Programming for Non-Programmers Bootcamp — Day 2

On the board:

Today's *very ambitious* goals:

- Review HTML, CSS and Git
- Learn the very basics of programming with JavaScript
- Use jQuery to make code that a user can interact with
- Make an image slider
- Make a frivolous app that tells you what your name would be if you were in Star Wars
- Make a cooler app that lets you search for movie info.

10:00

Warm up

I want to give you a chance to reinforce your knowledge and explore CSS, HTML, and Git. Let's break things, and fix them.

Let's spruce up your portfolio page. After each step that you do, make a commit in Github Desktop, and push to your github portfolio.

1. change the links to be your real social media profiles.
2. Copy your Resume HTML into your github repo and link to it with a *relative link* like this:

```
<a href="resume.html">My Resume</a>
```

This lets you make links within your website.

3. Drag a real selfie into the repository and use the same technique above to make the src attribute of the header photo reference your actual photo. See if you can figure out how to do this with the power of Googling (or Binging [or Duck duck going]).

4. Explore HTML and CSS by thinking of changes that you want to make to the styling. Some ideas:
    - Make your headshot round instead of rectangular.
    - Change the background photo to something tasteful. Maybe a city skyline or something
    - Change the three columns to two and see if you can arrange them nicely.
    - Add new content and make it look how you want it to look.

10:45

Let's learn JavaScript and jQuery!

A little history lesson

---

15 minutes

The web was originally created by Tim Berners Lee while he was working with CERN in Switzerland. It was mainly intended to make it easier for academics to share research and link between them.

Websites started out like that. A piece of paper with links. Once the page loaded, that was it!

http://info.cern.ch/hypertext/WWW/TheProject.html

Brendan Eich, an engineer at Netscape, was tasked with creating the first fully functional programming language that can be executed in browsers. He did this over the course of 10 days, and some of the 'gotchas' of the language have to do with that. Since then it has slowly evolved as each company making browsers tugged the language in different directions.

The rest involves a Game of Thrones-like power struggle amongst Silicon Valley giants who didn't do a great job of cooperating. The result was chaos and incompaitble implementations for years. This makes it a bit more confusing to learn, but this is just how things are right now.

Between 2009 and 2015, we had one specification — ECMAScript 5 — which is still implemented by modern browsers, but now we have ES6, which is WAAAY nicer and easier to work with. Most of that is implemented already by modern broswers.

By the way, when you hear "modern browser" just think "Not Internet Explorer or something really old"

You might have heard people ripping on Internet Explorer, and there is a lot of truth to it. While Mozilla and company tried to advance web standards, Microsoft started the first decade of the century off with near absolute browser dominance, so it just felt like it didn't need to be told what to do. Designers would have to write code specifically for IE as well as the more modern, standards compliant code. Today, the cross browser issues that used to make developers' lives miserable are

mostly in the past and there are countless tools to make sure that your code looks almost exactly the same in any browser.

We will be learning the Es6 syntax where applicable today.

As I mentioned yesterday, JavaScript is going through a cambrian explosion of tools and frameworks and it's going to move faster and faster.

There is an endless and endlessly expanding universe of frameworks for JavaScript that make it more powerful and easier to write. For example, we will experiment with jQuery the (still) most popular and easiest to learn framework for front-end JS.

What can I do with JavaScript?

- Breathe life into web pages, let users control stuff on the page
- Animation
- Fetch data from servers asynchronously
- Everything

11:00

Where do I put my JavaScript files?

```
NOTE the <script> tag

This is a new tag we have never seen before; remember that the <link> tag is
for CSS files and the <script> tag is for javascript files (for now).
the src attribute is what we use to link to the external js file
remember to CLOSE your script tag, unlike the <link> tag, <script> is NOT self
closing!
```

It goes way down at the bottom of body so the content of the page is visible prior to the javascript files downloading.

Quick example: Magic button

Make an html file starting with the doc shortcut:

start by showing the script attribute, type text/javascript, then write in the magic button code into it.

```
<!doctype html>
```

```
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>
</head>
<body>
  <h1>make the text all red</h1>
  <button id="alertButton">Click me</button>
  <script type="text/javascript">
    document.getElementById("alertButton").addEventListener("click",
function(){
    document.querySelector("body").style.color = "red";
    document.querySelector("h1").innerHTML = "Now it's red. Well done.";

});

  </script>
</body>
</html>
```

Trite! I know. But this is basically the foundation of how you make a ui respond to user input without just clicking a link, then having the page load again.

JavaScript gives is a chance to do stuff without a full request-response process.

Separation of concerns

This is a big deal for programmers. You wouldn't want your shower in your kitchen, right? The shower goes in the bathroom unless your landlord gets creative. This is why the example that I just made was kind of bad. The JavaScript should live with the other JavaScript rather than imposing on HTML turf.

Let's give this JavaScript code into it's own digs.

> Move it to app.js or whatever

11:30

JQuery Slick lab

60 minutes

What does jQuery do?

jQuery makes it MUCH easier to manipulate the stuff on the page. It's practically as easy as CSS!

I'll show you for example, how much easier jQuery makes that magic button lab:

How do I get jQuery?

Point them to the CDN. Which is....?

place CDN link in the bottom to get a hold of jquery and demonstrate in the console.... which is???

Comment out vanilla js then type this:

```
  <script
  src="https://code.jquery.com/jquery-3.2.1.min.js"
  integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4="
  crossorigin="anonymous"></script>
  ....

$("#alertButton").on("click", function(){
      $("body").css('color', 'red');
      $("h1").html("Now it is red. Thank you.");
    });
```

Okay what the heck just happened?

Walk thorugh code. Don't worry, this will all become clear in a bit. Don't worry about memorizing syntax. I had to look one thing up myself.

Okay browser, please download jQuery so we can use jQuery.

Now that jquery is here we can talk to jquery and ask it to do stuff for us.

Oh hey jquery, can you find the element with the id 'alertButton' and listen for when someone clicks it. when someone clicks it do the following: take the element called body, you know, the whole thing on the page, and with regards to its css, make the color red.

Now take the element called h1 (see how this is just like css?) and make its html (text) Now it's red thanks.

Refactoring

You might have heard programmers talk about how they can't be bothered to do something because they need to refactor something.

Refactoring is when you alter code to make the code better with no visible changes at runtime. As you can see with the jQuery example, I changed a bunch of syntax yet from the perspective of a user, nothing changed. No better performance, no new features. But for the programmer herself (either the future self that forgets writing the code or a new developer joining the team) it is easier to grasp what the code does, easier to maintain it and quickly make updates.

Benefits of open source

JQuery is open source software, and there is a lot of other open source stuff in its orbit. Using jQuery opens us up to all the lovely UI tools that other people have already written, so we don't have to reinvent the wheel every time we build a bike.

Slick slider lab

Check out the HTML in the slick demo folder.

Explain Semantic UI and walk through the HTML.

So this is going to be a slideshow.

So we want to make a slideshow. That could take all day! Fortunately there are like a million jQuery plugins for everything under the sun. We just have to find the tools, read the docs, and figure out how to use them for our purposes.

This bears repeating. Look for answers and open source tools, read the manual.

Go to Slick's website https://kenwheeler.github.io/slick/

Read the getting started manual.

RTFM!

throw this after semantic ui css

```
<link rel="stylesheet" type="text/css"
href="https://cdn.jsdelivr.net/jquery.slick/1.6.0/slick.css"/>
```

and after jquery at the bottom

```
<script type="text/javascript"
src="//cdn.jsdelivr.net/jquery.slick/1.6.0/slick.min.js"></script>
```

```
$('#slideshow').slick();

//make images 100% width to solve one problem
//then solve the height problem by looking at the docs.

$('#slideshow').slick({
  adaptiveHeight: true
});
```

12:30

Writing our own JavaScript

So that's awesome. We can already make our websites interactive with JavaScript and jQuery. But if we want to actually know what we are doing with it and get more creative on our own, we will need to learn some basic programming.

Wax on, wax off, before you know it you will be a champ.

Review of declarations, expressions and statements.

Declaring variables in JavaScript

3 ways:

```
// hi! I am a comment

const pi = 3.141592654;
// the = sign means assignment, not equality.
// You can get away with no semicolon, but Santa Clause is watching.
// const means the variable can never be reassigned.

let x = 42;
// this is the new way of doing it. (Note that if you start with let, you
can't let the same variable again. you will have to reassign th)

var p = 55;
// you will see tons of tutorials still using var instead of let. I won't get
into the difference.

//you can declare something without setting it.

let n;
// not defined vs undefined

//Concatenation and addition (polymorphism)

let x = 5;
let y = 3;
let answer = x + y; // => 8
alert(answer);
let x = "five";
let y = "three";
alert(answer);

//strings and numbers

typeof 'imastring';
typeof 4;


// redeclaring variables
```

```
let city = 'New York';

let city = 'London'; // this will cause an error

city = 'London' // now it works.

// pi is already defined as a const therefore.....
pi = '23o42o4392' //this will be an error.

Unless a variable will >definitely< change later, use const.

If you like it put a const on it.

sorry..... dad joke.
```

`/js-practice/declarations.js`

Try it in your code editor, then try it in the console of your browser (doesn't matter which site you are on)

repl.it is a great scratchpad for practicing JavaScript

Types of data

A variable type is a way to classify the different kinds of data we can save to a variable. There are exactly 6 types of variables:

Primitives

- `undefined`
- `null`
- boolean
- number
- string

Non Primitive

- Object

Primitives

A Primitive type is a most basic bit of information that you can store. For example, a number is a primitive because it cannot be made up of any of the other types of variables

Alternate definition: Think of this as an atom -- atoms are atoms because we cannot break them down into any more basic bits, same goes for primitives

`undefined`

Undefined is the default state of any variable. Basically means the variable is empty or has not yet been assigned a value, primitive or otherwise

`null`

The null variable is different from the `undefined` type, but only subtly so.

1. the `null` type is assigned to a variable, but its "value" is empty.
2. the `undefined` type is by default the value of each variable that is declared but not defined

13:00

Lunch

13:30

Numbers

```
let myNumber = 1;
let pi = 3.14159; // ...approximately
// all the rules of math apply
// show modular division in action
// ++
// --
```

Your turn

Let's write the code to actually implement the Farenheit to Celsius conversion. We can do that already with our JavaScript knowledge! Write the code to convert 212 degrees F into Celsius. Then try it with some more inputs of your own. Remember to `console.log` the result!

`https://repl.it/@trivett/tempConvert`

```
let farenheit = 100;
let celsius = (farenheit - 32) * 1.8;
console.log(celsius)
```

13:45

Strings

Escape `'` with `\'`

If you start with `"` you have to end with it.

Concatenation & interpolation

`.length`

Template literals with `

Plus operator can add a string to a number but can't subtract or divide. If that happens you get the dreaded NaN

Booleans

True or false. Basically.

Booleans are super handy for control flow, which we will get to in a bit.

```
let myBooleanValue = true; // true
let myBooleanValueThatIsFalse = false; // false
console.log( typeof myBooleanValue );
```

Expressions often evaluate to booleans

14:00

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(){
  console.log("why hello!")
}

//nothing happened.

//Right! gotta call the function

greet()

//Let's give it an argument

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



//return value vs console.log rigth now these functions are returning
undefined
```

```
//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.

// setting variables to return value of a function
```

14:30

We do

- concatenate strings
- area of a circle
- hypoteneuse

15:15

You do

- Go back to your Repl.it for converting temperature. Make this a function that takes an argument and RETURNS the celsius temperature.

15:45

Expressions, Conditionals and Operators

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

JavaScript Statements use a set of operators

- `===` `3 == '3' => true` `3 === "3" =>false`
- `!==` and `!=`
- `>`
- `<`
- `>=`
- `<=`
- `&&`
- `||`
- `!`
- 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.

16:15

Let's do that temperature conversion thing but actually make it accessible to users!

temp convert lab. Do together in Repl.it

1) paste in temp conversion function

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


function farenheitToCelsiusConverter(f){
  let celsius = (f - 32) * (5/9);
  return celsius;
}

function clearInputsAndWarnings() {
  $("#boiling-warning").hide();
  $("#freezing-warning").hide();
```

```
  $('#farenheit-input').val("");
}

function showAnswersAndWarnings(c){

  $("#answer").html(c);
  if (c >= 100){
    $("#boiling-warning").fadeIn();
  } else if (c <= 0){
    $("#freezing-warning").fadeIn();
  }
}

// I wrote this for you, let's look at it together.

$("#temperature-conversion-form").on("submit", function(){
  let f = $('#farenheit-input').val();
  clearInputsAndWarnings();
  let c = farenheitToCelsiusConverter(f);
  showAnswersAndWarnings(c);
});
```

Note how these functions don't have to be in any particular order, but it is good practice to keep it logical. Explain how JavaScript reads and evaluates code *twice*. This is called hoisting and it really throws people off.

16:30

You do: your star wars name

First, open html in browser and show how the JS console has access to the myStarWarsName function.

Hit submit. But?

Make a function that takes input and returns your "Star Wars name"

10 minutes in present String Slice as a hint

step one:

```
function myStarWarsName(firstName, lastName, street, city) {

  let swFirstName = `${firstName.slice(0,3)}${lastName.slice(0,3)}`;
  let swLastName = `${street.slice(0,3)}${city.slice(0,3)}`;

  swFirstName = swFirstName.toLowerCase();
  swLastName = swLastName.toLowerCase();
  swFirstName = swFirstName.charAt(0).toUpperCase() + swFirstName.slice(1);
  swLastName = swLastName.charAt(0).toUpperCase() + swLastName.slice(1);

  return `${swFirstName} ${swLastName}`;
}
```

Let's see your results

Get student results, then dry it up a little like this

Separate functions to make things clearer to the next developer

```
function firstThreeLettersDowncased(str){
  return str.slice(0,3).toLowerCase();
}

function capitalizeFirstLetter(str){
  return str.charAt(0).toUpperCase() + str.slice(1);
}

function myStarWarsName(firstName, lastName, street, city) {
  let swFirstName =
`${firstThreeLettersDowncased(firstName)}${firstThreeLettersDowncased(lastName
)}`;
  let swLastName =
`${firstThreeLettersDowncased(street)}${firstThreeLettersDowncased(city)}`;
  return `${capitalizeFirstLetter(swFirstName)}
${capitalizeFirstLetter(swLastName)}`;
}
```

Add user interaction that bind the html to JS code:

try both submit and keyup

```
  $("#sw-name-inputs").on('submit', function(event){
    event.preventDefault();
    let starWarsName = myStarWarsName($("#first-name").val(),$("#last-
name").val(),$("#street").val(),$("#city").val());
    $("#answer").html(starWarsName);
  });
```

```
// Refactor with Lodash

function myStarWarsName(firstName, lastName, street, city) {
  let first = _.startCase(firstName.slice(0, 3) + lastName.slice(0, 3));
  let last = _.startCase(street.slice(0, 3) + city.slice(0, 3));
  return `${first} ${last}`
}
```

17:30

Collections

Arrays

OPTIONAL

An array is an ordered list of stuff. That's it.

Some Array methods:

1. push
2. pop
3. shift
4. unshift
5. forEach

16:45

Objects

OPTIONAL

```
// objects are basically key-value pairs
// the values are primitives.
// Objects are the bedrock of most web APIs that deliver data in the form of
JSON (Java Script Object Notation)
// super important to get used to!
let vincent = {
    name: "Vincent",
    hasHadAllHisShots: true,
    likesFootball: false,
    age: 32,
    friends: ["Kejal", "Ben", "Phillip", "Paola"]
    introduceSelf: function(){
        console.log("Hi. I am " + name +".")
    }
}

//use dot notation to access these values

console.log(vincent.name);
console.log(vincent.friends[0])
```

Web APIs

What is an API exactly?

Application Programming Interface. It's a set of functions and objects etc that work over HTTP to let other programmers use other systems .

When people talk about APIs they mainly mean web apis, but there are tons. Like for example, a menu is like an API for users to send instructions to a kitchen (the back end!) and receive food.

Tons of web apps use third-party services like Twitter, Google Maps etc.

Today, we are going to use the Omdb API, since it is very easy to use!

OMDB API exercise

Look at the documentation on omdbapi.com

https://www.omdbapi.com/?t=rocky&apikey=2c2826e6&plot=short&r=json

Note that `"Response": "True"` bit. Try an invalid movie. False. okay.

1) Show that in the css, we hide the result and error divs by default.

2) Walk through the HTML

3) Explain `keyup` function, then `getJSON` by showing the docs

4) Show the form in action with the console open.

5) Change the keyup function to only fire if the query is 3 characters or more

```
    if (omdbData.Response == "True"){
      renderMovie(omdbData);
    } else {
      renderError();
    }
```

6) Now time to show the results on the front end. Do the title together. Then they do year and actors.

```
function renderMovie(data) {
  $('.result').show(); // this shows the div with class "result"
  $('#title').html(data.Title); //this adds the title from data into the page
  $('#year').html(data.Year);
    $('#actors').html(data.Actors);

}

function renderError() {
  $('.error').show();
}
```

7) Time for the poster. Have them look up attr in jquery docs. Manually pull of a url from the api response in the console logs, then paste into src. Now do it with the actual response.

`$('#poster').attr("src", data.Poster);`

8) Note that for when there is a bit between two valid movies, such as between Rocky, Rocky I (invalid) then Rocky II (valid), the error text persists. Fix that. `$('.error').hide()` goes in success block of api handling function.

9) what happens when you press enter? Default behavior. Pass in e to the function and call e.preventDefault();

Don't forget to