JavaScript Basics

Mendel Rosenblum

What is JavaScript?

From Wikipedia:

> ... **high-level**, **dynamic**, **untyped**, and **interpreted** programming language
>
> ... is **prototype-based** with **first-class functions**, …
>
> ... supporting **object-oriented**, **imperative**, and **functional programming**
>
> ... has an API for working with **text**, **arrays**, **dates** and **regular expressions**

- Not particularly similar to Java: More like C crossed with Self/Scheme
  - C-like statements with everything objects, closures, garbage collection, etc.
- Also known as ECMAScript (short ES)

Some thoughts about JavaScript

- Example of a **scripting language**
  - Interpreted, less declaring of things, just use them
- Seems like it was designed in a rush
  - Some “Good Parts”, some not so good
  - Got bad reputation



- Many programmers use a subset that avoids some common problems
  - `"use strict";` tweaks language to avoid some problematic parts
- Language being extended to enhance things: New ECMAScript every year!
  - Transpiling common so new features used: e.g ECMAScript Version N, TypeScript
- Code quality checkers (e.g. `jslint`, `jshint`, `eslint`) widely used

Good news if you know C - JavaScript is similar

```
i = 3;
i = i * 10 + 3 + (i / 10);


while (i >= 0) {
  sum += i * i; // comment
  i--;
}


for (i = 0; i < 10; i++)  {

}
/* this is a comment */
```

```
if (i < 3) {
  i = foobar(i);
}
else {
  i = i * .02;
}
```

Most C operators work:

```
* / % + - ! >= <= > < && || ?:
```

```
function foobar(i) { return i;}
continue/break/return
```

JavaScript has **dynamic** typing

```
var i;  // Need to define variable ('use strict';), note: untyped

typeof i == 'undefined' // It does have a type of ‘undefined’

i = 32; //now: typeof i == typeof 32 == 'number‘

i = "foobar"; //now: typeof i == typeof 'foobar' == 'string‘

i = true; // now: typeof i == 'boolean'
```

- Variables have the type of the last thing assigned to it
- Primitive types: undefined, number, string, boolean, **function**, object

Variable scoping: Lexical/static scoping

Two scopes: Global and function local

```
var globalVar;
function a() {
  var localVar;
  if (globalVar > 0) {
    var localVar2 = 2;
    console.log(localVar2);
  }
  // localVar2 is valid here
};
a();
```

All `var` statements **hoisted** to top of scope:

```
function foo() {
  var x;
  x = 2;
// Same as:
function foo() {
  x = 2
  var x;
```

`localVar2` is hoisted here but has value undefined

Var scope problems

- Global variables are bad in browsers - Easy to get conflicts between modules
- Hoisting can cause confusion in local scopes (e.g. access before value set)

```
function() {
  console.log('Val is:', val);      // Val is: undefined
  ...
  for(var i = 0; i < 10; i++) {
      var val = "different string"; // Hoisted to func start
```

- Some JavaScript guides suggest always declaring all var at function start
- ES6 introduced non-hoisting, scoped **`let`** and **`const`** with explicit scopes
  Some coding environments ban **`var`** and use **`let`** or **`const`** instead

Var scope problems

- Global variables are bad in browsers - Easy to get conflicts between modules
- Hoisting can cause confusion in local scopes (e.g. access before value set)

```
function() {
  console.log('Val is:', val);      // Syntax error!
  ...
  for(let i = 0; i < 10; i++) {
      let val = "different string"; // Works!
```

- Some JavaScript guides suggest always declaring all var at function start
- ES6 introduced non-hoisting, scoped **`let`** and **`const`** with explicit scopes
  Some coding environments ban **`var`** and use **`let`** or **`const`** instead

`number` type

`number` type is stored in floating point (i.e. double in C)

$$\texttt{MAX_INT} = (2^{53} - 1) = 9007199254740991$$

Some oddities: `NaN`, `Infinity` are numbers

```
1/0 == Infinity
Math.sqrt(-1) == NaN
```

Nerd joke: `typeof NaN` returns `'number'`

Watch out:

```
(0.1 + 0.2) == 0.3 is false // 0.30000000000000004
```

bitwise operators (e.g. `~`, `&`, `|`, `^`, `>>`, `<<`, `>>>`) are 32bit!

string type

string type is variable length (no char type)

```
var foo = 'This is a test'; // can use "This is a test"
foo.length  // 14
```

+ is string concat operator

```
foo = foo + 'XXX'; // This is a testXXX
```

Lots of useful methods: indexOf(), charAt(), match(), search(), replace(), toUpperCase(), toLowerCase(), slice(), substr(), …

```
'foo'.toUpperCase() // 'FOO'
```

`boolean` type

- Either **`true`** or **`false`**
- Language classifies values as either **truthy** or **falsy**
  - Used when a value is converted to a `boolean` (e.g. `if (foo) { … }`)
- Falsy:

  `false`, `0`, `""`, `null`, `undefined`, and `NaN`

- Truthy:

  Not falsy (`all objects, non-empty strings, non-zero numbers, functions, etc.`)

undefined and null

- **undefined** - does not have a value assign

```
var x;  //  x has a value of undefined
x = undefined;  // It can be explicitly store
typeof x == 'undefined'
```

- **null** - a value that represents whatever the user wants it to

  Use to return special condition (e.g. no value)

```
typeof null == 'object'
```

- Both are falsy but not equal (`null == undefined; null !== undefined`)

`function` type

```
var foobar = function foobar(x) { // Same as function foobar(x)
  if (x <= 1) {
    return 1;
  }
  return x * foobar(x - 1);
}

typeof foobar == 'function';  foobar.name == 'foobar'
```

- Function definitions are hoisted (i.e. can use before define)
- Can be called with variable arguments
  - `Array arguments` variable (e.g. `arguments[0]` is first argument)
  - Unspecified arguments have value `undefined`
- All functions return a value (default is `undefined`)

function type (cont.)

```
// function definitions are hoisted
// (i.e. can use before define)
console.log(foobar) // undefined
var foobar = function foobar(x) {
  console.log(arguments[0]);
  if (x <= 1) {
    return 1;
  }
  return x * foobar(x - 1);
}
console.log(typeof foobar); // function
console.log(foobar(3)); // output: 3, 2, 1, 6
```

First class function example

```
var aFuncVar = function (x) {
  console.log('Called with', x); return x + 1;
};
function myFunc(routine) {  // passed as a param
  console.log('Called with',  routine.toString());
  var retVal = routine(10);
  console.log('retVal', retVal);
  return retVal;
}
myFunc(aFuncVar);
```

Output

```
Called with function (x) {
            console.log('Called with', x);
            return x+1;
          }
Called with 10
retVal 11
```

`object` type

- Object is an unordered collection of name-value pairs called **properties**

```
var foo = {};
var bar = {name: "Alice", age: 23, state: "California"};
```

Name can be any string: `var x = { "": "empty", "---": "dashes"}`

- Referenced either like a structure or like a hash table with string keys:

```
bar.name == bar["name"]
x["---"]  // have to use hash format for illegal names
foo.nonExistent == undefined
```

- Global scope is an object in browser (i.e. `window[prop]`)

Properties can be added, removed, enumerated

- To add, just assign to the property:

```
var foo = {};
foo.name = "Fred"; // foo.name returns "Fred"
```

- To remove use `delete`:

```
var foo = {name: "Fred"};
delete foo.name; // foo is now an empty object
```

- To enumerate use `Object.keys()`:

```
Object.keys({name: "Alice", age: 23}) --> ["name", "age"]
```

Arrays

```
var anArr = [1,2,3]; or var anArr = new Array(1,2,3)
```

Are special objects: `typeof anArr == 'object'`

Indexed by non-negative integers: `(anArr[0] == 1)`

Can be **sparse** and **polymorphic**: `anArr[5]='FooBar'; //[1,2,3,,,'FooBar']`

Like strings, have many methods: `anArr.length == 3`

`push, pop, shift, unshift, sort, reverse, splice, …`

Oddity: can store properties like objects (e.g. `anArr.name = 'Foo'`)

`[1, 2, 3, , , , "FooBar", name: "Foo"]`

Some properties have implications: (e.g. `anArr.length = 0;`) `// [name: "Foo"]`

Dates

```
var date = new Date();
```

Are special objects: `typeof date == 'object'`

The number of milliseconds since midnight January 1, 1970 UTC

Timezone needed to convert. Not good for fixed dates (e.g. birthdays)

Many methods for returning and setting the data object. For example:

```
date.valueOf() --> 1452359316314
date.toISOString() --> '2019-09-01T17:08:36.314Z'
date.toLocaleString() --> '9/1/2019, 9:08:36 AM'
```

Regular Expressions

```
var re = /ab+c/;    or  var re2 = new RegExp("ab+c");
```

Defines a pattern that can be searched for in a string

- String: `search()`, `match()`, `replace()`, and `split()`
- RegExp: `exec()` and `test()`

Uses:

- Searching: Does this string have a pattern I'm interested in?
- Parsing: Interpret this string as a program and return its components

Regular Expressions by example - search/test

```
/HALT/.test(str);    // Returns true if string str has the substr HALT
/halt/i.test(str);   // Same but ignore case
/[Hh]alt [A-Z]/.test(str); // Returns true if str either “Halt L” or “halt L”

'XXX abbbbbbc'.search(/ab+c/);   // Returns 4 (position of ‘a’)
'XXX ac'.search(/ab+c/);         // Returns -1, no match
'XXX ac'.search(/ab*c/);         // Returns 4

'12e34'.search(/[^\d]/);         // Returns 2, non-digit position
'foo: bar;'.search(/...\s*:\s*...\s*;/); // Returns 0, . (dot) – any single
                                         character, \s – white space
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Cheatsheet

Regular Expressions - exec/match/replace

```
var str = "This has 'quoted' words like 'this'";
var re = /'[^']*'/g;

re.exec(str);  // Returns ["'quoted'", index: 9,  input: …
re.exec(str);  // Returns ["'this'", index: 29,  input: …
re.exec(str);  // Returns null

str.match(/'[^']*'/g);  // Returns ["'quoted'", "'this'"]

str.replace(/'[^']*'/g, 'XXX'); // Returns:
                                'This has XXX words with XXX.'
```

Exceptions - `try/catch`

- Error reporting frequently done with exceptions

  Example:

  ```
  nonExistentFunction();
  ```

  Terminates execution with error:

  ```
  Uncaught ReferenceError: nonExistentFunction is not defined
  ```

Exception go up stack: Catch exceptions with `try/catch`

```
try {
  nonExistentFunction();
} catch (err) { // typeof err 'object'
  console.log("Error call func", err.name, err.message);
}
```

Exceptions - `throw/finally`

- Raise exceptions with `throw` statement

```
try {
  throw "Help!";
} catch (errstr) {  // errstr === "Help!"
  console.log('Got exception', errstr);
} finally {
  // This block is executed after try/catch
}
```

- Conventions are to `throw` sub-classes of `Error` object

```
console.log("Got Error:", err.stack || err.message || err);
```

Getting JavaScript into a web page

- By including a separate file:

```
<script type="text/javascript" src="code.js"></script>
```

- Inline in the HTML:

```
<script type="text/javascript">
//<![CDATA[
JavaScript goes here...
//]]>
</script>
```