Introduction

Hey there! Full-Stack Development is super difficult, and it takes most people 1-2 years before they're able to truly understand the field well. Don't be afraid to ask questions and just keep plodding forward. Trust in the process and it'll all make sense eventually!

Tips to Succeed In This Course

Contrary to what many people believe, you do not have to be a wizard with math to write code. The two subjects share some overlap but also are quite different. Many artists, lawyers, historians, linguists, and others have great success learning how to code. Don't let those preconceived notions hold you back at all.

Also keep in mind that this is *hard*. Learning how to code is hard. It's similar to learning a foreign language where you have to learn a lot of small pieces of information to be able to make sense of the larger picture. It's easy to feel discouraged, especially when you feel like you *should* get something but don't. Keep your head up and keep trying. Walk away from the problem and come back later. Try explaining your problem to someone / your dog / a rubber ducky (https://en.wikipedia.org/wiki/Rubber_duck_debugging) (seriously, it works.)

Never feel afraid to Google *anything*. Every programmer you know, from the top person in the field to the newest student is Googling things *constantly*. This isn't cheating; it's a skill. It's a requirement. There is so much information coming at you that cannot possibly remember it all. Copy and paste code. Look at StackOverflow (https://stackoverflow.com/). Ask dumb questions. You get better by repeatedly exposing yourself to information. Each time a bit more will sink in.

Don't try to understand every piece all at once. There are times where it's okay to just trust that something works and come back later to understand how. It's a tough balance because you do want to try to understand what's going on. I'll try to signal to you what's worth diving into and what's worth leaving for another day but just know you don't have to understand it all at once.

Lastly, this isn't a get-rich-quick scheme. Learning to code is hard and requires a lot of hard work. While entry-level jobs are out there and you can get them with months of work, I guarantee you'll have to work hard for them.

The House Metaphor

HTML - The Frame

Why do you need three languages? Let's make the imperfect metaphor of building a house. To build a house you need tools (like your text editor, your browser, your command line.) After you have tools, you need all the building material: the 2x4s, the shingles, the dry wall, the windows: all the things you need to put together to make a house. This is the HTML, or hypertext markup language. However this house thus far isn't going to be very pretty to look and not very functional. It's not going to have any color or any sort of elaborate structures. It's going to be bland, inert, and boring.

Likewise, you can create a website that's *just* HTML but it's going to be a black-and-white text document with no style or interactivity.

CSS - The Decorations

In order to arrange, style, and generally make this house more useful, you're going to have some blueprints. In an overly-reductive way, you could think of the blueprints as being a set of rules: this 2x4 goes here, that shingling goes on the roof, this particular wall be blue, and this window goes here. You define a bunch of rules that dictates that if some item matches this condition, then some rule is applied to it.

If it is a 9x15 wall, it goes on the south side of the house. This is the CSS of your house, or the cascading style sheets. CSS is a series of rules that define that if you are an HTML thing that matches this condition, then apply some style to it. If you are the first paragraph in an article, your font size is 25px and your font color is blue.

JS - The Interaction

Okay, so now we have a well arranged and nice looking house. Now, being the modern age and me wanting all the gadgets, I want to install a whole slew of smart home devices. I want it so when I pull in the driveway with my smart-enabled car that the garage door opens, the lights turn on, the thermostat turns on the heat, the TV is set to continue my favorite TV show, and the smart cooker begins cooking dinner.

I am adding behavior to my house; I am adding a layer of programming on top of what exists. This is like frontend, or client-side, JavaScript. It's adding a layer of behavior on top of your website. Do you want to pop up a message if a user clicks a button? Do you want to refresh the stock-ticker on the page so it's accurate? Do you want to change the picture that's showing on your page every few seconds? These are things you'd typically do with JavaScript. And like all the smart home devices, having JavaScript on your page isn't always necessary.

This website, for example, probably wouldn't need javascript (it's actually just markdown), but any interactions that you see (for example the scrolling sidebar on the right or the dropdown menus on the left) use Javascript to add animations and additional spice to the website.

Backend - Services

Now, if I want to order delivery to my house, I have to call someone else. Someone not at my house. I'd use my smart assistant and ask them to call a pizza place and ask them to deliver pizza to my house. The pizza place in this example would be like a backend server. One pizza place can serve many houses, and it probably only does a few things (like make pizza, salad, drinks, etc.) and deliver that to all sorts of clients.

The clients, in this case, could be different peoples' computers, phones, smart assistants, smartwatches, smart ovens, who knows!? So one server can service many clients. In this case, we're just worried about people's browsers (like Chrome, Firefox, Edge, Safari, etc.) on their computers and phones. While the frontend code (the smart house stuff) is almost always in JavaScript (there's stuff like Web Assembly but for the purposes of this course just JS), the backend code can be in any number of languages: Python, Ruby, JavaScript, Java, Go, C#, etc.

Today we will only be using JavaScript on both the frontend and the backend so you don't have to learn two languages, but just know you *could* use a different language on the backend.

To recap:

1. The HTML is the structure. It's going to contain all the text, the various images tied to the text, and it will generally group things together. Just like the drywall in your house, HTML doesn't do anything other than exist. It relies on other things to do things for it and to it.
2. The CSS is the blueprint. It's all the rules of what goes where, what color it is, what size it is, what font it is, what the decorative background images are, like HTML, CSS doesn't *do* anything, it's just a set of rules that describe what things go where and how they look.
3. The frontend / client JavaScript is the smart home. It's all the cool pre-programmed stuff you can tell your house to do. JavaScript is what can change the HTML and CSS to react to various stimuli.
4. The backend code (we'll still use JavaScript) is the pizza place. It's a place where we can request things from and it will send back what we ask for. Or we can send things to it, like when you upload a new photo to your social media account. One server serves many clients, just like one pizza place serves many homes.

Evolution of the Web

Web 1.0

Web 1.0, also referred to as Syntactic web or read-only web was the first stage of the internet. The role of the user was static, they could consume information provided by content producers in the form of HTML and CSS.

This was before the widespread use of Javascript (which was created in 1995) and since Javascript is what enabled much interactivity, users couldn't actually interact with these sites.

Today, Web1.0 platforms are still common. Portfolio sites for example, can be completely static HTML and CSS pages with no interaction.

Web 2.0

Of course read-only is pretty boring, so when Javascript became more widespread developers went nuts to provide interaction to websites. Thus, Web2.0, also known as the Social Web or read-write web was created.

In this era, users can produce content and interact with websites. Today, most websites like Amazon, Twitter, ore ven this one are Web2.0 websites.

Web 3.0

I won't talk too much about this one since it's out of scope, but it's basically the semantic web (read-write-execute) and most commonly associated with Blockchain. feel free to google more about it.

The Development of Web2.0

HTML, CSS, and JavaScript are all powerful tools in their own right, and these are the languages that the entire internet is built on. Try inspecting the browser `right click → inspect element`. If you go to the "elements" tab at the top of the inspector, you'll be able to see the HTML, CSS, and some of the JavaScript for the site you're inspecting!

When Web2.0 first started, people manually wrote HTML, CSS, and JS files, linking them together and using Web API (https://developer.mozilla.org/en-US/docs/Web/API)'s to allow the JS to interact with the HTML.

When you write a webpage, the browser is what actually interprets and executes your code, rendering the HTML and CSS and executing the JS to make your site functional. All that you send is the code (today that often bundled and condensed).

Every browser decides how to execute your code differently. This means there are variances between browsers in how rendering and execution is handled, which can cause some nasty bugs on certain platforms even though the code works fine on another.

Today, much of HTML, CSS, and JS is standardized by various nonprofit organizations such as the HTML Living Standard (https://html.spec.whatwg.org/) or the ECMA Script standard (https://www.ecma-international.org/publications-and-standards/standards/ecma-262/) for Javascript.

An example of a basic HTML site is below:

```
<!-- All of this would be in a file like `index.html` -->
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <!-- CSS -->
  <style>
    body {
      background-color: powderblue;
    }
    h1 {
      color: blue;
    }
    p {
      color: red;
    }
  </style>
  <body>
    <h1>This is a Heading</h1>
    <p>This is a paragraph.</p>
    <span id="demo" />
  </body>
  <!-- Javascript-->
  <script>
    // This will set the `span` with an id of `demo` to have the child "Hello Javascript"
    document.getElementById('demo').innerHTML = 'Hello JavaScript!';
  </script>
</html>
```

However, you can quickly see how this can become a mess when having to deal with tons of interactions. For example, the simple act of "signing up" can involve over a hundred of lines of vanilla javascript. You have to watch for changes on inputs, collect these values in Javascript, handle when the user presses submit, deal with loading states, and deal with response dates (among other things).

Javascript libraries like JQuery (https://jquery.com/) were built to help this problem by giving Javascript more power in less lines of code by abstracting some of the tedious logic behind a library.

Even this wasn't enough, and thus frameworks were born.

Frameworks are tools or languages built on top of HTML, CSS, and Javascript to expand their power and make them easier to use. Some popular frameworks include React (https://reactjs.org/), Springboot (https://spring.io/projects/spring-boot) for Java, and Flask (https://flask.palletsprojects.com/en/2.0.x/) for Python.

Frameworks can encompass Frontend, Backend, or even both (fullstack).

As you can see, the web is a series of abstractions around abstractions. Frameworks are simply abstractions around Javascript which is an abstraction around the instruction set for the browser to run your code. Pretty neat!

Now that we've introduced the concepts of frameworks, let's talk about the structure of the course and what frameworks you'll be learning.

Trusted Resources

Something really important is that you choose to learn from good sources. Just like it's important to get your news from quality sources, it's important to get your technical information from sound sources. Here are some of my personal favorites:

- For anything to do with HTML, CSS, or JavaScript, Mozilla's MDN (https://developer.mozilla.org/en-US/) is my go-to. I literally have it open all the time.
- CSS Tricks (https://css-tricks.com/) has fashioned itself into a premier development website. It has great content not just for CSS but for HTML and JavaScript too. If I want a tutorial, I'll head there. If I want more technical how-to info, I head to MDN.
- For video content, you really can't beat the content on Frontend Masters (https://frontendmasters.com/). I love it.
- If I'm working with a library or a framework, it's a good idea to head directly to their GitHub (we'll talk about GitHub later) page or their official documentation. It's best to head straight to the source.

Frontend Masters also puts out a really awesome book every year called the Frontend Handbook (https://frontendmasters.com/books/front-end-handbook/2018/). It's a good way to get an overview of the whole industry.