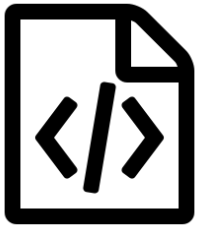
Web Technologies & Image Processing

Jean-Christophe Taveau

2016-2017

JS: Introduction

JavaScript (or JS)



Versions

→ JavaScript 5 (or ECMAScript 5.1)

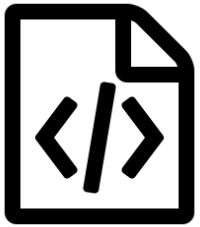
→ Available in all the browsers

→ ECMAScript 2015 (or ECMAScript 6 or ES6)

→ Not all the specs are implemented (browser checking required)

JS: Introduction

JavaScript (or JS)



Features

→ **Imperative** and structured programming (if statement, loops, etc.)

→ **Object-oriented** programming language based on **prototype** for inheritance (no class).

- → Function as constructor
- → Function as method

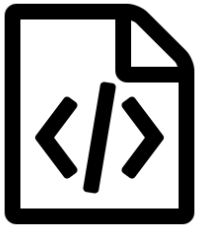
→ **Functional** programming paradigm (first-class: functions considered as object)

- → High-order functions
- → Anonymous
- → Variables, Arguments, etc.

Source: https://en.wikipedia.org/wiki/JavaScript#Features

JS: Introduction

In a HTML page, the code is written within a <script> element.



It may be located:

→ within the <head> element

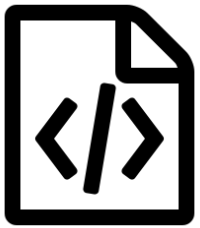
```
<script type="text/javascript">
    console.log("Hello World");
</script>
```

External file: `<script type="text/javascript" src="myScript.js"></script>`

→ Everywhere in the <body> element. **Best place**: At the end of <body> when the page is fully loaded.

→ **BAD!!** Directly in HTML element `<p onclick="alert('Hello');">Click on me </p>`

JS: comments



To document the program, you may add comments that won't be executed by the language engine.

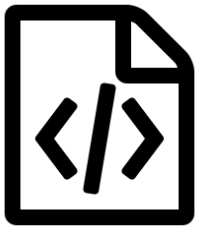
→ In JS, we have two different comments for single-line and multi-lines.

```
var pi = 3.14; // Single-line comment

// This code is commented → var i = 10;

/*
Multi-line
comment
*/
```

JS: semicolon

Each expression must be ended by a semicolon.

```
var i = 1;
i = i + 3;
i = 2 * i; i = i / 10;
```



JS: Basics

→ Variables

→ Conditionals

→ Loops

JS: variables

Variables are defined by keywords

```
var myVar;
```
← global or local variables (Good***)

```
let myVar = 100;
```
← local variables only (New ES6)

```
const PI = 3.14;
```
← constant variables (New ES6). Only readable.

Note: declaring a variable without keyword **var** or **let**

```
myVar = 10;
```
← global variables only (Bad practice - JS warning is sent)

JS: variables

A variable is a “box” allowing the storage of a value.

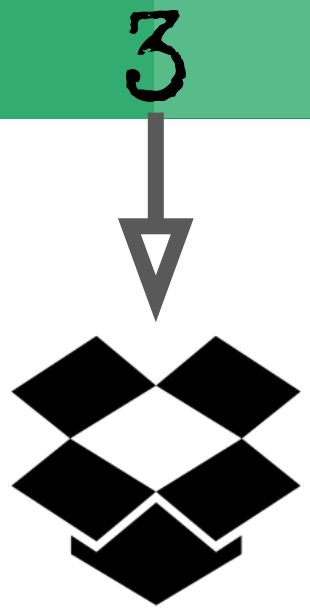
This value may be a:

- Number (1 or 3.14 or 1.2E+12)
- Boolean (true/false)
- String (‘text’ or “text”)
- Array (*List* in Python)
- Object
- No dictionary like in Python

```
var myVar;
```

(1) Declaration

Variable ***undefined***




```
var myVar = 3;
```

(2) Definition

Assign a value: **=**

JS: variables

Variable names must follow rules:

- Only characters of the US alphabet (ascii) are allowed.
- uppercase and lowercase or combination of both.
  - variable name in lowercase. Ex: *myVar* or *my_var* or *myvar*
  - constants in uppercase. Ex: PI.
- cannot begin with a digit.
- cannot contain symbols like -+/*@
- underscore ‘_’ is permitted
- cannot use a keyword of JS language.

JS: numbers

Math operators

- + / * % : Subtraction, Addition, Division, Multiplication, Modulo

+= Increment of quantity. Ex: `i+=2;  i = i + 2;`

-= Decrement of .1Ex: `i-=2;  i = i - 2;`

/= Divided by quantity. Ex: `i/=2;  i = i / 2;`

*= Multiplied by quantity. Ex: `i*=2;  i = i * 2;`

++ Increment of 1. Ex: `i++; i = i + 1;`

-- Decrement of 1. Ex: `i--; i = i - 1;`

Math functions

Object Math. Ex: `var x = Math.cos(Math.PI)`

JS: BOOLean

Comparison operators

x > y is true when x is strictly greater than y
x < y is true when x is strictly less than y
x >= y is true when x is greater than or equal to y
x <= y is true when x is less than or equal to y

x **===** y is true when x is strictly equal to y
x != y is true when x is different of y

Note: Do not use **==** for testing equality because there is an automatic type conversion.

```
var i = ('3' == 3) // True
var j = ('3'=== 3) // False
```

Logical operators

&& (and) **||** (or) **!** (not)

JS: String

```
var str1 = ‘this is a text’;
var str2 = “this is a text with ‘inner quotes’.”;
var str3 = `
This is a multi-line string
**New in ES6**
`;
```

Operators

```
var i = 5
var str = ‘My result is’+ i + ‘ pixels’; // ← My result is 5 pixels
var ch = str[3] // ← ‘r’
```

String Methods

```
var str = ‘My result’.toUpperCase() // ← MY RESULT
var len = str.length // ← Property of String
```

JS: String Methods

charAt()	Returns the character at the specified index.
charCodeAt()	Returns a number indicating the Unicode value of the character at the given index.
concat()	Combines the text of two strings and returns a new string.
includes()	Determines whether one string may be found within another string.
endsWith()	Determines whether a string ends with the characters of another string.
indexOf()	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
match()	Used to match a regular expression against a string.
padEnd()	Pads the current string from the end with a given string to create a new string from a given length.
padStart()	Pads the current string from the start with a given string to create a new string from a given length.
repeat()	Returns a string consisting of the elements of the object repeated the given times.
replace()	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
search()	Executes the search for a match between a regular expression and a specified string.
slice()	Extracts a section of a string and returns a new string.
split()	Splits a String object into an array of strings by separating the string into substrings.
startsWith()	Determines whether a string begins with the characters of another string.
substr()	Returns the characters in a string beginning at the specified location through the specified number of characters.
substring()	Returns the characters in a string between two indexes into the string.
toLowerCase()	Returns the calling string value converted to lower case.
toString()	Returns a string representing the specified object. Overrides the Object.prototype.toString() method.
toUpperCase()	Returns the calling string value converted to uppercase.
trim()	Trims whitespace from the beginning and end of the string. Part of the ECMAScript 5 standard.

JS: OBJECT

Object

A list of zero or more pairs of property names and associated values of an object, enclosed in curly braces ({}).

```
var obj = {} // ← empty object
var lang = {name: “JavaScript”, version: 5} // ← object with two properties
var nn = lang[‘name’]; // ← ‘JavaScript’

var n = lang.name; // ← ‘JavaScript’
var v = lang.version; // ← 5

lang.standard = ‘ECMAScript 5.1’;
```

JS: array

```
var arr = [“JS”,”C++”,3.14,”Python”];
var i0 = arr[0]; // ← ‘JS’
var i3 = arr[3]; // ← ‘Python’
arr[5]=”Java”; // ← [ "JS", "C++", 3.14, "Python", undefined, "Java" ]
```

Methods of Array object

```
var length = arr.length;// ← 4
var last = arr[arr.length - 1]; // ← ‘Python’
var newLen = arr.push("Ada");
//← [“JS”,”C++”,3.14,”Python”,undefined, "Java",”Ada”]
```

JS: array

Properties

length	Reflects the number of elements in an array.

Methods

These methods modify the array:

copyWithin()	Copies a sequence of array elements within the array.
fill()	Fills all the elements of an array from a start index to an end index with a static value.
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements to the end of an array and returns the new length of the array.
reverse()	Reverses the order of the elements of an array in place — the first becomes the last, and the last becomes the first.
shift()	Removes the first element from an array and returns that element.
sort()	Sorts the elements of an array in place and returns the array.
splice()	Adds and/or removes elements from an array.
unshift()	Adds one or more elements to the front of an array and returns the new length of the array.

These methods do not modify the array and return some representation of the array.

concat()	Returns a new array comprised of this array joined with other array(s) and/or value(s).
includes()	Determines whether an array contains a certain element, returning true or false as appropriate.
join()	Joins all elements of an array into a string.
slice()	Extracts a section of an array and returns a new array.
toString()	Returns a string representing the array and its elements. Overrides the Object.prototype.toString() method.
indexOf()	Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.

JS: Basics

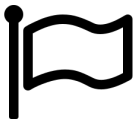
→ Variables

→ Conditionals

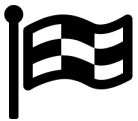
→ Loops

JS: Basics - Block

Block statement (or compound statement)



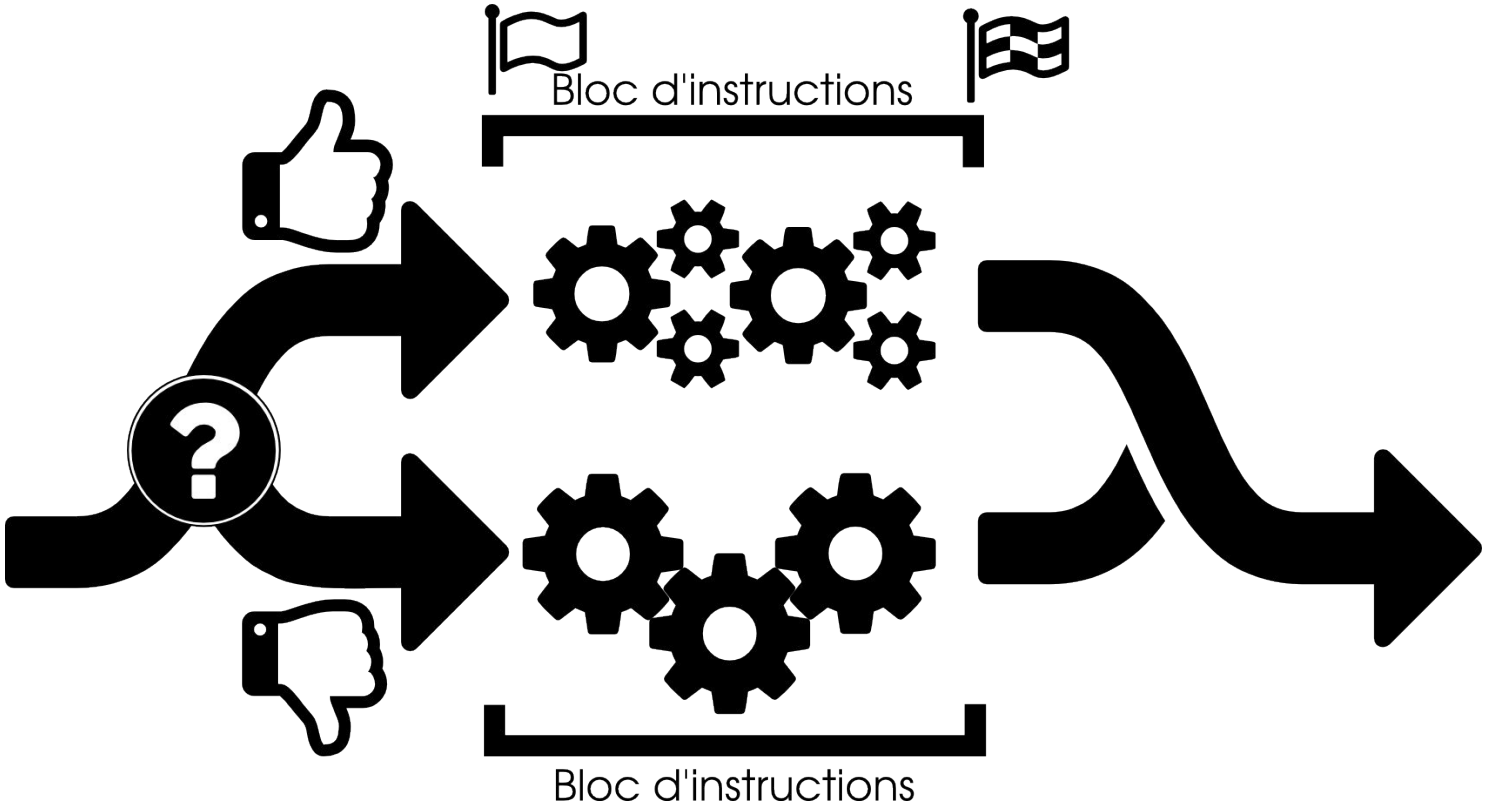
statement_1;
statement_2;
…
statement_n;



```
{
    statement_1;
    statement_2;
    ...
    statement_n;
}
```

Note: Indents are not mandatory. Good for sake of clarity.

JS: Basics - conditionals



JS:CONDITIONALS

```
if (condition) {
     do_something;
     do_something_else;
}
else if (condition2) {
     do_it;
     do_it_again;
}
else {
     otherwise_do_something;
}
```

```
switch (expression) {
case value1:
     do_something;
     break;
case value2:
     do_something_else;
     break;
…
case valueN:
     do_it;
     break;
default:
     otherwise_do_something;
}
```

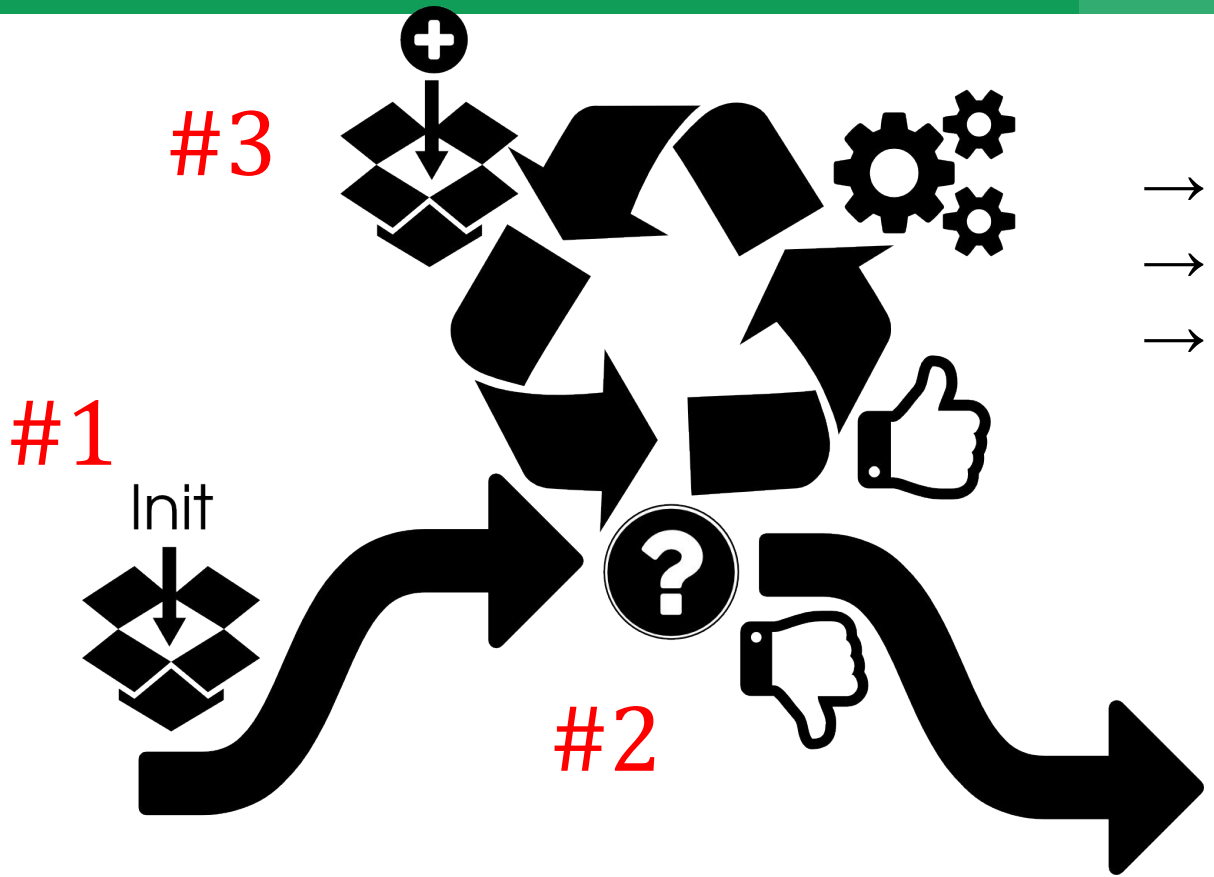
JS: Basics

→ Variables

→ Conditionals

→ Loops

JS: Basics - Loops



→ initializer
→ exit-condition
→ final-expression
→ increment

JS:LOOPS

Loops

→ while

→ do while

→ for

```
init;
while (condition) {
    do_something;
    final_exp;
}

init;
do {
    do_something;
    final_exp;
}
while (condition);

for (init;condition;final_exp) {
    do_something;
}
```

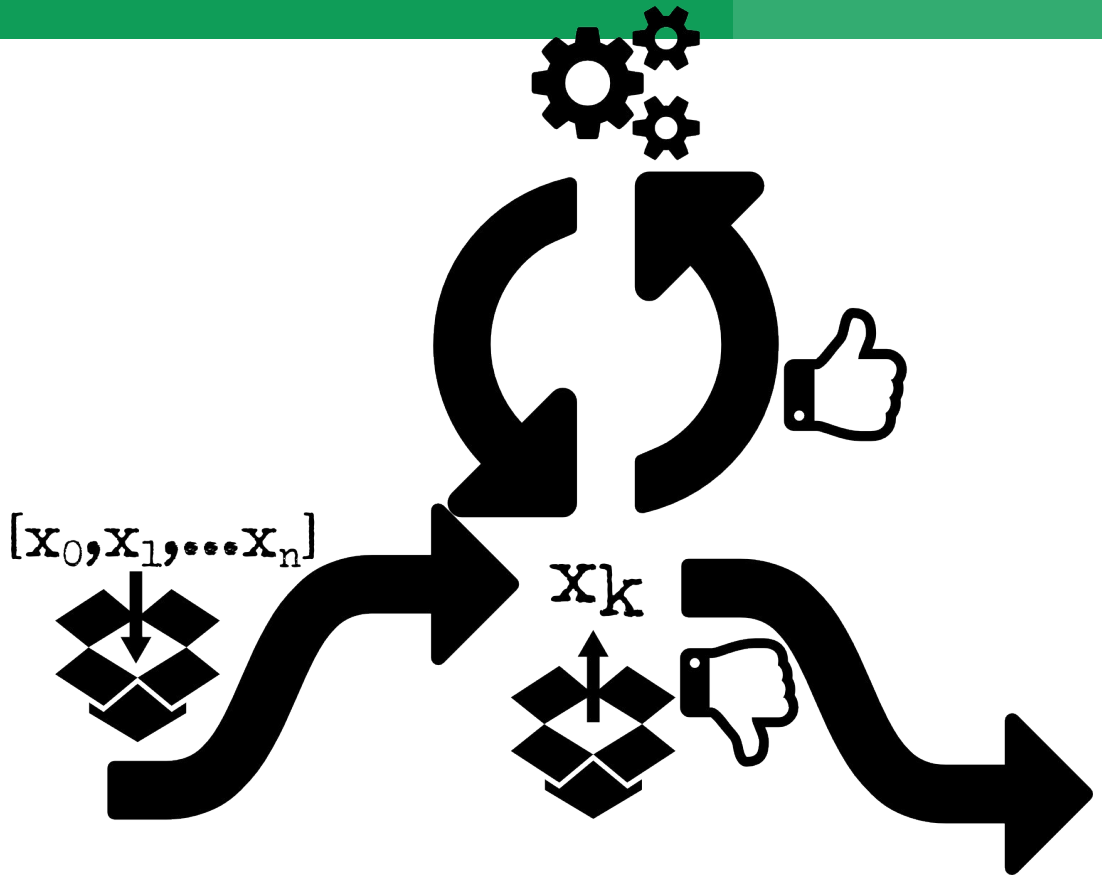
```
i=0;
while (i<10) {
    do_something;
    i++;
}

// Do … while
i=0;
do {
    do_something_else;
    i++;
} while (i<10);

// for
for (var i=0; i<10; i++) {
    do_it;
}
```

JS: LOOPS

Loops (ES6)

→ for … in

→ for … of



JS: LOOPS - FOR ... IN

Available since JS5

for … in → Iterates over enumerable properties of object

```
var arr = [2,4,6,8,10];
for (var i in arr) {
    console.log(i); // ← 0, 1, 2, 3, and 4
}
```

```
var obj = {name: 'Curie', firstname: 'Marie', awards: 'Nobel', children: 2};
for (var prop in obj) {
    console.log(prop)      ; // ← name, firstname, awards, children
    console.log(obj[prop]); // ← Curie, Marie, Nobel, 2

}
```

JS: LOOPS - FOR ... OF

Available in ES6

for … of → Iterates over **collection** (what is iterable)

```
var arr = [2,4,6,8,10];
for (let i of arr) {
    console.log(i); // ← 2, 4, 6, 8, and 10
}

var str = 'crazybiocomputing';
for (let i of str) {
    console.log(i); // ← 'c', 'r', 'a'...
}
```

→ Does not work for **object** because it is not a collection (or iterable).

JS: function

Functions

```
function fname (arg1, arg2,..., argn)
```

```
// Declare function foo
function foo(arg1,arg2) {
     result = do_something;
     return result;
}

foo(2,3);
```

JS: functions arguments

→ No argument

```
function foo() {
    // do something
}
foo();
```

→ Argument passed by value

Primitive types (Number, String,...)
If the function changes the value of an argument, this change is **not reflected globally** or in the calling function.

→ Argument passed by reference

References (object, collection,...)
Object references are values, too, and they are **special**: if the function changes the referred object's properties, that change is visible outside the function.

```
function  foo(a) {
    a +=2;
    return a;
}
var i = 2;
var j = foo(i); // i ← 2; j ← 4

function  bar(a) {
    a.value +=2;
    return a;
}

var k = {value: 2};
var m = bar(k);
// m←Object{value:4}; k←Object{value: 4}
```

JS: anonymous function

Anonymous functions

```
function (arg1, arg2,..., argn)
```

```
var bar = function (arg) {
    result = do_something;
    return result;
};
```

```
var add = function(a,b) {
    return a+b;
};
var result = add(2,3); // ← 5
```

```
var add = function(a,b) {
    return a+b;
};

if (error) {
    add = function(a,b) {
        // Do nothing
    }
}
```

JS: function - optional args

Optional arguments

→ JS5

```
function foo1 (arg1) {
  if (arg1 === undefined) {
    arg1 = 0;
  }
  return arg1;
}

function foo2 (arg1) {
  arg1 = arg1 || 0;
  return arg1;
}
```

→ES6

```
function foo3 (arg1=0) {
  return arg1;
}


foo1(); // ← 0
foo2(); // ← 0
foo3(); // ← 0
```

JS: Functions scopes

Scope of nested functions
→ global, local functions

```
function foobar() {
    function foo() {
        // Do something
    }
    function bar(v) {
        // Do something else
    }

    foo();
    bar();
}
```

```
function foobar(a,b) {
    var k = 3;
    function add(a,b) {
        return a+b;
    }
    function mult(a,b) {
        return a*b;
    }
    return mult(add(a,b),k);
}

var result = foobar(2,4); // ← 18
add(2,4) // ← ERROR
mult(2,4) // ← ERROR
```

JS: function

Functions used as argument of another function

```
// Declare function add
function add(a,b) {
    return a+b;
}

// Declare function scalar
function scalar(a,b,c,fun) {
    return c * fun(a,b);
}

var result = scalar(1,2,3,add); // ← 9 = 3 *(1+2)
var other  = scalar(1,2,3,subtract); // ← 3 * (1-2) = -3
```

Note: Ex: EventListener for DOM events

JS: METHODS

Syntax

```
object . method(args);
```

```
// Declare method foo in obj
var obj = {
  prop : 2016,
  foo  : function(arg1,arg2) {
    return do_something_with_arg1_and_arg2;
  },
  other: 'This is a text'
}

var res = obj.foo(u,v);
```

```
// Example
var myMath = {
  add : function(a,b) {
    return a+b;
  }
}

var res = myMath.add(2,3);
// ← 5
```

JS: OBJECT

How to create several objects (aka instances)?

```
function createObj (yyyy,msg) {
  // Return an object
  return {
    year : yyyy,
    foo  : function() {
      return this.year + ' is ' + this.text;
    },
    text: msg
  }
}

var obj1 = createObj(2025, 'sci-fi')  ;var i=obj1.foo(); // ← 2025 is sci-fi
var obj2 = createObj(2000, 'the past');var j=obj2.foo(); // ← 2000 is the past
```

JS: OBJECT -PROTOTYPE

How to create several objects (prototype)?

```
// Constructor
function Dummy (yyyy,msg) {
    this.year = yyyy;
    this.text = msg;
}

// Method
Dummy.prototype.foo = function() {
      return this.year + ‘ is ‘ + this.text;
}

var obj1 = new Dummy(2025, ‘sci-fi’)  ;var i=obj1.foo(); // ← 2025 is sci-fi
var obj2 = new Dummy(2000, ’the past’);var j=obj2.foo(); // ← 2000 is the past
```