

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:

```
1 | <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.
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!

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. 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 incompatible 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 browsers.

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

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 most popular 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.

```

1  <!doctype html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width,
initial-scale=1.0">
6
7      <title>Document</title>
8  </head>
9  <body>
10     <h1>make the text all red</h1>
11     <button id="alertButton">Click me</button>
12     <script type="text/javascript">
13
14         document.getElementById("alertButton").addEventListener(
"click", function(){
15             document.querySelector("body").style.color =
"red";
16             document.querySelector("h1").innerHTML = "Now it's
red. Well done.";
17         });
18
19     </script>
20 </body>
21 </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.

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:

```
1  <script
2  src="https://code.jquery.com/jquery-3.2.1.min.js"
3  integrity="sha256-
hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4="
4  crossorigin="anonymous"></script>
5  ....
6
7  $("#alertButton").on("click", function(){
8      $("body").css('color', 'red');
9      $("h1").html("Now it is red. Thank you.");
10     });
```

Okay what the heck just happened?

Walk thorough 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.

Now to be clear, you don't *need* jQuery to do everything that jQuery helps you do. It just makes your life a lot easier, and there's a huge universe of front-end tools to make common UI elements like image sliders easy.

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 <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>
```

```
1 $( '#slideshow' ).slick();
2
3 //make images 100% width to solve one problem
4 //then solve the height problem by looking at the docs.
5
6 $( '#slideshow' ).slick({
7     adaptiveHeight: true
8 });
```

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 whooping ass.

Review of declarations, expressions and statements.

Declaring variables in JavaScript

3 ways:

```
1 // hi! I am a comment
2
3 const pi = 3.141592654;
4 // the = sign means assignment, not equality.
5 // You can get away with no semicolon, but Santa
  Clause is watching.
6 // const means the variable can never be reassigned.
7
8 let x = 42;
9 // this is the new way of doing it.
10
11 var p = 55;
12 // you will see tons of tutorials still using var
   instead of let. I won't get into the difference.
13
14 //you can declare something without setting it.
15
16 let n;
17 // not defined vs undefined
18
```

```
19 //Concatenation and addition (polymorphism)
20
21 let x = 5;
22 let y = 3;
23 let answer = x + y; // => 8
24 alert(answer);
25 let x = "five";
26 let y = "three";
27 alert(answer);
28
29 //strings and numbers
30
31 typeof 'imastring';
32 typeof 4;
33
34
```

/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

```
1 let myNumber = 1;
2 let pi = 3.14159; // ...approximately
3 // all the rules of math apply
4 // show modular division in action
5 // ++
6 // --
```

Your turn

Let's write the code to actually implement the Fahrenheit 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!

```
1 let fahrenheit = 100;
2 let celsius = (fahrenheit - 32) * 1.8;
3 console.log(celsius)
```

13:45

Strings

Escape `'` with `\'`

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

Concatenation & interpolation

`.length`

Template literals with ```

Booleans

True or false. Basically.

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

```
1 let myBooleanValue = true; // true
2 let myBooleanValueThatIsFalse = false; // false
3 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.

```
1  function greet(){
2    //this function doesn't take any input, or arguments
   as we would put it
3    console.log("why hello!")
4  }
5
6  //this is the same thing
7
8  let greet = function(){
9    console.log("why hello!")
10 }
11
12 //nothing happened.
13
14 //Right! gotta call the function
15
16 greet()
17
18 //Let's give it an argument
19
20 function greet(name){
21   let response = "Why hello, " + name + "."
22   console.log(response);
23 }
24
25
26
```

```
27 //return value vs console.log righth now these
    functions are returning undefined
28
29 //you go and bake your cake, then you clean the
    kitchen and make a new cake.
30
31
32 //in the function, its like a separate environment.
    this is called scope.
```

14:30

We do

- concatenate strings
- area of a circle
- hypoteneuse

15:15

You do

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

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

Bouncer lab

When reviewing, show one with an `if/else`, then do one with just returning the expression, then with two arguments

16:30

Lab

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

10 minutes in present String Slice as a hint

step one:

```
1 function myStarWarsName(firstName, lastName, street,
2 city) {
3     let swFirstName =
4     `${firstName.slice(0,3)}${lastName.slice(0,3)}`;
5     let swLastName =
6     `${street.slice(0,3)}${city.slice(0,3)}`;
7
8     swFirstName = swFirstName.toLowerCase();
9     swLastName = swLastName.toLowerCase();
10    swFirstName = swFirstName.charAt(0).toUpperCase() +
11    swFirstName.slice(1);
12    swLastName = swLastName.charAt(0).toUpperCase() +
13    swLastName.slice(1);
14
15    return `${swFirstName} ${swLastName}`;
16 }
```

Let's see your results

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

Dry it up:

```
1 function firstThreeLettersDowncased(str){
2   return str.slice(0,3).toLowerCase();
3 }
4
5 function capitalizeFirstLetter(str){
6   return str.charAt(0).toUpperCase() + str.slice(1);
7 }
8
9 function myStarWarsName(firstName, lastName, street,
10 city) {
11   let swFirstName =
12   `${firstThreeLettersDowncased(firstName)}${firstThreeL
13 ettersDowncased(lastName)}`;
14   let swLastName =
15   `${firstThreeLettersDowncased(street)}${firstThreeLett
16 ersDowncased(city)}`;
17   return `${capitalizeFirstLetter(swFirstName)}
18   ${capitalizeFirstLetter(swLastName)}`;
19 }
```

Add user interaction that bind the html to JS code:

try both submit and keyup

```
1 $( "#sw-name-inputs" ).on( 'submit', function(event){
2   event.preventDefault();
3   let starWarsName = myStarWarsName( $( "#first-
4 name" ).val(), $( "#last-
5 name" ).val(), $( "#street" ).val(), $( "#city" ).val());
6   $( "#answer" ).html( starWarsName );
7 });
```

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

```
1 // objects are basically key-value pairs
2 // the values are primitives.
3 // Objects are the bedrock of most web APIs that
  deliver data in the form of JSON (Java Script Object
  Notation)
4 // super important to get used to!
5 let vincent = {
6     name: "Vincent",
7     hasHadAllHisShots: true,
8     likesFootball: false,
9     age: 32,
10    friends: ["Kejal", "Ben", "Phillip", "Paola"]
11    introduceSelf: function(){
12        console.log("Hi. I am " + name + ".")
13    }
14 }
15
16 //use dot notation to access these values
17
18 console.log(vincent.name);
19 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/)

<https://www.omdbapi.com/?t=rocky&apikey=c12529c3&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

```
1      if (omdbData.Response == "True") {  
2          renderMovie(omdbData);  
3      } else {  
4          renderError();  
5      }
```

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

```

1 function renderMovie(data) {
2     $('#result').show(); // this shows the div with
    class "result"
3     $('#title').html(data.Title); //this adds the title
    from data into the page
4     $('#year').html(data.Year);
5     $('#actors').html(data.actors);
6
7 }
8
9 function renderError() {
10    $('#error').show();
11 }

```

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