Introduction to HTML

Qitian Liao

University of California, Berkeley

Contents

1 Introduction to HTML

Now we will learn the basic structure of an HTML document.

1.1 Introduction to HTML

HTML is the skeleton of all web pages. It is often the first language learned by developers, marketers, and designers and is core to front-end development work. HTML provides structure to the content appearing on a website, such as images, text, or videos. Right-click on any page on the internet, choose "Inspect," and we will see HTML in a panel of your screen.

HTML stands for HyperText Markup Language:

- A *markup* language is a computer language that defines the structure and presentation of raw text.

- In HTML, the computer can interpret *raw text* that is wrapped in HTML elements.

- *HyperText* is text displayed on a computer or device that provides access to other text through links, also known as *hyperlinks*.

Learning HTML is the first step in creating websites, but even a bit of knowledge can help us inject code snippets into newsletter, blog or website templates. As we continue learning, we can layer HTML with CSS and JavaScript to create visually compelling and dynamic websites. Now, we are going to focus on how to add and modify basic content on a page, like text, images, and videos.

1.2 HTML Anatomy

HTML is composed of *elements*. These elements structure the webpage and define its content. Let us take a look at how they are written.
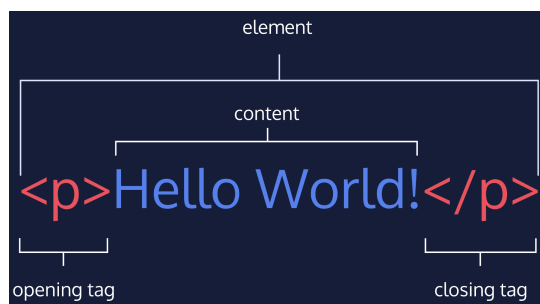
The diagram above displays an HTML paragraph element. As we can see, the paragraph element is made up of:

- An *opening tag* (`<p>`)

- The content ("Hello World!" text)

- A *closing tag* (`</p>`)

A *tag* and the *content* between it is called an HTML element. There are many tags that we can use to organize and display text and other types of content, like images.

Let us quickly review each part of the element pictured:

- HTML element (or simply, element) — a unit of content in an HTML document formed by HTML tags and the text or media it contains.

- HTML Tag — the element name, surrounded by an opening (<) and closing (>) angle bracket.

- Opening Tag — the first HTML tag used to start an HTML element. The tag type is surrounded by opening and closing angle brackets.

- Content — The information (text or other elements) contained between the opening and closing tags of an HTML element.

- 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.

1.3 The Body

One of the key HTML elements we use to build a webpage is the *body* element. Only content inside the opening and closing body tags can be displayed to the screen. Here is what opening and closing body tags look like:

```
1  <body>
2
3  </body>
```

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

```
1  <body>
2    <p>Hello, doc!</p>
3  </body>
```

1.4 HTML Structure

HTML is organized as a collection of family tree relationships. As you saw in the last exercise, we placed `<p>` tags within `<body>` tags. 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.

```
1  <body>
2    <p>This paragraph is a child of the body</p>
3  </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, and the `<body>` element is considered the parent. Notice that we have added two spaces of indentation (using the `space` bar) for better readability.

Since there can be multiple levels of nesting, this analogy can be extended to grandchildren, great-grandchildren, and beyond. The relationship between elements and their ancestor and descendent elements is known as *hierarchy*.

Let us consider a more complicated example that uses some new tags:

```
1  <body>
2    <div>
3      <h1>Sibling to p, but also grandchild of body</h1>
```

```
4        <p>Sibling to h1, but also grandchild of body</p>
5      </div>
6  </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 at the same level, they are considered siblings and are both grandchildren of the `<body>` element.

Understanding HTML hierarchy is important because child elements can inherit behavior and styling from their parent element. We will learn more about webpage hierarchy when you start digging into CSS.

1.5   Headings

Headings in HTML are similar 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:

```
1  <h1>BREAKING NEWS</h1>
```

1.6   Divs

One of the most popular elements in HTML is the `<div>` element. `<div>` is short for "division" or a container that divides the page into sections. These sections are very useful for grouping elements in your HTML together.

```
1  <body>
2    <div>
3      <h1>Why use divs?</h1>
4      <p>Great for grouping elements!</p>
5    </div>
6  </body>
```

`<div>`s can contain any text or other HTML elements, such as links, images, or videos. Remember to always add two spaces of indentation when you nest elements inside of `<div>`s for better readability.

1.7 Attributes

If we want to expand an element's tag, we can do so using an *attribute*. Attributes are content added to the opening tag of an element and can be used in several different ways, from providing information to changing styling. Attributes are made up of the following two parts:

- The *name* of the attribute

- The *value* of the attribute

One commonly used attribute is the id. We can use the id attribute to specify different content (such as <div>s) and is really helpful when you use an element more than once. ids have several different purposes in HTML, but for now, we will focus on how they can help us identify content on our page.

When we add an id to a <div>, we place it in the opening tag:

```
1  <div id="intro">
2    <h1>Introduction</h1>
3  </div>
```

1.8 Displaying Text

If you want to display text in HTML, you can use a *paragraph* or *span*:

- *Paragraphs* (<p>) contain a block of plain text.

- <span> contains short pieces of text or other HTML. They are used to separate small pieces of content that are on the same line as other content.

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

```
1  <div>
2    <h1>Technology</h1>
3  </div>
4  <div>
5    <p><span>Self-driving cars</span> are anticipated to
6    replace up to 2 million jobs over the next two decades.</p>
7  </div>
```

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

It is best to use a <span> element when you want to target a specific piece of content that is inline, or on the same line as other text. If you want to divide your content into blocks, it is better to use a <div>.

1.9 Styling Text

You can also style text using HTML tags. The <em> tag emphasizes text, while the <strong> tag highlights important text.

Later, when we begin to style websites, we will decide how you want browsers to display content within <em> and <strong> tags. Browsers, however, have built-in style sheets that will generally style these tags in the following ways:

- The `<em>` tag will generally render as italic emphasis.

- The `<strong>` will generally render as bold emphasis.

Take a look at each style in action:

```
1  <p><strong>The Nile River</strong> is the <em>longest</em>
2  river in the world, measuring over 6,850 kilometers long
3  (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).

As we can see, "The Nile River" is bolded and "longest" is in italics.

1.10   Line Breaks

The spacing between code in an HTML file does not 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 unique because it is only composed of a starting tag. You can use it anywhere within your HTML code and a line break will be shown in the browser.

```
1  <p>The Nile River is the longest river <br> in the world,
2  measuring over 6,850 <br> kilometers long (approximately
3  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).

1.11   Unordered Lists

In addition to organizing text in paragraph form, you can also 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 should not hold raw text and will not 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.

```
1  <ul>
2    <li>Limes</li>
3    <li>Tortillas</li>
4    <li>Chicken</li>
5  </ul>
```

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

The output will look like this:

- Limes

- Tortillas

- Chicken

1.12   Ordered Lists

Ordered lists (`<ol>`) are like unordered lists, except that each list item is numbered. They are useful when you need to list different steps in a process or rank items for first to last.

You can create the ordered list with the `<ol>` tag and then add individual list items to the list using `<li>` tags.

```
1  <ol>
2    <li>Preheat the oven to 350 degrees.</li>
3    <li>Mix whole wheat flour, baking soda, and salt.</li>
4    <li>Cream the butter, sugar in separate bowl.</li>
5    <li>Add eggs and vanilla extract to bowl.</li>
6  </ol>
```

The output will look like this:

---

1. Preheat the oven to 350 degrees.

2. Mix whole wheat flour, baking soda, and salt.

3. Cream the butter, sugar in separate bowl.

4. Add eggs and vanilla extract to bowl.

1.13   Images

All of the elements we have learned about so far (headings, paragraphs, lists, and spans) share one thing in common: they are composed entirely of text. What if we want to add content to our web page that is not composed of text, like images?

The `<img>` tag allows us to add an image to a web page. Most elements require both opening and closing tags, but the `<img>` tag is a self-closing tag. 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.

```
1  <img src="image-location.jpg" >
```

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.

1.14   Image Alts

Part of being an exceptional web developer is making our site accessible to users of all backgrounds. In order to make the Web more inclusive, we need to consider what happens when assistive technologies such as screen readers come across image tags.

The `alt` attribute, which means alternative text, brings meaning to the images on our sites. The `alt` attribute can be added to the image tag just like the `src` attribute. The value of `alt` should be a description of the image.

```
1  <img src="#" alt="A field of yellow sunflowers">
```

The `alt` attribute also serves the following purposes:

- 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.

- Visually impaired users often browse the web with the aid of screen reading software. When you include the alt attribute, the screen reading software can read the image's description out loud to the visually impaired user.

- The alt attribute also plays a role in Search Engine Optimization (SEO), because search engines cannot "see" the images on websites as they crawl the internet. Having descriptive alt attributes can improve the ranking of your site.

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 be left empty.

1.15   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.

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

In this example, the video source (`src`) is `myVideo.mp4`. The source can be a video file that is hosted alongside your webpage, or a URL that points to a video file hosted on another webpage.

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.

1.16   Review

Let us review what we have learned in this chapter:

- **HTML** stands for **H**yper**T**ext **M**arkup **L**anguage and is used to create the structure and content of a webpage.

- Most HTML elements contain opening and closing tags with raw text or other HTML tags between them.

- HTML elements can be nested inside other elements. The enclosed element is the child of the enclosing parent element.

- Any visible content should be placed within the opening and closing `<body>` tags.

- Headings and sub-headings, `<h1>` to `<h6>` tags, are used to enlarge text.

- `<p>`, `<span>` and `<div>` tags specify text or blocks.

- The `<em>` and `<strong>` tags are used to emphasize text.

- Line breaks are created with the `<br>` tag.

- Ordered lists (`<ol>`) are numbered and unordered lists (`<ul>`) are bulleted.

- Images (`<img>`) and videos (`<video>`) can be added by linking to an existing source.

Next, we will take the content that we have added to the website and transform it into an HTML document that is ready to go on the web.

2   HTML Document Standards

Now we will learn how to set up a proper HTML Document, how to link to other pages, and how to format our code for readability.

2.1   Preparing for HTML

Now it is 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:

```
1  <!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.

2.2   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 does not 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>:

```
1  <!DOCTYPE html>
2  <html>
3
4  </html>
```

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

2.3   The Head

So far we have 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.

Now, let us also give the browser some information about the page itself. We can do this by adding a <head> element.

The <head> element is part of the HTML metaphor as the <body> tag. It goes above the <body> element.

The <head> element contains the *metadata* for a web page. Metadata is information about the page that is not 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.

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

```
1  <head>
2  </head>
```

2.4   Page Titles

Let us investigate what kind of metadata about the web page can the <head> element contain.

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

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>My Coding Journal</title>
5    </head>
6  </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).

2.5   Where Does the Title Appear?

Our title would appear as depicted in the diagram below.
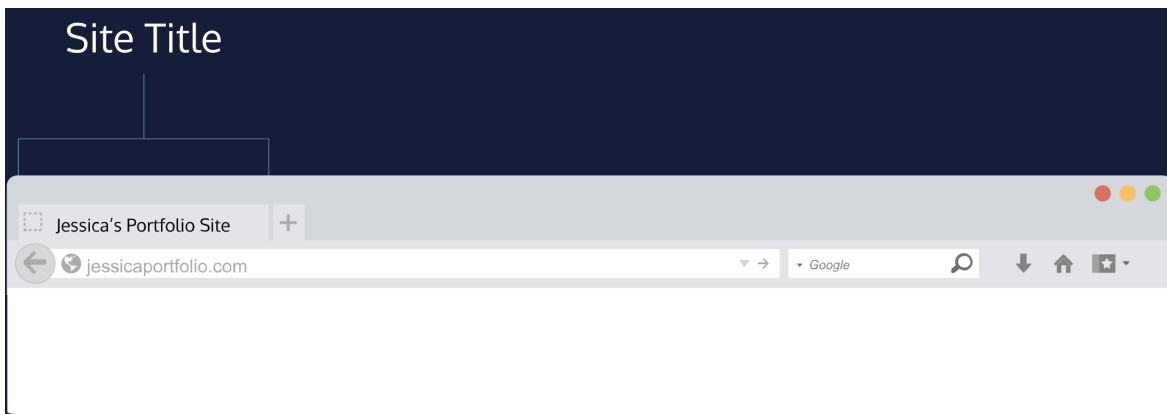
So far, we have learned about:

- <!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, we will learn about new types of elements that go inside the body.

2.6 Linking to Other Web Pages

One of the powerful aspects of HTML (and the Internet), is the ability to *link* to other web pages.

We 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.

```
1  <a>This Is a Link To Wikipedia</a>
```

Technically, the link in the example above is incomplete. The link above cannot work because 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.

```
1  <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, we may come across the term *hyperlink*. This is simply the technical term for link, and these terms are often used interchangeably.

2.7 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 is 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.

```
1  <a href="https://en.wikipedia.org/wiki/Brown_bear"
2  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 have used the terminology "open in a new window." It is 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.

2.8 Linking to Relative Page

Thus far we 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 us 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.

```
1  project-folder/
2  |__ about.html
3  |__ contact.html
4  |__ 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*.

```
1  <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.

2.9   Linking At Will

You have 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 have added links that were made up of only text, like the following:

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

Text-only links, however, would significantly decrease our flexibility as web developers.

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

```
1  <a href="https://en.wikipedia.org/wiki/Opuntia"
2  target="_blank">(*@@*)<img
3  src="https://www.Prickly_Pear_Closeup.jpg" alt="A(*@ @*)red
4  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.

2.10 Linking to Same Page

At this point, we have all the content we want on our page. Since we have so much content, it does not 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:

```
1  <p id="top">This is the top of the page!</p>
2  <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`.

```
1  <ol>
2    <li><a href="#top">Top</a></li>
3    <li><a href="#bottom">Bottom</a></li>
4  </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`.

2.11 Whitespace

Now we 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 an HTML file.

For example, if we 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 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?

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

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

```
1  <body>
2      <p>Paragraph 1</p>
3      <p>Paragraph 2</p>
4  </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:

```
1  Paragraph 1
2  Paragraph 2
```

Next we will learn how to use indentation to help visualize nested elements.

2.12   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, 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.

```
1  <body>
2    <p>Paragraph 1</p>
3    <div>
4      <p>Paragraph 2</p>
5    </div>
6  </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.

2.13   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.

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

Including comments in our code is helpful for many reasons:

1. They help us (and others) understand our code if we decide to come back and review it at a much later date.

2. They allow us to experiment with new code, without having to delete old code.

```
1  <!-- Favorite Films Section -->
2  <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.

```
1  <!-- <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.

2.14 HTML Tags

We now know all of the basic elements and set-up we need to structure an HTML page and add different types of content. With the help of CSS, very soon we will 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. 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.

Let us review what we have learned:

- The `<!DOCTYPE html>` declaration should always be the first line of code in our HTML files. This lets the browser know what version of HTML to expect.

- The `<html>` element will contain all of your HTML code.

- Information about the web page, like the title, belongs within the `<head>` of the page.

- You can add a title to your web page by using the `<title>` element, inside of the head.

- A webpage's title appears in a browser's tab.

- Anchor tags (`<a>`) are used to link to internal pages, external pages or content on the same page.

- You can create sections on a webpage and jump to them using `<a>` tags and adding `id`s to the elements you wish to jump to.

- Whitespace between HTML elements helps make code easier to read while not changing how elements appear in the browser.

- Indentation also helps make code easier to read. It makes parent-child relationships visible.

- Comments are written in HTML using the following syntax: `<!-- comment -->`.

3 HTML Tables

Now we will learn how to use tables to present tabular data to users.

3.1 Introduction to Tables

There are many websites on the Internet that display information like stock prices, sports scores, invoice data, and more. This data is naturally tabular in nature, meaning that a table is often the best way of presenting the data. Now we will learn how to use the HTML `<table>` element to present information in a two-dimensional table to the users.

3.2 Create a Table

Before displaying data, we must first create the table that will contain the data by using the `<table>` element.

```
1  <table>
2
3  </table>
```

The `<table>` element will contain all of the tabular data we plan on displaying.

3.3 Table Rows

In many programs that use tables, the table is already predefined for you, meaning that it contains the rows, columns, and cells that will hold data. In HTML, all of these components must be created.

The first step in entering data into the table is to add rows using the table row element: `<tr>`.

```
1  <table>
2    <tr>
3    </tr>
4    <tr>
5    </tr>
6  </table>
```

In the example above, two rows have been added to the table.

3.4 Table Data

Rows are not sufficient to add data to a table. Each cell element must also be defined. In HTML, you can add data using the *table data* element: `<td>`.

```
1  <table>
2    <tr>
3      <td>73</td>
4      <td>81</td>
5    </tr>
6  </table>
```

In the example above, two data points (`73` and `81`) were entered in the one row that exists. By adding two data points, we created two cells of data.

If the table were displayed in the browser, it would show a table with one row and two columns.

16

3.5 Table Headings

Table data does not make much sense without titles to describe what the data represents.

To add titles to rows and columns, you can use the table heading element: `<th>`.

The table heading element is used just like a table data element, except with a relevant title. Just like table data, a table heading must be placed within a table row.

```
1  <table>
2    <tr>
3      <th></th>
4      <th scope="col">Saturday</th>
5      <th scope="col">Sunday</th>
6    </tr>
7    <tr>
8      <th scope="row">Temperature</th>
9      <td>73</td>
10     <td>81</td>
11   </tr>
12 </table>
```

First, a new row was added to hold the three headings: a blank heading, a Saturday heading, and a Sunday heading. The blank heading creates the extra table cell necessary to align the table headings correctly over the data they correspond to.

In the second row, one table heading was added as a row title: Temperature.

Note, also, the use of the scope attribute, which can take one of two values:

1. row - this value makes it clear that the heading is for a row.

2. col - this value makes it clear that the heading is for a column.

HTML code for tables may look a little strange at first, but analyzing it piece by piece helps make the code more understandable.

3.6 Table Borders

So far, the tables we have created have been a little difficult to read because they have no borders.

In older versions of HTML, a border could be added to a table using the border attribute and setting it equal to an integer. This integer would represent the thickness of the border.

```
1  <table border="1">
2    <tr>
3      <td>73</td>
4      <td>81</td>
5    </tr>
6  </table>
```

The code in the example above is deprecated, so please do not use it. It is meant to illustrate older conventions you may come across when reading other developers' code.

The browser will likely still interpret your code correctly if you use the border attribute, but that does not mean the attribute should be used.

We use CSS to add style to HTML documents, because it helps us to separate the structure of a page from how it looks. You can achieve the same table border effect using CSS.

```
1  table , td {
2    border: 1px solid black;
3  }
```

The code in the example above uses CSS instead of HTML to show table borders.

3.7   Spanning Columns

Next let us investigate the case where the table contains data that spans multiple columns.

For example, a personal calendar could have events that span across multiple hours, or even multiple days.

Data can span columns using the colspan attribute. The attributes accepts an integer (greater than or equal to 1) to denote the number of columns it spans across.

```
1  <table>
2    <tr>
3      <th>Monday</th>
4      <th>Tuesday</th>
5      <th>Wednesday</th>
6    </tr>
7    <tr>
8      <td colspan="2">Out of Town</td>
9      <td>Back in Town</td>
10   </tr>
11 </table>
```

In the example above, the data Out of Town spans the Monday and Tuesday table headings using the value 2 (two columns). The data Back in Town appear only under the Wednesday heading.

3.8   Spanning Rows

Data can also span multiple rows using the rowspan attribute.

The rowspan attribute is used for data that spans multiple rows (perhaps an event goes on for multiple hours on a certain day). It accepts an integer (greater than or equal to 1) to denote the number of rows it spans across.

```
1  <table>
2    <tr> <!-- Row 1 -->
3      <th></th>
4      <th>Saturday</th>
5      <th>Sunday</th>
6    </tr>
7    <tr> <!-- Row 2 -->
8      <th>Morning</th>
9      <td rowspan="2">Work</td>
10     <td rowspan="3">Relax</td>
11   </tr>
12   <tr> <!-- Row 3 -->
13     <th>Afternoon</th>
```

```
14    </tr>
15    <tr> <!-- Row 4 -->
16      <th>Evening</th>
17      <td>Dinner</td>
18    </tr>
19  </table>
```

In the example above, there are four rows:

1. The first row contains an empty cell and the two column headings.

2. The second row contains the Morning row heading, along with Work, which spans two rows under the Saturday column. The "Relax" entry spans three rows under the Sunday column.

3. The third row only contains the Afternoon row heading.

4. The fourth row only contains the Dinner entry, since "Relax" spans into the cell next to it.

	Saturday	Sunday
Morning	Work	Relax
Afternoon		
Evening	Dinner	

3.9   Table Body

Over time, a table can grow to contain a lot of data and become very long. When this happens, the table can be sectioned off so that it is easier to manage.

Long tables can be sectioned off using the *table body* element: <tbody>.

The <tbody> element should contain all of the table's data, excluding the table headings.

```
1   <table>
2     <tbody>
3       <tr>
4         <th></th>
5         <th>Saturday</th>
6         <th>Sunday</th>
7       </tr>
8       <tr>
9         <th>Morning</th>
10        <td rowspan="2">Work</td>
11        <td rowspan="3">Relax</td>
12      </tr>
```

```
13          <tr>
14            <th>Afternoon</th>
15          </tr>
16          <tr>
17            <th>Evening</th>
18            <td>Dinner</td>
19          </tr>
20      </tbody>
21  </table>
```

In the example above, all of the table data is contained within a table body element. Note, however, that the headings were also kept in the table's body.

3.10 Table Head

Previously, the table's headings were kept inside of the table's body. When a table's body is sectioned off, however, it also makes sense to section off the table's column headings using the <thead> element.

```
1   <table>
2     <thead>
3       <tr>
4         <th></th>
5         <th scope="col">Saturday</th>
6         <th scope="col">Sunday</th>
7       </tr>
8     </thead>
9     <tbody>
10      <tr>
11        <th scope="row">Morning</th>
12        <td rowspan="2">Work</td>
13        <td rowspan="3">Relax</td>
14      </tr>
15      <tr>
16        <th scope="row">Afternoon</th>
17      </tr>
18      <tr>
19        <th scope="row">Evening</th>
20        <td>Dinner</td>
21      </tr>
22    </tbody>
23  </table>
```

In the example above, the only new element is <thead>. The table headings are contained inside of this element. Note that the table's head still requires a row in order to contain the table headings.

Additionally, only the **column** headings go under the <thead> element. We can use the scope attribute on <th> elements to indicate whether a <th> element is being used as a "row" heading or a "col" heading.

3.11 Table Footer

The bottom part of a long table can also be sectioned off using the <tfoot> element.

```
 1  <table>
 2    <thead>
 3      <tr>
 4        <th>Quarter</th>
 5        <th>Revenue</th>
 6        <th>Costs</th>
 7      </tr>
 8    </thead>
 9    <tbody>
10      <tr>
11        <th>Q1</th>
12        <td>$10M</td>
13        <td>$7.5M</td>
14      </tr>
15      <tr>
16        <th>Q2</th>
17        <td>$12M</td>
18        <td>$5M</td>
19      </tr>
20    </tbody>
21    <tfoot>
22      <tr>
23        <th>Total</th>
24        <td>$22M</td>
25        <td>$12.5M</td>
26      </tr>
27    </tfoot>
28  </table>
```

In the example above, the footer contains the totals of the data in the table. Footers are often used to contain sums, differences, and other data results.

3.12 Styling with CSS

Tables, by default, are very bland. They have no borders, the font color is black, and the typeface is the same type used for other HTML elements.

We can use CSS to style tables. Specifically, we can style the various aspects mentioned above.

```
1  table, th, td {
2    border: 1px solid black;
3    font-family: Arial, sans-serif;
4    text-align: center;
5  }
```

The code in the example above demonstrates just some of the various table aspects we can style using CSS properties.

3.13 Review

We learned how to create a table, add data to it, and section the table into smaller parts that make it easier to read.

Let is review what this chapter covered:

- The `<table>` element creates a table.

- The `<tr>` element adds rows to a table.

- To add data to a row, you can use the `<td>` element.

- Table headings clarify the meaning of data. Headings are added with the `<th>` element.

- Table data can span columns using the `colspan` attribute.

- Table data can span rows using the `rowspan` attribute.

- Tables can be split into three main sections: a head, a body, and a footer.

- A table's head is created with the `<thead>` element.

- A table's body is created with the `<tbody>` element.

- A table's footer is created with the `<tfoot>` element.

- All the CSS properties you learned about in this course can be applied to tables and their data.

4 HTML Forms

Now we will cover how to add a <form> element populated with various elements that accept input on a web page.

4.1 Introduction to HTML Forms

Forms are a part of everyday life. When we use a physical form in real life, we write down information and give it to someone to process. Think of the times you have had to fill out information for various applications like a job, or a bank account, or dropped off a completed suggestion card — each instance is a form.

Just like a physical form, an HTML <form> element is responsible for collecting information to send somewhere else. Every time we browse the internet we come into contact with many forms and we might not even realize it. There is a good chance that if we are typing into a text field or providing an input, the field that we are typing into is part of a <form>.

Now we will go over the structure and syntax of a <form> and the many elements that populate it.

4.2 How a Form Works

We can think of the internet as a network of computers which send and receive information. Computers need an *HTTP request* to know how to communicate. The HTTP request instructs the receiving computer how to handle the incoming information. More information can be found in the article about HTTP requests.

The <form> element is a great tool for collecting information, but then we need to send that information somewhere else for processing. We need to supply the <form> element with both the location of where the <form>'s information goes and what HTTP request to make. Take a look at the sample <form> below:

```
1  <form action="/example.html" method="POST">
2  </form>
```

In the above example, we have created the skeleton for a <form> that will send information to **example.html** as a POST request:

- The action attribute determines where the information is sent.

- The method attribute is assigned a HTTP verb that is included in the HTTP request.

Note: HTTP verbs like POST do not need to be capitalized for the request to work, but it is done so out of convention. In the example above we could have written method="post" and it would still work.

The <form> element can also contain child elements. For instance, it would be helpful to provide a header so that users know what this <form> is about. We could also add a paragraph to provide even more detail. Let us see an example of this in code:

```
1  <form action="/example.html" method="POST">
2    <h1>Creating a form</h1>
3    <p>Looks like you want to learn how to create an HTML form. Well, the best
         way to learn is to play around with it.</p>
4  </form>
```

4.3 Text Input

If we want to create an input field in our <form>, we will need the help of the <input> element.

The <input> element has a type attribute which determines how it renders on the web page and what kind of data it can accept.
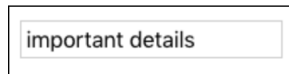
The first value for the type attribute we are going to explore is "text". When we create an <input> element with type="text", it renders a text field that users can type into. It is also important that we include a name attribute for the <input> — without the name attribute, information in the <input> will not be sent when the <form> is submitted. We will explain more about submissions and the submit button later. For now, let us examine the following code that produces a text input field:

```
1  <form action="/example.html" method="POST">
2    <input type="text" name="first-text-field">
3  </form>
```

Here is a screen shot of how the rendered form looks like on a web page for the Chrome browser (different browsers have different default rendering). When initially loaded, it will be an empty box:

After users type into the <input> element, the value of the value attribute becomes what is typed into the text field. The value of the value attribute is paired with the value of the name attribute and sent as text when the form is submitted. For instance, if a user typed in "important details" in the text field created by our <input> element:

When the form is submitted, the text: "first-text-field=important details" is sent to /example.html because the value of the name attribute is "first-text-field" and the value of value is "important details".

We could also assign a default value for the value attribute so that users have a pre-filled text field when they first see the rendered form like so:

```
1  <form action="/example.html" method="POST">
2    <input type="text" name="first-text-field" value="already pre-filled">
3  </form>
```
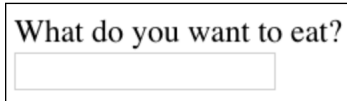
This renders:

4.4 Adding a Label

Previously we created an <input> element but we did not include anything to explain what the <input> is used for. For a user to properly identify an <input> we use the appropriately named <label> element.

The <label> element has an opening and closing tag and displays text that is written between the opening and closing tags. To associate a <label> and an <input>, the <input> needs an id attribute. We then assign the for attribute of the <label> element with the value of the id attribute of <input>, like so:

```
1  <form action="/example.html" method="POST">
2    <label for="meal">What do you want to eat?</label>
3    <br>
4    <input type="text" name="food" id="meal">
5  </form>
```

The code above renders:

What do you want to eat?

Now users know what the <input> element is for. Another benefit for using the <label> element is when this element is clicked, the corresponding <input> is highlighted/selected.

4.5   Password Input

Think about all those times we have to put sensitive information, like a password or PIN, into a <form>. We would not want our information to be seen by anyone peeking over our shoulder. Luckily, we have the type="password" attribute for <input>.

An <input type ="password"> element will replace input text with another character like an asterisk (*) or a dot (•). The code below provides an example of how to create a password field:

```
1  <form>
2    <label for="user-password">Password: </label>
3    <input type="password" id="user-password" name="user-password">
4  </form>
```

After a user types into the field, it would look like:

Password: •••••••

Even though the password field obscures the text of the password, when the form is submitted, the value of the text is sent. In other words, if "hunter2" is typed into the password field, "user-password=hunter2" is sent along with the other information on the form.

4.6   Number Input

We have now gone over two type attributes for <input> related to text. But, we might want our users to type in a number — in which case we can set the type attribute to "number".

By setting type="number" for an <input> we can restrict what users type into the input field to just numbers (and a few special characters like −, +, and .). We can also provide a step attribute which creates arrows inside the input field to increase or decrease by the value of the step attribute. Below is the code needed to render an input field for numbers:

```
1  <form>
2    <label for="years"> Years of experience: </label>
3    <input id="years" name="years" type="number" step="1">
4  </form>
```

The code above renders:

Years of experience: [⌄]

4.7 Range Input

Using an <input type="number"> is great if we want to allow users to type in any number of their choosing. But, if we wanted to limit what numbers our users could type we might consider using a different type value. Another option we could use is setting type to "range" which creates a slider.

To set the minimum and maximum values of the slider we assign values to the min and max attribute of the <input>. We could also control how smooth and fluid the slider works by assigning the step attribute a value. Smaller step values will make the slider more fluidly, whereas larger step values will make the slider move more noticeably. Take a look at the code to create a slider:

```
1  <form>
2    <label for="volume"> Volume Control</label>
3    <input id="volume" name="volume" type="range" min="0" max="100" step="1">
4  </form>
```

The code above renders:

Volume Control ═══════⬤════════

In the example above, every time the slider moves by one, the value of the <input>'s value attribute changes.

4.8 Checkbox Input

So far the types of inputs we have allowed were all single choices. But, what if we presented multiple options to users and allow them to select any number of options? Sounds like we could use checkboxes. In a <form> we would use the <input> element and set type="checkbox". Examine the code used to create multiple checkboxes:

```
1  <form>
2    <p>Choose your pizza toppings:</p>
3    <label for="cheese">Extra cheese</label>
4    <input id="cheese" name="topping" type="checkbox" value="cheese">
5    <br>
6    <label for="pepperoni">Pepperoni</label>
7    <input id="pepperoni" name="topping" type="checkbox" value="pepperoni">
8    <br>
9    <label for="anchovy">Anchovy</label>
10   <input id="anchovy" name="topping" type="checkbox" value="anchovy">
11 </form>
```

The code above renders:

> Choose your pizza toppings:
>
> Extra cheese ☐
> Pepperoni ☐
> Anchovy ☐

Notice in the example provided:

- there are assigned values to the value attribute of the checkboxes. These values are not visible on the form itself, that is why it is important that we use an associated <label> to identify the checkbox.

- each <input> has the same value for the name attribute. Using the same name for each checkbox groups the <input>s together. However, each <input> has a unique id to pair with a <label>.

4.9 Radio Button Input

Checkboxes work well if we want to present users with multiple options and let them choose one or more of the options. However, there are cases where we want to present multiple options and only allow for one selection — like asking users if they agree or disagree with the terms and conditions. Let us look over the code used to create radio buttons:

```
1  <form>
2    <p>What is sum of 1 + 1?</p>
3    <input type="radio" id="two" name="answer" value="2">
4    <label for="two">2</label>
5    <br>
6    <input type="radio" id="eleven" name="answer" value="11">
7    <label for="eleven">11</label>
8  </form>
```

The code above renders:

> What is sum of 1 + 1?
>
> ○ 2
> ○ 11

Notice from the code snippet, radio buttons (like checkboxes) do not display their value. We have an associated <label> to represent the value of the radio button. To group radio buttons together, we assign them the same name and only one radio button from that group can be selected.
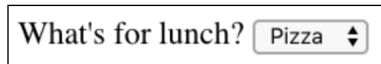
4.10 Dropdown list

Radio buttons are great if we want our users to pick one option out of a few visible options, but imagine if we have a whole list of options. This situation could quickly lead to a lot of radio buttons to keep track of.
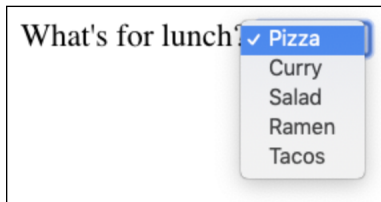
An alternative solution is to use a dropdown list to allow our users to choose one option from an organized list. Here is the code to create a dropdown menu:

```
1  <form>
2    <label for="lunch">What's for lunch?</label>
3    <select id="lunch" name="lunch">
4      <option value="pizza">Pizza</option>
5      <option value="curry">Curry</option>
6      <option value="salad">Salad</option>
7      <option value="ramen">Ramen</option>
8      <option value="tacos">Tacos</option>
9    </select>
10  </form>
```

The code above renders:

And if we click on the field containing the first option, the list is revealed:

Notice in the code that we are using the element `<select>` to create the dropdown list. To populate the dropdown list, we add multiple `<option>` elements, each with a `value` attribute. By default, only one of these options can be selected.

The text rendered is the text included between the opening and closing `<option>` tags. However, it is the `value` of the `value` attribute that is used in `<form>` submission (notice the difference in the text and `value` capitalization). When the `<form>` is submitted, the information from this input field will be sent using the `name` of the `<select>` and the `value` of the chosen `<option>`. For instance, if a user selected Pizza from the dropdown list, the information would be sent as `"lunch=pizza"`.

4.11 Datalist Input

Even if we have an organized dropdown list, if the list has a lot of options, it could be tedious for users to scroll through the entire list to locate one option. That is where using the `<datalist>` element comes in handy.

The `<datalist>` is used with an `<input type="text">` element. The `<input>` creates a text field that users can type into and filter options from the `<datalist>`. Let us go over a concrete example:

```
1  <form>
2    <label for="city">Ideal city to visit?</label>
3    <input type="text" list="cities" id="city" name="city">
4
5    <datalist id="cities">
6      <option value="New York City"></option>
```
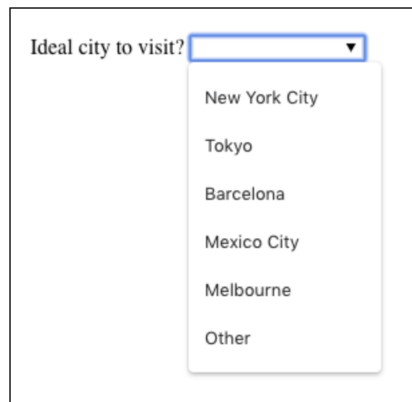
```
7        <option value="Tokyo"></option>
8        <option value="Barcelona"></option>
9        <option value="Mexico City"></option>
10       <option value="Melbourne"></option>
11       <option value="Other"></option>
12    </datalist>
13  </form>
```

Notice, in the code above, we have an <input> that has a list attribute. The <input> is associated to the <datalist> via the <input>'s list attribute and the id of the <datalist>.

From the code provided, the following form is rendered:

Ideal city to visit? [   ]

And when field is selected:

Ideal city to visit? [▼]

New York City

Tokyo

Barcelona

Mexico City

Melbourne

Other

While <select> and <datalist> share some similarities, there are some major differences. In the associated <input> element, users can type in the input field to search for a particular option. If none of the <option>s match, the user can still use what they typed in. When the form is submitted, the value of the <input>'s name and the value of the option selected, or what the user typed in, is sent as a pair.

4.12   Textarea element

An <input> element with type="text" creates a single row input field for users to type in information. However, there are cases where users need to write in more information, like a blog post. In such cases, instead of using an <input>, we could use <textarea>.

The <textarea> element is used to create a bigger text field for users to write more text. We can add the attributes rows and cols to determine the amount of rows and columns for the <textarea>. Take a look:

```
1  <form>
2    <label for="blog">New Blog Post: </label>
3    <br>
4    <textarea id="blog" name="blog" rows="5" cols="30">
5    </textarea>
6  </form>
```

In the code above, an empty <textarea> that is 5 rows by 30 columns is rendered to the page like so:

New Blog Post:

If we wanted an even bigger text field, we could click and drag on the bottom right corner to expand it.

When we submit the form, the value of <textarea> is the text written inside the box. If we want to add a default value to text to <textarea> we would include it within the opening and closing tags like so:

```
1   <textarea>Adding default text</textarea>
```

This code will render a <textarea> that contains pre-filled text: "Adding default text".

4.13   Submit Form

Remember, the purpose of a form is to collect information that will be submitted. That is the role of the submit button — users click on it when they are finished with filling out information in the <form> and they are ready to send it off. Now that we have gone over how to create various input elements, let us now go over how to create a submit button.

To make a submit button in a <form>, we are going to use the reliable <input> element and set the type to "submit". For instance:

```
1   <form>
2     <input type="submit" value="Send">
3   </form>
```

The code above renders:

Send

Notice in the code snippet that the value assigned to the <input> shows up as text on the submit button. If there is not a value attribute, the default text, Submit shows up on the button.

4.14   Review

Let us review what we learned about the <form> element:

- The purpose of a <form> is to allow users to input information and send it.

- The <form>'s action attribute determines where the form's information goes.

- The <form>'s method attribute determines how the information is sent and processed.

- To add fields for users to input information we use the <input> element and set the type attribute to a field of our choosing:

- Setting type to "text" creates a single row field for text input.

- Setting type to "password" creates a single row field that censors text input.

- Setting type to "number" creates a single row field for number input.

- Setting type to "range" creates a slider to select from a range of numbers.

- Setting type to "checkbox" creates a single checkbox which can be paired with other checkboxes.

- Setting type to "radio" creates a radio button that can be paired with other radio buttons.

- Setting type to "list" will pair the <input> with a <datalist> element if the id of both are the same.

- Setting type to "submit" creates a submit button.

- A <select> element is populated with <option> elements and renders a dropdown list selection.

- A <datalist> element is populated with <option> elements and works with an <input> to search through choices.

- A <textarea> element is a text input field that has a customizable area.

- When a <form> is submitted, the name of the fields that accept input and the value of those fields are sent as name=value pairs.

Using the <form> element in conjunction with the other elements listed above allows us to create sites that take into consideration the wants and needs of our users.

5 Form Validation

Now we learn how to add attributes to inputs and validate the information a user provides in a form.

5.1 Introduction to HTML Form Validation

Ever wonder how a login page actually works? Or why the combination of a username and password grants you access to a website? The answers lie in *validation*. Validation is the concept of checking user provided data against the required data.

There are different types of validation. One type is *server-side validation*, this happens when data is sent to another machine (typically a server) for validation. An example of this type of validation is the usage of a login page. The form on the login page accepts username and password input, then sends the data to a server that checks that the pair matches up correctly.

On the other hand, we use *client-side validation* if we want to check the data on the browser (the client). This validation occurs before data is sent to the server. Different browsers implement client-side validation differently, but it leads to the same outcome.

Shared among the different browsers are the benefits of using HTML5's built-in client-side validation. It saves us time from having to send information to the server and wait for the server to send back confirmation or rejection of the data. This can also help us protect our server from malicious code or data from a malicious user. It also allows us to quickly give feedback to users for specific fields rather than having them fill in a form again if the data they input into the form was rejected.
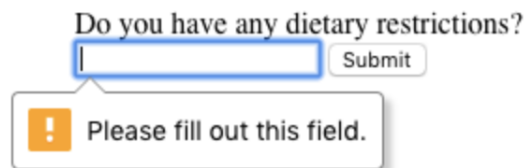
5.2 Requiring an Input

Sometimes we have fields in our `<form>`s which are not optional, i.e. there must be information provided before we can submit it. To enforce this rule, we can add the `required` attribute to an `<input>` element.

Take for example:

```
1  <form action="/example.html" method="POST">
2    <label for="allergies">Do you have any dietary restrictions?</label>
3    <br>
4    <input id="allergies" name="allergies" type="text" required>
5    <br>
6    <input type="submit" value="Submit">
7  </form>
```

This renders a text box, and if we try to submit the `<form>` without filling it out we get this message:
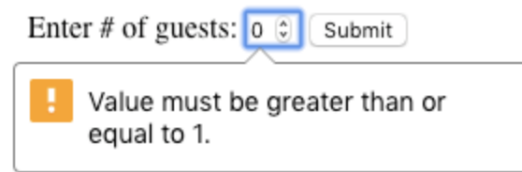

The styling of the message varies from browser to browser, the picture above depicts the message in a Chrome browser. We will also continue to show these messages as they appear in Chrome in later exercises.

5.3 Set a Minimum and Maximum

Another built-in validation we can use is to assign a minimum or maximum value for a number field, e.g. `<input type="number">` and `<input type="range">`. To set a minimum acceptable value, we use the `min` attribute and assign a value. On the flip side, to set a maximum acceptable value, we assign the `max` attribute a value. Let's see this in code:

```
1  <form action="/example.html" method="POST">
2    <label for="guests">Enter # of guests:</label>
3    <input id="guests" name="guests" type="number" min="1" max="4">
4    <input type="submit" value="Submit">
5  </form>
```

If a user tries to submit an input that is less than 1 a warning will appear:

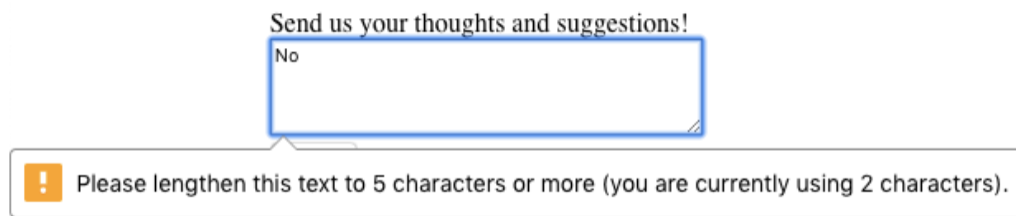A similar message will appear if a user tries to input a number greater than 4.

5.4 Checking Text Length

Previously, we were able to use `min` and `max` to set acceptable minimum and maximum values in a number field. Now let us consider about text fields. There are certainly cases where we would not want our users typing more than a certain number of characters (think about the character cap for messages on Twitter). We might even want to set a minimum number of characters. Conveniently, there are built-in HTML5 validations for these situations.

To set a minimum number of characters for a text field, we add the `minlength` attribute and a value to set a minimum value. Similarly, to set the maximum number of characters for a text field, we use the `maxlength` attribute and set a maximum value. Let us take a look at these attributes in code:

```
1  <form action="/example.html" method="POST">
2    <label for="summary">Summarize your feelings in less than 250 characters
         </label>
3    <input id="summary" name="summary" type="text" minlength="5" maxlength="
         250" required>
4    <input type="submit" value="Submit">
5  </form>
```

If a user tries to submit the `<form>` with less than the set minimum, this message appears:

And if a user tries to type in more than the maximum allowed number of characters, they do not get a warning message, but they cannot type it in.