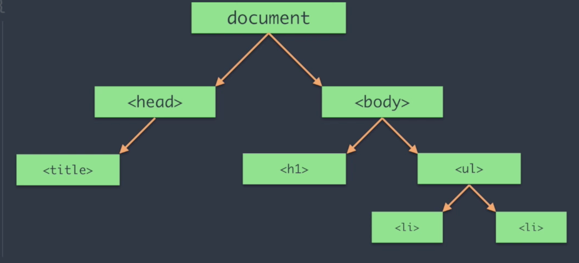
**JS Tutorials**

**Section 1**

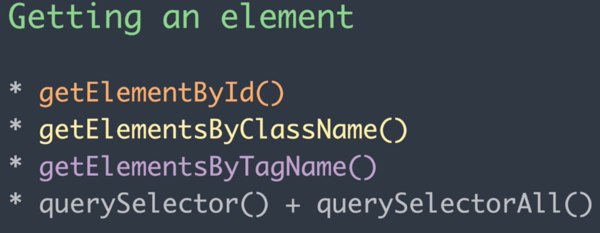
DOM:

The bridge btw the HTML and JS



Section 2

JS Common Queries



// Getting an Element()

document.getElementById('head') // displays all id with head

document.getElementsByClassName('para')//displays all class

document.getElementsByTagName('p')//displays all Tag

document.querySelector('.para')//displays the first one

document.querySelectorAll('.para')//displays all

document.querySelectorAll('.para')[0].innerHTML= 'Changed it' //changing the text

//Creating Things in the Documents

document.createElement()// to create an element

var p = document.createElement('p')

p.innerText = "This is me"

document.body.appendChild(p)

document.createAttribute() //to create attributes like id and class, styles etc

var att = document.createAttribute('id')

att.value = "created"

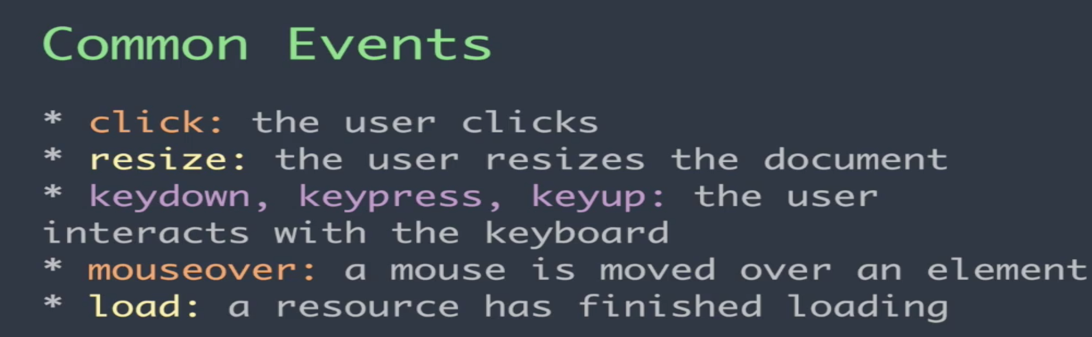
p.setAttributeNode(att) //sets the new p tag to have an id called created

//<p id="created">This is it</p>

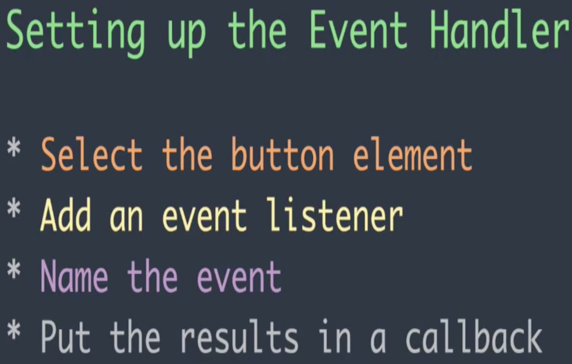
**Section 3**

Events and Call-backs

Events: are occurrences that happens in a browser



Callbacks: is an event handler. It’s a function that runs when an event occurs

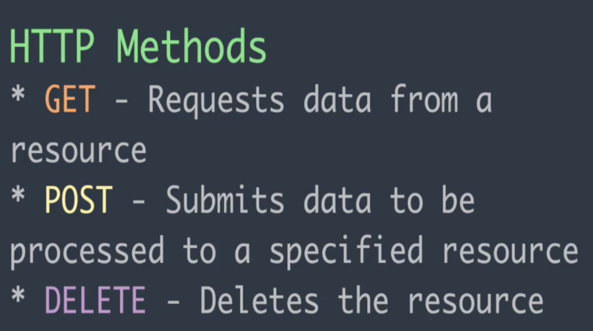
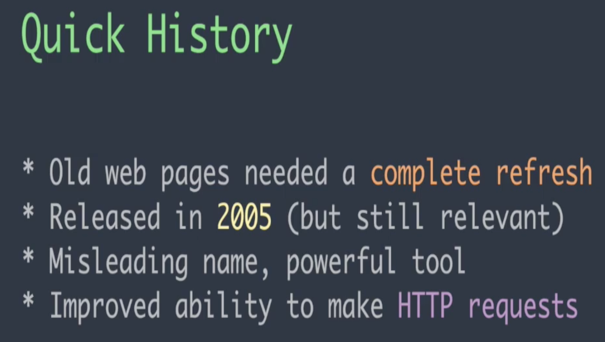
load is the event, function is the call-back. So the function runs when the event occurs

got to folder 3 for examples

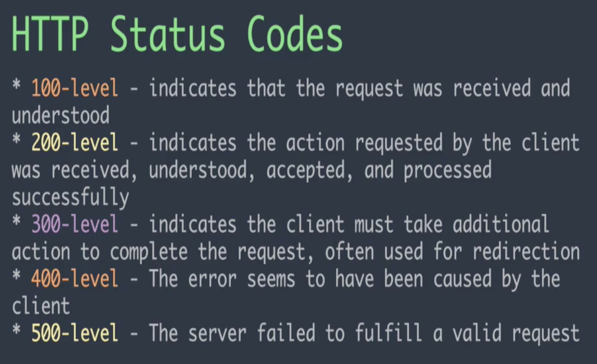
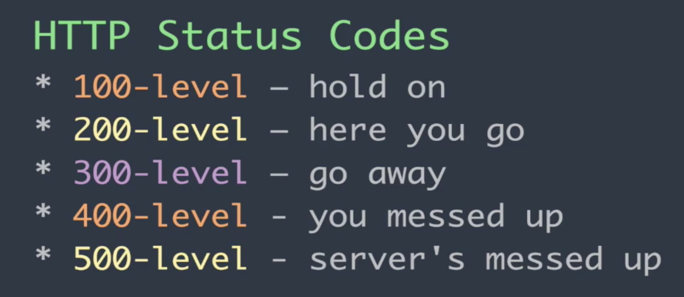
**Section 4**

AJAX & HTTP Requests

AJAX makes asynchronous request: meaning other things can still be done while waiting for the response to our requests



GET: Pulls, Post: creates Delete: Remove

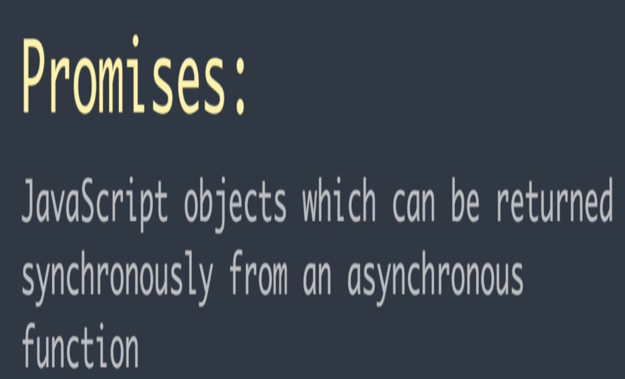
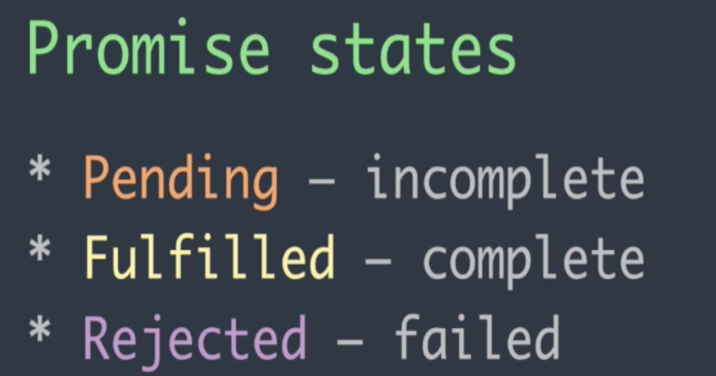
 

PROMISES:

<https://developers.google.com/web/fundamentals/primers/promises>

<https://dev.to/damcosset/i-promise-i-wont-callback-anymore-cp3>

<https://developers.google.com/web/updates/2015/03/introduction-to-fetch>

**FETCH()**

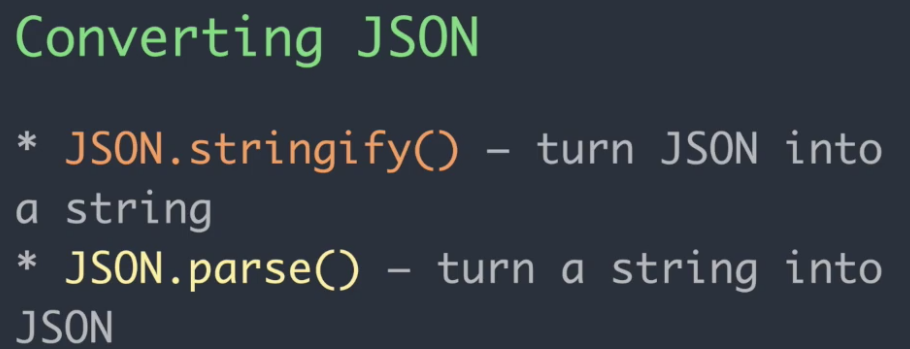
This is a better alternative for events and call backs.

Fetch uses promises to get the desired output.

Check folder “Ajax(4)” to see the example

**SECTION 5**

**JSON**



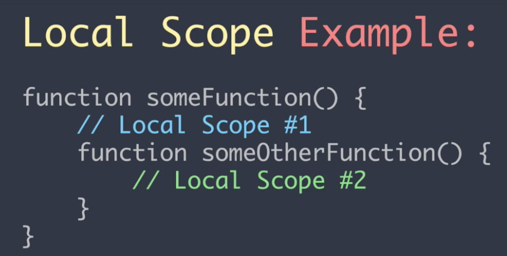
check folder 5 json

**SECTION 6**

SCOPES AND THE VARIABLE THIS

TWO TYPES OF SCOPE

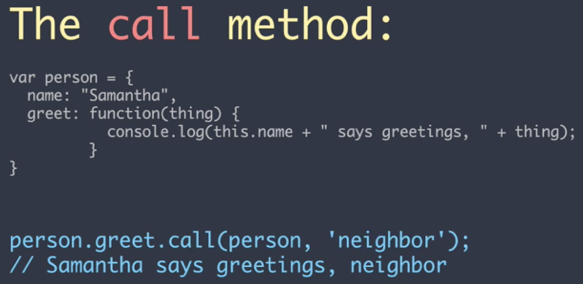
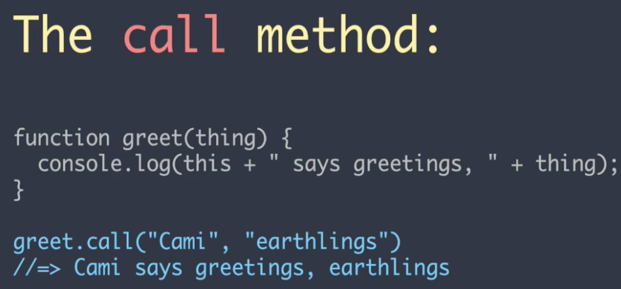
1. LOCAL SCOPE: IN A FUNCTION
2. GLOBAL SCOPE: NOT IN A FUNCTION

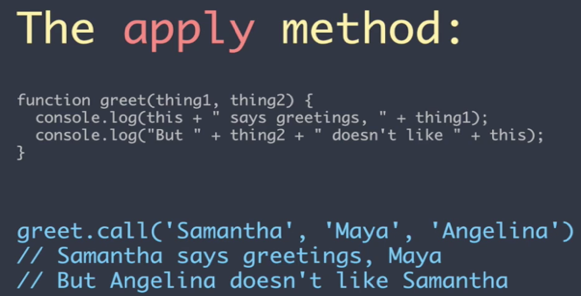
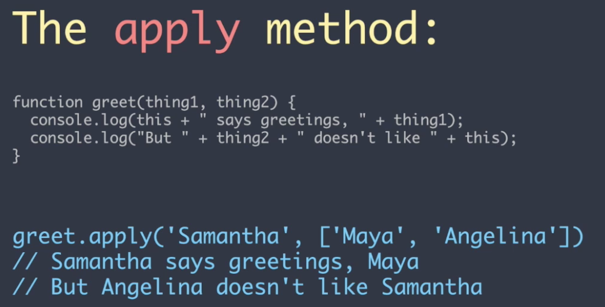


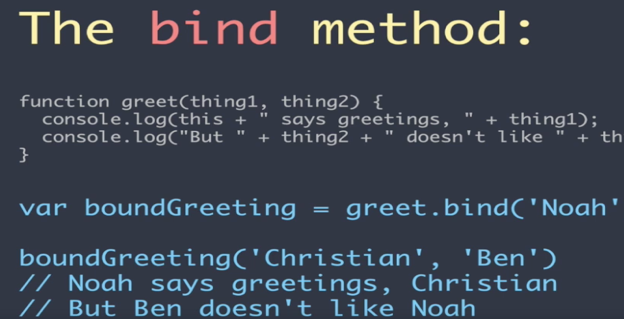
THIS VARIABLE

THIS VARIABLE HAS 3 METHODS

1. CALL: USED TO CALL USING THIS AS THE FIRST VARIABLE
2. APPLY: FOR MULTIPLE USING ARRAY, THIS VARIABLE COMES FIRST AND NOT IN THE ARRAY
3. BIND: ASSIGN THIS TO A VARIABLE



<https://medium.com/@omergoldberg/javascript-call-apply-and-bind-e5c27301f7bb>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this>

<https://codeplanet.io/javascript-apply-vs-call-vs-bind/>

**SECTION 7**

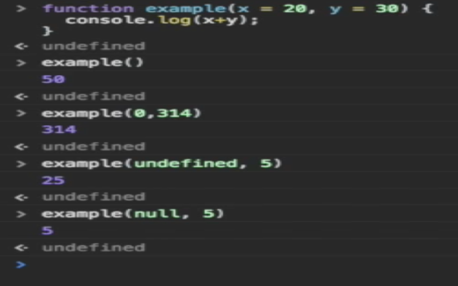
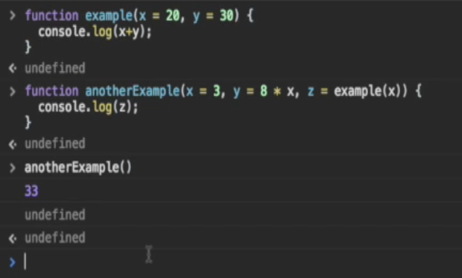
VAR, LET AND CONST

<https://hackernoon.com/js-var-let-or-const-67e51dbb716f>

**SECTION 8**

DEFAULT VALUES

Basic default values it can also take expressions and functions

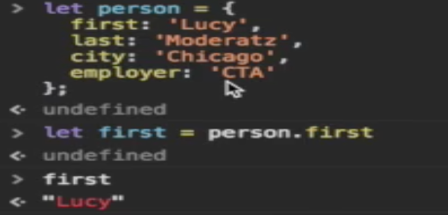
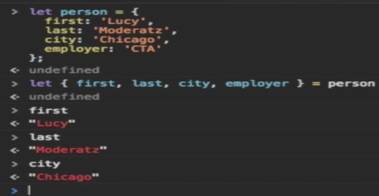
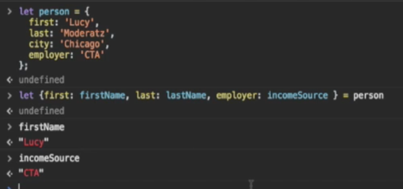
 

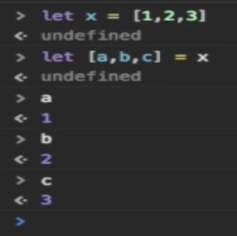
**SECTION 9**

DESTRUCTURING

Is an expression that allows us to pull data from objects and arrays into their own variable

Different ways of destructuring

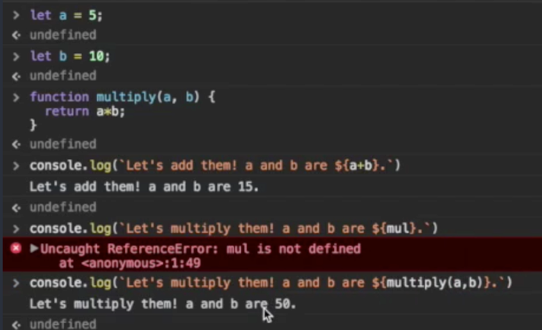
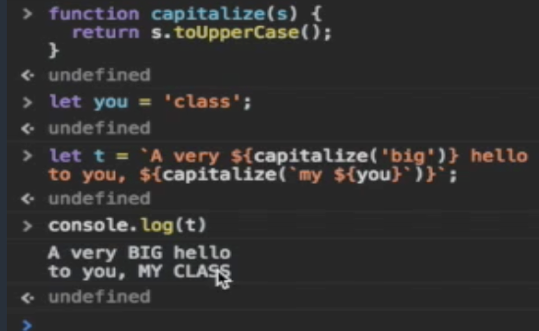


**SECTION 10**

LITERALS

TWO TYPES OF LITERALS

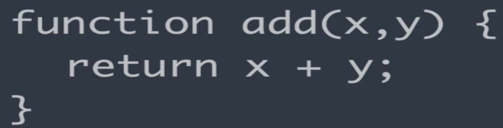
* OBJECT LITERALS
* TEMPLATE LITERALS

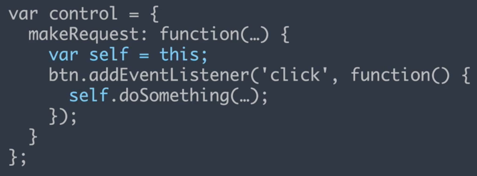
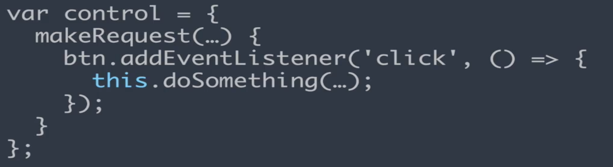
 

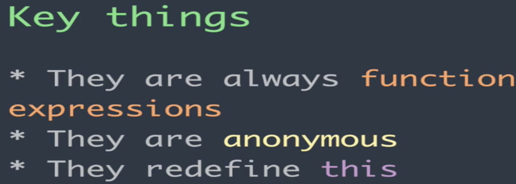
**SECTION 11**

ARROW FUNCTION

BEFORE NOW(ES6)



**INSTALL DEPENDENCIES**

**Package managers**

install nodejs, it automatically installs npm

Brew install yarn

Npm install -g grunt-cli

Npm install -g gulp-cli

Npm install –g webpack

**HOW TO USE NPM**

Create new project

Mkdir

Npm init

**Dependencies:**

Npm install vaca

**HOW TO USE YARN**

*Create new project*

Mkdir

yarn init

**Dependencies:**

yarn add dogefy

**GRUNT**

Grunt is a task manager, it manages your tasks and executes them for you

*Create new project*

Mkdir

npm init

npm install grunt

create Gruntfile.js

add default task to gruntfile.js

run grunt

**Dependencies:**

npm install grunt-contrib-watch grunt-contrib-uglify

**GULP**

Gulp is a streaming build system similar to grunt and it handles automation for JavaScript

*Create new project*

Mkdir

npm init

npm install gulp

create gulpfile.js

add default task to gulpfile.js

run gulp

**Dependencies:**

for the portfolio site

npm install handlebars gulp-compile-handlebars gulp-rename

the partials folder

are pieces of our larger handlebar templates in index.hbs

the hbs files are files that provide templating options for more complex sites in HTML formats.

All of the tags in the index.hbs file are defined in the partials folder

A task is created which compiles the index.hbs into a index.html file.

The task is named html.

Run gulp html

**VOCABULARY**

**1.Project**

First vocabulary project is the code base you're working with whether it be an application a framework a plug in or a library.

**2.Module**

A module is a unit of source code usually just a file sometimes just a function within a file that provides some functionality. A module may depend on another module and that's known as a dependency

**3.Package**

a package is a collection of one or more modules that is published somewhere so that others can use it whether it's published online or elsewhere

**4.Package Manager**

a package manager is a tool that installs a package into your system. We installed and PM and yarned earlier and those are package managers.

Speaking of which these are the technologies we learned NPM is node package manager.

**Webpack is a module bundler for JavaScript applications.**

**REACT**

*How to create a react template app*

npm install -g create-react-app

create-react-app name of app

follow the dirxn to run the app

then ctrl c

yarn add mousetrap

mousetrap is a lib for detecting keyboard commands

1. **Components Lifecycle Methods**

Each component in react has a set of lifecycle method. They define how a component is Initialized, Lives and Removed in a react app

**First Phase: Initialization:**

It has only one method called constructor(). Its constructs the component with the initial state

**Second Phase: Mounting:**

Has 3 methods:

componentWillMount():is executed right before the component renders on the page. Its mostly used to configure your component

render(): sticks the component into the user interface in your app

componentDidMount():occurs right after the component is rendered. That’s where an Ajax request is called to load data into the component or add event listeners

**Third Phase: Updating:**

Has several methods:

componentWillReceiveProps():The first is component will receive props. This means that properties from a parent component will change the current components state which again

componentShouldUpdate():The next method is component should update. It controls exactly what our component will render. It's similar to component will receive prop's but it returns a Boolean saying whether or not the component

componentWillUpdate():It says that we should only update the component if the properties we care about change. It is a rarely used method that's for providing app configuration. If the previous two functions decided that we should update

render():if these three functions have said OK it's time to render it updates the component in the user interface

componentDidUpdate():This is kind of the update version of componentDidMount() in the mounting section. It happens right after the updated render function

**Last Phase: Unmounting**

that's when you're saying OK this component is not going to exist anymore let's take it off the page and it has the one function componentWillUnmount(): the component is about to go away.

This is where you remove event listeners or network requests coming from said component.