More JavaScript

JavaScript and OOP

□ JavaScript is not an *really* object oriented programming language

- Does not support data abstraction in the form of Classes
- There is no support for data protection

JavaScript and OOP

□ Values of the type object are arbitrary collections of properties, and we can add or remove these properties as one pleases

□ An object can be created in various ways:

- Bracketed notation:

  -
    ```
    var day = {
    squirrel: false,
    events: ["work", "touched tree", "pizza",
    "running", "television"]
    };
    ```
- The classical way:
  ```
  var day = new Object;
  day.squirrel = false;
  ```
- …

User Defined Objects

```
Incr = new Object();
Incr.count = 0;
Incr.increment = function(inc) {
    if (inc == undefined) {
        inc = 1;
    }
    this.count += inc;
    return this.count;
}
```

User-defined objects

- You may add, remove, or alter your object’s properties and methods at any time:
- The syntax for changing the value of a property is:

  `object.property = expression`

  - ***object*** is the JavaScript name of the object you want to manipulate
  - ***property*** is a property of that object
  - ***expression*** is a JavaScript expression that assigns a value to the property

Functions Can Have Properties

```
function plus1(value) {
    if (plus1.invocations ==
undefined) {
        plus1.invocations = 0;
    }
    plus1.invocations++;
    return value+1;
}
```

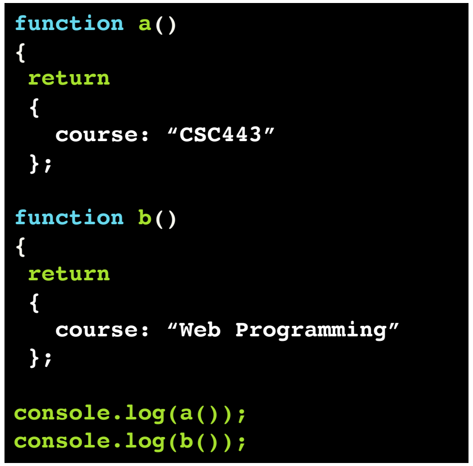
- More of this later!

Constructor

```
function Rectangle(width, height) {
    this.width = width;
    this.height = height;
}


r = new Rectangle(26, 14);
```

On Curly Braces and Semicolons

- Where should the curly braces of a function go:
  - Should be on the same line, or should it be on the next line?
- JavaScript automatically appends a ; at the end of a line



```
function a()
{
 return
 {
   course: “CSC443”
 };

function b()
{
 return
 {
   course: “Web Programming”
 };

console.log(a());
console.log(b());
```

On Curly Braces and Semicolons

- □ Both functions return undefined!
- □ Why?

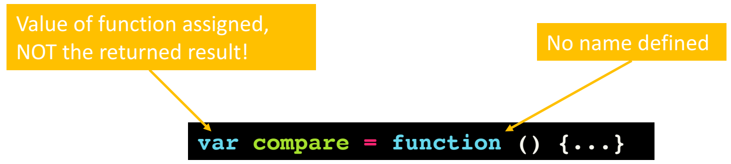
```
function a()
{
 return
 {
   course: “CSC443”
 };

function b()
{
 return
 {
   course: “Web Programming”
 };

console.log(a());
console.log(b());
```

JavaScript Functions

- □ A function is defined in JavaScript in a very similar to defining a variable
  - Define a function in JavaScript by either using the keyword `function` followed by the function name.
  - Create a variable and set it equal to a function
    - No name is defined after the function keyword
- □ So after either of these definitions you can now invoke, otherwise known as execute, function by referring to it by it's name `compare`.

JavaScript Functions

```
function compare () {...}
```


Value of function assigned, NOT the returned result!

No name defined

```
var compare = function () {...}
```

JavaScript Functions

```
function compare (x, y) {
 return x > y;
}
```

```
function compare (x, y) {...}
var a = compare(4, 5);
compare(4, "a");
compare();
```

Both are legal!

Scope

- Global
  - Variables and functions defined here are available everywhere
  - Variables and functions defined here are available only within this function

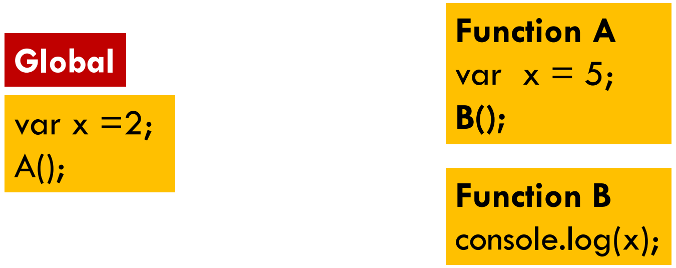
Scope Chain

- Everything is executed in an **Execution Context**
- Function invocation creates a new Execution Context
- Each Execution Context has:
  - Its own *Variable Environment*
  - Special 'this' object
  - Reference to its *Outer Environment*
- Global scope does not have an *Outer Environment* as it's the most outer there is

Scope Chain

- Referenced (not defined) variable will be searched for in its current scope first. If not found, the Outer Reference will be searched.
- If not found, the Outer Reference's Outer Reference will be searched, etc.
- This will keep going until the Global scope.
- If not found in Global scope, the variable is undefined.

What is the Value of x?



Global

```
var x =2;
A();
```

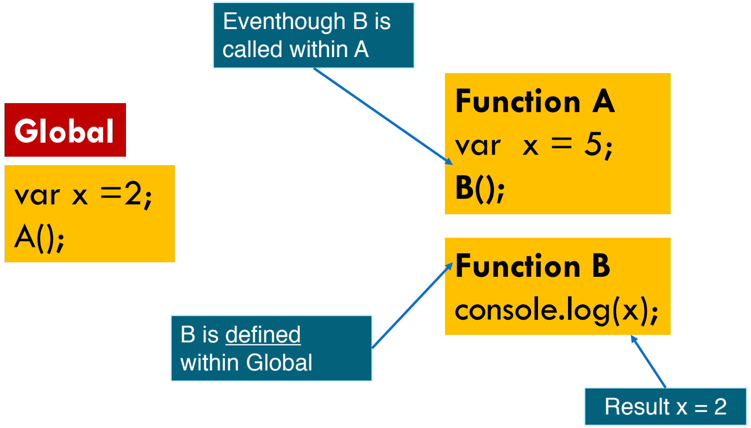
Function A

```
var  x = 5;
B();
```

Function B

```
console.log(x);
```

What is the Value of x?



JavaScript Functions

- Functions in JavaScript are what's called, first class data types
  - What that means is, is that whatever you could do with the variable, whatever you could do with an object you could also do with the function

Functions in JavaScript are objects.

- Function can be:
  - Passed around
  - Assigned it to a variable (we just saw this!)
  - Passed as an argument to another function
  - Returned as a return result from a function

Functions in JavaScript are Objects!

- Functions in JavaScript are regular objects that have some special properties to them
  - Can set properties on them just like we set properties on objects

```
function multiply(x, y) {
 return x * y;
}
```

```
multiply.version = “v.1.0.0”;
```

Function Factory

Functions in JavaScript are objects

```
function multiply(x, y) {
 return x * y;
}

function makeMultiplier(multiplier) {
 var myFunc = function (x) {
       return multiplier * x;
 };
 return myFunc;
}

var MultiplyByFour = makeMultiplier(4);       // What is the output?

MultiplyByFour(10);                           // How about this one?
```

Immediately Invoked Function Expression (IIFE)

- Recall executing functions in JavaScript:

```
function compare (x, y)
{...};
...
compare(3,5);
```

```
var compare = function (x, y)
{...}
...
compare();
```

- But since a function is an Object:

```
(
 function (x, y) {...};
)(3,5);
```

- The function will be *immediately invoked!*
  - Produces a function object!