Introduction to JavaScript

Web Dev, Spring 2021

The web triumvirate

Last week —

- HTML: the skeleton of web documents
- CSS : the skin of web documents

This week —

- JavaScript: the muscles (?) of web documents

History and context

JavaScript was developed back in the days of the first web browsers

- Netscape Navigator (ancestor of Firefox) ~ 1995

It is a pretty standard dynamically typed imperative programming language

- think Python, Ruby, Scheme, Lua

JavaScript is an interesting beast

- Execution engine is web browsers [no IO, networking, graphics]
- Development has been seat-of-the-pants until ~ 2009 (ECMAScript 5)
- Strong incentives to maintain backward compatibility

JavaScript and ECMAScript

ECMAScript is the specification — JavaScript is one implementation of the spec

The spec is updated every year, adding features to the language

- think Python 3.6, 3.7, 3.8, 3.9, and soon 3.10

ES6 = ES2015
ES7 = ES2016
ES8 = ES2017
...

JavaScript and ECMAScript

ECMAScript is the specification — JavaScript is one implementation of the spec

The spec is updated every year, adding features to the language

- think Python 3.6, 3.7, 3

ES6 = ES2015
ES7 = ES2016
ES8 = ES2017
...

Up-to-date browsers mostly implement the latest specification

Problem is that you don't control whether your clients have up-to-date browsers

How to cover a language in one lecture

1. Assume students know another language of the same type
2. Cover the basic data types
3. Cover the basic statements
4. Cover functions
5. Cover classes
6. Cover gotchas

Basic data types

Floating point numbers	`0  1  3.14159  -2.71  ...`
Strings	`""  "I'm a string"  'also a string'  ...`
Booleans	`true  false`
Regular expressions	`/a.*b/  ...`
Arrays	`[]  [10, 'hello', 20]`
Objects	`{}  {a: 10, b: 20 }`

...

Basic statements

Declaration

```
var x      var x = 10       var x = [1, 2, 3]
let x      let x = 10       let x = [1, 2, 3]
const x = 10          const x = [1, 2, 3]
```

Assignment

```
x = 1      x = f(y) * 2
```

Conditional

```
if (x > 0) {
    ...
} else {
    ...
}
```

Basic statements

Declaration

```
var x
let x
const x = 1
```

Assignment

```
x = 1
```

Conditional

```
if (x > 
    ...
} else {
    ...
}
```

Technically all statements end with semicolons

```
x = 1;
```

The parser does not require it —
it will insert them for you automatically

Does it right 99.9% percent of the time

We'll drop semicolons — they're visual noise

Basic statements

Loops

```
while (i > 0) {
    ...
}


for (i = 0; i < j; i += 1) {
    ...
}


for (x of xs) {
    ...
}
```

Arrays

Pretty much like Python lists

```
let x = [1, 2, 3, 4]
let z = x[0] + x[2]
x[0] = 99
```

Arrays have methods:

```
let len = x.length
let arr1 = x.concat([10, 11, 12])
x.push(99)
```

Objects

Play two roles:

- Python-like dictionaries
- Objects in the OO sense

```
let obj = {x: 10, y: 20, test: {a: 1, b: [2, 3, 4]}

let z = obj.x + obj.y

obj.x = 99

obj['x'] = 98

obj[field] = 97
```

Functions

Completely standard

```
function square (a) {
   return a * a
}
```

Anonymous functions:

```
let square = function(a) { return a * a }

let square = (a) => { return a * a }

let square = (a) => (a * a)
```

Functions

Iterate over a list and square all its elements:

```
function squareAll (ls) {
   var result = []
   var i
   for (i = 0; i < ls.length; i += 1) {
      result.push(ls[i] * ls[i])
   }
   return result
}
```

Functions

Iterate over a list and square all its elements:

```
function squareAll (ls) {
   let result = []
   for (let x of ls) {
      result.push(x * x)
   }
   return result
}
```

Functions

Iterate over a list and square all its elements:

```
function squareAll (ls) {
   return ls.map((x) => x * x)
}
```

Functions are first-class values

```
function isGreaterThan (num) {
    let test = function(i) {
      return i > num
    }
    return test
}

let isPositive = isGreaterThan(0)


if (isPositive(10)) {
   …
}
```

Functions are first-class values

```
function isGreaterThan (num) {
    let test = function(i) {
      return i > num
    }
    return test
}

let isPositive = isGreate

if (isPositive(10)) {
   …
}
```

Functions form **closures**

Free identifiers in the function refer to identifiers existing **where the function is defined**

Again, pretty standard: Python, Scheme, Java

Classes (ES6)

```
class Counter {
   constructor (init) {
      this.count = init
   }

   value () {
      return this.count
   }

   inc (i) {
      this.count += i
   }
}
```

```
let counter = new Counter(10)

counter.inc(1)
counter.inc(3)
counter.inc(5)


let result = counter.value()


// result == 19
```

Subclasses (ES6)

```
class MCounter extends Counter {
   constructor (init) {
      super(init)
   }

   mult (m) {
     let v = this.value()
     this.inc(v * (m - 1))
   }
}
```

```
let counter2 = new MCounter(10)

counter2.inc(1)
counter2.mult(2)
counter2.inc(3)


let result2 = counter2.value()


// result2 == 25
```

Gotchas

Many equalities:

- `a == b` `a === b` `Object.is(a, b)`

Declarations:

- `var x = 10` `let x = 10` `const x = 10`

Null values:

- `undefined` `null`

Truthiness — expressions evaluate to `true` or `false` in a Boolean context (e.g., `if` condition)

- all values are `true` except: `0` `""` `undefined` `null`

More to JavaScript...

Prototype-based object system

Module system(s)

Promises

Async functions

Iterators

...