relative path to link from the current HTML file to the `contact.html` file in the same folder. On the web page, `Contact` will appear as a link.

Same Page

At this point, we have all the content we want on our page. Since we have so much content, it doesn't all fit on the screen. How do we make it easier for a user to jump to different portions of our page? When users visit our site, we want them to be able to click a link and have the page automatically scroll to a specific section.

In order to link to a *target* on the same page, we must give the target an *id*, like this:

```
<p id="top"> This is the top of the page! </p>
<h1 id="bottom">This is the bottom! </h1>
```

In this example, the `<p>` element contains `id` of top and the `<h1>` element contains `id` of bottom. An `id` can be added to most elements on a webpage.

An `id` should be descriptive to make it easier to remember the purpose of a link. The target link is a string containing the `#` character and the target element's `id`.

```
<ol>
  <li><a href="#top">Top</a></li>
  <li><a href="#bottom">Bottom</a></li>
</ol>
```

In the example above, the links to `<p id="top">` and `<h1 id="bottom">` are embedded in an ordered list. These links appear in the browser as a numbered list of links. This is why we have been adding `id`s to our `divs` all along!

13/15==========

```
<!DOCTYPE html>
<html>

<head>
  <title>Brown Bears</title>
</head>

<body>
  <a href="./index.html">Brown Bear</a>
  <a href="./aboutme.html">About Me</a>
  <h1>The Brown Bear</h1>
  <ul>
    <li><a href="#introduction">Introduction</a></li>
<li><a href="#habitat">Habitat</a></li>
<li><a href="##media">Media</a></li>
  </ul>
  <div id="introduction">
    <h2>About Brown Bears</h2>
    <p>The brown bear (<em>Ursus arctos</em>) is native to parts of northern Eurasia and North America. Its conservation status is currently <strong>Least Concern</strong>.<br /><br /> There are many subspecies within the brown bear species, including the
      Atlas bear and the Himalayan brown bear.</p>
    <a href="https://en.wikipedia.org/wiki/Brown_bear" target="_blank">Learn More</a>
    <h3>Species</h3>
    <ol>
      <li>Arctos</li>
      <li>Collarus</li>
      <li>Horribilis</li>
      <li>Nelsoni (extinct)</li>
```

```
    </ol>
    <h3>Features</h3>
    <p>Brown bears are not always completely brown. Some can be reddish or yellowish. They have very large, curved claws and huge paws. Male brown bears are often 30% larger than female brown bears. They can range from 5 feet to 9 feet from head to toe.</p>
  </div>
  <div id="habitat">
    <h2>Habitat</h2>
    <h3>Countries with Large Brown Bear Populations</h3>
    <ol>
      <li>Russia</li>
      <li>United States</li>
      <li>Canada</li>
    </ol>
    <h3>Countries with Small Brown Bear Populations</h3>
    <p>Some countries with smaller brown bear populations include Armenia, Belarus, Bulgaria, China, Finland, France, Greece, India, Japan, Nepal, Poland, Romania, Slovenia, Turkmenistan, and Uzbekistan.</p>
  </div>
  <div id="media">
    <h2>Media</h2>
    <img src="https://s3.amazonaws.com/codecademy-content/courses/web-101/web101-image_brownbear.jpg" />
    <video src="https://s3.amazonaws.com/codecademy-content/courses/freelance-1/unit-1/lesson-2/htmlcss1-vid_brown-bear.mp4" height="240" width="320" controls>Video not supported</video>
  </div>
</body>

</html>
```

==========

Navigation

In the previous two exercises, you added numerous links to your page that allow a user to navigate content on the same page, to other pages on the same website, or to external websites.

Linking to elements on the same page or to other pages on the same site is called navigation. HTML has a special tag called `<nav>` that is used to wrap these links in order to organize the content on your web page.

Some of the tags we have used, such as `<div>`, are called *non-semantic* tags. This means that they do not describe the content that is inside of them. However, many tags are used to describe the content that they surround, which helps us modify and style our content later. These are called *semantic* tags and `<nav>` is one of them!

Day 3

quiz on html tags

Getting Started with Atom

INTRODUCTION

Text editors, also called code editors, are applications used by developers to write code. They highlight and format your code so that it's easier to read and understand. If you've used Codecademy, you're already familiar with a text editor! It’s the area you write your code in.

Text editors provide a number of advantages to web developers:

- Language-specific syntax highlighting
- Automatic code indentation
- Color schemes to suit your preferences and optimize code readability
- Plug-ins to catch errors in the code
- A tree view of your project's folders and files, so you can conveniently navigate your project
- Key shortcuts for faster development

1. CHOOSING A TEXT EDITOR

There are a number of text editors to choose from. Atom and Sublime Text are two of the most popular text editors used by developers.

Sublime Text has been the text editor of choice for many years. It is stable and reliable.

Atom was released by GitHub after Sublime Text. It's a fully customizable text editor. Since Atom is written in HTML, CSS, and JavaScript, you can customize it yourself once you've learned those languages.

Either text editor is great for development, so you can't make a bad decision here. When you are further along in your coding career, try another code editor to see what features work well with your workflow.

Exercise I: Download Atom

In this exercise, we recommend you follow these steps to download Atom.

OS X

Atom works on Macs running OS X 10.8 or later. Visit the Atom homepage and click Download For

Mac. In a few moments, Atom will appear in your Downloads folder as a .zip file:

Click on atom-mac.zip to extract the application, then drag the new icon into your Applications folder.

Double-click the application icon to load Atom and get started.

Windows

Atom supports Windows 7 and 8. Visit this webpage and download **atom-windows.zip**. In a few moments, Atom will appear in your Downloads folder as a .zip file.

Follow the instructions in the Windows Installer to get started. You can visit Atom's Windows install page for more detailed instructions.

2. DEVELOPMENT FOLDERS

Before using your text editor, it's important to establish an organized file system. As the number and size of your projects grow, it becomes increasingly important to know where to save new projects and find old projects.

Most developers store their projects in an easy-to-find directory (what you might be used to calling a folder). Here at Codecademy, we recommend naming this directory `projects`. It will store all of your coding projects. Whenever you create a new project, no matter how small, you should always make a new folder inside your `projects` directory. You will find that single-file projects can quickly turn into large, multi-folder projects.

Exercise II: Create a dev folder

Below are the steps you need to follow to create a new folder for all of your programming projects. You will also learn how to load a new project folder into Atom. For steps 1 and 2, navigate to a folder using Finder (Mac users) or My Computer (PC users).

1. Navigate to a folder you visit regularly and create a new folder called `projects`. On Mac, this may be your User account. On PC, you may want to save this on your C drive.
2. Inside the `projects` directory, create a new folder called `HelloWorld`. Everything you add to this folder will be part of your `HelloWorld` project.
3. Open Atom on your computer.
4. Atom provides a tree view of your project, so you can conveniently navigate to different folders and files. In the Atom menu bar, choose File > Add Project Folder. This will launch your file manager. Navigate to the `HelloWorld` folder and select Open. The folder will open in Atom's side pane. At this point, there should not be any contents in the folder. We'll add a file in the next exercise.

3. ADDING A FILE

When you open Atom, the Welcome Guide will appear. For now, we'll skip getting to know Atom and start writing some code.

Before you learn how to add files to a project folder, it is important to understand the purpose of file extensions.

A *file extension* is the suffix of a filename and describes the type of content the file contains. The file

extension is always the last 3 or 4 characters in a filename, preceded by a period. For example, the HTML file extension is `.html`, and it tells the browser (and other applications) to interpret the contents of the file as a web page.

Once Atom loads a project folder, you can add files. The steps below describe how to add files. Don't worry about doing this on your own computer. We'll get to that in Exercise III.

1. In Atom's top bar, select File > New File. An untitled, blank file will appear.
2. In Atom's top bar, choose File > Save or Save As. Name the file with its appropriate file extension (`.html`, `.css`, `.csv`). It is critical that you include the correct file extension, so programs know how to interpret its contents.
3. Begin coding! Save your file often. This will decrease the chances of losing unsaved work.

Exercise III: Add a file

In this exercise, you will create an **index.html** file in your Hello World project.

1. In Atom's top bar, choose File > New File. An untitled, blank file will appear.
2. Before you save the file, copy and paste the following boilerplate HTML code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <h1>Hello World</h1>
  </body>
</html>
```

Notice: All of the text in your file is the same color. This will change after you save the file as `.html`.

3. In Atom's top bar, choose File > Save or Save As.

4. Name the file **index.html**. It's crucial that you use the file extension `.html` so the text editor and web browser know how to interpret your code.

4. FILE EXTENSIONS AND SYNTAX HIGHLIGHTING

Atom and other text editors are able to interpret file extensions and provide language-specific syntax highlighting. Syntax highlighting is a tool for making code *easier* to read. Take a look at your **index.html** file. The text and tags are different colors. This is how Atom highlights **.html** syntax. With each new language you learn, Atom will highlight text in a way that makes your code easy to read. This may be different than other text editors and also different than the way your code is highlighted on Codecademy.

Exercise IV: Open your HTML File in a web browser

At this point, your file is ready to be viewed in a web browser. The following steps should be taken

outside of Atom:

1. Back in your file system, navigate to the **index.html** file in your Hello World folder.
2. Double click **index.html**. the page should open in your default web browser.

Congratulations! You can create web pages on your own computer!

Mark Complete

Day 4

Today you will start by taking a quiz to review yesterday's content on text editors and creating local websites.
You will then create a full website on your computer using all of the content you've learned in this unit. We'll be there to guide you along on this exciting task.
However, tomorrow we will take the training wheels off as you create a full website all on your own. So be sure to pay close attention to everything we do today. At the end of the day, make sure to review any content that you aren't comfortable with in preparation for tomorrow's project.
Good luck!

test your knowledge in this text editor and website basics quiz!

Project
Travel Blog

In this project you will put what you've learned to practice. In this guided practice project you will create a web site for a Paris Travel Blog.

Travel Blog
In this project, you will follow step-by-step instructions to produce a travel blog website on your own computer.
There will be a few different sections on your page: a nav bar, images from your travels, highlights from the trip, the full text of the post, and your contact information at the end of the page.
In order to complete this project, you must know how HTML code is structured, and a few basic HTML tags.
You can preview what you'll be building in the browser to the right!

Day 5

Two Sandals

In this project, you will recreate a web page that contains the restaurant's menu. The solution is provided to the right. You must use your knowledge of HTML and the website to the right to recreate it on your own computer.

In addition to the visual content in this image, the website must also contain:

1. Two HTML files in a project folder: **index.html** and **home.html**. Note: we do not provide any instructions for what you should add to **home.html**. The purpose of this file is to practice linking to an HTML document in a shared folder.
2. Links from `Home` and `Menu` to the files **home.html** and **index.html**, respectively.
3. Links from `Soups`, `Salads` and `Desserts` in the unordered list to their section on the menu.
4. The three images displayed on the web page (see image links below).

Image links:

1. Copy and paste the URL for this tomato soup picture
2. Copy and paste the URL for this caesar salad picture
3. Copy and paste the URL for this chocolate chip cookiepicture

Use the website to the right and the resources above to guide you through the project.

========

```
<!DOCTYPE html><html><head>
        <title>Two Sandals | Menu</title>
    </head>
    <body>

        <!-- Navigation Bar -->

        <nav>
            <a href="home.html">Home</a>
            <a href="./index.html">Menu</a>
        </nav>

        <!-- Welcome Section -->

        <h1>Two Sandals Soups and Salads</h1>
        <h2>Menu</h2>
```

```
<p>Here at Two Sandals Soups and Salads, we serve a delicious spread of traditional and not-so-traditional, you guessed it... soups and salads! All our ingredients are farm-to-table and free-trade certified! Come in salad-ivating. We&#39;ll make sure you leave soup-er salad-isfied.</p>
<ul>
    <li><a href="#soups">Soups</a></li>
    <li><a href="#salads">Salads</a></li>
    <li><a href="#desserts">Desserts</a></li>
</ul>

<!-- Soups Section -->

<div id="soups">
    <h3>Soups</h3>
    <img src="https://s3.amazonaws.com/codecademy-content/courses/freelance-1/unit-1/freelance1-img_tomato-soup.jpg"/>
    <ol>
        <li><strong>Tomato</strong> Classic creamy tomato soup <em>$4</em></li>
        <li><strong>Chicken Noodle</strong> Chunky chicken noodle with carrots and celery <em>$4.50</em></li>
        <li><strong>Buffalo Chicken</strong> Shredded chicken in a thick cheesy buffalo sauce <em>$5</em></li>
    </ol>
</div>

<!-- Salads Section -->

<div id="salads">
    <h3>Salads</h3>
    <img src="https://s3.amazonaws.com/codecademy-content/courses/freelance-1/unit-1/freelance1-img_caesar-salad.jpg"/>
    <ol>
        <li><strong>The Caesar</strong> The classic with a light sprinkle of parmesan cheese <em>$6</em></li>
```

```
                    <li><strong>The California</strong> Romaine lettuce with avocado and cheddar cheese, with a light vinaigrette <em>$7</em></li>
                    <li><strong>Key Lime Thai</strong> Iceberg lettuce with a light lime vinaigrette and onion crisps <em>$8</em></li>
                </ol>
            </div>

            <!-- Desserts Section -->

            <div id="desserts">
                <h3>Desserts</h3>
                <img src="https://s3.amazonaws.com/codecademy-content/courses/freelance-1/unit-1/freelance1-img_chocchip-cookie.jpg"/>
                <ol>
                    <li><strong>Chocolate Chip Cookie</strong> Enough said <em>$2</em> </li>
                    <li><strong>Brownie Sundae</strong> Vanilla ice cream with chocolate sauce, nuts and a cherry on top <em>$6</em></li>
                </ol>
            </div>
```