Preparing for HTML

Now that we’ve learned about some of the most common HTML elements, it’s time to learn how to set up an HTML file.

HTML files require certain elements to set up the document properly. We can let web browsers know that we are using HTML by starting our document with a *document type declaration*.

The declaration looks like this:

<!DOCTYPE html>

This declaration is an instruction, and it must be the first line of code in your HTML document. It tells the browser what type of document to expect, along with what version of HTML is being used in the document. 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. To make sure your document is forever interpreted correctly, always include <!DOCTYPE html> at the very beginning of your HTML documents.

Lastly, HTML code is always saved in a file with an **.html** extension.

The <html> tag

The <!DOCTYPE html> declaration provides the browser with two pieces of information (the type of document and the HTML version to expect), 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 after declaring <!DOCTYPE html>:

<!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.

The Head

So far you’ve done two things to set up the file properly:

* Declared to the browser that your code is HTML with <!DOCTYPE html>
* Added the HTML element (<html>) that will contain the rest of your code.

We have added these elements to the Brown Bears page you previously created. Now, let’s also give the browser some information about the page itself. We can do this by adding a <head> element.

Remember the <body> tag? The <head> element is part of this HTML metaphor. It goes above our <body> 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. Unlike the information inside of the <body> tag, the metadata in the head is information about the page itself. You’ll see an example of this in the next exercise.

The opening and closing head tags typically appear as the first item after your first HTML tag:

<head>  
</head>

Page Titles

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

If you navigate to the Codecademy [catalog](https://www.codecademy.com/learn/all)and look at the top of your browser, you’ll notice the words Full Catalog Courses & Tutorials | Codecademy, which is the *title* of the web page.

A browser’s tab displays the title specified in the <title> tag. The <title> tag is always inside of the <head>.

<!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).

Where Does the Title Appear?

Good work! Unfortunately, the browser panel used by the Codecademy environment does not have a title bar, so the “Brown Bear” title you wrote in the previous exercise will not be displayed. Outside of the Codecademy environment, however, your title would appear as depicted in the diagram to the right.

So far, we have learned about:

* <!DOCTYPE html>, the declaration specifying the version of HTML for the browser
* The <html> tags that enclose all of your HTML code
* The <head> tag that contains the metadata of a webpage, such as its <title>

Next, you will learn about new types of elements that go inside the body.

Page Titles

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

If you navigate to the Codecademy [catalog](https://www.codecademy.com/learn/all)and look at the top of your browser, you’ll notice the words Full Catalog Courses & Tutorials | Codecademy, which is the *title* of the web page.

A browser’s tab displays the title specified in the <title> tag. The <title> tag is always inside of the <head>.

<!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).

Where Does the Title Appear?

Good work! Unfortunately, the browser panel used by the Codecademy environment does not have a title bar, so the “Brown Bear” title you wrote in the previous exercise will not be displayed. Outside of the Codecademy environment, however, your title would appear as depicted in the diagram to the right.

So far, we have learned about:

* <!DOCTYPE html>, the declaration specifying the version of HTML for the browser
* The <html> tags that enclose all of your HTML code
* The <head> tag that contains the metadata of a webpage, such as its <title>

Next, you will learn about new types of elements that go inside the body.

Linking to Other Web Pages

One of the powerful aspects of HTML (and the Internet), is the ability to *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 *path*, or the address to where a file is located (whether it is on your computer or another location). The paths provided to the href attribute are often URLs.

<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.

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.

Opening Links in a New Window

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 how a link should open.

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.

In this exercise, we’ve used the terminology “open in a new window.” It’s 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="\_blank" attribute, when used in modern browsers, will open new websites in a new tab.

Linking to Relative Page

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.

project-folder/  
|—— about.html  
|—— contact.html  
|—— index.html

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

HTML files are often stored in the same folder, as shown in the example above. If the browser is currently displaying **index.html**, it also knows that **about.html** and **contact.html** are in the same folder. Because the files are stored in the same folder, we can link web pages together using a *relative path*.

<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.

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 https://www.codecademy.com/learn/learn-html which is stored in a different folder). The ./ in ./index.html tells the browser to look for the file in the current folder.

Linking At Will

You’ve probably visited websites where not all links were made up of text. Maybe the links you clicked on were images or some other form of content.

So far, we’ve added links that were made up of only text, like the following:

<a href="https://en.wikipedia.org/wiki/Opuntia" target="\_blank">Prickly Pear</a>

Text-only links, however, would significantly decrease your flexibility as a web developer!

Thankfully, HTML allows you to turn nearly any element into a link by wrapping that element with an anchor element. With this technique, it’s possible to turn images into links by simply wrapping the <img> element with an <a> element.

<a href="https://en.wikipedia.org/wiki/Opuntia" target="\_blank"><img src="https://www.Prickly\_Pear\_Closeup.jpg" alt="A red prickly pear fruit"/></a>

In the example above, an image of a prickly pear has been turned into a link by wrapping the outside of the <img> element with an <a> element.

Linking to 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 is assigned an id of “top” and the <h1> element is assigned “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. An id is especially helpful for organizing content belonging to a div!

Whitespace

The rest of this lesson will focus on some tools developers use to make code easier to interpret.

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.

Indentation

The second tool web developers use to make the structure of code easier to read is *indentation*. The spaces are inserted using the space and tab bars on your keyboard.

The [World Wide Web Consortium](https://www.w3.org/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.

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.

HTML Tags

You now know all of the basic elements and set-up you need to structure an HTML page and add different types of content. With the help of CSS, very soon you’ll be creating beautiful websites!

While some tags have a very specific purpose, such as image and video tags, most tags are used to describe the content that they surround, which helps us modify and style our content later. There are seemingly infinite numbers of tags to use (many more than we’ve taught). Knowing when to use each one is based on how you want to describe the content of your HTML. Descriptive, well-chosen tags are one key to high-quality web development. A full list of available HTML tags can be found [in Mozilla documentation](https://developer.mozilla.org/en-US/docs/Web/HTML/Element).

Let’s review what you’ve learned this lesson:

1. The <!DOCTYPE html> declaration should always be the first line of code in your HTML files. This lets the browser know what version of HTML to expect.
2. The <html> element will contain all of your HTML code.
3. Information about the web page, like the title, belongs within the <head> of the page.
4. You can add a title to your web page by using the <title> element, inside of the head.
5. A webpage’s title appears in a browser’s tab.
6. Anchor tags (<a>) are used to link to internal pages, external pages or content on the same page.
7. You can create sections on a webpage and jump to them using <a> tags and adding ids to the elements you wish to jump to.
8. Whitespace between HTML elements helps make code easier to read while not changing how elements appear in the browser.
9. Indentation also helps make code easier to read. It makes parent-child relationships visible.
10. Comments are written in HTML using the following syntax: <!-- comment -->.

Take some time to edit the workspace you created and observe how it changes!

**Instructions**

Congratulations on completing this lesson!

If you want to review how to use HTML in a project, watch the video below and follow along with one of our experts:

https://www.youtube.com/watch?v=B4tCt6elrU0&feature=emb\_title