WEB Development Learning Path Notes

HTML

## Id attributes

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’ll 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:

<div id="intro">

<h1>Introduction</h1>

</div>

## Paragraphs & Spans

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:

<div>

<h1>Technology</h1>

</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>. 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.

It’s 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’s better to use a <div>.

## em and strong tags

The <em> tag emphasizes text, while the <strong> tag highlights important text.

Later, when you begin to style websites, you 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:

<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).

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

## Line Breaks

. 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.

<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).

## 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 won’t 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.

The output will look like this:

 Limes

 Tortillas

 Chicken

## Ordered List

*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.

<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>

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.

## 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. 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.

<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.

Part of being an exceptional web developer is making your 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.

<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.

## Video

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 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.

# HTML Standards

HTML files require certain elements to set up the document properly. 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, 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 <!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.

## Head Eliment

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>

### Title element

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 Catalog | 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).

## Links

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.

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 internal pages

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 using pics etc

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 A different part of the 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!

## White Space

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

## Tables

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

<table>

</table>

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

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](https://en.wikipedia.org/wiki/Table_(information)). 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>.

<table>

<tr>

</tr>

<tr>

</tr>

</table>

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

Rows aren’t 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>.

<table>

<tr>

<td>73</td>

<td>81</td>

</tr>

</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.

Table data doesn’t 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.

<table>

<tr>

<th></th>

<th scope="col">Saturday</th>

<th scope="col">Sunday</th>

</tr>

<tr>

<th scope="row">Temperature</th>

<td>73</td>

<td>81</td>

</tr>

</table>

What happened in the code above?

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.

### Spanning columns

What if 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.

<table>

<tr>

<th>Monday</th>

<th>Tuesday</th>

<th>Wednesday</th>

</tr>

<tr>

<td colspan="2">Out of Town</td>

<td>Back in Town</td>

</tr>

</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.

### 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.

<table>

<tr> <!-- Row 1 -->

<th></th>

<th>Saturday</th>

<th>Sunday</th>

</tr>

<tr> <!-- Row 2 -->

<th>Morning</th>

<td rowspan="2">Work</td>

<td rowspan="3">Relax</td>

</tr>

<tr> <!-- Row 3 -->

<th>Afternoon</th>

</tr>

<tr> <!-- Row 4 -->

<th>Evening</th>

<td>Dinner</td>

</tr>

</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.

If you’d like to see how the browser interprets the code above, feel free to copy and paste it into the code editor to understand it a little better.

### Limiting tables with table body , table head and table foot

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 (more on this in a later exercise).

<table>

<tbody>

<tr>

<th></th>

<th>Saturday</th>

<th>Sunday</th>

</tr>

<tr>

<th>Morning</th>

<td rowspan="2">Work</td>

<td rowspan="3">Relax</td>

</tr>

<tr>

<th>Afternoon</th>

</tr>

<tr>

<th>Evening</th>

<td>Dinner</td>

</tr>

</tbody>

</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 — we’ll change this in the next exercise.

In the last exercise, 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.

<table>

<thead>

<tr>

<th></th>

<th scope="col">Saturday</th>

<th scope="col">Sunday</th>

</tr>

</thead>

<tbody>

<tr>

<th scope="row">Morning</th>

<td rowspan="2">Work</td>

<td rowspan="3">Relax</td>

</tr>

<tr>

<th scope="row">Afternoon</th>

</tr>

<tr>

<th scope="row">Evening</th>

<td>Dinner</td>

</tr>

</tbody>

</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.

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

<table>

<thead>

<tr>

<th>Quarter</th>

<th>Revenue</th>

<th>Costs</th>

</tr>

</thead>

<tbody>

<tr>

<th>Q1</th>

<td>$10M</td>

<td>$7.5M</td>

</tr>

<tr>

<th>Q2</th>

<td>$12M</td>

<td>$5M</td>

</tr>

</tbody>

<tfoot>

<tr>

<th>Total</th>

<td>$22M</td>

<td>$12.5M</td>

</tr>

</tfoot>

</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.

## Forms

How Forms Work

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:

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

</form>

In the above example, we’ve 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 and 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’s 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’s see an example of this in code:

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

<h1>Creating a form</h1>

<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>

</form>

The example above doesn’t collect any user input, but we’ll do that in the next exercise. For now, let’s practice making the foundation of an HTML <form>!

### Text Input

If we want to create an input field in our <form>, we’ll 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’re 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’s also important that we include a name attribute for the <input> — without the name attribute, information in the <input> won’t be sent when the <form> is submitted. We’ll explain more about submissions and the submit button in a later exercise. For now, let’s examine the following code that produces a text input field:

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

<input type="text" name="first-text-field">

</form>

Here’s 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:

rendered empty text field from input element type='text'

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:

rendered filled text field which reads 'important details' 

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:

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

<input type="text" name="first-text-field" value="already pre-filled">

</form>

Which renders:

pre-filled text box due to assigned `value` attribute

In the previous exercise we created an <input> element but we didn’t 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:

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

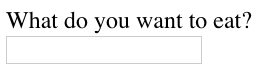
<label for="meal">What do you want to eat?</label>

<br>

<input type="text" name="food" id="meal">

</form>

The code above renders:



Look, 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.

Let’s see the <label> element in action!

### Password Input

Think about all those times we have to put sensitive information, like a password or PIN, into a <form>. We wouldn’t 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:

<form>

<label for="user-password">Password: </label>

<input type="password" id="user-password" name="user-password">

</form>

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

password field in a form with 6 dots showing text added to the field

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.

### Numeric input

We’ve 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… (you guessed it)… "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:

<form>

<label for="years"> Years of experience: </label>

<input id="years" name="years" type="number" step="1">

</form>

Which renders:

rendered number input field with arrows to the right hand side of the field

### 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:

<form>

<label for="volume"> Volume Control</label>

<input id="volume" name="volume" type="range" min="0" max="100" step="1">

</form>

The code above renders:rendered slider for volume control

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

### Checkbox Inputs

So far the types of inputs we’ve 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:

<form>

<p>Choose your pizza toppings:</p>

<label for="cheese">Extra cheese</label>

<input id="cheese" name="topping" type="checkbox" value="cheese">

<br>

<label for="pepperoni">Pepperoni</label>

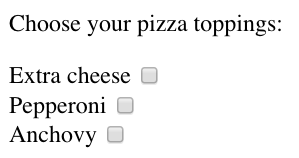
<input id="pepperoni" name="topping" type="checkbox" value="pepperoni">

<br>

<label for="anchovy">Anchovy</label>

<input id="anchovy" name="topping" type="checkbox" value="anchovy">

</form>

Which renders:

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’s 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>.

### Radio Buttons

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’s look over the code used to create radio buttons:

<form>

<p>What is sum of 1 + 1?</p>

<input type="radio" id="two" name="answer" value="2">

<label for="two">2</label>

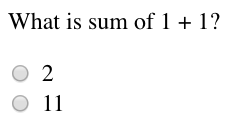
<br>

<input type="radio" id="eleven" name="answer" value="11">

<label for="eleven">11</label>

</form>

Which renders:



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.

### Drop Down 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’s the code to create a dropdown menu:

<form>

<label for="lunch">What's for lunch?</label>

<select id="lunch" name="lunch">

<option value="pizza">Pizza</option>

<option value="curry">Curry</option>

<option value="salad">Salad</option>

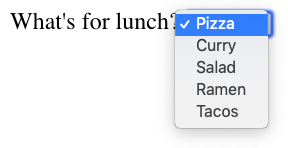
<option value="ramen">Ramen</option>

<option value="tacos">Tacos</option>

</select>

</form>

Which renders:rendered dropdown list with the first option showing

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

Notice in the code that we’re 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".

### 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’s 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’s go over a concrete example:

<form>

<label for="city">Ideal city to visit?</label>

<input type="text" list="cities" id="city" name="city">

<datalist id="cities">

<option value="New York City"></option>

<option value="Tokyo"></option>

<option value="Barcelona"></option>

<option value="Mexico City"></option>

<option value="Melbourne"></option>

<option value="Other"></option>

</datalist>

</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:input field with a label 'Ideal city to visit?'

And when field is selected:clicking on the input field reveals a dropdown 
list

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.

### Text Area

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:

<form>

<label for="blog">New Blog Post: </label>

<br>

<textarea id="blog" name="blog" rows="5" cols="30">

</textarea>

</form>

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



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:

<textarea>Adding default text</textarea>

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

### Submit Button

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

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

<form>

<input type="submit" value="Send">

</form>

Which renders:

rendered submit button

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

## Form Validation

### Compulsory 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:

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

<label for="allergies">Do you have any dietary restrictions?</label>

<br>

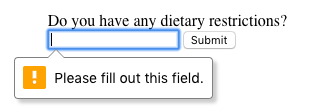
<input id="allergies" name="allergies" type="text" required>

<br>

<input type="submit" value="Submit">

</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’ll also continue to show these messages as they appear in Chrome in later exercises.

### Min & Max Numbers

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:

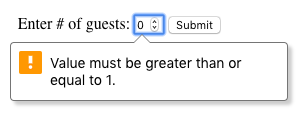
<form action="/example.html" method="POST">

<label for="guests">Enter # of guests:</label>

<input id="guests" name="guests" type="number" min="1" max="4">

<input type="submit" value="Submit">

</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. Let’s now see this action.

### Check text length

In the previous exercise, we were able to use min and max to set acceptable minimum and maximum values in a number field. But what about text fields? There are certainly cases where we wouldn’t want our users typing more than a certain number of characters (think about the message cap for 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’s take a look at these attributes in code:

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

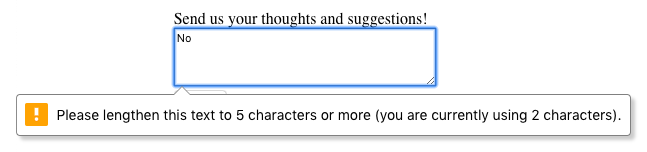
<label for="summary">Summarize your fillings in less than 250 characters</label>

<input id="summary" name="summary" type="text" minlength="5" maxlength="250" required>

<input type="submit" value="Submit">

</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 don’t get a warning message, but they can’t type it in!

### Pattern Matching (regular expressions)

In addition to checking the length of a text, we could also add a validation to check how the text was provided. For cases when we want user input to follow specific guidelines, we use the pattern attribute and assign it a *regular expression*, or regex. Regular expressions are a sequence of characters that make up a search pattern. If the input matches the regex, the form can be submitted.

Let’s say we wanted to check for a valid credit card number (a 14 to 16 digit number). We could use the regex: [0-9]{14,16} which checks that the user provided only numbers and that they entered at least 14 digits and at most 16 digits. To add this to a form:

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

<label for="payment">Credit Card Number (no spaces):</label>

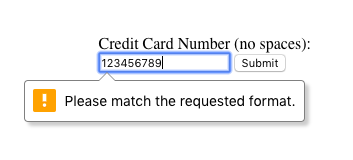
<br>

<input id="payment" name="payment" type="text" required pattern="[0-9]{14,16}">

<input type="submit" value="Submit">

</form>

With the pattern in place, users can’t submit the <form> with a number that doesn’t follow the regex. When they try, they’ll see a validation message like so:



If you want to find out more about Regex, read more at [MDN’s regex article](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions).

## Semantic HTML

When building web pages, we use a combination of non-semantic HTML and *Semantic HTML*. The word semantic means “relating to meaning,” so semantic elements provide information about the content between the opening and closing tags.

By using Semantic HTML, we select HTML elements based on their meaning, not on how they are presented. Elements such as <div> and <span> are not semantic elements since they provide no context as to what is inside of those tags.

For example, instead of using a <div> element to contain our header information, we could use a <header> element, which is used as a heading section. By using a <header> tag instead of a <div>, we provide context as to what information is inside of the opening and closing tag.

**Why use Semantic HTML?**

* **Accessibility:** Semantic HTML makes webpages accessible for mobile devices and for people with disabilities as well. This is because screen readers and browsers are able to interpret the code better.
* **SEO:** It improves the website SEO, or *Search Engine Optimization*, which is the process of increasing the number of people that visit your webpage. With better SEO, search engines are better able to identify the content of your website and weight the most important content appropriately.
* **Easy to Understand:** Semantic HTML also makes the website’s source code easier to read for other web developers.

To better understand this, you can think of comparing non-semantic HTML to going into a store with no signs on the aisles. Since the aisles aren’t labeled, you don’t know what products are in those aisles. However, stores that do have signs for each aisle make it a lot easier to find the items you need, just like Semantic HTML.

### Header & Nav

Let’s take a look at some semantic elements that assist in the structure of a web page. A <header> is a container usually for either navigational links or introductory content containing <h1> to <h6> headings.

The example below shows <header> in action:

<header>

<h1>

Everything you need to know about pizza!

</h1>

</header>

This can be compared to the code below which uses a <div> tag instead of a <header> tag:

<div id="header">

<h1>

Everything you need to know about pizza!

</h1>

</div>

By using a <header> tag, our code becomes easier to read. It is much easier to identify what is inside of the <h1>‘s parent tags, as opposed to a <div> tag which would provide no details as to what was inside of the tag.

A <nav> is used to define a block of navigation links such as menus and tables of contents. It is important to note that <nav> can be used inside of the <header> element but can also be used on its own.

Let’s take a look at the example below:

<header>

<nav>

<ul>

<li><a href="#home">Home</a></li>

<li><a href="#about">About</a></li>

</ul>

</nav>

</header>

By using <nav> as a way to label our navigation links, it will be easier for not only us, but also for web browsers and screen readers to read the code.

### Main & Footer

Two more structural elements are <main> and <footer>. These elements along with <nav> and <header> help describe where an element is located based on conventional web development standards.

The element <main> is used to encapsulate the dominant content within a webpage. This tag is separate from the <footer> and the <nav> of a web page since these elements don’t contain the principal content. By using <main> as opposed to a <div> element, screen readers and web browsers are better able to identify that whatever is inside of the tag is the bulk of the content.

So how does <main> look when incorporated into our code? That’s a great question.

<main>

<header>

<h1>Types of Sports<h1>

</header>

<article>

<h3>Baseball</h3>

<p>

The first game of baseball was played in Cooperstown, New York in the summer of 1839.

</p>

</article>

</main>

As we see above, <main> contains an <article> and <header> tag with child elements that hold the most important information related to the page.

The content at the bottom of the subject information is known as the *footer*, indicated by the <footer> element. The footer contains information such as:

* Contact information
* Copyright information
* Terms of use
* Site Map
* Reference to top of page links

For example:

<footer>

<p>Email me at Codey@Codecademy.com</p>

</footer>

In the example above, the footer is used to contain contact information. The <footer> tag is separate from the <main> element and typically located at the bottom of the content.

### Section & Article

Now that we covered the body of Semantic HTML, let’s focus on what can go in the body. The two elements we’re going to focus on now are <section> and <article>.

<section> defines elements in a document, such as chapters, headings, or any other area of the document with the same theme. For example, content with the same theme such as articles about cricket can go under a single <section>. A website’s home page could be split into sections for the introduction, news items, and contact information.

Here is an example of how to use <section>:

<section>

<h2>Fun Facts About Cricket</h2>

</section>

In the code above we created a <section> element to encapsulate the code. In <section> we added a <h2> element as a heading.

The <article> element holds content that makes sense on its own. <article> can hold content such as articles, blogs, comments, magazines, etc. An <article> tag would help someone using a screen reader understand where the article content (that might contain a combination of text, images, audio, etc.) begins and ends.

Here is an example of how to use <article>:

<section>

<h2>Fun Facts About Cricket</h2>

<article>

<p>A single match of cricket can last up to 15 days.</p>

</article>

</section>

In the code above, the <article> element containing a fact about cricket was placed inside of the <section> element.It is important to note that a <section> element could also be placed in an <article> element depending on the context.

### Aside

The <aside> element is used to mark additional information that can enhance another element but isn’t required in order to understand the main content. This element can be used alongside other elements such as <article> or <section>. Some common uses of the <aside> element are for:

* Bibliographies
* Endnotes
* Comments
* [Pull quotes](https://en.wikipedia.org/wiki/Pull_quote)
* Editorial sidebars
* Additional information

Here’s an example of <aside> being used alongside <article>:

<article>

<p>The first World Series was played between Pittsburgh and Boston in 1903 and was a nine-game series.</p>

</article>

<aside>

<p>

Babe Ruth once stated, “Heroes get remembered, but legends never die.”

</p>

</aside>

As shown above, the information within the <article> is the important content. Meanwhile the information within the <aside> enhances the information in <article> but is not required in order to understand it.

### Figure & Figcaption

With <aside>, we learned that we can put additional information next to a main piece of content, but what if we wanted to add an image or illustration? That is where <figure> and <figcaption> come in.

<figure> is an element used to encapsulate media such as an image, illustration, diagram, code snippet, etc, which is referenced in the main flow of the document.

<figure>

<img src="overwatch.jpg"/>

</figure>

In this code, we created a <figure> element so that we can encapsulate our <img> tag. In <figure> we used the <img> tag to insert an image onto the webpage. We used the src attribute within the <img> tag so that we can link the source of the image.

It’s possible to add a caption to the image by using <figcaption>.

<figcaption> is an element used to describe the media in the <figure> tag. Usually, <figcaption> will go inside <figure>. This is different than using a <p> element to describe the content; if we decide to change the location of <figure>, the paragraph tag may get displaced from the figure while a <figcaption> will move with the figure. This is useful for grouping an image with a caption.

<figure>

<img src="overwatch.jpg">

<figcaption>This picture shows characters from Overwatch.</figcaption>

</figure>

In the example above, we added a <figcaption> into the <figure> element to describe the image from the previous example. This helps group the <figure> content with the <figcaption> content.

While the content in <figure> is related to the main flow of the document, its position is independent. This means that you can remove it or move it somewhere else without affecting the flow of the document.

### Audio & Attributes

Now that we learned about text-based content, let us dig into <audio>! Surely everyone needs <audio>— how else would you listen to your Korean hip hop?

The <audio> element is used to embed audio content into a document. Like <video>, <audio> uses src to link the audio source.

<audio>

<source src="iAmAnAudioFile.mp3" type="audio/mp3">

</audio>

In this example, we created an <audio> element. Then we created a <source> element to encapsulate our audio link. In this case, iAmAnAudioFile.mp3 is our audio file. Then we specified the type by using type and named what kind of audio it is. Although not always necessary, it’s recommended that we state the type of audio as it helps the browser identify it more easily and determine if that type of audio file is supported by the browser.

We linked our audio file into the browser but now we need to give it controls. This is where *attributes* come in. Attributes provide additional information about an element.

Attributes allow us to do many different things to our audio file. There are many attributes for <audio> but today we’re going to be focusing on controls and src.

* controls: automatically displays the audio controls into the browser such as play and mute.
* src: specifies the URL of the audio file.

As you might have noticed, we already used the src attribute. Most attributes go in the opening tag of <audio>. For example, here’s how we could add both autoplay functionality and audio controls:

<audio autoplay controls>

You can find other attributes here: [Useful attributes](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio#Attributes).

### Video & Embed

As demonstrated in the previous exercise, media content can be a useful addition to a website. By using a <video> element, we can add videos to our website. The <video> element makes it clear that a developer is attempting to display a video to the user.

Some attributes that can alter a video playback include:

* controls: When added in, a play/pause button will be added onto the video along with volume control and a fullscreen option.
* autoplay: The attribute which results in a video automatically playing as soon as the page is loaded.
* loop: This attribute results in the video continuously playing on repeat.

Below is an example of <video> being used with the controls attribute:

<video src="coding.mp4" controls>Video not supported</video>

In the code above, a video file named coding.mp4 is being played. The “Video not supported” will only show up if the browser is unable to display the video.

Another tag that can be used to incorporate media content into a page is the <embed> tag, which can embed any media content including videos, audio files, and gifs from an external source. This means that websites that have an embed button have some form of media content that can be added to other websites. The <embed> tag is a self-closing tag, unlike the <video> element.

Below we’ll take a look at <embed> being used in action.

<embed src="download.gif"/>

In the example above, <embed> is being used to add in a gif from a local file known as download.gif. Embed can be used to add local files as well as media content straight from some other websites.