**HTML-CSS**

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>

**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.

<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.

# 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’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>

# 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:

<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>.

# Styling Text

You can also style text using HTML 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

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

The line break element is 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 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.

<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.

# Image Alts

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.

# Videos

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

<video src="myVideo.mp4" width="320" height="240" controls>

Video not supported

</video>

In this example, the video source (src) is myVideo.mp4 The source 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.

[**https://youtu.be/uxmB8MlO3m8**](https://youtu.be/uxmB8MlO3m8)

**Documentation**

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

The declaration looks like this:

<!DOCTYPE html>

This declaration is an instruction, and it must be the first line of code in your HTML document. It tells the browser what type of document to expect, along with what version of HTML is being used in the document. For now, the browser will correctly assume that the html in <!DOCTYPE html> is referring to HTML5, as it is the current standard.

In the future, however, a new standard will override HTML5. To make sure your document is forever interpreted correctly, always include <!DOCTYPE html> at the very beginning of your HTML documents.

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

# The <html> tag

<!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.

* 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).

# Linking 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.

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.

# Opening Links in a New Window

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

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

In the example above, setting the target attribute to "\_blank" instructs the browser to open the relevant Wikipedia page in a new window.

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

# Linking to Relative Page

When making multi-page static websites, web developers often store HTML files in the *root directory*, or a main folder where all the files for the project are stored. As the size of the projects you create grows, you may use additional folders within the main project folder to organize your code.

project-folder/

|—— about.html

|—— contact.html

|—— index.html

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

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

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

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

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

# Linking At Will

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

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

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

# Linking to Same Page

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!

# Comments

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

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

**REVIEW**

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

# Table Headings

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>

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.

</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

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

# The rowspan attribute is used for data that spans multiple rows (perhaps an event goes on for multiple hours on a certain day).

# Table Body

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.

# Table Head

<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.

# Table Footer

# <tfoot> </tfoot>

# REVIEW

* 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.

# 14-05-2020

# How a Form Works

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.
* 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

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

# Adding a Label

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">

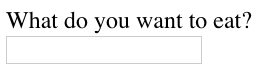
<label for="meal">What do you want to eat?</label>

<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.

# Password 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.

# Number Input

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

 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 Input

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>

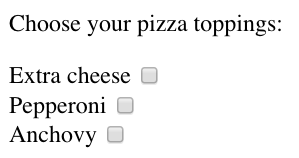
<input id="pepperoni" name="topping" type="checkbox" value="pepperoni">

<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 Button Input

<form>

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

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

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

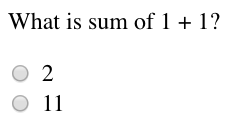
<br>

<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.

# Dropdown list

<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>

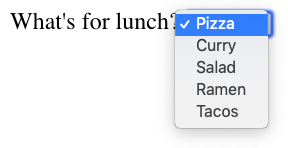
<option value="ramen">Ramen</option>

<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

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 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:

<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 Form

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.

**REVIEW**

In this lesson we went over:

* 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.

[**https://www.w3schools.com/html/html\_form\_input\_types.asp**](https://www.w3schools.com/html/html_form_input_types.asp)

Here are the different input types you can use in HTML:

* <input type="button">
* <input type="checkbox">
* <input type="color">
* <input type="date">
* <input type="datetime-local">
* <input type="email">
* <input type="file">
* <input type="hidden">
* <input type="image">
* <input type="month">
* <input type="number">
* <input type="password">
* <input type="radio">
* <input type="range">
* <input type="reset">
* <input type="search">
* <input type="submit">
* <input type="tel">
* <input type="text">
* <input type="time">
* <input type="url">
* <input type="week">

# Introduction to HTML Form Validation

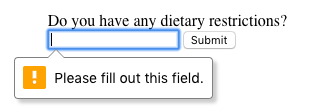
Ever wonder how a login page actually works? Or why the combination of a username and password grants you access to a website? The answers lie in validation. Validation is the concept of checking user provided data against the required data.

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.

f we try to submit the <form> without filling it out we get this message:



# 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:

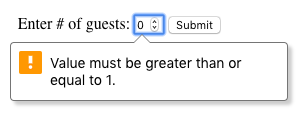
<form action="/example.html" method="POST">

<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.

# Checking 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 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’s take a look at these attributes in code:

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

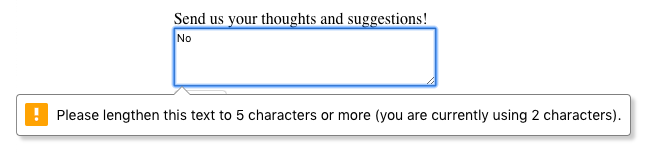
<label for="summary">Summarize your feelings in less than 250 characters</label>

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

<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!

# Matching a Pattern

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>

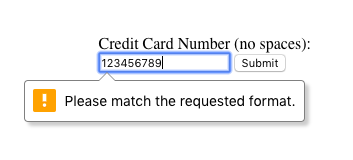
<br>

<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).

**REVIEW**

* Client-side validations happen in the browser before information is sent to a server.
* Adding the required attribute to an input related element will validate that the input field has information in it.
* Assigning a value to the min attribute of a number input element will validate an acceptable minimum value.
* Assigning a value to the max attribute of a number input element will validate an acceptable maximum value.
* Assigning a value to the minlength attribute of a text input element will validate an acceptable minimum number of characters.
* Assigning a value to the maxlength attribute of a text input element will validate an acceptable maximum number of characters.
* Assigning a regex to pattern matches the input to the provided regex.
* If validations on a <form> do not pass, the user gets a message explaining why and the <form> cannot be submitted.

# Introduction to 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.

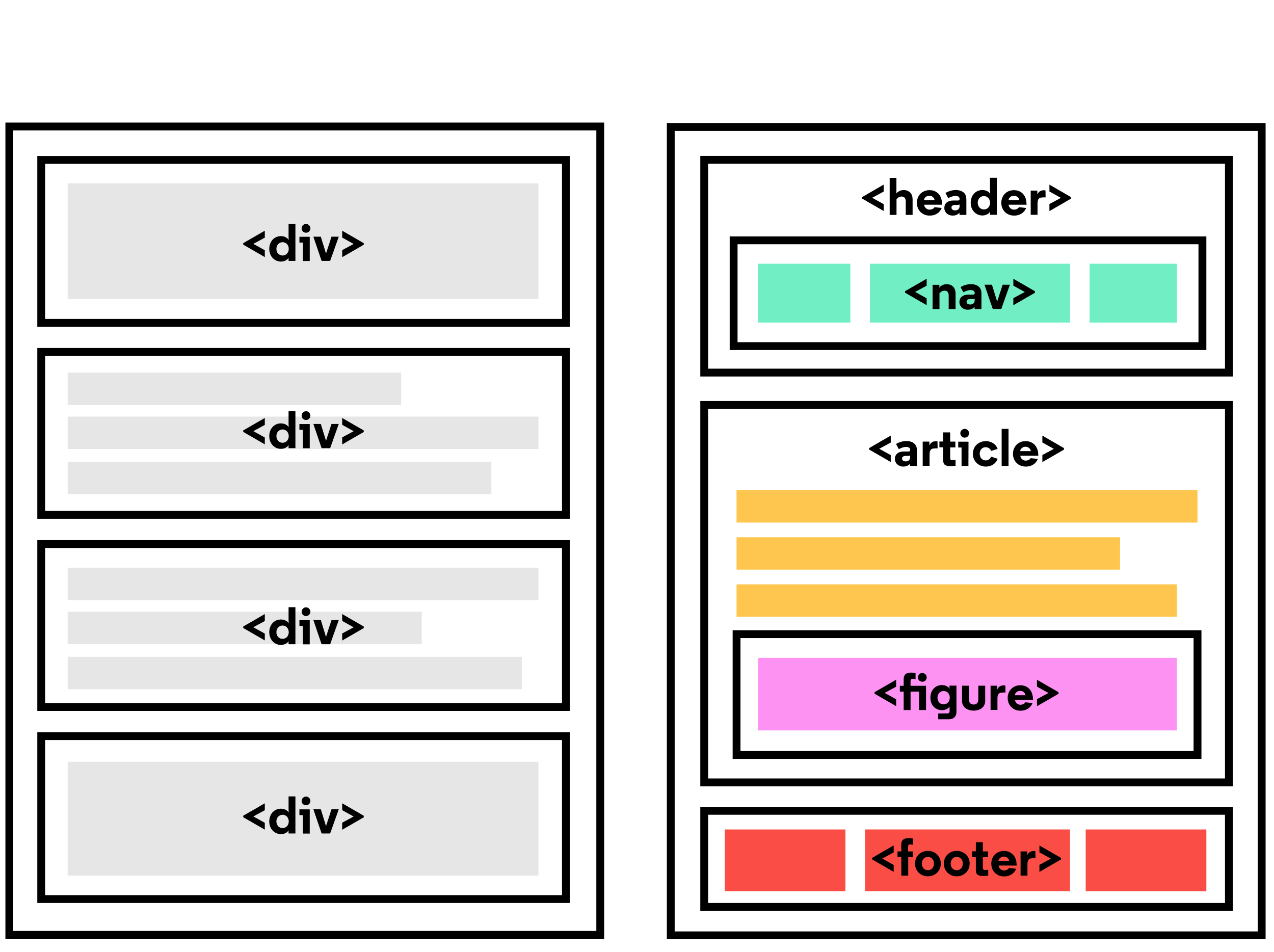
By using Semantic HTML, we select HTML elements based on their meaning, not on how they are presented. Elements such as <div> and <span> are not semantic elements since they provide no context as to what is inside of those tags.

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.

**NON SEMANTIC SEMANTIC**

# Header and 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 and 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.

# Article and Section

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 5 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.

# The Aside Element

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 and 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 and 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>

# Video and 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.

**REVIEW**

* Semantic HTML introduces meaning to a page through specific elements that provide context as to what is in between the tags.
* Semantic HTML is a modern standard and makes a website accessible for people who use screen readers to translate the webpage and improves your website’s SEO.
* <header>, <nav> , <main> and <footer> create the basic structure of the webpage.
* <section> defines elements in a document, such as chapters, headings, or any other area of the document with the same theme.
* <article> holds content that makes sense on its own such as articles, blogs, comments, etc.
* <aside> contains information that is related to the main content, but not required in order to understand the dominant information.
* <figure> encapsulates all types of media.
* <figcaption> is used to describe the media in <figure>.
* <video>, <embed>, and <audio> elements are used for media files.

**15-05-2020**

**CSS**

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

# Inline Styles

Although CSS is a different language than HTML, it’s possible to write CSS code directly within HTML code using inline styles.

To style an HTML element, you can add the style attribute directly to the opening tag. After you add the attribute, you can set it equal to the CSS style(s) you’d like applied to that element.

<p style="color: red;">I'm learning to code!</p>

The code in the example above demonstrates how to use inline styling. The paragraph element has a style attribute within its opening tag. Next, the style attribute is set equal to color: red;, which will set the color of the paragraph text to red within the browser.

If you’d like to add *more* than one style with inline styles, simply keep adding to the style attribute. Make sure to end the styles with a semicolon (;).

<p style="color: red; font-size: 20px;">I'm learning to code!</p>

# The <style> Tag

Inline styles are a fast way of styling HTML, but they also have limitations. If you wanted to style, for example, multiple <h1> elements, you would have to add inline styling to each element manually. In addition, you would also have to maintain the HTML code when additional <h1> elements are added.

Fortunately, HTML allows you to write CSS code in its own dedicated section with the <style> element. CSS can be written between opening and closing <style> tags. To use the <style> element, it must be placed inside of the <head> element.

<head>

<style>

</style>

</head>

After adding a <style> tag in the head section, you can begin writing CSS code.

<head>

<style>

p {

color: red;

font-size: 20px;

}

</style>

</head>

The CSS code in the example above changes the color of all paragraph text to red and also changes the size of the text to 20 pixels. Note how the syntax of the CSS code matches (for the most part) the syntax you used for inline styling. The main difference is that you can specify which elements to apply the styling to.

# The .css file

You can create a CSS file by using the **.css** file name extension, like so: **style.css**

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

# Linking the CSS File

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

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

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

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

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

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

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

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

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

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

# Tag Name

CSS can select HTML elements by using an element’s tag name. A tag name is the word (or character) between HTML angle brackets.

For example, in HTML, the tag for a paragraph element is <p>. The CSS syntax for selecting <p> elements is:

p {

}

In the example above, all paragraph elements will be selected using a CSS *selector*. The selector in the example above is p. Note that the CSS selector matches the HTML tag for that element, but without the angle brackets.

# Class Name

CSS is not limited to selecting elements by tag name. HTML elements can have more than just a tag name; they can also have *attributes*. One common attribute is the class attribute. It’s also possible to select an element by its class attribute.

For example, consider the following HTML:

<p class="brand">Sole Shoe Company</p>

The paragraph element in the example above has a class attribute within the <p> tag. The class attribute is set to "brand". To select this element using CSS, we could use the following CSS selector:

.brand {

}

To select an HTML element by its class using CSS, a period (.) must be prepended to the class’s name. In the example above case, the class is brand, so the CSS selector for it is .brand.

**The HTML <head> element is a container for all the head elements: <title>, <style>, <meta>, <link>, <script>, and <base>.**

**<meta charset="UTF-8">  
<meta name="description" content="Free Web tutorials">  
<meta name="keywords" content="HTML,CSS,XML,JavaScript">  
<meta name="author" content="John Doe">**

[**https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_head\_base**](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_head_base)

**Types of list:**

**Distribution list:**

**<dl> <dt>Coffee</dt> <dd>- black hot drink</dd> <dt>Milk</dt> <dd>- white cold drink</dd> </dl>**

**​**

**output**

**​**

**Coffee**

**- black hot drink**

**Milk**

**- white cold drink**

**Tags used for table:**

|  |  |
| --- | --- |
| **Tag** | **Description** |
| [**<table>**](https://www.w3schools.com/tags/tag_table.asp) | **Defines a table** |
| [**<th>**](https://www.w3schools.com/tags/tag_th.asp) | **Defines a header cell in a table** |
| [**<tr>**](https://www.w3schools.com/tags/tag_tr.asp) | **Defines a row in a table** |
| [**<td>**](https://www.w3schools.com/tags/tag_td.asp) | **Defines a cell in a table** |
| [**<caption>**](https://www.w3schools.com/tags/tag_caption.asp) | **Defines a table caption** |
| [**<colgroup>**](https://www.w3schools.com/tags/tag_colgroup.asp) | **Specifies a group of one or more columns in a table for formatting** |
| [**<col>**](https://www.w3schools.com/tags/tag_col.asp) | **Specifies column properties for each column within a <colgroup> element** |
| [**<thead>**](https://www.w3schools.com/tags/tag_thead.asp) | **Groups the header content in a table** |
| [**<tbody>**](https://www.w3schools.com/tags/tag_tbody.asp) | **Groups the body content in a table** |
| [**<tfoot>**](https://www.w3schools.com/tags/tag_tfoot.asp) | **Groups the footer content in a table** |

**Padding – space b/n text in cell and border**

[**https://www.w3schools.com/html/html\_responsive.asp**](https://www.w3schools.com/html/html_responsive.asp)

[**https://bootstrapmade.com/demo/iPortfolio/**](https://bootstrapmade.com/demo/iPortfolio/)

# Multiple Classes

Luckily, it’s possible to add more than one class name to an HTML element’s class attribute.

For instance, perhaps there’s a heading element that needs to be green and bold. You could write two CSS rules like so:

.green {

color: green;

}

.bold {

font-weight: bold;

}

Then, you could include both of these classes on one HTML element like this:

<h1 class="green bold"> ... </h1>

We can add multiple classes to an HTML element’s class attribute by separating them with a space. This enables us to mix and match CSS classes to create many unique styles without writing a custom class for every style combination needed.

# ID Name

If an HTML element needs to be styled uniquely (no matter what classes are applied to the element), we can add an ID to the element. To add an ID to an element, the element needs an id attribute:

<h1 id="large-title"> ... </h1>

Then, CSS can select HTML elements by their id attribute. To select an id element, CSS prepends the id name with a hashtag (#). For instance, if we wanted to select the HTML element in the example above, it would look like this:

#large-title {

}

The id name is large-title, therefore the CSS selector for it is #large-title.

# Classes and IDs

CSS can select HTML elements by their tag, class, and ID. CSS classes and IDs have different purposes, which can affect which one you use to style HTML elements.

CSS classes are meant to be reused over many elements. By writing CSS classes, you can style elements in a variety of ways by mixing classes on HTML elements.

For instance, imagine a page with two headlines. One headline needs to be bold and blue, and the other needs to be bold and green. Instead of writing separate CSS rules for each headline that repeat each other’s code, it’s better to write a .bold CSS rule, a .green CSS rule, and a .blue CSS rule. Then you can give one headline the bold green classes, and the other the bold blue classes.

While classes are meant to be used many times, an ID is meant to style only one element.

Since IDs override class and tag styles, they should be used sparingly and only on elements that need to always appear the same.

# Specificity

Specificity is the order by which the browser decides which CSS styles will be displayed. A best practice in CSS is to style elements while using the lowest degree of specificity, so that if an element needs a new style, it is easy to override.

IDs are the most specific selector in CSS, followed by classes, and finally, tags. For example, consider the following HTML and CSS:

<h1 class="headline">Breaking News</h1>

h1 {

color: red;

}

.headline {

color: firebrick;

}

In the example code above, the color of the heading would be set to firebrick, as the class selector is more specific than the tag selector. If an ID attribute (and selector) were added to the code above, the styles within the ID selector’s body would override all other styles for the heading. The only way to override an ID is to add another ID with additional styling.

Over time, as files grow with code, many elements may have IDs, which can make CSS difficult to edit, since a new, more specific style must be created to change the style of an element.

To make styles easy to edit, it’s best to style with a tag selector, if possible. If not, add a class selector. If that is not specific enough, then consider using an ID selector.

**<id> >> <class> >> <tag> : overriding elements**

**Denoted by id= #idname{}**

**Class=.classname{}**

**Tag=tagname{}**

# Chaining Selectors

When writing CSS rules, it’s possible to require an HTML element to have two or more CSS selectors at the same time.

This is done by combining multiple selectors, which we will refer to as chaining. For instance, if there was a .special class for h1 elements, the CSS would look like:

h1.special {

}

The code above would select only the h1 elements that have a class of special. If a p element also had a class of special, the rule in the example would not style the paragraph.

# Nested Elements

In addition to chaining selectors to select elements, CSS also supports selecting elements that are nested within other HTML elements. For instance, consider the following HTML:

<ul class='main-list'>

<li> ... </li>

<li> ... </li>

<li> ... </li>

</ul>

The nested <li> elements are selected with the following CSS:

.main-list li {

}

In the example above, .main-list selects the .main-list element (the unordered list element). The nested <li> are selected by adding li to the selector, separated by a space, resulting in .main-list li as the final selector (note the space in the selector).

Selecting elements in this way can make our selectors even more specific by making sure they appear in the context we expect.

# Chaining and Specificity

In the last exercise, instead of selecting all h5 elements, you selected only the h5 elements nested inside the .description elements. This CSS selector was more specific than writing only h5. Adding more than one tag, class, or ID to a CSS selector increases the specificity of the CSS selector.

For instance, consider the following CSS:

p {

color: blue;

}

.main p {

color: red;

}

Both of these CSS rules define what a p element should look like. Since .main p has a class and a p tag as its selector, only the p elements inside the .main element will appear red. This occurs despite there being another more general rule that states p elements should be blue.

# Important

There is one thing that is even more specific than IDs: !important. !important can be applied to specific attributes instead of full rules. It will override any style no matter how specific it is. As a result, it should almost never be used. Once !important is used, it is very hard to override.

The syntax of !important in CSS looks like this:

p {

color: blue !important;

}

.main p {

color: red;

}

Since !important is used on the p selector’s color attribute, all p elements will appear blue, even though there is a more specific .main p selector that sets the color attribute to red.

The !important flag is only useful when an element appears the same way 100% of the time. Since it’s almost impossible to guarantee that this will be true throughout a project and over time, it’s best to avoid !important altogether. If you ever see !important used (or are ever tempted to use it yourself) we strongly recommend reorganizing your CSS. Making your CSS more flexible will typically fix the immediate problem and make your code more maintainable in the long run.

# Multiple Selectors

In order to make CSS more concise, it’s possible to add CSS styles to multiple CSS selectors all at once. This prevents writing repetitive code.

For instance, the following code has repetitive style attributes:

h1 {

font-family: Georgia;

}

.menu {

font-family: Georgia;

}

Instead of writing font-family: Georgia twice for two selectors, we can separate the selectors by a comma to apply the same style to both, like this:

h1,

.menu {

font-family: Georgia;

}

By separating the CSS selectors with a comma, both the h1 and the .menu elements will receive the font-family: Georgia styling.

**REVIEW**

* CSS can change the look of HTML elements. In order to do this, CSS must select HTML elements, then apply styles to them.
* CSS can select HTML elements by tag, class, or ID.
* Multiple CSS classes can be applied to one HTML element.
* Classes can be reusable, while IDs can only be used once.
* **IDs are more specific than classes, and classes are more specific than tags. That means IDs will override any styles from a class, and classes will override any styles from a tag selector.**
* Multiple selectors can be chained together to select an element. This raises the specificity, but can be necessary.
* Nested elements can be selected by separating selectors with a space.
* **The !important flag will override any style, however it should almost never be used, as it is extremely difficult to override.**
* Multiple unrelated selectors can receive the same styles by separating the selector names with commas.

# Font Family

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

h1 {

font-family: Garamond;

}

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

1. The font specified in a stylesheet must be installed on a user’s computer in order for that font to display when a user visits the web page.
2. The default typeface for all HTML elements is Times New Roman. You may be familiar with this typeface if you have ever used a formatted word processor. If no font-family attribute is defined, the page will appear in Times New Roman.
3. It’s a good practice to limit the number of typefaces used on a web page to 2 or 3. This helps the page load faster in some cases and is usually a good design decision.
4. When the name of a typeface consists of more than one word, it’s a best practice to enclose the typeface’s name in quotes, like so:

h1 {

font-family: "Courier New";

}

# Font Size

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

p {

font-size: 18px;

}

# Font Weight

In CSS, the font-weight property controls how bold or thin text appears.

p {

font-weight: bold;

}

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

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

# Text Align

To align text we can use the   property. The text-align property will align text to the element that holds it, otherwise known as its parent.

h1 {

text-align: right;

}

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

1. left — aligns text to the left hand side of its parent element, which in this case is the browser.
2. center — centers text inside of its parent element.
3. right — aligns text to the right hand side of its parent element.

# Color

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

* Foreground color
* Background color

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

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

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

* color: this property styles an element’s foreground color
* background-color: this property styles an element’s background color

h1 {

color: red;

background-color: blue;

}

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

# Opacity

Opacity is the measure of how transparent an element is. It’s measured from 0 to 1, with 1 representing 100%, or fully visible and opaque, and 0 representing 0%, or fully invisible.

Opacity can be used to make elements fade into others for a nice overlay effect. To adjust the opacity of an element, the syntax looks like this:

.overlay {

opacity: 0.5;

}

In the example above, the .overlay element would be 50% visible, letting whatever is positioned behind it show through.

# Background Image

CSS has the ability to change the background of an element. One option is to make the background of an element an image. This is done through the CSS property background-image. Its syntax looks like this:

.main-banner {

background-image: url("https://www.example.com/image.jpg");

}

1. The background-image property will set the element’s background to display an image.
2. The value provided to background-image is a url. The url should be a url to an image. The url can be a file within your project, or it can be a link to an external site. To link to an image inside an existing project, you must provide a relative file path. If there was an image folder in the project, with an image named mountains.jpg, the relative file path would look like:

.main-banner {

background-image: url("images/mountains.jpg");

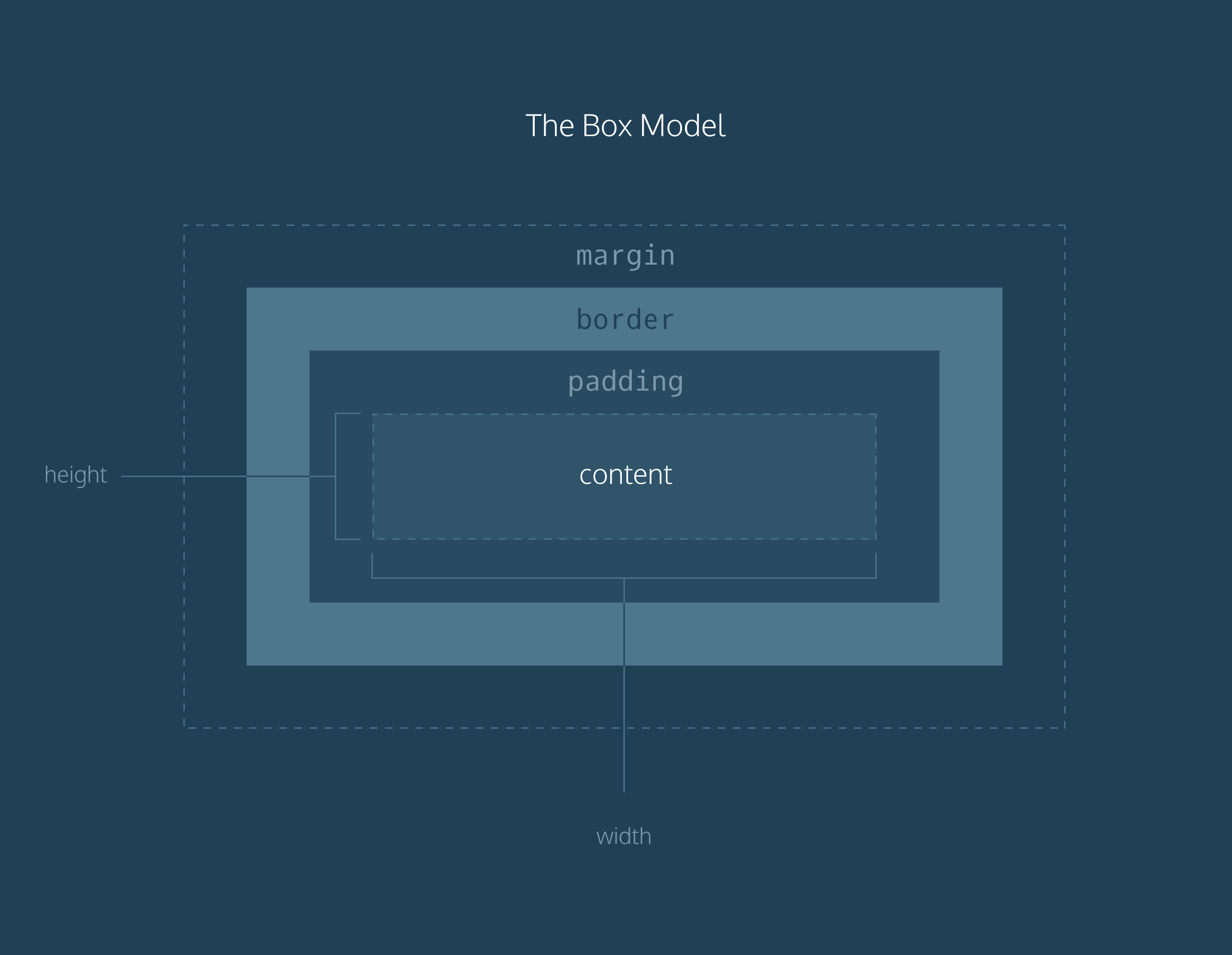
}

# The Box Model

The box model comprises the set of properties which define parts of an element that take up space on a web page. The model includes the content area’s size (width and height) and the element’s padding, border, and margin. The properties include:

1. Width and height — specifies the width and height of the content area.
2. Padding — specifies the amount of space between the content area and the border.
3. Border — specifies the thickness and style of the border surrounding the content area and padding.
4. Margin — specifies the amount of space between the border and the outside edge of the element.

The image to the right is a visual representation of the box model.

****

# Height and Width

An element’s content has two dimensions: a height and a width. By default, the dimensions of an HTML box are set to hold the raw contents of the box.

The CSS height and width properties can be used to modify these default dimensions.

p {

height: 80px;

width: 240px;

}

In this example, the height and width of paragraph elements are set to 80 pixels and 240 pixels, respectively — the px in the code above stands for pixels.

Pixels allow you to set the exact size of an element’s box (width and height). When the width and height of an element are set in pixels, it will be the same size on all devices — an element that fills a laptop screen will overflow a mobile screen.

# Borders

A border is a line that surrounds an element, like a frame around a painting. Borders can be set with a specific width, style, and color.

1. width — The thickness of the border. A border’s thickness can be set in pixels or with one of the following keywords: thin, medium, or thick.
2. style — The design of the border. Web browsers can render any of [10 different styles](https://developer.mozilla.org/en-US/docs/Web/CSS/border-style#Values). Some of these styles include: none, dotted, and solid.
3. color — The color of the border. Web browsers can render colors using a few different formats, including [140 built-in color keywords](https://developer.mozilla.org/en-US/docs/Web/CSS/color_value).

p {

border: 3px solid coral;

}

In the example above, the border has a width of 3 pixels, a style of solid and a color of coral. All three properties are set in one line of code.

The default border is medium none color, where color is the current color of the element. If width, style, or color are not set in the CSS file, the web browser assigns the default value for that property.

p.content-header {

height: 80px;

width: 240px;

border: solid coral;

}

In this example, the border style is set to solid and the color is set to coral. The width is not set, so it defaults to medium.

The border-style property may be specified using one, two, three, or four values.

* When **one** value is specified, it applies the same style to **all four sides**.
* When **two** values are specified, the first style applies to the **top and bottom**, the second to the **left and right**.
* When **three** values are specified, the first style applies to the **top**, the second to the **left and right**, the third to the **bottom**.
* When **four** values are specified, the styles apply to the **top**, **right**, **bottom**, and **left** in that order (clockwise).

Each value is a keyword chosen from the list below.

### **Values**

**<line-style>**

Describes the style of the border. It can have the following values:

|  |  |  |
| --- | --- | --- |
| none |  | Like the hidden keyword, displays no border. Unless a [background-image](https://developer.mozilla.org/en-US/docs/Web/CSS/background-image) is set, the computed value of the same side's [border-width](https://developer.mozilla.org/en-US/docs/Web/CSS/border-width) will be 0, even if the specified value is something else. In the case of table cell and border collapsing, the none value has the lowest priority: if any other conflicting border is set, it will be displayed. |
| hidden |  | Like the none keyword, displays no border. Unless a [background-image](https://developer.mozilla.org/en-US/docs/Web/CSS/background-image) is set, the computed value of the same side's [border-width](https://developer.mozilla.org/en-US/docs/Web/CSS/border-width) will be 0, even if the specified value is something else. In the case of table cell and border collapsing, the hidden value has the highest priority: if any other conflicting border is set, it won't be displayed. |
| dotted |  | Displays a series of rounded dots. The spacing of the dots is not defined by the specification and is implementation-specific. The radius of the dots is half the computed value of the same side's [border-width](https://developer.mozilla.org/en-US/docs/Web/CSS/border-width). |
| dashed |  | Displays a series of short square-ended dashes or line segments. The exact size and length of the segments are not defined by the specification and are implementation-specific. |
| solid |  | Displays a single, straight, solid line. |
| double |  | Displays two straight lines that add up to the pixel size defined by [border-width](https://developer.mozilla.org/en-US/docs/Web/CSS/border-width). |
| groove |  | Displays a border with a carved appearance. It is the opposite of ridge. |
| ridge |  | Displays a border with an extruded appearance. It is the opposite of groove. |
| inset |  | Displays a border that makes the element appear embedded. It is the opposite of outset. When applied to a table cell with [border-collapse](https://developer.mozilla.org/en-US/docs/Web/CSS/border-collapse) set to collapsed, this value behaves like groove. |
| outset |  | Displays a border that makes the element appear embossed. It is the opposite of inset. When applied to a table cell with [border-collapse](https://developer.mozilla.org/en-US/docs/Web/CSS/border-collapse) set to collapsed, this value behaves like ridge. |

**Color**

<https://developer.mozilla.org/en-US/docs/Web/CSS/color_value>

# Border Radius

Ever since we revealed the borders of boxes, you may have noticed that the borders highlight the true shape of an element’s box: square. Thanks to CSS, a border doesn’t have to be square.

You can modify the corners of an element’s border box with the border-radius property.

div.container {

border: 3px solid rgb(22, 77, 100);

border-radius: 5px;

}

The code in the example above will set all four corners of the border to a radius of 5 pixels (i.e. the same curvature that a circle with radius 5 pixels would have).

You can create a border that is a perfect circle by setting the radius equal to the height of the box, or to 100%.

div.container {

height: 60px;

width: 60px;

border: 3px solid rgb(22, 77, 100);

border-radius: 100%;

}

The code in the example above creates a <div> that is a perfect circle.

# Padding I

The space between the contents of a box and the borders of a box is known as padding. Padding is like the space between a picture and the frame surrounding it. In CSS, you can modify this space with the padding property.

p.content-header {

border: 3px solid coral;

padding: 10px;

}

The code in this example puts 10 pixels of space between the content of the paragraph (the text) and the borders, on all four sides.

The padding property is often used to expand the background color and make content look less cramped.

If you want to be more specific about the amount of padding on each side of a box’s content, you can use the following properties:

1. padding-top
2. padding-right
3. padding-bottom
4. padding-left

Each property affects the padding on only one side of the box’s content, giving you more flexibility in customization.

p.content-header {

border: 3px solid fuschia;

padding-bottom: 10px;

}

In the example above, only the bottom side of the paragraph’s content will have a padding of 10 pixels.

# Padding II

Another implementation of the padding property lets you specify exactly how much padding there should be on each side of the content in a single declaration.

p.content-header {

border: 3px solid grey;

padding: 6px 11px 4px 9px;

}

In the example above, the four values 6px 11px 4px 9px correspond to the amount of padding in a clockwise rotation. In order, it specifies the amount of padding on the top (6 pixels), right (11 pixels), bottom (4 pixels), and left (9 pixels) sides of the content.

When using this implementation of the padding property, we must specify a padding value for all four sides of the element.

However, if the top and bottom values for padding will equal each other, and the left and right values for padding will also equal each other, you can use the following shortcut:

p.content-header {

padding: 5px 10px;

}

The first value, 5px, sets the padding value for the top and bottom sides of the content. The second value, 10px, sets the padding value for the left and right sides of the content.

**MARGINS1**

Margin refers to the space directly outside of the box. The margin property is used to specify the size of this space.

p {

border: 1px solid aquamarine;

margin: 20px;

}

The code in the example above will place 20 pixels of space on the outside of the paragraph’s box on all four sides. This means that other HTML elements on the page cannot come within 20 pixels of the paragraph’s border.

If you want to be even more specific about the amount of margin on each side of a box, you can use the following properties:

1. margin-top
2. margin-right
3. margin-bottom
4. margin-left

Each property affects the margin on only one side of the box, providing more flexibility in customization.

p {

border: 3px solid DarkSlateGrey;

margin-right: 15px;

}

**MARGINS II**

A similar implementation of the margin property is used to specify exactly how much margin there should be on each side of the box in a single declaration.

p {

margin: 6px 10px 5px 12px;

}

In the example above, the four values 6px 10px 5px 12px refer to the amount of margin around the box in a clockwise rotation. In order, it specifies the amount of margin on the top (6 pixels), right (10 pixels), bottom (5 pixels), and left (12 pixels) sides of the box.

When using this implementation of the margin property, the margin value must be specified for all four sides of the box.

Just like the padding shortcut, when you’re certain that the top and bottom values for margin will equal each other, and that the left and right values for margin will also equal each other, you can use the following shortcut:

p {

margin: 6px 12px;

}

The first value, 6px, sets a margin value for the top and bottom of the box. The second value, 12px, sets a margin value for the left and right sides of the box.

# Auto

The margin property also lets you center content. However, you must follow a few syntax requirements. Take a look at the following example:

div {

margin: 0 auto;

}

In the example above, margin: 0 auto; will center the divs in their containing elements. The 0 sets the top and bottom margins to 0 pixels. The auto value instructs the browser to adjust the left and right margins until the element is centered within its containing element.

The div elements in the example above should center within an element that fills the page, but this doesn’t occur. Why?

In order to center an element, a width must be set for that element. Otherwise, the width of the div will be automatically set to the full width of its containing element, like the <body>, for example. It’s not possible to center an element that takes up the full width of the page.

div.headline {

width: 400px;

margin: 0 auto;

}

In the example above, the width of the div is set to 400 pixels, which is less than the width of most screens. This will cause the div to center within a containing element that is greater than 400 pixels wide.

# Margin Collapse

As you have seen, padding is space added inside an element’s border, while margin is space added outside an element’s border. One additional difference is that top and bottom margins, also called vertical margins, collapse, while top and bottom padding does not.

Horizontal margins (left and right), like padding, are always displayed and added together. For example, if two divs with ids #div-one and #div-two, are next to each other, they will be as far apart as the sum of their adjacent margins.

#img-one {

margin-right: 20px;

}

#img-two {

margin-left: 20px;

}

In this example, the space between the #img-one and #img-two borders is 40 pixels. The right margin of #img-one (20px) and the left margin of #img-two (20px) add to make a total margin of 40 pixels.

Unlike horizontal margins, vertical margins do not add. Instead, the larger of the two vertical margins sets the distance between adjacent elements.

#img-one {

margin-bottom: 30px;

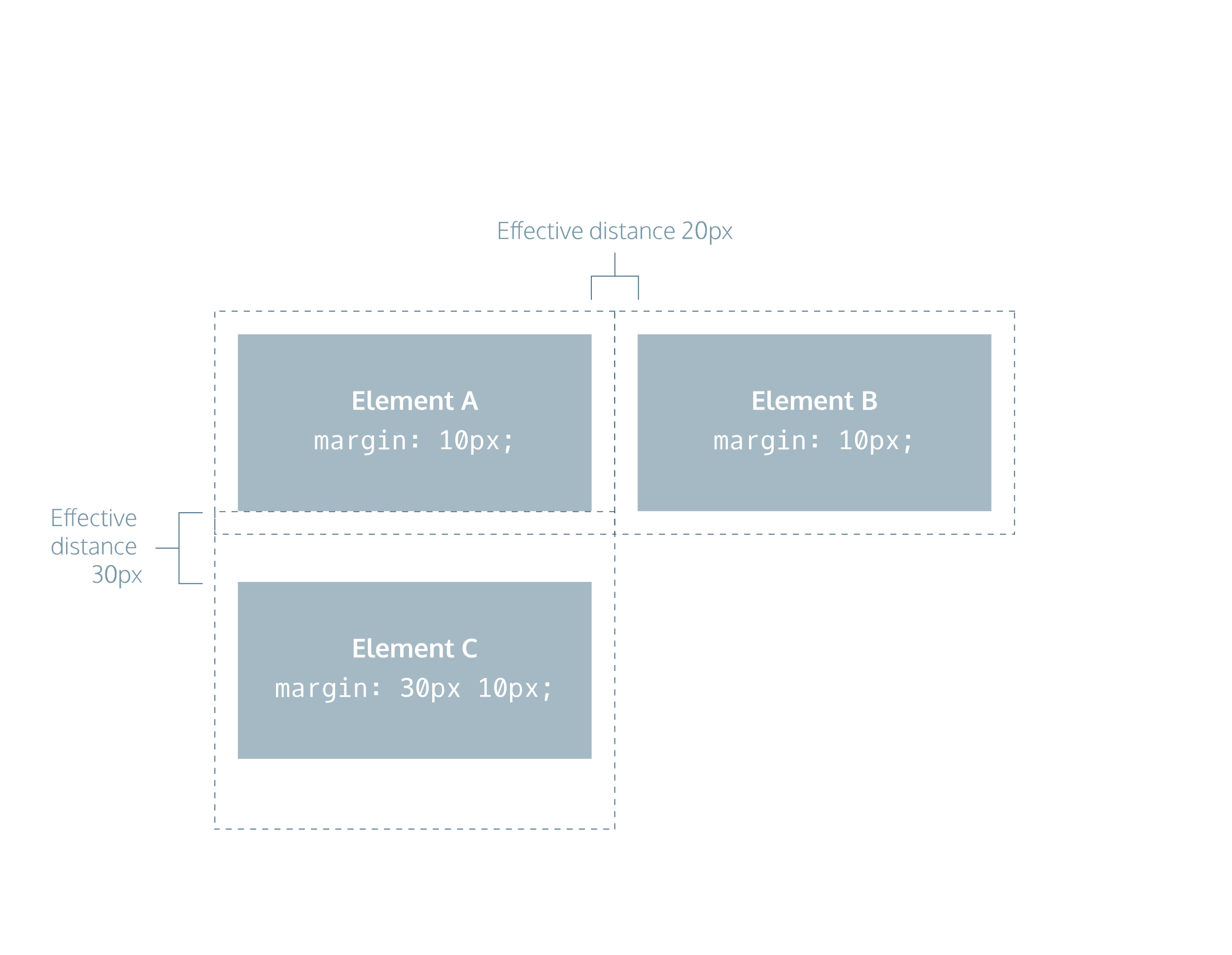
}

#img-two {

margin-top: 20px;

}

In this example, the vertical margin between the #img-one and #img-two elements is 30 pixels. Although the sum of the margins is 50 pixels, the margin collapses so the spacing is only dependent on the #img-one bottom margin.It may be helpful to think of collapsing vertical margins as a short person trying to push a taller person. The tall person has longer arms and can easily push the short person, while the person with short arms cannot reach the person with long arms.



# Overflow

All of the components of the box model comprise an element’s size. For example, an image that has the following dimensions is 364 pixels wide and 244 pixels tall.

* 300 pixels wide
* 200 pixels tall
* 10 pixels padding on the left and right
* 10 pixels padding on the top and bottom
* 2 pixels border on the left and right
* 2 pixels border on the top and bottom
* 20 pixels margin on the left and right
* 10 pixels margin on the top and bottom

The total dimensions (364px by 244px) are calculated by adding all of the vertical dimensions together and all of the horizontal dimensions together. Sometimes, these components result in an element that is larger than the parent’s containing area.

How can we ensure that we can view all of an element that is larger than its parent’s containing area?

The overflow property controls what happens to content that spills, or overflows, outside its box. It can be set to one of the following values:

* hidden - when set to this value, any content that overflows will be hidden from view.
* scroll - when set to this value, a scrollbar will be added to the element’s box so that the rest of the content can be viewed by scrolling.
* visible - when set to this value, the overflow content will be displayed outside of the containing element. Note, this is the default value.

p {

overflow: scroll;

}

In the example above, if any of the paragraph content overflows (perhaps a user resizes their browser window), a scrollbar will appear so that users can view the rest of the content.

The overflow property is set on a parent element to instruct a web browser how to render child elements. For example, if a div’s overflow property is set to scroll, all children of this div will display overflowing content with a scroll bar.

# Resetting Defaults

All major web browsers have a default stylesheet they use in the absence of an external stylesheet. These default stylesheets are known as user agent stylesheets. In this case, the term “[user agent](https://en.wikipedia.org/wiki/User_agent)“ is a technical term for the browser.

User agent stylesheets often have default CSS rules that set default values for padding and margin. This affects how the browser displays HTML elements, which can make it difficult for a developer to design or style a web page.

Many developers choose to reset these default values so that they can truly work with a clean slate.

\* {

margin: 0;

padding: 0;

}

The code in the example above resets the default margin and padding values of all HTML elements. It is often the first CSS rule in an external stylesheet.

Note that both properties are both set to 0. When these properties are set to 0, they do not require a unit of measurement.

# Visibility

Elements can be hidden from view with the visibility property.

The visibility property can be set to one of the following values:

1. hidden — hides an element.
2. visible — displays an element.

<ul>

<li>Explore</li>

<li>Connect</li>

<li class="future">Donate</li>

<ul>

.future {

visibility: hidden;

}

In the example above, the list item with a class of future will be hidden from view in the browser.

Keep in mind, however, that users can still view the contents of the list item (e.g., Donate) by viewing the source code in their browser. Furthermore, the web page will only hide the contents of the element. It will still leave an empty space where the element is intended to display.

**Note:** What’s the difference between display: none and visibility: hidden? An element with display: none will be completely removed from the web page. An element with visibility: hidden, however, will not be visible on the web page, but the space reserved for it will.

# Review

1. The box model comprises a set of properties used to create space around and between HTML elements.
2. The height and width of a content area can be set in pixels or percentage.
3. Borders surround the content area and padding of an element. The color, style, and thickness of a border can be set with CSS properties.
4. Padding is the space between the content area and the border. It can be set in pixels or percent.
5. Margin is the amount of spacing outside of an element’s border.
6. Horizontal margins add, so the total space between the borders of adjacent elements is equal to the sum of the right margin of one element and the left margin of the adjacent element.
7. Vertical margins collapse, so the space between vertically adjacent elements is equal to the larger margin.
8. margin: 0 auto horizontally centers an element inside of its parent content area, if it has a width.
9. The overflow property can be set to display, hide, or scroll, and dictates how HTML will render content that overflows its parent’s content area.
10. The visibility property can hide or show elements.

**17-05-2020**

**CHANGING THE BOX MODEL**

# Why Change the Box Model?

The last lesson focused on the most important aspects of the box model: box dimensions, borders, padding, and margin.

The box model, however, has an awkward limitation regarding box dimensions. This limitation is best illustrated with an example.

<h1>Hello World</h1>

h1 {

border: 1px solid black;

height: 200px;

width: 300px;

padding: 10px;

}

In the example above, a heading element’s box has solid, black, 1 pixel thick borders. The height of the box is 200 pixels, while the width of the box is 300 pixels. A padding of 10 pixels has also been set on all four sides of the box’s content.

Unfortunately, under the current box model, the border thickness and the padding will affect the dimensions of the box.

The 10 pixels of padding increases the height of the box to 220 pixels and the width to 320 pixels. Next, the 1-pixel thick border increases the height to 222 pixels and the width to 322 pixels.

Under this box model, the border thickness and padding are added to the overall dimensions of the box. This makes it difficult to accurately size a box. Over time, this can also make all of a web page’s content difficult to position and manage.

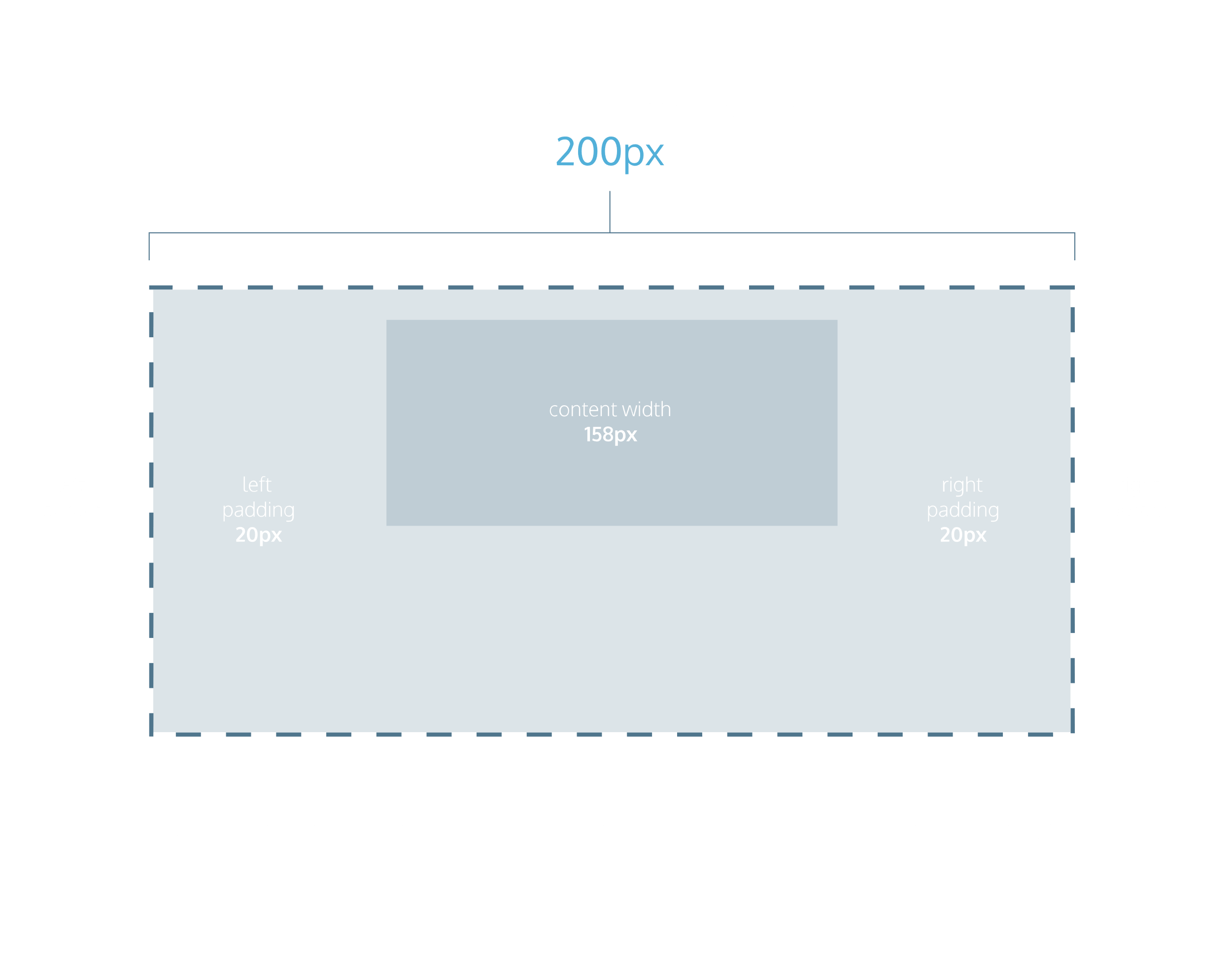
# Box Model: Content-Box

Many properties in CSS have a default value and don’t have to be explicitly set in the stylesheet.

For example, the default font-weight of text is normal, but this property-value pair is not typically specified in a stylesheet.

The same can be said about the box model that browsers assume. In CSS, the box-sizing property controls the type of box model the browser should use when interpreting a web page.

The default value of this property is content-box. This is the same box model that is affected by border thickness and padding.

****

# Box Model: Border-Box

Fortunately, we can reset the entire box model and specify a new one: border-box.

\* {

box-sizing: border-box;

}

The code in the example above resets the box model to border-box for all HTML elements. This new box model avoids the dimensional issues that exist in the former box model you learned about.

In this box model, the height and width of the box will remain fixed. The border thickness and padding will be included inside of the box, which means the overall dimensions of the box do not change.

<h1>Hello World</h1>

\* {

box-sizing: border-box;

}

h1 {

border: 1px solid black;

height: 200px;

width: 300px;

padding: 10px;

}

In the example above, the height of the box would remain at 200 pixels and the width would remain at 300 pixels. The border thickness and padding would remain entirely inside of the box.

**REVIEW**

Let’s review what you learned:

1. In the default box model, box dimensions are affected by border thickness and padding.
2. The box-sizing property controls the box model used by the browser.
3. The default value of the box-sizing property is content-box.
4. The value for the new box model is border-box.
5. The border-box model is not affected by border thickness or padding.

**CSS DISPLAY AND POSITIONING**

# Flow of HTML

A browser will render the elements of an HTML document that has no CSS from left to right, top to bottom, in the same order as they exist in the document. This is called the flow of elements in HTML.

In addition to the properties that it provides to style HTML elements, CSS includes properties that change how a browser positions elements. These properties specify where an element is located on a page, if the element can share lines with other elements, and other related attributes.

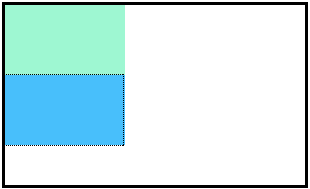
In this lesson, you will learn five properties for adjusting the position of HTML elements in the browser:

* position
* display
* z-index
* float
* clear

Each of these properties will allow us to position and view elements on a web page. They can be used in conjunction with any other styling properties you may know.

**Position**

Take a look at the *block-level* elements in the image below:



Block-level elements like these boxes create a *block* the full width of their parent elements, and they prevent other elements from appearing in the same horizontal space. The boxes in the image above were created with the following CSS:

.boxes {

width: 120px;

height: 70px;

}

and the following HTML:

<div class="boxes"></div>

<div class="boxes"></div>

Notice the block-level elements in the image above take up their own line of space and therefore don’t overlap each other. In the browser to the right you can see block-level elements also consistently appear on the left side of the browser. This is the default *position* for block-level elements.

The default position of an element can be changed by setting its position property. The position property can take one of four values:

1. static - the default value (it does not need to be specified)
2. relative
3. absolute
4. fixed

**Position: Relative**

One way to modify the default position of an element is by setting its position property to relative.

This value allows you to position an element *relative* to its default static position on the web page.

.box-bottom {

background-color: DeepSkyBlue;

position: relative;

}

Although the code in the example above instructs the browser to expect a relative positioning of the div, it does not specify where the div should be positioned on the page.

.box-bottom {

background-color: DeepSkyBlue;

position: relative;

top: 20px;

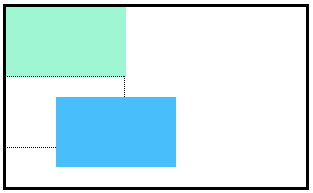
left: 50px;

}

In the example above, the <div> has been positioned using two of the four *offset properties*. The valid offset properties are:

1. top - moves the element down.
2. bottom - moves the element up.
3. left - moves the element right.
4. right - moves the element left.

In the example above, the <div> will be moved down 20 pixels and to the right 50 pixels from its default static position. The image below displays the new position of the box. The dotted line represents where the statically positioned (default) box was positioned.



Units for offset properties can be specified in pixels, ems, or percentages. Note that offset properties will not work if the value of the element’s position property is the default static.

# Position: Absolute

Another way of modifying the position of an element is by setting its position to absolute.

When an element’s position is set to absolute all other elements on the page will ignore the element and act like it is not present on the page. The element will be positioned relative to its closest positioned parent element.

.box-bottom {

background-color: DeepSkyBlue;

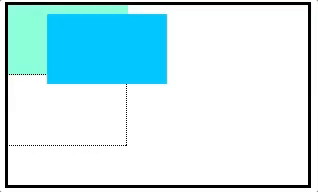
position: absolute;

top: 20px;

left: 50px;

}

In the example above, the .box-bottom <div> will be moved down and right from the top left corner of the view. If offset properties weren’t specified, the top box would be entirely covered by the bottom box. Take a look at the gif below:



The bottom box in this image (colored blue) is displaced from the top left corner of its container. It is 20 pixels lower and 50 pixels to the right of the top box.

# Position: Fixed

When an element’s position is set to absolute, as in the last exercise, the element will scroll with the rest of the document when a user scrolls.

We can fix an element to a specific position on the page (regardless of user scrolling) by setting its position to fixed.

.box-bottom {

background-color: DeepSkyBlue;

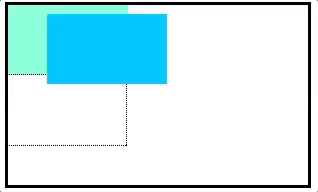
position: fixed;

top: 20px;

left: 50px;

}

In the example above, the .box-bottom <div> will remain fixed to its position no matter where the user scrolls on the page, like in the image below:



This technique is often used for navigation bars on a web page.

# Z-Index

When boxes on a web page have a combination of different positions, the boxes (and therefore, their content) can overlap with each other, making the content difficult to read or consume.

.box-top {

background-color: Aquamarine;

}

.box-bottom {

background-color: DeepSkyBlue;

position: absolute;

top: 20px;

left: 50px;

}

In the example above, the .box-bottom <div> ignores the .box-top <div> and overlaps it as a user scrolls.

The z-index property controls how far “back” or how far “forward” an element should appear on the web page when elements overlap. This can be thought of the depth of elements, with deeper elements appearing behind shallower elements.

The z-index property accepts integer values. Depending on their values, the integers instruct the browser on the order in which elements should be displayed on the web page.

.box-top {

background-color: Aquamarine;

position: relative;

z-index: 2;

}

.box-bottom {

background-color: DeepSkyBlue;

position: absolute;

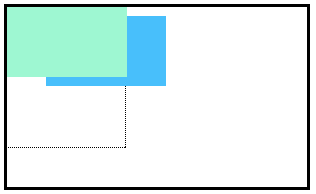
top: 20px;

left: 50px;

z-index: 1;

}

In the example above, we set the .box-top position to relative and the z-index to 2. We changed position to relative, because the z-index property does not work on static elements. The z-index of 2 moves the .box-top element forward, because it is greater than the .box-bottom z-index, 1. See the example image below:



In the image above, you can see the top box is moved in front of the bottom box.