**Bootcamp Week 2:**

Day 6:

New Terminology:

- Jest

- Servers

- Ports

- Express.js

- Supertest & Cheerio

- DNS – Domain Name Server translates Domain names (urls) into IP addresses.

- DHCP

- IP - An Internet Protocol address (IP address) is a numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication. An IP address serves two principal functions: host or network interface identification and location addressing.

- \_\_dirname – directory name, the directory that the file is in.

- path.join

- sendFile & send

- req.query & req.params

- .listen – listening to the server port

- ${target} – string templates

- GET() & POST() request – server requests, get() is requesting information and posting is submitting info.

**Asynchronous** – when things happen at once. When code is being tested at virtually the same time.

**Blocking** – when code is running, there’s other code that will be blocked until the first part of code has run. Opposite to Asynchronies.

**Synchronies Callbacks:**

- Generic Callbacks (functions)

math.js:

function doMath(op1, op2, operation) {

return operation(op1 + op2)

}

Passing in the two operands and passing in the operation. doMath is going to run this operation with op1 and op2.

function add (op1, op2) {

return op1 + op2

}

defining the function add

const result = add (2, 7)

calling the function – we know this because we are executing the function with parentheses.

Const result = doMath(2, 7, add)

In this example we are calling the function doMath() but the add function is just being PASSED through the function, even though it is a function itself.

console.log(result)

Defining Functions 🡪

Calling Functions 🡪

Passing Functions 🡪

**Asynchronous Callbacks:**

I/O – input/output (Asynchronies)

Const A = 4 (this is saved in memory (RAM) and takes microseconds to access this)

When you’re accessing something from the fileSystem (hard drive) it takes milliseconds.

When you’re accessing something from the Network it takes seconds.

filestuff.js

**fs.readdir(path[, options], callback)#**

* path [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type) | [<Buffer>](https://nodejs.org/api/buffer.html#buffer_class_buffer) | [<URL>](https://nodejs.org/api/url.html#url_the_whatwg_url_api)
* options [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type) | [<Object>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object)
  + encoding [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type) **Default:** 'utf8'
* callback [<Function>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function)
  + err [<Error>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Error)
  + files [<string[]>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type) | [<Buffer[]>](https://nodejs.org/api/buffer.html#buffer_class_buffer)

Asynchronous [readdir(3)](http://man7.org/linux/man-pages/man3/readdir.3.html). Reads the contents of a directory. The callback gets two arguments (err, files) where files is an array of the names of the files in the directory excluding '.' and '..'.

The optional options argument can be a string specifying an encoding, or an object with an encoding property specifying the character encoding to use for the filenames passed to the callback. If the encoding is set to 'buffer', the filenames returned will be passed as Buffer objects.

const fs = require(‘fs’)

fs.readdir(\_\_dirname, showFiles) //this happens first and keeps going then reads console.log immediately after. Unlike Synchronous operations.

console.log(‘I am here’)

function showFiles (err, files) {

if (err) {

return console.error(err)

}

files.forEach(file => console.log(file))

}

It’ll call showFiles once it has finished reading the directory.

When it hits readDir, because its asynchronous it makes everything happen at the same time – so it branches off to fileSystem while everything is still going (eg. The console.log()) and then the branch merges in the same time as the original run finishes. So theres different things happening at once. Whereas in synchronous operation, when it hit readDir, it would stop the operation to run that and then keep going, rather than continuing to run other stuff at the same time. Asynchronous is does a lot more in the same amount of time as Synchronous.

**fs.readdirSync(path[, options])**

* path [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type) | [<Buffer>](https://nodejs.org/api/buffer.html#buffer_class_buffer) | [<URL>](https://nodejs.org/api/url.html#url_the_whatwg_url_api)
* options [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type) | [<Object>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object)
  + encoding [<string>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type) **Default:** 'utf8'

Synchronous [readdir(3)](http://man7.org/linux/man-pages/man3/readdir.3.html). Returns an array of filenames excluding '.' and '..'.

The optional options argument can be a string specifying an encoding, or an object with an encoding property specifying the character encoding to use for the filenames passed to the callback. If the encoding is set to 'buffer', the filenames returned will be passed as Buffer objects.

In synchronous function (blocking function) its going to use the return keyword and get a return, and what it returns we need to save into a variable. In Asynchronous you don’t get this, theres no assignment operator – we get it in the callback, it’ll pass the value you want into the callback and then you will get it that way.

How to tell if your function is Async or Sync?

When you have a callback you generally assume its an async function.

**Testing Asynchronous Functions:**

**index.js:**

module.exports = {

readFolder,

showFiles

}

function readFolder (path, cb) {

fs.readdir(path, cb)

}

function showFiles (err, files) {

if (err) return console.error(err)

files.forEach(file => console.log(file))

}

readFolder(\_\_dirname, showFiles)

**index.test.js:**

/\* global test expect \*/

const index = require(‘index’)

test(‘Test harness is working’, () => {

expect(true).toBeTruthy()

})

test(‘readFolder calls the callback’, ()

index.readFolder(‘p’, assert)

function assert (err, files) {

expect(err).toBeFalsy()

expect(files.length).toBe(5)

}

})

**Bootcamp Day 7:**

New Terminology:

- Asynchronous Functions – you have to put your assertions in your callback for Async functions. You don’t access the FS with Async.

- Synchronous Functions - .map, .filter, .reduce are synchronous functions that take callback functions. Blocking.

- Callbacks

- I/O

- Time-slicing

- fs.readdir – specific to node

- fs.readfile - specific to node

- fs.writefile - specific to node

-fs.appendfile - specific to node

- prompt.

- Dependency Injections:

dependency injection is a technique whereby one object supplies the dependencies of another object. A dependency is an object that can be used (a service). An injection is the passing of a dependency to a dependent object (a client) that would use it. The service is made part of the client's state.[1] Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern.

**Handlebar Framework:**

To install:

Npm init –y

Yarn add express express-handlebars

**index.js:**

const server = require(‘./server’)

const port = 3000

server.listen(port, () => {

console.log(Server is listening on port’, port)

})

**server.js:**

const express = require(‘express’)

const hbs = require(‘express-handlebars’)

const server = express ()

module.exports = server

server.engine(‘hbs’, hbs({

extname: ‘hbs’

}))

server.set(‘view engine’, ‘hbs’)

server.get(‘/’, (req, res) => {

res.send(‘Hello world!’)

})

// run node index to test if Server is listening on port 3000