Unit 2 Tutorials: Introduction to Coding Syntax

INSIDE UNIT 2

Anatomy of HTML

by Devmountain Tutorials

WHAT'S COVERED

This section will explore how to understand HTML by discussing:

1. REVIEW: HOW HTML IS USED IN WEB PAGES
2. HTML ELEMENTS AND TAGS
3. NESTING ELEMENTS
4. ATTRIBUTES
5. EMPTY ELEMENTS

1. REVIEW: HOW HTML IS USED IN WEB PAGES

In the previous section of this course, we talked a lot about the day to day for a web developer, the history of the internet and modern web application architecture. We ended by giving a brief look at CSS, HTML and JavaScript. In this unit, we will be expanding on all 3 of these in greater detail.

Let's start with HTML and review what we've learned so far.

HTML, or HyperText Markup Language, is markup language. Markup languages are used to group, annotate, and categorize content to describe the content's semantics in a way that computers can understand. Without HTML, the contents of a web page would look like this to a computer:

Plain Text

sophia-logo.png
Menu Search
Sign In
Purchase a Sophia membership for just $79 per month.
Choose a course to get started.
Low-cost, self-paced courses for college credit in a fresh and fun learning environment.
See how they work.
woman-in-chair.jpg

By annotating and grouping parts of the page with HTML, computers are better able to understand how content is related. For example, HTML is used to indicate that certain parts of a webpage are part of a navigational menu containing hyperlinks or buttons used to navigate from page to page.

```
<nav>
  <img src="sophia-logo.png">
  <button>Menu</button>
  <button>Search</button>
  <a href="/sign-in">Sign In</a>
</nav>
<aside class=”announcement”>
  <p>
```

```
    Purchase a Sophia membership for just $79
    per month.
  </p>
</aside>
<main>
  <section id="about-courses">
    <div class="left">
      <p>
        Low-cost, self-paced courses for college credit in
        a fresh and fun learning environment.
      </p>
      <button>See how they work</button>
    </div>
    <div class="right">
      <img src="woman-in-chair.jpg">
    </div>
  </section>
</main>
```

The example HTML above showcases many components of HTML syntax. We'll go over those components in more detail; then, you'll be able to understand exactly what's going on in the example HTML.

---

2. HTML ELEMENTS AND TAGS

HTML consists of a series of **elements** that enclose different parts of the content.

Elements are created with HTML tags. For example, this is a paragraph (p) element:

```
<p>The web is fun.</p>
```

To create a paragraph in HTML, start with the content of the paragraph.

The web is fun.

Then, enclose it in a **p** element by sandwiching the content in between an **opening tag**

```
<p>
```

and its corresponding **closing tag**.

```
</p>
```

.

So that it looks like this:

```
<p>The web is fun.</p>
```

Altogether, HTML elements are made of an opening tag, then content, then a closing tag.

This section of the course includes some embedded practice areas like the box below. You can practice following the instructions to see how the code changes the lines on the mini screen. The system does not record correct or incorrect answers, so feel free to experiment and practice with your new coding knowledge.

Edit the code in the **Editable Code** area in the box below. Follow the steps to mark that it is a strongly-worded phrase by wrapping it with the strong tags. This should make the text look bold. You'll see your changes update live, as you type, in the **Live Output** section. You can reset everything with the **Reset** button. Click on the **Show Solution** button to see the answer.

TRY IT

Follow these steps in the box below

1. Type the opening tag

```
<strong>
```

before the word **This**.

2. Close the element by typing

```
</strong>
```

after the word **phrase** .

3. Don't forget the **Reset** button if you need to start over.
4. Click the **Show Solution** button to see the answer. Hint: If your code matches the solution code, the **Editable Code** box will look like nothing has happened. You'll see the button name change to **Hide Solution** and you'll see the code applied in the Live output section right above.

Congratulations, you just wrote code! Next we'll talk about how to structure elements with nesting.

TERMS TO KNOW

HTML element

The individual building blocks of HTML. You can create an HTML element with HTML tags. For example, a paragraph element consists of an opening tag, then the content of the paragraph, then its closing tag.

Opening tag

Opening tags mark the beginning of an HTML element. It consists of the name of the element (for example, p for paragraph), wrapped in angle brackets.

Closing tag

Closing tags mark the end of an element. They look the same as opening tags, except they have a forward slash before the name of the element.

3. NESTING ELEMENTS

Elements can be **nested** within other elements. When an HTML element exists inside another HTML element, it is said to be nested within the outer HTML element.

For example, if we wanted to emphasize that the web is *very* fun, we could wrap the word **very** in an emphasis (**em**) element to indicate that it should be emphasized.

The web is very fun.
Take care to open and close the tags of nested elements properly! Note how, in the example above, we open the **p** element first then open the **em** element, which means we should close the **em** element first before closing the **p** element.

If you don't open and close HTML elements properly, the browser will try and guess at what you're trying to do which may lead to unexpected results. So, it's important to have tags open and close in a way that they're either inside or outside one another.

Here's an example of *improperly* nested elements:

The web is very fun.
Properly nesting elements might not sound like a big deal, but it is important for organizing your code and being able to apply attributes. We'll talk about attributes next.

TERM TO KNOW

Nesting elements

When an HTML element exists inside another HTML element, it is said to be nested within the outer HTML element.

4. ATTRIBUTES

Elements can also have **attributes**. Attributes contain extra information about the element and don't appear in the content of the page.

For example, we can add a **class** attribute to our paragraph:

The web is very fun.
The class attribute gives elements an identifying name that can be used to select the element to apply style information to it in CSS or to add scripted behavior to it in JavaScript. In the example above, class is the name of the attribute and comment is its value.

Another element you'll see used often is the anchor element

<a/>

which can turn its content into a hyperlink. Anchors can have several attributes, but the most common ones used with them are:
href="https://sophia.org/"
title="The Sophia homepage"

Attribute Name	Description	Example
href	The URL of the link.	
title	Extra information about the link, like a description of the page being linked to. This appears when a cursor hovers over the link as a tooltip.	

TRY IT

Follow these steps in the box below

Turn the content in the *Editable Code* section into a link to the Sophia homepage. Again, you'll be able to see your changes appear in the *Live Output* section. The *Reset* button will discard your changes and you can view the solution with *Show Solution* button.

1. Type
<a
before the words "'A link'".
2. Type
/a>
after the word "'homepage'". This is called "wrapping" the element, with a tag at the beginning and at the end of the line.
3. Look at the example column in the table above for the attributes. Using the whole phrase including the=, add the **href** and **title** attributes within the
<a>
element.
4. You can always peek at the solution if you need a hint.

If you look at more examples of HTML code, you might see attribute values written without quotation marks:

A link to the Sophia homepage.
The example above is completely valid, but it's still not a good idea to omit quotation marks because it can break your HTML in other circumstances. For example, if your attribute's value contains a space, leaving out quotation marks will break your markup:

A link to the Sophia homepage.
Even though you might see examples of HTML attributes without quotation marks, you should always include them. Doing so avoids unexpected problems and results in more readable code.

You might also see attributes written without values:

<input type="text" disabled>
The HTML above is shorthand for:

<input type=”text” disabled=”disabled”>
Attributes like **disabled** are **Boolean attributes** because their value can be set to true or false. The**disabled** attribute is given to **input** elements in order to prevent the user from typing into them. Boolean attributes are set to true if the attribute is present in the element or if its value is the same as the attribute name:

<input type=”text” disabled>
<input type=”text” disabled=”disabled”>
They’re set to false if they’re omitted from the element:

<input type=”text”>

TERMS TO KNOW

Attributes
Attributes contain extra information about the element and don’t appear in the content of the page.

Boolean attribute
A Boolean attribute is an attribute that can be set to true or false.

5. EMPTY ELEMENTS

Not all elements have an opening tag, content, and a closing tag. Some tags are empty elements and consist of a single tag.

Empty elements are typically used to insert or embed something — like an image — in the page. For example, the **img** element embeds an image file onto the page:

<img src="https://sophialearning.s3.amazonaws.com/static/logo_sophia-d5c178b4e98abe275fb9aa947126eb27021850b2c7a8c10b710b5245e20a655f.svg">
On the left you see the code, and on the right you see the image displayed.

This section has introduced you to many different parts of HTML. In the next section, we'll look at how to organize them to make a page.

TERMS TO KNOW

Empty elements
Empty elements, also known as void elements, consist of a single tag.

TERMS TO KNOW

Attributes
Attributes contain extra information about the element and don’t appear in the content of the page.

Boolean attribute
A Boolean attribute is an attribute that can be set to true or false.

Closing tag
Closing tags mark the end of an element. They look the same as opening tags, except they have a forward slash before the name of the element.

Empty elements
Empty elements, also known as void elements, consist of a single tag.

HTML element
The individual building blocks of HTML. You can create an HTML element with HTML tags. For example, a paragraph element consists of an opening tag, then the content of the paragraph, then its closing tag.

Nesting elements
When an HTML element exists inside another HTML element, it is said to be nested within the outer HTML element.

Opening tag
Opening tags mark the beginning of an HTML element. It consists of the name of the element (for example, p for paragraph), wrapped in angle brackets.

Creating an HTML Page

by Devmountain Tutorials

WHAT'S COVERED

This section will explore how to create an HTML page by discussing:

1. ELEMENTS IN A BASIC HTML PAGE
2. WHITESPACE AND COMMENTS IN HTML
3. CREATING A PAGE WITH TEXT CONTENT

1. ELEMENTS IN A BASIC HTML PAGE

Now that you've learned to recognize HTML elements and attributes, you'll be able to read HTML pages. After this section, you'll be able to create your own HTML page.

HTML pages or HTML documents on the web consist of several elements combined. Let's break down the specific elements that are used to create the example below.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>A Basic Page</title>
  </head>
  <body>
    <p>This is a basic page.</p>
  </body>
</html>
```

The elements that make up this example, in order of appearance, are:

Code Element	Element Name	Element Description
<!DOCTYPE html>	**doctype**	This is a historical artifact that is functionally obsolete in most cases but is required at the start of every HTML document.
<html></html>	**html**	It wraps all the content on the page and is also known as the **root** element.
<head></head>	**head**	This element contains everything you want to include on the page that isn't content that will appear to users. This may include a page description to appear in search results, CSS to style content, the title of the page, and more.
<meta charset="utf-8">	**meta**	This particular meta element tells browsers that the document uses the UTF-8 character set, which includes most characters from the majority of human written languages and emoji. It's not required for the rest of the page to work correctly, but it can help improve the page's backwards compatibility.
<title></title>:	**title**	Sets the title of the page. The title of the page will appear in the browser tab that the page is loaded in. It's also used to describe the page when it is bookmarked by the user.
<body></body>	**body**	This element contains all the content that should be displayed on the page including text, images, videos, etc.

BIG IDEA

HTML pages must start with the **doctype** element and wrap everything else in an html element. The **head** element contains content that won't appear to users and the**body** element contains content that should appear to users.

2. WHITESPACE AND COMMENTS IN HTML

In the example HTML page above, we formatted the code with lots of**whitespace**.

This is optional — for example, these two lines of code are equivalent:

```
<p>The web is fun.</p>
<p>The        web is
          fun.</p>
```

That's because, no matter how much whitespace you use in the content of an HTML element, the browser will reduce any sequential whitespace characters to a single space when parsing the code. For example, here's a paragraph with a nested link:

```
<p>
  You can learn more by clicking on
  this link:
  <a href="https://sophia.org/">
  link to Sophia
  </a>
</p>
```

Although the example above is nicely formatted, it'll read like this to the browser:

```
<p>You can learn more by clicking on this link: <a href="https://sophia.org/">link to Sophia</a></p>
```

Since the browser doesn't care about nicely formatted HTML, it seems like there's little point in putting effort into writing code that looks nice. However, even though browsers don't appreciate well-formatted code, you probably do!

It can be easier to understand your code if it's nicely formatted. In our examples, we've chosen to indent nested elements by two spaces. The visual cue makes it easier to tell that an element is nested inside another element. Writing well-formatted, readable code is so important in the software industry that many companies will require their employees to adhere to a style guide. For example, here is **Google's style guide for writing HTML**.

Another thing you can do to make code easier to read is to add annotations called comments. Most coding languages have syntax for writing comments that are ignored by the parser and invisible to users but remain in the source code so developers and engineers are able to read them.

```
<!-- This is an HTML comment -->
```

Comments may be used to explain the purpose or logic of a section of code. While you're learning, you can use comments to write notes to remind yourself what each portion of your code is doing. They're also used to communicate with coworkers and collaborators. Developers also use comments "remove" portions of their code without having to completely delete their work.

```
<p>This paragraph looks good.</p>
<!-- <p>But I'm not so sure about this one</p> -->
```

TERM TO KNOW

Whitespace

Whitespace is any amount of text that appears as blank space. Some examples of whitespace are space between words in a sentence, space between lines of a song, and indentation before a paragraph.

3. CREATING A PAGE WITH TEXT CONTENT

Let's continue working on the HTML page you saw earlier and start adding more content to it. Here are the contents of that HTML page again:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>A Basic Page</title>
  </head>
  <body>
    <p>This is a basic page.</p>
  </body>
</html>
```

The code above is a bit hard to follow since it contains so many tags. To make the example easier to read, we'll truncate it and leave out all other tags except the ones inside the **body** element:

```
<p>This is a basic page.</p>
```

Our HTML page isn't very exciting right now. It's just a basic example page. Let's make the page more purposeful and turn it into an article that users can read to learn more about you.

Most content on the web is text content like blog posts and news articles. This section will show you how to work with text content in HTML.

One of HTML's main jobs is to describe the semantics — the structure and meaning — of content. Most text content is structured with headings and paragraphs. So far, you've seen that HTML paragraphs are created with the p element. Headings are wrapped in a heading element, like so:

```
<h1>About Me</h1>
```

There are six heading elements — **h1, h2, h3, h4, h5, h6** — each representing a different level of content in the page. **h1** is the main heading, **h2** represents subheadings, h3 represents sub-subheadings, and so on. For example, here's how headings and paragraphs can be used to describe the semantics of a book:

```
<h1>How the Web Works</h1>
<p>By Anjelica</p>
<h2>Chapter 1: Inventing the World Wide Web</h2>
<p>The web was invented by Sir Tim Berners-Lee.</p>
<h2>Chapter 2: How HTML Works</h2>
<p>HTML is great.</p>
<h3>Anatomy of HTML</h3>
<p>HTML is made of elements.</p>
```

Use the example above to guide you as you create a new **About Me** page. Add content to the *Editable Code* field below in the box. Press *Show Solution* to check that your content looks similar to our solution. Press *Reset* to start over.

TRY IT

Follow these steps in the box below:

1. Replace the contents of our basic page, below, and turn it into an **About Me** page. The main heading should say **About Me**.
2. Add a short paragraph to introduce yourself and your name.
3. Create two subsections, titled **Likes** and **Dislikes**.
4. Add a short paragraphs to each new section where you talk about the things you like and dislike.

TERMS TO KNOW

Whitespace

Whitespace is any amount of text that appears as blank space. Some examples of whitespace are space between words in a sentence, space between lines of a song, and indentation before a paragraph.

Semantic HTML

by Devmountain Tutorials

WHAT'S COVERED

This section will explore semantic HTML by discussing:

1. WHY WRITE SEMANTIC HTML?
2. SEMANTIC ELEMENTS

1. WHY WRITE SEMANTIC HTML?

In a previous section, you learned that it's possible to make any HTML element look and behave differently using a combination of CSS and JavaScript. For example, in **Challenge 1.3 – Tools of the Trade**, we made a paragraph look like a button with CSS and used JavaScript to make the paragraph clickable.

We've included this interactive example again. When you click on **Hello, friend!** an alert box displays asking, "What's your name?" If you type in your name and click OK, you'll see the button update to say hello to you.

As you continue learning about HTML, you'll find many resources and articles about the importance of using **semantic HTML**. This means using HTML correctly by staying true to an element's original, intended purpose. In other words, even though it's possible to make a paragraph look and act like a button, it's bad practice and you shouldn't do it.

Semantic HTML doesn't just help make content more readable for humans, it also helps computer programs. Programs like assistive technology software rely on HTML to make sense of a web page's contents. For example, severely visually impaired people might not read web pages very often — they listen to them using programs called screen readers instead. Screen readers don't read out the entire contents of a web page from top to bottom. Instead, they might first read out the outline of the document using the page's headings and subheadings. That way, users can quickly find the information they care to hear more about. Without proper headings and subheadings, users will be forced to listen to the whole document read out loud. Other programs might use headings to determine the important keywords of a page. For example, Google looks at a page's headings to influence a page's search rankings.

Another way to understand the importance of writing semantic HTML is to compare and contrast how developers wrote HTML before semantics became a widespread concern. The next section will provide some historical context around HTML and semantics, starting with a look at earlier versions of HTML and their limited semantics.

TERM TO KNOW

Semantic HTML
To write semantic HTML means to use HTML correctly by staying true to an element's original, intended purpose.

2. SEMANTIC ELEMENTS

Early HTML had limited semantics, especially when it came to grouping related content together. Developers only had one element they could use to group content — the div element. However, since the div element does not suggest any meaning about its content to a browser, it only helped humans understand how content is grouped. Consider this navigation menu:

```
<div class=”navigation”>
  <a href=”/”>Home</a>
  <a href=”/about”>About</a>
  <a href="/sign-in">Sign In</a>
</div>
```

A screen reader wouldn't be able to tell that the **div** above makes up a navigational menu, even though it has **class=”navigation”**.

In the latest version of HTML — HTML5 — new elements, called **semantic elements**, were introduced and gave developers a more diverse set of elements to choose from when defining the structure of a document. Below are the names of a few of the new HTML5 elements with descriptions of how they might be used.

```
<<nav></nav>>
<html></html>
<article></article>
<section></section>
<aside></aside>
<header></header>
<footer></footer>
```

Code Element	Element Name	Element Description
	nav	A section with navigation links that appear often on a site.
	html	It wraps all the content on the page and is also known as the **root** element.
	article	A piece of content that is self-contained like a blog post, comment, or widget.
	section	A section of a document. For example, a video and its play/pause buttons should each be put in a separate section (one for the video and one for the play/pause buttons).
	aside	A section that is related to some content but doesn't belong in its main flow. One example that you're familiar with is the **Term to Know** box.
	header	A section that contains a heading. For example, many web pages have headers that contain the website's logo, title, and navigation menu. It can also be used inside **article** or **section**. For example, an article's header might have a title and a byline with the name of the author. Headers do not have to appear at the beginning of a page or section.

	footer	A page footer. This might have a copyright or links. Footers do not have to appear at the end of a page or section.

We can rewrite the example navigation menu above using the semantic element,**nav**:

```
<nav>
  <a href=”/”>Home</a>
  <a href=”/about”>About</a>
  <a href="/sign-in">Sign In</a>
</nav>
```

You've learned a lot in this challenge! You can recognize the different parts of HTML, combine elements to create a web page, and identify different semantic elements. In the next challenge we'll explore how to style HTML by using CSS.

TERMS TO KNOW

Semantic HTML

To write semantic HTML means to use HTML correctly by staying true to an element's original, intended purpose.

Anatomy of CSS

by Devmountain Tutorials

WHAT'S COVERED

This section will explore the parts of CSS by discussing:

1. REVIEW: HOW CSS IS USED IN WEB PAGES
2. CSS RULES SYNTAX
3. PROPERTIES AND VALUES
4. COMMENTS AND WHITESPACE

1. REVIEW: HOW CSS IS USED IN WEB PAGES

In Unit 1 we covered the role that CSS (Cascading Style Sheets) plays in a web page. CSS is a style sheet language. Style sheet languages are used to specify how a document should be presented to users. When the browser converts HTML to a visual format for users to see, it uses CSS to determine how each HTML element should look.

Web browsers will typically include very basic, minimal styles in their default CSS. If they didn't have default CSS, you wouldn't be able to view the content of any HTML page that didn't have its own CSS! Default styles exist to make sure content is readable even if the author of the page didn't include their own CSS. Different browsers might have different default styles but, in general, they're pretty similar — headings are large and bold, paragraphs are separated on different lines, and links are blue or sometimes purple.

Below you can see the default styles applied to each element.

Default styles are very bland though, so developers will add their own, additional CSS to give their pages a more distinctive look and feel. CSS can be used to make simple stylistic changes, like changing the font of a particular element.

Below you can see the same HTML code with CSS added to change the heading font.

In Unit 1, we demonstrated how CSS can be used to make more dramatic changes in visual style by making a paragraph look like a button. The language is so extensive that it can even be used to create animations. We also showed you examples from Codepen of developers using CSS to create visual art.

---

2. CSS RULES SYNTAX

When writing HTML, you use tags to create elements. When writing CSS, you use **selectors** and **declarations** to create rules. In other words, CSS syntax is used to define rules that specify groups of styles to apply to a particular elements or groups of elements on your web page.

Here's a rule you saw earlier that changes the font of **h1** elements:

```
h1 {
  font-family: sans-serif;
}
```

To create a rule, start with a **selector**.

```
h1
```

h1 will select the element that our rule should be applied to. In this case, we're selecting any **h1** elements.

Then, add a pair of curly braces. In the curly braces, you'll list one or more **style declarations**.

```
h1 {

}
```

Style declarations are pairs of **property** names and **values** that end with a semicolon.

```
h1 {
  font-family: sans-serif;
}
```

In the example above, **font-family** is the name of a property and **sans-serif** is its value.

Additional declarations should be listed on separate lines. For example, if we also want to change the color of **h1** elements, we can just add another declaration inside the curly braces like so:

```
h1 {
  font-family: sans-serif;
  color: purple;
}
```

Edit the CSS below in the *Editable Code* area so that **a** elements use a green, bold, sans-serif font. So far, the rule that selects all **a** elements only sets their **font-family** to sans-serif and **color** to green. Your task is to add another declaration to set the **font-weight** property to **bold**. You'll see your changes update live, as you type, in the *Live Output* section. You can reset everything with the *Reset* button. Click on the *Show Solution* button to see the answer.

TRY IT

Follow these steps in the box below:

1. Add a new line below **color: green**; and press the *Tab* key to indent the declaration by 2 spaces.
2. Start the declaration with the property name, **font-weight**. Then, add a colon (:) and a space.
3. Add the value you want to set the property to —**bold**.
4. End the your declaration with a semicolon (;).

Now that you understand how to read CSS and how CSS rules are structured, we can take a more detailed look at the particulars of the language, starting with properties and values.

TERMS TO KNOW

Rules
Rules are the basic building blocks of CSS. Rules are made of selectors and declarations.

Selectors
In CSS, selectors specify the elements or groups of elements that a rule should be applied to. A rule can have one or more selectors.

Style declarations
Style declarations, or simply declarations, are property-value pairs. A rule can have zero or more declarations.

3. PROPERTIES AND VALUES

CSS rules are made of declarations — property-value pairs used to tell the browser how elements should be styled. **Properties** are identifiers that represent the stylistic features to be modified. So far, you've encountered properties like font-family, font-weight, and color. Each property is assigned a value that indicates how the property should be styled.

CSS values can take many different forms. So far, we've shown examples of using keyword values to set an element's font and color like using the keyword **bold** to set **font-weight** and **green** to set **color**.

```
a {
  font-weight: bold;
  color: green;
}
```

Besides keywords, there are also numeric values and values that take the form of a function.

Numeric values are usually numbers followed by a unit of measure. For example, the **width** and **height** properties are styled with numeric values.

In the box below you can see how the width uses a numeric value.

Use numeric values to set an element's **padding-top** and **padding-bottom** properties by adding to the CSS rule in the Editable code section below. So far, the rule selects the **ul** element and sets its **background-color** to **aqua**. Add two more declarations so **padding-top** and **padding-bottom** are both set to **15px**. Click *Show Solution* to verify that you're on the right track. If you need to start over, press *Reset*.

TRY IT

Follow these steps in the box below:

1. Add a new line below background-color: aqua; to start a new declaration.
2. Set padding-top to 15px. Don't forget to end the declaration with a semicolon.
3. Add another declaration so padding-bottom is 15px. Again, don't forget the semicolon!

Other values are **functions**. For example, the **rgb()** function can be used to specify a color by the color's red, green, and blue values.

TERMS TO KNOW

Properties
Properties are identifiers that represent the stylistic features to be modified.

Font weight
Property that allows a choice such as "bold" to stylize text.

Color
Property that changes the color of the text.

Numeric values
Numeric values are usually numbers followed by a unit of measure, for height and width.

Function
CSS has functions that can be used as values. For example rgb() specifies a color and linear-gradient() specifies a gradient.

rgb()
Function that can be used to specify a color by the color's red, green, and blue values.

4. COMMENTS AND WHITESPACE

Like HTML, CSS has syntax for comments. CSS comments start with /* and end with */. Developers use CSS comments for the same reasons they use HTML comments — they're used to describe how the code works so others can understand it and can be used to temporarily "delete" a style without actually having to remove it.

```
aside {
  /* Asides should be indented just a bit so they're
     not confused with main body text. */
```

```
  margin-left: 50px;
}

p {
  /* font-size: 40px; */
}
```

For the most part, browsers ignore whitespace in CSS just like they do in HTML. For example the CSS below, though it's hard to read, is completely valid:

```
body {margin: 0 auto;}
h1, h2, h3, h4, h5, h6 {font-family: 'Helvetica', sans-serif; border-bottom: 1px solid blue;}
```

Of course, just because you can leave out whitespace you should still be diligent about adding whitespace to make your CSS easier to read and follow.

TERMS TO KNOW

Color
Property that changes the color of the text.

Font weight
Property that allows a choice such as "bold" to stylize text.

Function
Functions are used as a value for various CSS properties.

Numeric values
Numeric values are usually numbers followed by a unit of measure, for height and width.

Properties
Properties are identifiers that represent the stylistic features to be modified.

Rules
Rules are the basic building blocks of CSS. Rules are made of selectors and declarations.

Selectors
In CSS, selectors specify the elements or groups of elements that a rule should be applied to. A rule can have one or more selectors.

Style declarations
Style declarations, or simply declarations, are property-value pairs. A rule can have zero or more declarations.

rgb()
Function that can be used to specify a color by the color's red, green, and blue values.

Working with CSS and HTML Together

by Devmountain Tutorials

WHAT'S COVERED

This section will explore how CSS and HTML work together by discussing:

1. INCLUDING CSS IN AN HTML PAGE
2. SELECTING HTML ELEMENTS

1. INCLUDING CSS IN AN HTML PAGE

Now that you understand some basics about how CSS works, we can start integrating CSS into HTML and talk about how they work together. There are a couple different ways to apply CSS to an HTML page.

You can write CSS rules in a file separate from your HTML page. These files are known as **external stylesheets** and they usually have a **.css** file extension (though having a certain file extension is not a requirement). Then, you can reference any external stylesheet in your HTML with a link element. You can add the link anywhere in your HTML, but it usually goes in the page's head element.

Here's an example HTML document with a link that references an external stylesheet called **styles.css**.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=”utf-8”>
    <title>Yay, CSS</title>
    <link rel=”stylesheet” href=”styles.css”>
  </head>
  <body>
    <p>Ah, how nice.</p>
  </body>
</html>
```

And here's an example of the CSS in styles.css:

```
p {
 font-size: 40px;
 font-family: sans-serif;
}
```

Another way to apply CSS to HTML is to use an **internal stylesheet**. Unlike external stylesheets where you write CSS in a separate file, an internal stylesheet is written directly in your HTML inside a style element. Again, like with external stylesheets, you can place internal stylesheets anywhere in your HTML but it usually goes in the **head** element.

Here's our example again, this time with an internal stylesheet in place of an external one:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=”utf-8”>
    <title>Yay, CSS</title>
    <style>
      p {
      font-size: 40px;
      font-family: sans-serif;
      }
    </style>
  </head>
  <body>
    <p>Ah, how nice.</p>
  </body>
</html>
```

You can also add style declarations to an element through its style attribute. Below, we've removed the internal stylesheet and moved the **font-size** and **font-family** declarations into the **p** element's style attribute:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=”utf-8”>
    <title>Yay, CSS</title>
  </head>
  <body>
    <p style=”font-size: 40px; font-family: sans serif;”>
    Ah, how nice.
    </p>
```

```
  </body>
</html>
```

CSS declarations in an element's style attribute are known as **inline styles**; so-called because they appear "in line" with the HTML element being styled. Inline styles only apply to one element at a time. Best practices suggest avoiding using inline styles as best you can. Inline styles might make your page harder to change and update, especially since they only apply to one element at a time rather than a group of elements. They also make code harder to read since they mix CSS syntax with HTML syntax.

You have a variety of techniques for applying CSS to HTML — you can reference an external stylesheet with a link element, write CSS directly into the HTML as an internal stylesheet in a style element, or by styling one element at a time with inline styles. There are also different ways to write CSS rules so they select a wider or more narrow group of elements. That's what we'll cover next.

TERMS TO KNOW

External stylesheet
CSS rules in a file separate from your HTML page.

Internal stylesheet
An internal stylesheet is written directly in your HTML inside a style element.

Inline styles
CSS declarations in an element's style attribute. so-called because they appear "in line" with the HTML element being styled. Inline styles only apply to one element at a time.

2. SELECTING HTML ELEMENTS

So far, you've only seen examples of selectors being used to select a type of element, like using the h1 selector to apply styles to all level-one headings but there are many other types of selectors you have at your disposal. Some selectors select elements with a more narrow degree of specificity — for example, "select only paragraphs that are part of the important class" — and others can select elements based on where they appear relative to other elements — for example, "select the first paragraph of within a section".

Let's say you have a couple paragraphs to style:

```
<p>Let's put CSS in our HTML.</p>
<p>We can use an external stylesheet.</p>
<p>There are also internal stylesheets and inline styles.</p>
<p>Wow, exciting!</p>
```

You've already seen how to select all **p** elements and change their font.

```
p {
  font-family: sans-serif;
}
```

The selector above and the selectors we used in the examples so far are all type selectors Type selectors are simple to write — they just look exactly like the name of the element you want to select. For example, h1 will select all **h1** elements and **p** will select all **p** elements.

However, type selectors aren't sufficient when you want to select a more specific subset of elements. For example, we want some paragraphs to have a different color to show that they contain important information. First, it'd be easier for us to distinguish between info-heavy paragraphs and regular paragraphs if we categorized them in a class. We can do that by adding a class to the info-heavy paragraphs.

You can make up whatever class name you'd like — we decided to use **info** since it's short and descriptive:

```
<p>Let's put CSS in our HTML.</p>
<p class="info">We can use an external stylesheet.</p>
<p class="info">There are also internal stylesheets and inline styles.</p>
<p>Wow, exciting!</p>
```

Adding the **info** class does two things — it helps us remember that some paragraphs have important information in them and it enables us to use the **class selector** to select and style the info elements so they stand out. **Class selectors** start with a period (.) followed by the name of a class and will select all elements belonging to that class. For example, to select all the elements that are members of the info class, start with the class selector:

```
.info
```

Then we can list style declarations in a pair of curly braces like usual:

```
.info {
  color: blue;
}
```

Let's make those paragraphs even more exciting. Edit the HTML and CSS code in the *Editable Code* section to make the last paragraph seem "loud". You'll use HTML to give the last paragraph a class and CSS to select that paragraph and give it a larger font size. Like usual, *Show Solution* will show you the solution and you can start over with *Reset*.

TRY IT

Follow these steps in the box below:

1. In the HTML, give the last paragraph a class attribute and set its value to **"loud"**.
2. In CSS, add a new rule. Start with the class selector **.loud** followed by an opening curly brace **({)**.
3. Add a style declaration **font-size: 40px;**.
4. Don't forget the closing curly brace **(})**!

There are many other types of selectors you can take advantage of. Unfortunately, we won't be able to cover all of them in this course but there are plenty of excellent, free resources you can check out to learn more about different selectors. CSS Tricks is a great website for learning CSS, HTML, and JavaScript and they have an easy-to-follow breakdown on how CSS selectors work, which you can view by clicking on this link. If you like interactive educational materials, CSS Diner is a game where you can learn about different selectors and use them to select plates and food on a table; you can play the game here.

TERMS TO KNOW

External stylesheet

CSS rules in a file separate from your HTML page.

Inline styles

CSS declarations in an element's style attribute. so-called because they appear "in line" with the HTML element being styled. Inline styles only apply to one element at a time.

Internal stylesheet

An internal stylesheet is written directly in your HTML inside a style element.

Anatomy of JavaScript

by Devmountain Tutorials

WHAT'S COVERED

This section will explore the parts of JavaScript by discussing:

1. REVIEW: HOW JAVASCRIPT IS USED IN WEB PAGES
2. JAVASCRIPT SYNTAX PREVIEW
3. VARIABLES AND VALUES
4. CONDITIONALS AND LOOPS

1. REVIEW: HOW JAVASCRIPT IS USED IN WEB PAGES

In Unit 1, we showed an example of using JavaScript to make a paragraph element interactive and clickable. You also learned how browsers take in HTML, CSS, and JavaScript source code and execute the code, in order, to create a web page. We also covered how developers use JavaScript APIs like browser APIs and third-party APIs to build complex applications and went a bit into how browsers execute JavaScript.

Since JavaScript is a programming language, it's significantly more complex compared to HTML and CSS. There are entire, multi-week-long courses devoted to helping engineers learn JavaScript basics. Don't worry — we don't assume that you want to embark on that journey...*yet*. Instead, we hope that the content in this section will pique your interest and inspire you to continue learning JavaScript.

2. JAVASCRIPT SYNTAX PREVIEW

JavaScript belongs to a family of programming languages known as **C-like languages**— languages inspired by the C programming language — so you'll find that it shares features with other C-like languages like Python and Ruby. We'll give you a high-level overview of some of these features since you'll likely encounter them again, even if you decide to learn a different programming language.

JavaScript allows you to create and manipulate data like text and numbers, save their values in variables, control the flow of logic in a program with conditionals and loops, organize complex data in data structures, and create reusable portions of code. You might not understand what this means yet, but don't worry we'll step through it together. Here's an example JavaScript program:

```
alert("Hello, world!");
```

In the line above, **alert** is the name of a function that displays alerts to the user. You can use **alert** by adding parentheses after its name. Then, inside the parentheses, we specify what the alert should say — in this case, the alert will contain a message reading **Hello, world!**. The semicolon at the end of the line tells JavaScript that this is the end of the statement.

---

3. VARIABLES AND VALUES

"Hello, world!" is a **value**. Other values include numbers like **100**, values that represent true and false, and even values that are groups of code. Values can be assigned to variables. Variables are created with the keywords **const** or **let**.

```
const greeting = "Hello, world!";
let luckyNum = 100;
```

Variables allow you to reuse values. For example, since we've saved **"Hello, world!"** to the variable **greeting**, we can use the variable with the **alert** function:

```
alert(greeting);
```

Values that are groups of code are called **functions**. The **alert** function is a group of code that is built into the browser and can be used to display alerts to the user. You can also define your own function and save it to a variable so that you can repurpose it later. For example:

```
const sendGreeting = function () {
  alert("Hello, world!");
};
```

This will allow you to send a Hello, world! alert again by writing the function's name followed by a pair of parentheses:

```
sendGreeting();
```

Add code to the *Editable Code* box below to display an alert that says **1 + 2 is 3**. You'll do this by reusing a function stored in the variable **addNums**. You can use the *Click to Run Code* button to execute your code. If you want to start over, click *Reset*. Click *Show Solution* to show the solution.

TRY IT

Follow these steps in the box below:

1. Remove the line that says **// Replace this** and, in its place, write the name of the function, **addNums**.
2. Then add an opening parenthesis **(**.
3. List the two numbers you want to add, separated by a comma. Since you want to add **1** and **2**, you should write **1, 2**.
4. Add the closing parenthesis **)** and end the statement with a semicolon **;**.
5. Press *Click to Run Code* to verify that everything works! You'll see an alert dialog show in your browser if the code is correct.]}

4. CONDITIONALS AND LOOPS

There's also syntax to control the flow of logic throughout a program based on a condition being true or false. For example, the if statement will execute code **if** a condition is true.

(const myNum = 100;

if (myNum > 500) {

  alert("That's a large number.");
}

The code above will send an alert if **myNum** is greater than **500**. Since **myNum** is **100** and **100** is not greater than **500**, **myNum > 500** is false and the user will not receive an alert. We can combine **if** with an **else** statement. An else statement tells the program what to do when the if statement is not true.

```
const myNum = 100;

if (myNum > 500) {
  alert("That's a large number.");
} else {
  alert("myNum is less than 500.");
}
```

Now the user will receive a different alert based on if **myNum** is greater than **500** or not. Since **myNum > 500** is false, the user's alert will say **myNum is less than 500**.

It seems a bit too boring to send an alert that says **myNum is less than 500** when myNum is smaller than 500. It'd be more exciting if we sent an alert when myNum is greater than 500 or when myNum is a negative number.

Edit the code in the *Editable Code* so the user gets an alert if myNum is greater than 500 or if myNum is less than 0. If myNum is less than 0, send an alert that says **Whoa, a negative number!**. Notice there are two conditions to check, myNum>500 and myNum<0.

You will need an **else if** statement instead of an else statement. **else if** statements must go after **if** statements and before any **else** statements (if they exist). We don't need the **else** statement any more, so you'll replace it with an **else if.**

TRY IT

Follow these steps in the box below:

1. Add a space after **else** and type **if**.
2. Add another space and an opening parenthesis **(**.
3. Write the second condition **myNum < 0** after the opening parenthesis.
4. Close everything up with a closing parenthesis **)**.
5. Remember, *Click to Run Code* will run your code, *Reset* will reset the workspace, and *Show Solution* will show you the solution.

Another way to control a program's flow is to use a **while** statement or a **for** statement to repeat a block of code.

```
let count = 0;
while (count < 3) {
  alert(count);
  count = count + 1;
}
```

The example above uses the **while** statement to send an alert that displays the value of the **count** variable. The **while** statement will evaluate the condition — in this case the condition is **count<3** — and execute the code inside the **while** statement if the condition is true. The code above will send three alerts — an alert that says **0**, an alert that says **1**, and an alert that says **2**.

We can also rewrite the code above so it does the exact same thing but with a **for** statement instead of a **while** statement:

```
for (let count = 0; count < 3; count = count + 1) {
  alert(count);
}
```

The example above will also send three alerts — an alert that says **0**, an alert that says **1**, and an alert that says **2** — and follows the same flow of logic as the code in the example **while** statement.

STEP BY STEP

Here's a step-by-step breakdown of how JavaScript will execute the example code with the **while** statement and the one with the for statement:

1. Create a variable named **count** and set its value to **0**.
2. Evaluate count < 3 since it evaluates to true, execute the code inside the statement:
   - Send an alert that says **0**
   - Reassign the value of **count** to **1**
3. Evaluate count < 3 since it evaluates to true, execute the code inside the statement:
   - Send an alert that says **1**
   - Reassign the value of **count** to **2**
4. Evaluate count < 3 since it evaluates to true, execute the code inside the statement:
   - Send an alert that says **2**
   - Reassign the value of **count** to **3**
5. Evaluate count < 3 since it evaluates to false, terminate the loop.

Writing JavaScript for the Web

by Devmountain Tutorials

WHAT'S COVERED

This section will explore how to write JavaScript for the web by discussing:

1. WEB APIs
2. THE DOM
3. THE EVENT SYSTEM

1. WEB APIs

The real power of JavaScript on the web comes from its ability to give web developers access to browser APIs. Earlier, you learned that APIs are pre-built pieces of code that you can repurpose in your own code. Web browsers come with a collection of APIs known as browser APIs. The subset of browser APIs used for writing code for the web are known as Web APIs. There are a lot of **Web APIs**; we won't go over all of them in this section but **Mozilla Developers' Network has a list of all available Web APIs** with links to pages where you can learn more about each one.

This section will focus on the most commonly used Web API — the **Document Object Model (DOM)** — an API that gives you programmatic access to the contents, style, and structure of an HTML page. It's the API used to dynamically change the contents of a web page which allowed developers to program complex behaviors like ones found in web applications.

2. THE DOM

Arguably the most commonly used Web API is the **Document Object Model (DOM)**. The DOM allows you to access and manipulate the structure of an HTML document by representing it as objects in memory, where the objects are accessible via JavaScript. In other words, it's a model of an HTML document where elements are represented as JavaScript **objects**. You can access these objects in order to change the web page's structure, style, or content.

Here's an example of changing a web page's structure. Consider the following HTML:

```
<h1>Hello,</h1>
<p>world!</p>
```

By default, since the **h1** element appears before the **p** element, **Hello**, will appear before **world!**. With JavaScript, we can mess with the natural order of things so **world!** appears before **Hello,**:

The DOM can also be used to generate content. The list items in the example below are generated using JavaScript. The only HTML present on the page is an empty **ul** element.

These examples are a bit contrived though. After all, if someone *really* wanted

```
<p>world!</p>
```

to appear before

Hello,

they could just write their HTML that way to begin with and save themselves the trouble of adding any JavaScript! The real power that programmatic access to an HTML document gives yo
Here's an example that will display a different message depending on your local time. If you're reading this before noon, the paragraph will say **Good morning!**, otherwise, it'll say **Good evening (or afternoon)!**

The DOM also gives you access to the browser's event system, which you can leverage to trigger code based on events like clicking on a button or scrolling past a certain area on the page. We'll cover the event system in more detail next.

TERM TO KNOW

Document Object Model (DOM)
An API that gives programmatic access to the contents, style, and structure of an HTML page. It is used to dynamically change a web page's contents.

Object
In programming, the term object is equivalent with value. For example, the value 100 can be described as a number object.

3. THE EVENT SYSTEM

The browser's event system is based on a programming design pattern called **event-driven programming** where the flow of a program is determined by events like mouse clicks, key presses, or even messages from other programs. The system consists of **event listeners** that listen for certain events and event handlers that execute code when certain events occur.

Using the DOM, you can add an event listener to an HTML element using a special function called **addEventListener**. **addEventListener** needs two inputs — the name of an event and a function containing the code that should be triggered when the event occurs. For example, here's a button that adds items to a list whenever it's clicked:

Pay close attention to the last lines of code:

```
const listAdderButton = document.querySelector('button');
listAdderButton.addEventListener('click', addItemToList);
```

In the code above, **listAdderButton.addEventListener** adds an event listener to the **button** element. **'click'** is the type of event — a mouse click — and **addItemToList** is a variable

whose value is a function that generates an **li** element and adds it to the list.

To add an event listener to an HTML element, you need three values:

- The HTML element itself
- The type of event you want to listen for, as a string
- A function that contains the code you want to execute whenever the event occurs

The code below has a text box and a paragraph that should display the number of words that the user has entered into the text box. Follow the steps to add code to the Editable code area below to add an event listener to the text box so that it'll update the word count whenever the user types into the text box. Remember, *Reset* will reset everything and *Show Solution* will show the solution.

TRY IT

Follow these steps in the box below:

1. Add this line at the bottom to declare a variable and set its value to the text box:**const textbox = document.querySelector('#textbox');**
2. Press the *Enter* key to start a new line. Then, type**textbox.addEventListener** and an opening parenthesis**(**
3. Next, list the two inputs that**addEventListener** needs:
   - Type **keyup**. This event occurs whenever a key has been pressed. Then, add a comma**,** and a space.
   - Type **updateWordCount**, which is the name of the function that updates the word count.
4. Close everything up with a closing parenthesis**)** and a semicolon**;**.

We've covered many concepts in this unit! It probably still feels overwhelming, and that's okay. The more you experiment with code, the easier it will be. If this intrigued you, we encourage you to keep learning. There are many online resources, college courses, or even full bootcamp programs like Devmountain offers. In the next, and final unit, we'll introduce you to some computer engineering concepts that many web developers leverage.

TERMS TO KNOW

Document Object Model (DOM)
An API that gives programmatic access to the contents, style, and structure of an HTML page. It is used to dynamically change a web page's contents.

Object
In programming, the term object is equivalent with value. For example, the value 100 can be described as a number object.

Terms to Know

Attributes

Attributes contain extra information about the element and don't appear in the content of the page.

Boolean attribute

A Boolean attribute is an attribute that can be set to true or false.

Closing tag

Closing tags mark the end of an element. They look the same as opening tags, except they have a forward slash before the name of the element.

Color

Property that changes the color of the text.

Document Object Model (DOM)

An API that gives programmatic access to the contents, style, and structure of an HTML page. It is used to dynamically change a web page's contents.

Empty elements

Empty elements, also known as void elements, consist of a single tag.

External stylesheet

CSS rules in a file separate from your HTML page.

Font weight

Property that allows a choice such as "bold" to stylize text.

Function

Functions are used as a value for various CSS properties.

HTML element

The individual building blocks of HTML. You can create an HTML element with HTML tags. For example, a paragraph element consists of an opening tag, then the content of the paragraph, then its closing tag.

Inline styles

CSS declarations in an element's style attribute. so-called because they appear "in line" with the HTML element being styled. Inline styles only apply to one element at a time.

Internal stylesheet

An internal stylesheet is written directly in your HTML inside a style element.

Nesting elements

When an HTML element exists inside another HTML element, it is said to be nested within the outer HTML element.

Numeric values

Numeric values are usually numbers followed by a unit of measure, for height and width.

Object

In programming, the term object is equivalent with value. For example, the value 100 can be described as a number object.

Opening tag

Opening tags mark the beginning of an HTML element. It consists of the name of the element (for example, p for paragraph), wrapped in angle brackets.

Properties

Properties are identifiers that represent the stylistic features to be modified.

Rules

Rules are the basic building blocks of CSS. Rules are made of selectors and declarations.

Selectors

In CSS, selectors specify the elements or groups of elements that a rule should be applied to. A rule can have one or more selectors.

Semantic HTML

To write semantic HTML means to use HTML correctly by staying true to an element's original, intended purpose.

Style declarations

Style declarations, or simply declarations, are property-value pairs. A rule can have zero or more declarations.

Whitespace

Whitespace is any amount of text that appears as blank space. Some examples of whitespace are space between words in a sentence, space between lines of a song, and indentation before a paragraph.

rgb()

Function that can be used to specify a color by the color's red, green, and blue values.