Code Fellows

401 Advanced Javascript

seattle-javascript-401n14

Meet Your Instructor

Sonia Kandah (she / her)

sonia@codefellows.com

- Microsoft, startups, indie games
- Art, design, illustration
- Cats, games, comedy, crafting

Talk About You!

- Your name + your pronouns
- Where you're from
- Why you're here (aside from getting a job!)
- One fun fact about you
- The names of everyone who went before you 😮

Welcome to Night (+ Saturday) Class

- Spread out = easier to forget, so co-working and self-review is crucial!
- Co-working nights Mon and Thurs for help with labs and other assignments
- We can do this!

Course Outline

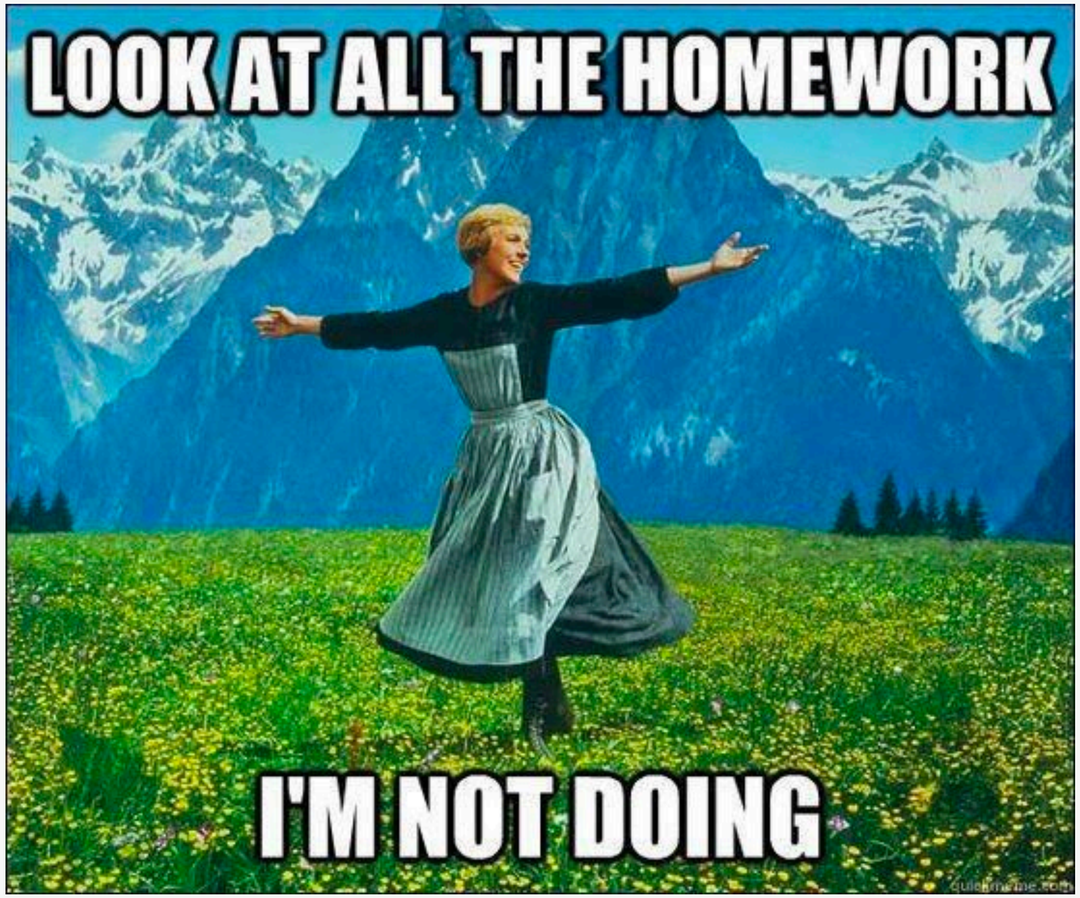
Week 01	Week 02	Week 03	Week 04	Week 05	Week 06
1. Node Ecosystem	2. Classes, Inheritance and Functional Programming 3. Async	4. Data Modeling 5. Data Modeling with NoSQL Databases	6. HTTP and REST 7. Express	8. Express Routing and Connected API 9. API Server	10. Authentication 11. OAuth
Week 07	Week 08	Week 09	Week 10	Week 11	Week 12
12. Bearer Authorization 13. Access Control (ACL)	14. API Server Revisited 15. Event Driven Applications	16. TCP Protocol 17. Socket.io	Midterm Project Week	Midterm Project Week	18. Component Based UI 19. React Testing and Deployment
Week 13	Week 14	Week 15	Week 16	Week 17	Week 18
20. Props and State 21. Routing and Component Composition	Holidays	22. Hooks API	23. Custom Hooks: Socket and Fetch 24. Context API	25. Login and Auth 26. Application State	27. Dynamic Forms 28. Remote APIs
Week 19	Week 20	Week 21	Week 22		
29. Combined Reducers, Drag and Drop 30. React Native	31. Advanced React Native 32. Gatsby and Next	Final Project Week	Final Project Week		

Holidays

Thanksgiving	Monday NOV 25 2019 Co-working	Tuesday NOV 26 2019 Lecture	Wednesday NOV 27 2019 Lab	Thursday NOV 28 2019 Thanksgiving/ Co-working	Friday NOV 29 2019	Saturday NOV 30 2019 Lecture + Lab	Sunday DEC 01 2019
Christmas + New Years	Monday DEC 23 2019 Co-working	Tuesday DEC 24 2019 Lecture	Wednesday DEC 25 2019 Lab	Thursday DEC 26 2019 Co-working	Friday DEC 27 2019	Saturday DEC 28 2019 Lecture + Lab	Sunday DEC 29 2019
	Monday DEC 30 2019 Co-working	Tuesday DEC 31 2019 Lecture	Wednesday JAN 01 2020 Lab	Thursday JAN 02 2020 Co-working	Friday JAN 03 2020	Saturday JAN 04 2020 Lecture + Lab	Sunday JAN 05 2020

Assignment Outline

- **Code Challenge** - Due Mon/Thurs @ Midnight
- **Readings** - Due before Tues/Sat class
- **Learning Journal** - Due Wed/Sat @ Midnight
- **Labs** - Due before next Tues/Sat class
- **Feedback Survey** - Due Sundays @ Midnight
- **Career Coaching** - Due Sundays @ Midnight



Course Calendars

- Class Schedule - <Click Here>
- Assignment Schedule - <Click Here>
- If you have issues accessing these calendars, contact your Instructor!

Your Course Repo

- You can find all the content for the course in our git repo
- Your pre-work should have asked you to fork the repo
- Now we need to do the work to keep that repo up-to-date!
- First, link your forked repo to the class repo

```
cd <my-forked-repo-directory>

git remote set-url upstream https://github.com/codefellows/seattle-javascript-401n14

git remote -v [to check your work]
```

- Then, use the following command to get it in-sync with the master

```
git pull upstream master
```

** Try to run this every class, or at least once a week

Code Fellows

Class 01: Node Ecosystem

seattle-javascript-401n14

What is Node.js?

What is Node.js?

Also referred to as “Node”, Node.js is a run-time **Javascript environment** that executes Javascript code outside of a browser. Even though it has a “.js”, it’s not a single Javascript file (the creators just named it that way 😒)

```
soniakandah … class-01 demo envs master php app.php
Hi there! You're running me in the PHP environment.
soniakandah … class-01 demo envs master perl app.pl
Howdy! You're running me in the Perl environment.
soniakandah … class-01 demo envs master python app.py
Hey Hey! You're running me in the Python environment.
soniakandah … class-01 demo envs master node app.js
Hello! You're running me in the Node environment.
soniakandah … class-01 demo envs master
```

Demo

class-01/ demo/envs

We run code in an **environment**.

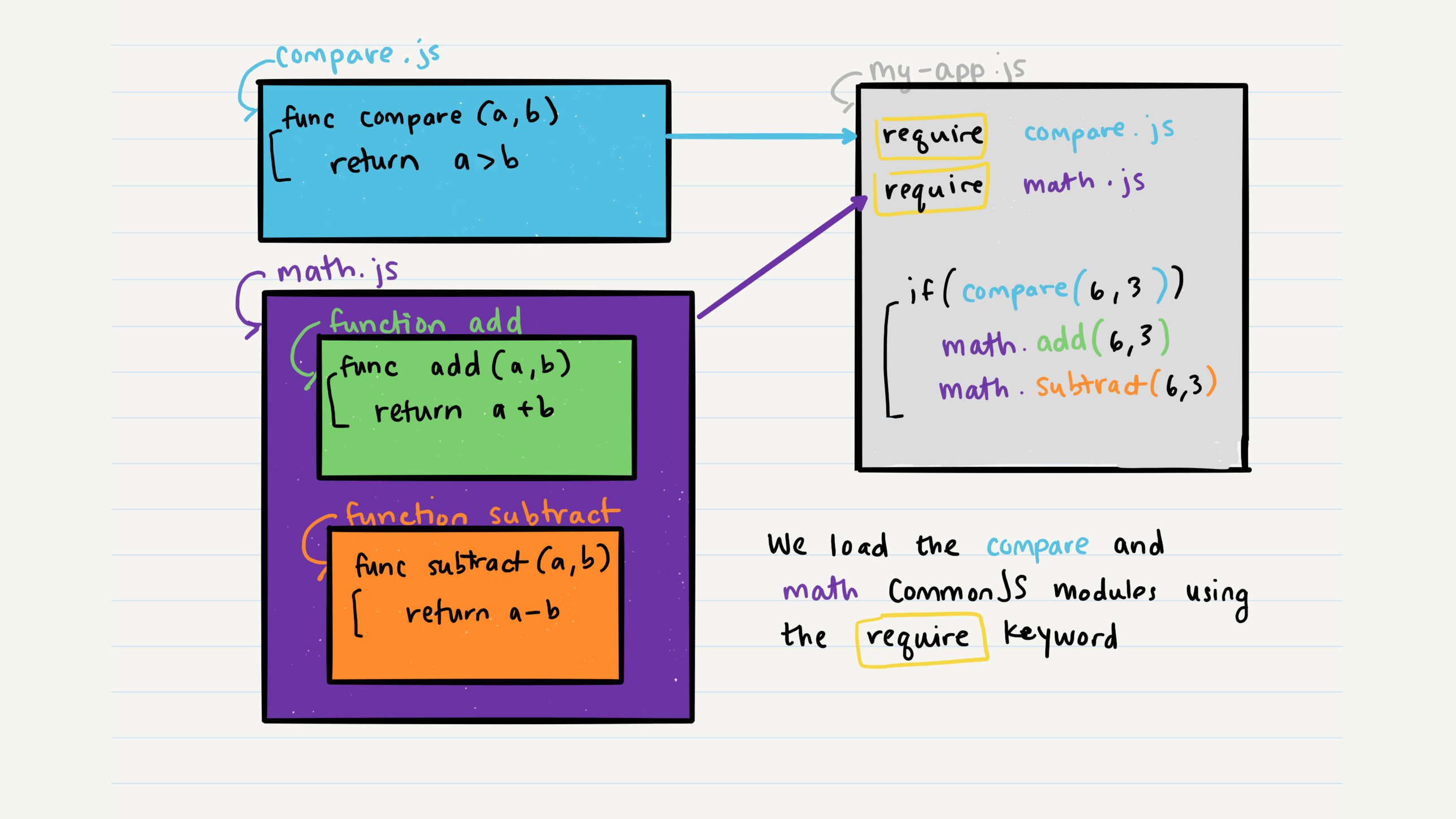
Environments interpret our code and do some administrative work so it runs correctly.

Why Are We Using Node?

- It runs Javascript code - one language for the front-end AND back-end
- Super-fast when it runs code
- Has a lot of **packages** that developers have made that you can plug-and-play
- Can handle “**deploying**” your code really well, so that you can create powerful web apps like Facebook 😮

Wait, Packages?

- **Packages** = **CommonJS Modules** = **Modular** code = code that is independent, and can be incorporated into almost any application easily
- When using node, we can easily write our own CommonJS Modules
- This lets us break up our code into small chunks in distinct files, or lets us download someone else's code and add it to our projects easily


compare.js
func compare(a,b)
return a>b
math.js
function add
func add(a,b)
return a+b
function subtract
func subtract(a,b)
return a-b
my-app.js
require compare.js
require math.js
if(compare(6,3))
math.add(6,3)
math.subtract(6,3)
We load the compare and math CommonJS modules using the require keyword

```
// We use module.exports to define this
// function as a CommonJS module that we're
// exposing to whoever wants to use add this
// file to their project

module.exports = exports = name => {
    console.log(`Hello ${name}`);
};

```

```
soniakandah … class-01 demo modules-single-export node index.js
[Function: exports]
Hello john
soniakandah … class-01 demo modules-single-export master
```

Demo

class-01/demo/ modules-single-export

A **CommonJS** module can be used in a different file using the `require` keyword.

You create a CommonJS module by setting `module.exports` equal to some content.

```
3  // We want to export multiple things this time.
4  // Lets say that this module returns an OBJECT
5  // instead of a function this time
6  module.exports = exports = {};
7
8  // We want to add some data to this exported object.
9  // Lets start simple with a name for the module!
10 exports.myModuleName = "401js Hello World Module";
```

```
soniakandah … class-01 demo modules-multiple-exports master node index.js
{ myModuleName: '401js Hello World Module',
  sayHello: [Function],
  sayGoodbye: [Function] }
401js Hello World Module
Hello, Sonia
Goodbye
soniakandah … class-01 demo modules-multiple-exports master
```

Demo

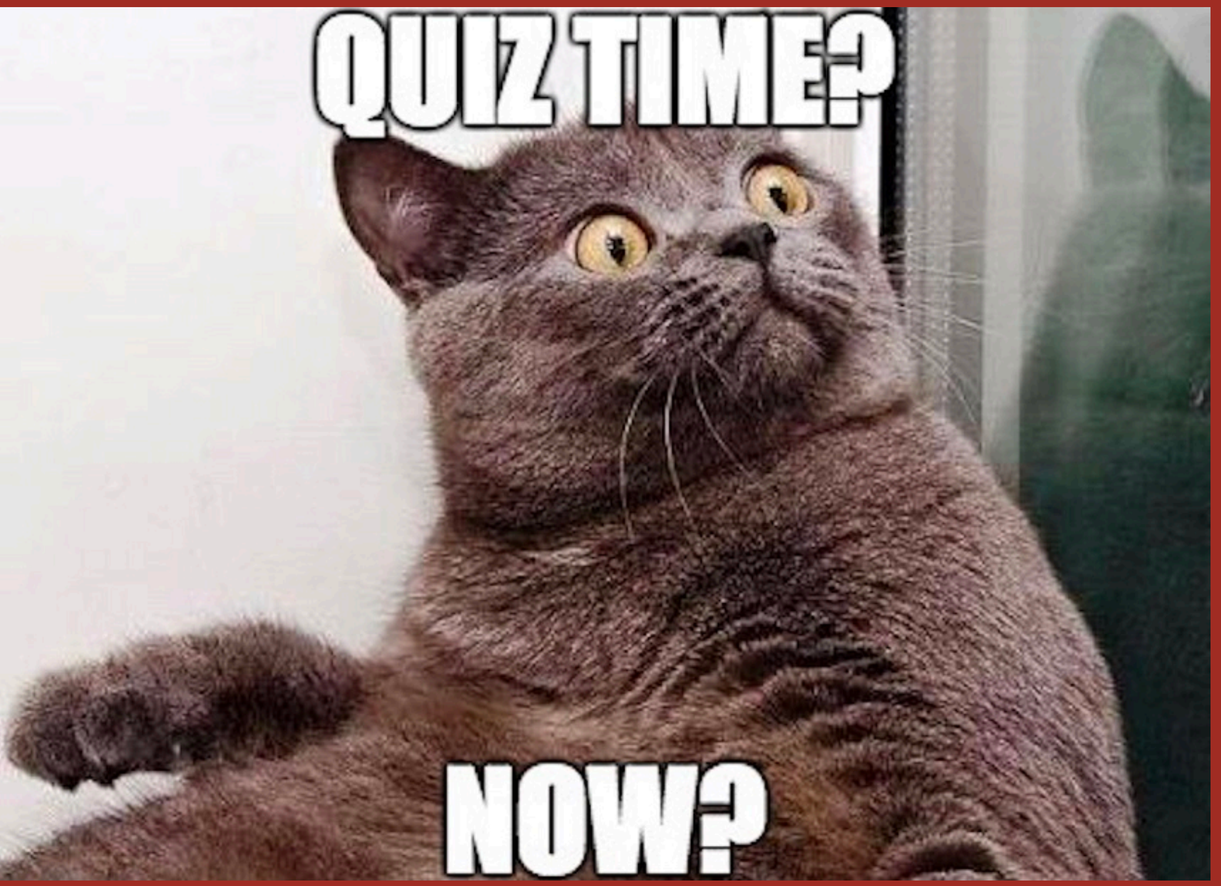
class-01/demo/modules-multiple-exports

You can export more than just a single function, you can export variables as well.

You keep some things in the file **internal** - they will never be exported or usable by other files.

`module.exports = exports = …`

This is how we create a CommonJS module. `module` is an object that all JavaScipt files have. `module.exports` is usually an empty object unless we manually set it equal to something within our file. We set `module.exports` equal to `exports`, so that we can use the keyword exports as a shortcut for `module.exports` (coders love to type less!)



Vocab Review!

What is Node.js?

What is Node.js?

Also referred to as “Node”, Node.js is a run-time **Javascript environment** that executes Javascript code outside of a browser. Even though it has a “.js”, it’s not a single Javascript file (the creators just named it that way 😒)

What does modular mean?

What does modular mean?

When code is modular, it usually means it’s small, independent and has a singular purpose. Modular code is designed to be **reused** by multiple different files or applications.

What is a CommonJS Module?

What is a CommonJS Module?

A CommonJS module is a JavaScript file that uses `module.exports` to define what is exported and usable by external files.

What is a package?

What is a package?

A package is all the content that you need in order to use a module. At a bare minimum, this means the file where the module is defined. Packages can also include supporting CSS, HTML, image, or other media files. Packages can even include other JavaScript files that support the primary module code, such as test files, internal functions, etc.

What does it mean for something to be internal?

What does it mean for something to be internal?

Typically, this means that it cannot be referenced, used, or tampered with outside of the current file or function.

Whew, time for a break...

But wait!

It’s not over!!

What Test Driven Development (TDD)?

What Test Driven Development (TDD)?

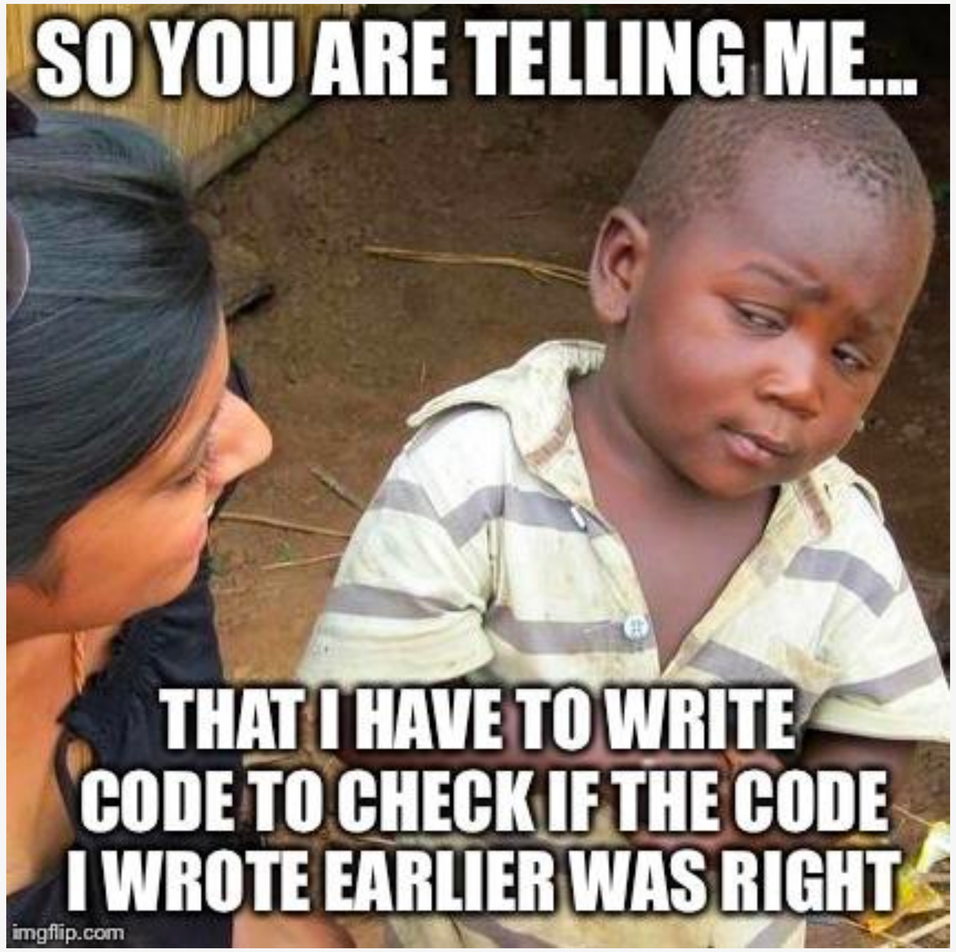
Test Driven Development (TDD) is a practice where you write a test very early in your coding process, and then try to write the smallest chunk of code that will pass that test. TDD encourages modular coding and also verifies that your code is correct early on, letting you focus on making your code better, cleaner and more smart!

More about TDD

- Make code in small chunks (**units**), and write tests for those chunks right away
- Three steps:
  - **RED**: Write tests before code is finished. This means most of your tests will fail
  - **GREEN**: Refine your code so that the tests all pass
  - **REFACTOR**: Make your code neater and more compact, checking your tests along the way

How Do We Write Tests?

- We can use different **packages** to help us test our code
- **Jest** is a **package** that will go through your code, run it, and compare the output of the code with what you define as the “correct answer”. If the comparison fails…it’s bug squashing time! 🐞



More About Packages!

- When you use **npm install** to download packages, your code app creates a **package.json**, which defines **dependencies**
- You'll see a lot of versions on these packages, which follow the format: ##.##.## (Major version #, minor version #, patch #)
- **npm init** can help you create the package.json

Setting up a TDD Project

- `mkdir <my-project-directory-name>`
- `npm init`
    - Go through all the terminal prompts, filling out the information you like, or just hitting enter to accept the defaults
- `npm install jest`
- `touch index.js`
- `touch index.test.js`

```
// Require the module we're testing
const hello = require("../src/hello.js");

describe("Hello", () => {
  it("requires one param", () => {
    let message = hello.sayHello();
    expect(message).toBeNull();
  });
```

```
soniakandah … class-01 demo modules-tested master + 1 … 1 npm test

> hw-tested@1.0.0 test /Users/soniakandah/cf/js-401n14/curriculum/class-01/demo/modules-tested
> jest --verbose --coverage

 PASS  test/hello.test.js
  Hello
    ✓ requires one param (4ms)
    ✓ only allows one param
    ✓ does not allow numeric values
    ✓ does not allow arrays as a param
    ✓ does not allow objects as a param (1ms)
    ✓ works when given a word

----------|----------|----------|----------|----------|-------------------|
File      |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
```

Demo

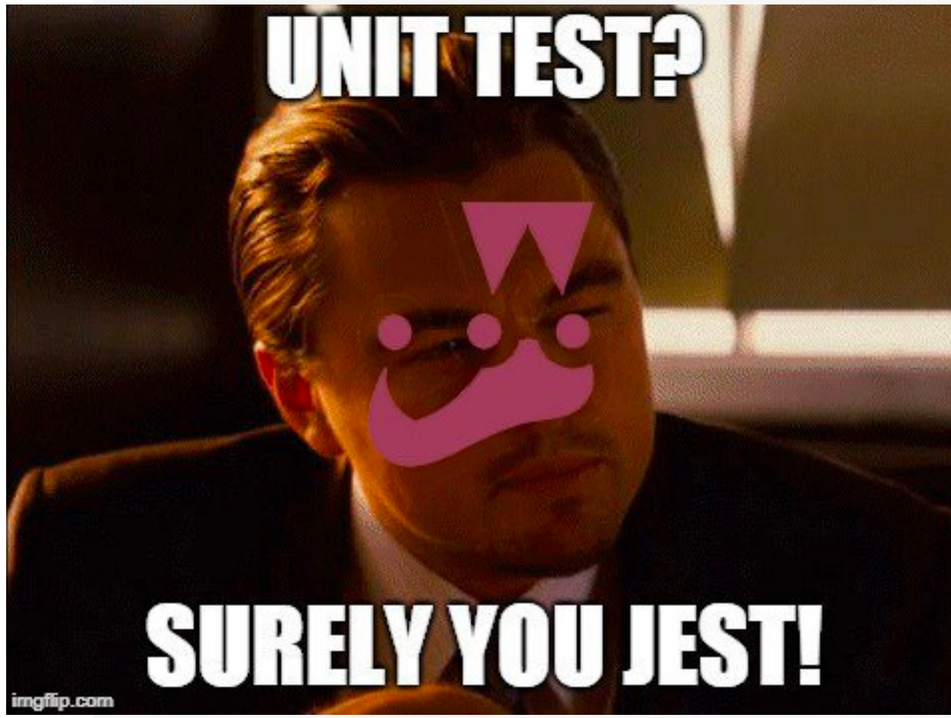
class-01/demo/ modules-tested

After installing Jest and setting up our `package.json`, we can use `npm test`.

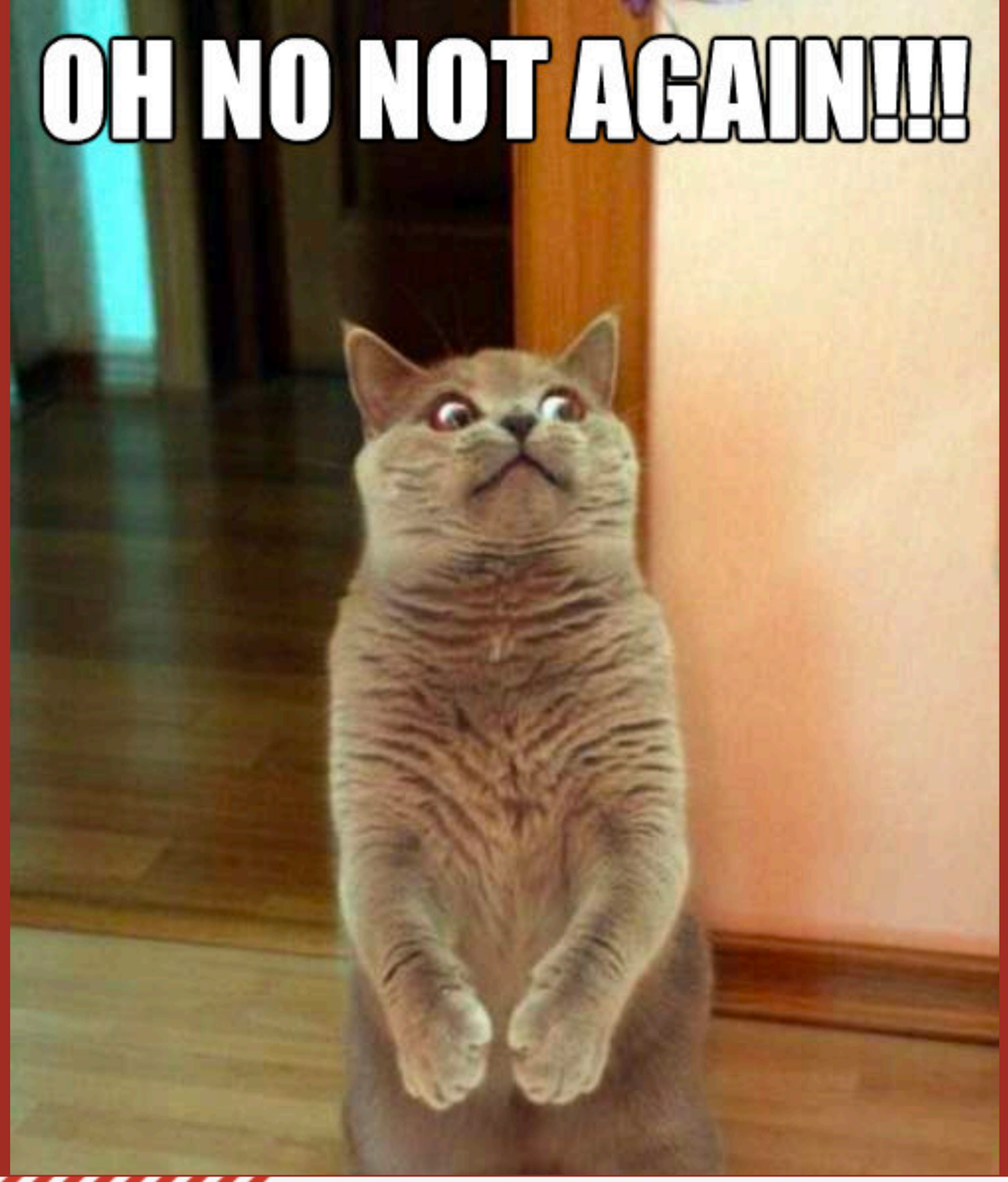
We write tests in a `.test.js` file

Structure of a Test

```
describe( name of test group, () => {
    it( name of test, () => {
      // do something
      // save result in a variable
      expect ( variable ).toXXXX();
    }
    // more it statements
});
```



There are a lot of cool expect functions. Check out the full list here!



Vocab Review!

What does `npm` stand for?

What does `npm` stand for?

`npm` stands for “**Node Package Manager**”. This is the primary way you can add packages to your code, using commands like `npm install`. You can also use it to run scripts from you package.json, such as `npm test` or `npm start`.

What is Jest?

What is Jest?

Jest is a **JavaScript Testing Framework**. You install it as a package using `npm install jest`, and by adding the script `"test": "jest --verbose –coverage"` to your `package.json`, you can use `npm test` to have Jest run all your test.js files and see if your code passes those tests.

What is unit test?

What is unit test?

A unit test is a test (or group of tests) which fully covers the functionality of a small independent module or piece (a.k.a. **unit**) of code.

What is a dependency?

What is a dependency?

A dependency is a module or package that the current code project needs to run correctly. There can also be **dev dependencies** which are only needed for testing or other development processes to run correctly. Dependencies are defined in the `package.json` of a project, and are installed using `npm install`.

What is the node_modules folder?

What is the node_modules folder?

This is a folder created after running `npm install` in your project. It is a folder that contains all the downloaded content from packages / dependencies specified in your `package.json`. You should never commit a node_modules folder into your repository.

What Test Driven Development (TDD)?

What Test Driven Development (TDD)?

Test Driven Development (TDD) is a practice where you write a test very early in your coding process, and then try to write the smallest chunk of code that will pass that test. TDD encourages modular coding and also verifies that your code is correct early on, letting you focus on making your code better, cleaner and more smart!

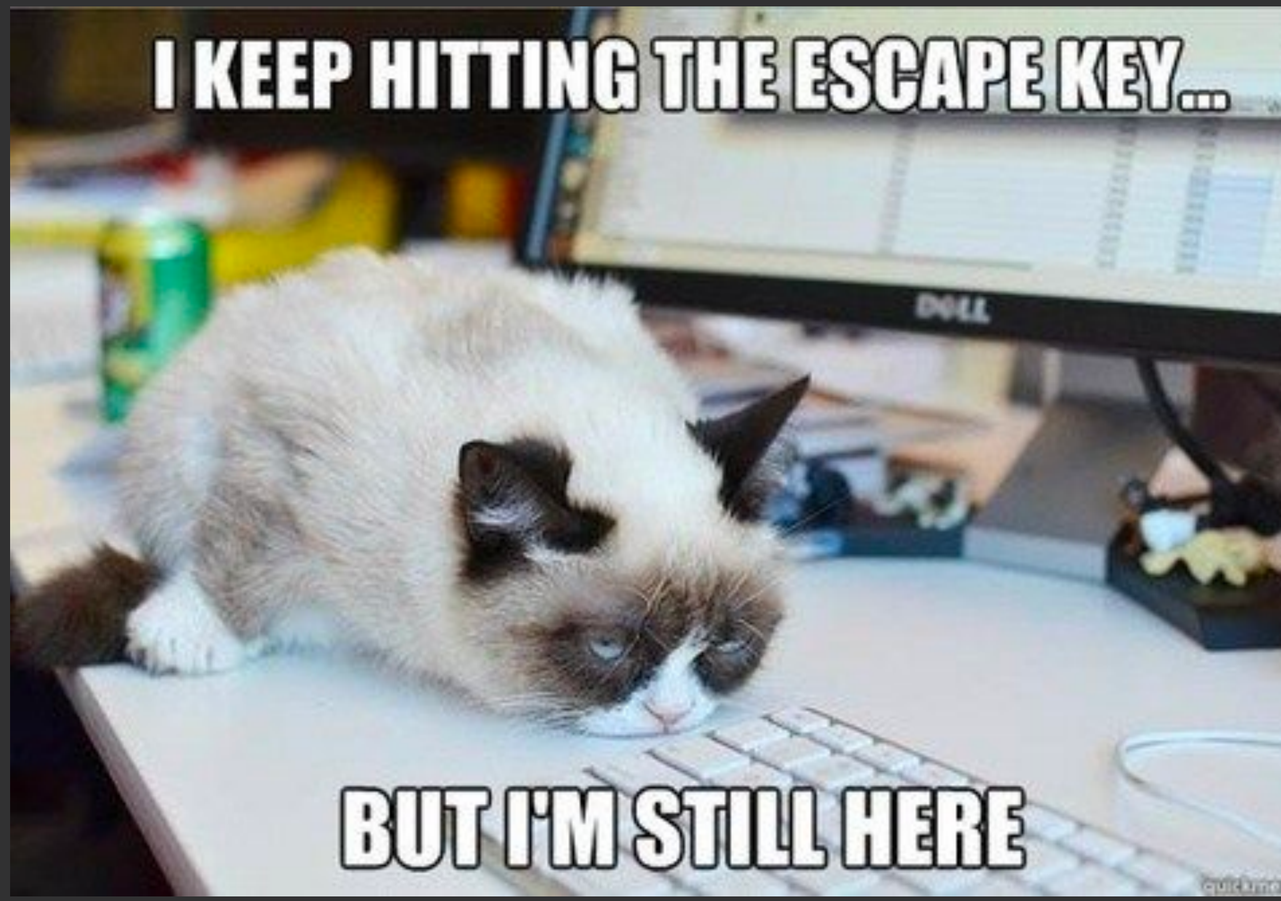
What is Continuous Integration (CI)?

What is Continuous Integration (CI)?

Basically, Continuous Integration (CI) means merging code changes into a repository as often as possible. CI has expanded to also mean services that automate this process - continuously checking your code for changes, running tests, enforcing coding standards and more! In this class, we'll be using **TravisCI** to do continuous integration.

Whew, you made it!

(For real this time)



What’s Next:

- Due by Midnight today: **Learning Journal 01**
- Due by Midnight tomorrow:
    - **Feedback Week 01 Survey**
    - **Career Coaching Overview**
- Due by Midnight Monday: **Code Challenge 01**
- Due by 6:30pm Tuesday:
    - **Lab 01**
    - **Read: Class 02**
- Next Class: **Classes, Inheritance and Functions**

Questions?