JavaScript

The Basics

Assert.that(WeUnderstand(C))

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Agenda

- Introduction
- Values, Variables and Types
- Control flow

Why?

- JavaScript is the programming language of the Web.
  - All web browsers have a JavaScript interpreter that can execute JavaScript

- JavaScript is used to:
  - Create dynamic Web pages
  - Validate user input in the browser
  - Ajax (send data to, and retrieve data from, the server asynchronously)
  - Create Web applications

Characteristics

- JavaScript is a multi-paradigm language, supporting:
  - **Imperative**,
  - **Object-oriented**, and
  - **Functional** programming styles
- JavaScript is a **dynamic** language
- JavaScript is **weakly typed** and supports duck typing
  - types are associated with values, not with variables
- The key design principles within JavaScript are taken from:
  - **Self** (prototype-based approach to objects)
  - **Scheme** (functional programming)
  - **C** (imperative programming and syntax)
  - **Java** (names and naming conventions)
  - Perl (regular expressions)

History

- JavaScript was originally developed in Netscape, by Brendan Eich
  - *stayed home for two weeks to rewrite Mocha as the codebase that became known as SpiderMonkey - the firste JavaScript engine*
- Netscape wanted a lightweight interpreted language that would complement Java by appealing to nonprofessional programmers, like Microsoft's VB
- First released with Netscape Navigator 2.0 in 1995
- Microsoft introduced JavaScript (JScript) support in Internet Explorer 3.0 in 1996
- Netscape submitted JavaScript to Ecma International for standardization in 1996
  - And the official name is now ECMAScript

Versions

Year	JavaScript	JScript	ECMAScript
1995	1.0		
1996	1.1	1.0	
1997	1.2	2.0	1.0
1998	1.3	3.0	2.0
1999	1.4	4.0 5.0	
2000	1.5	5.5	3.0
2005	1.6		
2006	1.7		
2008	1.8		
2009			**5.0**
2011			5.1
2015	2.0		6.0 (aka ES2015)
2016			ES2016
Work in progress			ES2017

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Conformance

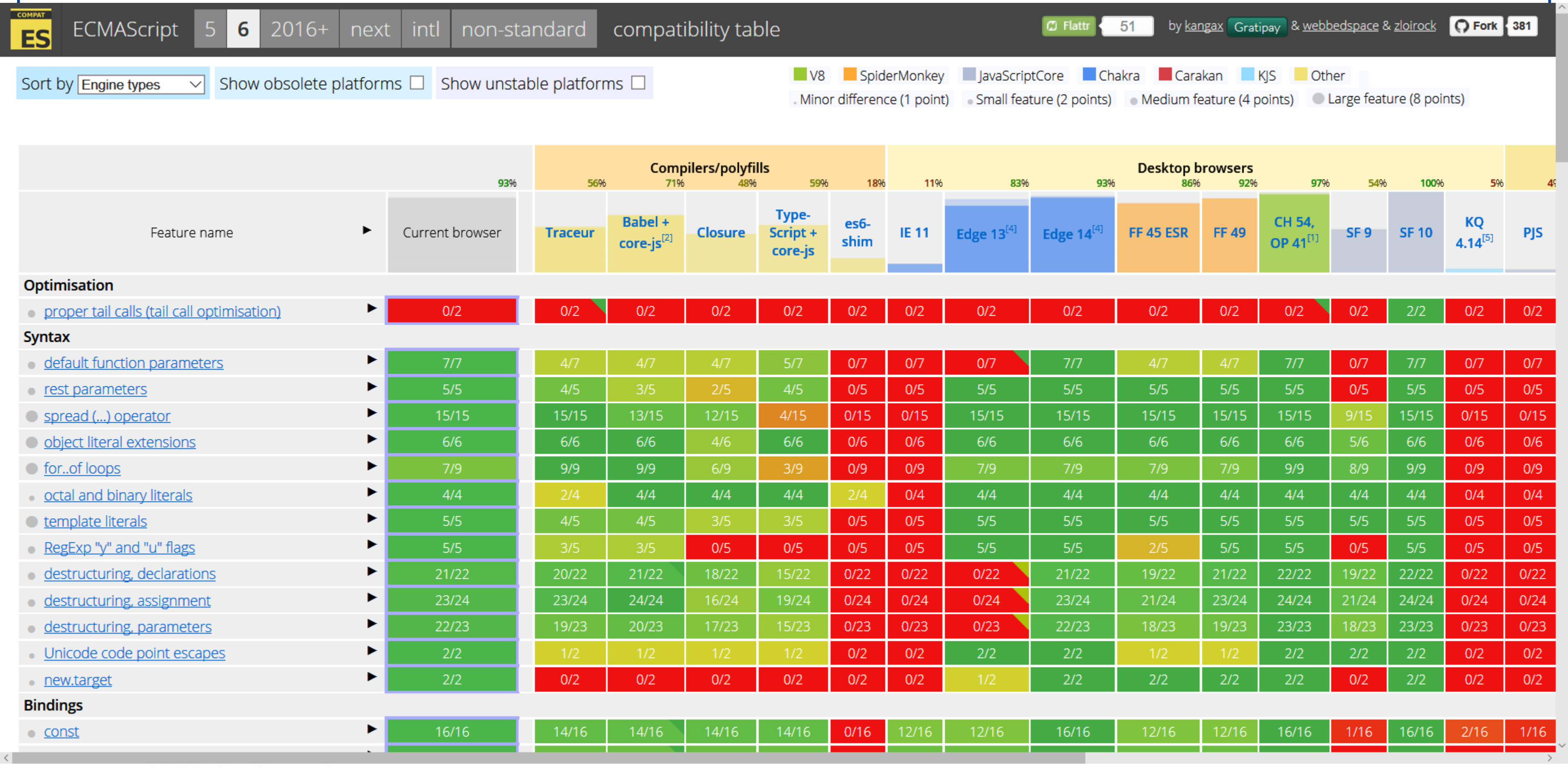
- Test262 is an ECMAScript conformance test suite that can be used to check how closely a JavaScript implementation follows the **ECMAScript 5th** Edition Specification.
  - Test262 testsuite contains more than 11,000 tests.

Browser	Version	Tests failed
Safari	7.1 (9537.85)	7
Opera	25.0.1614.68	8
Chrome	38.0.2125.122 m	8
Firefox	33.1	53
Internet Explorer	11.0.14	8

http://en.wikipedia.org/wiki/ECMAScript (2014-09-18)

EcmaScript 6

- A sixth edition of the standard was published 17th, June 2015
  - But it will take a while to reach full support in all major browsers
  - ECMAScript 6 compatibility table http://kangax.github.io/compat-table/es6/



ECMAScript | 5 | 6 | 2016+ | next | intl | non-standard | compatibility table

Flattr 51 by kangax Gratipay & webbedspace & zloirock Fork 381

Sort by Engine types | Show obsolete platforms ☐ | Show unstable platforms ☐

🟩 V8 🟧 SpiderMonkey ▨ JavaScriptCore 🟦 Chakra 🟥 Carakan ▨ KJS 🟨 Other

● Minor difference (1 point) ● Small feature (2 points) ● Medium feature (4 points) ● Large feature (8 points)

		Compilers/polyfills					Desktop browsers									
	93%	56%	71%	48%	59%	18%	11%	83%	93%	86%	92%	97%	54%	100%	5%	4
Feature name	Current browser	Traceur	Babel + core-js[2]	Closure	Type-Script + core-js	es6-shim	IE 11	Edge 13[4]	Edge 14[4]	FF 45 ESR	FF 49	CH 54, OP 41[1]	SF 9	SF 10	KQ 4.14[5]	PJS
Optimisation																
proper tail calls (tail call optimisation)	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	0/2	0/2
Syntax																
default function parameters	7/7	4/7	4/7	4/7	5/7	0/7	0/7	0/7	7/7	4/7	4/7	7/7	0/7	7/7	0/7	0/7
rest parameters	5/5	4/5	3/5	2/5	4/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	0/5	5/5	0/5	0/5
spread (...) operator	15/15	15/15	13/15	12/15	4/15	0/15	0/15	15/15	15/15	15/15	15/15	15/15	9/15	15/15	0/15	0/15
object literal extensions	6/6	6/6	6/6	4/6	6/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	5/6	6/6	0/6	0/6
for..of loops	7/9	9/9	9/9	6/9	3/9	0/9	0/9	7/9	7/9	7/9	7/9	9/9	8/9	9/9	0/9	0/9
octal and binary literals	4/4	2/4	4/4	4/4	4/4	2/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	0/4
template literals	5/5	4/5	4/5	3/5	3/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	0/5
RegExp "y" and "u" flags	5/5	3/5	3/5	0/5	0/5	0/5	0/5	5/5	5/5	2/5	5/5	5/5	0/5	5/5	0/5	0/5
destructuring, declarations	21/22	20/22	21/22	18/22	15/22	0/22	0/22	0/22	21/22	19/22	21/22	22/22	19/22	22/22	0/22	0/22
destructuring, assignment	23/24	23/24	24/24	16/24	19/24	0/24	0/24	0/24	23/24	21/24	23/24	24/24	21/24	24/24	0/24	0/24
destructuring, parameters	22/23	19/23	20/23	17/23	15/23	0/23	0/23	0/23	22/23	18/23	19/23	23/23	18/23	23/23	0/23	0/23
Unicode code point escapes	2/2	1/2	1/2	1/2	1/2	0/2	0/2	2/2	2/2	1/2	1/2	2/2	2/2	2/2	0/2	0/2
new.target	2/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	2/2	2/2	2/2	2/2	0/2	2/2	0/2	0/2
Bindings																
const	16/16	14/16	14/16	14/16	14/16	0/16	12/16	12/16	16/16	12/16	12/16	16/16	1/16	16/16	2/16	1/16

Engine Names

- All companies name their JavaScript engine

Company	Browser	HTML Layout Engine	JavaScript Engine
Apple	Safari	WebKit	SquirrelFish Extreme (based on KJS)
Opera Software	Opera	Was Presto Now Blink	Was Carakan Now V8
Google	Chrome	Was WebKit Now Blink	V8
Mozilla Foundation	Firefox	Gecko	SpiderMonkey (Rhino in Java)
Microsoft	Internet Explorer/ Edge	Trident	Chakra

VALUES, VARIABLES AND TYPES

Values

- In JavaScript data is separated into things called values
- Some possible values

```
27
3.1415927
"Hello World"
true
```

- Every value has a type

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Types

- There are six types of values:
  - number
  - string
  - boolean
  - object
  - function
  - Undefined

number, string, boolean: Are called primitive types. Have value type semantics

object, function: Have reference type semantics

Undefined: *(not really a type – it is a special word like null)*

No char
And only one number type

- The typeof operator produces a string value naming the type of the value you give it.

```
typeof 4.5
```

(But every type in JavaScript is implemented as a variation of the type object)

number

IEEE-754
Double

- Numbers in JavaScript has 64 bits
  - 144 is stored as 0100000001100010000000000000000000000000000000000000000000000000
- 3.1415927 is also of type number!
- 1 bit is used for the sign and 11 are used to store the position of the fractional dot within the number. That leaves 52 bits
- Any whole number less than 2^{52} (which is more than 10^{15}) will safely fit in a JavaScript number
- Calculations with whole numbers (integers) that fit in 52 bits are guaranteed to always be precise
  - calculations with fractional numbers are generally not precise (but usually good enough)
  - alert(0.94 - 0.01); // displays 0.9299999999999999
  - Use toFixed() method to round numbers whenever they are formatted for output
  - alert((0.94 - 0.01).toFixed(2)); // displays 0.93
- JavaScript has the normal arithmetic operations, e.g.

```
115 * 4 - 4 + 88 / 2
```

string

- Strings are written by enclosing their characters in quotes:

```
"Patch my boat with chewing gum."
```

- And ‘\’ is used as an escape character like we are used to
- The + operator can be used to glue two strings together

```
"con" + "cat" + "e" + "nate"
```

- Strings are made of unicode characters
- Strings are immutable

boolean

- Has the usual two values: `true` and `false`

- 3 > 2
  will produce the `true` value

Expressions

- A piece of code that produces a value is called an expression
- Every value that is written directly is an expression

```
"Patch my boat with chewing gum."
42
2.998e8
2 * 10 + 7
33 % 5
typeof(2.4)
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Statements

- A program is built as a list of statements
- Most statements end with a semicolon ";"
- The simplest kind of statement is an expression with a semicolon after it:

```
42;
2 * 10 + 7;
!false;
```

- In some cases, JavaScript allows you to omit the semicolon at the end of a statement
  - In other cases, it has to be there or strange things will happen
- The rules for when it can be safely omitted are complex and weird
  - So don't leave out any semicolons!
    (*actally there are only 2 cases where a semicolon is needed, so some programmers don't use semicolons at all*)

Variables

- The word **`var`** or **`let`** is used to create a new variable
- After var, the name of the variable follows

```
var caught = 5 * 5;
var name = "John";
```

- The variable itself is typeless, and it is capable of holding any type of value (the value has a type - not the variable)
- Variable names can be almost every word, but they may not include spaces
- Digits can be part of variable names, catch22 is a valid name, but the name must not start with a digit
- The characters '$' and '_' can be used in names as if they were letters, so $_$ is a valid variable name
- A variable name can be used as an expression

Variables are References to a Value

```
var myRef;
console.log(typeof myRef);  // prints undefined
myRef = 22;
console.log(typeof myRef);  // prints number
myRef = "Hello";
console.log(typeof myRef);  // prints string
myRef = 5 > 3;
console.log(typeof myRef);  // prints boolean
```

typeof gives you the type of the value that the variable holds on to at the moment…

Garbage Collection

- JavaScript has garbage collection
  - All storage occupied by values and variables are automatically reclaimed when they are no longer being referenced
  - How this works (the GC algorithm) is not a part of the standard, but depends on the implementation in the JavaScript engine
- Chakra uses a concurrent, generational, mark and sweep algorithm

Operators

- JavaScript has the usual operators, e.g.

Binary arithmetic operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Comparison:

==	Equal
!=	Not equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
===	Identical (equal and of the same type)
!==	Not identical

- And they generally work as expected
  - But sometime automatic type coercion is the root to some surprisingly behaviour
  - And “||” and ”&&” work different than in a typical C derived language – except when used on boolean values

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Comparison Rules

- The number 0, NaN, the empty string "", null, undefined, and of course false, will all count as false
- `null == undefined` will produce true
- `NaN == NaN` equals false! (use the function `isNaN()`)
- When comparing values that have different types, JavaScript tries to convert one of the values to the type of the other value
  - but when null or undefined occur, it only produces true if both sides are null or undefined

They all prints true:

```
log(null == undefined);
log(false == 0);
log("" == 0);
log("5" == 5);
```

They all prints false:

```
log(null === undefined);
log(false === 0);
log("" === 0);
log("5" === 5);
```

Operator ||

On booleans it works as we are used to:

```
var a = false;
var b = true;
console.log(a || b); // normal boolean OR -> prints true
```

```
var a = null;
var b = "b";
console.log(a || b); // if a is true, return a, otherwise return b
```

Typical use of || operator:

```
var input = prompt("What is your name?");
alert("Well hello " + (input || "dear"));
```

Environment

- The collection of variables and their values that exist at a given time is called the environment
- When a program starts up, this environment is not empty
  - It always contains a number of standard variables
- When your browser loads a page, it creates a new environment and attaches these standard values to it
- **The variables created and modified by programs on that page survive until the browser goes to a new page**

- It is possible to give almost every variable in the environment a new value
  - This can be useful, but also **dangerous!**

The Type 'function'

- A function is a piece of program wrapped in a value
- In a browser environment, the variable alert holds a function that shows a little dialog window with a message.
- It is used like this:

```
alert("Your hair is on fire!");
```

- Every expression that produces a function value can be invoked by putting parentheses after it

```
prompt("Tell us everything you know.", "...");
```

Converting string to number

- The function Number converts a value to a number

```
var theNumber = Number(prompt("Pick a number", ""));
console.log("Your number is the square root of " +
      (theNumber * theNumber));
```

- There are similar functions called `String` and `Boolean` which convert values to those types

The Type object

- Objects are entities that have an identity (they are only equal to themselves) and that map property names to values
- Objects is a collection of (key, value) pairs (a dictionary) where the key is of type string and the value can be any type
- Objects have a prototype chain

How to Create Objects

- You can create objects in two different ways:
  1. Object literal

```
var myObject = {member1: 'value 1', 'my number': 27};
console.log(myObject.member1);
console.log(myObject['my number']);

var b = {};
b.x = 42;
console.log(b.x);
```

Constructor function

  2. Constructor

```
var anObject = new Object();
```

Reserved Words

- Keywords like **var** and **while** and a number of words which are 'reserved for future use' are called reserved words
- A reserved word cannot be used as:
  - As a name in literal object notation
  - As a member name in dot notation
  - As a function argument
  - As a var
  - As an global variable
  - As a statement label

abstract
boolean **break** byte
case **catch** char class const **continue**
debugger **default** **delete** **do** double
else enum export extends
false final **finally** float **for** **function**
goto
if implements import **in** **instanceof** int interface
long
native **new** **null**
package private protected public
return
short static super **switch** synchronized
this **throw** throws transient **true** **try** **typeof**
var volatile **void**
while **with**

CONTROL FLOW

if

- Has the usual syntax and behaviour

```
var day = 1;
if (day > 7)
  day = 1;
```

```
var day = 1;
if (day > 7)
  day = 1;
else
  day++;
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Conditional Operator ?

- Is similar to the if statement

```
result = condition ? expression : alternative;
```

```
var a = 6;
var x = 5;
var res = (x > 7) ? a : 5 * a;
console.log(res);
```

switch

- Has the usual syntax and behaviour
  - Except you can also switch on strings

```
switch (day) {
   case 1:
     console.log ("Monday");
     break;
   case 2:
     console.log ("Thusday");
     break;
   default:
     console.log ("Wow");
     break;
 }
```

while

- Has the usual syntax and behaviour
- Curly braces({ and }) are used to group statements into blocks

```
var currentNumber = 0;
while (currentNumber <= 12) {
  console.log(currentNumber);
  currentNumber = currentNumber + 2;
}
```

```
var txt = "false";
var i = 0;
while (txt) { //any non empty string evaluates to true
  console.log(i);
  i += 1;
  if (i > 5)
    txt = "";
  }
```

do while

- Has the usual syntax and behaviour

```
var i = 1;
do {
  console.log(i);
  i += 2;
} while (i <= 7)
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

for

- Has the usual syntax and behaviour

```
for (var i = 0; i <= 5; i++)
  console.log(i);
```

```
for (var i = 0; i <= 10; i = i + 2) {
  console.log(i);
  }
```

```
for (var counter = 0; counter < 20; counter++) {
  if (counter % 4 == 0)
    console.log(counter);
  else
    console.log("(" + counter + ")");
}
```

for in

- Iterates through all enumerable properties of an object

```
for (var property_name in some_object) {
   //statements using some_object[property_name];
 }
```

```
var myObject = {member1: 'value 1', 'my number': 27};

for (var prop in myObject) {
   console.log(myObject[prop]);
}
```

```
console.log("+++++++++"); // window ~ this ~ this.window
for (var p in window)
  console.log(p);
```

break

- Has the usual syntax and behaviour

```
for (var current = 30; ; current++) {
  if (current % 7 == 0)
    break;
}
console.log(current);
```

continue

- Jumps to the next iteration of the loop

```
for (var i = 0; i < 10; i++) {
  if (i % 3 != 0)
    continue;
  console.log (i, " is divisible by three.");
}
```

with

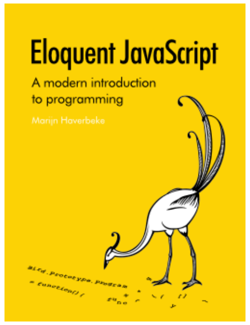
- Inside the block, the properties of the object given to with act as variables

```
var scope = "outside";
var object = {name: "Jeffrey ", scope: "inside"};
with(object) {
  print("Name == ", name, ", scope == ", scope);
  name = "Raoul";
}
print(object.name);
```

Do not use!

The with statement is considered harmful

References and Links



- **Eloquent JavaScript** by Marijn Haverbeke
  http://eloquentjavascript.net
- The JavaScript guru: **Douglas Crockford**'s blog
  http://www.crockford.com/javascript/
  http://javascript.crockford.com/survey.html ☆
- Reference
  https://developer.mozilla.org/en-US/docs/JavaScript/Reference
- ECMA-262 ECMAScript Language Specification
  http://www.ecma-international.org/memento/TC39-M.htm

- http://en.wikipedia.org/wiki/JavaScript
- http://en.wikipedia.org/wiki/JavaScript_syntax
- http://en.wikipedia.org/wiki/ECMAScript

References and Links

- **JavaScript free Books**
  - Learning JavaScript Design Patterns
    http://www.addyosmani.com/resources/essentialjsdesignpatterns/book/
  - Speaking JavaScript - An In-Depth Guide for Programmers
    http://speakingjs.com/es5/index.html
  - JavaScript Allongé
    https://leanpub.com/javascript-allonge/read
  - JavaScript Spessore
    https://leanpub.com/javascript-spessore/read

- **Idiomatic Style Manifesto**
  https://github.com/rwaldron/idiomatic.js