Homework 4

With **Callback** functions we can pass in functions as arguments to other functions. For example, if we have a function specifically for callback called:

function invokeCallback(cb){

cb();

}

The use of cb is as a reference point. Now we have a function that prints out ‘Hello World!’

function hello(){

console.log("Hello World!");

}

We can pass the function of hello into the argument of invokeCallback like so:

invokeCallback(hello);

And we get back ‘Hello World!’. Basically what’s happening is that function hello is getting placed where cb is like so:

function invokeCallback(hello){

cb();

}

Then it is going back to check if hello is a thing, it’ll move back up to where cb(); is and look at the argument again, change the cb() into the hello function and then do whatever is in the hello function!

**Closure**’s are functions that can return functions inside of itself. Say we have a function called hello:

function hello(){

}

We can then return a function inside of it and have it print something like so:

function hello(){

return function(){

console.log("hello world!");

};

}

var world = hello();

world();

Let’s make a variable called world and set it to the hello function. When we invoke world we get back ‘Hello World!’ What’s happening is we give the function hello() the value that returned from the returned function.

The **arguments** is an object that acts like an array but it’s not, it doesn’t have any of the array properties but .length. We can use this to find and add together the total number of something in an array that we don’t know.

**Recursion** is essentially another way of looping over a function. With very big functions it’s better to use recursion then to looping with a for loop. The function is basically calling itself unlike a closure which is returning another function.

The **prototype** property can get pretty confusing as every object has a prototype and every prototype is a property. However, we are just going to focus on the prototype property. This property allows us to add new properties and new methods to a prototype. Say, for instance we wanted to add a new method to Programmer that makes them say ‘Hello World!’ let’s start with

function Programmer(first, last, age){

this.firstname = first;

this.lastname = last;

this.age = age;

}

Next we are going to add our prototype property

Programmer.prototype.hello = function(){

console.log(“Hello World!”);

};

Then we’ll create a new programmer and have him say hello!

var newProgrammer = new Programmer("john", "Doe", 30);

newProgrammer.hello();

Our new programmer is now saying ‘Hello World!’. There is a number of prototype properties to mess around with but the important thing to remember is you should NEVER modify the prototypes of standard JavaScript objects, you can however modify your own.

**Constructors** are basically templates for all the different objects that we want to create and use. They use the keyword .this and owns the value of an object if it’s in the object. To see the standard way to create an object type look at the prototype example.