JavaScript by Immersion

Brazos Valley Makers

June 1, 2015

1 About JavaScript

JavaScript is a prototypal, weakly typed, dynamically typed scripting language created for Netscape by Brendan Eich in 10 days in 1995[1]. All of these characteristics have specific implications for JavaScript, and we will explore each of them and others as we go along.

2 When in Doubt, Type it Out

This primer will take a kinesthetic, learning by doing, approach to JavaScript. As such, we will need some things, most notably a JavaScript REPL (Read Eval Print Loop, in our case, Node) and a browser with decent console capabilities (Chrome or Firefox will suffice).

2.1 I Seriously Hope You Have a Browser Installed Already

If not, there is no hope for you, but I heard they are serving refreshments somewhere.

2.2 Installing Node

2.2.1 On a Mac

```
$ ruby -e "$(curl -fsSL\
 https://raw.githubusercontent.com/Homebrew/install/master/install)
     "
```

[1] https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript

```
$brew install node
```

When installing node with homebrew, you will see the following message: “If you update npm itself, do NOT use the npm update command. The upstream-recommended way to update npm is: npm install -g npm@latest ”

This is nothing to worry about. It simply means, that when you want to update the node package manager, do it like so:

```
$npm install -g npm@latest
```

2.2.2 On Windows

Go to https://nodejs.org/download/ and download the installer. Run the installer.

3 Functions

Simply put, a function performs one or more operations. Functions can exist on their own. Functions can be use to define objects. Functions can be used to give objects behavior.

3.1 Functions Can Produce a Value

At the command prompt in a terminal window (Type “Terminal” in Spotlight in OS X or Press Windows + R to bring up the Run box and type cmd.exe to open a terminal), type in node and hit Enter. You should be greeted with the Node prompt (a “>”). Type in the following:

```
function addOne (n)  { return n+1; }
```

And press the *Enter* key.

Then type the following pressing enter at the end of the line:

```
addOne(1);
```

You should see:

```
2
>
```

From here on out, it is implied that you need to press the *Enter* key after typing in code to the Node prompt.

3.2 Functions Don't Need a Name

Type:

```
(function () { return 'You never even call me by my name.'; })();
```

3.3 Functions Can be Function Parameters

Type:

```
function after () {
    console.log('after');
}

function before (callback) {
    console.log('before');
    callback();
}

before(after);
```

This feature is taken advantage of in Continuation Passing Style.

4 Numbers

Type:

```
typeof 1;
typeof 1.1;
```

JavaScript is very egalitarian when it comes to numbers. It treats them all as floats.[2] It will print numbers with nothing after the decimal place as integer numbers though.

Side note: *typeof* is a unary operator, not a function, so you don't need parentheses to use it.

4.1 Arithmetic

Type:

```
1 + 2 * 6;
(1 + 2) * 6;
```

Arithmetic operations are performed from left to right in precedence order. Expressions in parentheses are evaluated first.[3]

[2] http://speakingjs.com/es5/ch11.html

[3] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence

5 Strings

Type the following at the Node prompt:

```
'Hola';
"Mundo";
```

Strings are basically just text enclosed in quotation marks.[4]

Type:

```
'one' + 'two' + 'three';
```

As you can see, strings can be composed from smaller strings. This is called string concatenation.

Try:

```
'hello world'.replace('world', 'college station');
```

JavaScript gives us all kinds of neat things we can do to string primitives courtesy of type coercion and the String constructor.[5]

Try:

```
"Hook 'Em".toLowerCase();
'whoop'.toUpperCase();
```

A *string primitive* is a value that is a string. A *string literal* is literally "a string".

6 Objects

Type:

```
({ 'sayHi' : function() { return 'Hi'; } }).sayHi();
```

You just made your first object and made it talk, congrats! Everything up to and including the left-most { and the right-most } is whatâĂŹs referred to as an *object literal*.

Everything that is not a number, a boolean value (true or false), null, or undefined is an object. Objects can be created using literals like above or *new*-ed using a constructor function.

Try:

```
typeof 'abc';
typeof new String('abc');
typeof String('abc');
typeof String(1);
```

It is important to note that the usage of *new* with one of the wrapper constructors for the primitive types with yield an object. Using one of the wrapper constructors without *new* works as a cast.

---

[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

[5] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var

7 Assignment

Try:

```
function spacesFromWord (word) {
    var arr = word.split(/[a-zA-Z]/);
    return arr.join(' ');
}

spacesFromWord('word');

console.log(arr);

function dashesFromWord(word) {
    arr = word.split(/[a-zA-Z]/);
    return arr.join('-');
}

console.log(arr);
```

Variables are a bit, well, interesting in JavaScript. Not really in a good way either. When the *var* keyword is used, the scope of the variable declared is the containing function if declared inside a function.[6] Because everything happens inside a global context, we want to be very careful to always use *var* when declaring a variable.

Alternatively, you might try:

```
function tildesFromWord (word) {
    return word.split(/[a-zA-Z]/).join('~');
}
```

Since assignments are a potential source of errors in your code, you might consider only using them when the task you are trying to perform is sufficiently complex enough to warrant their use.

8 Arrays

Try the following:

```
[1, 2, 3];
[1, 2, 3].join(',');
[1, 2, 3].pop();
var arr = [1, 2, 3];
arr.push(4);
arr;
arr.sort(function(a, b) { return a > b ? - 1 : a  < b ? 1 : 0; })
```

Arrays in JavaScript are containers whose elements can be accessed by their index. Array indicies in JavaScript are zero-based.

Try:

[6] http://speakingjs.com/es5/ch11.html

```
[1,2,3][0];
[1,2,3][1];
```

9 Loops

Loops lie at the heart of every daemon, video game, desktop app, and collection processing routine under the sun. We'll cover loops next.

9.1 For

Type the following into the node prompt hitting *Enter* after each line.

```
function countToTen () {
    for (var i = 1; i <= 10; i++) {
        console.log(i.toString());
    }
}

countToTen();
```

A *for* loop consists of an initializer, a test, an incrementer, and a body.

Above, *var i = 0;* is the initializer. *i <= 10;* is the test. *i++* is the incrementer. The body of the for loop is *console.log(i.toString());*.

var i = 0; is also an assignment statement.

9.2 Other Types of Loops

The *while* and *do-while* loops are other types of loops available in JavaScript, but everything that can be done with those can be done with a for loop, so we won't be going into them in this primer.

10 Control Flow

Control flow statements do as advertised: they control the flow of program execution.

10.1 If and If-else

Type in the following at the prompt:

```
function addTwo (n) {
    if ( isNaN(n) ) {
        console.log("not a number");
        return n;
    }
    return n + 2;
}

addTwo(1);

addTwo('r');
```

11 Truth is Stranger Than Fiction

Try typing in the values in the left hand column.

Expression	Value [7]
'1'== 1	true
'1'=== 1	false
0 == false	true
0 === false	false
1 == true	true
1 === true	false
''== false	true
undefined == null	true
undefined === null	false
'true'== true	false
'1'== true	true

Equality in JavaScript is also interesting. In JavaScript, there are two kinds of equality: equality (via the == operator) and strict equality (via the === operator). Strict equality does a type check first. If the types of the operands (the terms on the left and right of the ===) don't match, the value of the expression is *false*.

12 Brief Introduction to Dev Tools

Let's look at a simple todo list app.

TODO: expand on this

13 Brief Introduction to AJAX

Get the example files by entering the following command at a regular (i.e. not Node) command prompt:

```
cd ~
```

or the following if Windows is the operating system:

```
cd %HOME%
```

```
git clone git@github.com:bvmake/classes.git
```

This will clone the classes folder into your home directory.

If you are uncomfortable with git, there is a "Download ZIP" option on `https://github.com/bvmake/classes`.

In order to run the todo list app from `https://github.com/bvmake/classes/tree/master/javascript_immersion/todo`, we are going to need to use the Node Package Manager to get a few things.

Enter the following commands at the regular command prompt:

```
$npm install sqlite3

$npm install uuid
```

Once those packages have downloaded, run the app like so:

```
node todo.js
```

TODO: expand on this