HTML

Introduction

The Best way to become a programmer is to code!

The only way we're going to learn how to be a better programmer is to write code! Part of being a programmer is to think like a programmer to solve real problems. We will therefore work on real assignments/projects to do and create programs to solve these problems. We want to maximize the time we spend each day writing code.

Learn Actively

Just because a concept seems straightforward when we read it or watch a video doesn't mean it will be that way when we go to apply that knowledge. While the readings are important, they are best utilized as supplemental to the actual doing of assignments. Code along when there are code blocks to start seeing the code in action. Then start working through the assignments. We believe in learning by doing and strength through struggle -- as we actually work through the assignments, the content will make more sense because we'll have a context for all the new terminology and code syntax.

This methodology is probably contrary to the traditional schooling we're used to, but don't be afraid to jump in and make mistakes. That's where the best learning happens!

Don't Compare Yourself to Others

We know that it's human nature and it's something that we all struggle with. Everyone comes from a different programming background and different habits. Focus each day on improving and getting to the next level and building good habits that will lead to us continuing to be successful. If we have to compare ourselves to something: compare ourselves at the end of the day to the start. Learning takes time, we're on a journey and this is only the beginning.

Don't Chase Best Practices (Yet)

While we're on this journey, don't chase best practices until we have the experience to know why they are best or know why best is subjective in many cases. Our time is limited so we must ask ourselves what can we afford to leave out so that we maximize learning the most important topics. Focus on making the code work before making the code perfect. There will always be time later to *refactor* (restructure our code to improve its performance and readability).

Once we're in the industry we'll have ample time to spend working with our new teams and learning the naming conventions and formatting conventions they use.

When you're behind, ask for help:

1. **Classmates** - They're in this together with you, building relationships with them now will pay dividends later when you can partner up on projects, share resources or even network for job opportunities. If a classmate approaches you for help this is a chance to practice what you have already learned and really reinforce your knowledge or clear up something you may have misunderstood in a low pressure environment (you'll want to have all of this figured out before you find yourself in a technical interview).
2. **TAs** - They've been in your shoes and are all accomplished programmers in their own right. They'll bring perspectives and approaches that you may not have seen in the readings or lectures.
3. **Instructors** - They want nothing more than for you to understand the topic at hand and are very good at explaining that topic. Their time can be limited so make sure you ask questions that get to the heart of your misunderstanding and demonstrate what you have already tried.

# Setting Up Our Editor - Visual Studio Code

In order to effectively write code we will need to install software that makes it easier for us. There are many text editors available but we will recommend you use [Visual Studio Code.](https://code.visualstudio.com/) **Visual Studio Code** is lightweight and has a lot of nice features built in like terminals for running code, a file explorer where we can manipulate files and folders, as well as **git** integration.

## Installation steps

Throughout the bootcamp we will frequently need to install programs and set up our Development Environment. This can at times be easy (like installing **Visual Studio Code**) or can at times be tricky, but it is a necessary skill for programmers and something we'll be expected to handle in the industry. The installation instructions in this bootcamp will often be specific to which Operating System we are installing the software on with instructions provided for both **macOS** and **Windows**.

### **macOS Installations**

1. [Download Visual Studio Code](https://go.microsoft.com/fwlink/?LinkID=534106) for macOS.
2. Open the browser's download list and locate the downloaded archive.
3. Select the 'magnifying glass' icon to open the archive in Finder.
4. Drag Visual Studio Code.app to the Applications folder, making it available in the macOS Launchpad.
5. Add VS Code to your Dock by right-clicking on the icon to bring up the context menu and choosing Options, Keep in Dock.

### **Windows Installations**

1. Download the [Visual Studio Code installer](https://go.microsoft.com/fwlink/?LinkID=534107) for Windows.
2. Once it is downloaded, run the installer (VSCodeUserSetup-{version}.exe). This will only take a minute.
3. By default, VS Code is installed under C:\users\{username}\AppData\Local\Programs\Microsoft VS Code.

### **Recap**

Once installed, we recommend considering installing some helpful extensions in Visual Studio Code:

* **Indent-rainbow** - colorful indentation
* **Material icon theme** - clearer looking folder & file icons
* **Open in browser** - right click .html file tab to open in browser

We also recommend creating a new folder somewhere on your computer (maybe our Documents), where we can save all the files we'll be creating during this course.

# How the Web Works

### **Objectives:**

* Gain a high-level understanding of how the internet works
* Learn about the request-response cycle

You probably use the internet every day, but may not really understand exactly what it is and how it works. Simply put, the internet is a large **network of computers that are connected** and can communicate with each other. The web is made up of computers that we call **clients** and **servers**.

* **Clients** are the computers and other internet-connected devices that use the internet.
* **Servers** are computers that store web pages or applications. They are computers, just like the machine you are on right now, but typically without the keyboard, trackpad, or screen.

A group of laptops

Description automatically generated with low confidence

## The Request-Response Cycle

When a **client** wants to access a web page (like Google or Facebook), a copy of the web page is downloaded from the server onto the client machine to be displayed in the user's web browser.

In other words, the client makes a **request** and the server answers back with a **response**.

### **Requests**

As clients, we might make requests by:

* typing a URL in the address bar of our browser,
* clicking on a link,
* submitting a form, or
* refreshing a page.

### **Responses**

The server we are trying to reach then responds with some kind of content. Maybe it's search results, a friend's profile, or video content. In all of these cases, this content is provided to the client as some combination of HTML, CSS, and JavaScript.

Diagram

Description automatically generated

For the duration of this course, we'll just be focusing on the content of a response, particularly creating potential responses that, once we have a server running, we might send to any clients requesting to visit our site.

# Intro to HTML

HyperText Markup Language (HTML) is the main markup language for all web pages. HTML elements are the basic building-blocks of the internet.

### **Objectives**

* Understand the theoretical components of the web and how they interact
* Understand the client-server model
* Understand the most important parts of what is displayed on the browser
* Understand the roles of HTML vs. CSS vs. Javascript
* Understand how to incrementally break down building a web page into its fundamental components

## ****What is HTML****

HTML is the language of the Web. It stands for Hypertext Markup Language.

Its purpose is to let us communicate with the browser the meaning of the content that we wish to place on a web page. To achieve this, HTML defines a number of **tags**that we can wrap the contents with.

* Headings and paragraphs of text
* Images
* Links
* Lists
* Tables
* Forms

HTML has pre-defined tags for each of these, which we will cover in the following tabs.

HTML tags **describe the content they contain.**For example, there are paragraph tags for paragraphs, anchor tags for links, etc. **The browser does not display the HTML tags** but uses the tags to interpret the content of the page.

! tab shortcut generates an html 5 ( which is HTML version 5) document.

## Why HTML and CSS

The HTML and CSS chapters are designed to take you through the theoretical components of the web as well as the first step in creating a web application (building out the client-side). When building a web application we believe it is best to always start with a clickable prototype involving only HTML, CSS, and Javascript so it is only natural to teach these technologies first.

#### ****Create a Simple HTML Document****

* Create a new file in the Text Editor of your choice
* Type "Hello World!" into the file
* Save the file as **hello\_world.html**
* Drag the file into your browser to open it

That's it! You should see the text "Hello World!" displayed in your browser.

Now, you may have noticed that we didn't add any tags to our text. To see how the browser handles this situation, right click in your browser and click **Inspect**. A new module should open at the bottom of your browser with various tabs.

We're currently interested in the **Elements** tab. When you click on it, you should see:

<!DOCTYPE HTML>

<html>

<head>

<meta charset="UTF-8">

<title>Example</title>

</head>

<body>

Hello World!

</body>

</html>

copy

In the case where your HTML document isn't "valid", the browser will try to add appropriate tags to fix it.

One of each of **<html>**, **<head>**, and **<body>** above are required for each valid HTML document.

Some terminology here:

* **Opening and closing tags**

Since tags are supposed to wrap things, **most tags come in pairs**: one opening and one closing tag, that denote the beginning and end of content. Tags without a leading forward slash are called opening tags while tags with leading forward slashes are called closing tags.

In the above example, <body> is an opening tag - it's equivalent to you telling the browser "Hey, I'm going to start putting in body content now". </body> is a closing tag that means "OK, I'm done with body content."

* **Nesting**

Notice that between the opening and closing HTML tags, we also have head and body tags.

**Tags can encapsulate other tags**. This is called nesting. Nested items are **indented with a tab** to make the document easier to read.

The hierarchy that rises from **nesting** is called **DOM** - Document Object Model.

* **Commenting**

The format for commenting in an HTML file is as follows:

<p> Some paragraph content. </p>

<!-- This is a comment -->

copy

## Quick Start

To quickly get started editing an HTML document, we will first want to make sure we have created an HTML file in **Visual Studio Code**. We can do this by naming the file with a "dot html" at the end along the lines of index.html.

Next we can generate the boilerplate HTML code using a handy shortcut ! followed by TAB.

We can see this in action below

# Text Elements

### **Objectives:**

* Learn how HTML affects what we see on a page
* Recognize the syntax of HTML elements

HyperText Markup Language (HTML) is the main markup language for all web pages. Its purpose is to communicate to the browser, and ultimately the client, the meaning of the content that we wish to place on a web page. Even as you look at this page, there are lots of different pieces: the navigation pane on the left, the breadcrumb navigation above, a search bar, headers, regular text, lists, and more! HTML is the language we use to create this structure.

### **Syntax**

Every HTML web page is composed of **elements**, which specify different types of content. The syntax for indicating the type of element is with **tags**. A section of content is wrapped in a tag, where the content begins with an **opening tag** (<element\_name>) and ends with a **closing tag** (</element\_name>). (There are a couple of exceptions--a handful of tags don't have closing tags.)

HTML has a finite, pre-defined list of tags that browsers will recognize. **The browser does not display the HTML tags**, but uses the tags to interpret the content of the page. HTML tags **describe the content they contain.** For example, there are paragraph tags for paragraphs, header tags for headers, etc.

## Common Elements

Here are some common elements. On the left is how we'd write them in code, and on the right is how that code would be displayed in the browser!

**Headers**: <h1> through <h6>

<h1>Most important</h1>

<h2>h2</h2>

<h3>h3</h3>

<h4>h4</h4>

<h5>h5</h5>

<h6>least important header</h6>copy

# Most important

## h2

### **h3**

#### h4

##### **h5**

###### **Least important header**

**Paragraphs of text** : <p>

<p>This is a paragraph.</p>

<p>This is a second paragraph.</p>copy

This is a paragraph.

This is a second paragraph.

**Hyperlinks (anchor tags)**: <a>

Important attribute: href (the url to route to)

<a href="http://learn.codingdojo.com">Learn Platform</a>

[Learn Platform](http://login.codingdojo.com/)

**Emphasis and Strong:**<em>**and**<strong>

<strong> and <em> are often used inside of paragraphs <p>

<p><em>This</em> is an <strong>example!</strong></p>copy

This is an **example!**

### **Don't forget - test the code as we go!**

# Lists and Tables

## Objectives:

* Recognize the syntax of more HTML elements
* Learn about Parent, Child, Sibling relationships among HTML elements

HyperText Markup Language (HTML) is the main markup language for all web pages. Its purpose is to communicate to the browser, and ultimately the client, the meaning of the content that we wish to place on a web page. Even as you look at this page, there are lots of different pieces: the navigation pane on the left, the breadcrumb navigation above, a search bar, headers, regular text, lists, and more! HTML is the language we use to create this structure.

### **Syntax**

Every HTML web page is composed of **elements**, which specify different types of content. The syntax for indicating the type of element is with **tags**. A section of content is wrapped in a tag, where the content begins with an **opening tag** (<element\_name>) and ends with a **closing tag** (</element\_name>). (There are a couple of exceptions--a handful of tags don't have closing tags.)

## Lists and Tables

Here are some more elements. On the left is how we'd write them in code, and on the right is how that code would be displayed in the browser!

**Unordered (bullet) lists**: <ul> with nested <li> that contain the list items

<ul>

<li>Dog</li>

<li>Cat</li>

<li>Dragon</li>

<li>Monkey</li>

</ul>copy

* Dog
* Cat
* Dragon
* Monkey

**Ordered (numbered) lists**: <ol> with nested <li> that contain the list items

<ol>

<li>Dog</li>

<li>Cat</li>

<li>Dragon</li>

<li>Monkey</li>

</ol>copy

1. Dog
2. Cat
3. Dragon
4. Monkey

**Tables**: <table> with nested <tr> (table rows) that contain nested <th> or <td> for header or regular data cells, respectively

<table>

<tr>

<th>Movie</th>

<th>Year</th>

<th>Rating</th>

</tr>

<tr>

<td>Up</td>

<td>2009</td>

<td>PG</td>

</tr>

<tr>

<td>The Matrix</td>

<td>1999</td>

<td>R</td>

</tr>

</table>copy

|  |  |  |
| --- | --- | --- |
| **Movie** | **Year** | **Rating** |
| Up | 2009 | PG |
| The Matrix | 1999 | R |

### **Parent, Child, Sibling**

In the code examples above, tags like <li> or <td> are always used inside of other tags like <ul> or <tr>. When an HTML tag is nested inside of another tag, we call the inside tag a child of the outer tag. Adjacent child tags like the multiple <li> are considered siblings of each other.

# Images and Videos

## Objectives:

* Learn how use img and video elements
* Learn how to make video autoplay in modern browsers

HyperText Markup Language (HTML) is the main markup language for all web pages. Its purpose is to communicate to the browser, and ultimately the client, the meaning of the content that we wish to place on a web page. Even as you look at this page, there are lots of different pieces: the navigation pane on the left, the breadcrumb navigation above, a search bar, headers, regular text, lists, and more! HTML is the language we use to create this structure.

### **Syntax**

Every HTML web page is composed of **elements**, which specify different types of content. The syntax for indicating the type of element is with **tags**. A section of content is wrapped in a tag, where the content begins with an **opening tag** (<element\_name>) and ends with a **closing tag** (</element\_name>). (There are a couple of exceptions--a handful of tags don't have closing tags.)

HTML has a finite, pre-defined list of tags that browsers will recognize. **The browser does not display the HTML tags**, but uses the tags to interpret the content of the page. HTML tags **describe the content they contain.** For example, there are paragraph tags for paragraphs, header tags for headers, etc.

## More Elements

Here are some more elements. On the left is how we'd write them in code, and on the right is how that code would be displayed in the browser!

**Images**: <img>

Important attributes: src (the image file path) and alt

<img src="panda.jpg" alt="panda pic">copy

 (if not found; otherwise, image will be shown)

**Videos**: <video>

Important attributes: src (the video file path) and controls

<video src="monkey2.mp4" controls></video>copy

**More about videos**

If we want a video to autoplay on modern browsers, we need to include autoplay, loop, and muted.

Text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

# Why Indentation?

# A close-up of a lizard Description automatically generated with medium confidence

In this chapter, we will be covering 2 very important concepts: proper  **Indentation** and the**Parent, Child, Sibling Relationship**.

Consider the following example:

**Lorem Ipsum Dolor Sit?***Lorem ipsum dolor sit amet consectetur adipisicing elit. Sed, ipsam placeat earum fugit sapiente unde blanditiis porro recusandae dolor! Dolorem quisquam deleniti earum aspernatur voluptas fugit dignissimos doloremque pariatur ipsum! Animi quo molestiae omnis iure suscipit cum, autem, illum debitis voluptates dicta alias mollitia provident est perspiciatis minima aspernatur ratione tenetur, neque ab accusamus quis.*

Notice how the header is centered in the middle of the page. Also notice that the first word at the beginning of the paragraph is indented a few spaces over. As programmers, we also want our co-programmers to distinguish where our

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

<h3>Lorem Ipsum Dolor Sit?</h3>

<p> Lorem ipsum dolor sit amet consectetur adipisicing elit. Sed, ipsam placeat earum fugit sapiente unde blanditiis porro recusandae dolor! Dolorem quisquam deleniti earum aspernatur voluptas fugit dignissimos doloremque pariatur ipsum! Animi quo molestiae omnis iure suscipit cum, autem, illum debitis voluptates dicta alias mollitia provident est perspiciatis minima aspernatur ratione tenetur, neque ab accusamus quis.</p>

</body>

</html>

copy

Wait. Isn’t it supposed to be indented like below?

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

<h3>Lorem Ipsum Dolor Sit?</h3>

<p> Lorem ipsum dolor sit amet consectetur adipisicing elit. Sed, ipsam placeat

earum fugit sapiente unde blanditiis porro recusandae dolor! Dolorem quisquam deleniti earum

aspernatur voluptas fugit dignissimos doloremque pariatur ipsum! Animi quo molestiae omnis iure

suscipit cum, autem, illum debitis voluptates dicta alias mollitia provident est perspiciatis

minima aspernatur ratione tenetur, neque ab accusamus quis.</p>

</body>

</html>

copy

Simple answer, **NO.**

A picture containing linedrawing

Description automatically generated

Remember, we don't indent our code by our website's  content, we indent them via the HTML tag's **parent, child, sibling relationship**. So how did we get the title centered? Good question! That'll require some CSS and we'll be learning all about positioning tomorrow. For now, just understand that indentation is determined by the PCS relationship, not by how we want the HTML to be visually rendered on the browser.

## What is the HTML PCS Relationship?

The PCS relationship is just a term to describe the relationship between tags/elements inside an HTML document. To know the relationship between tags, we must first determine the parents, children, and siblings within our HTML document.

Example:

<!DOCTYPE html>

<html>

<head>

<title> Hello World! </title>

</head>

<body>

<h1> Here is a heading tag </h1>

<p> Now a paragraph tag </p>

</body>

</html>

copy

From the HTML above you can read:

* <html> is the root elementA person in a garment

  Description automatically generated with medium confidence
* <html> has no parents
* <html> is the parent of <head> and <body>
* <head> is the first child of <html>
* <body> is the last child of <html>

and:

* <head> has one child: <title>
* <title> has one content (text): "Hello World!"
* <body> has two children: <h1> and <p>
* <h1> has one content (text): "Here is a heading tag"
* <p> has one content (text): "Now a paragraph tag"
* <h1> and <p> are siblings
* <head> and <body> are siblings

## Is it really required to indent HTML code?

It's not required, but it is highly recommended, and here are the reasons why:

* Beautiful-looking code can make you happy, and one way to make them look beautiful is to properly indent them.
* Properly indented code leads to less time spent on troubleshooting. Imagine being stuck with a problem, and you curse the inventor of HTML (damn you, Tim Berners-Lee!) -- only to find out that you just missed a closing tag. You were not able to quickly see it because you did not properly indent!
* It's easy to browse through your own code. With a glance, you know right away which part is the header part, and which part is the footer.
* It's a good coding habit. Popular programming languages such as PHP, Java, or Ruby recommend proper code indentation. Python even enforces it.

### **How to Indent**

* Just tap the TAB key. 1 TAB is equal to 4 spaces.
* For a quick demo, please watch the video below:

At this point, you may not appreciate the significance of HTML PCSrelationship -- and that's okay. Just know that as early as the HTML course, you will be dealing with them (through properly indenting your code), but will not be directly working on them (target, and manipulate related elements via CSS and or JS).

# Forms

### **Objectives:**

* Learn which elements to use inside of Forms

Forms allow our users to interact with our website in a more meaningful way. They can tell themselves about themselves, give us useful feedback, or even create their own stories and upload meaningful images and videos using forms. As such, it's important that we are familiar with a wide variety of different input types. But don't get too focused on learning all of them now; it's enough that right now we are aware these exist.

## Common Inputs

Here are some common elements used inside of forms! You can see the code on the left and test the inputs on the right.

**Text Inputs**

<input type="text" name="first\_name">copy

**Number Inputs**

<input type="number" name="age">copy

**Passwords**

<input type="password" name="password">copy

**Date**

The date input isn't supported on [*all browsers.*](https://caniuse.com/input-datetime)

<input type="date" name="dob">copy

**Color**

<input type="color" name="text\_color">copy

**Radio**

Including the "id" and "for" attributes allow for clicking the label text to change the inputs.

<input type="radio" name="font" value="bold" id="bold">

<label for="bold"><strong>Bold</strong><label>

<input type="radio" name="font" value="normal" id="normal">

<label for="normal">Normal<label>copy

**Bold**  Normal

**Checkbox**

<input type="checkbox" name="accept" id="accept">

<label for="accept">I agree<label>copy

 I agree

**Textarea**

<textarea name="comment" cols="20" rows="3"></textarea>copy

**Select**

<select name="snack">

<option>Almonds</option>

<option>Cheese and Crackers</option>

<option>Pita and Hummus</option>

<option>Pears</option>

</select>copy

AlmondsCheese and CrackersPita and HummusPears

## A Typical Form

Forms typically have some more tags than just the inputs. Commonly they are used with label, div as well as a button (<input type="submit" value="Button Text" >) to submit the form. We don't need to worry too much about the action or the method yet. They will become important once we start working with a back-end server.

**Common form**

<form action="/process" method="post">

<div>

<label>Name: </label>

<input type="text" name="name"

placeholder="Your name..." >

</div>

<div>

<label>Email: </label>

<input type="text" name="email"

placeholder="Your email..." >

</div>

<div>

<input type="checkbox" name="offers"

id="offers" checked >

<label for="offers">I would like

to receive email offers!</label>

</div>

<input type="submit" value="Sign Up!" >

</form>copy

Name:

Email:

  I would like to receive email offers!

In the previous chapter we may have seen the following code and wondered what the heck are those div tags.

As seen in the last module...

**Common form**

<form action="/process" method="post">

<div>

<label>Name: </label>

<input type="text" name="name"

placeholder="Your name..." >

</div>

<div>

<label>Email: </label>

<input type="text" name="email"

placeholder="Your email..." >

</div>

<div>

<input type="checkbox" name="offers"

id="offers" checked >

<label for="offers">I would like

to receive email offers!</label>

</div>

<input type="submit" value="Sign Up!" >

</form>copy

Name:

Email:

  I would like to receive email offers!

The <div></div> element exists for us to create a division in our code where we can group tags that have a common purpose together. In the example above we do this so that the <label> and <input> tags are lined up next to each other. As we learn more about **HTML** we'll see many cases where using a div will let us define boxes of content or (with enough styling) they can be made into any shape we can think of.

Validating Our HTML

Important Tips To Avoid Headaches

HTML is a language that allows poor code to execute and render to varying levels of accuracy. Successful rendering, however, does not mean that our code is correct or guarantee that it will validate as standards compliant. Poor code is unpredictable, and you can't be certain what you're going to get when it renders. We have to pay close attention when writing HTML and be sure to nest/indent and close all elements correctly and to always validate our code.

Use Proper Document Structure

Pages will render without the use of the <!DOCTYPE html> doctype or <html>, <head>, and <body> elements. However, without the doctype and these structural elements, pages will not render properly in every browser.

Bad Code

<html>

<h1> Hello World! </h1>

<p> This is my first website </p>

</html>

copy

Good Code

<!DOCTYPE html>

<html>

<head>

<title> My First Website </title>

</head>

<body>

<h1> Hello World! </h1>

<p> This is my first website </p>

</body>

</html>

copy

Constantly Validate Your Code!

While writing HTML, make a habit to validate frequently; this will save you from issues that are harder to pinpoint (or redo) once your work is completed and lengthier.

HTML validation services such as the free  [W3C Markup Validation Service](http://validator.w3.org/) are useful debuggers that help you identify rendering errors.

Organize HTML Syntax

As your HTML gets bigger, managing it can become quite a task. Below are quick rules that can help you keep your syntax clean and organized:

* Use lowercase letters within element names, attributes, and values.
* Indent nested elements.
* Use double quotes, not single or completely omitted quotes to store in values for HTML attributes. Good example:

<h1 id="page\_title">My First Website!</h1>

<p class="sub\_title">This is <span class="emphasize">Cool!</span></p>copy

Avoid Using Too Many *div*s!

When writing HTML, it is easy to get carried away adding <div> elements here and there to build out necessary styles. While this works, it can add quite a bit of bloat to a page, and before too long we're not sure what each <div> element does.

Bad Code

<div class="container">

<div class="article">

<div class="headline"> HTML Rocks! </div>

</div>

</div>

copy

Good Code

<div class="container">

<article>

<h1>HTML Rocks!</h1>

</article>

</div>

copy

Make Use of Semantic Elements

Deciding which elements to use to describe different content may be difficult, but these elements are the backbone of semantics.

*Note: Semantic HTML is the use of HTML markup to reinforce the semantics, or****meaning****, of the information in webpages rather than merely to define its presentation or look.  
via:*[en.wikipedia.org/wiki/Semantic\_HTML](https://en.wikipedia.org/wiki/Semantic_HTML)

Here the HTML doesn't use the proper heading and paragraph elements; instead, it uses meaningless elements to group content.

Bad Code

<span class="heading"><strong>Welcome Back</span></strong>

<br><br>

It has been a while. What have you been up to lately?

<br><br><br>

copy

Good Code

<h1>Welcome Back</h1>

<p> It has been a while. What have you been up to lately?</p>

copy

Keep Your Tag Names Lowercase

Technically, you can get away with capitalizing your tag names.

<DIV>

<P> I'm out of random stuff! </P>

</DIV>

copy

Having said that, the best practice is to keep all tags lowercase.

Use *alt* Attribute With Images

Using meaningful *alt*attributes with <img> elements is a must for writing valid and semantic code. The alt information is helpful for when a user cannot view your image - whether due to a connection issue, a missing image, or because the user is utilizing a screen reader.

Bad Code

<img id="logo" src="images/logo.png" alt="logo.png">

copy

Good Code

<img id="logo" src="images/logo.png" alt="My First Website's logo">