Week 2 - Part 1

GETTING STARTED WITH (REAL) JAVASCRIPT

BACKGROUND

It has nothing to do with JAVA.

Originally developed by Brandan Eich for Netscape, first released in 1995.

(Brandan Eich is now CTO of Mozilla)

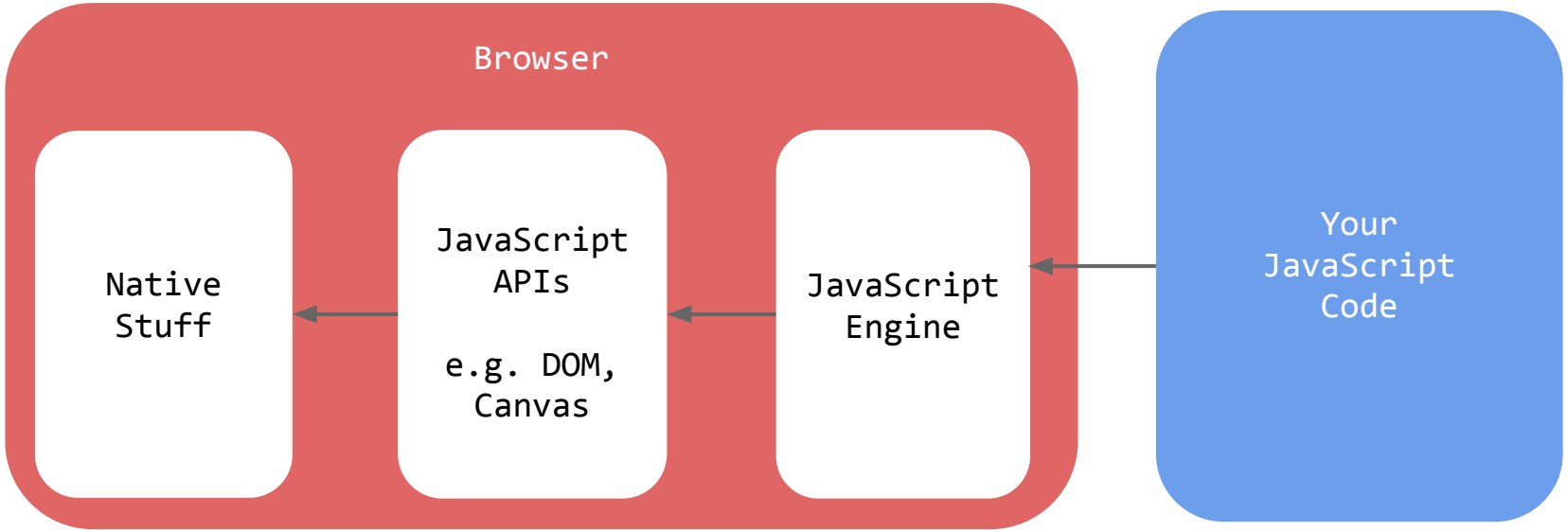
Today it has an international standard:

ECMAScript

Now at version 5, aka ES5.
ES5 is implemented in most modern browsers.
ES6 (Harmony) is work in progress.

It runs in Browsers, of course

But it can also run elsewhere, given there is a JavaScript engine that can interpret the code.



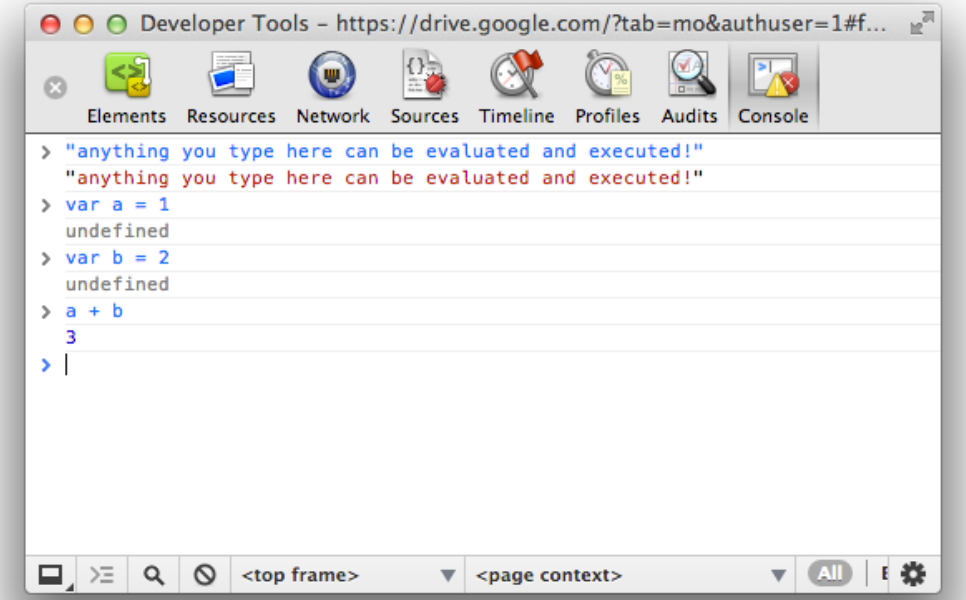
TOOLS

Pick a good editor

Don't use clunky IDEs for JavaScript.

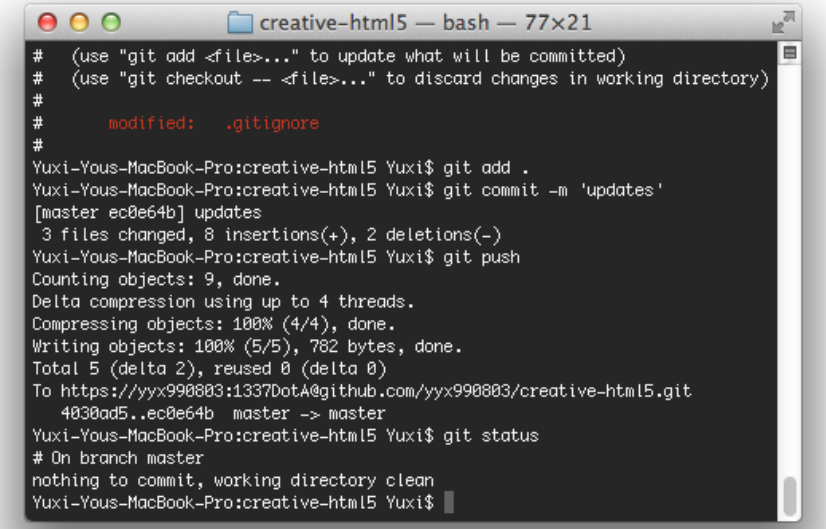
Browser Inspector

It is your best friend.



Command Line Tools

We'll talk about these later.



FUNDAMENTALS

VARIABLE

Create a variable with the `var` keyword

```
var something = 1
```

Variable names are case-sensitive, and cannot be language keywords.

```
var something = 1
var someThing = 2 // this is different.
var return = 1 // this will cause a syntax error because
                  'return' is a language keyword.
```

VARIABLE

You can assign something else to an existing variable:

```
something = "I'm a string now!"
```

Since variables can basically hold anything, we can do basic type-checking with `typeof`

```
typeof something // "string"
typeof 12345 // "number"
```

NUMBERS

No distinction between integers and float numbers, They are all Numbers.

```
var a = 1
a/3 // a is now 0.3333333333333333
```

The global `Math` object contains useful methods to work with numbers.

BOOLEANS

It's either true or false

```
var a = true
var b = false
```

Booleans are results of comparisons:

```
2 > 1 // true
3 >= 3 // true
3 != 3 // false
3 == '3' // true
3 === '3' // false. we will talk about these two later
```

BOOLEANS

You can use comparisons in `if` statements:

```
if (1 == 2) {
  // things here will never happen
}
```

You can also combine multiple of them:

```
1 == 1 && 2 == 2 // true (&& means AND)
1 == 1 && 1 == 2 // false
1 == 1 || 1 == 2 // true (|| means OR)
1 == 2 || 2 == 3 // false
```

STRINGS

Single-quote, double-quote, doesn't really matter

```
"aaa" == 'aaa' // true
"'aaa'" // you can use one inside the other
"\"123\"" // you need to backslash escape to use it in itself
```

Strings come with useful things

```
'abc'.length // 3
'abc'.slice(1,2) // 'b'
'abc'.indexOf('b') // 1
```

STRINGS

Be careful when using string methods, because strings are immutable.

```
var str = 'testing'
str.slice(0,4)
str // str is still 'testing'. Whyyyyyyy???
```

Once a string is created, they can't change. String operations always result in a new string being created.

```
var str2 = str.slice(0,4)
// str2 is 'test'
```

OBJECTS

Creating an object is dead simple in JavaScript:

```
var obj = {}
typeof obj // "object"
```

Objects hold properties that can be accessed with the dot syntax.

```
var obj = {
    luckyNumber: 7
}
obj.luckyNumber // 7
```

OBJECTS

You can nest objects inside one another:

```
var group = {
    team1: {
        name: 'Beavers'
    },
    team2: {
        name: 'Squirrels'
    }
}

group.team1.name // 'Beavers'
```

OBJECTS

You can dynamically add or change properties of an existing object:

```
var obj = {}
obj.luckyNumber = 7
obj.luckyNumber += 6 // obj.luckyNumber is now 13
```

You can delete properties, too, but it's rarely needed.

```
delete obj.luckyNumber // obj.luckyNumber is now undefined
```

OBJECTS

Object property names (a.k.a keys) can be Strings. And there is a useful alternative syntax to access the values.

```
var obj = {
    'this is a string': 123
}
obj['this is a string'] // 123

var key = 'this is a string'
obj[key] // 123
```

FUNCTIONS

A Function can contain a block of code that can be executed later and for multiple times. There are two slightly different ways to declare a function:

```
function doSomething (argument) {
    // do something here
}

var doSomethingElse = function (argument) {
    // do something else here
}
```

They are slightly different, but we'll leave that detail for later.

FUNCTIONS

You "invoke/call/execute" a function by adding a pair of parentheses after its name:

```
doSomething() // something is done
```

You can pass arguments inside the parentheses into functions, and get a 'return value' back:

```
function sum (a, b) {
    return a + b
}

var result = sum(5, 6) // result is 11
```

FUNCTIONS

You can, of course, call other functions inside a function. In addition, you can even call a function inside itself. Use with caution though, as this could result in an infinite loop.

```
function beStupid () {
    beStupid()
}

beStupid() // "RangeError: Maximum call stack size exceeded"
```

FUNCTIONS

You can pass a function as an argument into another function. The function being passed in is usually referred to as the 'callback function.'

```
function sayYes () { console.log('yes') }
function sayNo () { console.log('no') }

function doIt (callback) {
    // you can do some work here...
    callback() // then call the function passed in
}

doIt(sayYes) // 'yes'
doIt(sayNo) // 'no'
```

FUNCTIONS

You can also pass in a function declared on the fly (a.k.a anonymous function). This happens A LOT when using jQuery so you've probably seen this before.

```
function doIt (callback) {
    var result = 5
    callback(result)
}

doIt (function (result) {
    console.log('result is: ' + result)
})
```

SCOPE

When you use var not within any functions, you are defining a global variable:

```
var a = 'test' // this is in the global scope
```

All JavaScript files on the same web page share the same global scope. In fact, inside Browsers the global scope equals the window object.

```
window.a // 'test'
```

SCOPE

JavaScript is 'function scoped.' That means when a function is called, it creates a scope of its own.

```
var global = 1

function test () {
  var local = 2 // this is only accessible inside this
                // function, a.k.a local scope
  console.log(global) // You can still access global variables
  var global = 3 // You can override them locally.
                 // This will NOT affect the global scope.
}
```

SCOPE

If a function is declared on the fly inside another function, then it will have access to the scope in which it is declared in.

```
function parent () {
  var parentVar = 'Hi!'
  function child () {
      var childVar = 'Yo!'
      console.log(parentVar) // 'Hi!'
  }
  console.log(childVar) // undefined, because parent can't
                        // access child's scope
}
```

CLOSURE

By returning a function or object that has access to the local variables, you've created a closure. (see closure.js in code folder for more details)

```
function parent () {
  var parentVar = 'Hi!'
  return function () {
      console.log(parentVar)
  }
}

var logParent = parent()
logParent() // 'Hi!'
```

CODE STYLE

It's mostly personal preference, but making your code easy to read is a virtue. Treat it like your design craft :)

- Indent your code!
- Naming conventions
  - Use camel case
  - Use descriptive names
- Write comments, not only for others but also for yourself
- Semi-colons are optional