Javascript+DOM+CSS

Davide Morelli

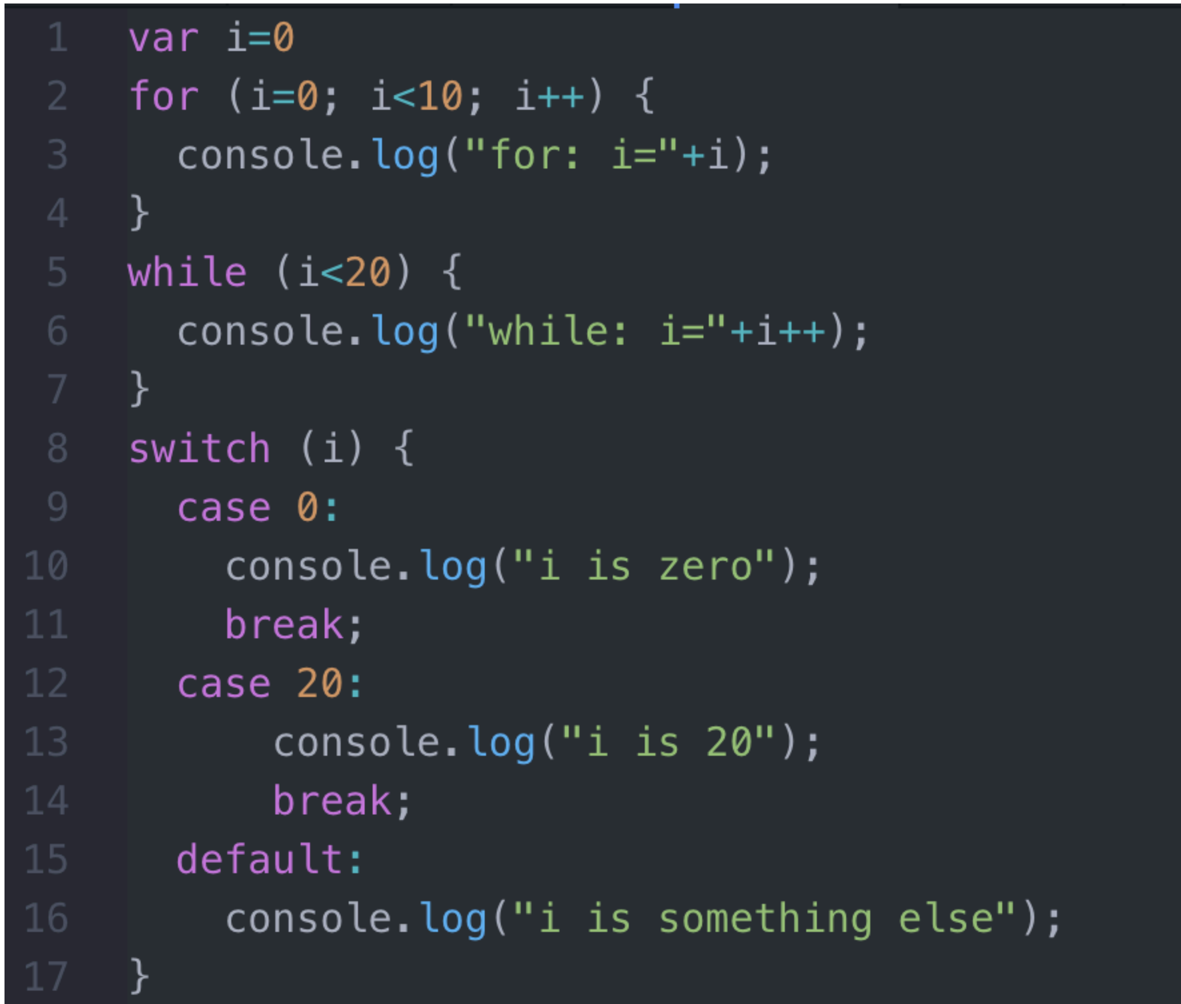
CI 2018

Javascript

- client side: Netscape in 1995
- DHTML 1997 (expressive DOM)
- server side: Jscript 1996, Node.js 2009
- standardized (ECMA)
- asm.js (js subset, performance)
  - https://jslinux.org/
  - WebAssembly
- docs: https://developer.mozilla.org/en-US/docs/Web/JavaScript

JS language

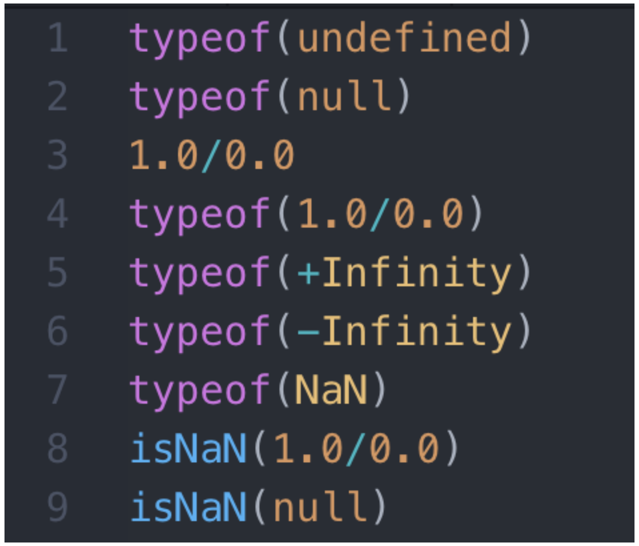
- imperative
  - if, for, while, switch, etc



```
var i=0
for (i=0; i<10; i++) {
  console.log("for: i="+i);
}
while (i<20) {
  console.log("while: i="+i++);
}
switch (i) {
  case 0:
    console.log("i is zero");
    break;
  case 20:
      console.log("i is 20");
      break;
  default:
    console.log("i is something else");
}
```

JS language

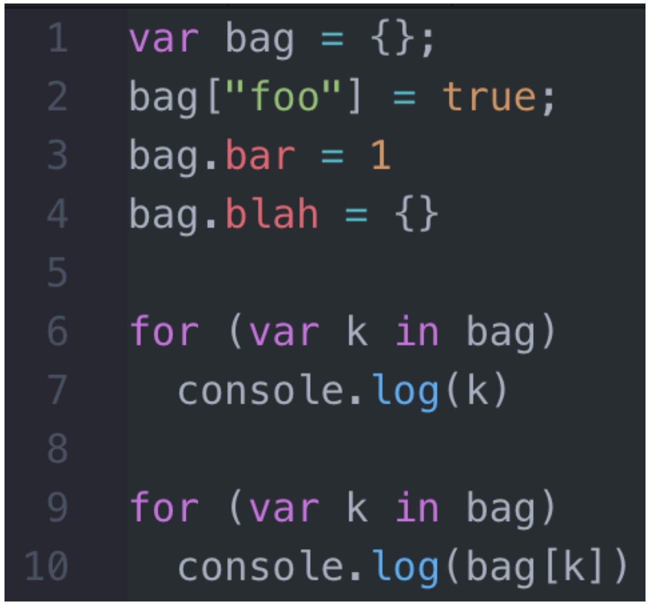
- data types
  - primitives
    - Null
    - Undefined
    - Boolean
    - Number
    - String
    - Symbol (ECMAScript 6)
  - objects



```
typeof(undefined)
typeof(null)
1.0/0.0
typeof(1.0/0.0)
typeof(+Infinity)
typeof(-Infinity)
typeof(NaN)
isNaN(1.0/0.0)
isNaN(null)
```

JS language

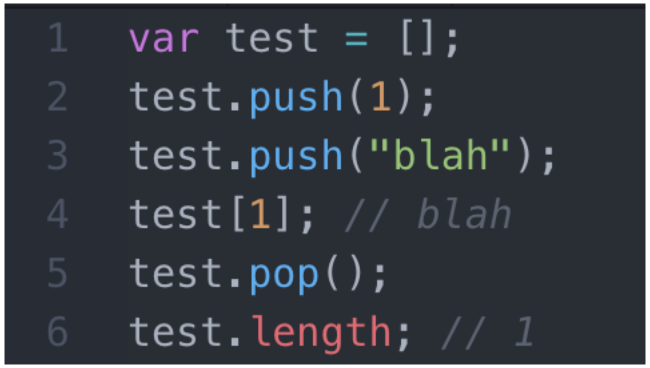
- objects
  - mappings between keys (a string or a Symbol) and values (anything)



```
var bag = {};
bag["foo"] = true;
bag.bar = 1
bag.blah = {}

for (var k in bag)
  console.log(k)

for (var k in bag)
  console.log(bag[k])
```

JS language

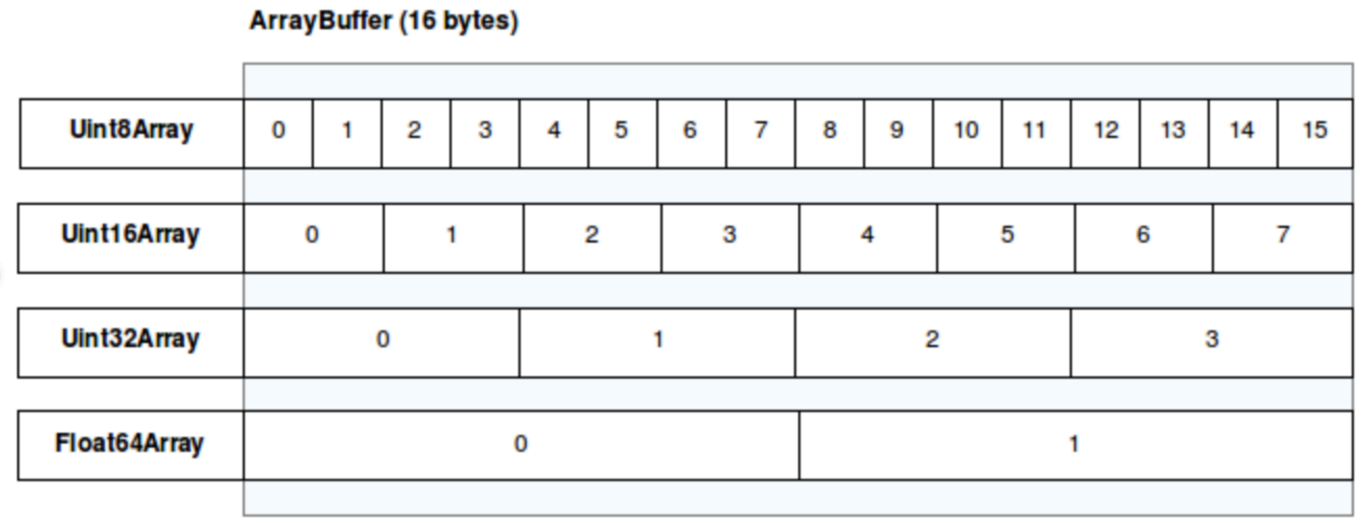
- objects
  - mappings between keys (a string or a Symbol) and values (anything)
  - Array



```
var test = [];
test.push(1);
test.push("blah");
test[1]; // blah
test.pop();
test.length; // 1
```

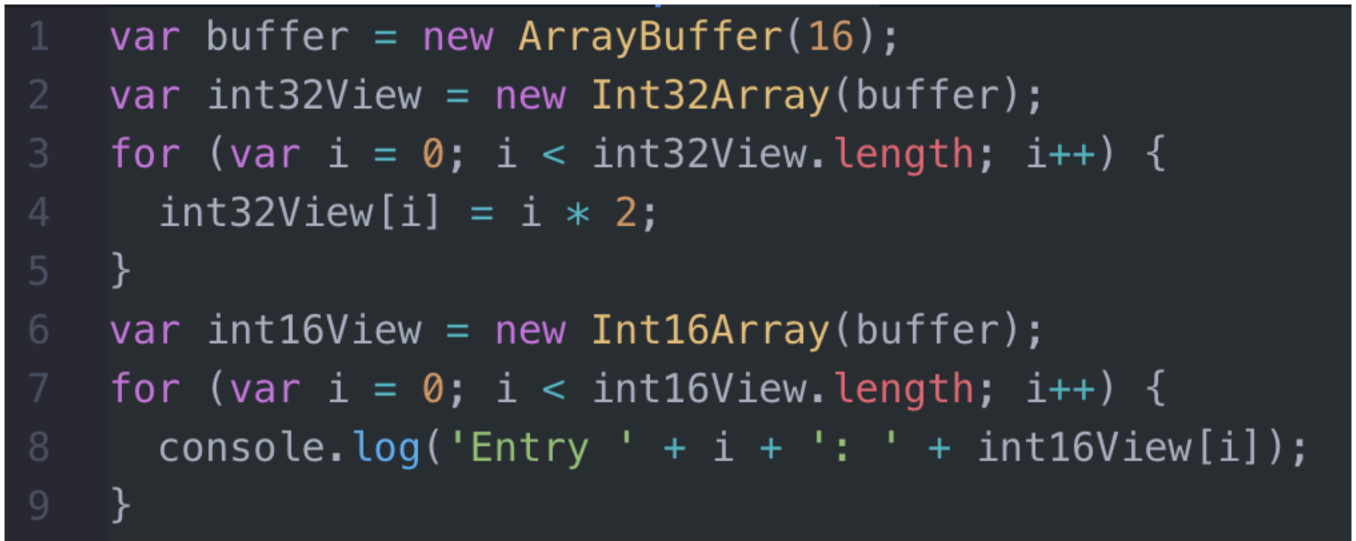
JS language

- objects
  - mappings between keys (a string or a Symbol) and values (anything)
  - Array
  - Typed Array (ECMAScript 2015)



JS language

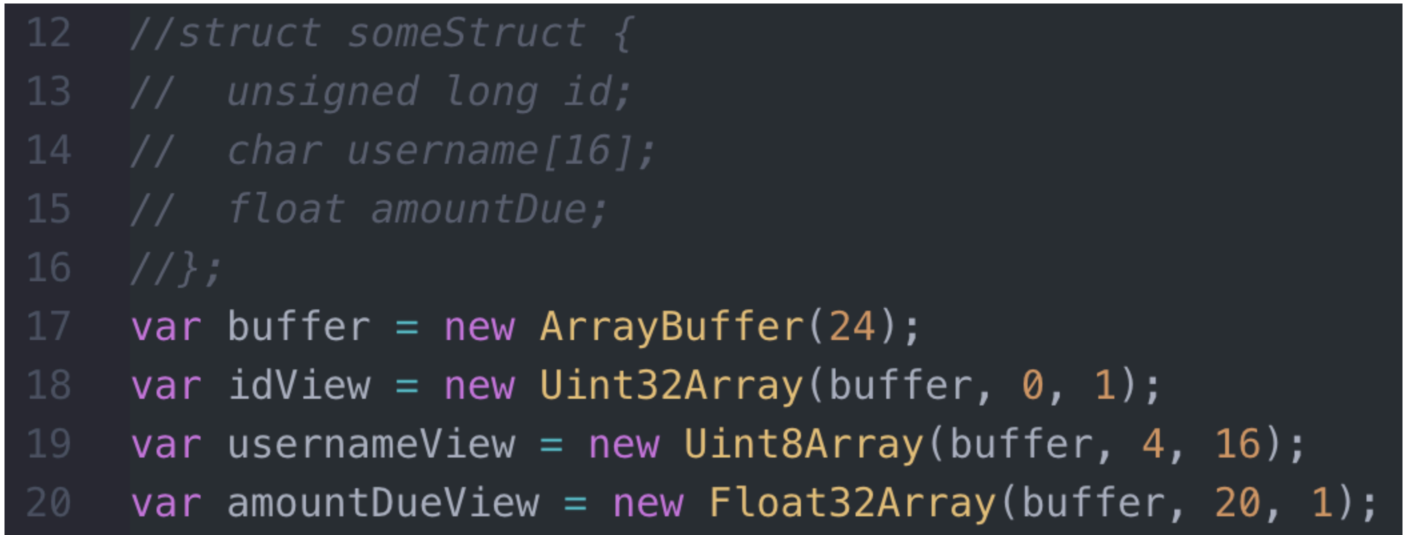
- objects
  - mappings between keys (a string or a Symbol) and values (anything)
  - Array
  - Typed Array (ECMAScript 2015)



```
var buffer = new ArrayBuffer(16);
var int32View = new Int32Array(buffer);
for (var i = 0; i < int32View.length; i++) {
  int32View[i] = i * 2;
}
var int16View = new Int16Array(buffer);
for (var i = 0; i < int16View.length; i++) {
  console.log('Entry ' + i + ': ' + int16View[i]);
}
```

JS language

- objects
  - mappings between keys (a string or a Symbol) and values (anything)
  - Array
  - Typed Array (ECMAScript 2015)



```
//struct someStruct {
//   unsigned long id;
//   char username[16];
//   float amountDue;
//};
var buffer = new ArrayBuffer(24);
var idView = new Uint32Array(buffer, 0, 1);
var usernameView = new Uint8Array(buffer, 4, 16);
var amountDueView = new Float32Array(buffer, 20, 1);
```

JS language

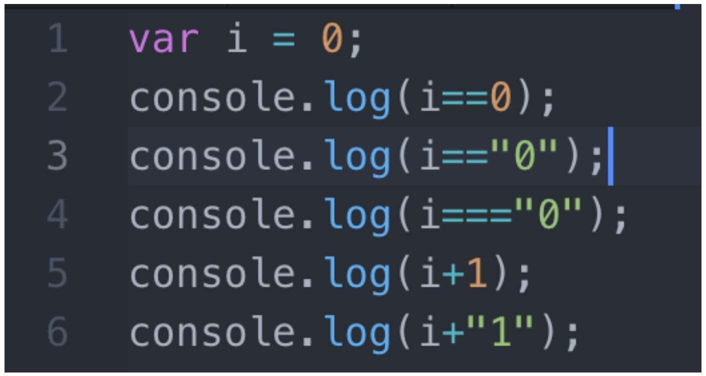
- objects
  - mappings between keys (a string or a Symbol) and values (anything)
  - Array
  - Typed Array (ECMAScript 2015)
  - Set, Weak Set

JS language

- objects
  - mappings between keys (a string or a Symbol) and values (anything)
  - Array
  - Typed Array (ECMAScript 2015)
  - Map, Set, WeakMap, WeakSet
  - JSON

JS language

- dynamic types



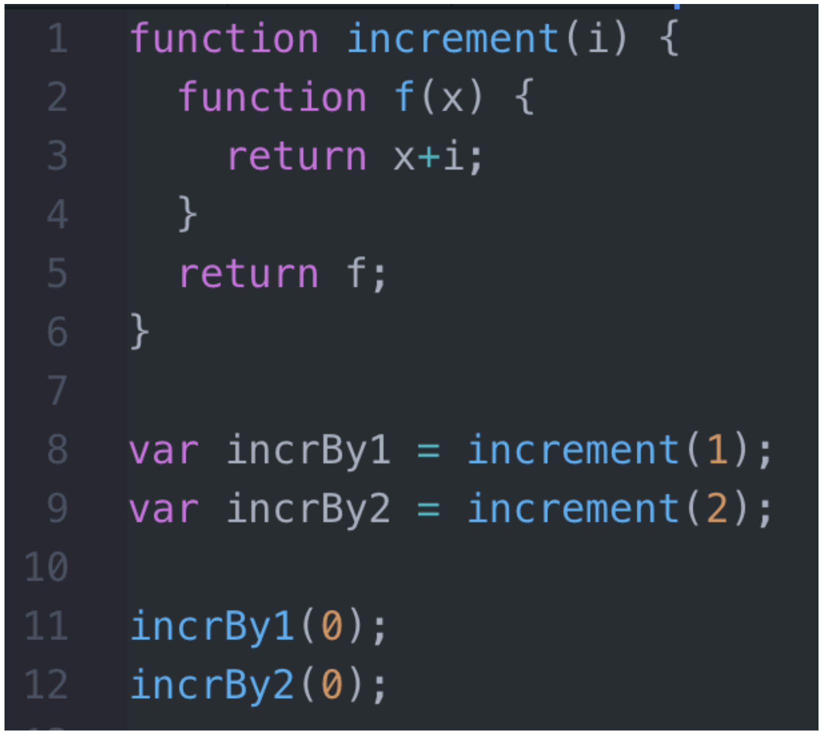
```
var i = 0;
console.log(i==0);
console.log(i=="0");
console.log(i==="0");
console.log(i+1);
console.log(i+"1");
```

JS language

- scoping
  - function scope
    - global scope if defined outside a function
    - local scope if inside a function
    - shadowing (local var with same name as global)
    - automatically global
  - hoisting
    - declarations (not assignments) are moved at beginning of scope
  - (ECMA 2015) blocks, let, const

JS language

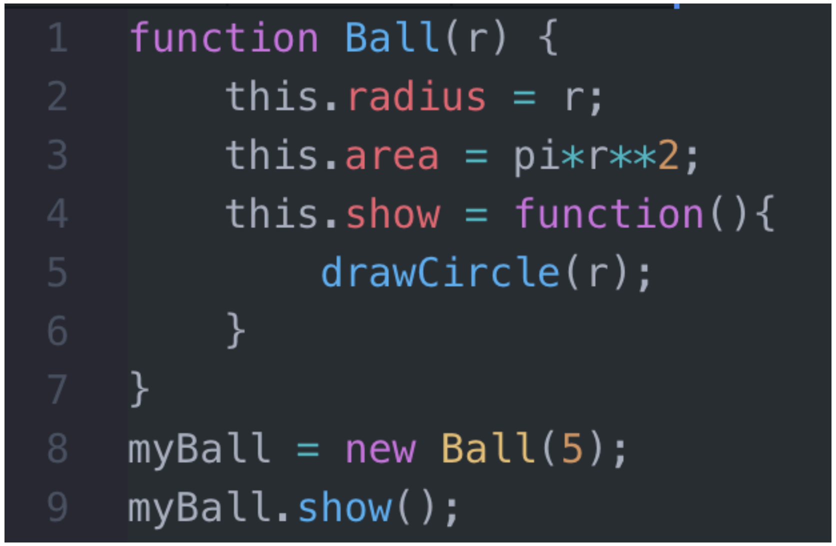
- functional
  - capture environment
  - closures



```
function increment(i) {
  function f(x) {
    return x+i;
  }
  return f;
}

var incrBy1 = increment(1);
var incrBy2 = increment(2);

incrBy1(0);
incrBy2(0);
```

JS language

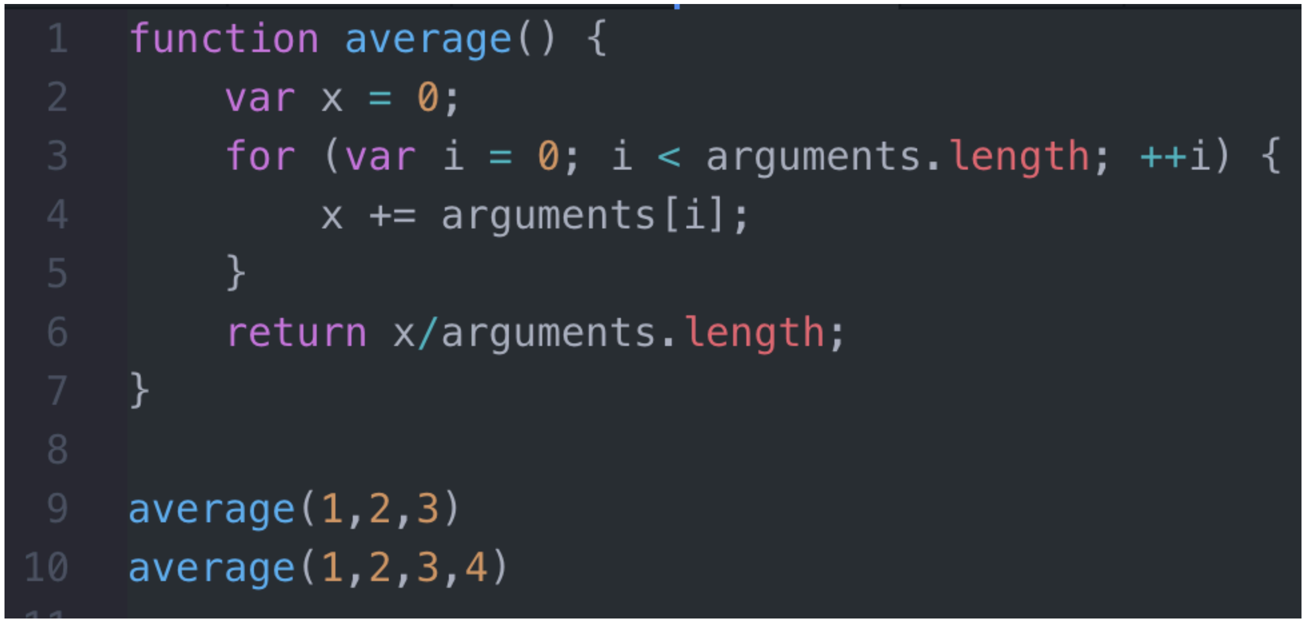
- functional objects
  - functions are objects (with the state of the variables)
  - can have properties



```
function Ball(r) {
    this.radius = r;
    this.area = pi*r**2;
    this.show = function(){
        drawCircle(r);
    }
}
myBall = new Ball(5);
myBall.show();
```

JS language

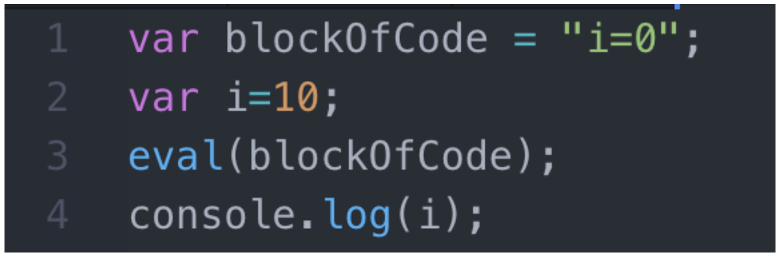
- variadic functions



```
function average() {
    var x = 0;
    for (var i = 0; i < arguments.length; ++i) {
        x += arguments[i];
    }
    return x/arguments.length;
}

average(1,2,3)
average(1,2,3,4)
```

JS language

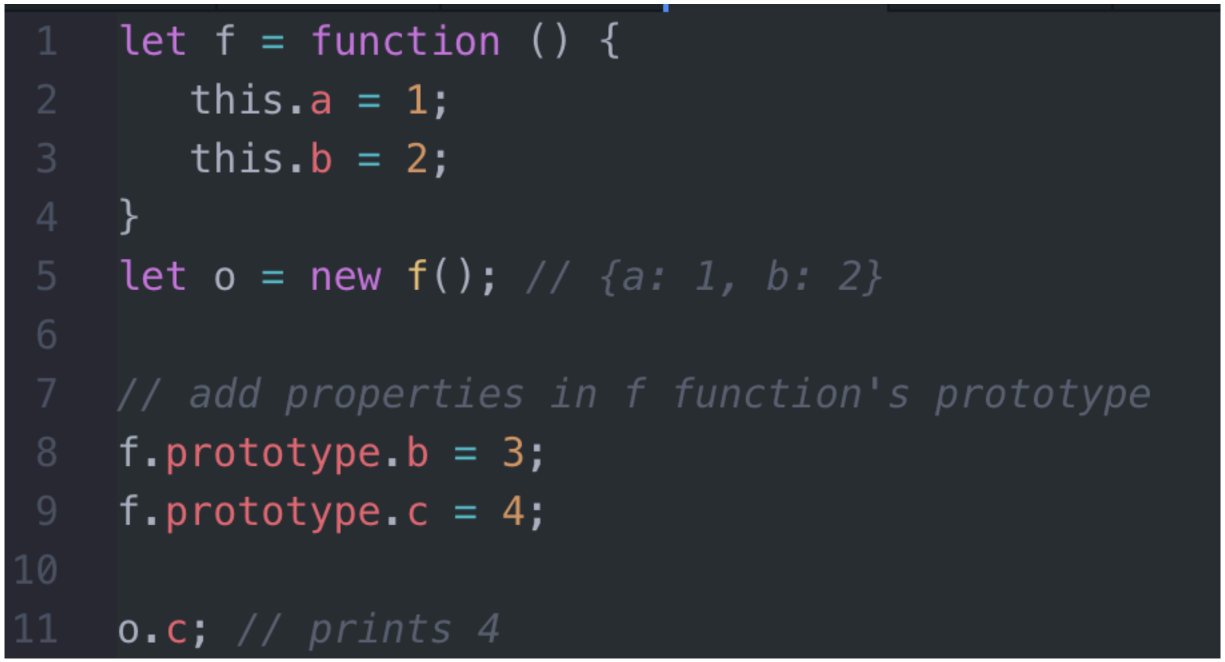
- Runtime evaluation
  - eval



```
var blockOfCode = "i=0";
var i=10;
eval(blockOfCode);
console.log(i);
```

JS language

- Prototype (Object Oriented)



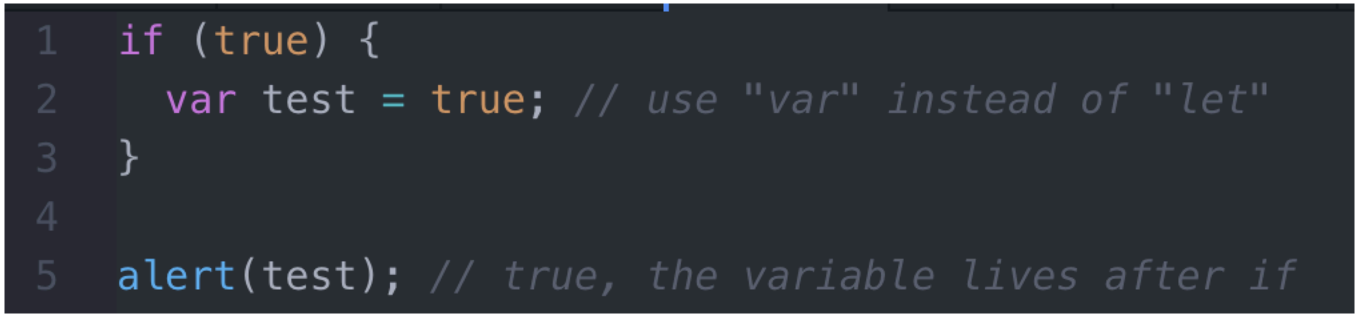
```
let f = function () {
   this.a = 1;
   this.b = 2;
}
let o = new f(); // {a: 1, b: 2}

// add properties in f function's prototype
f.prototype.b = 3;
f.prototype.c = 4;

o.c; // prints 4
```

JS language

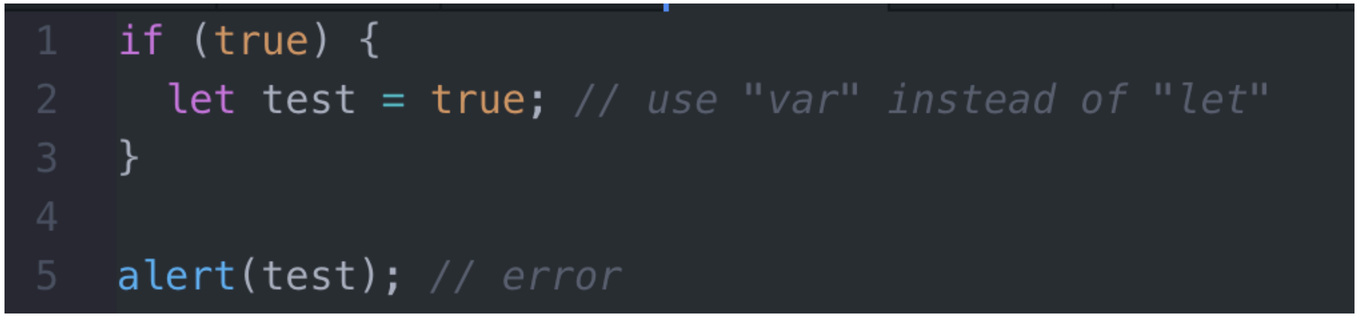
- Prototype (Object Oriented)
  - Arrays and Sets are just objects with premade prototypes

JS language

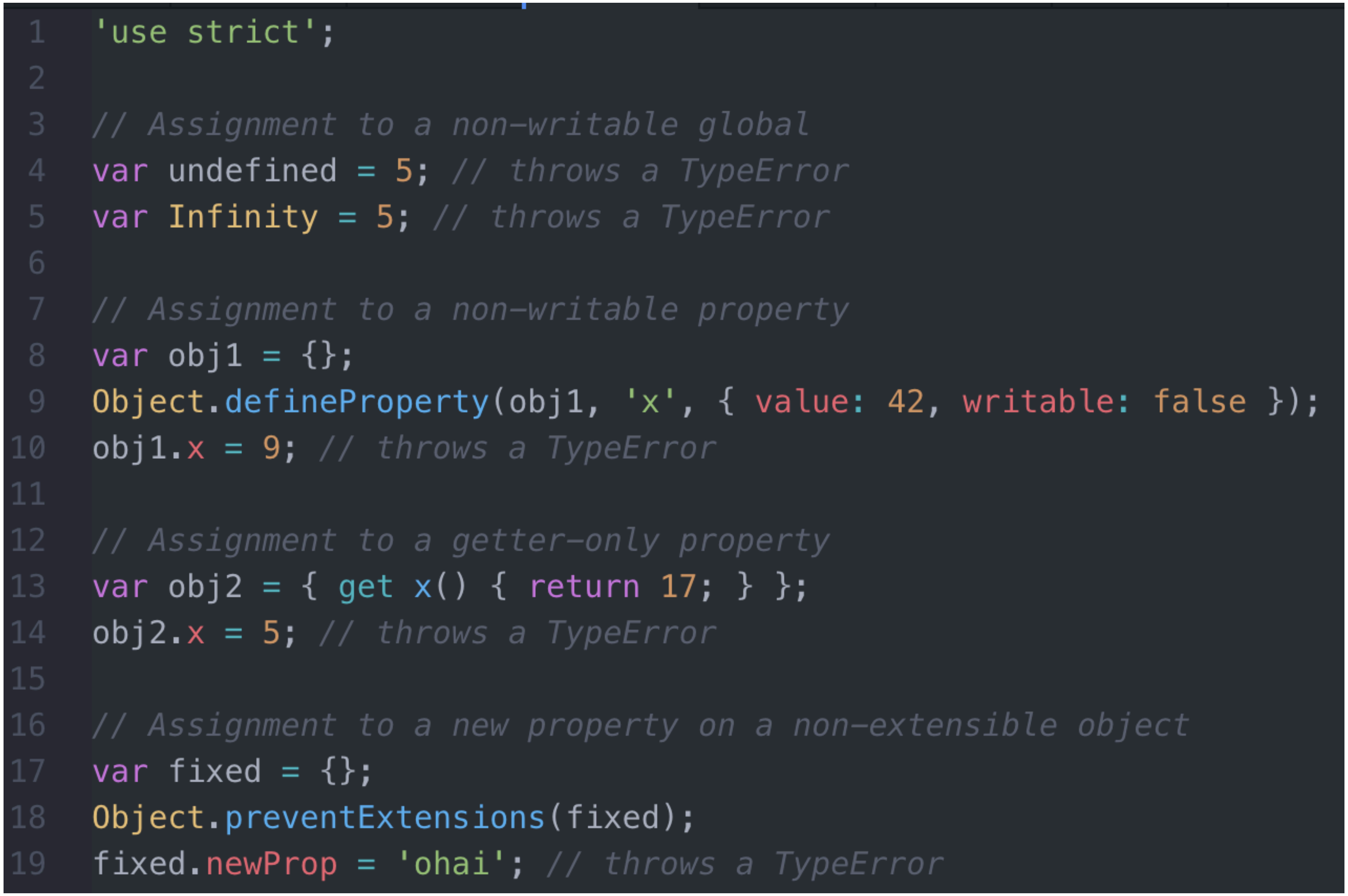
- modern js
  - var vs let
    - scope



```
if (true) {
  var test = true; // use "var" instead of "let"
}

alert(test); // true, the variable lives after if
```



```
if (true) {
  let test = true; // use "var" instead of "let"
}

alert(test); // error
```



```javascript
'use strict';

// Assignment to a non-writable global
var undefined = 5; // throws a TypeError
var Infinity = 5; // throws a TypeError

// Assignment to a non-writable property
var obj1 = {};
Object.defineProperty(obj1, 'x', { value: 42, writable: false });
obj1.x = 9; // throws a TypeError

// Assignment to a getter-only property
var obj2 = { get x() { return 17; } };
obj2.x = 5; // throws a TypeError

// Assignment to a new property on a non-extensible object
var fixed = {};
Object.preventExtensions(fixed);
fixed.newProp = 'ohai'; // throws a TypeError
```

see https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode

DOM

- Document Object Model
- programming interface for HTML (and XML)

JS+DOM

- javascript can:
  - add, remove, modify all HTML of a page
  - add, remove events
  - modify all CSS

DOM

- retrieve nodes from the document
  - document.getElementById(id)
  - document.getElementsByTagName(tagname)
  - …

DOM

- create/change nodes
  - document.createElement(tagname)
  - then node.appendChild(newnode)
  - node.innerHTML
  - node.innerText
  - node.childNodes
  - ...

DOM

- change behaviour
  - node.onclick = function(e) {}
  - node.onmouseenter / onmouseleave
  - node.onmousemove
  - onscroll
  - onresize
  - onfocus
  - …

CSS

- TODO

Canvas

- HTML5 element
- provides a 2D graphics context, or access to WebGL (close to OpenGL ES 2.0)
- see http://curran.github.io/HTML5Examples/