JavaScript

Slides prepared by Jean-Claude Dufourd and Cyril Concolato.

pdf

Overview

- History
- Bases
  - Syntax, variables, functions, expressions, loops, conditions
- Advanced programming
  - Object, Array and other globals
  - Function, arguments, call, apply, map
- ES6 :
  - classes, destructuring, arrow, Promise…

History

- Programming language created by Brendan Eich, Netscape, in 1994/1995
  - Written in a few weeks
  - At the height of the battle with Microsoft (“browsers war”, JScript)
  - First named Mocha, then LiveScript, then JavaScript,
  - Called JavaScript to quell an argument with Sun, in reference to the Java language
    - But very different from Java : do not confuse !
- Standardized under the name ECMA-262, ECMAScript . . .
  - 1st version (1997)
  - 3rd edition (1999)
  - 5th Edition - ES5 (2009)
  - 6th edition - ES6 (2015)
  - 7th edition - in progress

Language specifics

- Interpreted language (client side)
- Inspired by Scheme and Self languages (and Java for syntax)
- Very powerful, but with youthful errors worthy of a beta
- Videos : The Good Parts by Douglas Crockford

Syntax

- Syntax inspired by C, Java
  - Using {} to separate blocks of code
  - Using () for functions, if, . . .
  - Comments
    - On a line with `//`
    - On multiple lines with `/* */`
  - Case sensitivity : variable `a` is different from `A`
- Some peculiarities :
  - The use of ; after each expression is not obligatory, but strongly advised !

```
a
=
3
console.log(a)
```

equivalent to

```
a = 3;
```

Everything by example : variables

- declare a variable
  - We do not declare the type of a variable
  - The initial value defaults to the special value : `undefined`

```
var x;
```

- declare and assign a value to a variable

```
var y = 0;
```

- Values can be any type in : `boolean`, `number`, `string`, `object`, `function`, or `undefined`
  - check with : `typeof`
- The type of a variable can change over time :

```
x = 0;              typeof x; // integer         -> "number"
x = -0.01;          typeof x; // float           -> "number"
x = "hello";        typeof x; // string          -> "string"
x = 'Hello world!'; typeof x; // string          -> "string"
x = true;           typeof x; // boolean         -> "boolean"
                                                    "object"
```

Arrays

- Declare an array :

```
var primes = [2,3,5,7];
```

- Access one item :

```
primes[0];                    // → 2
primes.length;                // → 4
primes[primes.length - 1];   // → 7
primes["2"];                  // → 5
primes.2;                     // → SyntaxError: unexpected numbe
```

- An array is dynamic and each item can be anything :

```
var tableau = [];          // initially empty
tableau[0] = "test";       // item 0 is added and its value set
tableau[1] = true;         // each item may have a different typ
tableau[2] = {};
tableau[3] = null;
```

Objects

- An object is a set of ***properties***, that is to say of (name, value) pairs
- Declare an object (***object literal expression***) :

```
var book = {
    topic: "JavaScript",
    fat: true,
    "major version": 1 // space in the name: to avoid !!!
};
```

- Access a property of the object :

```
book.topic; // → "JavaScript", pointed syntax
book ["fat"]; // → true, an object is an array of properties
```

- You can assign a property at any time :

```
book.author = "John Smith"; // add the 'author' property
book.contents = {}; // add the 'contents' property
[illegible] 'title'
```

Arrays and objects

```
var empty = []; // empty array
empty.length; // → 0

var points = [ // array of objects
   {x: 0, y: 1},
   {x: 1, y: 1},
   {x: 1, y: 1, z: 2}
];

var data = { // object containing objects
   p1: {x: 0, y: 1},
   p2: {x: 1, y: 1}
};

var trials = {
   trial1: [[1, 2], [3, 4]], // array of array ~ matrix
   trial2: [[1, 2], [4, 6]]
};
```

Expressions

```
3 + 2;         // → 5
3 * 2;         // → 6
3 - 2;         // → 1
3 / 2;         // → 1.5
3 % 2;         // → 1    // modulo
"3"+"2";       // → "32"  // concatenation
"3"-"2";       // → 1  // type casting
var count = 0;
count++;
count--;
++count;
--count;
count += 2;
count -= 4;
count *= 5;
+"21";               // → 21  // converts "21" to a number 21
+"21toto";           // → NaN // impossible
parseInt("21");      // → 21  // converts "21" to a number 21
```

Boolean expressions

```
var x=3, y=2;
x == y                   // → false
x != y                   // → true
x < y                    // → false
x >= y                   // → true
"two" == "three"         // → false
"two" > "three"          // → true (lexicographic)
false == (x < y)         // → true
(x == 2) && (y == 3) // → true
(x == 2) || (y == 4) // → true
!(x == y)                // → true
```

Tests

```
if (b == 0) x = 4;

if (b == 0) { x = 4; }

if (b == 0) { x = 4; y = 2; }

if (b == 0) {
    x = 4;
    y = 2;
}
```

```
if (b == 0) {
    x = 4;
    y = 2;
} else {
    x = -4;
}
```

Beware

- == (resp. !=) tests equality (resp. difference) after conversion

```
2 == 2              // → true
"2" == 2            // → true !!!attention
"2" != 2            // → false !!!attention
```

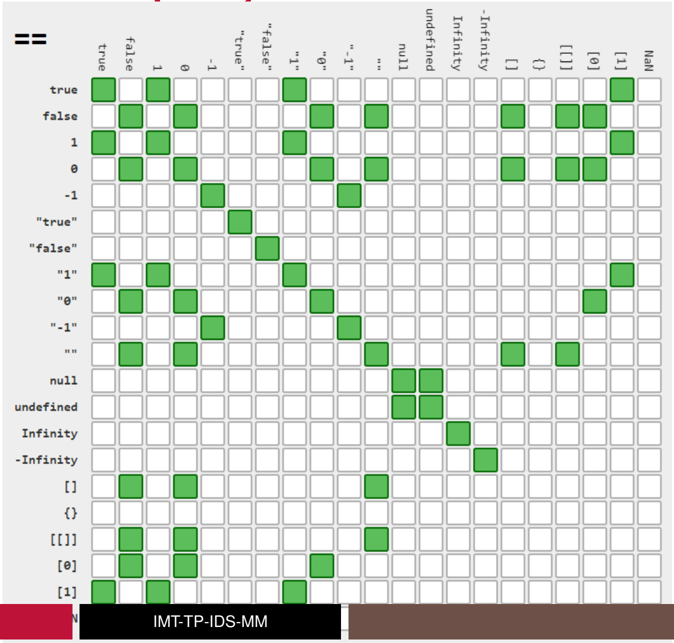
- === (resp. !==) tests equality (resp. difference) with no conversion, on original types
  - The above syntax is to be preferred

```
"2" === 2           // → false
"2" !== 2            // → true
```

- Inequality and conversions

```
"22" > "3"          // → false
+"22" > "3"         // → true
```

Equality



==	true	false	1	0	-1	"true"	"false"	"1"	"0"	"-1"	""	null	undefined	Infinity	-Infinity	[]	{}	[[]]	[0]	[1]	NaN
true	🟩		🟩					🟩												🟩	
false		🟩		🟩					🟩		🟩					🟩		🟩	🟩		
1	🟩		🟩					🟩												🟩	
0		🟩		🟩					🟩		🟩					🟩		🟩	🟩		
-1					🟩					🟩											
"true"						🟩															
"false"							🟩														
"1"	🟩		🟩					🟩												🟩	
"0"		🟩		🟩					🟩										🟩		
"-1"					🟩					🟩											
""		🟩		🟩							🟩					🟩		🟩			
null												🟩	🟩								
undefined												🟩	🟩								
Infinity														🟩							
-Infinity															🟩						
[]		🟩		🟩							🟩										
{}																					
[[]]		🟩		🟩							🟩										
[0]		🟩		🟩					🟩												
[1]	🟩		🟩					🟩													

Beware of some tests

- The following tests evaluate to `false` and thus the code does not execute

```
if (false) { ... }
if (0) { ... }
if ("") { ... }
if (null) { ... }
if (undefined) { ... }
if (NaN) { ... })
```

- All of this evaluates to `true`, which may not be obvious to you

```
if (true) { ... }
if (1) { ... }
if (-1) { ... }
if ("true") { ... }
if ("false") { ... }
```

Loops

```
while (...) {
 ...
}

do {
 ...
} while (...);

var i;
for (i=0; i<10; i++) {
 ...
}

for (var i=0; i<10; i++) {
 ...
}

// enumerate the properties of an object
```

switch

```
// strict equality test
switch(type) {
    case "a": // string
     ...
     break;
    case 1:  // number
     ...
     break;
    default:
     ...
}
```

Functions

```
function inc(x) { return x+1; }
function mul(x, y) { return x*y; }

inc(4);                 // → 5
mul(2, inc(3));         // → 8


function (x) { return x-1; };
// anonymous function, useless
// because it cannot be called

(function() { ... })();
// anonymous function called upon definition

function f() { ... }
var points = {};
points.dist = f;
points.dist();
```

Functions and formal parameters

- The arguments are separated by commas, and used in order
- Unspecified arguments are `undefined`
- Arguments can be accessed via the `arguments` array

```
function f(x,y) {
    console.log("x: "+x+", y: "+y+", z: "+arguments[2]);
}

f(1,2,3);       // x: 1, y: 2, z: 3
f(1,2);         // x: 1, y: 2, z: undefined
f(1);           // x: 1, y: undefined, z: undefined
```

Function

- A function is an instance of Object
- A function can have properties like another object
- A function can be put in a variable
- A function can be passed as a parameter of a function call
- A function can be returned by another function
- So a function is a “first class object” in the language

```
var foo = function bar(){ return 12; };
typeof bar();   -> Reference Error
```

Exceptions

```
throw new Error('This is an Error!');

try {
     . . .
} catch(e) {
     . . .
} finally {
     . . .
}
```

delete

- JavaScript uses a Garbage Collector
- We can help

```
var o = { x: 1, y: 2};
delete o.x;
"x" in o;  // → false, the property has disappeared

var a = [1,2,3];
delete a[2];
a.length;          // → 3, unchanged array length
a[2];              // → undefined as new value
```

- delete operators don't affect local variables. The delete operator doesn't delete prototype property.

Variable scope

- in Java or C, variables have a block scope
- in JS, variables have a function scope (except using the keyword ES6 `Let`)

```
function test(o) {
    var i = 0;
    if (o !== null) {
        var j = 0;
        for(var k=0; k < 10; k++) {
            console.log(k, i);
        }
        console.log(k); // variable k is still accessible
    }
    console.log(j); // variable j is still accessible
}
```

- An undeclared variable (no var statement) is global ! !

Variables and call stack

- How to determine which variable to use when multiple local variables to nested functions have the same name ?
- In the call stack, the variable used is searched :
  - in the current function
  - then in the calling function (going back)
  - then in the code outside the functions

```
var currentScope = 0; // global scope
(function () {
  var currentScope = 1, one = 'scope1';
  (function () {
    var currentScope = 2, two = 'scope2';
    (function () {
      var currentScope = 3, three = 'scope3';
      alert(currentScope); // 3
      alert(one + two + three); // scope1scope2scope3
    })();
    alert(currentScope); // 2
```

Scope of function declaration

- declaring a function is declaring a variable of type function

```
function run(obj) { ... }
run(a);
```

is equivalent to :

```
run = function (obj) { ... }
run(a);
```

- a function can be defined inside another function, and will not be accessible from outside, just like any other variable

```
function run(obj) {
    function myPrint(x) {
        console.log(x);
    }
    myPrint(obj.a);
    myPrint(obj.b);
}
```

Advanced typing

- Primitive types : contain a single value, no method, no properties
  - `boolean` : `true`, `false`
  - `number` : integers and floats (IEEE 754 watch the precision), `+Infinity`, `-Infinity`, `NaN`

```
0.1 + 0.2; // → 0.30000000000000004
```

  - `string`
  - `null`
  - `undefined`

```
1.toString(); // Uncaught SyntaxError: Unexpected token ILLEG
true.toString(); → "true"
null.toString(); // Uncaught SyntaxError: Unexpected token ILL
"toto".toString(); → "toto"
undefined.toString(); // Uncaught SyntaxError: Unexpected toke
```

Advanced typing 2

- Complex types, with predefined methods
  - `Object` and its derivatives
    - `Boolean`, `Number`, `String`, `Array`, `Math`, `Date`, `Regexp`, `Function`, `Set`, `JSON` ...

```
var b = new Boolean(true);
b.toString(); // → "true"
var n = new Number(3.14);
n.toString(); // → "3.14"
```

Watch the types

- The variables of primitive type are not objects, which can give rise to bizarre behaviors (***type coercion***)

```
var s = "hello, world";
typeof s; // → "string" primitive type, not an object
s.x = 15; // → 15 no error because equivalent (new String (s)
typeof s; // → "string" s did not change type
s.x // → undefined problem because the temporary String objec
```

- In this case, directly use a variable of type String

```
s = new String ("hello, world");
typeof s; // → object: s is an object in its own right
s.x = 15; // → 15
s.x; // → 15 the property is persistent
```

String

```
var s = new String("hello, world");
s.charAt(0);
s.charAt(s.length-1);
s.substring(1,4);
s.slice(1,4);
s.slice(-3);
s.indexOf("l");
s.lastIndexOf("l");
s.indexOf("l", 3);
s.split(",");
s.replace("h", "H");
s.toUpperCase();
```

Math

```
Math.pow(2,53);
Math.round(.6);
Math.ceil(.6);
Math.floor(.6);
Math.abs(-5);
Math.max(x,y,z);
Math.min(x,y,z);
Math.random();
Math.PI;
Math.E;
Math.sqrt(3);
Math.pow(3, 1/3);
Math.sin(0);
Math.log(10);
Math.log(100);
Math.LN10;
Math.log(512);
Math.LN2;
```

Array

```
a = new Array();    // usage not recommended
a = [];             // recommended usage
a = [1, 2, 3];
a.length;           // → 3
a.push(4);
b = a.pop();        // → 4
delete a[1];        // a[1] is now undefined
1 in a;             // → false
a.length;           // → 3 : length unmodified by delete
a.join();           // → "1,2,3"
a.join(" ");        // → "1 2 3"
a.reverse();
a.sort();
a.concat(. . .);
a.slice(. . .);        // -1 is the last element . . .
a.splice(. . .);       // complex surgery
a.shift();          // = pop to the left
a.unshift();        // push push to the left
```

Array in ES 5

```
a.forEach(f); // applies function f
a.map(f);
// applies function f + returns the array of results
a.filter(f);  // selects according to predicate f
a.every(f);   // && on f applied to items
a.some(f);    // || on f applied to items
a.reduce(f,i);// applies f to items and sums
// results from left to right
a.reduceRight(f,i); // same from right to left
a.indexOf(i);
a.lastIndexOf(i);
Array.isArray(a);
```

this

- `this` is a keyword similar but different from other languages such as Java or C++
- JavaScript code (almost) always runs with a `this` defined :
  - Outside of a function, `this` represents the context overall execution, ie :
    - The `window` object in browsers

```
this === window; // true
```

    - The `global` object in NodeJS

```
this === global; // true
```

  - Inside a function :
    - the object on which the function was called (if it exists),
    - the `this` of the global execution context (if no object calling) in normal mode or
    - `undefined` in" strict mode ".

this - examples

```
var A = {};
function f() { return this === A; }
A.g = function () {
  this.z = 2;
  return this === A;
}
f();            // → false
A.g();          // → true
A.z;            // → 2: property z is assigned on object A
this.z;         // → undefined: z is unknown outside of A
function h() { this.x = 2; }
var B = new h();// any function can be a constructor
                // in this case, in h: this === B → true
B.x;            // → 2
```

Setting this : call, apply et bind

- It is possible to set the value of `this`

```
var A = {
    x: 2,
    f: function (y, z) { console.log(this.x+y+z); }
};
A.f(1,2);              // 5: f is called with this=A
var B = { x: 3 };
B.f(1,2);        // TypeError: undefined is not a function
A.f.call(B, 1, 2);     // 6: f is called with this=B
A.f.apply(B, [1, 2]); // 6: f is called with this=B
```

- It is possible to create a function with a different this and call it later

```
var g = A.f.bind(B, 1, 2); // creates a function
      // this and other arguments are preset
```

that/self/me

- when you define a fonction inside a function, the internal this is not the one you think

```
var A = { id: "toto"};
var id = "titi";
A.getId = function () {
  alert(this.id);
  setTimeout(function() {alert(this.id)}, 100);
}
A.getId(); // shows "toto" and 100ms later "titi"
```

- `this` in the inner function is not A but global
- one solution is to remove the ambiguity by defining another variable such as `that`, `self` or `me`.

```
var A = { id: "toto"};
var id = "titi";
A.getId = function () {
```

Closures

Now this is somewhere else

JavaScript : OO programming

Now, this is somewhere else

Warning : for/in

- Loops `for (a in b) { . . . }` include inherited properties (including system ones)
- unless you use `hasOwnProperty()`

```
for(a in b){
    if (b.hasOwnProperty(a)) {
        . . .
    }
}
```

Strict JavaScript

A stricter version of JavaScript can be used

```
"use strict";
```

```
function f() {
    "use strict";
}
```

- `var` is not optional
- functions called without `this` :`this` is `undefined`, instead of being the global object
- “silent” errors make a throw
- `eval ()` does not create anything in global (no variables, no functions)
- no `with` (which is so evil it is only mentioned here)

“eval is evil”

- Ability to evaluate a string of characters as being JavaScript

```
eval("3+2"); // → 5
```

- `eval` is a function that runs where it is called, in the local context
- to avoid : prevents optimizations

Cleaning up your code

- JSHint
- JSLint

JS on the server

- `node.js` : runtime environment for JavaScript programs on the command line
  - based on the V8 engine (Google Chrome)
  - allows to use JavaScript outside the browser
  - use similar to many other languages (java.exe, perl, python, . . .)
- http daemon included by default
  - makes it easy to deploy a web server
  - to develop server logic in JavaScript
  - equivalent of J2EE, Apache Tomcat and servlets in Java
- Has a system of modules
  - Possibility to import a library developed by someone else
  - there is a module / package manager : `npm`

JSON

- JSON is the use of JS object litterals to transmit data
- Lighter alternative to XML
- Read and Write

```
obj = JSON.parse(line);
line = JSON.stringify(obj);
```

- Example

```
JSON.stringify(book)
// → '{"topic":"JavaScript","fat":true,"author":"Jean Dupont"
```

- Availability
  - frequent extension of ES3
  - native in ES5+

New in ES 6

- `class`, `extends` for prototype-based inheritance, see this
- Arrow function, using =>

```
function (s){ return s.length }
```

is equivalent to

```
s => s.length
```

An arrow function does not have a this (or a context), it shares the this/context of the enclosing function.

- `$` and ‘ (backtick) : string templating

```
var name = "Bob", time = "today";
var t = `Hello ${name}, how are you ${time}?`;
console.log(t); // → "Hello Bob, how are you today?"
```

More ES 6

- Destructuring

```
var t = [1,2,3];
var [a, , b] = t; // works for objects too
console.log(a,b); // → 1 3
var t2 = [...t, 4, 5, 6];
console.log(t2); // → [1, 2, 3, 4, 5, 6]
```

- `let` : variable with block scope
- `const` : constant
- `export`, `import` : gestion des modules
- Default values for formal parameters to a function

```
function f (x, y = 7, z = 42, ...a) {
    return x + y + z
}
f(1) === 50
```

More ES 6

- Support for asynchronous programming : Promise
- For requests made to an entity that does not answer immediately
- Replaces callbacks
- Example

```
// with callback
fs.readFile("emails1A.txt",
            function (content) { /* do something with content
// with promise
fs.readFile("emails1A.txt")
  .then(content => /* do something with content */)
  .catch(error => /* display the error */);
```

- . . . see es6-features.org
- Test on ES6 Fiddle

../tp ../logo-IPP-s

Summary of this lesson

- JS, history, type, syntax
- objects, arrays, expressions, tests, conversions, equality, loops
- functions, scopes, exceptions, types, this
- packages : string, math, array
- good JS : eval, use strict, cleanup
- node.js, JSON, ES6 new elements

Closures and OO moved elsewhere