Programming for Non-Programmers Bootcamp

Begin, introductions

On the board at the start:

Repo URL

Download VS Code

Download Github Desktop

Get Github account

Goals for today:

- Learn about the programming landscape
- Learn basics that every programming language shares without learning code first
- Get to know the Chrome Developer Tools
- Learn basic HTML
- Learn basic CSS
- Create a resume and portfolio site
- Deploy to the web with Git

10:00

Instructor Intro, class intro.

Students introduce themselves.

10:20 - Lunch

Intro to programming and the internet

High level overview of various programming languages and what they do.

Essential Terminology

Learning Objective: Learn the essential words and concepts that are used on a daily basis by engineers and project/product managers on the job.

Here are some words and concepts that will hopefully give you a more holistic view of the more technical aspects of the industry.

Define: Program

Discrete, highly logical and explicit instructions that are parsed and executed by a computer.

We call this set of human-readable instructions **source code**, or colloquially, a **computer program**.

Compilers can take this source code and transform it into **machine code**, a representation of the source that can be executed by the computer's **central processing unit** or **CPU**.

Not all programs are compiled though, some are **interpreted**. The difference is that compiled languages need a step where the source code is physically transformed into machine code. However, with an interpreted language, this additional step is **excluded** in favor of **parsing** and **executing** the source code directly when the program is run.

Speaking of source code, sometimes programmers, even ones acting on behalf of corporate giants like Apple, publish the source code for their programs. This is called **open source**.

How programs are written

All programs are composed with a collection of **fundamental** concepts that, when combined, can essentially dictate a wide variety of tasks a computer can perform.

Here are a collection of these most important concepts:

- **Declarations**: typically, we can store and retrieve data in our programs by associating them with intermediary values that we call **variables**
- **Expressions**: we use expressions to evaluate stuff. For example, **2 + 2** is an example of an expression that will **evaluate** a value, namely 4.
- **NOTE**: typically we can use expressions and declarations in tandem to perform complex tasks. For instance, we can reference a variable we declared in an expression to help us evaluate new values which can then be stored.
- **Statements**: statements will use expressions and declarations to alternate a program's **control flow**, which is essentially the order in which declarations, expressions, and other statements are executed.

Aside from these fundamental concepts, we also talk a lot about this idea of **algorithms**.
An **algorithm** is simple a series of declarations, expressions, and statements that can be used over and over again to solve well defined problems of a certain type.

People tend to talk about algorithms like they are mysterious forces of nature, but I guarantee that you know a few algorithms yourself.

For example, we can implement an algorithm that converts temperature from **fahrenheit** to **celsius**. It would look something like this:

1. **Declare** F = 32;
2. **Expression** (**F** - 32) / 1.8;
3. **Declare** C = **Evaluated** expression from **[2]**

This is a form of **pseudo** code where we define the steps a computer program — **any** — computer program can take to convert **fahrenheit** to **celsius**.

The beauty of programming is that all of it revolves around the same key set of concepts and ideas. For this reason, we do not need to specify any **particular programming language** when discussing the functional aspects of a program.

Define: Programming languages

A programming language is a series of **grammar** and **rules** that we can define towards writing source code.

Languages are effectively different approaches towards communicating the same ideas in programming. Essentially, we can communicate in say both **French** and **English**, what mainly differs is the structure of our sentences and the actual words and sounds themselves.

The **same analogy** can be made with programming languages.

Examples of programming languages

There are many. Way too many.

Here are some of the most popular ones, though.

1. **JavaScript**: this language is interpreted.
2. **Python**: this language is interpreted.
3. **Java**: this language is compiled
4. **Ruby**: this language is interpreted.
5. **C/C++**: these languages are compiled.

These languages all build on the same concepts defined above; the main difference lies in **how** they are run (compiled vs interpreted) and also **how** they are used.

In general, anything programmable can be programmed in each of the languages defined above. However, some languages are better suited for certain tasks above others.

For example, to perform web programming on the front-end, you'll want to write JavaScript. This is because all browsers collectively support running javascript within it's environment

Lunch

Slides end here!!

13:30

What is web development?

The use of programming languages and concepts towards producing a system that listens to user requests over the internet and serves back HTML, CSS, and interaction code to the client.

What is the difference between a web site and web app

Web sites are more **static**. Think of them as sort of like interactive brochures. They typically don't hold any state and are usually just information.

A web app is more like an iOS or Android app. It holds state, does not refresh itself and in general **does something** vs **showing something**.

What are the stages of web development

Depends on the methodology used by team. Typically, there is a UX phase, a Design phase, a Development phase bundled with extensive QA. This was the process used in the large company I used to work in.

For a startup, the process is similar, however there is a lot more room for iteration and testing out ideas by putting something up for users to use and deliver feedback.

What is the difference between front-end and back-end web development

---

Front end: Client side interactions and structure. This includes stuff like iOS apps and whatever you load in browser.

Back end: Server side, data storage and retrival. Needs a client **of some sort** (not necessarily a browser) to work. Deals with storing and retriving information.

Main Class Deliverable

In order to explore the core concepts of this class and achieve our learning objectives, we will recreate the **traditional resume template** for the web.

Key Requirements

- Template should resemble the typical resume.pdf we all maintain
- Template should use tasteful CSS3 styles to accentuate key peices of information and design principles
- Template should work across all screens including (and specifically) mobile
- Template should be **printer friendly** and take up exactly one page when printed.

HTML lesson

Show them pop code and script ed sheet.

Front end code is totally on your own machine

Show them dev tools

look at dev tools in facebook

show them the You Tube trick

```
$('video').currentTime = $('video').duration
```

Go and change some stuff in the browser's rendered HTML.

Browsers understand 3 languages.

- HTML: structure and content
- CSS: Style
- JavaScript: Interactivity

said differently:

- HTML: bones
- CSS: skin, hair, and clothes
- JavaScript: brain

Take a look at the elements, console, and styles tabs of dev tools.

You do

- Go to a news website and have some fun.
- Make your uncle the king of Thailand. Change the election results. Add some digits to your bank account.
- Refresh the page and see your dreams melt away.

This is what I mean by front end or client side.

Exercise:

Go to Airbnb.com and look at the following

- What language is the text of the navbar written in?
- What language makes the button hot pink?
- Click Sign up. What language makes the modal pop up?

14:20

We Do: HTML Elements

Go to popcode.com

Before your browser even knows that it's supposed to roll with HTML, you have to tell it.

```
<!doctype html>
<!-- this is a comment that will show up in source code-->

<!-- Most users won't see this -->

<html>
 <head>
```

```
  <!--
  the HEAD section of html does not have any content
  that the user can see

  instead, we place things like:
  page title
  external css links
  SEO keywords
  here
  -->
  <meta charset="utf-8">
  <title>My resume</title>
  </head>
  <body>
  <!--

  the BODY section will contain all the tags
  that the user can _SEE_ and _INTERACT_ with
  -->
  </body>
</html>
```

https://twitter.com/iamdevloper/status/481859418137853952

HTML is made of *elements*.

Each little thing is an element.

They can nest to kingdom come if you like

`<!doctype html>`

This tag tells the browser to read our HTML content as HTML5, the latest and greatest revision of the HTML spec. We MUST include it as the first thing on our .html files. I usually type it as all lower case, but it's common practice to also type as: <!DOCTYPE html>

`<html>`

This is the root tag. Basically, all other tags in your html file must live inside this tag. Note how on the bottom of the code snippet, we have a This is called closing a tag and we must close all tags that we open (with the exception of a few). If we do not do this, our HTML markup becomes invalid.

`<head>`

This section contains content that does not show the user things. Typically, we would expect to see things such as the page title, external links, and SEO tags here.

`<body>`

All the magic happens here. All the tags the user sees and interacts with should live in this tag.

Comments and why they are so handy for programmers

```
<!--
 the h1 - or heading one - will have the most important text on page
 by the old guard, we should really only have one h1 per page
 -->
 <h1>Hello, Wrold</h1>

 <h2>This is a h2</h2>

 <h3>This is an h3</h3>

 <h4>This is an h4</h4>

 <h5>This is an h5</h5>

 <h6>This is an h6</h6>

 <!--
 this is an inline element
 unline the block element, which is meant to provide structure
 the inline element is interpreted as content
 this means that will appear next to one another
 -->
 <strong>This is an inline element</strong>
 <strong>This is another inline element</strong>

 <!-- this is a block element so it will NOT be on the same line -->
 <h1>Will this be on the same line?</h1>

 <!--

 differences between block elements and inline elements
 block: takes up entire width of page unless otherwise told
 (we don't know how yet)
 we can impose dimensions on block elements
 inline: meant to be content or text
 we cannot impose dimensions on inline elements
```

```
 -->

 <!-- how to add more spaces or line breaks?? -->
 <h1>THIS will have many       spaces</h1>

 <h1>This is <br> Sparta</h1>

 <a href="http://www.google.com">Hello, Wrold I'm a link, yo</a>

 <!--
 this is one mode
 <tagName attribute1="someValue" attribute2="someOtherValue"></tagName>

 this is a self closing tag
 <tagName attribute1="someValue" attribute2="someOtherValue">
 -->

 <!--
 convention:
 external links open up in new tab
 absolute URLs

 internal links open up in same tab
 relative URLs
 -->

 <a href="http://www.google.com" target="_blank">Hello, Wrold I'm ALSO a link,
yo</a>

 <em>This is an em</em>

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

 <img src="http://placekitten.com/g/200/300">
 <span>This is a kitten. S/he is cute.</span>

 <ul>
 <li>This is a list item <div></div></li>
 <li>This is another list item</li>
 <li>This is a third list item</li>
 </ul>
```

```
 <ol>
 <li>This is an ordered list item</li>
 <li>This is another ordered list item</li>
 <li>This is a third ordered list item</li>
 </ol>

<hr>
<p> wow look at that line

</p>
 <div>
    <!-- a div is just a box. that's it.
    it displays 'block' meaning by default it just takes up the whole
horizontal space.
-->

</div>

 <span>
    <!-- a span is just a box, but it displays inline rather than block,
meaning that it doesn't break up a line of text or a paragraph.  -->
</span>

<!-- spans and divs are good for grouping pieces of the ui together so we can
target them for styling or even interaction. -->
```

relative links

relative images

Show them PopCode

15:20

Assignment lab:

You now have everything you need to make a resume in HTML.

Create a folder on your desktop called `resume_html`

Drag it onto VS Code in the dock or windows taskbar.

Create a new file in there called `resume.html`

It's going to be all Times New Roman and stuff but it's important to get the practice into your very fingers.

Try and use all of the following tags once:

```
<html>

<head>

<body>

<h1>

<h2>

<ul>

<ol>

<li>

<p>

<img>

<a>

<div>

<span>
```

Use your actual resume or generate nonsense text at Hipster Ipsum

https://hipsum.co/

After you are in good shape, we'll use some CSS to give these Crappy Times New Roman things style.

now do doc + tab in VS Code to wow them

15:50

CSS!

CSS is code that tells the browser how to make things look. It's hella fast and powerful.

CSS Zen Garden

- Show inline styles and explain why they are the devil
- Show how to put it in a style tag
- Show how to link in a stylesheet relatively
- Show how to link in a Google font
- difference between block and inline elements

CSS selectors and properties

```
/* this is a comment in css */
/* not the same as html or javascript, sorry. I wish it was consistent */

/* this is all there is to it: */

selector {
  property: value;
}

/* that is pretty much all there is to css syntax. Part of what makes it great
is how accessible it is. It gets more complicated for sure, but from here on,
it's just a matter of getting used to things and trying stuff until it looks
good.

This would be a good time to show the family guy thing.
*/
```

selectors

- tag name
- id
- class
- mixture of the two
- child selector

Properties

- color
- font-famliy
- font-size
- width
- float
- height
- width
- font
- background image
- width percentage

  show

Exercise with the About me Code along

Final CSS:

```
@import url('https://fonts.googleapis.com/css?family=Montserrat|Slabo+27px');

h1, h2, h3 {
  font-family: 'Montserrat', sans-serif;
  color: #3c250a;
}

p, li {
  font-family: 'Slabo 27px', serif;
  color: #3c250a;
}


body {
```

```
  background: url('abstract-mosaic-background.png') no-repeat center center
fixed;
  background-size: cover;
}

.wrapper {
  width: 960px;
  margin: 0 auto;
}

header, footer {
  background-color: #595959;
  text-align: center;
  padding: 15px 30px;
}

.main-content {
  background-color: #808f85;
  padding: 10px 0;
  border-top: 2px solid #cfd11a;
}

header img {
  width: 50%;
  border-radius: 15px;
}

.main-content ul {
  list-style-type: none;
  padding-left: 0;
  margin: 0 auto;
  text-align: center;
}
.main-content ul li {
  padding: 10px;
  display: inline;
  font-weight: bold;
}

a {
  color: #CFD11A;
  text-decoration: none;
}

.main-content p {
  margin: 10px 40px;
```

```
}

.main-content h2 {
  text-align: center;
}

#projects {
  display: grid;
  grid-template-columns: 30% 30% 30%;
  grid-gap: 3%;
  justify-content: center;
}

.project {
  width: 100%;
  display: inline-block;
  background-color: #f2e9dc;
  border-radius: 15px;
}

.project h2 {
  text-align: center;
}

.project img {
  max-width: 100%;
}
```

16:50

Deploy to the web with github pages

45 mins

Please stay with me on this. I will go as slow as we need to because if you miss a step, you will be trampled underfoot.

Make sure that you have a Github.com account.

Important: Make sure that you *verify your email address* by opening your email and clicking the verify link.

Brief review of git and GitHub. Show the commits to the actual class code repo.

https://desktop.github.com/

Create a repo !!!!!!!!!!on your desktop so you can find the thing!!!!!!!!!! called .github.io

In finder, move index.html into that folder that was created.

Look in the github client to see what results.

Commit to master with a suitable commit message.

Publish.

Look at your github account

Go to .github.io in your browser. (Note: There might be a minute or two of lag. Github is hosting your code and website for free, give them a break. If it isn't working after two minutes or so, I will help debug.)

Edit one line of index.html and commit/push.

Move the resume html in there as well. Repeat.

Your turn

Move the styles files into the repo and publish to the web.