Programming for Non-Programmers Bootcamp — Day 2

Deploy to the web with github pages

45 mins

Make sure that you have a Github.com account.

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 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.

Holy smokes!

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.

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 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 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

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!

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.

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!
```

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.

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???

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/`

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
});
```

Wow!

`<br />` for lunch

Writing our own JavaScript

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.

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;

```

You do:

https://codepen.io/trivett/pen/yVvPaq

`/js-practice/declarations.js`

Try it in atom, 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

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!

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

Strings

Escape `'` with `\'`

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

Concatenation

`.length`

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

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.

```

You do

- concatenate strings

- area of a circle
- hypoteneuse
- create a function for temperature conversion that lets you convert any temperature.

```
1
2  function myConcatenate( firstStr, secondStr, thirdStr )
   {
3      let answer = firstStr + " " + secondStr + " " +
   thirdStr;
4      console.log(answer);
5  }
6
7  myConcatenate('I', 'am', 'iron man'); // 'I am iron
   man'
```

Collections

Arrays

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

Some Array methods:

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

```
a = ["a", "b", "c"];

a.forEach(function(element) {
    console.log(element);
});
```

Objects

```
// just primitives
const name = "Vincent;
let hasHadAllHisShots = true;
let likesFootball = false;
let age = 32;
let friends = ["Kejal", "Ben", "Phillip", "Paola"]
let introduceSelf = function(){
  console.log("Hi. I am " + name +".")
}


//imagine doing this for more than one person!

// 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])
```

Conditionals and Operators

JavaScript Statements use a set of operators

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

For and While loops

There is a handy way to loop through arbitrary data.

for loops

```
for (var i = 0; i < 9; i++) {
   console.log(i);
   // more statements
}
```

while loops

```
var n = 0;
var x = 0;

while (n < 3) {
  n++;
  x += n;
}
console.log(x);
```

Avoid creating infinite loops!

Check for understanding

Fizzbuzz exercise

```
hoping that it is about 3:30 p.m. at this point . if its only like 2 then do the madlibs thing
```

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.

First step to work with an API is to look at the docs. RTFM!

OMDB API exercise

Look at the documentation on omdbapi.com

http://www.omdbapi.com/?t=zootopia

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();
    }
```

Remove comments

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();

---

start:

```
$('#movie-search-form').keyup(function() {
  // this function fires on keyup. as you type in the searchbar, it fires searchIMDB function
```

```
  $('.result').hide(); // while we wait for the API to respond, we hide the last search result for that split second
  searchIMDB(this.query.value); // calling the next function with the text the user inputs
});

function searchIMDB(query) {
  // this is the function that makes the request with jQuerys's getJSON

  $.getJSON('http://www.omdbapi.com/', {
    t: query, // this query comes from the argument passed in parens above, which
    plot: "short",
    r: 'json'
  }, function(omdbData) { //this function fires after the network request finishes.
    if (omdbData.Response == "True"){
      console.log(omdbData);
    // Things worked! Show the movie data by calling the renderMovie function, pass in the omdbData variable
    } else {
      // render an error here.
      console.log(omdbData);
    }
  });
}

function renderMovie(data) {
  $('.result').show(); // this shows the div with class "result"
  // we aren't done here! we need some more code to show the data and the
```

```
28
29  }
30
31  function renderError() {
32    $('.error').show();
33  }
34
```

finish:

```
$('#movie-search-form').keyup(function(e) {
  e.preventDefault();
  if (this.query.value.length > 2){
    $('.result').hide();
    searchIMDB(this.query.value);
  }
});

function searchIMDB(query) {
  $.getJSON('http://www.omdbapi.com/', {
    t: query,
    plot: "short",
    r: 'json'
  }, function(omdbData) {
    if (omdbData.Response == "True"){
      $('.error').hide();
      renderMovie(omdbData)
    } else {
      renderError()
    }
  });
}

function renderMovie(data) {
  $('.result').show();
  $('#poster').attr("src", data.Poster)
  $('#title').html(data.Title);
  $('#year').html(data.Year);
  $('#actors').html(data.Actors);
}

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