401 Advanced Javascript

seattle-javascript-401n16

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!
- Optional co-working night on Monday and lab work night on Thursday
- We can do this!

401 Pre-work

Due March 28th before class (9am) ☀️

- Lab 00 - Deployment Workshop
- Install Node.js
- Install and Setup Git
- Install MongoDB
- Install REST Clients
- Code Challenge 01 - JS Fundamentals
- Code Challenge 02 - ES6 Classes
- Code Challenge 03 - Callbacks
- Code Challenge 04 - Promises
- Code Challenge 05 - Async/Await
- Create NPM Developer Account
- Create AWS Developer Account
- Create Heroku Account
- Engineering Topics
- Fork and Setup the Class Repo
- Setup your Slack Account
- Career Coaching Status Report
- Setup Reading Notes Repo

Course Outline

March 2020

01 — Node Ecosystem
02 — Classes, Inheritance
03 — Data Modeling
04 — Advanced Mongo
05 — DSA Linked Lists
06 — HTTP and REST

April 2020

07 — Express
08 — Express Routing
09 — API Server
10 — Authentication
11 — DSA Stacks/Queues
12 — OAuth
13 — Bearer Authorization
14 — Access Control (ACL)

May 2020

15 — Event Driven Apps
16 — TCP Server
17 — DSA Trees
18 — Socket.io
19 — Message Queues
20 — Midterm Prep
XX — Midterm
XX — Midterm
XX — Midterm

June 2020

XX — Midterm
21 — Component UI
22 — React Testing
23 — Props and State
24 — Routing
25 — DSA HashTables
26 — Hooks API
27 — Custom Hooks
28 — Context API

July 2020

XX — JULY 4th HOLIDAY
29 — Application State
30 — DSA Graphs
31 — Combined Reducers
32 — Asynchronous Actions
33 — Additional Redux
34 — React Native
35 — Advanced React Native

August 2020

36 — Gatsby and Next
37 — JavaScript Frameworks
38 — Finals Prep
XX — Finals
XX — Finals
XX — Finals

Your Course Repo

- seattle-javascript-401n16
- Stay up-to-date with any changes by running `git pull upstream master` before every class!
- Every class folder will have:
    - Class Reading (`README`)
    - Lab (`/lab`)
    - In-class demos (`/demo`)

Class 01

Node Ecosystem, TDD, CI/CD

seattle-javascript-401n16

```
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 **ecosystem**.

Ecosystems contain compilers that interpret our code and do some administrative work so it runs correctly.

Why are we using Node.js?

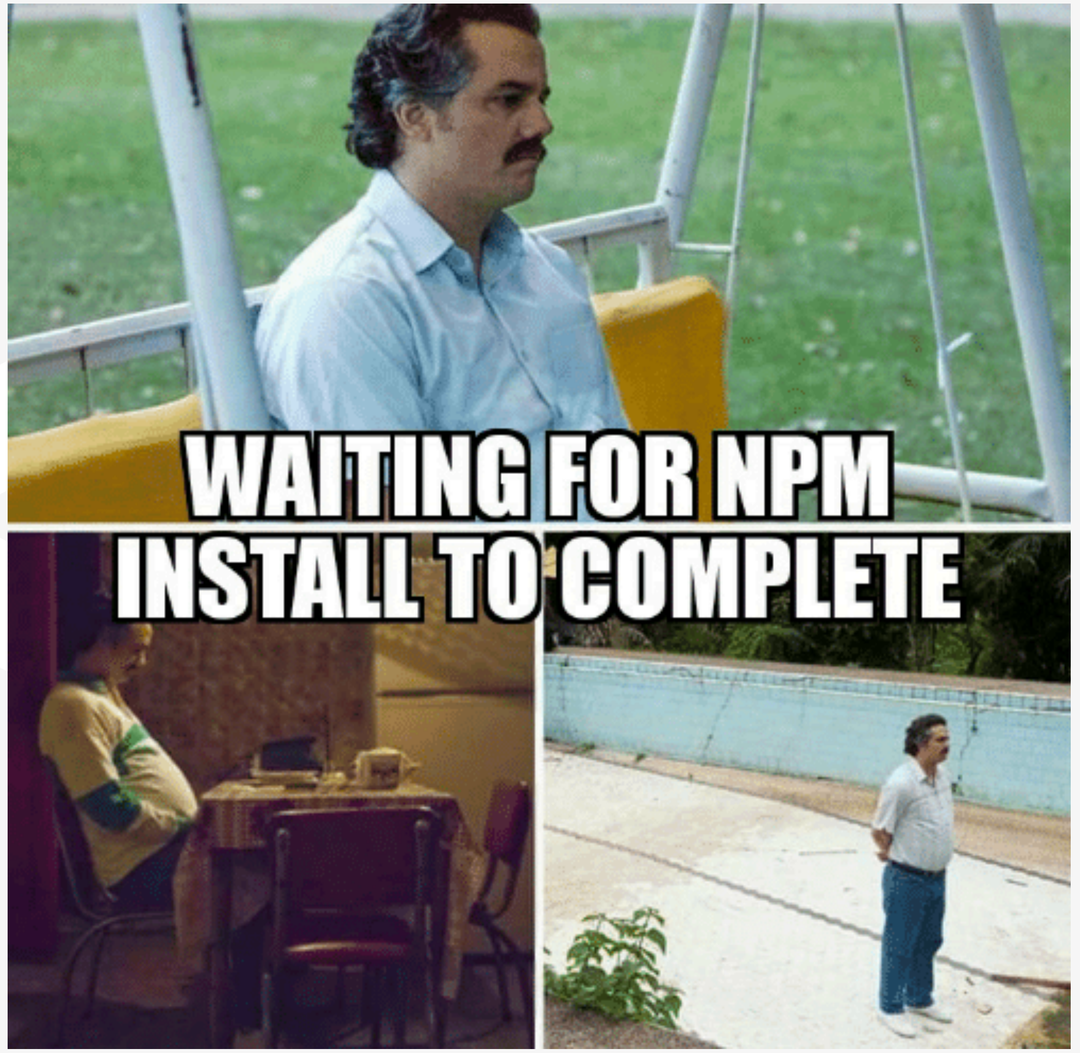
- Let’s make JavaScript a standalone language, not tied to HTML / browser
- Node runs JavaScript code - one language for the **front-end** AND **back-end**
- Super-fast when Node runs code
- Has a lot of **packages** from other developers that you can easily install
- Can handle “**deploying**” your code really well, so that you can create powerful web apps like Facebook 😮

Wait, Packages?

- **Packages** = **modules** + **dependencies** + scripts
- A module is a self-contained piece of code, meant to be imported and called
- Packages have a package.json file. Node uses this to figure out…
    - What dependencies to install
    - How to run your code (which file to run)
    - Any custom scripts developers might want

Cool So How Do I Use?

- `npm init` — Initialize your project as a package and create a *`package.json`*
- `npm install <package>` — How to install a package to your project
- `npm install` — Automatically install any dependencies defined in the *`package.json`*



```
// 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

A module can be imported into an application using the `require` keyword.

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

Compare.js

```
1   const compare = (a, b) => {
2       return a > b;
3   };
4
5   modules.exports = compare;
6
```

math.js

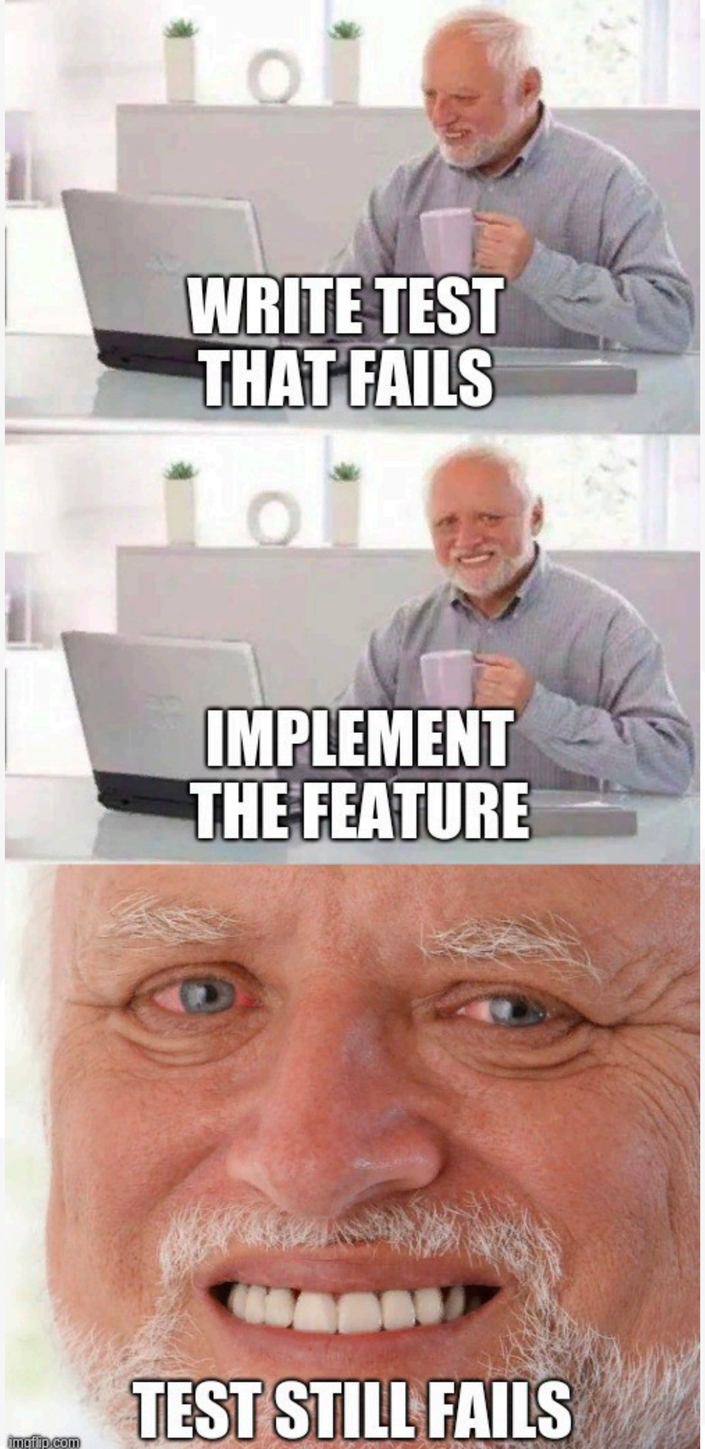
```
1   const add = (a, b) => {
2       return a + b;
3   };
4
5   const subtract = (a, b) => {
6       return a - b;
7   };
8
9   module.exports = { add, subtract };
```

index.js

```
1   const compare = require('./compare.js');
2   const math = require('./math.js');
3
4   if (compare(6, 3)) {
5       math.add(6, 3);
6       math.subtract(6, 3);
7   }
```

Test Driven Development

- Conceptually break up your planned code into small segments, or **units**
- Three steps:
  - **RED**: Write tests before code is finished. All tests start off failing.
  - **GREEN**: Write code the dirty quick way until all tests pass
  - **REFACTOR**: Clean up your code, checking that tests pass along the way



Jest

- A **package** that makes it very simple to test JavaScript code
    - Installed globally (**Why?**)
- We **describe** a collection of tests
- For each test, we say what **it** does
- Within each test, we **expect** something to be a certain result

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 interesting `expect` functions. Check out the full list here!

```
// 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

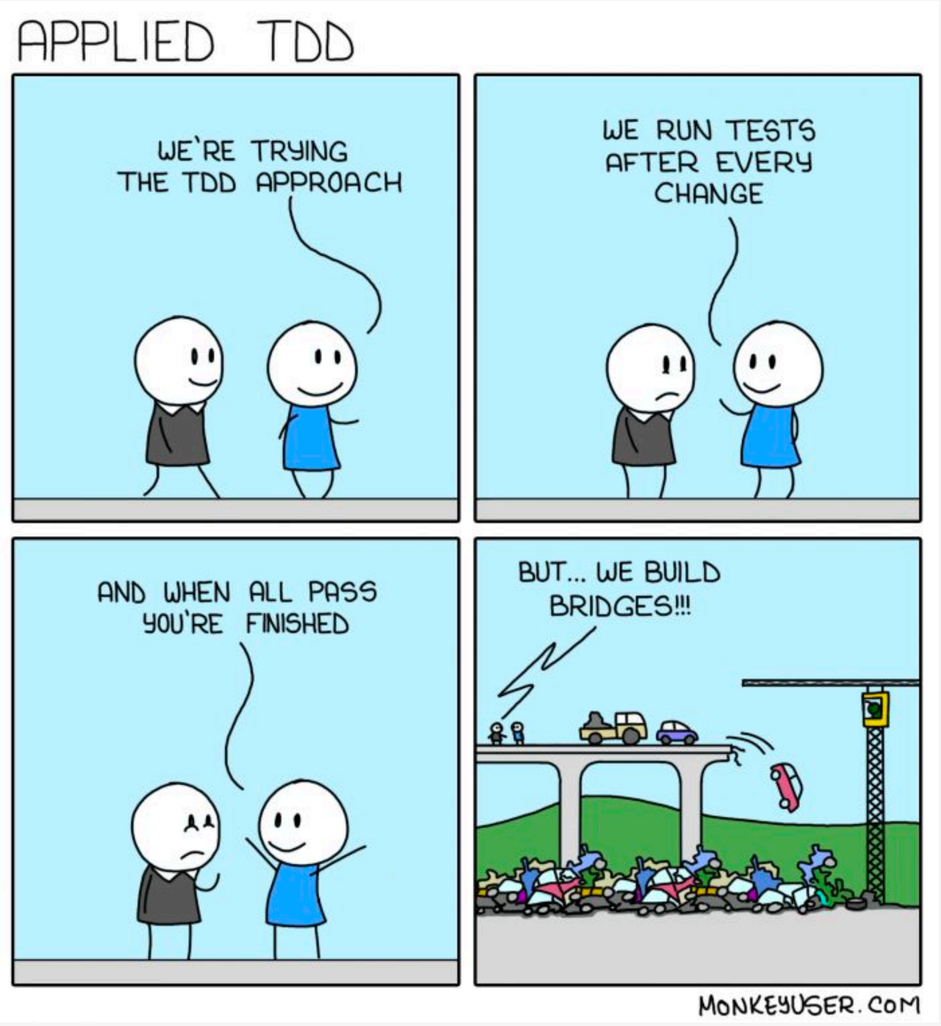
class-01/demo/testing

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

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

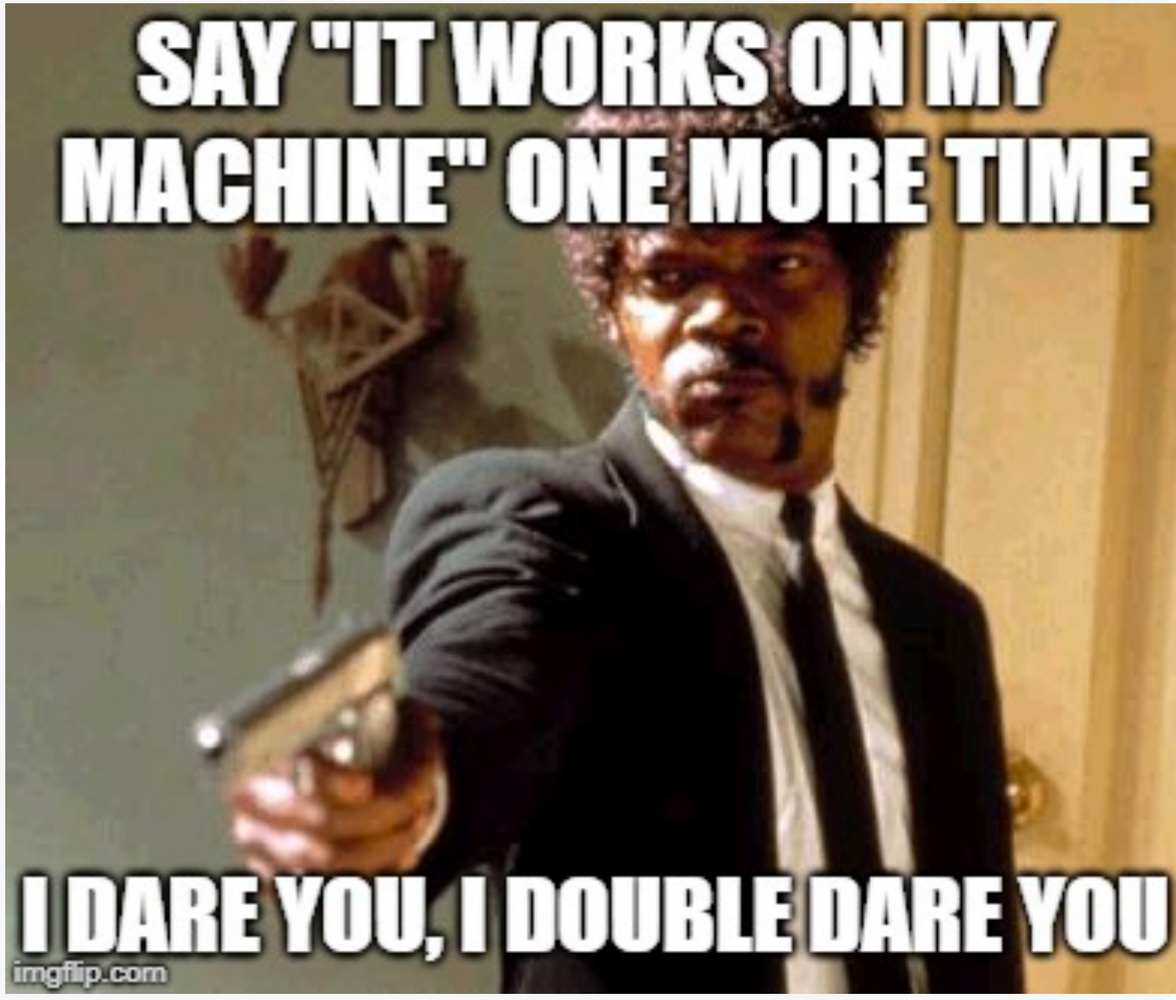
TDD - The Good and Bad

- TDD has been gaining popularity
- It forces developers to plan ahead
- It makes refactoring simple, because you just verify test are passing
- It makes it easier to catch mistakes
- It’s time consuming
- Some unit tests can be redundant / unnecessary
- It may not be possible to plan code ahead of time



Continuous Integration

- In a team, lots of people make changes to code
- Every time someone changes code, there's a chance for errors
- **Continuous Integration** = for every small code change, treat it like a complete "build"
  - Run all tests
  - Merge with master branch



Vocab Review

ecosystem

ecosystem

The collection of tools you’re using to develop and run your code. Ecosystems (also referred to as environments) can include the type of compiler, IDE, language and infrastructure you’re using as a developer for your application.

Node.js

Node.js

An open-source JavaScript runtime environment which allows to run JavaScript code outside of a browser. Node.js (often called Node) also enables developers to write JavaScript server-side code. Node has unified web development such that JavaScript can be used in every part of application development.

V8 Engine

V8 Engine

Google’s high performant open-source program that runs JavaScript code. The V8 Engine is what powers Chrome web browsers.

module

module

A module is a collection of functions or JavaScript code that is defined in a single JavaScript file and then exported for later use. Application code imports various modules and calls the functions or code defined within those modules.

package

package

A file or directory that is described by a `package.json` file.

This `package.json` lists which modules the file/directory needs to install, how to run the application located in the file/directory, and any scripts for development use. Packages usually contain a module(s), and are formatted for easy sharing across multiple developers.

node package manager (npm)

node package manager (npm)

The node package manager allows us to install packages from external developers and maintain package versions and updates.

test driven development (TDD)

test driven development (TDD)

A development process where tests for new code is written *before* the code. TDD encourages developers to plan their code and their code's test cases thoroughly before developing. This reduces errors during the development process, and makes it simpler to refine and refactor code.

Jest

Jest

Jest is a JavaScript testing framework that makes it very simple for developers to write and run unit tests for the JavaScript code. Jest relies on the idea of “expecting” an outcome from a code operation. Jest also has the ability to show how much of an application’s code is covered by unit tests.

continuous integration (CI)

continuous integration (CI)

This is a development practice where all code changes go through an automatic process of re-running tests, and then merging new code with existing tested code if no tests are broken. CI helps teams with multiple developers unify their codebase frequently.

unit test

unit test

A unit test is a small, modular test that covers one “unit” of code in an application. The definition of “unit” can be arbitrary or vary from team to team, but it generally refers to independent functions

Your Next Week

Saturday March 14	Sunday March 15	Monday March 16	Tuesday March 17
9 AM **— DUE Class 01 Reading** — Class 01 *MIDNIGHT* **— DUE Class 01 Learning Journal**	*MIDNIGHT* **— DUE Career Coaching Overview** **— DUE Class 01 Feedback**	*6:30 PM* — Optional Co-working	*6:30 PM* **— DUE Class 01 Code Challenge** **— DUE Class 01 Lab** **— DUE Class 02 Reading** — Class 02A
Wednesday March 18	**Thursday March 19**	**Friday March 20**	**Saturday March 21**
6:30 PM — Class 02B *MIDNIGHT* **— DUE Class 02 Learning Journal**	*6:30 PM* — Class 02 Lab and Code Challenge Co-working		*9 AM* **— DUE Class 02 Code Challenge** **— DUE Class 02 Lab** **— DUE Class 03 Reading** — Class 03 *MIDNIGHT* **— DUE Class 03 Learning Journal**

Lab Overview

Code Challenge Overview

Questions?