Code Fellows

Class 02

Classes, Inheritance and Functional Programming

seattle-javascript-401n14

```
'use strict';

const faker = require('faker');
const validator = require('../lib/validator.js');

let str = 'yes';
let num = 1;
let arr = ['a'];
let obj = {x:'y'};
let func = () => {};
let bool = false;

const schema = {
  fields: {
    id: {type: 'string', required: true},
    name: {type: 'string', required: true},
    age: {type: 'number'},
    children: { type: 'array', valueType: 'string' },
  },
};

describe('Validator module performs basic validation of', () => {

  it('strings', () => {
    expect(validator.isString(str)).toBeTruthy();
    expect(validator.isString(num)).toBeFalsy();
    expect(validator.isString(arr)).toBeFalsy();
```

Lab 01 Review

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 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 `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 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 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 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 functional programming?

(Hint: it’s related to modularity)

What is functional programming?

Functional programming defines a way to program: make everything **modular** and use as many functions as possible instead of writing (or re-writing) custom code.

```
5  let numbers = [1, 2, 3, 4, 5, 6];
6
7  console.log(numbers.filter(val => val % 2 !== 0));
8  console.log(numbers.map(val => val * val));
9  console.log(numbers.reduce((red, val) => red + val));
10 numbers.forEach(val => console.log(val));
11
```

```
functional  master + 1  node functional.js
[ 1, 3, 5 ]
[ 1, 4, 9, 16, 25, 36 ]
21
1
2
3
4
5
6
```

Demo

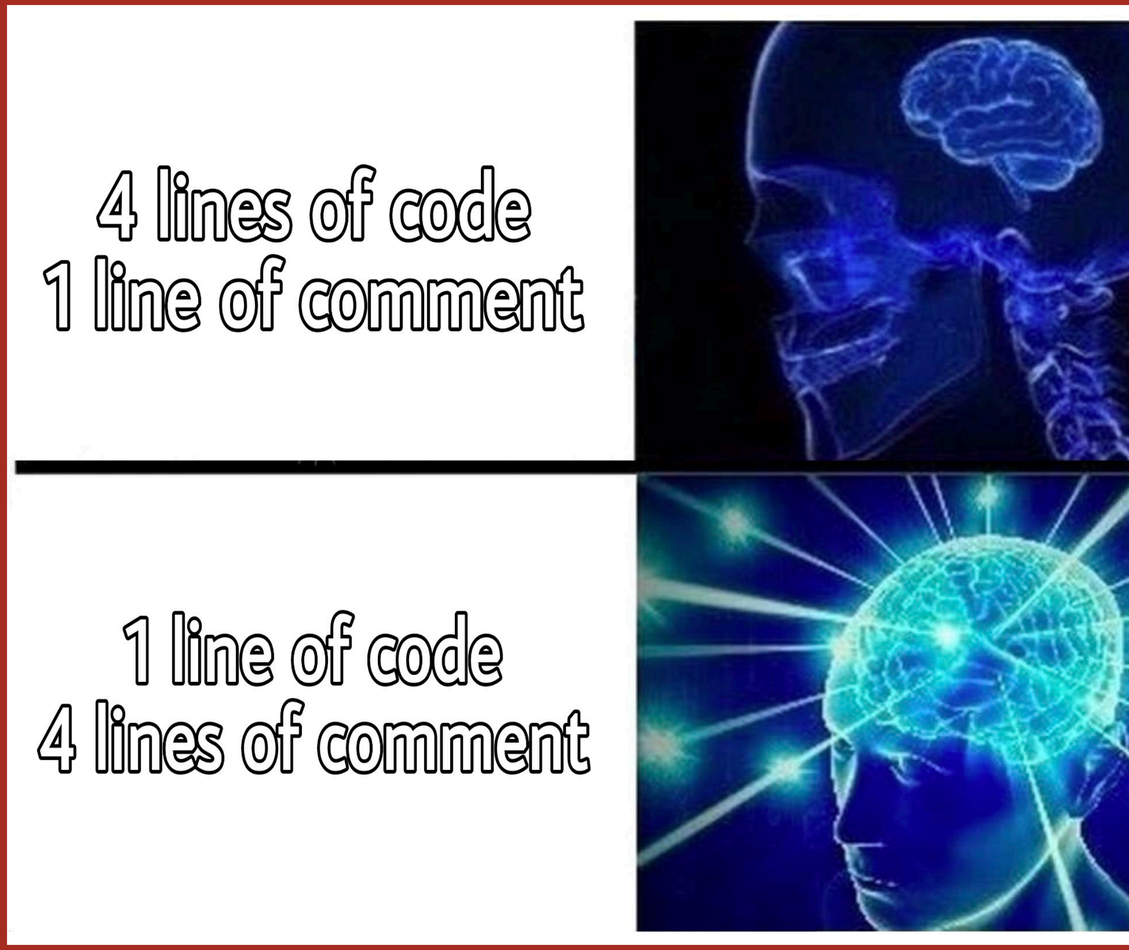
class-01/ demo/ functional

Strive to use functions where you can in your code.

Strive to make those functions modular, reusable and generic.

Functional Programming

Functional programming defines a way to program: make everything **modular** and use as many functions as possible instead of writing (or re-writing) custom code.



Some Cool Functions!

- **Map Function:**

```
newArray = array.map( val => { // change val in some way } )
```

- **Filter Function:**

```
newArray = array.filter( val => { // true/false comparison })
```

- **Reduce Function:**

```
reducedValue = array.reduce( red, val => { // use val to change red })
```

- **ForEach Function:**

```
array.forEach( val, in => { // do something with that data })
```

Pure & Anonymous Functions

- **Pure Functions**:
  - **Immutable**: Don’t modify the value of any arguments or external variables
  - Predictable: Given the same arguments, always return the same result (nothing unpredictable like API calls)
- **Anonymous Functions**: `() => {}`
  - Helpful for ad hoc / one-off logic
  - They are nameless and are usually set as an argument, a variable, or a return value

What is an object?

What is an object?

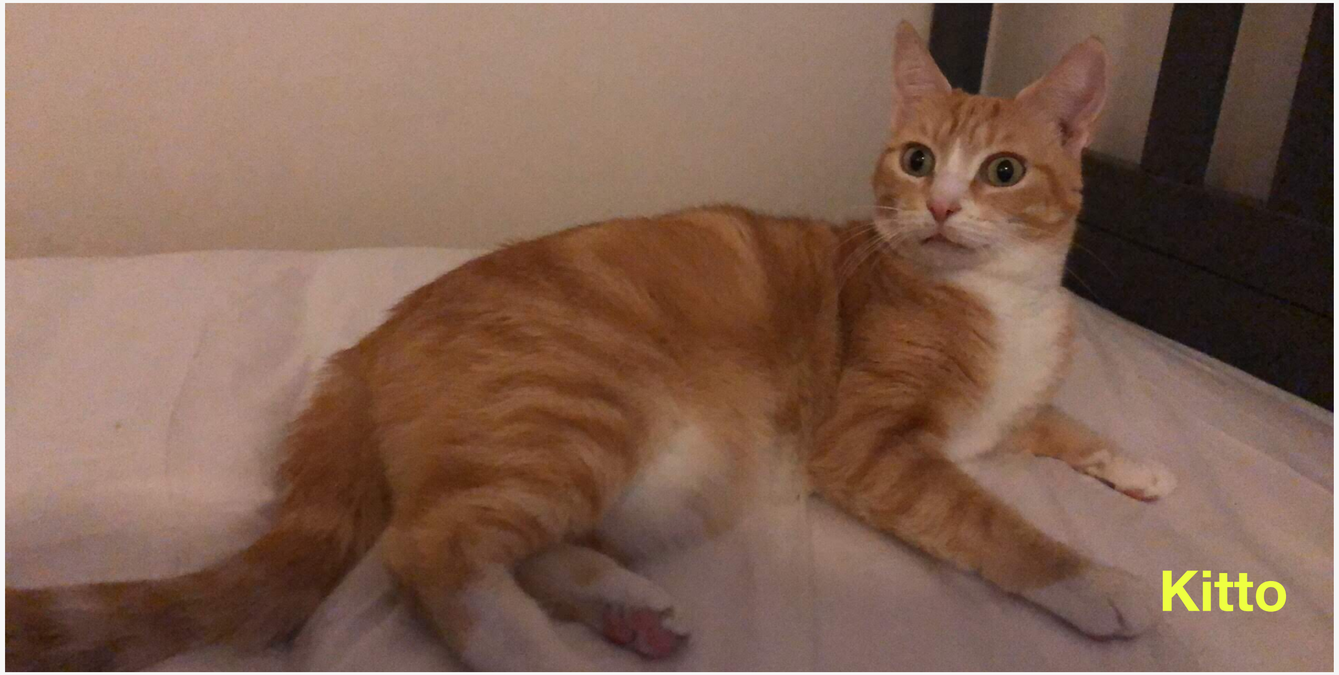
Also called an **instance**, an object is a basic **unit of data/code** that we use in our program. It can be a variable, a **data structure**, a function, etc. Objects usually have a **type** that defines what kind of data and operations that object can do.

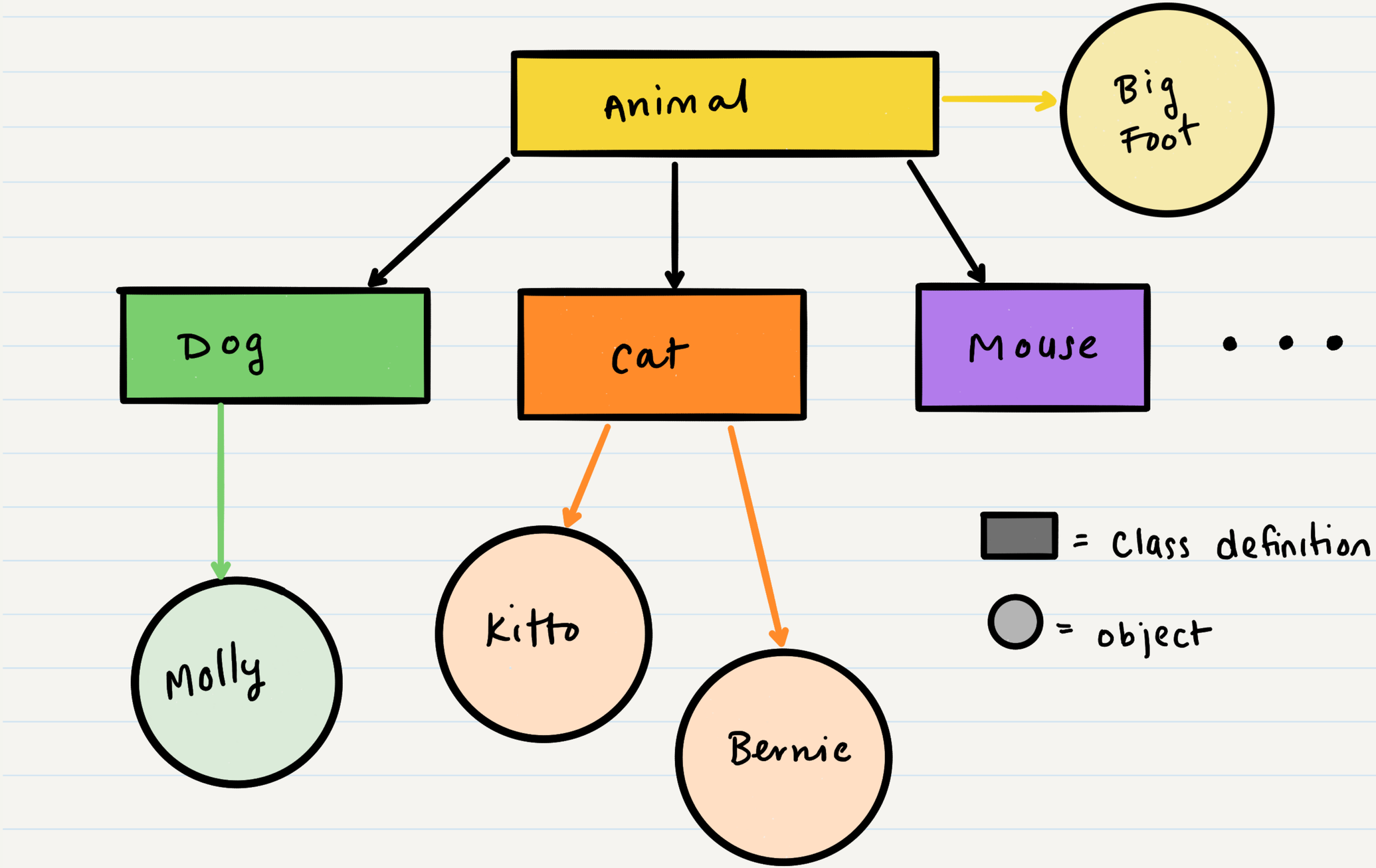
What is a class?

What is a class?

A class is a template for the structure/**type** of an object. Classes define what kind of data an object should hold and what kind of operations you can do on an object.

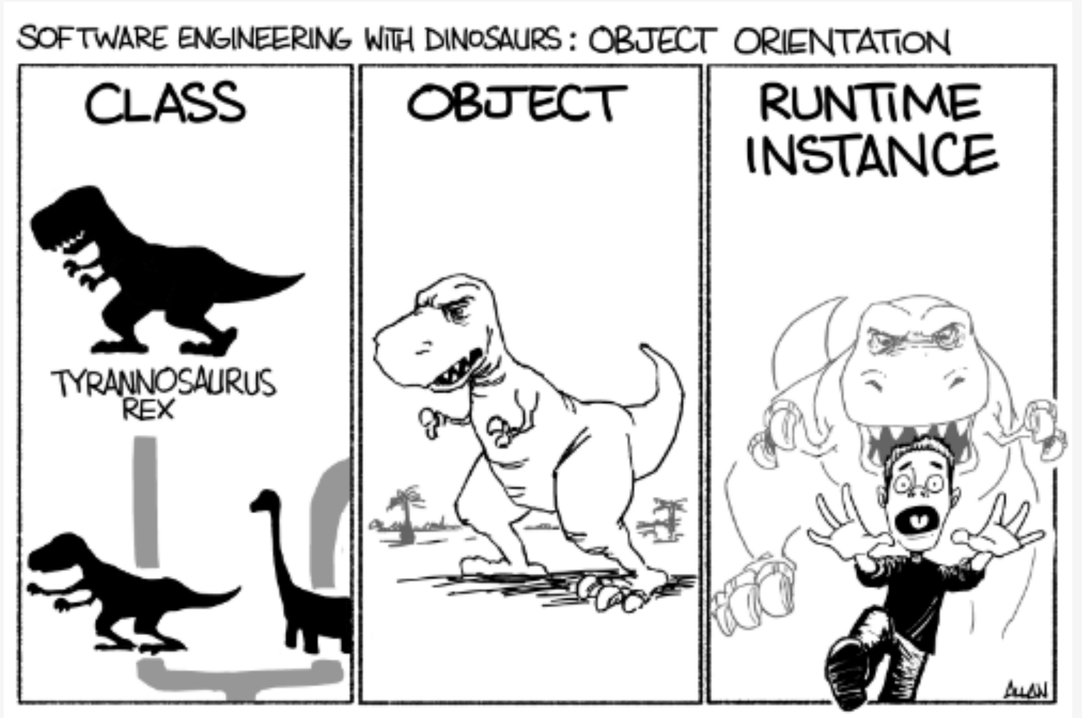
Diving into Objects and Classes




Animal
Big Foot
Dog
Cat
Mouse
• • •
Molly
Kitto
Bernie
= class definition
= object

Objects and Classes

- When we think of object "**types**" (string, array, number, boolean, etc.) we're actually thinking of the original **class** that defines the object's structure
- Objects are **instances** of a class, created when the code is running (at **runtime**)
- We set objects equal to a variable name that we use to **reference** the object



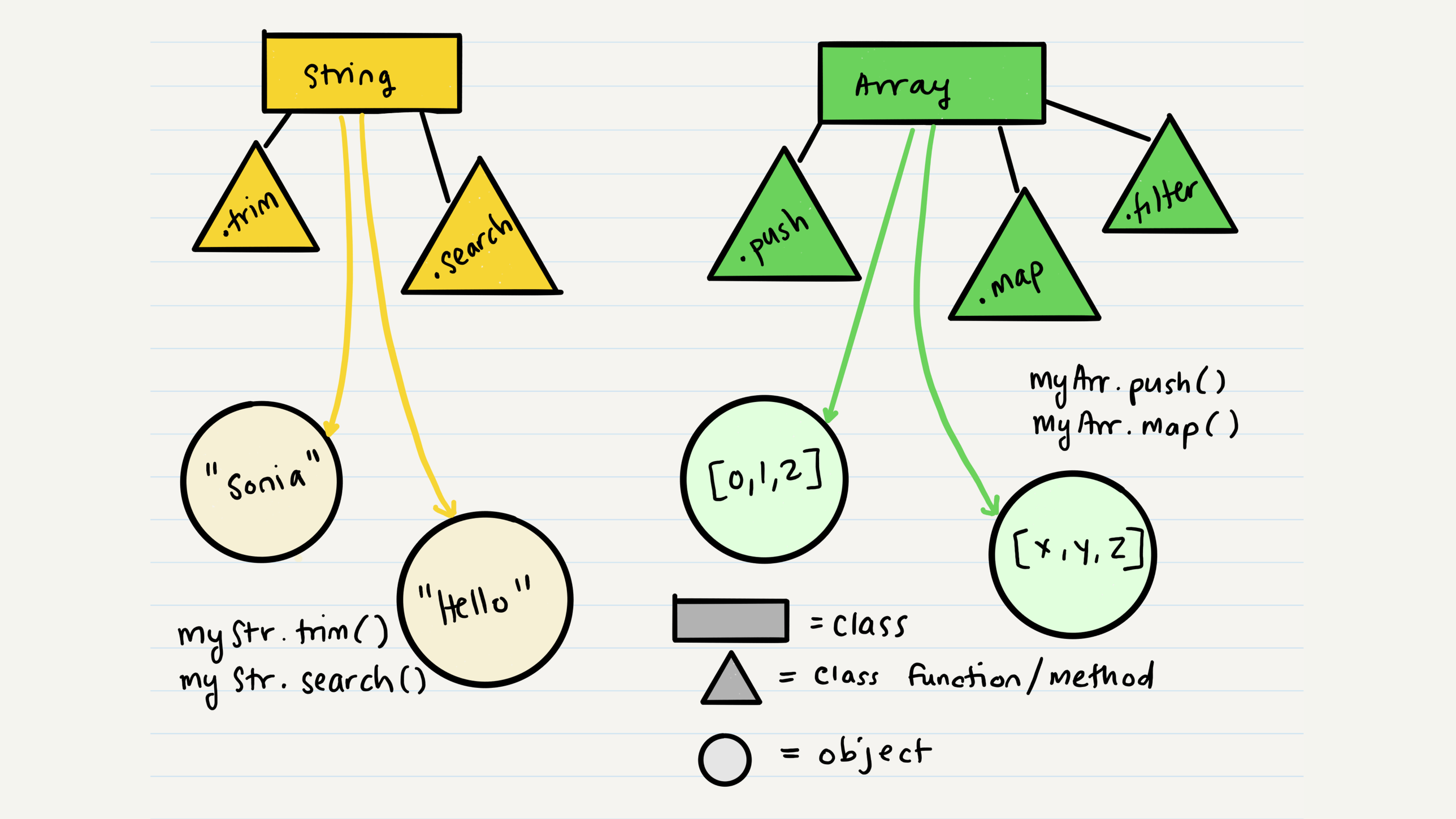
Inheritance

- Inheritance allows us to reuse logic from a previously defined class, and build on top of it
- Dog and Cat are classes that inherit from the Animal class
    - They all share some basic functionality
    - Dog and Cat can independently add to, change, or remove from Animal functionality

So What's in a Class?

- How to initialize (or *construct*) any object from that class
  - We do this via a **constructor**
  - If I'm creating an Animal object, I need a name and a species
  - If I'm creating a Dog object, I need a name, a breed, and an owner
- Any functions/**methods** you can do upon the object

```
molly.feed();      kitto.pet();      bigFoot.getInfo();
```


String
.trim
.search
"Sonia"
"Hello"
myStr.trim()
myStr.search()
Array
.push
.map
.filter
myArr.push()
myArr.map()
[0,1,2]
[x,y,z]
= class
= class function/method
= object

```
4  class Animal {
5    // When we create a new animal, this
6    // is all the data we need
7    constructor(name, species) {
8      this.name = name;
9      this.species = species;
10   }
11
12   info() {
13     console.log(`${this.name} is a ${this.species}.`);
14   }
15
16   eat() {
17     console.log(`${this.name} eats food.`);
18   }
```

```
soniakandah … class-02 demo classes master node index.js
Bessie is a cow.
Bessie eats food.
Bessie walks around.
You pet Bessie.
```

Demo

class-01/ demo/classes

Classes let us define the data and functions that an object use.

After we define a class, we create instances of that class using the keyword `new`

What is `this`.

- `this` refers to itself. When used in classes, it is a stand-in for the name of the instance we create
- We use `this` as an abstract way to talk about an instance
- `this.name = “Molly”` means when I create an object, make it have a key `name` equal to `“Molly”`

Ben Halpern
@bendhalpern

Sometimes when I'm writing Javascript I want to throw up my hands and say "this is bullshit!" but I can never remember what "this" refers to

Vocab Review!

What is an object?

What is an object?

Also called an **instance**, an object is a basic **unit of data/code** that we use in our program. It can be a variable, a **data structure**, a function, etc. Objects usually have a **type** that defines what kind of data and operations that object can do.

What is a class?

What is a class?

A class is a template for the structure/**type** of an object. Classes define what kind of data an object should hold and what kind of operations you can do on an object.

What is inheritance?

What is inheritance?

Inheritance is the idea that we can make new classes based off of an existing class, building on top of it. We can define a class that inherits from another class by using the `extends` keyword. Using the function `super()`, we can access the parent class **constructor**.

```
class Dog extends Animal { }
```

What is a constructor?

What is a constructor?

This is a function that defines how to initialize / construct an object of the current class. It usually takes in some arguments that are the initial values, and saves or does something with those initial values. This function runs when we use the `new` keyword.

What is an instance?

What is an instance?

An instance is an object created based off of a class specification during runtime. For example, a particular dog, Molly, is an instance of the Dog class. We define an instance using the keyword `new`:

```
let molly = new Dog();
```

What does `super()` do?

What does `super()` do?

If you are using a class that `extends` a parent class, calling `super()` will call the parent class constructor. We typically want to do this before adding our own constructor code.

What is a class method?

What is a class method?

A class method is a function defined in a class, that acts upon a class object / class instance. You can access a class method on an instance by using `.methodName()`

```
molly.feed();        kitto.pet();        bigFoot.getInfo();
```

What is `this`?

What is `this`?

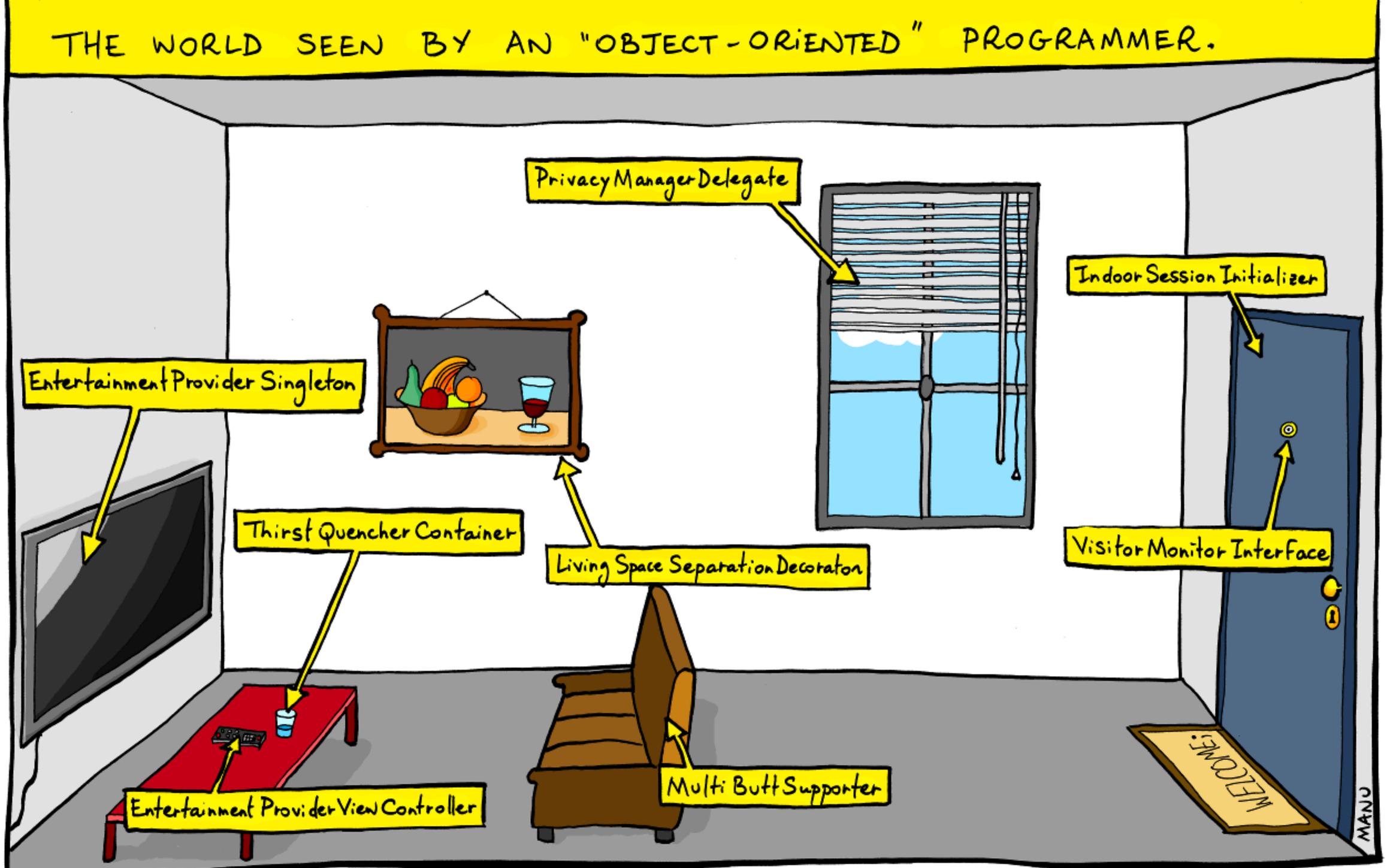
`this` is a keyword in JavaScript used to refer to the object or entity that the current running code is within. What `this` ends up being equal to depends on the **context** that the code is executed in.

What is Object-Oriented Programming?

What is Object-Oriented Programming?

Object Oriented Programming (OOP) is a way to develop code such that you're thinking about how to break up as much of the code into **objects**: classes, functions, data strcutures, etc.

THE WORLD SEEN BY AN "OBJECT-ORIENTED" PROGRAMMER.



Let's get back to `this`.

- Remember how we said `this` changes based on the **context**?
  - Context is similar to the idea of scope
- You can manually mess around with context by using certain functions such as `bind`, `call`, and `apply`

```
23    callback: function(fn) {
24      fn.call();
25    },
26
27    runBadly: function() {
28      this.callback(function() {
29        console.log("I'm running badly!");
30        this.bar();
31        this.foo();
32      });
33    },
34
```

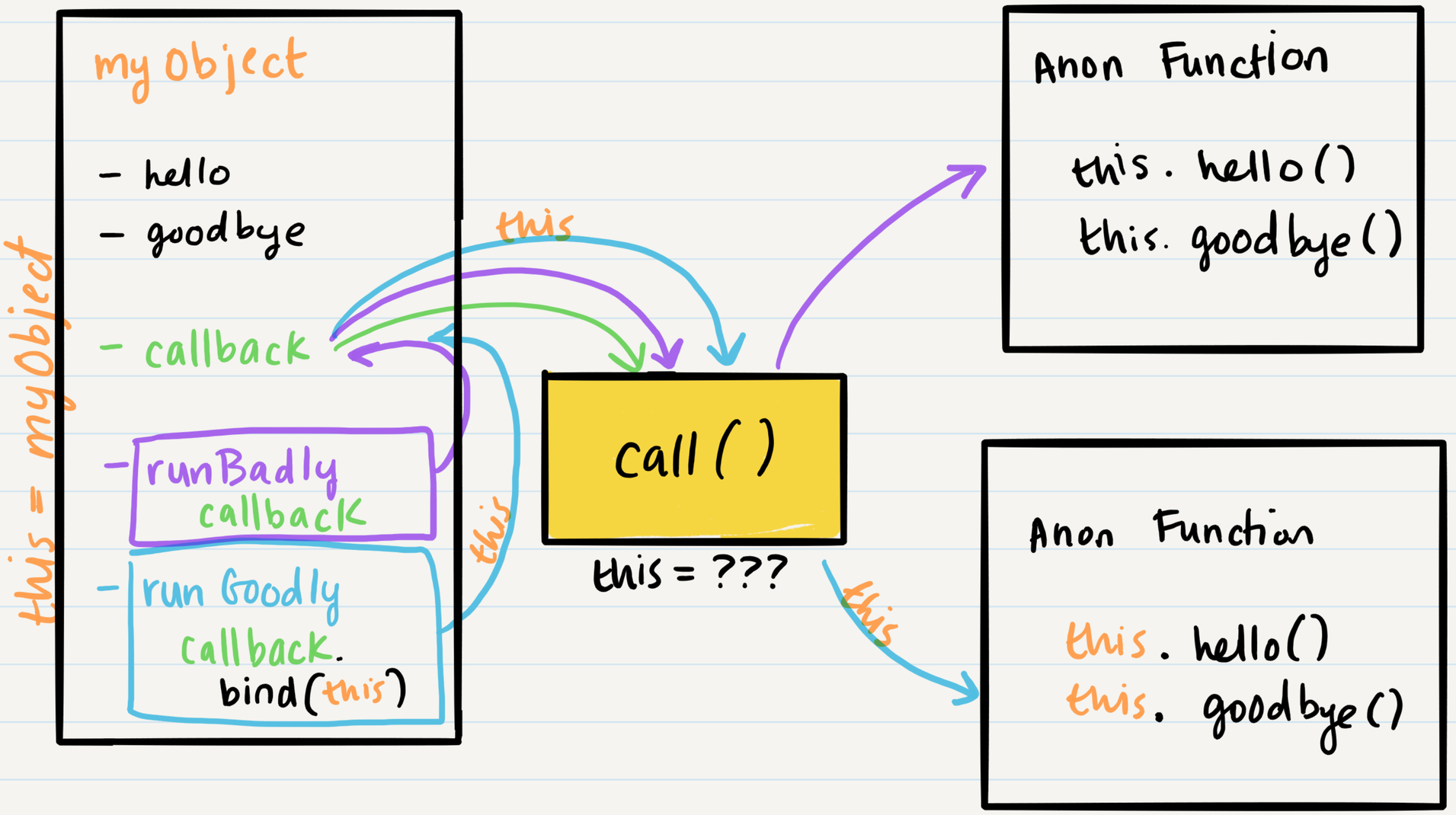
```
I'm running badly!
/Users/soniakandah/cf/js-401n14/curriculum/class-02/demo/context/bind.js:30
      this.bar();
           ^

TypeError: this.bar is not a function
    at /Users/soniakandah/cf/js-401n14/curriculum/class-02/demo/context/bind.js:30:12
    at Object.callback (/Users/soniakandah/cf/js-401n14/curriculum/class-02/demo/context/
bind.js:24:8)
    at Object.runBadly (/Users/soniakandah/cf/js-401n14/curriculum/class-02/demo/context/
```

Demo

class-01/ demo/context

When you try to use `this` in callback statements, the **context** of this can change. You can manually set the context in a couple of different ways to get the right result.


myObject
- hello
- goodbye
- callback
- runBadly callback
- run Goodly callback.bind(this)
this = myObject
this
this
call()
this = ???
this
Anon Function
this.hello()
this.goodbye()
Anon Function
this.hello()
this.goodbye()

Cool Context Functions

- **Call Function:**

```
myFunc.call(thisObject, param1, param2, param3 ….)
```

- **Apply Function:**

```
myFunc.apply(thisObject, [params])
```

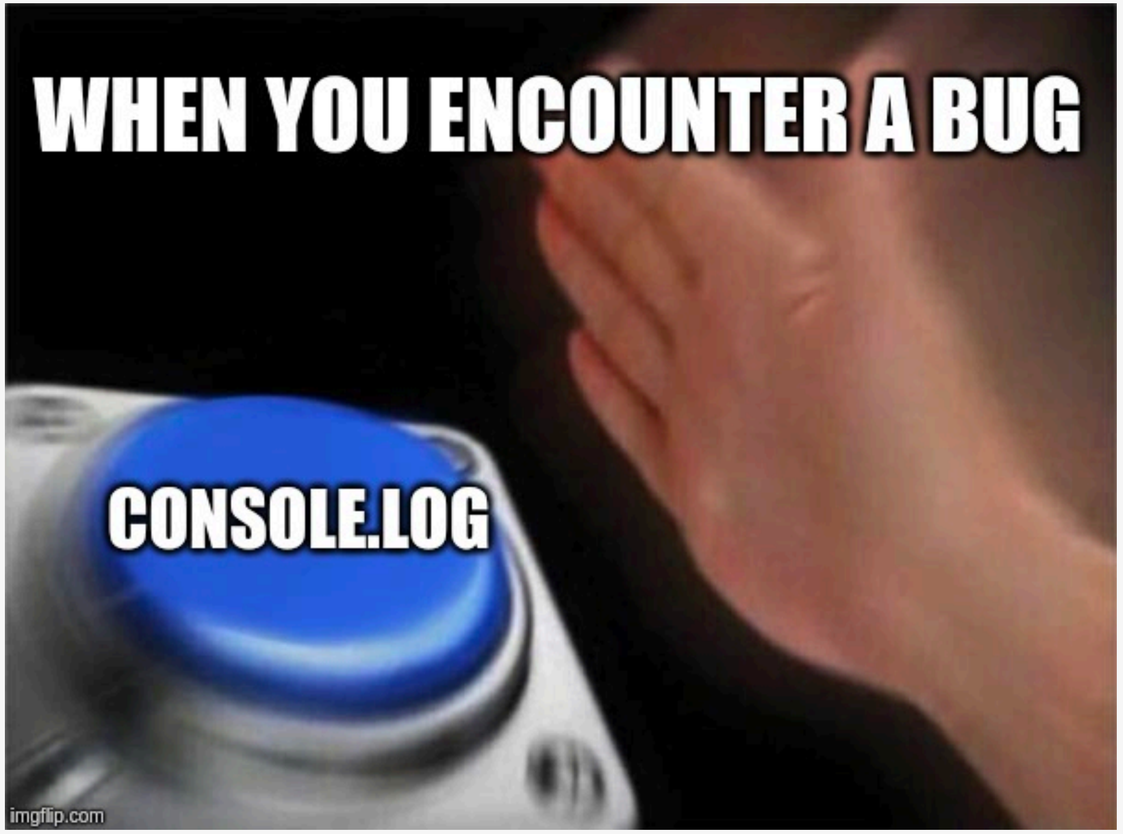
- **Bind Function:**

```
myFunc().bind(this)
```

```
() => {}.bind(this)
```

The Error Class

- One of the most common things you'll do as a programmer is deal with errors
    - Let's move away from `console.log()` for our error-checking
- JavaScript has a built-in **Error class** that is super handy!



```
21  let petName = "";
22
23  try {
24    petName = person.pets.molly.name;
25  } catch (e) {
26    console.log("That pet doesn't exist!");
27  }
28
29  // Know that you can always throw your own errors ...
30  throw new Bug("You really messed up!");
31  throw new Error("You really messed up again!");
32
33  // but better would be to implement an error handling module
34  // and use that within all of your code.  This would allow you
35  // control how it logs, looks, and functions.  For example...
36
```

```
Kitto
That pet doesn't exist!
/Users/soniakandah/cf/js-401n14/curriculum/class-02/demo/errors/index.js:30
throw new Bug("You really messed up!");
^

Error: __ERROR__ undefined: undefined (LEVEL 0) (TIMESTAMP 2019-10-03T00:24:25.570Z)
    at Object.<anonymous> (/Users/soniakandah/cf/js-401n14/curriculum/class-02/demo/erro
rs/index.js:30:7)
    at Module._compile (internal/modules/cjs/loader.js:778:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:789:10)
```

Demo

class-01/ demo/errors

We can use the JavaScript Error class to handle errors in a better way.

We can use `try catch` blocks to respond to an error instead of crashing the app.

Try Catch is Good!

- By using `try catch` blocks, we prevent our apps from crashing when they see an error
- `try catch` will automatically move to the `catch` block and send over the caught error if any code in the `try` block raises an error

```
try {
    // some code
} catch(error) {
    // handle error }
```

What Kind of Errors Exist?

- **Error**: a generic error
- **TypeError**: you provided an argument that was the incorrect type, or tried to treat something like it was of a different type
- **SyntaxError**: you wrote JavaScript bad
- **ReferenceError**: you tried to use something that doesn't exist
- **SystemError**: Node did something bad

```
Error: __ERROR__ undefined: undefined (LEVEL 0) (TIM
    at Object.<anonymous> (/Users/soniakandah/cf/js-
```

```
TypeError: Cannot read property 'name' of undefined
    at Object.<anonymous> (/Users/soniakandah/cf/js-
```

```
SyntaxError: Unexpected end of input
    at Module._compile (internal/modules/cjs/loader
```

```
ReferenceError: pet is not defined
    at Object.<anonymous> (/Users/soniakandah/cf/js-
```

```
3  const Vehicle = function(name, wheels) {
4    this.name = name;
5    this.wheels = wheels;
6  };
7
8  Vehicle.prototype.drive = () => {
9    return 'Moving Forward';
10 };
11
12 Vehicle.prototype.stop = () => {
13   return 'Stopping';
14 };
15
16 // Car Constructor
17 const Car = function(name) {
18   Vehicle.call(this, name, 4);
19 };
20
21 Car.prototype = new Vehicle();
22
23 const Motorcycle = function(name) {
```

Lab 02 Pair Program

Vehicles

```
3  class List {
4    constructor() {
5      this.length = 0;
6      this.data = {};
7    }
8
9    reindex() {
10     let data = Object.keys(this.data).sort().reduce((
11       acc[idx] = this.data[val];
12       return acc;
13     },{});
14
15     this.length = Object.keys(data).length;
16     this.data = data;
17   }
18
19   push(item) {
20     if ( arguments.length === 1 ) {
21       this.data[this.length++] = item;
22     }
23     return this.length;
```

Lab 02
Pair Program

List

```
16  /**
17   * Is this a string?
18   * @param input
19   * @returns {boolean}
20   */
21  validator.isString = (input) => {
22    return typeof input === 'string';
23  };
24
25  validator.isObject = (input) => {
26    return typeof input === 'object' && !(input instanc
27  };
28
29  validator.isArray = (input, valueType) => {
30    return Array.isArray(input) && (valueType ? input.e
31  };
32
33  validator.isBoolean = (input) => {
34    return typeof input === 'boolean';
35  };
36
```

Lab 02 Overview

Validator

Vocab Review!

What is functional programming?

What is functional programming?

Functional programming defines a way to program: make everything **modular** and use as many functions as possible instead of writing (or re-writing) custom code.

What is an object?

What is an object?

Also called an **instance**, an object is a basic **unit of data/code** that we use in our program. It can be a variable, a **data structure**, a function, etc. Objects usually have a **type** that defines what kind of data and operations that object can do.

What is a class?

What is a class?

A class is a template for the structure/**type** of an object. Classes define what kind of data an object should hold and what kind of operations you can do on an object.

What is inheritance?

What is inheritance?

Inheritance is the idea that we can make new classes based off of an existing class, building on top of it. We can define a class that inherits from another class by using the `extends` keyword. Using the function `super()`, we can access the parent class **constructor**.

```
class Dog extends Animal { }
```

What does `super()` do?

What does `super()` do?

If you are using a class that `extends` a parent class, calling `super()` will call the parent class constructor. We typically want to do this before adding our own constructor code.

What is Object-Oriented Programming?

What is Object-Oriented Programming?

Object Oriented Programming (OOP) is a way to develop code such that you're thinking about how to break up as much of the code into **objects**: classes, functions, data strcutures, etc.

What is `this`?

What is `this`?

`this` is a keyword in JavaScript used to refer to the object or entity that the current running code is within. What `this` ends up being equal to depends on the **context** that the code is executed in.

How can you set/ change context?

How can you set/ change context?

You can use `call()`, `apply()` or `bind()` to manually set a context for a function, or maintain a context for a function even when it’s called from a different context (for example, through a callback)

What is a `try` `catch` block?

What is a `try` `catch` block?

A `try catch` block prevents errors from crashing our app by allowing us to create some error handling code. When any code in the `try` block creates an error, we send that error to the `catch` block, where we can write code to deal with it.

What is `Object.prototype`?

What is `Object.prototype`?

All JavaScript objects can be thought of as originating from a class. All JavaScript objects **inherit** from `Object.prototype`. By using `.prototype` on variables or functions, we can hook into the constructor for that object and change it.

What is a factory function?

What is a factory function?

A factory function is a function that returns a new object, but is not a class or a constructor. We've made a lot of factory functions before! Now we want to move to using classes.

What's Next:

- Due by Midnight tonight: **Learning Journal 02**
- Due by Midnight Thursday: **Code Challenge 02**
    - **We're going to do this in pairs!**
- Due by 9am Saturday:
    - **Lab 02**
    - **Read: Class 03**
- Next Class:
    - **Class 03 - Async**

Questions?