ENSIMAG 3ème année
2014-2015

JAVASCRIPT MINI-MEMENTO

JavaScript is a lightweight, interpreted, object-oriented language with first-class functions, most known as the scripting language for Web pages, but used in many non-browser environments as well such as node.js or Apache CouchDB. It is a prototype-based, multi-paradigm scripting language that is dynamic, and supports object-oriented, imperative, and functional programming styles. Read more about JavaScript.

Attention: this is a mini memento for more details see `https://developer.mozilla.org/en-US/docs/Web/JavaScript`

1. Values

JavaScript recognizes the following five types of primitive values:

Type	Examples of typed values / Notes
Numbers	42, 3.14159
Logical (Boolean)	true / false
Strings	"Howdy"
`null`	a special keyword denoting a null value; `null` is also a primitive value. Because JavaScript is case-sensitive, `null` is not the same as `Null`, `NULL`, or any other variant
`undefined`	a top-level property whose value is undefined; `undefined` is also a primitive value.

2. Variables

You use variables as symbolic names for values in your application. The names of variables, called identifiers, conform to certain rules. A JavaScript identifier must start with a letter, underscore (`_`), or dollar sign (`$`); subsequent characters can also be digits (`0-9`). Because JavaScript is case sensitive, letters include the characters "`A`" through "`Z`" (uppercase) and the characters "`a`" through "`z`" (lowercase). Some examples of legal names are `Number_hits`, `temp99`, and `_name`.

Declaring variables

You can declare a variable in two ways:

- With the keyword `var`. For example, `var x = 42`. This syntax can be used to declare both local and global variables.
- By simply assigning it a value. For example, `x = 42`. This always declares a global variable and cannot be changed at the local level. It generates a strict JavaScript warning. You shouldn't use this variant.

Evaluating variables

A variable declared using the `var` statement with no initial value specified has the value undefined.

3. Constants

You can create a read-only, named constant with the `const` keyword. The syntax of a constant identifier is the same as for a variable identifier: it must start with a letter, underscore or dollar sign and can contain alphabetic, numeric, or underscore characters. A constant cannot change value through assignment or be re-declared while the script is running.

4. Expressions and operators

An expression is any valid unit of code that resolves to a value. Conceptually, there are two types of expressions: those that assign a value to a variable and those that simply have a value.

The expression `x = 7` is an example of the first type. This expression uses the = operator to assign the value seven to the variable `x`. The expression itself evaluates to seven.

The code `3 + 4` is an example of the second expression type. This expression uses the + operator to add three and four together without assigning the result, seven, to a variable.

JavaScript has the following expression categories:

- *Arithmetic*: evaluates to a number, for example `3.14159`. (Generally uses arithmetic operators.)
- *String*: evaluates to a character string, for example, "`Fred`" or "`234`". (Generally uses string operators.)
- *Logical*: evaluates to `true` or `false`. (Often involves logical operators.)
- *Object*: evaluates to an object. (See special operators for various ones that evaluate to objects.)

Operator	Description	Examples returning true

Equal (==)	Returns true if the operands are equal.	`3 == var1`
		`"3" == var1`
		`3 == '3'`

Not equal (!=)	Returns true if the operands are not equal.	`var1 != 4` `var2 != "3"`
Strict equal (===)	Returns true if the operands are equal and of the same type. See also `Object.is` and sameness in JS.	`3 === var1`
Strict not equal (!==)	Returns true if the operands are not equal and/or not of the same type.	`var1 !== "3"` `3 !== '3'`
Greater than (>)	Returns true if the left operand is greater than the right operand.	`var2 > var1` `"12" > 2`
Greater than or equal (>=)	Returns true if the left operand is greater than or equal to the right operand.	`var2 >= var1` `var1 >= 3`
Less than (<)	Returns true if the left operand is less than the right operand.	`var1 < var2` `"2" < "12"`
Less than or equal (<=)	Returns true if the left operand is less than or equal to the right operand.	`var1 <= var2` `var2 <= 5`

Figure 1 Comparison operators

Operator	Usage	Description
&&	`expr1 && expr2`	(Logical AND) Returns `expr1` if it can be converted to false; otherwise, returns `expr2`. Thus, when used with Boolean values, && returns true if both operands are true; otherwise, returns false.
||	`expr1 || expr2`	(Logical OR) Returns `expr1` if it can be converted to true; otherwise, returns `expr2`. Thus, when used with Boolean values, || returns true if either operand is true; if both are false, returns false.
!	`!expr`	(Logical NOT) Returns false if its single operand can be converted to true; otherwise, returns true.

Figure 2 Logic operators

5. Statements (Control Flow)

JavaScript supports a compact set of statements, specifically control flow statements, that you can use to incorporate a great deal of interactivity in Web pages. Use the semicolon (;) character to separate statements in JavaScript code.

Block Statement

A block statement is used to group statements. The block is delimited by a pair of curly brackets. Block statements are commonly used with control flow statements (e.g. `if, for, while`).

```
while (x < 10) {
  x++;
}
```

Conditional Statements

A conditional statement is a set of commands that executes if a specified condition is true. JavaScript supports two conditional statements: `if...else` and `switch`.

if...else Statement

Use the `if` statement to execute a statement if a logical condition is `true`. Use the optional `else` clause to execute a statement if the condition is `false`. An `if` statement looks as follows:

```
if (condition) {
  statement_1;
} else {
  statement_2;
}
```

`condition` can be any expression that evaluates to `true` or `false`. If `condition` evaluates to `true`, `statement_1` is executed; otherwise, `statement_2` is executed. `statement_1` and `statement_2` can be any statement, including further nested if statements. You may also compound the statements using `else if` to have multiple conditions tested in sequence.

switch Statement

A `switch` statement allows a program to evaluate an expression and attempt to match the expression's value to a `case label`. If a match is found, the program executes the associated statement. A switch statement looks as follows:

```
switch (expression) {
   case label_1:
      statements_1
      [break;]
   case label_2:
      statements_2
      [break;]
   ...
   default:
      statements_def
      [break;]
}
```

The program first looks for a case clause with a label matching the value of expression and then transfers control to that clause, executing the associated statements. If no matching label is found, the program looks for the optional default clause, and if found, transfers control to that clause, executing the associated statements. If no default clause is found, the program continues execution at the statement following the end of switch. By convention, the default clause is the last clause, but it does not need to be so.

Loop Statements

A `loop` is a set of commands that executes repeatedly until a specified condition is met. JavaScript supports the `for`, `do while`, and `while` loop statements, as well as `label` (label is not itself a looping statement, but is frequently used with these statements).

for Statement

A `for` loop repeats until a specified `condition` evaluates to `false`. The JavaScript for loop is similar to the Java and C for loop. A for statement looks as follows:

```
for ([initialExpression]; [condition]; [incrementExpression])
   statement
```

do...while Statement

The `do...while` statement repeats until a specified condition evaluates to `false`. A `do...while` statement looks as follows:

```
do
   statement
while (condition);
```

while Statement

A `while` statement executes its statements as long as a specified `condition` evaluates to `true`. A `while` statement looks as follows:

```
while (condition)
   statement
```

Object Manipulation Statements

JavaScript uses the `for...in, for each...in`, and with statements to manipulate objects.

for...in Statement

The `for...in` statement iterates a specified variable over all the properties of an object. For each distinct property, JavaScript executes the specified statements. A `for...in` statement looks as follows:

```
for (variable in object) {
   statements
}
```

for each...in Statement

`for each...in` is a loop statement introduced in JavaScript 1.6. It is similar to `for...in`, but iterates over the values of object's properties, not their names.

```
var sum = 0;
var obj = {prop1: 5, prop2: 13, prop3: 8};
for each (var item in obj) {
  sum += item;
}
print(sum); // prints "26", which is 5+13+8
```

Comments

Comments are author notations that explain what a script does. Comments are ignored by the interpreter. JavaScript supports Java and C++-style comments:

- Comments on a single line are preceded by a double-slash (`//`).
- Comments that span multiple lines are preceded by `/*` and followed by `*/`

6. Objects

JavaScript is designed on a simple object-based paradigm. An object is a collection of properties, and a property is an association between a name and a value. A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object.

```
var myCar = new Object();
myCar.make = "Ford";
```

```
myCar.model = "Mustang";
myCar.year = 1969;
```

Or

```
myCar["make"] = "Ford";
myCar["model"] = "Mustang";
myCar["year"] = 1969;
```

7. Sameness in JavaScript

x	y	==	===	Object.is
undefined	undefined	true	true	true
null	null	true	true	true
true	true	true	true	true
false	false	true	true	true
"foo"	"foo"	true	true	true
{ foo: "bar" }	x	true	true	true
0	0	true	true	true
+0	-0	true	true	false
0	false	true	false	false
""	false	true	false	false
""	0	true	false	false
"0"	0	true	false	false
"17"	17	true	false	false
[1,2]	"1,2"	true	false	false
new String("foo")	"foo"	true	false	false
null	undefined	true	false	false
null	false	false	false	false
undefined	false	false	false	false
{ foo: "bar" }	{ foo: "bar" }	false	false	false
new String("foo")	new String("foo")	false	false	false
0	null	false	false	false
0	NaN	false	false	false
"foo"	NaN	false	false	false
NaN	NaN	false	false	true