JAVASCRIPT CLASSES

OBJECT ORIENTED PROGRAMMING (OOP)

- Object oriented programming is a programming paradigm that states “everything is an object”.
- This is foundational to many programming languages, JS is not one of them.
- Javascript is considered “multi-paradigm”, we can do object oriented programming or functional programming with it.
- Technically speaking… javascript is function-based, with a prototypical inheritance pattern. Sound confusing? It is notoriously confusing and difficult to use. We won’t be deep diving into this, especially because this isn’t needed as much anymore!

OOP

- What do we mean when we talk about modular code?
- So far we've been putting a lot of energy into building modular code. So far our modular code is organized by putting our programs in functions and calling those functions on different variables.
- In object oriented programming we also focus on modular code, but instead of wrapping everything in functions, we wrap everything inside of objects.
- Variables exist as a key/value pair in an object
- Functions still exist but they are bound inside of an object as well. ***When an object has a key/value pair where the value is a function, we call it a "method".***

OOP

- Again, the key idea in OOP is that everything is inside of some kind of object. Additionally, objects should represent real things!
- Let's take a look at this example:

```
var door = {
  open: false,
  material: 'wood',
  openDoor: function(){
    this.open = !this.open;
  }
}
```

```
var door = {
  open: false,
  material: ‘wood’,
  openDoor: function(){
    this.open = !this.open;
  }
}

console.log(door.material);
door.openDoor();
```

OOP

- In this case we have a ‘door’ variable that represents a real object.
- It has state variables that say something about the object, is the door open? What is the door made of?
- Lastly it has a function. This function toggles the door’s own ‘open’ property. Changing it to ‘true’ if the ‘open property’ is currently ‘false’, and ‘false’ if the door property is currently ‘true’. But wait, what is ‘this’?

```
var door = {
  open: false,
  material: ‘wood’,
  openDoor: function(){
    this.open = !this.open;
  }
}

console.log(door.material);
door.openDoor();
```

THIS

- The ‘this’ keyword is contextually bound.
- It is a special variable that looks to see what object it belongs to, then assigns itself to the object.
- This means that we access properties and methods from the object by using ‘this.open’ as if we had written ‘door.open’.
- If you want to call a method, you use the object name + ‘.’ + method name with clappers ();

```
var dog = {
  name: 'Rosco',
  barksAtDoor: true,
  sayName: function(){
    console.log('I am ' + this.name);
    if (this.barksAtDoor) {
      console.log('arf arf!');
    } else {
      console.log('…');
    }
  }
};

dog.sayName();
```

WHAT DOES THIS DO?

MULTIPLE OBJECTS

- What would happen if we want to make 100 dog objects with unique names and barksAtDoor properties?
- We would have to type a lot. We would have to hand type new variables, new curly brackets and the same properties with new values over and over again.
- This is not good! We always want to have DRY code right?
- This is where classes come in.

CLASSES

- What if I told you that there was a solution to this problem? Enter “classes”.
- Classes are a way for us to make multiple objects that have the same structure with a minimum amount of code.
- A class is like blue print for objects. Once you make the blue print you can generate many new objects from that blue print in a single line of code!

```
class Dog {
  constructor(name, barksAtDoor) {
    this.name = name;
    this.barksAtDoor = barksAtDoor;
  }
  greet: function(){
    console.log('woof I am ' + this.name);
  }
};

var rosco = new Dog('Rosco', true);
var bandit = new Dog('Bandit', true);
var romanov = new Dog('Romanov', false);

romanov.greet();
```

CLASS SYNTAX

- First we begin with using the keyword “class” and the name of the class in capital letters.
- Then we use ‘constructor’ keyword. There is a lot going on here. Let’s break it into pieces:
- After the constructor keyword we have parameters inside of clappers. ***These parameters declare which properties we want to set when we create a new object from the class.***
- Inside the constructor there are ‘{ }’, inside of here is where we are actually setting the values for objects made from the class.
- When we create a new object from the class we set a new variable then put ‘new Dog(parameters, from, constructor)’

CONSTRUCTORS

- I added a third line to the constructor, what would this do?

```
class Dog {
   constructor(name, barksAtDoor) {
      this.name = name;
      this.barksAtDoor = barksAtDoor;
      this.wantsPets = true;  // what would this do?
   }
   greet: function(){
      console.log('woof I am ' + this.name);
   }
};
```

```
class Dog {
   constructor(name, barksAtDoor) {
      this.name = name;
      this.barksAtDoor = barksAtDoor;
      this.wantsPets = true;  // what would this do?
   }
   greet: function(){
      console.log('woof I am ' + this.name);
   }
};
```

CONSTRUCTORS

- I added a third line to the constructor, what would this do?
- It adds a default property! This way every dog that is created from the Dog class will have a default property called 'wantsPets' that has a value of 'true'

WHAT DOES THIS DO?

```
class Person {
 constructor (firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
 sayHiTo(other) {
    console.log('Hi ' + other.firstName + ', I am ' + this.firstName);
  }
}

var jackie = new Person('Jackie', 'Casper');
var trevor = new Person('Trevor', 'Preston');

jackie.sayHiTo(trevor);
```

CODEALONG

Classes are confusing. Let's do more examples.

REVIEW

- What is a method? How do you invoke it?
- What is object oriented programming?
- What is a class? What is the benefit of using classes?
- What does the constructor do?
- What are the parameters in a constructor?
- What does the stuff in the curly brackets of a constructor do?
- How do you create a new object from a class?

LAB

Work with a partner to create a Monkey class, which has the following properties: name, species, foodsEaten. Set the species to have a default property ‘monkey’, and foodsEaten to be an empty array. The name should be set when you create a new monkey object from the class.

Add the following methods:
* eatSomething(food) takes in a string as a parameter then adds it to the foods Eaten array.
* introduce: produces a string introducing itself, including its name, species, and what it's eaten.

Create 3 monkeys total. Make sure all 3 monkeys have a name property set. Feed your monkeys some food then have them each introduce themselves!

CODEALONG

How can we use classes in our jQuery apps?

INHERITANCE

INHERITANCE

- We can also make classes that are subclasses of another class.
- Classes that are subclasses of other classes will inherit properties from the parent class.
- Say for example we want to create a class for a Person. A person has a first and last name.
- We also want to create a class for a Programmer. A programmer has a GitHub handle and a website, AND has a first and last name.
- We can create a Person class for ‘normal’ people and make new people from it.
- We can also create a Programmer class for amazing people that take all the same properties from the Person class as well as some more!

```
class Programmer extends Person {
  constructor(firstName, lastName, options) {
    super(firstName, lastName);
    this.githubHandle = options.githubHandle;
    this.website = options.website;
  }

  getResume() {
    return (
      this.firstName + ' ' + this.lastName + ‘Github:’ +
      this.githubHandle +'Website: ' + this.website
    );
  }
}

var marty = new Programmer(‘Marty’, ‘McFly’, {githubHandle:
    gh.co/mf4evah, website: ‘martydabest.com'});

var thanos = new Programmer('Thanos', 'McPower', {githubHandle:
'gh.co/thecleanser', website: 'aliens4tw.com'});
```

INHERITANCE

- Open up a text editor to play around with this and discuss the following:
- When we declare a sub-class we declare the class and then say `extends ParentClassName` to designate that it is a sub-class.
- Then in the constructor we have the keyword ‘super()’.
- The super keyword lets us access arguments from the parent’s constructor.
- The sub-class has access to the parent-classes methods. For example marty.sayHiTo(thanos) works!
- Sidenote: Also check out how we made new properties with an “options” parameter.

```
class Programmer extends Person {
  constructor(firstName, lastName, options) {
    super(firstName, lastName);
    this.githubHandle = options.githubHandle;
    this.website = options.website;
  }

  getResume() {
    return (
      this.firstName + ' ' + this.lastName + ‘Github:’ +
      this.githubHandle +'Website: ' + this.website
    );
  }
```

```
   sayHiTo(other) {
      super.sayHiTo(other);
      console.log('I AM A PROGRAMMER!');
   }
}

var marty = new Programmer(‘Marty’, ‘McFly’, {githubHandle:
    gh.co/mf4evah, website: ‘martydabest.com'});

var thanos = new Programmer('Thanos', 'McPower', {githubHandle:
'gh.co/thecleanser', website: 'aliens4tw.com'});
```

INHERITANCE

- You can also do “super” inside of any sub-classes method to invoke a function in the parent class before doing something else.

REVIEW

- What is inheritance? What is a subclass?
- What is the benefit of classes that inherit from other classes?
- What is the syntax to create a new class that inherits from another class?
- What is the purpose of “super”?