**Week 3 Notes – JavaScript**

**MAJOR FEATURES OF JAVASCRIPT**

Static and Dynamically Typed Language

C/C ++

* Compiled, compiler tells you what you did wrong and stop compiling
* The compiler statically checks types, does not run code

JavaScript

* Dynamically typed,
  + Waits until a variable is used or a function is called to see if types are correct
* Upsides:
  + If you don’t specify a variable’s type before you run the program, you have more flexibility
  + Can have functions that can take a lot of different types and do things based on the

type of thing passed (in C++ it would probs be easier to write a display function for each thing passed)

Duck Typing

(In C++ if you wanted a duck, you make a class duck with duck properties and duck metods, then write functions that accepted duck objects as args)

Duck typing means that there are not explicit types, JavaScript only cares about functionality. As long as the function doesn’t throw an error, JavaScript does not care. e.g.

Function makeQuack(foo) {

foo.quack();

foo is not necessarily a duck, but as long as calling the method .quack() works, nbd for JS

Interpreted vs Compiled

C/C++ is compiled, you write it, compile it, and you get an executable in machine code

JavaScript is interpreted, the code will essentially be scanned as it is run and the computer will interpret the code on the fly.

Interpreted languages have the positive of being able to stop execution, modify code, then continue running, without having to compile

* On the flipside, there is nothing to force you to check your syntax before deployment

**JAVASCRIPT DEVELOPMENT ENVIRONMENT**

Pure JavaScript

Using Node.js

* Node.js is an interpreter

Using a browser

* Can create a skeleton HTML page that loads the JS when the page is loaded

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>title</title>

</head>

<body>

<script src="yourscript.js"></script>

</body>

</html>

Above is skeleton html, yourscript.js would be replaced with the script to execute

JavaScript with HTML

Can use JS combined with HTML to make dynamic web pages

JavaScript with Ajax

Chrome doesn’t allow you to make Ajax calls to get files on localhost, so it has to be running on a local webserver (use OSU engineering server for this)

**JAVASCRIPT TYPES**

Type Conversion

A variable can hold two different types of values at two different times

* JS is forgiving with its types, and will attempt to make the program run without errors
* JS may attempt to convert types to other types when you use an operator between different types (e.g. 5 + ‘Hello world’ converts the number to a string) this can happen with mixed results

Primitive Types

* Number
* String (there are simple and complex strings. Complex strings are objects)
* Boolean
* Undefined
* null (actually object)

type(null) returns ‘object’, this is due to a bug in JavaScript, and hasn’t been updated so far because it is hard to just make a change like that

Objects

var notAPrimitive = {property1:“foo”

property2:5}; //Fancy Object Literal Notation

notAPrimitive is an object, with the properties property1, a string, and property2, a number

* You can nest objects as deep as you want (objects within objects)

var notAPrimitive = {property1:"foo",

property2:5,

anotherObject:{propertyA:"I am nested"}

}; //Fancy Object Literal Notation

The last thing the typeof operator can return is “function”, e.g.

var add = function(a,b){ return a + b; }

typeof(add) returns function

Strings and strings

‘string’ and ‘String’ are two different things, ‘String’ is the constructor of a String object, ‘string’ is a primitive type

**FUNCTIONS IN JAVASCRIPT**

Function Declaration

function square(x) {

return x \* x;

}

Apparently similar to C/C++ function definition. The return type is replaced with the keyword ‘function’

Calling square() without an argument will not throw an error in JavaScript (would in C++) but would throw an error when it reached return x\*x because x was not defined. Could have the function check and work fine without passing a number

* You can call a function from anywhere within its scope

Function Definitions

A variable can be of type *pointer* *to function taking an int returning a pointer to char* in C++. JS can do it too, but it’s easier done, by:

var square = function(x) {

return x \* x;

};

console.log(square(12));

// 144

* Above we assign a function expression to a variable
  + Both this function and the other above can be called with square(number)
  + But, with this type of function there is no function hoisting, so you cannot call this function before assignment

**JAVASCRIPT OBJECTS AND OBJECT NOTATION**

Objects in JavaScript

Objects in JS are closer to structs than actual objects. No public or private properties

Example object in JS:

var day1 = {

squirrel: false,

events: ["work", "touched tree", "pizza", "running",

"television"]

};

The format is {key1: value1, key2: value2, ... keyn: valuen}

* You can use strings with invalid identifier symbols like spaces if you put them in quotes. Don’t tho

Can access content in two ways, by accessing with ‘.’ or by treating the object like a map i.e .:

* console.log(day1.squirrel) will log false to the console
* console.log(day1[“squirrel”]) will log false to the console

day1 is an REFERENCE to an object, not an object itself, if you did var otherDay = day1, otherDay and day1 would both be referencing the same object, though you could could change

otherDay = {foo:”bar”}; without updating day1

JSON

* The notation of key/value pairs inside curly braces is the basis of the JSON notation
  + There are few differences between JSON and object notation
    - The key must have quotation marks around it
    - You cannot have a function in JSON notation as JSON is concerned with data

For … In

for (*variable* in *object*) {

...

}

ex.

for (var prop in obj) {

console.log("obj." + prop + " = " + obj[prop]);

}

var obj = {a:1, b:2, c:3};

Gives output

obj.a = 1

obj.b = 2

obj.c = 3

Very similar to Python, obvs. In this case, prop holds the keys so for the first iter, ‘a’, and obj[‘a’] returns its value, so 1, next iter is ‘b’ and so forth.