Programming for Non-Programmers — The Basics

On the board at the start:

Repo URL

Download VS Code

Goals:

Part 1:

What programming is, what languages do what

Part 2:

About web development, how programmers work and how they fit into an organization

Part 3:

We will learn HTML and CSS by doing it!

Part 4:

We will learn a tiny bit of JavaScript by doing it!

Begin, introductions

Instructor Intro, class intro.

Students introduce themselves.

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++**: this language is 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.

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 consulting firm 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.

Slides end here

Mess with You Tube

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

Mess with NYT

Let's write our temperature conversion in actual JavaScript code!

Declarations use the `=` syntax with a keyword `let`, `const` or `var`

```
  let step1 = 212 - 32;
  let step2 = step1 * 5;
  let c = step2 / 9;
  c;

// should be 100
```

Functions

```
function farenheitToCelsius(farenheit){
  let step1 = farenheit - 32;
  let step2 = step1 * 5;
  let c = step2 / 9;
  return c;
}
```

Paste into chrome console.

Now paste into app.js and open index.html in a browser.

HTML

`<!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.

```
<!-- comment! -->

<h1>Hello, Wrold</h1>

 <h2>This is a h2</h2>

 <h3>This is an h3</h3>

 <strong>This is an inline element</strong>

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

 <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>

 <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>
```

Desert Island level HTML tags

```
<html>
<head>
<body>
<h1>
<h2>
<ul>
<ol>
<li>
<p>
<img>
<a>
<div>
<span>
```

CSS!

CSS is code that tells the browser how to make things look.

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

Properties

- color
- font-famliy
- font-size
- width
- width percentage
- display

Functions

Imagine having to write the same stuff or edit code every time you need to change inputs.

Functions (just like proper math functions) let us reuse code.

In football, when the quarterback is shouting out orders, he doesn't say "Hey, I'm going to throw the ball in that direction. You run up the middle and do a buttonhook to catch it, oh and you, please act like you are going to catch it off to the left. Thank you!"

Nope. The QB has to use some kind of shorthand for a series of instructions that the players already know. Perhaps the QB might say `OMAHA, OMAHA` and the other players know that the running back will then execute the Omaha move (running up the middle.)

These are functions. Blocks of code that you can reuse. You define them, then you call them, or execute them.

```
function greet(){
  //this function doesn't take any input, or arguments as we would put it
  console.log("why hello!")
}

//this is the same thing

let greet = function(){
    return "why hello!";
}

//nothing happened.

//Right! gotta call the function

greet()

//Let's give it an argument

function greet(name){
  let response = "Why hello, " + name +"."
  return response;
}

function myAddition(a,b){
  return a + b;
}

function circleArea(radius){
  // radius means something different here than outside of the block: it is
THIS radius passed in as a parameter.
  const pi = 3.141592654;
  return pi * radius * radius;
}
```

```
//you go and bake your cake, then you clean the kitchen and make a new cake.


//in the function, its like a separate environment. this is called scope.
```

You do

- create a function for temperature conversion that lets you convert any temperature input.

Expressions, Conditionals and Operators

What happens when we abuse that temperature conversion function and input "lizard"?

JavaScript Statements use a set of operators here is a small subset of them we can use today use:

- `>`
- `<`
- `>=`
- `<=`
- if
- else if
- else

Let's change the temperature conversion function together so it returns a friendly error if you don't put in a number.

21:45

Wrap up with HTML, CSS and JavaScript working in concert

Step 1: Look at the html in the browser

style.css

```
#freezing-warning, #boiling-warning {
  display: none;
}
```

app.js

```
// paste your function that converts farenheit to celcius here:

function farenheitToCelsius(farenheit){
  let step1 = farenheit - 32;
  let step2 = step1 * 5;
  let c = step2 / 9;
  return c;
}

// function here that shows the answer and shows whether it is freezing of
boiling.

function displayAnswer(c){
  $("#answer").html(c);
  if (c >= 100){
    $("#boiling-warning").fadeIn();
  } else if (c <= 0){
    $("#freezing-warning").fadeIn();
  }
}
// listen for submit of the form to run the function that converts and the one
that

$("#temperature-conversion-form").on("submit", function(){
  $("#boiling-warning").hide();
  $("#freezing-warning").hide();
  let f = $("#farenheit-input").val();
  let c = farenheitToCelsius(f);
  displayAnswer(c);
});
```