ELEMENTS OF JAVASCRIPT

April 16, 2025 Condor Informatique – Turin

Arguments table

PREFACE

This manual is an introductory course on the JavaScript language; the subjects interested must have some knowledge of programming and of HTML and CSS. The course did not pretend to be exhaustive on the subjects treated, because its purpose is to give a panoramic view enough to develop applications; one can easily find on the Internet all details wants, for example:

http://www.w3schools.com/

On the convention of the notations adopted

The syntax and commands are on `COURIER NEW`, optional parts are enclosed in `[square brackets]` and variables era in *`italics`*.

Examples are also `COURIER NEW`.

Warning

Comments, suggestions and criticisms are welcomed: mail to rossati@libero.it

1 JavaScript Language

JavaScript (often shortened JS) is a programming language mostly used interpreted in the interactive Web pages to give dynamism at Internet applications. On the Web pages JavaScript is used to carry out controls before sending data to the web server and to manage effects of animation on the page or even to create components of the page.

The scripts are inserted in the page by one or several TAGs <script> containing the code or the name of the file with code or simply **instructions** inserted in TAGs for the management of events of that TAG. TAGs <script> are, in general, between the tags <head> </head>, unless the script must run when all elements of the page were loaded, in that case this must be inserted after the closing tag TAG </body>.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <title> Essays de Javascript </title>
      <script type="text/javascript">
      // JavaScript source
      ...
      </script>
      <script type='text/javascript' src='js/handleaid.js'></script>
   </head>
   <body>
      ...
      <input type=button value='w3schools'
onClick='open("http://www.w3schools.com","w3schools","",false)'/>
      ...

   </body>
      <script type="text/javascript">
      // JavaScript source
      ...
      </script>
</html>
```

Figure 1: Where we can insert JavaScript

2 Language elements

2.1 Syntax

A JavaScript program is a text containing lines with language statements:

- JavaScript is case-sensitive,
- the instructions are completed by an optional semicolon (it is needed for multiple instructions on the same line),
- the blocks of instructions are contained in curly braces `{}`,
- `//` allows you to insert a comment on a line,
- `/ * ` contains a commentary on several lines ` * /`

2.2 Data Types

JavaScript allows the execution of tasks acting on data by operators, functions and control structures.
The JavaScript data are objects, between them numbers, character set, dates, arrays of data, functions etc.:

- Number
- String set of characters
- Boolean
- Object
- objects natives
  - Function
  - Array
  - Date
  - RegExp (Regular expression)
  - Math
  - ...
- Null
- Undefined

An object is set or structure of data (properties) and of programs (methods); the data structure define the state and the set of methods describe his comportment.
There are two paradigms for creation of objects: by class or by prototype; the class is a definition of methods and variables (a template) and an object is a specific instance of class and it contains instead of variables real values; the prototype is an object used for create new objects.
JavaScript adopts the prototype paradigm of objects.

2.2.1 Variables

The data or variables are known by name which starts by a letter or `_` or `$`, followed by letters, numbers or underlines. The variables of JavaScript do not have a defined type, must be the user to manage with opportune instructions; variables can contain numbers, characters, and objects, which can contains data (properties) and programs (methods).

Every variable must be declared before use, possibly we can assign a value using the syntax: *`variable = expression`*:

```
discount = 0.05;      // numeric variable
```

Character strings are enclosed between double quotes or single quotes; special characters, such as the carriage return and `\`, are inserted prefixed by `\`, (*escape*) followed by a symbolic name (`'\n'` newline and `'\r'` carriage return), or by itself (`\\`, `\"`, `\'`) or by the hexadecimal value (`\x5B` = `[`):

```
var info = "Condor Informatique - \x5BTurin\x5D";      // characters string
```

The indication `var` before the variable name indicates that the variable is valid locally i.e. it exists only (*scope*) within the function where it is declared, so it is not accessible outside the function. If it is declared outside a function it is created in the `global`[1] object it and it is accessible everywhere.

decimal (base 10)	`0, 117, -345`	`0-9`
octal (base 8)	`015, 0001, -077`	`0-7`
hexadecimal (base 16)	`0x1123, 0x00111, -0xF1A7`	`0-9 A B C D E F` (case is indifferent)

☞ the `0` before the number means a number in octal notation, so `077` is different from decimal `77`.

1. Question: What is the decimal value of 077?

2.2.2 Data sets

The data sets, for example the names of the months, the set of monthly average temperatures in a year, etc. can be arranged in arrays whose single component can be accessed by its position. The elements may be of the same type or

1 In browser JavaScript the `global` object coincide with the `window` object.

different types, that is an array can contain numbers, strings, or even objects.

There are two ways to create an array:

```
Month = Array("January","February","March","April","Mai","June","July",
              "August","September","October","November","December");
Days = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"];
```

Script 1: Two way to declare an array

Arrays keys start at 0, the property `length` returns the size of the array.

Questions.

- *What is the key of October?*
- *Which month has the key equal to 7?*
- *The maxim key of Days array?*

Another important structure are the key value arrays, more properly said objects, that will be detailed in chapter 3 Objects; they are created with this sintax: *objetName* = {*key*$_1$:*value*$_1$[,*key*$_2$:*value*$_2$[, ...]}.

2.3 Operators

The data are transformed by application of functions on them, but normally, especially in the arithmetic expressions, are used, by historical reasons, operators (with the usual precedence of application) and JavaScript, like most of the programming languages, follow these conventions.

- **Arithmetic operators:** * + - / % (division remainder)
- **Comparison operators:** == (equal) < <= > >= != (different)
- **Logical operators:** && (And), || (Or), ! (Not)
- **Concatenation operator:** +
- **Assignation operator:** =
- **Comma operator:** , (to separate instructions, the value of the latter is returned)
- **Others**: typeof, logical on datas (&, |, ~, ^...) ...

There is difference from the assignation operator (=) and comparison operator (==).

The + operators has two meanings[2] when used with string and arithmetic expression it is a concatenation operator; to force the expression evaluation is better to use parentheses:

expression	value	note
"El Condor "+7-12	NaN	NaN is a constant meaning Not a Number
"El Condor "+7+12	El Condor 712	The + operator works only for concatenation
"El Condor "+(7+12)	El Condor 19	The parentheses force the evaluation of the expression
"El Condor "+7*12	El Condor 84	The multiplication operator has greater precedence

2. Question : How many Operators in the statement below and what kind:

```
piGrec = 355/113    // 3.141592... (there is Math.PI)
```

2.4 Assignation

The syntax of the assignment instruction is [var] *variable = expression*. There are also abbreviated forms where the = operator is prefixed by another operator: instead of *variable = variable operator expression* the syntax is *variable operator*= expression.

- += Add (or concatenate) and assign
- -= Subtract and assign
- *= Multiply and assign
- /= Divide and assign

A special case are the operators ++ and -- that add or subtract 1 to a variable; they may precede or follow the variable; therefore the operation is executed before or after the variable evaluation.

There is also a useful syntactic structure which allows the conditional assignment:

2 Technically this is known as *overloading*.

```
variable = (condition) ? value_if_true : value_if_false;
```

3. Exercises

Use only the subtraction operator.

- *Move variable A in the variable B.*
- *Swap the variables A and B.*
- *Add the variables A and B.*

2.4.1 Destructuring assignment

The *destructuring assignment* syntax is an expression for extracts values from arrays or properties from objects setting it into distinct variables.

2.4.1.1 Spread and rest syntax

The spread (. . .) syntax allows to expand an array, for example to provide parameters in a function that accept a variable numbers of them:

```
<input type=button id=Show value="Show">
...
<style>
.rus {color:red}
.gros {width:100px}
</style>
...
var re = /\s*,\s*|\s+/        // split a list on comma or space
document.getElementById("Show").classList.add(..."gros, rus".split(re));
```

Script 2: Use spread for add classes to DOM object

Rest syntax, `...arrayName`, creates the array `arrayName` from multiple element; it allows a function to accept an indefinite number of arguments as array.

```
<input type=button onClick='console.log(means(1,2,-3,4,-5))' value="Means">
...
means = function(...Args) {    // returns media and Standard deviation
      const N = Args.length
      let s = 0,s2 = 0;
      for (const arg of Args) {
            s += arg;
            s2 += arg * arg;
      }
      return [s/N,Math.sqrt(N*s2 - s*s)/N]
}
```

Script 3: Use of rest syntax

2.4.1.2 Extract data from array

`[variable1, variable2, ...] = Array`

Let be the array: `primes = [2,3,5,7,11,13,17,19]`

Assign first 3 values	`[two,three,five] = primes`	`2 3 5`
Omit some values	`[,,,seven,,thirteen] = primes`	`7 13`
Get `eleven` and a set of last in variable `rest`	`[,,,,eleven,...rest] = primes`	`11 [ 13, 17, 19 ]`
Get default value	`[two,,,,,,,extra=97] = primes`	`2 97`

2.4.1.3 Extract data from object

`{variable1, variable2, ...} = Object`

`variable1` can be:

- a name of object property, see 1,
- change the variable name: `ObjectProperty: newName`, see 2,
- assign a default value if the variable doesn't exists, see 3.

Let be an Object: `var parms = {width:-1,height:-1,top:-1,left:-1,padding:5}`

1.	Get some values	`const {top, left} = parms`	`top = -1 left = -1`

2. Changing the variable name `const {top:high, left} = parms  high = -1 left = -1`
3. Get default value `const {margin=10, padding} = parms margin = 10 padding = 5`

2.4.1.4 Extract data from array and object

```
var buttonData = ["After",{"Caption":"Ok","Type":"Submit"},"echo.php"]
const [position,{Caption, Type:type},fnz] = buttonData
console.log(position,Caption,type,fnz)  // After Ok Submit echo.php
```

Script 4: Destructuring from Array and Object

2.5 Program control

2.5.1 Conditional statement

The complete structure of the conditional statement is:

```
if (condition) instruction(s)
else if (condition) instruction(s)
else instruction(s)
```

`else if` and `else` may be missing, `else if` can be repeated.

If there is a block of instruction, it is best, for readability, to indent the instructions.

4. Exercise

Transform the following code using the ? operator.

```
number = Math.ceil(100*Math.random());    // integer number between 1 and 100
if (number % 2 == 0) {
      res = " pair";
} else {
      res = " impair";
}
alert(number + res);
```

2.5.2 switch

`switch` instruction seems an `else if` series:

```
switch (expression) {
            case value1[,value2]:
                  instructions
                  break;
            case value2:
                  instructions
                  break;
            ...
            [default:
                  instructions]
}
```

```
var dt = new Date;          // aujourd'hui
switch (dt.getDay()) {
      case 0:
            d = "Dimanche";
            break;
      case 2:
            d = "Mardi";
            break;
      default:
            d = "un autre jour"
}
alert("today " + d);
```

Script 5: Instruction `switch`

☞ `break` is necessary because when a value satisfies, the following `case` blocks would be executed; `default`, for `cases` not provided.

2.5.3 Loops

The loops allow executing repetitively a set of instructions; we can exit the loop by `break` instruction.

2.5.3.1 Iterations

The simplest form of loop is the `for` loop to cycle a numbers of time on a block of instructions:

```
for (variable=expression;exit_condition; variable_modification)
   Instruction(s)
```

The execution of the `for` ends when *`exit_condition`* is satisfied.

```
number = 2+Math.ceil(998*Math.random());  // integer number between 3 and 1000
signal = " prime";
for (i=3;i*i<=number;i+=2)
      if (number % i == 0) {
            signal = " not prime";
            break;
      }
alert(number+signal);
```

Script 6: Test if a number is prime (with error)

5. Question

What is the error in the script above?.

2.5.3.2 Iterations on arrays and objects

To access the elements of an array we can use the `for` statement we have seen in paragraph 2.5.3.1, however if there are elements that it does not have a numeric key they will not be accessed, in this case we can use a variant:

`for(`*`variable`* `in` *`object_name`*`)` *`Instruction(s)`*

```
for(jour in Jours) if (dt.getDay() == jour) alert("Bonjour " + Jours[jour]);
```

There is also a method of arrays that allows executing a function on all elements (with numeric key):

`array``.forEach(`*`function`*`)`[3]

`function` function has three parameters: *`value`*, *`index`*, and *`array`*.

```
function sigma(value,index,matrice) {
      this.sum = ((index == 0) ? value : this.sum+value)
      if (index == matrice.length-1) alert(this.sum)
}
var a = Array()
for (var i=0;i<10;i++) a[i] = Math.random()
a.forEach(sigma)
```

Script 7: Example of forEach: sum of the elements of an array

2.5.4 while

The following loops instructions are most powerful of `for` :

- `while (`*`condition`*`)` *`instruction(s)`*
- `do` *`instruction(s)`* `while (`*`condition`*`)`

The loop is repeated until `condition` meets.

```
while (a != b) {        // the GCD is a when a=b
    if (a>b) a-= b;
      else b -=a;
}
```

Script 8: Euclid's Greatest Common Divisor

6. Question

What is the difference between `while ...` and `do ... while` ?

3 There is also a second optional parameters.

3 Objects

JavaScript is a language with objects that is to say data are not simply rooms of memory but they have properties and manipulation methods which depend on the type of data, for instance a group of characters (**string**) has, between others, the `length` properties and the method `substr` for take a sub-string; besides all objects have `toString` method, that it gives a legible picture of the object.

The objects, in addition to their methods and properties, can contain others object, as the object `Window` that it contains all structure of the Web page and, in effect, the native functions of JavaScript are contained in the `Window` object.

Objects native of JavaScript can be create implicitly or expressly, definite creation is by the operator `new`; the syntax for methods and properties is *`object.propertie`* and *`object.method`*`([`*`parameters`*`])`.

```
cardinalPoints = new Array("South","North","East","West");
cardinalPoints.length          // 4
cardinalPoints.sort()          // East, North, South, West
```

3.1 Structures accessible by key (arrays and objects)

We have already seen arrays, they are objects whose `length` property gives the number of accessible elements by position, furthermore JavaScript accepts arrays with keys, in this case the declaration is different because, indeed, it is a generic object declaration while arrays are special objects (with own methods):

objetName = {*key*$_1$:*value*$_1$[,*key*$_2$:*value*$_2$[, ...]}

```
objDay =
{"dim":"Dimanche","lun":"Lundi","mar":"Mardi","mer":"Mercredi","jeu":"Jeudi
",5:"Vendredi","sam":"Samedi"};
```

3.1.1 Add elements

`aPoids[aPoids.length] = 44;`	Add an element after the last
`aPoids.push(44);`	Add an element after the last par the array method `push`
`aPoids.unshift(44);`	Add an element before the first
`aPoids["unitMisure"] = "Kilo";`	Add an element with non numeric key, in fact, it adds a property

3.1.2 Accessing and finding elements

The syntax for accede to an element of an array is:

`arrayName``[`*`position_or_key`*`]`

for example Day`[3]`, `objDay["dim"]`.

For non-numeric key there is an alternative syntax:

`arrayName.position`

for example `objDay.dim`.

`arrayName``.indexOf(`*`value`*`)` can be used to test a presence of a particular element, if it not exists the method returns `-1`.

7. Question

- *There are two cases in which the alternative syntax is not applicable. What are they?*

3.1.3 Methods

Sort: *`array`*`.sort()` sorts the elements like a characters so, for example, sorting this array: `[7,100,15,32]` give `[100,15,32,7]` ; with *`array`*`.reverse()` the sort is descendant.

To obtain a numeric sort or a sort according to some criteria, it must be used a variant of the `sort` method:

`array``.sort(`*`function`*`).`

Where *`function`* has two parameters (two elements of the array) and returns one value which tells the relative position of the two (see the Figure 2, where `a` and `b` are the two values).

For example in a numeric sort the function contains `return a - b;`.

`< 0`	`a` must precede `b`
`= 0`	leave unchanged
`> 0`	`b` must precede `a`

Figure 2: Values returned for determine the sorting

Choice : *`array`*`.filter(`*`function`*`)` applies *`function`* to each element of the array and returns an array where there are present only those elements for which *`function`* returns `true`; the first parameter of *`function`* is the value.

3.2 Strings

The `String` object is the builder of the sequence of characters and it inherits its methods such as `substring`, `toLowerCase`, `toUpperCase`, `replace`, `indexOf`, `charAt` etc and the `length` property. One of the methods of the object is `String.fromCharCode` which it creates a character from its numeric representation:

```
alert(String.fromCharCode(0x261b,64))      // ☛@
```

☞ We can access a single position of a string as an array, but only in reading, by the syntax: *`string[index]`*. For example *`string`*`[`*`string`*`.length-1]` is the last character.

There are also methods, deprecated, to add stylistic TAG (HTML), for example `"pig".big()` become `<big>pig</big>`.

3.2.1 Methods whit regular expressions

In the methods to find and/or replace sub strings these can be strings or regular expressions (see par. 9.1 Notes on regular expressions) . JavaScript recognizes the regular expressions because they are enclosed between `//` (or explicitly declared as a `RegExp` object).

The method *`string`*`.search(`*`needle`*`)` gives the position of the subject of research :

```
file = "house.JPG";
alert(file+((file.search(/\.jpg$/i) == -1)?" it isn't":" is")+" an image");
```

The `replace` method returns a new string with parts replaced:

`string``.replace(`*`toReplace`*`,`*`replacement`*`).`

The replacement can be a string or a function which is called for each replacement; in this case the return value of the function must be the replacement string.

☞ Normally `replace` replaces only the first occurrence, for to replace all occurrences we must write *`toReplace`* expressly with the delimiters of regular expression, followed by the modifiers `g` (global):

```
'alfaabetaaa'.replace(/aa/g,'a').
```

In the regular expression what is between parenthesis is memorized in the `$1`, `$2`, … variables; if *`replacement`* is a function, we can access the extracted tokens in the `arguments` array beginning with the key 1, see the script below for how to use.

```
var condor = "Condor Informatique Turin";
var re = /(\w+)\s+(\w+)\s+(\w+)/;
condor.replace(re, "$3 - $1 $2");
condor = condor.replace(re,function(){return arguments[3]+" - "+arguments[1]});
```

Script 9: Examples of `replace` method

```
function checkPsw(psw) {
   var signal = [];
   var lengthPsw = psw.length;
   if (lengthPsw < 6) signal.push("Too short");
   var copyPsw = psw.replace(/[a-z]+/g,"")
   if (lengthPsw == copyPsw.length) signal.push("Almost one lower case letter");
   var copyPsw2 = copyPsw.replace(/[A-Z]/g,"")
   if (copyPsw2.length == copyPsw.length)  signal.push("Almost one upper case letter");
   copyPsw = copyPsw2.replace(/\d/g,"")
   if (copyPsw2.length == copyPsw.length)  signal.push("Almost one digit");
   if (copyPsw.replace(/ /,"").length == 0)  signal.push("Almost one special character");
   return signal.join("\n")
}
```

Script 10 Password constraints, an examples of `replace` method

```
var anchor = "https://www.sermig.org/ target=_blank"
const re = /(.*?(?:\*?target=[']?)(?<target>_\w+)[']?.*)/i
var target = anchor.replace(re,"$<target>")
```

Script 11: Extract value by replace

8. Questions

- *What is the string resulting of the replace method of the examples above?*

3.2.2 Converting strings and arrays

Convert a string to an array:

```
jours = "Dimanche,Lundi,Mardi,Mercredi,Jeudi,Vendredi,Samedi".split(",");
```

Convert an array to a string:

```
pointCard = ["South","North","East","West"];
document.write(pointCard.join(" - "));    // South - North - East - West
document.write(pointCard.toString());     // South,North,East,West
```

3.2.3 Converts to numbers

When a number is contained in string it must be converted for handle correctly the + operator, by the functions `parseFloat` and `parseInt`; the `Number(object)` function convert *`object`* into number or `NAN` (Not A Number).

3.3 Functions

A function is a set of instructions that transform data and possibly returns a value; routine is a synonym of function, but the JavaScript diction is *method*.

There are native functions, they are the methods of the `window` object or methods of objects such as `Date` or `Math`, but naturally we can write our functions for the needs of our applications.

3.3.1 Creation

A function normally has a name, operands, a (possible) return value and some way to declare it:

1) `function` *`functionName`*`(`*`operand`*$_1$`[,`*`operand`*$_2$`[,...]]) {`*`instruction(s)`*`}`
2) `[var]` *`functionName`* `= function(`*`operand`*$_1$`[,`*`operand`*$_2$`[,...]]) {`*`instruction(s)`*`}`
3) `[var]` *`functionName`* `= new Function("`*`operand`*$_1$`"[,"`*`operand`*$_2$`"[,...]], "`*`instruction(s)`*`");`
4) `[var] [`*`functionName`* `=] (`*`operand`*$_1$`[,`*`operand`*$_2$`[,...]]) => {`*`instruction(s)`*`}`

☞ In the first case (*Function Declaration*) the function is disposable before the code parsing, therefore it can be used prior or after its declaration; this is not the case of the other two forms (*Function Expressions*) where the function is created at the execution time.

☞ The fourth form of function declaration (*arrow function*) is concise and more flexible, for example when the body has only one statement it is the value returned like in `log10` function (number 4)).

```
1) function add(x, y) {return(x + y);}
2) var add = function(x, y) {return(x + y);}
3) var add = new Function("x", "y", "return(x+y)");
4) var log10 = x => Math.log(x) / Math.LN10;
```

```
function x(item) {
     var s = 0;
     for (var i=0, n=item.length;i<n;i++) s += item[i];
     return (n > 0)? s/n:0;
}
```

9. Questions

1. *What calculates the function `x`?*
2. *What should be the `typeof` of item?*

3.3.2 Usage

To use a function the syntax is:

```
[[var] variable =] myFunction(operand1[,operand2[,...]])
```

`variable` will contain the result which *`myFunction`* has calculated based on the values of the operands. It is possible to use the result of a function as an operand in another function.

Variables declared as `var` are accessible only inside the function without var they are created in the window object and therefore they are accessible to all functions.

The possible result calculated by the function is returned by the `return` statement.

3.3.3 Advanced aspects

3.3.3.1 Anonymous functions

A function is said anonymous when it is written directly where it is expected a function, for example the second parameter of the `replace` method can be a string or a function that can be written directly:

```
var condor = "z-index:1;font-size: 12pt";
condor = condor.replace(/(\-[a-z])/g,
              function($1){return $1.toUpperCase().replace('-','');});
```

10. Question

What calculates the above anonymous function?

3.3.3.2 Variable number of arguments and polymorphism

The syntax for the functions expects a fixed number of operands, however it is easy to create functions with a variable number of operands, simply, before using a possible operand, check if it is set, using the `typeof` operator; this is useful for create methods with optional operands with a default value.

```
Jours = ["Dimanche","Lundi","Mardi","Mercredi","Jeudi","Vendredi","Samedi"];
function joinArray(array,glue) {
      if (typeof glue == "undefined") var glue = ",";
      return array.join(glue);
}
alert(joinArray(Jours)+"\n" + joinArray(Jours,";"));
```

Script 12: Variable number of parameters and value by default

☞ We can access the operands by the pseudo matrices `arguments` (existing on the inside of the functions), where `arguments.length` is the number of the parameters present.

The *polymorphism* is a property of the languages that have the same function name for functions that may have a number or a different type of parameters; in JavaScript the polymorphism must be solved inside the function with the parameter inspection. An example of polymorphism is the *`string`*`.replace` method, where the second parameter can be a combination of characters or a function.

3.3.3.3 Creation of structures and maintains values

Functions are objects that can contain properties; they are created with the syntax *`objetName`* `= new` *`functionName`* and in the function properties are created by the syntax. *`T`*`his.`*`propertieName`*. Properties can also be inserted after the creation of the object.

Script 13:The object function

```
function person(fname,lname,age,eyecolor) {
   this.firstname=fname;
   this.lastname=lname;
   this.age=age;
   this.eyecolor=eyecolor;
}
staff = [];
staff["Doe"]=new person("John","Doe",50,"blue");
staff["Brush"]=new person("Dick","Brush",55,"blue");
staff["Doe"]["Sex"] = "Male"  // add properties
alert(staff["Doe"].age+staff["Doe"].Sex)
```

JavaScript does not have a variable declaration like `static` in C or PHP, for survive a value in the function so that it is accessible in the following calls, we can add a property:

```
function sigma(value,index,matrice) {
      if (sigma.sum == undefined) sigma.sum = 0;
      sigma.sum += value;
}
var a = Array();
for (var i=0;i<10;i++) a[i] = Math.random();
a.forEach(sigma);
alert(sigma.sum)
```

Script 14: Creation of a permanent variable in a function

3.3.3.4 Encapsulation (*JavaScript module pattern*)

All JavaScript objects are contained in the global object `window`[4], from here the need to avoid name collisions, especially if we use some third part code.

With JavaScript a method is creating an object that contains the properties and methods required but all elements of the object are externally accessible; a solution, which allows to create complex components, uses a particular declaration of JavaScript functions (the n. 2 of paragraph 3.3.1 Creation): the function instead of simply be declared, is immediately executed returning an array of methods and properties that one want to make public; the structure is the following:

```
functionName = (function([operands]) {
      instruction(s);
      var publics = {
            method: function(parameters) {      // method declaration
                  ...
            },
            ...
            get property() {return property;},  // get property
            set property(parameter) {property = parameter;} // set property
      }
      return {publics}
      }([operands]));
```

Note the syntax to `set` and `get` the property.

"`()`" at the end of declaration allows his immediate execution and the function with its *methods and public properties* is added to the `window` space (*Anonymous Closure*) ; methods and properties are accessible by the syntax :

```
functionName.methodName(parameters)
functionName.propertieName
```

The externals parentheses `()` are required by compatibility with all browsers; the possibly `operands` are objects we want to access its methods and properties.

```
chain = (function() {// ********************** chain
  var nRot = 1;            // private variable
  return {
      rotate: function(data,n) {        // rotate words
          if (typeof n == "undefined") var n = nRot;
             var re = /(.+)\s+(\w+)$/;  // $1 all but last word, $2 last word
             var a = data;
             for (var i=0;i < n;i++) {a = a.replace(re, "$2 $1")}
             return a;
      },
      count: function(data) {
             return (data.split(" ")).length;
             },
      sort: function(data) {
             dataMatrix = data.split(" ");
             dataMatrix.sort();
             return dataMatrix.join(" ");
             },
      push: function(data,item) {
             return data+ " "+item;
             }
   }
}());
...
var str = "El Condor";
alert(chain.push(str,"pasa"));
```

Script 15: Functions for handle a word string like an array

11. Exercise

- *Add a method for return the position of one word in the string.*

4 Of course for JavaScript executed in the browser.

3.3.3.5 Add a method to an object

One can add properties and methods to an object, even to native objects. The method or property can be related to instances of the object or to the object itself; In the first case the method acts on an object, in the second case creates an object: for example the `String` object has the `prototype` property that is the object containing methods to act on strings and the `fromCharCode` method to create strings.

The syntax is:

objet.prototype.*newMethod* = function(*operands*) {*instructions*};

objet.newMethod = function(*operands*) {*instructions*};

```
String.fill = function(n,char) {
    if (typeof char == "undefined") var char = " ";
    var a1 = "";
    var a2 = char;
    while (n > 1) {
           if (n %2 == 1) {
                 n--;
                 a1 += a2;
              } else {
                    a2 += a2;
                    n /= 2;
              }
       }
       return a1+a2;
}
String.prototype.rotate = function(n) {        // rotate words
       var re = /(.+)\s+(\w+)$/;  // $1 all but last word, $2 last word
       var a = this;
       for (var i=0;i < n;i++) a = a.replace(re, "$2 $1");
       return a;
}
var condor = "Condor Informatique Turin";
alert((condor.rotate(2)));
alert(String.fill(3,"BA"));
```

Script 16: Added Methods (rotates string word and fill)

12. Exercise

- *Add a method to the object of exercise 11 for replace e a word with another word.*

3.3.4 Native functions

3.3.4.1 Time management

The two functions `setInterval` and `setTimeout` indicates to the browser to execute a JavaScript function after a certain time; they have two arguments: the function to execute, and the waiting time in milliseconds; the functions return a handle which it serves mainly to possibly terminate the management of timer:

handleTimeout = setTimeout(*function*,*time*)

handleInterval = setInterval(*function*,*time*)

The function indicated in `setTimeout` is executed only one time, while that of `setInterval` is run indefinitely; we can terminate with `clearTimeout(`*timeoutHandle*`)` method and `clearInterval(`*intervalHandle*`)` method respectively.

```
...
var varTimer=setTimeout(function(){alert("S'il vous plaît insérez un mot")},10000);
...
<form id=formulaire >
 Ins&eacute;rer un mot<input name="nom" type="text" value="">
 <br>
 <input type=submit value='Envoyer' onSubmit='clearInterval(varTimer))'/>
</form>
```

Figure 3: Example on timer and onSubmit event

3.4 Date

The `Date` object handles dates from the January 1, 1970; there are several ways to create a date:

```
d = new Date();                    // d contains the date of today
```

```
d = new Date(milliseconds);       // 0 is 1 January 1970
d = new Date(dateString);         // ex. d=new Date("July 21, 1983 01:15:00");
d = new Date(year, month, day, hour, minutes, seconds, milliseconds);
```

☞ Only the year is mandatory in the last example.

The `Date` object has several methods, either to get either to change these components; here a few ones:

`Date.now()`	Returns the milliseconds elapsed from January 1, 1970
`getDate()`, `setDate(day)`	Returns or modifies the day of month (between `1-31`)
`getFullYear()`, `setFullYear(year)`	Returns or modifies the year (four digits)
`getHours()`, `setHours(hour)`	Returns or modifies the hour (between `0-23`)
`getMilliseconds()`, `setMilliseconds(millis)`	Returns or modifies the milliseconds (between `0-999`)
`getMinutes()`, `setMinutes(minutes)`	Returns or modifies the minutes (between `0-59`)
`getMonth()`, `setMonth(month)`	Returns or modifies the month (between `0-11`)
`getSeconds()`, `setSeconds(seconds)`	Returns or modifies the seconds (between `0-59`)

☞ *month* is a value between `0` and `11` and *day* between `1` and `31`; `setDate(day)` accept days outside the range 1-31, and amends the date of consequence.

13. Exorcises

- *Take the date of the day and change the day by -1 and 32.*
- *Check if it is possible give the value outside the range for hours, minutes and seconds.*

3.5 Math

The `Math` object contains arithmetic, trigonometric, logarithmic functions, and some constants as π (`Math.PI`) and e (`Math.E`).

In the following example there are used:

- `Math.random` which gives a random real number between `0` (inclusive) and `1` (excluded),
- `Math.ceil` to get the great closest whole number of a real number,
- `Math.pow(base, exponent)`.

```
function testChi(n){
       var a = Array(0,0,0,0,0,0);
       for (var i=0; i < n;i++) a[Math.ceil(6*Math.random())-1] += 1;
       var chi2 = 0;
       for (i=0; i < 6;i++) chi2 += 6*Math.pow(a[i]-n/6,2)/n;
       return chi2;         // error Probability < 0,05 maximum 11,1
}
```

Script 17: Simulation of roll of the dice and the chi square test

3.6 Introspection

Introspection is the ability to examine the type or properties of an object at run-time, in JavaScript we have the `typeof` operator whit possibly answers `number`, `string`, `boolean`, `object`, `null` and `undefined`; for array the method `Array.isArray(obj)`.

The `instanceof` operator is useful for check the type of object:

```
function complexNumber(real, imaginary) {
  this.real = real;
  this.imaginary = imaginary;
}
var point1 = new complexNumber(3,7);
var a = point1 instanceof complexNumber;        // returns true
var b = point1 instanceof Object;               // returns true
```

4 Error’s management

If we have an instruction that can give an error but that doesn’t stops the program, for example a possible receive of data, we put it in a try block and follow it with a catch block where the error is treated see the example below:

```
try {
  document.getElementById("notExistentID").value = "***";
}
catch(error) {
      console.log(error);
      console.log("The ID doesn't exists!");
//    throw("The ID doesn't exists!");
}
finally {
  console.log("Finally: after try-catch");
}
console.log("Continue");
```

The above script signals:

```
TypeError: "document.getElementById(...) is null"
    <anonymous> http://127.0.0.1/condorinformatique/formgen/f.html:34
The ID doesn't exists!
Finally: after try-catch
Continue
```

If we uncomment the `throw` statement the results is:

```
TypeError: "document.getElementById(...) is null"
    <anonymous> http://127.0.0.1/condorinformatique/formgen/f.html:34
The ID doesn't exists!
Finally: after try-catch
uncaught exception: The ID doesn't exists!
```

The `finally` statement is always executed; the statements after `throw` won't be executed.

5 Dealing with asynchronous events

Developing interactive applications requires writing non-blocking asynchronous code that, in JavaScript, can be done by the use of *callback* functions and with the newest added functionalities based on the concept of *promise*.
The necessity to deals whit asynchronous mechanism is for the interaction of application with external services like to ask data to a remote database or to get a web page.

5.1 Callback functions

Callback function deals with event i.e. some code must that be executed when an event occurs; in Figure 3: Example on timer and onSubmit event there is a sample of Callback function. Here below a fragment of a web (executed by node).

```
// server definition and activation ********************
var port = 1337;
var server = require('http').createServer().listen(port);
console.log(getTime() + ' Server running at 127.0.0.1:' + port);
// end server definition and activation ***************
// global variables ***********************************
global.__basedir = __dirname;
var spawn = require('child_process').spawn;
var fs = require('fs');        // handle file system
var path = require("path");
var Regexp = /name=\"(.+)\"\r\n\r\n(.*)\r\n/gm;        // for extract form data
server.on('request', function(request, response) {
  var Url = require('url').parse(request.url,true)
  var pathName = Url['pathname'];
  var inData = '';      //    for POST data recovery (possibly)
  request.on('data', function(chunk) {
      inData += chunk.toString();
  });
  request.on('end', function() {
      if (pathName == "/") pathName = "\\index.html"; // first call
      console.log(getTime() + " Request resource: " + pathName)
      fs.access(__basedir + pathName,(err) => {       // check if file exists)
        if (err == null) {
            var type = path.extname(pathName).substring(1).toLowerCase();
            var cType = "text/html; charset=utf-8"
            if (typeof types[type] != "undefined") cType = types[type];
            var parms = Url['query'];           // data from GET or URI
            // extract fields
            match = Regexp.exec(inData);  // data from form (POST method)
            while (match != null) {
              parms[match[1]] = match[2];
              match = Regexp.exec(inData);
            }
            if (type == "php") {
            ...
```

How we can see the callbacks are “nested” not facilitating the understanding of the program.

5.2 Promise

Instead of using functions that accept a callback, we make a function that returns a *Promise* object, that is, an object representing a value that is generated in the future.
The promise is built around a function that manage an asynchronous request; the constructor of promise is a function with two parameters respectively a ***resolve*** function and a ***reject*** function, that, in the function body, appears depending on the outcome of the request.
The *Promise* can be in one of three states: the initial state (*pending*), the successful state (*fulfilled*) or the failed state (*rejected*). In the last two cases the promise has been honored and the program can manage the answer by the `Promise` method `then` which has two parameters respectively the ***resolve*** function and the ***reject*** function.

In the example below a timer acts as asynchronous function with another timer, with random interval, used for simulate

a ***reject*** clearing the first timer.

```
let promise = new Promise(function(resolve, reject) {
  var endTimeout = function(stop,handle,sTime) {
    if (typeof stop == "undefined") stop = false;
       if (!stop) {resolve({"msg":'Timer has been ended'});
       } else {
         handle.clearTimeout;
          reject(`Timer has been cleared after ${Date.now()-sTime} milliseconds`);
       }
  }
  var startTime = Date.now();
  var handle = setTimeout(endTimeout,2000);
  var millisec = Math.ceil(4000*Math.random());       // integer between 1 and 4000
  setTimeout(() => {endTimeout(true,handle,startTime)},millisec);
});
promise.then(
  (result) => {console.log("Success", result["msg"])},
  (error) => {console.log("Horror", error);}
)
```

Script 18: Promise example: timer possibly cleared

5.3 Async and Await

Asynchronous programs using *Promise* have a sequential structure that permits a better understanding of the code. An alternative that has a simplicity of sequential execution without the block of other tasks, is given by functions declared asynchronous that contain the waiting of asynchronous task(s) i.e one resolution of the *Promise*.

```
(functionName = async function() {
      await promise;
      ...
}
```

```
async function timeout(ms,ms2) {
  var startTime = Date.now();
  await new Promise((resolve, reject) => {
      var endTimeout = function(stop,handle,sTime) {
        if (typeof stop == "undefined") {resolve('Timer has been ended');
        } else {
            handle.clearTimeout;
            reject(`Timer has been cleared after ${Date.now()-sTime} ms`);
        }
      }
      var handle = setTimeout(endTimeout,ms);
      setTimeout(() => {endTimeout(true,handle,startTime)},ms2);
      }).then(
        (result) => {console.log("Success", result)},
        (error) => {console.log("Horror", error);})
}
timeout(2000,Math.ceil(4000*Math.random()))
```

Script 19: Async await example: first timer can be cleared

In the async function it is possible wait a set of events by `await`ing a **`Promise.all`**:

```
function startTimer (Id,ms) {
      return new Promise(resolve => {
            setTimeout(function() {resolve("");
            console.log(`Timer ${Id} has been ended after ${ms} milliseconds`)}
            ,ms)})
}
let events = [];
var sTime = Date.now();
for (i=0;i<3;i++)
    events.push(startTimer("ABC"[i],Math.ceil(4000*Math.random())));
(waitTimer = async function() {
  await Promise.all(events);
```

```
  console.log(`Timers has been ended after ${Date.now()-sTime} milliseconds`)
})();
```

Script 20: Async await example: Promise.all waits multiple events

A possible run of the above script is like this:

```
C:\www\condorinformatique\nodejs>node awaitAll.js
Timer C has been ended after 1418 milliseconds
Timer A has been ended after 1848 milliseconds
Timer B has been ended after 2612 milliseconds
Timers has been ended after 2614 milliseconds
```

6 DOM (Document Object Model)

The **Document Object Model** (**DOM**) is a standard which represents an interface independent from any programming language allowing programs to achieve and, possibly, to change contents, structure and/or style of XML and HTML documents.

DOM represents the document, and his elements, as a tree structure; in the browser this structure is contained in the window `object` (which coincide with the `global` object), see the underneath figure.

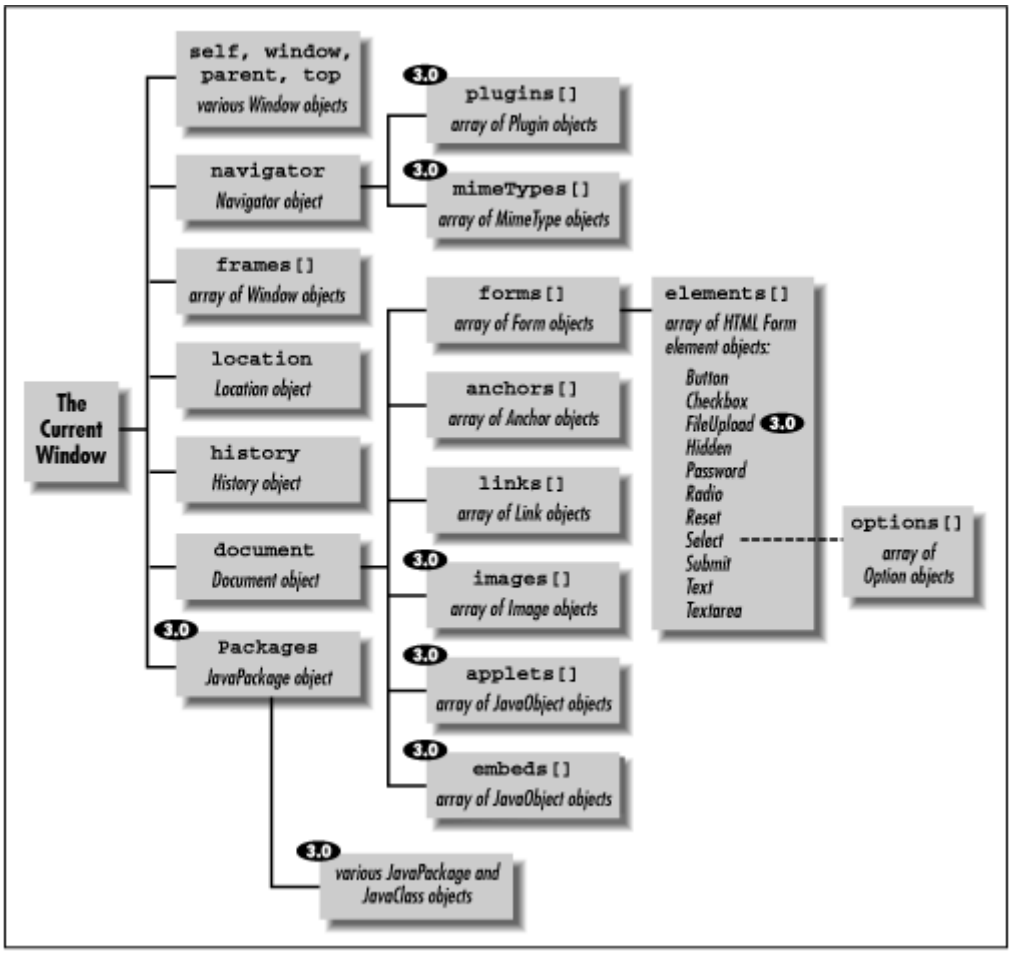


Figure 4: The window object

The square brackets that follows some objects indicate that there is an array of objects. The name of each object is prefaced by the names of all preceding objects from the root, separated by dots. For example, the `window` object contains an array of `forms` objects where each can contains objects for inputting text, buttons, etc:

```
window.document.forms[0].elements[3]   // first form fourth object
```

All object names begin with the name of the root object, namely `window`, but JavaScript permits omitting:

```
document.forms[0].elements[3]          // first form fourth object
```

Properties, methods, and window objects are global and accessible in any JavaScript code.

6.1 Methods of window object

6.1.1 Interaction with user

- `alert(`*`message`*`)` signal to the user
- `confirm(`*`message`*`)` ask to confirm (answer `true`) or cancel
- `prompt(`*`message`*`,`*`defaultText`*`)` show a dialog box for insert data

6.1.2 Interaction with the browser

- `open(`*`URL`*`,`*`name`*`,`*`specs`*`,`*`replace`*`)` `open` a page (and creates a new `window` object)

- *window*.close() close a page
- *window*.resizeTo(*width, height*) scales a page
- *window*.moveBy(*x,y*), *window*.moveTo(*x,y*) relative or absolute page displacement

☞ the opener property of the document object permits access to the parent window.

```
function w() {
       var nw=window.open("", "", "width=100, height=100");
       nw.resizeTo(500,400);
       nw.document.write("<button onclick='javascript:window.opener.document.bgColor =
\"green\"'>vert</button>");
       nw.document.write("<br><button onclick='javascript:window.opener.document.bgColor
= \"silver\"'>argent</button>");
       nw.document.close();
}
w();
```

Script 21: Window creation

☞ Instead of these commands, which pose problems on different browsers, it is better to use the creation of a 'window' by HTML tags (such) which may be rendered visible (and hidden).

6.2 The object document

When an HTML document is loaded in the browser, it becomes the document object; the document object is the root node of the HTML document and the "ancestor" of all other nodes; with JavaScript we can access to the properties and to the methods of all the objects of the nodes.

6.2.1 Accessing elements

The document object contains the node body and a collection of nodes (elements) such as forms and images (all of Figure 4 where there are square brackets), besides all nodes are accessible by some methods:

- *element*.getElementsByTagName(*tagName*) creates an array of objects, if *tagName* is * the array will contain all element 's nodes.

  ☞ *element* can be from document to a node, for example a form node.
- document.getElementById(*id*) accede to an element by is *id*,
- document.getElementsByClassName(*className*) return an array of elements which has the class *className*,
- document.getElementsByName(*name*) array of object of forms that it have the same name.

☞ There are properties with ... Element ... and ... Elements ... this means that the value obtained is, in one case an object, in the other an array. The id attribute must be unique in the page (but it is not controlled), on the contrary the a set of radio buttons have the same name and the same name can be in several forms.

The property parentNode of an element refers to its parent; the property this in the context of the management of the events of a form references the object that generated the event and this.form is the form itself.

```
var tags = document.getElementsByTagName("*");       // all document tags
var ids = Array();
var str = "";
for(var i=0,n=tags.length;i<n;i++) {
  if (tags[i].id != "") {
       ids[tags[i].id] = typeof ids[tags[i].id] != "undefined") ? (ids[tags[i].id]+1):1
}
for (i in ids) {
  if (ids[i] > 1) str += i+"\n";
}
alert((str != "")? "Multiple Id\n"+str:"Not multiples id")
```

Figure 5: Control for multiple id

6.2.2 Add a node

We must create a node and the possible text content, or "clone" a node present, by methods:

- document.createElement(*tagType*)
- document.createTextNode(*text*)

- *element*.cloneNode(*boolean*) if *boolean* is true are copied the possible nodes children.

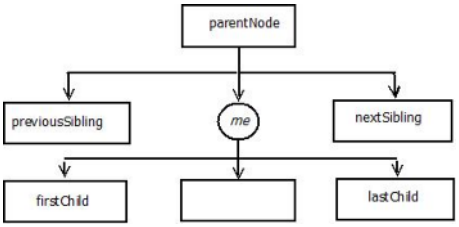
We insert the node at the point wanted by the methods:

- *parentNode*.appendChild(*node*)
- *parentNode*.insertBefore(*node*, *place*)
- *node*.after(*newNode*)
- *node*.before(*newNode*)

parentNode can be a node that can have a child like body, form, etc.
parentNode and *node* can be:

- a node individuated by its *id*,
- a node individuated by its relative position (see *Figure 6: Position relative to (me) node*)
- a node contained in one collection.

also *place* is a node (not body of course).



- *Figure 6: Position relative to (me) node*

☞ These instructions must be performed after the document has been built, see the example below.

```
...
<p id='parag'>Paragraph</p>
</body>
</html>
<script type="text/javascript">
var btn=document.createElement("input");
btn.setAttribute("type","button");
btn.setAttribute("value","Silver");
document.body.appendChild(btn)
btn2=btn.cloneNode(true)
btn2.setAttribute("value","Olive");
document.getElementById("parag").appendChild(btn2);
</script>
```

Script 22: Add a button in body and after an element

☞ For some elements like TABLEs and lists there are methods ad hoc.

6.2.3 Move or delete a node

The methods *parentNode*.appendChild and *parentNode*.insertBefore applied to an existing node have the effect of moving the node:

```
<input type=button value='Send'
 onClick='this.form.appendChild(document.getElementById("condor"))'/>
```

To delete nodes we must know the parent of the node and use the method:

- *Node*.remove() (not supported by Internet Explorer)
- *parentNode*.removeChild(*node*)

```
<p>paragraphe.</p>
<input type=button value='Efface'
       onClick='deleteElement(document.getElementById("parag"))' >
<span id="parag">Anothere paragraph.</span>
</div>
...
<script>
function deleteElement(el) {
       el.parentNode.removeChild(el);
}
</script>
```

Figure 7: Deleting a node

6.2.4 Modify a node

6.2.4.1 Access to the properties

The node can have attributes (the one who are contained in the TAG) and properties which may be confused because they are related, but in reality are two different things although, in general, the properties are synchronized with the attributes.

	Attribute name	Attribute	Properties
Input type text	`value`	The initial value	The actual value
Input type check box	`checked`	Empty string	`true` or `false`
Anchor (A TAG)	`href`	The initial value	Complete URL

Figure 8: Differences between attributes and properties

Other differences are:

- the name of the attributes are case-insensitive,
- the `class` attribute corresponds to the `classname` property,
- in the tag we can insert "owners" attributes, who do not become properties,
- with JavaScript we can add properties to the node, they not become attributes.

Access to the HTML attributes of a node is possible using the following methods:

- *node*`.hasAttribute(`*name*`)` control if exists the attribute *name* on *node*
- *node*`.getAttribute(`*name*`)` take the attribute *name* of *node*
- *node*`.setAttribute(`*name*`,`*value*`)` modifies the attribute *name* of *node*
- *node*`.removeAttribute(`*name*`)` remove the attribute *name* of *node*

14. Exercise

Prepare an HTML page with a form to evaluate attributes and properties

6.2.4.2 innerHTML, innerText and textContent properties

These properties are used to read or modify the contents of a TAG: `innerHTML` insert a structure HTML which becomes immediately operative (☞except scripts for security reasons), `innerText` (IE) and `textContent` (all browsers) inserts a text.

☞ With `innerHTML` we can create a structure, also complex, more easily than by methods views in paragraph 6.2.2. They are adapted for interactive use while `innerHTML`, in general, is used by add a content coming from the WEB server.

```
var rough=setInterval(roughClock,1000);
function roughClock() {
  var d = new Date();
  var clock = (d.getHours()+9900)+":"+(d.getMinutes()+9900)+":"+(d.getSeconds()+9900);
  $("clock").innerHTML = clock.replace(/99/g,"");
}
...
<div id='clock'></div>
```

Script 23: innerHTML and a rough clock

6.2.4.3 Change the node style

The object `style` of an element contains his stylistics properties; the name of the property coincides, normally, with the equivalent name of CSS, except for the names with dash which are translated according to the rule, known as *CamelCase*: the dash is deleted and the following letter is changed to uppercase.

`document.getElementById(`*id*`).style.`*property*`=`*style*

ex. `document.getElementById(id).style.backgroundColor = "olive";`

```
String.prototype.toCamel = function(){
        return this.replace(/(\-[a-z])/g, function($1){
            return $1.toUpperCase().replace('-','');
            });
};
...
alert("background-color:yellow;".toCamel())
```

Script 24: Change the CSS name style to JavaScript name style (Camel function)

The *element*`.setAttribute(`*attributeName*`,`*attributeValue*`)` can be used to set the attribute `style`. The *attributeValue* must be a string of attribute with the CSS syntax:

```
var elem = document.getElementById(id);
elem.setAttribute("style","width:500px; background-color: yellow;");
```

☞ because `setAttribute` replaces the existing styles, so possibly changes by setting `style` property must occurs after `setAttribute`.

6.2.5 Events

The `document` object can associate JavaScript code to events occurring on one of his components, the code will be executed only when the event will take place; the word `this` is the reference to the object where the event has taken place (Figure 9). The capture of an event causes the creation of the `event` object whose main properties are:

- `type` event name
- `target` the object where the event had take place
- `srcElement` the object where the event had take place (IE)

```
<div id=div1 onClick='alert(event.type+" "+event.target+" "+event.target.id)'>
<img src="images/condor.gif" id=condor >
<input type="button" value=" 17 " id=button>
<img src="images/logopace.gif" id='drapeau paix' >
</div>
```

☞ in the HTML tags the events name aren't case-insensitive, this is not the case of JavaScript:

```
document.getElementById("div1").onclick = function(event){alert(event.type+"
"+event.target+" "+event.target.id)};
```

Mouse events

`onclick`, `ondblclick`	when click on object one or two times,
`onmousedown`, `onmouseup`	capture start and end of the click action on object,
`onmousemove`	when the mouse pass on an object,
`onmouseover`, `onmouseout`	when the mouse enter and leave an object.

```
<script language="Javascript" type="text/javascript">
essay = {
 pi: 3.141592653589,
 perimeter: function(r) {return 2*this.pi*r;}
}
</script>
...
Perimeter and Pi Greek
<br><input type="button" value="Pi Greek" onClick='alert(essay["pi"])' />
<br><input type="button" value="Perimeter"
      onClick='alert(essay.perimeter(10))' />
<img src="images/condor.gif"
                    onMouseOver='this.src="images/logopace.gif"'
      onMouseOut='this.src="images/condor.gif"' />
```

Figure 9 : Events onClick, onMauseOver, onMauseOut end `this` object

Keyboard events

`onkeydown`, `onkeypress`, `onkeyup`

Objects events

`onload`	when an object (image, page) is loaded on the browser.
`onerror`	when an object has not loaded, for example because is a corrupted image.

`onabort`, `onresize`, `onscroll`, `onunload`

Forms events

`onblur`	when an object lost the focus.
onchange	when an object has changed, for example a select of a combo item, a change of check box status, etc.
`onreset`	when the `reset` button has been clicked.
`onsubmit`	before to send of form to web server.

7 Ajax

Ajax (*Asynchronous JavaScript and XML*) is based on the `XMLHttpRequest` object that it allows an easy way to retrieve data from a URL[5] to update only a portion of the page.

The object has methods:

- `open(`*`parameters`*`)` determine URL, mode and type of communication, in particular, the parameters are:
  - *`method`* `GET` or `POST` for send to or receive data from WEB server[6].
  - *`URL`* the server script which receive the query; if the method is GET it may be followed by data in the format `?`*`name=value`*`[&`*`name=value`*`[...]]`. Multiple values has the form: *`name`*`[]=`*`value`*.
  - *`asyncFlag`* optional: `true` is the default value which indicates that the request is asynchronous, in this case it must be specified a function that will handle the response.
  - *`userName`* *and* *`password`* needed only if are requested for accessing the site.
- `send(`*`content`*`)` Transmit the request; content is present only if the method is `POST`, it can be a simple string of characters with syntax as in the GET method (without `?`), or an object, for example `formData` that lets we send the contents of a form.

and properties:

- `onreadystatechange = `*`function`*`)` this is an event property where *`function`* is the name of the function which handle the change of the status of the request; it is necessary if *`asyncFlag=true`*.
- `readyState` property which indicates the status of the query, the value 4 indicates the end.
- `status` is the result of the query: `404` for 'Not Found', `200` for 'OK'.
- `statusText` description of the result,
- `responseText` the data.

7.1 Usage

Synchronous utilization (deprecated)	Asynchronous utilization
Creation of the `XMLHttpRequest` object	
	Activation of the propriety `onreadystatechange`
Start of communication by the `open` method	
	Possible sending of header (s) by the method `setRequestHeader`
Sending of data by `send` method	
Collection of data by the property `responseText`	

7.1.1 Usage of HTTP `GET` method:

The possible data are the part of URL called *query string*; they have the form:

```
field=value[&field=value[…]]
```

The data are separated from the address by character **`?`**, for example :

```
www.sitenom/scriptnom.php?weight=76&name=El+Condor
```

In the *query string* are accepted only alphanumeric characters and **`- _ . ~`**; all other characters must be encoded with `%`*`nn`* where *`nn`* is the hexadecimal value of the character. The space is `%20`, it can be also coded by `+`, however it is better to use the function `encodeuricomponent`.

The `send` method must be used with parameter `null`; in the example below there is a request to get a file and for evaluate an expression:

```
function getData(url) {
    AJAX=new XMLHttpRequest();
```

5 URL (*Uniform Resource Locator*) or, informally, web address.
6 There are other methods.

```
    AJAX.open("GET", url, false);              // false = requête synchrone
    AJAX.send(null);
    return AJAX.responseText;
}
function callPHP() {
  var cmd = 'wrapper.php';
  cmd += "?eval='+encodeURIComponent($('cmd').value))";  // caractères spéciaux
codifiés
  cmd += "R"+ Math.random()+"=0";         // empêche à IE de retourner la cache
  return(getData,cmd);
}
...
<form id='formulaire' >
 Ins&eacute;rez une commande <input name="cmd" id="cmd" type="text" size=30>
 <br>
      <input type=button value='Evaluer' onClick='alert(callPHP())'/>
      <input type=button value='GET Source'
onclick='alert(getData("prova.js"))' />
 </form>
```

Script 25: Example with method GET and synchronous communication

7.1.2 Usage of HTTP `POST` method

Whit `POST` method data to send are indicated in the `send` method (`null` if in `open` the method is `GET`); data can have different format: strings in the format of *query string* or some objects, between what has particular importance the `FormData` object; in the first case before the `send` method, you must specify the type of data that must be sent by the method `setRequestHeader`.
`FormData` object permits the creation of a set of couples of key-value; the data are sent whit the same format of the submitted forms; we can add couples of key-value by the `append` method.

☞ `FormData` generated by a form doesn't contain the values of radio buttons, check boxes and combo boxes if those objects aren't selected.

```
function fromForm(frm) {
  var fData = new FormData(frm);
  if (!frm["radio"][0].checked && !frm["radio"][1].checked
      fData.append("radio", false);
  if (!frm["checkbox"].checked) fData.append("checkbox", false);
  if (frm["combo[]"].selectedIndex == -1) fData.append("combo", Array(""));
  ajax("wrapper.php",fData,function(content) {alert(content)});
}
...
 <input type=button value='Submit form' onClick='fromForm(this.form)'/>
...
function ajax(url,data,handler) {       // if no handler synchronous call
  var async = (typeof handler != "undefined")? true:false;
  var ajx=new XMLHttpRequest();
  if (async) ajx.onreadystatechange = function(){
      if (ajx.readyState == 4) {
            if (ajx.status == 200) {return handler(ajx.responseText);}
              alert((ajx.status == 500)? "Erreur du Serveur":"Erreur " + ajx.status);
            }
  }
  ajx.open("POST", url, async)
  if (typeof data != "object") {
      ajx.setRequestHeader("Content-type", "application/x-www-form-urlencoded")
      ajx.send(data);
  } else ajx.send(data);
  if (!async) return ajx.responseText;
}
```

Script 26: Ajax : example of send of strings or forms by `POST` method

7.2 Json

JSON (JavaScript Object Notation) is a data-interchange format based on a subset of JavaScript; JSON data is a string of characters enclosed in curly brackets, here below a simplified syntax.

```
{                                   {
data:value,                         "Name":"Joe Condor",
...                                 "Age":78,
array:[item,...],                   "Languages":["C","PHP","JavaScript"],
...                                 "Competence":{"C":"low","PHP":"high",
object:{key:value,...}                  "JavaScript":"medium"}
...                                 }
}
```

JavaScript has the object JSON that has two methods `parse` that create an object from the JSON string and `stringify`, that constructs a JSON texts of a JavaScript variable.

```
> var obj =
JSON.parse('{"id":"","center":"Yes","top":"100","left":"100","width":"-1","height":"-1"}
')
> obj
{ id: '',
  center: 'Yes',
  top: '100',
  left: '100',
  width: '-1',
  height: '-1' }
> JSON.stringify(obj)
'{"id":"","center":"Yes","top":"100","left":"100","width":"-1","height":"-1"}'
```

8 Graphics

HTML5 introduces the TAG `<canvas ... >`, which is a container used to achieve graphics using JavaScript.
The `canvas` tag essentially wants the `width` and `height` attributes, which give the size in pixels of the graphic object; `id` attribute is useful for to individuate it.

```
<canvas id="canvasID" width="300" height="300"></canvas>
```

The `canvas` object has (indirectly) more methods to create and manipulate lines, rectangles, arcs, text, and images; in fact these methods belong to a HTML5 object accessible via the `getContext` method.

```
var ctx=document.getElementById(canvasID).getContext("2d");
```

☞ At the beginning the origin of coordinated, the point (0.0), is in the upper left corner; the coordinates of the y-axis are growing toward the bottom of the screen.

The drawing surface may be limited by the `clip` method which identifies a portion of the canvas through a closed shape within which subsequent graphical instructions take effect, leaving unchanged the abroad.

8.1 Colors

The properties `fillStyle`, `strokeStyle` and `shadowColor` set the color used respectively for the filling, the strokes and the shadows. The color values are the same as in CSS:

Hexadecimal value	`#rrggbb` ou `#rgb`	`fillStyle = '#800000'  // red` `strokeStyle = '#0f0'  // green`
Color name	16 key-word taken from palette of VGA Windows: **aqua**, **black**, **blue**, **fuchsia**, **gray**, **green**, **lime**, **maroon**, **navy**, **olive**, **purple**, **red**, **silver**, **teal**, **white**, et **yellow**.	`shadowColor = 'silver'` `strokeStyle = 'fuchsia'`
Function	`rgb(red,green,blue)` or `rgba(rrd,green,blue,transparency)`	`fillStyle = 'rgb(0,0,255)' // blue`

color is a string or an object `gradient` (see parag. 8.6.3 Gradients).

`rgba(red,green,blue,transparency)` the values of colors are integers number between 0 and 255 included; the value of *transparency* ranges between 0 (transparent) and 1 (opaque).

☞
```
function tryGround(canvasID,color) {
      var id = document.getElementById(canvasID)
      var ctx=id.getContext("2d");
      ctx.fillStyle=color;
      ctx.fillRect(0,0,id.width-1,id.height-1);
}
...
<input type=button value='Ground'
onClick='tryGround("canvas1","rgb(0,255,255)")'/>
```

Script 27: Filling the background color of canvas

8.2 Texts

There are two methods to draw text: `fillText(text,x,y)` and `strokeText(text,x,y)`. The first draws the text filled, the second draws the outline. Each method also provides a fourth optional argument for the maximum width.

The `font` property specifies the font of the text according to the syntax of the property *font-family* of CSS.

The properties `textAlign` and `textBaseline` are used to align the text, for example:

```
ctx.textAlign = 'center';
ctx.textBaseline = 'middle';
```

draws the text centered on the given coordinates.

```
function essayText(ID) {
       var ctx=document.getElementById(ID).getContext("2d");
       ctx.fillStyle="#800000";
       ctx.strokeStyle="#008000";
       ctx.font = '24px Unknown Font, sans-serif'
       ctx.textAlign = "center";
       ctx.strokeText("strokeText",100,30);
       ctx.fillText("fillText",100,65);
}
```

Script 28: Canvas: texts

The method `ctx.measureText(text)` expose a property `width` of the *text* itself:

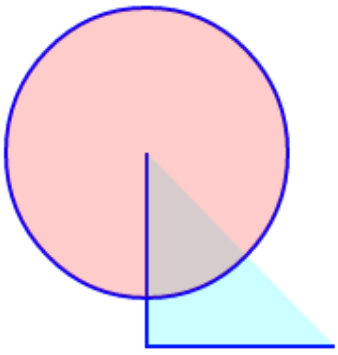
```
var textWidth = ctx.measureText(title).width
```

8.3 Lines and shapes

Line drawing begins with the `beginPath()` method and ends with `stroke()` or `fill()` method. The start point of the draft is individuated by the `moveTo(x,y)` method or implicitly by the `arc()` method; the end of the line becomes the beginning of the next line.

- `lineWidth(dimension)` dimension of lines,
- `setLineDash(pattern)` *pattern* is an array of numbers that specifies alternately a line and a gap ex. `setLineDash([2,2])`; an empty array restore the continuous line,
- `fill()` fill one shape (and possibly close it),
- `stroke()` draws,
- `arc(x,y,radius,startAngle,endAngle,direction)` draws an arc; the angles are expressed in radiant, if *endAngle - startAngle* is equal to 2Π (`2*Math.PI`) a circle is draws; the draft is according to the movement of the watch if *direction* is `false`,
- `lineTo(x,y)` draws a line until the coordinate *x,y*,
- **`fillRect`**`(x,y,width,height)` draws and fill a rectangle,
- **`strokeRect`**`(x,y,width,height)` draws a rectangle,
- **`clearRect`**`(x,y,width,height)` efface a rectangular surface,
- `quadraticCurveTo(cp1x,cp1y,x,y)` et `bezierCurveTo(cp1x,cp1y,cp2x,cp2y,x,y)` draw a curve that connects the current point to the point (*x, y,*) according to one or two control points (said intuitively points of attraction).

```
function tryShapes(canvasID,couleur) {
  var id = document.getElementById(canvasID);
  var ctx = id.getContext("2d");
  ctx.beginPath();
  ctx.fillStyle=couleur;
  ctx.strokeStyle="rgb(0,0,255)";
  ctx.lineWidth = 2;
  ctx.moveTo(100, 100);
  ctx.lineTo(100, 200);
  ctx.lineTo(200, 200);
  ctx.stroke();
  ctx.fill();
  ctx.beginPath();
  ctx.arc(100,100,75,0,2*Math.PI,true)
  ctx.stroke();
  ctx.fillStyle="rgba(255,0,0,0.2)";      //transparent
  ctx.fill();
}
...
<input type=button value='Shapes'
onClick='tryShapes("canvas1","rgba(0,255,255,0.2)")'/>
```



Script 29: Canvas: circle and triangle

☞ The omission of `closePath()` method avoids the drawings of the diagonal edge of the triangle, but `fill()` method consider form as closed.

8.4 Images

8.4.1 Create image

The images in PNG, GIF, or JPEG can be used in the canvas, by `drawImage()` method who has three variants (***dx***, ***dy*** is the coordinate of the point of insertion of the image).

`drawImage(`*`image`*`,`***`dx`***`,`***`dy`***`)`	the image is inserted unchanged
`drawImage(`*`image`*`,`***`dx`***`,`***`dy`***`,`*`dw`*`,`*`dh`*`)`	the image is inserted modified
`drawImage(`*`image`*`,`*`sx`*`,`*`sy`*`,`*`sw`*`,`*`sh`*`,`***`dx`***`,`***`dy`***`,`*`dw`*`,`*`dh`*`)`	the image or a part is inserted and modified

The first parameter is an image: a `canvas` or `img` tag that it can be accessed by its `ID` or an `Image` object, whose property `source` can be an image on the server and, in this case, it must wait for the end of his loading (see example).

```
function essayImage(canvasID,image) {
  var id = document.getElementById(canvasID);
  var ctx = id.getContext("2d");
  ctx.fillStyle="#007FFF";
  ctx.fillRect(0,0,id.width-1,id.height-1);
  var img = new Image();
  img.onload = function(){
      ctx.drawImage(img,70,70);
      ctx.drawImage(img,10,10,110,60);
      ctx.drawImage(img,0,0,100,100,160,160,70,120);
  }
  img.src = image;
}
...
<input type=button value='Image'
onClick='essayImage("canvas1","images/condor.gif")'/
>
```

Script 30: The three variants of drawImage

8.4.2 Manipulate image

The `getImageData` method create an `imageData` object whose property `data` is the graphic or part in form of array of pixels, where every pixel is represented by four bytes: red, green, blue and transparency and it can be used for manipulate the graphic.

```
...
var imageData = ctx.getImageData(30,30,100, 100);
var data = imageData.data;
for (var i = 3; i < data.length; i += 4) data[i]  = 127; // half transparency
...
```

Script 31: getImageData

The insertion of a portion of image is by `putImageData` method that, in its complete form, it allows to put only a part of `imageData` object, the so called *dirty* rectangle, in fact the entire image is virtually copied back but not what is outside the dirty rectangle.

```
var canvas = document.getElementById("MyCanvas");
var ctx = canvas.getContext("2d");
ctx.fillStyle = 'rgb(0,0,255)' // blue
ctx.fillRect(50, 50, 50, 50);
var canvasImg = ctx.getImageData(0,0,300, 300);
var data = canvasImg.data;
for (var i=0;i<250;i++) {
      var rnd = Math.floor(2500*Math.random()); // 50*50
```

```
      var x = 50+Math.floor(rnd/50);
      var y = 50+rnd % 50;
      data[4*(y*300+x)+3] = Math.floor(255*Math.random());  // set alpha
}
ctx.putImageData(canvasImg,-50,-50,50,50,50,50)
```

Script 32: putImageData

8.4.3 Save image

`toDataURL(imageType)` method returns the contents of the `canvas` as a picture (base64 codified) which we can use as source of another `canvas` or `img` tag or to send to WEB Server. Default of *`imageType`* is `image/png`.

```
...
<canvas id="canv" width="300" height="300"></canvas>
...
<input type=button value='Save' onClick='uploadImage(document.getElementById("canv"))'/>
...
function uploadImage(canvasID) {
       var canvasData = canvasID.toDataURL("image/jpg");
       var ajax = new XMLHttpRequest();
       ajax.open("POST",'saveImage.php',false);
       ajax.setRequestHeader('Content-Type', 'application/upload');
       ajax.send(canvasData);
}
```

Script 33: Upload (synchronous) canvas image as jpeg

8.5 Geometrical transformation

Draft space is controlled by a `3x3` transformation matrix which, at start is unitary (left matrix on Figure 10); the change of the values on the first two lines of the matrix allow modifying the successive drafts:

- **a** change the size of the horizontal draft
- **b** skew the drawing horizontally, the value is the tangent of the angle
- **c** skew the drawing vertically, the value is the tangent of the angle
- **d** change the size of the vertical draft
- **e** move the drawings horizontally
- **f** move the drawings vertically

1	0	0
0	1	0
0	0	1

a	c	e
b	d	f
0	0	1

Figure 10: Transformation matrix

There are methods that modify the matrix for move, rotate and resizing; the `transform` and `setTransform` methods modify the matrix which allows, with a single statement, to perform multiple transformations; both methods have six parameters:

`context``.setTransform(a,b,c,d,e,f);`

`setTransform()` creates a new matrix; `transform()` modifies the matrix by multiplying it by the parameters.

☞ All transformation affect only the drawings created after the transformation methods.

8.5.1 Save end restore draft status

The `save()` and `restore methods()` are useful for the creation of complex graphics: `save()` push the actual status of the graphic (transformation matrix, colors, etc) in a stack memory, and `restore()` pops the previous status.

8.5.2 Move the origin

The `translate(x,y)` method moves the origin la position that is to say the initial coordinates of the upper left corner will become (-x, -y):

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
// au centre du canvas
ctx.translate(canvas.width/2,canvas.height/2);
ctx.textAlign = 'center';
ctx.textBaseline = 'middle';
ctx.fillStyle = 'red';
ctx.font = '30pt Calibri';
ctx.fillText('El Condor pasa!', 0, 0);
ctx.fillStyle = 'blue';
```

```
ctx.beginPath();
ctx.arc(-canvas.width/2,- canvas.height/2,
       75,0,2*Math.PI,true);
ctx.fill();
```

Script 34: Canvas: move of origin

8.5.3 Change the scale

The `scale(w,h)` method modifies the ratio of graphic; negative values gets a change of the direction of the axes.

```
...
ctx.fillStyle="#FFFF00";
ctx.save();
ctx.scale(1.5,1);
ctx.arc(100,100,30,0,2*Math.PI)
ctx.fill();
ctx.restore();
ctx.font = '20pt Calibri'
ctx.fillText("Ellipse",118,160);
ctx.scale(1,-1);
ctx.fillStyle="#00FF00";
ctx.fillText("Ellipse",118,-165);
...
```

Script 35: Canvas: ellipse and mirror effect

8.5.4 Rotation

The `rotate(angle)` method execute a rotation of graphic, *`angle`* is en radiant. This method is a transformation that acts on the scale and the inclination simultaneously.

8.5.5 Complex transformations

The `setTransform()` method can perform translation, rotation and scale simultaneously; the method is required if we want to draw, in a non-orthogonal space:

```
ctx.setTransform(1.5,0,1,-1,100,100)
```

the above transformation dilate the x axis, incline the y axis by 45 degrees, the coordinates of the y-axis are growing toward the top of the screen and the point (0,0) is moved.

8.5.6 Set the classical Cartesian coordinates

By geometrical transformation we can set the origin in the center of the canvas and the coordinates developing from left to right and from bottom to top, this is a translation and an inversion of the direction of `y` ordinate; the two fragments are equivalent:

```
ctx.setTransform(1,0,0,-1,canvas.width/2,canvas.height/2);
```

```
ctx.translate(ctx.canvas.width / 2, ctx.canvas.height/2);
ctx.scale(1,-1);
```

In the script below a function for draw text correctly.

```
function textOut(ctx,txt,x,y,fnz) {
      if (typeof fnz == "undefined") fnz = "strokeText";
      ctx.save();
      ctx.setTransform(1,0,0,1,0,0);
      ctx[fnz](txt,x+ctx.canvas.width/2,-y+ctx.canvas.height/2)
      ctx.restore();
}
...
textOut(ctx,"R'cos(\u03B1'/2)",20,-95,"fillText"); // \u03B1 is Greek alpha
```

Script 36: Draw text in classical Cartesian coordinates

8.6 Special effects

8.6.1 Image composition

The `globalCompositeOperation` propriety definite or return the way in which a source image (new) is drawn on an existing image destination; among its values there is:

- `source-over` the source image is draft above the destination image, this is the initial composition,
- `destination-over` the source image is draft below the destination image,
- `source-atop` it is visible only the source image part internal to the destination image,
- `destination-atop` it is visible only the image destination part internal to the source image.

```
if (ctx.globalCompositeOperation == "source-over")
                    ctx.globalCompositeOperation = "source-atop";
```

8.6.2 Shadows

The size of the shadow is the size of the object plus the size of the area of blur; there are four property to create the shading effect:

`shadowOffsetX`	pixel distances between the object and the shadow	`5`
`shadowOffsetY`		`-5`
`shadowBlur`	pixel size of the blur area	`15`
`shadowColor`	shadow color	`"red"`

Figure 11: Properties for shadowing

15. Exercise

Draw the image of Figure 11.

8.6.3 Gradients

The gradient is a gradual transition between at least two colors, which can be applied to lines and forms by the methods `strokeStyle()` and `fillStyle()`. The gradient is an object that it can be created linear or radial.

The colors are indicated by the gradient method `addcolorstop(`*`position`*`, `*`color`*`)`; where position varies between 0 and 1 and tells where, in the gradient, the *`color`* stops, from this point the color is transformed in the next color that it is reached to its stop point:

```
var grd=ctx.createLinearGradient(0,0,150,0);
grd.addColorStop(0.3,'red');
grd.addColorStop(0.7,'green');
```

In the example there are red at the beginning for 3/10 of the gradient, followed by a transition to green who finished at 7/10 of the gradient; the rest is filled by the last color (green).

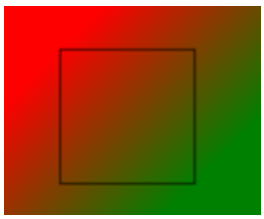
8.6.3.1 Linear gradients

This gradient is obtained by parallels colored lines.

`grad = ctx.``createLinearGradient(`x_0`, `y_0`, `x_1`, `y_1`);`

The parameters of method are the coordinates of two points which individuate the direction of the transition lines and also the portion of the canvas space where it applies:

```
var id = document.getElementById("canvas1");
var ctx = id.getContext("2d");
ctx.fillStyle="#00ff00";
var grd=ctx.createLinearGradient(50,50,150,150); // diagonal gradient
grd.addColorStop(0.0,'red');
grd.addColorStop(1,'green');
ctx.fillStyle=grd;
ctx.fillRect(0,0,id.width,id.height);   // all canvas
ctx.strokeRect(50,50,100,100)
```



Script 37: Canvas: linear gradient

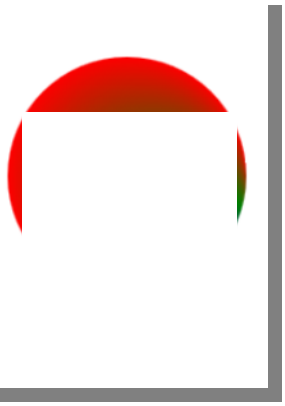
8.6.3.2 Radial gradients

In the radial gradient the transition is by a truncated cone identified by two circles and the transition is from the edge of the first circle to the edge of the second circle.

`grad = ctx.``createRadialGradient(`x_1`, `y_1`, `r_1`, `x_2`, `y_2`, `r_2`);`

If r_1 is > `0` he color inside the first circle is is the first color.

```
var x1 = 200;       // x of 1. circle center point
var y1 = 200;       // y of 1. circle center point
var r1 = 0;         // radius of 1. circle
var x2 = 100;       // x of 2. circle center point
var y2 = 100;       // y of 2. circle center point
var r2 = 90;        // radius of 2. circle
var radialGrad = ctx.createRadialGradient( x2, y2, r2,x1,y1,r1);
radialGrad.addColorStop(0, "red");
radialGrad.addColorStop(.5, "green");
radialGrad.addColorStop(1,"blue" );
ctx.fillStyle = radialGrad;
ctx.beginPath();
ctx.arc( x2, y2, r2,0,2*Math.PI,true)
ctx.fill();
```



Script 38: Canvas: radial gradient

8.6.4 Motifs

The method `createpattern(`*`image`*`, `*`repeat`*`)` uses the image to create a pattern object. The second argument can be a string with one of the values `repeat` (the default value), `repeat-x`, `repeat-y`, and `no-repeat`.

```
var ptrn = ctx.createPattern(img,'repeat');
ctx.fillStyle = ptrn;
ctx.fillRect(0,0,150,150);
```

9 Annexes

9.1 Notes on regular expressions

A regular expression is a string of characters used to search, check, extract part of text in a text; it has a cryptic syntax and here there is a sketch with a few examples.

The regular expression is contained between `//` and can be followed by modifiers such as **`i`** to ignore the case.

The expression is formed with the characters to search in the text and control characters, among the latter there is a `\` said *escape* used to introduce the control characters or categories of characters:

- **`\` escape character**, for special characters (for example asterisk) or categories of characters:
  - **`\w`** any alphabetical character, **`\W`** any non alphabetical and numerical character,
  - **`\s`** *white space* namely. tabulation, line feed, form feed, carriage return, and space,
  - **`\d`** any numeric digits, **`\D`** any non digit,
- **`.`** any character,
- **quantifiers**, they apply to the character(s) or group that precede:
  - **`*`** zero or more characters
  - **`+`** one or more characters
  - **`?`** zero or one character (means possibly)
  - **`{n}`, `{n,}` et `{n,m}`** respective exactly *n* characters, almost *n* characters and from *n* to *m* characters.

`(...)` what is between parentheses is memorized (unless see line above),

`(?:...)` a non-capturing group,

`?=pattern` checks if *pattern* exists,

`[a-z]` any letter from `a` to `z` comprises,

`[a|b]` `a` or `b`,

`$` (at the bottom),

`^` (at start).

9.1.1 Examples

`^\s*$`	empty set or white spaces
`aa+`	find a sequence of two or more `a`, like `aa`, `aaa`,....
`(\w+)\s+(\w+)\s+(\w+)`	find and memorize three words
`/(.+)\s+(\w+)$/;`	`$1` is all words except the last, `$2` is the last word
`(\-[a-z])`	find and memorize minus followed by one alphabetic character
`.+\.jp[e]?g$`	controls file type `jpg` or `jpeg`
`^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$`	control of mail address
`^[-+]?\d{1,2}(\.\d{1,2})?$`	**Numeric values** `[-+]?` the sign is possible `\d{1,2}` one or two digits `(\.\d{1,2})?` It is possible to have a decimal point followed by one or two digits

```
function validateEmail(mailValue){  // tahnks to Zaheer
   var mailPattern = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
   return mailPattern.test(mailValue);
}
```

Script 39: Control of mail address

9.2 Tables

9.2.1 Figures

9.2.2 Scripts

9.2.3 Question's answers

Question 6 In `do ... while` the *`statements`* are always executed one time.
Question 7 1) The name is not a valid variable name. 2) the name is contained in a variable.
Questions 8 1) Turin - Condor Informatique 2) Turin – Condor 3(_blank

Questions 9 1) arithmetic average 2) `object` array
Question 10 Clears the dash and capitalize the following letter (Camel function).