YOU CAN MAKE A BAD WEBSITE!

LESSON ONE: WHAT MAKES A WEBSITE WORK?

A Shitty Guide by Zenith

(It's not not that quick or easy but it can surely be free!)

DISCLAIMER

I am a hobbyist webmaster and I don't know JavaScript. So this series of documents will solely deal with HTML and CSS capabilities. This isn't intended to be used as if it's a master guide. As the title implies, I strongly believe in "you can make it better later, but first you have to make it exist." My own coding style is pretty sloppy, and I frequently have to access online guides for help while working. This is just as much a refresher and practice for me as it may be for you.

Everything is difficult until it is easy. When first learning HTML and CSS I was confused all the time and easily discouraged by others having websites that I found prettier than my own. Always remember that the shittiest indie website has more heart in it than the best corporate owned platform. It's okay to ask friends for help, to look at tutorials, and most importantly, to accept you fucked something up beyond repair and start over. The good thing about being alive is that the point of it is to learn. Don't give up, always take breaks, and remember it's only text and there is no grade or money on the line.

TABLE OF CONTENTS

OKAY SO WHAT THE FUCK ARE HTML AND CSS?

HTML and CSS are the languages that you'll be coding in. Every website you use is built using these (and then probably JavaScript. But I don't know how to do all that, so don't worry about it.)

If you imagine a website like a house, HTML is the wall and CSS is the paintings you hang on it: HTML builds all the elements that CSS will style. This is reflected in their names:

HTML stands for Hypertext Markup Language

CSS stands for Cascading Style Sheets

In order for a website to work, it needs HTML in order to structure the individual pieces of your website. CSS is not required, but using it will make your website much prettier and easier to read. First, let's go over some HTML tags and explain what they mean and do.

To create an HTML or CSS document, you simply need to create a text file with the end-of-file marker .html or .css.

WHY IS EVERYTHING IN BRACKETS?: HTML TAGS

When you look at an HTML document, you'll notice that there are several tags written in angle (< >) brackets. These are the commands you're giving the browser so it can display your content. It is vitally important to make sure you always close them. I have spent countless hours pouring over my code and rewriting line after line following a breakage, only to then discover I simply forgot to close an HTML tag and subsequently confused the browser, leaving it unable to show anything (or show everything, but completely incorrectly.) I won't be covering every HTML tag ever (as that would be pointless) but I will give you a rundown of commonly used tags.

At the top of every HTML document you'll be able to make following my advice, you will have to type up something similar to the following:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8"/>
<link rel="stylesheet" href="/style.css">
<title>YOUR PAGE TITLE</title>
</head>
```

Let's break down what each of these lines are doing, and why you need them.

<!DOCTYPE html>	This line tells the browser that the document you are submitting to it is an HTML document and to read the text on it as HTML commands.
<html lang="en">	What language you are writing your page? Mine is "en" because I am using English.
<head>	This is the opening tag that tells your browser that the following information shouldn't be visible on the webpage itself. It is information for the browser to use to correctly format the following HTML and CSS. This information is called **metadata**.
<meta charset="UTF-8"/>	This specifies what characters you are typing in. UTF-8 is the preferred character set and includes every current Unicode character
<link rel="stylesheet" href="/style.css">	This tells your page to access a CSS document to apply styling rules to your HTML elements.

<title>YOUR PAGE TITLE</title>	These tags allow the text written between them to display as the text written at the top of the tab. If you've ever noticed the name of your tab switch to match the page you've clicked on, it was through the use of this tag.
</head>	This closes the <head> tag we started with. It tells the browser that everything after this will be displayed on the page itself and should be visible.

Now that we've gone over the tags you'll use in the head of an HTML document, let's review some of the ones used in the body of one. Here is an example of what an HTML document with no CSS styling could have in its body.

```
<body>
<center>
<a href="/index.html"><img src="IMAGE.PNG" alt="IMAGE DESCRIPTION"></a>
<h1>THIS IS THE TITLE OF MY PAGE!</h1></center>
<p>And this is the text on it!</p>
</body>
```

Like before, let's go over what these tags mean.

<body>	This tag tells the browser that everything after it is to be visually displayed on the page.
	As the name implies, this centers anything in it!
...	<a> is what defines a hyperlink in HTML. We need the **href** attribute in order to make the link usable. The text after the equals sign, **"/index.html"**, refers to the homepage of your website. AKA, the first page you see when you open your site. In this example, we're making an image clickable as a link to lead to your homepage.
<img src="IMAGE.PNG" alt="IMAGE DESCRIPTION">	This tag defines images. The **src** attribute allows you to type the link to your image after the equals sign. The **alt** attribute allows you to provide alt text for screen readers to read to describe the image to users relying on them. <img> tags are self contained, they don't have a closing tag, Everything is typed within one set of angle brackets.
THIS IS THE TITLE OF MY PAGE!	<h1> defines heading text. These tags range from h1 to h6, with h1 being the largest text and h6 being the smallest.
	This closing tag lets the browser know to stop centering everything.

And this is the text on it!	defines paragraph text. Because it is outside the center bracket, it will display on the left side of the browser window as there is no styling applied to it.
</body>	This ends the visible content on a page. Things written after this, such as scripts, will not be visible but impact the website's function in other ways.

While the examples given cover a lot of ground for HTML tags, I'll explain some more commonly used ones.

…	Makes text bold.
…	Makes text italicized.
<u>…</u>	Underlines text.
<blockquote>… </blockquote>	Makes text into a blockquote. This is intended to be used to show that something is being quoted from another source, however, I frequently use it to "indent" sections of text.
 	Indicates a line break in text. HTML won't read hitting the "enter" key on a keyboard as a line break, so you need to use any time you want one.
<audio>	Defines an audio element.
	Defines a section of a web page. elements will typically be styled with CSS.
<header>…</header>	Defines a section of a website as a header.
<footer>…</footer>	Defines a section of a website as a footer.
<style>…</style>	You can use <style> tags to define the CSS of a document. **I strongly recommend not doing this unless absolutely necessary as it can make your document more confusing to read after the fact.** <style> can also be used to modify HTML elements that would ordinarily be styled by your CSS document to have different properties than what is standard. I sometimes do this, but try to keep it to a minimum to keep my documents readable.
<script>…</script>	Defines something as a piece of script for the document to execute. While I don't know JavaScript, my website includes some JS elements from free resources online and therefore uses the <script> tags.

Of course, this list does not cover all HTML tags as previously stated, but by using the example tags I've given, you can create a lot of cool web pages when using CSS to style them.

NOW GIVE IT COLORS: ABSOLUTE ESSENTIALS OF CSS

Like with HTML, there are many elements of CSS and I cannot describe them all in this document. In some ways, CSS can seem much more confusing while also being more straightforward than HTML. For the purposes of this document, I will explain the basics of changing the color of elements, the size of fonts and images, and offer a brief overview of position and flex boxes. The final two functions will **NOT** be explained in depth in this document, merely defined in order for you to become more familiar with these terms. **Don't worry about them right now!**

Unlike HTML, CSS does not use angle brackets nor HTML's tag system. CSS operates by specifying a **selector** and then writing **declarations**. The selector of a CSS rule is an element of HTML that you want to style. In order for you HTML document to follow your CSS rules, you must link the two together by designating a .css file as your **stylesheet**, or use <style> tags in the <head> of a document. Below, I will write a CSS rule and describe what each part of it is doing.

```
p { color: red;
font-size: 1.5em; }
```

In this example, we are styling the element <p>, the paragraph text. We are designating its font color as red and it's size as 1.5em. As you can see, in order to designate the selector, we have written the name of its tag without angle brackets, followed by a curly bracket containing declarations. Notice that after each declaration there is a semicolon. This is necessary to end each declaration in order for the document to read and understand the next declaration to the selector. Because CSS can be applied to any HTML element, I will focus mostly on the following for this intro tutorial:

1. Very beginner styling (This will not include the use of flex boxes or float properties.)
2. How to change the font of your page through local files
3. A basic definition of **flexboxes** and **position declarations**. These topics will be discussed more in-depth in following lessons.

Now, let's look at a snippet of CSS that is styling a with the class "container." I will explain what each line means in text highlighted in blue. The CSS itself will be highlighted in yellow.

BEGINNER <DIV> STYLING

.container	We are using a period to tell the document that this will be the class of a

{ width: 500px; The width of the will be 500px. This means that when text exceeds it will go to another line.

height: auto; This means the 's height will adjust for the amount of content held in it.

margin: auto; The margin of the page determines the amount of space surrounding the outside of the . "Auto" allows the browser to automatically calculate the margin of the on the page.

padding: 5px; Padding determines the amount of space between the content of the and the borders of the itself. We're allowing it to generate 5px between the content and border.

overflow: auto; Overflow determines what will happen to elements placed inside the that are too large for the specified width and height. Using "auto" means that if an element is bigger than the its contained in, the browser will create a scroll bar to allow you to scroll and see more of the element. However, all elements fit in the , no scroll bar will be shown.}

There are many other ways you may want to style a . A good collection of how to use various CSS elements can be found at W3Schools. Now, let's look at how CSS works to change the font of an element. CSS styling will be highlighted in yellow. Explanations of what each line does will be highlighted in blue.

USING LOCAL FONTS

@font-face { Specifies a rule to display a custom font.

font-family; MyFont What you will call your font when specifying which text elements should use it.

src:url (/Fonts/MyFont.otf);} The file for you font. You can use TrueType (.ttf), OpenFont (.otf), Web Open Font Format (woff), and Web Open Format 2 (woff2), files to do this. In this example, we are using an OpenFont file.

h1 { Specifying these rules are for heading text.

color: #6F8FAF; Turns the color of the text to the color that displays the hex code #6F8FAF (a shade of blue).

font-size: 2em; Specifies the size of the font.

font-family: MyFont;} Specifies the font itself using the name for the file you designated for it.

BRIEF FLEXBOX AND POSITION DEFINITIONS

Because these CSS functions are commonly seen in examples of code, I will be giving brief definitions of what they do. I will provide a more in-depth breakdown of their functions in a later lesson. This is intended as a truly brief overview in order to make understanding online posts more doable.

Flexboxes are created by styling a with the declaration “display: flex;”. A flexbox is a way to lay out your s and other elements. It is one-dimensional, meaning it only controls the layout of a column or a row, but not both at the same time. The direction a flexbox modifies can be specified using further CSS declarations. When you style a to have the flex property, the s placed inside it will place themselves in a line. It is also possible to cause flexboxes to wrap elements inside them when they are too large to display on a single line.

The position declaration declares… the position of an element. Below I will create a table listing some position declarations and basic summaries of what that will do to an element. This will only include positions which I feel are the most commonly used by hobby webmasters. It is important to note that these positions determine how more of how an element moves with the scrolling of a page, rather than it’s position horizontally or vertically. There are the **top, bottom, left, and right** declarations to do that.

static	The element moves normally when scrolling on the page. A static element cannot have further styling to its **left, right, top or bottom** position.
sticky	The element moves normally when scrolling until its parent element is no longer visible on the screen.
absolute	Unlike relative, when an absolute element is positioned using **left, right, top or bottom**, the space it would have taken up is ignored by the elements which come after it, meaning it can have elements overflown onto its space. If its parent container is not set to relative, it will determine its position based on the top of the HTML container.
relative	Works similarly to static, but with the ability to determine it’s position with the **left, right, top, and bottom** declarations. Relative elements will overflow static elements, displaying above them. This is because when a relative element’s position is styled, the space where it is offset is made static.
fixed	The element will stay on screen in the designated position regardless of scrolling. It always considers the HTML container itself its parent.

KEEPING IT TOGETHER: AN ORGANIZATION LESSON

So you've written all your code but you took a two week break from updating it and now you have no idea what any of it means and does. A rookie mistake, and one I make often. Here are some ways you can organize and notate your work so it's less of a nightmare to read back and update.

COMMENTS

HTML and CSS both have unique tags for leaving comments on their function. **You should use them often**! HTML and CSS comments, when written correctly, will not appear on the page itself or impact how it looks in any way. They are purely for **your benefit**!

<!-- Your comment here! -->	An HTML comment. All comment text is kept within the brackets and their opening and closing notation (the "--" punctuation.)
/* Your comment here! */	A CSS comment. All comment text is kept within the slashes and their opening and closing notation (the "/*" punctuation.)

FOLDERS

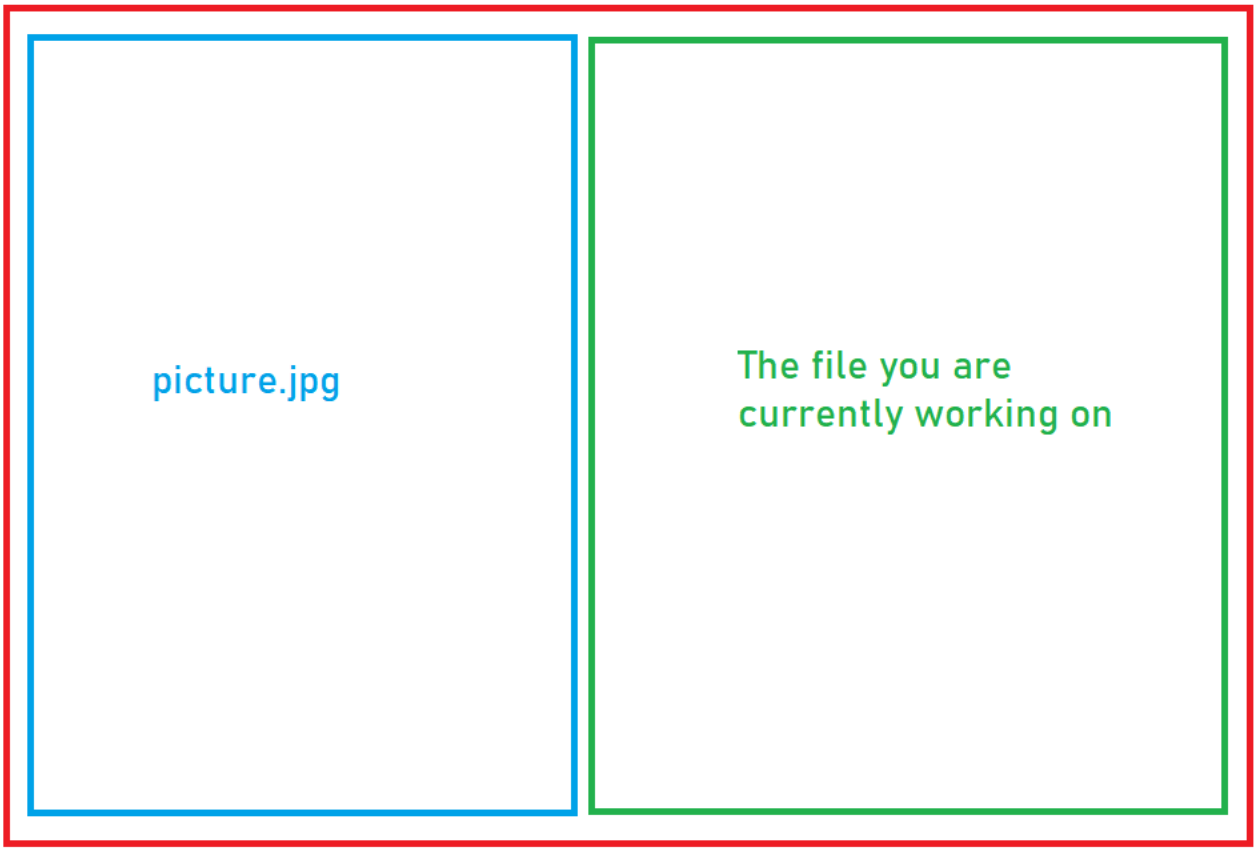
To return to the metaphor as HTML and CSS as houses, folders can be considered different wings of a home, with each HTML document in it being a room. There are many methods you can use when using folders for your website, but I prefer using them to break down **categories of pages** on my site and to keep all **site-wide documents and assets together**.

While I will be talking about the places where you can write code later in this document, a good standard practice no matter where you're writing it is to create a single folder to house all things related to the website and web page(s) you'll be coding. Within that folder you can create subfolders to organize your documents.

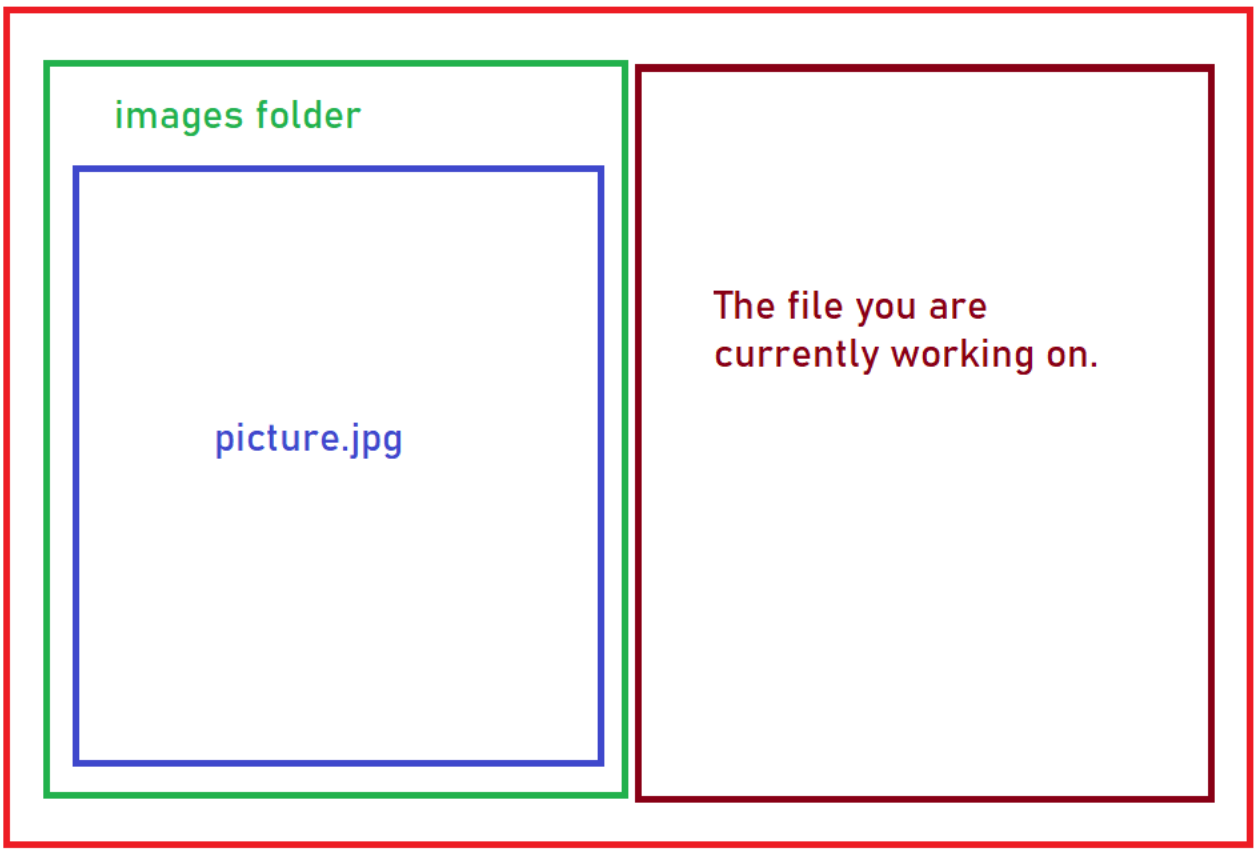
When using folders, it's crucial to be aware of **file paths**. File paths describe the location of a file in your documents and folders. You can think of file paths like Russian nesting dolls. When there is a folder inside another folder, you still have to look through each one to find it, just like you have to open every large doll in order to see the smaller ones. Let's look at some different ways file paths can look and explain where they're finding the file they link to. This explanation will contain both a **written explanation in a table**, a **drawn structural explanation**, and **examples of what code containing folders and their file paths could look like**. In all of our examples we will be trying to link to an image for the sake of continuity and ease of understanding.

<img src="picture.jpg">	“picture.jpg” is in the same folder as the file you are working on.
<img src="images/picture.jpg">	“picture.jpg” is in a folder titled images in the same folder as the file you are working on.
<img src="/images/picture.jpg">	“picture.jpg” is in a folder titled images in a folder outside of the folder of the file you are working on.
<img src= ”https://image.com/picture.jpg”>	“picture.jpg” is hosted on another website and you are hotlinking to it.

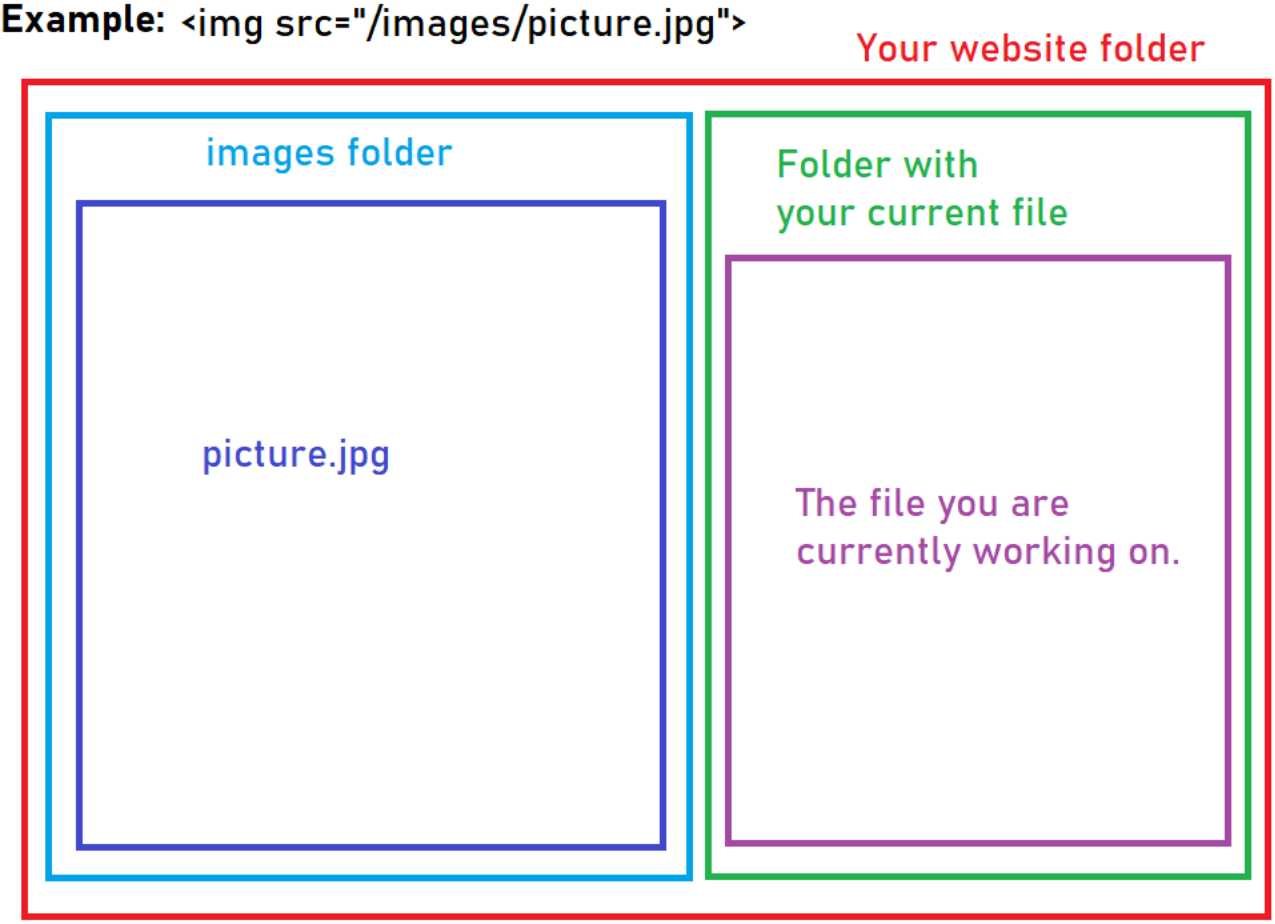
Example: <img src="picture.jpg">



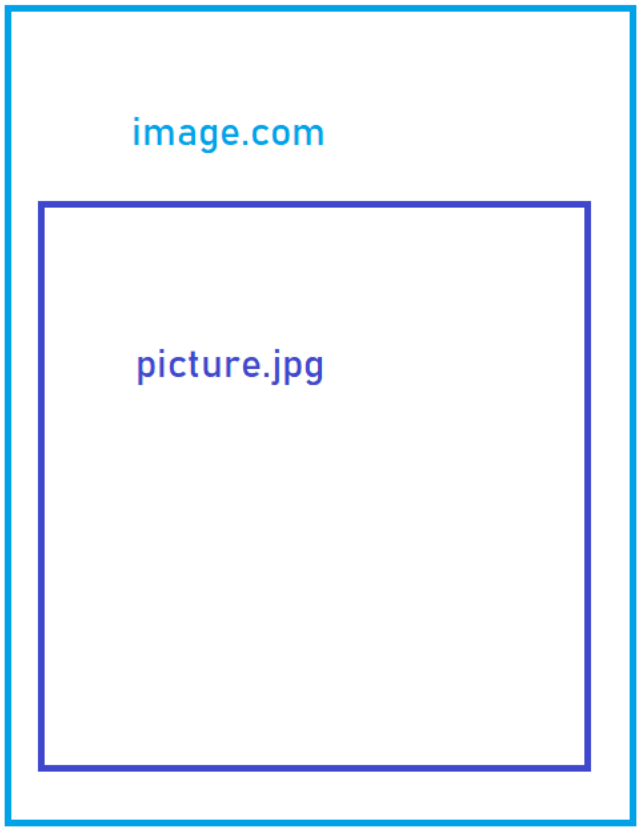
Example: <img src="images/picture.jpg">



Example: <img src="/images/picture.jpg">



Example: <img src= "https://image.com/picture.jpg">



The following are two examples of what someone using files in their document could code. I will break down what each line of code is doing. Code will be highlighted in yellow. Explanations of each line will be highlighted in blue.

Example 1:

<!DOCTYPE html> Defines the document as an HTML document.

<html lang="en"> Defines the language of the document as English.

<head> Contains text not seen on the page itself; the metadata of the page.

<meta charset="UTF-8"/> Defines the characters being used on the document.

<link rel="stylesheet" href="/style.css"> Links to a CSS document to style the page.

<title>YOUR PAGE TITLE</title> The title shown on the tab containing the page.

</head> Metadata ends, body elements will be visible.

<body> Body elements begin.

I AM TYPING ABOUT picture.jpg! CLICK picture.jp FOR MORE INFORMATION! Paragraph text talking about image.jpg.

 A link to image.jpg.

<img src=”/images/picture.jpg> The file for image.jpg is in a folder outside of the folder your current file is in. image.jpg is clickable.

 The link ends.

</body> The body elements end.

Example 2:

<!DOCTYPE html> Defines the document as an HTML document.

<html lang="en"> Defines the language of the document as English.

<head> Contains text not seen on the page itself; the metadata of the page.

<meta charset="UTF-8"/> Defines the characters being used on the document.

<link rel="stylesheet" href="/style.css"> Links to a CSS document to style the page.

<title>YOUR PAGE TITLE</title> The title shown on the tab containing the page.

</head> Metadata ends, body elements will be visible.

<body>	Body elements begin.
LOOK AT image.png!	Header text talking about image.jpg.
I AM TYPING ABOUT picture.jpg!	Paragraph text talking about image.jpg
<img src="https://image.com/picture.jpg" alt="Description of picture.jpg">	The file for image.jpg is on a different website than the one you are making. image.jpg is completely contained somewhere. It is being described in alt text so a screen reader can read it aloud.
</body>	The body elements end.

While there are many more ways to link things than just the two examples provided, hopefully they provide enough information in tandem with previous examples to allow you to make educated guesses about how to link something on your site.

NAMING YOUR <DIV>S BETTER (ADVICE I AM BAD AT TAKING)

While creating classes for your various s with CSS, you are able to give them each unique names. You should use this to your advantage by naming them something related to their purpose on the site. If you are repeatedly using the same on multiple pages and want to give it a shorter name for ease of typing, make sure to use comments to describe any shorthand that isn't immediately understandable. An example of this could be naming a "d1" and using comments to designate that that contains a sidebar to the left side of the page.

WHERE TO WRITE CODE (AND HOW IT CAN BE EASIER)

Technically, you can write code in any word processing program. Some webhosting services also have their own pages where you can write your HTML and CSS code. However, I strongly advise against this as it can become confusing to manage if you are inexperienced or working with long documents.

I would recommend using the program Phoenix Code for doing HTML and CSS. It is also able to handle documents in other coding languages. The cool thing about Phoenix Code is that it has a live preview feature, allowing you to watch how you code is changing the appearance and function of your site in real time. Additionally, it has a function where every time an HTML tag is opened, it automatically generates the closed tag for you to write your text or additional code in.

PUTTING YOUR BULLSHIT UP PUBLICLY

While self-hosting is possible, it costs money and therefore I am unfamiliar with its process. Because of this, I use the free version of the platform Neocities to host my website. While there are other platforms you can freely host your work on, I find Neocities to be the most well-known and easiest to use. Neocities also has a paid version of the platform which includes more space, capabilities, and a custom URL.

Despite the limitations of Neocities' free version, it is still possible to make an entirely functional website using it, and yes, you can even include audio if you know what you're doing. Simply host it on another site and link it via an audio element.

On the topic of webhosting, do not use Neocities as a file dump. Not only is it against their rules, it also fucks up their servers and is just plainly rude. There are many alternative file hosting services available, my favorite being FileGarden. Other things to keep in mind on the indie web is that blind people and people with photosensitive disorders still exist. Which means there is no excuse to refuse to write alt text and intentionally use bright and flashing gifs without proper warning. Don't be an asshole.

FUCK CORPORATIONS AND CONFORMITY

So why did I write all this? I'm a hater of social media and I love teaching. I strongly believe that the censorship and surveillance laws being passed in the recent months are highly unethical. So I think the only people who are here for us, the weird, the queer, the disabled, and by god those of us who want to watch porn or play violent video games. Make your own website. Dedicate it solely to one weird cartoon you're obsessed with. Show off your bad sewing techniques. Write about your day. It doesn't matter what it's about. Make it yours. Be proud of it. And encourage your friends to make their own bad website.

I'm not very good at coding but it doesn't stop me from running this site. If you need help with something and you think I could help you figure it out, email me at **letters2zenith@gmail.com**.

"You have to go the way your blood beats. If you don't live the only life you have, you won't live some other life, you won't live any life at all."

— James Baldwin