Intro To JavaScript

Part 3

IIFE - Immediately Invoked Function Expression

```
var counter = (function() {
  var i = 0;
  return {
    get: function() { return i; },
    set: function( val ) { i = val; },
    increment: function() {
      return ++i;
    }
  };
}());
```

IIFE - Immediately Invoked Function Expression

```
var trafficLight = (function() {
  var state = "red";
  var name  = "Peachtree and North Ave";
  return {
    getName  : function() { return name;       },
    getState : function() { return state;      },
    goGreen  : function() { state = "green";   },
    goYellow : function() { state = "yellow"; },
    goRed    : function() { state = "red";     }
  };
}());
```

What is the value of “trafficLight?
What does the return statement return?
What is the scope of the “state” and “name” vars?

```
trafficLight.goYellow();
console.log('The light at ' +
            trafficLight.getName() + ' is ' +
            trafficLight.getState());
```

Closures

```
var sum = function(a, b) { return a + b; };
console.log(sum(3, 5));

var adder = function(a) {
  return function(b) { return a + b; };
};
adder(3)(5);  // what are the values for a and b?

var add3 = adder(3);  // what is add3?

add3(5); // where does the 5 go?
```

OOP - The JavaScript Way

There is no class!

Object Literals

```
// Simple way to create an object.
var miko = {
  name: “Miko”,
  speak: function() { return this.name + “ says woof.”; }
};
```

So why not just use Object Literals
whenever we need to create an object?

How do we create another dog?

```
// Creating another dog.
var meisha = {
  name: “Meisha”,
  speak: function() { return this.name + “ says woof.”; }
};
```

That’s not very DRY!

There has to be a better way!

Object Constructors

```
// Dog constructor
function Dog(name) {
  this.name = name;
  this.speak = function() { return this.name + " says woof."; };
}


// Create some dogs
var miko = new Dog("Miko");
var meisha = new Dog("Meisha");

// Make them speak
console.log(miko.speak());    // Miko says woof.
console.log(meisha.speak());  // Meisha says woof.

// What does this do?
var snoopy = Dog("Snoopy");
```

JavaScript Prototypes

```
// Dog constructor
function Dog(name) {
  this.name = name;
//this.speak = function() { return this.name + " says woof."; };
}
// What is this???
Dog.prototype.speak = function() {
  return this.name + " says woof.";
};

// Create some dogs
var miko = new Dog("Miko");
var meisha = new Dog("Meisha");

// Make them speak
console.log(miko.speak());    // miko says woof.
console.log(meisha.speak());  // meisha says woof.
```

OOP - The JavaScript Way

- There are no cookie cutters!
- There are only cookies!

- Well, I guess you could say that the constructor functions are the cookie cutters.

Classes vs. Constructors and Prototypes

```
class Person
  attr_accessor: firstName
  attr_accessor: lastName

  def initialize(firstName,
                 lastName)
    @firstName = firstName
    @lastName = lastName
  end

  def fullName
    @firstName +
      " " + @lastName
  end
end

john = Person.new("John", "Doe")
john.fullName # "John Doe"
```

```
function Person(firstName,
                lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}

Person.prototype.fullName =
  function() {
    return this.firstName +
      " " + this.lastName;
}

var john = new Person("John", "Doe");
john.fullName(); // "John Doe"
```

Lab #3a - NodeJS

- Create a Pet Constructor Function
  - pet's name
  - owner's name
  - pet's age
  - pet's species
    - toString method on prototype that returns a string like this:
    - Snoopy is Susan's 3 year old dog

```
var snoopy = new Pet("Snoopy", "Susan", 3, "Dog");
console.log(snoopy.toString());
```