Imperial College London

CO331 – Network and Web Security

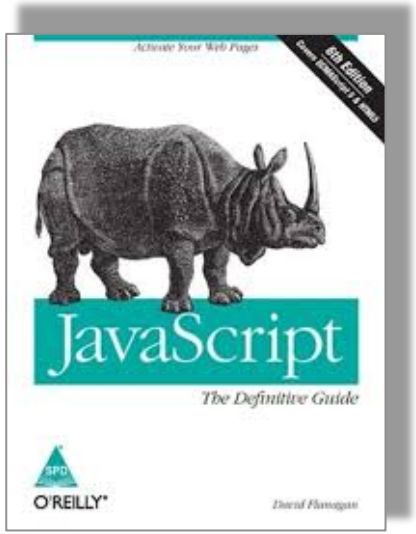
14. JavaScript

Dr Sergio Maffeis
Department of Computing
Course web page: http://www.doc.ic.ac.uk/~maffeis/331

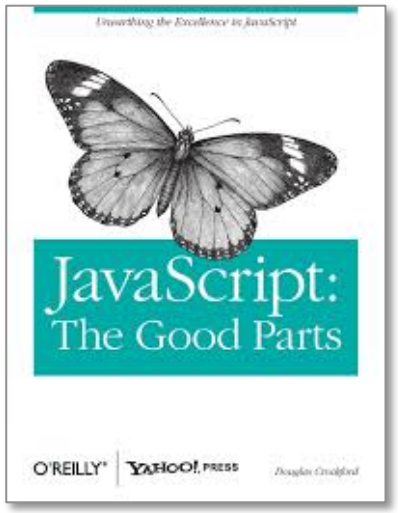
JavaScript

Imperial College London

- 1995: a small language to validate web form inputs in the browser (Brendan Eich)
- 2018: “Language of the web”, and more
  - All major browsers
  - On the server: Node.js
  - Smartphones: React Native
  - Desktop apps: Electron
- Powerful and dangerous
  - Easy to make mistakes: many practical examples of injection and XSS are on JavaScript
  - Most browser-based malware is JavaScript code, or at least installed by it
- **Goal**
  - **Understand how a web page works, and analyse its vulnerabilities**
- Non-goal
  - Become a proficient JavaScript programmer



1,096 pages

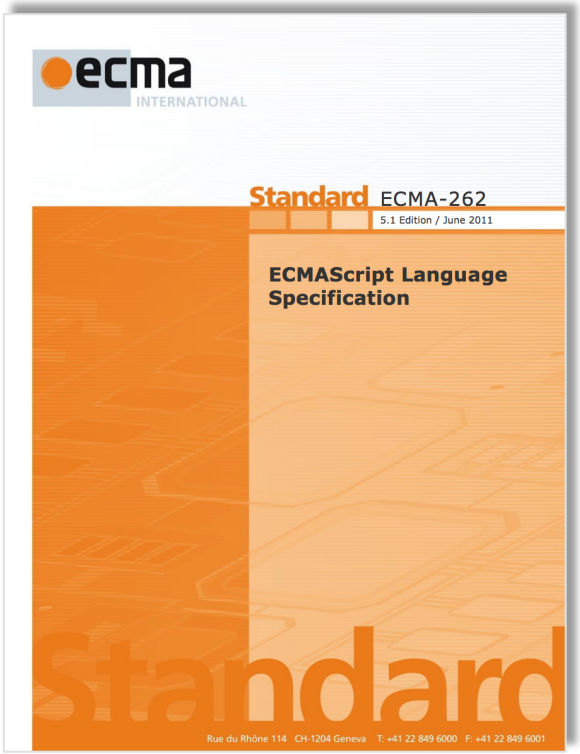


172 pages

ECMAScript

Imperial College London

- **ECMAScript** is the standardised core of JavaScript
  - After initial divergence (Microsoft JScript, etc.) and "browser wars", big players agreed on core language
  - ECMA-262 standard document
    - Clear and concise English prose and pseudo-code
    - ECMA3 (1999), ECMA5 2011, ECMA6 2015
  - Key players
    - Yahoo!, Google, Microsoft, IBM, Adobe
    - Imperial is involved in standard body as NFP member
- Main current implementations
  - Fully support ECMA5.1, and many features of ECMA 6
  - V8: Chrome, Opera, Node.js
  - Spidermonkey: Firefox, Adobe Acrobat
  - JavaScriptCore: Webkit, Safari (as *Nitro*)
  - Chakra: Microsoft Edge (IE replacement)
- Other implementations
  - JScript: Internet Explorer (only ECMA5.1)
  - Rhino: Java (ECMA3+)



JavaScript formal semantics

Imperial College London

- Together with ML, C, Java and PHP, JavaScript is one of the few programming languages to have a formal operational semantics
  - Covering all the language core
  - Allowing to state properties of programs, and prove them
  - A basis for program analysis tools with correctness guarantee
- *An Operational Semantics for JavaScript*
  - Maffeis, *et al*: APLAS 2008
  - Operational semantics: http://jssec.net/semantics/
  - Enables hand-proofs of security properties
- *The Essence of JavaScript*
  - Guha, *et al*: ECOOP 2010
  - Translation into Scheme-like language
  - First type systems for JavaScript
- *A trusted mechanised JavaScript specification*
  - Gardner, Maffeis, *et al*: POPL 2014
  - Formalisation in Coq proof assistant: http://jscert.org/
  - Aims to build *certification infrastructure* for program properties

JavaScript features

Imperial College London

- Objects as mutable records of functions with implicit `this`:

```
o = {b:function(){return this.a}};
```

- Prototype-based object inheritance:

```
Object.prototype.a = "foo";
```

- Implicit type conversions, that can be redefined.

```
Object.prototype.toString = o.b;
```

- Can convert strings into code:

```
eval("o + o['b']()"); // returns "foofoo"
```

Variables and scope

Imperial College London

- The scope can be manipulated like a language object:

```
window.o === o; // global scope
var s = {x:41};
with (s) {s.x++; console.log(x);} // local scope
```

- Nested scoping of functions (does not happen in PHP!)

```
x=1; y=2;
function a(z){var y=z+x;
              function b(w){return w+x+y}
              return b(z)}
a(20) // returns 42
```

- Can encapsulate scope via function closures

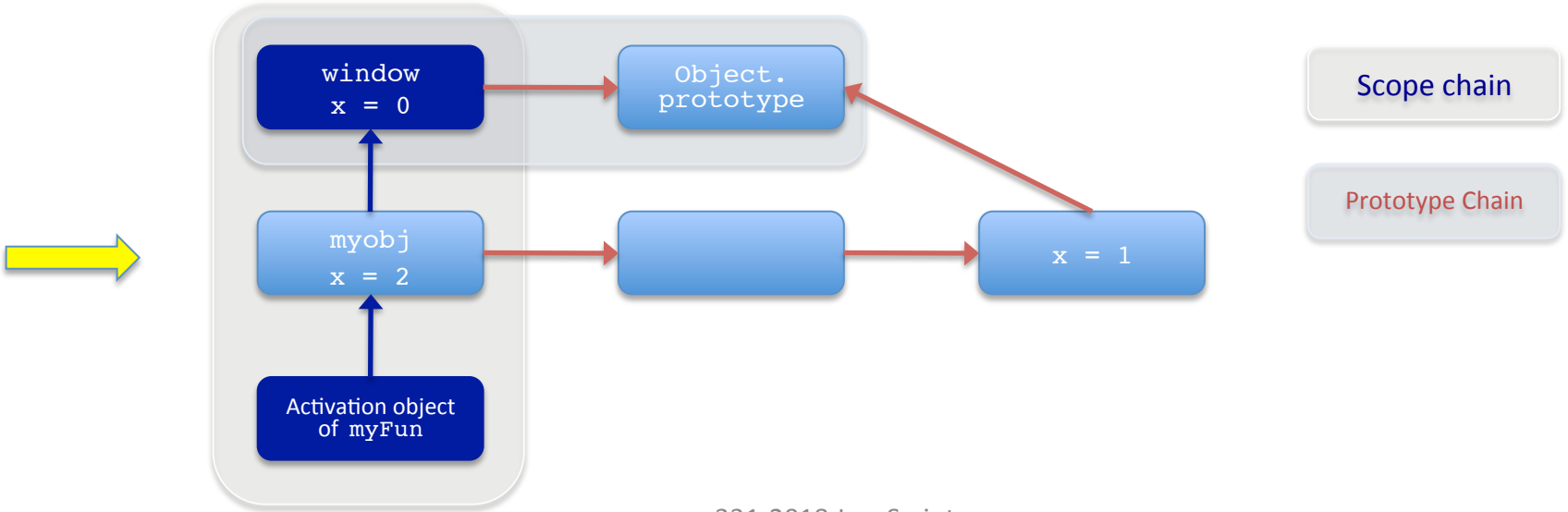
```
var API = (function(){
     var x=13;
     return [function(y){return x+y;},
             function(z){x=z; return x;}]}
          )();
API[1](API[0](29)); //returns 42
```

Scopes and prototypes

Imperial College London

- Variable `x` is resolved as property `x` of the current scope object.
  - If `x` is not present, look in the parent scope object.
- Expression `myObj.x` evaluates to the property `x` of object `myObj`.
  - If `x` is not present, look in the prototype of `myObj`.
- Example:

```
with(myObj){myFun = function(){return x+=1};};
myFun();
```



JavaScript compilation

Imperial College London

- JavaScript compilers
  - In principle JavaScript is an interpreted language
  - Main engines use bytecode and *just-in-time* (JIT) compilation to machine code
  - Optimising compilers: IonMonkey (Mozilla), Crankshaft (Google)
- asm.js
  - Fast subset of JavaScript, close to machine code
    - No nested functions, no objects
    - Main data structure are typed-arrays
    - All values are Int, Double, Float (signed/unsigned)
    - Roll-your-won memory management!
- WebAssembly (wasm)
  - Portable size- and load-time-efficient binary format suitable for compilation to the web
  - Aims for native speed
  - No longer JavaScript: think C/C++ for the web, interoperable with JavaScript
- Emscripten
  - Compiles any LLVM bitcode to asm.js, wasm

```
var log = stdlib.Math.log;
var values = new stdlib.Float64Ar

function logSum(start, end) {
  start = start|0;
  end = end|0;

  var sum = 0.0, p = 0, q = 0;

  // asm.js forces byte addressin
  for (p = start << 3, q = end <<
    sum = sum + +log(values[p>>3]
  }

  return +sum;
}
```

Frameworks and types

Imperial College London

- JavaScript frameworks
  - jQuery, Angular, EmberJs, Mocha, React…
  - Wrap DOM and other commos interfaces (AJAX)
  - Provide convenient syntactic sugar and programming patterns
  - Facilitate unit-testing, portability
- TypeScript
  - Statically typed, class-based superset of JavaScript
    - Best effort typing, no general soundness guarantee
  - Compiled down to JavaScript, hence fully compatible
  - Originated by Microsoft
- Flow
  - Facebook's answer to TypeScript
  - Static type checking and type inference for JavaScript

```
class Student {
    fullName: string;
    constructor(public firstName: string,
string) {
        this.fullName = firstName + " " +
    }
}

interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person : Person) {
    return "Hello, " + person.firstName +
}

let user = new Student("Jane", "M.", "User

document.body.innerHTML = greeter(user);
```