ÉLÉMENTS DE JAVASCRIPT

16 avril 2025 Condor Informatique – Turin

Table des matières

PRÉFACE

Ce manuel est un cours introductif sur le langage JavaScript ; il est nécessaire que les sujets intéressés aient quelque connaissance de programmation, de HTML et de CSS. Le cours n'a pas la prétention d'être exhaustif sur les sujets traités, car son but est de donner une panoramique suffisant à débuter le développement des applications ; on peut trouver aisément sur Internet tous les détails voulu, par exemple :

http://www.w3schools.com/

Sur la convention des notations adoptées

La syntaxe des commandements et en `COURIER NEW`, les parties optionnelles sont renfermées de `[crochet]` et les mots variables sont en *`italique`*.
Les exemples aussi sont en `COURIER NEW`.

Les mots anglais (*english*) sont normalement en *italique* et entre parenthèses.

Avertissement

Les codes dans ce manuel (LOGICIELS) sont fourni par El Condor sans aucunes garanties de n'importe quelle sorte, concernant la sécurité, la conformité, le manque de virus, inexactitudes, erreurs typographiques, ou d'autres composantes malfaisantes de ces LOGICIELS.

Il y a des dangers inhérents dans l'utilisation de n'importe quel logiciel et vous êtes uniquement responsables de déterminer si ces LOGICIELS sont compatibles avec votre équipement et d'autre logiciels installé sur votre équipement. Vous êtes aussi uniquement responsables de la protection de votre équipement et de vos données, pourtant El Condor ne sera pas responsable pour aucuns dommages que vous pouvez subir avec l'utilisation, la modification ou la distribution de ces LOGICIELS.

Vous pouvez utiliser ces LOGICIELS librement, si vous voulez vous pouvez me créditer dans le commentaire du programme :

El Condor – CONDOR INFORMATIQUE – Turin

Les commentaires, les suggestions et les critiques sont bien accueillies ; le courrier est rossati@libero.it

1 Le langage JavaScript

JavaScript (souvent abrégé JS) est un langage de programmation principalement utilisé dans les pages web interactives pour donner du dynamisme aux applications Internet. Dans les pages web JavaScript est utilisé pour exécuter des contrôles avant d'envoyer les données au web serveur et pour gérer des effets d'animation sur la page ou même pour créer des composants de la page.

Les pièces de programme sont inséré dans la page par un ou plusieurs TAGs `<script>` contenants ou le **code** ou le nom du **fichier** avec le code ou simplement des **instructions** insérés dans des TAGs pour la gestion des événements du TAG même. Les TAGs `<script>` sont, en général, entre les tags `<head></head>`, à moins que le script doit être exécuté lorsque tous les éléments de la page ont été chargés, dans ce cas le script doit être inséré après le TAG de fermeture `</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: Où se peut insérer JavaScript

2 Éléments du langage

2.1 Syntaxe

Un programme JavaScript est un texte contenant des lignes avec des instructions du langage.

- JavaScript est sensible à la casse,
- les instructions sont terminées par un point-virgule (facultatif),
- les blocs d'instructions sont renfermés entre accolades `{}`,
- `//` permet d'insérer un commentaire sur une ligne,
- `/*` renferme un commentaire sur plusieurs lignes `*/`

2.2 Types de données

JavaScript permet d'exécuter des tâches agissant sur des données par opérateurs, fonctions et structure de contrôle de l'exécution des instructions. Les données de JavaScript sont des objets, entre eux il y a nombres, ensembles de caractères, date etc :

- nombres (*Number*)
- chaînes de caractères (*String*)
- Boolean
- objets (*Object*)
- objets natives
- Fonctions (*Function*)
- matrices (*Array*)
- Date
- RegExp
- Math
- ...
- Null
- Undefined

Un objet est une ensemble ou structure de données (propriété) et de logiciels (méthodes) ; cette structure de données définit son état et l'ensemble des méthodes décrit son comportement.
Il y a deux paradigmes de création d'objets : par classe ou par prototype ; la classe est une définition de méthodes et de variables (un modèle) et un objet est une instance spécifique d'une classe qu'il contient à la place des variables des valeurs réelles ; le prototype est un objet à partir duquel on crée des nouveaux objets.
JavaScript adopte le paradigme des objets à prototype.

2.2.1 Variables

Les données ou variables, sont connu par un nom. Le nom de variable commence par une lettre ou `_` ou `$`, éventuellement suivi par lettres, nombres ou soulignements. Les variables de JavaScript n'ont pas un type défini, il doit être l'utilisateur à gérer le contenu avec instructions opportunes ; les variables peuvent contenir des données numériques, des caractères et des objets avec leurs données (*propriétés*) leurs programmes (*méthodes*).

Toutes variables doivent être déclarées avant usage, éventuellement on leur peut assigner une valeur en utilisant la syntaxe *`variable = expression`* :

```
escompte = 0.05;      // variable numérique
```

Les chaînes de caractères sont renfermées entre guillemet ou entre apostrophes ; les caractères spéciaux, comme le retour a la ligne (*new line*) et `\`, sont inséré préfixé par `\`, (*escape*) suivi par un nom symbolique (`\n` retour à la ligne), soi même (`\\`, `\"`, `\'`) ou la valeur hexadécimal (`\x5B` = `[`) :

```
var info = "Condor Informatique - \x5BTurin\x5D";      // chaîne de caractères
```

L'indication `var` avant le nom de la variable indique que la variable est valable localement c'est-à-dire elle existe seulement (portée ou *scope*) à l'intérieur de la fonction où elle est déclaré, donc elle n'est pas accessible au dehors de la fonction ; si elle est déclarée au dehors d'une fonction elle est créée dans l'objet `global`[1] accessible partout et sa portée est globale.

Les nombres sont acceptés en notation décimal, octal et hexadécimal :

Notation	Exemples	Chiffres acceptées
décimale (base 10)	`0, 117, -345`	`0-9`
octale (base 8)	`015, 0001, -077`	`0-7`
hexadécimal (base 16)	`0x1123, 0x00111, -0xF1A7`	`0-9 A B C D E F` (casse indifférente)

☞ Il est le `0` avant le numéro qui indique la notation octale, donc `077` il est différent de `77` décimal.

1. Question

Quel est la valeur décimal de `077` ?

1 Dans le JavaScript du navigateur l'objet `global` coïncide avec l'objet `window`.

2.2.2 Ensembles de données

Les ensembles de données, par exemple les noms des mois, des jours, les températures moyenne d'un mois, etc., peuvent être organisées en matrices (*array*) dont le single composant est accédé par sa position. Les éléments peuvent être du même type ou différents, c'est-à -dire une matrice peut contenir nombres, chaînes de caractères ou même objets.

Il y a deux façons de créer une matrice :

```
Mois = Array("Janvier","Février","Mars","Avril","Mai","Juin","Juillet",
             "Août","Septembre","Octobre","Novembre","Décembre");
Jours = ["Dimanche","Lundi","Mardi","Mercredi","Jeudi","Vendredi","Samedi"];
```

Script 1: Deux façons de déclarer une matrice

Les clés des matrices débutent de 0.

2. Questions

- *Quel est la clé `d'octobre` ?*
- *La clé `7` de la matrice `Mois` corresponde à quel mois ?*
- *La clé maxime de la matrice `Jours` ?*

2.3 Opérateurs

Les données sont transformées par application de fonctions sur eux, mais normalement, surtout dans les calcules arithmétiques, ils sont utilisée, par des raisons historiques, des opérateurs avec leur précédence et JavaScript, comme la plupart des langages de programmation, suit ces conventions.

- **Opérateurs arithmétiques :** `* + - / %` (reste de la division)
- **Opérateurs de comparaison :** == (égale) < <= > >= != (divers)
- **Opérateurs Logiques :** **`&&`** (And), **`||`** (Or), **`!`** (Not)
- **Opérateurs d'Enchaînement :** +
- **Opérateur d'assignation :** =
- **Opérateur virgule :** **`,`** (pour séparer des instructions, dont la valeur de la dernière est retourné)
- **Autres** : `typeof`, logiques sur les données (`&`, `|`, `~`, `^`...)...

Il y a différence entre opérateur d'assignation (=) et l'opérateur de comparaison (==).

L'opérateur + a deux usages[2], lorsqu'il est utilisé avec chaîne de caractères et expression arithmétique il est un opérateur de concaténation ; pour forcer l'évaluation de l'expression il est préférable d'utiliser des parenthèses :

expression	valeur	note
`"El Condor "+7-12`	`NaN`	`NaN` (*Not a Number*)est une constante qui signifie valeur non numérique
`"El Condor "+7+12`	`El Condor 712`	L'opérateur + agisse pour enchaîner
`"El Condor "+(7+12)`	`El Condor 19`	Les parenthésés forcent l'évaluation de l'expression
`"El Condor "+7*12`	`El Condor 84`	L'opérateur de multiplication a la précédence sur l'opérateur d'addition

3. Question

Combien d'opérateurs dans l'instruction ci-dessous et de quel type :

```
piGrec = 355/113    // 3.141592... (il y a Math.PI)
```

2.3.1 Assignation

La syntaxe de l'assignation est `[var] variable = expression`. Il y a aussi des formes abrégés où l'opérateur = est préfixées par un autre opérateur : au lieu de `variable = variable opérateur expression` la syntaxe est `variable opérateur= expression`.

- `+=` Ajoute et assigne
- `-=` Soustrait et assigne
- `*=` Multiplie et assigne
- `/=` Divise et assigne

Un cas particulier sont les opérateurs ++ et -- qu'additionnent ou soustraient 1 à une variable ; ils peuvent précéder ou suivre la variable, en conséquence l'opération sera exécutée avant ou après l'évaluation de la variable.

2 Techniquement, cela est connu comme surcharge (*overloading*).

Il y a aussi une utile structure syntactique qui permet l'assignation conditionnelle :

variable = (*condition*) ? *valeur_si_vrai* : *valeur_si_fausse*;

4. Exercices

Utilisez seulement l'opérateur de soustraction

- *Déplacer la variable A dans la variable B.*
- *Échangez les variables A et B.*
- *Additionnez les variables A et B.*

5. Questions

Quelle est la valeur après exécution de la variable a ?

```
a = 12          a = 12
a *= --a        a *= a--
```

2.3.2 Affectation par décomposition

L'affectation par décomposition (*destructuring* en anglais) est une expression pour extraire des données d'une matrice ou d'un objet en les insérant dans des variables.

2.3.2.1 Spread et rest syntaxe

La syntaxe `spread` (`...`) permet d'étendre une matrice, par exemple pour fournir des paramètres dans une fonction qui en accepte un nombre variable :

```
<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: Utilise de la syntaxe `spread` pour ajouter des classes à un objet DOM

La syntaxe de `rest`, `...`*`ArrayNom`*, crée la matrice *`ArrayNom`* à partir de plusieurs éléments; il permet à une fonction d'accepter un nombre indéfini d'arguments en forme de matrice.

```
<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: Utilisation de la syntaxe `rest`

2.3.2.2 Extraire des données d'une matrice

`[`*`variable`*$_1$`,` *`variable`*$_2$`, ...] = Array`

Soit la matrice : `primes = [2,3,5,7,11,13,17,19]`

Attribuer les 3 premières valeurs	`[two,three,five] = primes`	`2 3 5`
Omettre certaines valeurs	`[,,,seven,,thirteen] = primes`	`7 13`
Obtenir `eleven` et l'ensemble des derniers variable dans `rest`	`[,,,,eleven,...rest] = primes`	`11 [ 13, 17, 19 ]`
Obtenir la valeur par défaut	`[two,,,,,,,extra=97] = primes`	`2 97`

2.3.2.3 Extraire des données d'un objet

`{`*`variable`*$_1$`,` *`variable`*$_2$`, ...} = Object`

Soit l'objet : `var parms = {"width":-1,"height":-1,"top":-1,"left":-1,"padding":5}`

Obtenir des valeurs	`const {top, left} = parms`	`Top = -1 left = -1`
Attribution avec change du noms de la variables	`const {top:high, left} = parms`	`High = -1 left = -1`

Obtenir la valeur par défaut `const {margin=10, padding} = parms` `Margin = 10 padding = 5`

2.3.2.4 Extraire des données d'une matrice et d'un objet

```
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

2.4 Contrôles du programme

2.4.1 Instruction conditionnel

La structure complète de l'instruction conditionnel `if` est :

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

`else if` et `else` peuvent manquer, `else if` peut être répétée.

S'il y a un bloc d'instruction, il est mieux, pour raisons de lisibilité, d'indenter les instructions (voir l'exercice).

6. Exercice

Transformez le code suivant utilisant l'opérateur `?`.

```
nombre = Math.ceil(100*Math.random());    // nombre entier entre 1 et 100
if (nombre % 2 == 0) {
      res = " pair";
} else {
      res = " impair";
}
alert(nombre + res);
```

2.4.2 switch

L'instruction `switch` corresponde à une série d'`else if` :

```
switch (expression) {
            case valeur1[,valeur2]:
                  instructions
                  break;
            case valeur3:
                  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("Aujourd'hui " + d);
```

Script 5: Instruction `switch`

☞ `break` est nécessaire car quand une valeur satisfit, les cases suivantes seraient exécutés ; `default` est pour les cas non prévus.

2.4.3 Boucles

Les boucles permettent d'exécuter répétitivement un ensemble d'instructions ; on peut forcer la sortie de la boucle avec l'instruction `break`.

2.4.3.1 Itérations

La plus simple forme de boucle est la boucle `for` :

```
for (variable=expression1;condition de sortie; modification de la variable)
   Instruction(s)
```

L'exécution du `for` termine quand *`condition de sortie`* est satisfaite ou par l'instruction `break`.

```
numero = 2+Math.ceil(998*Math.random());  // integer number between 3 and 1000
signal = " premier";
for (i=3;i*i<=numero;i+=2)
      if (numero %  i == 0) {
            signal = " pas premier";
```

```
            break;
      }
alert(numero+signal);
```

Script 6: Teste si un nombre est premier (avec erreur)

7. Question
Quelle est l'erreur dans le script ci-dessus ?

2.4.3.2 Itérations sur matrices et objets

Pour accéder aux éléments d'une matrice on peut utiliser l'instruction `for` vue au paragraphe Errore: sorgente del riferimento non trovata, toutefois s'il y a des éléments qu'il n'ont pas la clé numérique ils ne seront pas parcourus, dans ce cas on peut utiliser une variante :

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

Il y a aussi une méthode des matrices qui permet d'exécuter une fonction sur tous éléments (à clé numérique) :

matrice`.forEach(`*fonction*`)`[3]

fonction a trois paramètres : *valeur*, *index*, et *matrice*.

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

Script 7: Exemple de forEach: somme des éléments d'une matrice

2.4.3.3 while

Les instructions de boucle suivantes sont plus puissantes du `for` :

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

La boucle est répétée jusqu'à quand *condition* est vérifiée. On peut sortir de la boucle avec l'instruction `break`.

```
while (a != b) {        // le plus grand commun diviseur est quand a=b
    if (a>b) a-= b;
      else b -=a;
}
```

Script 8: L'algorithme d'Euclide pour le plus grand commun diviseur

8. Question
Quelle est la différence entre `while ...` et `do... while` ?

3 Il y a un deuxième paramètre optionnel qui représente un objet.

3 Objets

JavaScript est un langage à objets c'est-à-dire les données ne sont pas simplement des pièces de mémoire mais ils ont des propriétés et des méthodes de traitement qui dépendent du type de donnée, par exemple un chaîne de caractères (*string*) a, entre autres, la propriété `length` et la méthode `substr`, en plus tous objets ont la méthode `toString`, qu'il donne une image lisible de l'objet.
Les objets, outre à méthodes et propriétés, peuvent contenir des autres objet, comme l'objet `Window` qu'il contient toute la structure de la page WEB et, en effet, les instructions de JavaScript sont contenues dans l'objet `window`.
Les objets natifs de JavaScript peuvent être crées implicitement ou explicitement, la création explicite est par l'opérateur `new`: `dt = new Date();`

3.1 Ensembles accessibles par clé (matrices et objets)

On a déjà vu les matrices, elles, sont des objets dont la propriété `length` donne le nombre des éléments accessibles par la position, en outre JavaScript accepte des matrices avec clés quelconques, dans ce cas la déclaration est différente car elle est une déclaration d'objet générique pendant que les matrices sont des objets particuliers (avec des méthodes à soi) :

nomObjet = {*cle*$_1$:*valeur*$_1$[,*cle*$_2$;*valeur*$_2$[, ...]}

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

3.1.1 Ajouter des éléments

`aPoids(aPoids.length) = 44;`	Ajoute un élément est une matrice après le dernier
`aPoids.push(44);`	Ajoute un élément après le dernier, par la méthode des matrices `push`
`aPoids.unshift(44);`	Ajoute un élément avant le premier
`aPoids["unitMisure"] = "Kilo";`	Ajoute un élément à clé non numérique, en fait, il ajoute une propriété

3.1.2 Accéder aux éléments des structures

La syntaxe pour accéder à un élément d'une matrice est :

nomMatrice[*position_ou_clé*]

par exemple `Jours[3]`, `JoursObj["mar"]`.
Pour les clés non numériques il y a une syntaxe alternative :

nomMatrice.position

par exemple `JoursObj.mar`.
arrayName`.indexOf(`*valeur*`)` peut être utilisé pour vérifier la présence d'un particulaire élément, s'il n'exist pas la méthode répond `-1`.

9. Question

- *Il y a deux cas dans lesquels la syntaxe alternative n'est pas applicable. Quels sont-ils ?*

3.1.3 Méthodes

Triage : *matrice*`.sort()` trie les éléments comme chaîne de caractères donc, par exemple, le triage de cette matrice : `[7,100,15,32]` donne `[100,15,32,7]`; avec *matrice*`.reverse()` on obtient un triage descendant.

Pour obtenir un triage numérique ou selon autres critères, on utilise une variante de la méthode sort : *matrice*`.sort(`*fonction*`)`, où *fonction* reçoit comme paramétrés deux éléments et retourne une valeur qui indique la position relative des deux éléments (voir la Figure 2, où `a` et `b` sont les deux valeurs). Par exemple pour un triage numérique la fonction contient `return a - b;`.

`< 0`	`a` doit précéder `b`
`= 0`	Laisser inchangée
`> 0`	`b` doit précéder `a`

Figure 2: Valeurs retournées pour déterminer le triage

Choix : *matrice*`.filter(`*fonction*`)` applique à chaque élément de la matrice *fonction* et retourne une matrice où sont présentes les éléments pour lesquels *fonction* retourne `true` ; le premier paramètre de la fonction est la valeur.

3.2 Chaînes (*Strings*)

L'objet `String` est le bâtisseur des ensembles de caractères (*string*), chaque string hérite de l'objet `String` la propriété `length` et les méthodes[4] comme `substring`, `toLowerCase`, `toUpperCase`, `replace`, `indexOf`,

4 En effet les méthodes appartiennent à la propriété `prototype` de l'objet `String`.

`charAt` etc. :

```
"Condor Informatique Turin".toLowerCase()
String.toLowerCase("Condor Informatique Turin")
```

☞ On peut accéder à un single position d'une chaîne comme une matrice, mais seulement en lecture, par la syntaxe *`string[index]`*.

Il y a aussi des méthodes (obsolètes!), pour ajouter des TAGs stylistiques (HTML), par exemple `"pig".big()` devient : `<big>pig</big>`.

Un des méthodes de l'objet `String` est `String.fromCharCode` qu'il engendre un caractère à partir de sa représentation numérique :

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

3.2.1 Méthodes avec expressions régulières

Dans les méthodes pour rechercher et remplacer sous-chaînes, celles-ci peuvent être chaînes ou expressions régulières (voir par. 9.1 Notes sur les expressions régulières) ; JavaScript reconnaît l'expression régulière parce qu'elle est renfermé entre `//`, (ou déclaré expressément comme un objet `RegExp`).

La méthode *`chaine`*`.search(`*`aiguille`*`)` donne la position du sujet de recherche :

```
fichier = "Maison.JPG";
alert(fichier+((fichier.search(/\.jpg$/i) == -1)?" n'est pas":" est")+" une image");
```

La méthode `replace` retourne une nouvelle chaîne avec des pièces remplacées :

`chaine``.replace(`*`aReplacer`*`,`*`replacement`*`)`

Le remplacement peut être une chaîne ou une fonction qui est appelée pour chaque remplacement, dans ce cas la valeur de retour de la fonction est la chaîne de remplacement.

☞ Normalement `replace` remplace la première pièce, il faut écrire expressément *`aReplacer`* avec les délimiteurs des expression réguliers, suivi du modificateurs **`g`** (*global*) : `"alfaabetaaa".replace(/aa+/g,"a")`.

Avec les expression réguliers on peut extraire des pièces de la chaîne car les pièces extraites, qu'ils sont celles entre parenthèses, ont les nom $1, $2, `...` :

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

Script 9: Exemples de la méthode replace

Si *`replacement`* est une fonction, elle peut accéder à une matrice où les pièces extraites commencent par la clé `1`.

10. Question

- *Quel est la chaîne de caractères résultat des deux méthodes `replace` de l'exemple dessus ?*

3.2.2 Transformer chaînes et matrices

Transformer une chaîne en matrice par la méthode split : *`chaine`*`.split(`*`séparateur`*`)`

```
Jours = "dimanche,lundi,mardi,mercredi,jeudi,vendredi,samedi".split(",");
```

Le *`séparateur`* peut être une expression régulière.

Transformer une matrice en chaîne par la méthode join : *`matrice`*`.join(`*`collant`*`)`

```
pointCard = ["Sud","Nord","Est","Ouest"];
document.write(pointCard.join(" - "));    // Sud - Nord - Est - Ouest
document.write(pointCard.toString());     // Sud,Nord,Est,Ouest
```

3.2.3 Conversion des numéros

Quand un numéro est dans une chaîne il doit être converti pour être correctement traité par l'opérateur +, avec les fonctions `parseFloat` et `parseInt`; la fonction `Number(`*`objecte`*`)` convers *`objecte`* dans un numéro ou `NAN` (Not A Number).

3.3 Fonctions

La fonction est un ensemble d'instructions pour exécuter une tâche et éventuellement obtenir une valeur ; un synonyme de fonction est routine, mais la diction de JavaScript est *méthode*.

Il y a des fonctions natives, sont les méthodes de l'objet `window` ou méthodes des objets tel que `Date` ou `Math` ; mais naturellement on peut écrire des fonctions nouvelles pour les besoins de notre application.

3.3.1 Création

Une fonction a normalement un nom, des opérandes, une (possible) valeur de retour et quelque façon de la créer :

1) `function nomFonction(operand1[,operand2[,...]]) {instruction(s)}`
2) `[var] nomFonction = function(operand1[,operand2[,...]]) {instruction(s)}`
3) `[var] nomFonction = new Function("operand1"[,"operand2"[,...]], "instruction(s)");`
4) `[var] [functionName =] (operand1[,operand2[,...]]) => {instruction(s)}`

☞ Dans la première façon (*Function Declaration*) la fonction est prise en charge au moment de l'analyse syntaxique du code et pourtant elle peut être utilisée avant au après sa déclaration ; ceci il n'est pas le cas des deux autres formes (*Function Expressions*) où la fonction est crée au moment de l'exécution du code.

☞ La quatrième forme de déclaration de fonction (fonction flèche) est concise et plus flexible, par exemple lorsque le corps n'a qu'une seule instruction, c'est la valeur renvoyée comme dans la fonction log10 (numéro 4)).

```
1) fonction somme (x, y) {return (x + y);}
2) var somme = function(x, y) {return(x + y);}
3) var somme = 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;
}
```

11. Questions

1. *Qu'est ce que calcule la fonction `x` ?*
2. *Quel doit être le `typeof` de `item` ?*

3.3.2 Utilisation

Pour utiliser une fonction la syntaxe est :

`[[var] variable =] maFonction(operande1[,operande2[,...]])`

`variable` contiendra le résultat qui *`maFonction`* a calculée en base aux valeurs d*`es opérandes`*. Il est possible d'utiliser le résultat d'une fonction comme un opérande dans une autre fonction.

Les variables déclarées comme `var` sont accessibles uniquement à l'intérieur de la fonction, sans var elles sont créés dans l'objet `window` et donc elles sont accessibles dans toutes les fonctions.

L'éventuel résultat calculé par la fonction est retourné par l'instruction `return`.

3.3.3 Aspects évolué

3.3.3.1 Fonctions anonymes

Une fonction est dite anonyme quand elle est écrite directement où il est prévu une fonction, par exemple le second paramètre de la méthode `replace` peut-être un ensemble de caractères ou une fonction qui peut être écrite directement :

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

12. Question

Qu'est ce que calcule la fonction anonyme ci-dessus ?

3.3.3.2 Nombre variable d'arguments et polymorphisme

La syntaxe des fonctions prévoit un nombre fixe d'opérandes, toutefois il est facile de créer des fonctions avec un nombre variable d'opérandes, il suffit, avant d'utiliser un possible opérande, contrôler s'il est défini, à l'aide de l'opérateur `typeof`, ceci permet aussi de créer des méthodes avec des opérandes facultatifs avec une valeur de défaut :

```
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 10: Nombre variable de paramètres et valeur par défaut

☞ On peut accéder aux opérandes par la pseudo matrices `arguments` (existante à l'intérieur de la fonctions), où `arguments.length` est le numéro des paramètres présents.

Le polymorphisme est une propriété des langages qui acceptent le même nom de fonction pour fonctions qui peuvent avoir un nombre et ou type différent de paramètres ; en JavaScript le polymorphisme doit être résolu à l'intérieure de la fonction, avec l'inspection des paramètres. Un exemple de polymorphisme est la méthode *`chaine`*`.replace` où le deuxième paramètre peut être une ensemble de caractères ou une fonction.

3.3.3.3 Création de structures et préservation des valeurs

Les fonctions sont des objets qui peuvent contenir des propriétés ; elles sont créées avec la syntaxe *`nomObjet`* `= new` *`nomFonction`* et dans la fonction les propriétés sont créées par `this.`*`nomDeProprieté`*. Les propriétés peuvent être insérées aussi après la création de l'objet.

Script 11: Les objets fonction

```
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"] = "Mâle"  // ajoute proprieté
alert(staff["Doe"].age+staff["Doe"].Sex)
```

JavaScript n'as pas une déclaration de variable, comme `static` en C ou PHP, pour faire survivre une valeur dans la fonction, pour qu'il soit accessible dans les appels suivants, on lui peut ajouter une propriété :

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

Script 12: Création d'une variable permanente dans une fonction

3.3.3.4 Encapsulation (*JavaScript module pattern*)

Tous objets de JavaScript sont contenu dans l'espace `window`[5], d'ici la nécessite, surtout si on utilise du code externe, d'éviter des collisions de noms. Avec JavaScript une méthode est de créer un objet qui contient propriétés et méthodes voulus, mais, touts éléments de l'objet sont accessibles ; une solution qui permet de créer des composants complexes, utilise une déclaration particulier des fonctions de JavaScript (la n. 2 du paragraphe 3.3.1 Création) : la fonction au lieu d'être simplement déclaré est immédiatement exécute et retourne une matrice de méthodes et propriétés qui on veut rendre publiques ; la structure est la suivante :

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

5 Naturellement pour JavaScript exécuté par le navigateur.

```
        return {publics}
        }([operands]));
```

Notez la syntaxe pour définir (`set`) et obtenir (`get`) la propriété.

"`()`" à la fin de la déclaration permet son exécution immédiate et la fonction, avec les *`méthodes et propriétés publiques`* retournées, est ajoutée à l'espace `window` (*Anonymous Closure*) ; ces dernières sont accessibles avec la syntaxe :

`nomFonction.nomMéthode(paramétres)`

`nomFonction.nomPropriété`

Les parenthèses externes `()` sont nécessaires par compatibilité avec tous navigateurs ; les `opérandes` éventuels sont des objets , de cette façon la fonction peut accéder à leurs propriétés et méthodes.

```
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 13: Fonctions pour traiter une chaîne de mots comme une matrice

13. Exercice

- *Ajoutez une méthode qui retourne la position d'un mot dans la chaîne.*

3.3.3.5 Ajouter des méthodes à un objet

On peut ajouter des méthodes et propriétés à un objet, même aux objets natives. La méthode ou la propriété peut être relative aux instances de l'objet ou à l'objet même ; dans le premier cas la méthode agisse sur un objet, dans le second cas crée un objet : par exemple l'objet `String` a la propriété `prototype` qui est l'objet contenant les méthodes pour agir sur les chaînes et la méthode `fromCharCode` pour créer des chaînes de caractères.

La syntaxe est :

`objet``.prototype.`*`nouveauMethode`* `= function(`*`opérandes`*`) {`*`instructions`*`};`

`objet.nouveauMethode` `= function(`*`opérandes`*`) {`*`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
    if (typeof n == "undefined") var n = 1;
      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 14: Méthodes ajoutées (rotation de mots et remplissage)

14. Exercice

- *Ajoutez une méthode à l'objet de l'exercice 13 pour changer un mot avec un autre.*

3.3.4 Fonctions natives

3.3.4.1 Manager les temps

Les deux fonctions `setInterval` et `setTimeout` permettent d'indiquer au navigateur d'exécuter une fonction JavaScript après un certain temps ; elles ont deux arguments : la fonction à exécuter, et le temps d'attente en millisecondes ; les fonctions retournent un `handle`, qu'il sert surtout pour terminer, si le cas, la gestion du timer :

```
handle = setTimeout(fonction,millisec)    // exécute fonction après millisec.
handle = setInterval(fonction,millisec)   // exécute fonction toutes les millisec.
```

La *`fonction`* indiquée dans `setTimeout` est exécute une seule fois, tandis que celle de `setInterval` est exécuté indéfiniment ; on peut interrompre la temporisation avec l'aide de `clearTimeout(handle)` et `clearInterval(handle)` respectivement.

```
...
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: Exemple utilisation du timer et onSubmit

3.4 Date

L'objet `Date` gère les dates à partir du 1 janvier 1970 ; il y a plusieurs façons de créer une date :

```
d = new Date();                  // d contient la date du jour
d = new Date(milliseconds);      // 0 est 1 janvier 1970
d = new Date(dateString);        // ex. d=new Date("July 21, 1983
01:15:00");
d = new Date(année, mois, jour, heures, minutes, seconds, millisecondes);
```

☞ Seul année est nécessaire dans le dernier exemple.

L'Objet `Date` a plusieurs méthodes, soit pour prélever soit pour changer ses composants ; ici quelques-uns :

`getDate(), setDate(jour)`	Retourne ou modifie le jour du mois (de `1-31`)
`getFullYear(), setFullYear(année)`	Retourne ou modifie l'année (quatre chiffres)
`getHours(), setHours(heure)`	Retourne ou modifie l'heure (de `0-23`)
`getMilliseconds(), setMilliseconds(millis)`	Retourne ou modifie les millisecondes (de `0-999`)
`getMinutes(), setMinutes(minutes)`	Retourne ou modifie les minutes (de `0-59`)
`getMonth(), setMonth(mois)`	Retourne ou modifie le mois (de `0-11`)
`cetSeconds(), setSeconds(seconds)`	Retourne ou modifie les seconds (de `0-59`)

☞ *`mois`* est une valeur entre `0` et `11` et *`jour`* entre `1` et `31` ; `setDate(jour)` accepte jours au dehors du intervalle `1-31`, et modifie la date de conséquence.

15. Exercices.

- *Prenez la date du jour et modifiez le jour par -1 et 32.*
- *Vérifiez s'il est possible donner des valeurs au dehors du intervalle pour heures, minutes et seconds.*

3.5 Math

L'objet `Math` contient des fonctions arithmétiques, trigonométriques, logarithmiques et quelques constantes comme π (`Math.PI`) et `e` (`Math.E`).

Dans l'exemple suivant il sont utilisées :

- `Math.random` qui donne un nombre réel casuel entre `0` (inclus) et `1` (exclus),
- `Math.ceil` pour obtenir le nombre entier supérieur plus proche d'un nombre réel,
- `Math.pow(`*`base`*`,`*`exposant`*`)`.

```
function testKhi(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;        // Probabilité d'erreur < 0,05 maximum 11,1
}
```

Script 15: Simulation du rouleau de dés et test du khi-carré

3.6 Introspection

Introspection est la capacité de examiner le type ou les propriétés d'un objet à run-time, in JavaScript il y a l'opérateur `typeof` avec le réponses possibles : `number`, `string`, `boolean`, `object`, `null` et `undefined`; pour les matrices il y a la méthode `Array.isArray(`*`obj`*`)`.

L' opérateur `instanceof` est utile pour contrôler le type d'une objet :

```
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 4 Gestion des erreurs

Si on a une instruction qui peut donner une erreur mais qui ne doit pas arrêter le programme, par exemple une réception possible de données,on l'insère un bloc `try` et après un bloc `catch` où l'erreur est traitée, voir l'exemple ci-dessous :

```
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");
```

Le script ci-dessus signale :

```
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
```

Si on dé-comment l'instruction `throw`, les résultats sont les suivants ;

```
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!
```

L'instruction `finally` est toujours exécutée; les instructions après `throw` ne seront pas exécutées.

5 Gérer les événements asynchrones

Le développement d'applications interactives nécessite l'écriture de code asynchrone non bloquant qui, en JavaScript, peut être effectué par l'utilisation de *CallBack* fonctions et avec les dernières fonctionnalités ajoutées basées sur le concept de promesse (*promise*).
La nécessité de traiter avec un mécanisme asynchrone est pour l'interaction de l'application avec des services externes comme pour exemple demander des données à une base de données distante ou pour obtenir une page Web.

5.1 Fonctions CallBack

La fonction *CallBack* traite un événement, c'est-à-dire qu'un certain code doit être exécuté lorsqu'un événement se produit; dans la Figure 3: Exemple utilisation du timer et onSubmit , il y a un exemple de fonction *CallBack*. Ci-dessous un fragment d'un site web (exécuté par 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") {
            ...
```

Comme on peut voir les *CallBacks* sont «imbriqués» ce qui ne facilite pas la compréhension du programme.

5.2 5.2 Promesse

Au lieu d'utiliser des fonctions qui acceptent un *CallBack*, on crée une fonction qui renvoie un objet `Promise`, c'est-à-dire un objet représentant une valeur qui sera disponible à l'avenir.
La promesse est construite autour d'une fonction qui gère une requête asynchrone; le constructeur de promesse est une fonction à deux paramètres, respectivement une fonction de résolution (***resolve***) et une fonction de rejet (***reject***), qui, dans le corps de la fonction, apparaît en fonction du résultat de la requête.
La `Promise` peut être dans l'un des trois états: l'état initial (*pending*), l'état réussi (*fulfilled*) ou l'état échoué (*rejected*).
Dans les deux derniers cas, la promesse a été honorée et le programme peut gérer la réponse par la méthode `Promise` qui a deux paramètres respectivement la fonction ***resolve*** et la fonction ***reject***.
Dans l'exemple ci-dessous, un temporisateur agit comme une fonction asynchrone avec un autre temporisateur, avec intervalle aléatoire, qu'il est utilisé pour simuler un ***reject*** effaçant le premier temporisateur.

```
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 16: Promise exemple: timer éventuellement effacée

5.3 Async et Await

Les programmes asynchrones utilisant *Promise* ont une structure séquentielle qui permet une meilleure compréhension du code. Une alternative qui a une simplicité d'exécution séquentielle sans bloquer d'autres tâches, est donnée par des fonctions déclarées asynchrones qui contiennent l'attente de(s) tâche(s) asynchrone(s), c'est-à-dire une résolution de la *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 17: Async await exemple: premier timer éventuellement effacée

Dans la fonction asynchrone, il est possible d'attendre un ensemble d'événements en attendant une **`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 18: Async await exemple: Promise.all attend plusieurs événements

Une exécution possible du script ci-dessus est comme ceci :

```
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)

Le **Document Object Model** (ou **DOM**) est un standard qui décrit une interface indépendante de tout langage de programmation et plate-forme, permettant à des programmes d'accéder et, éventuellement, de modifier le contenu, la structure et ou le style des documents XML et HTML.

DOM représente le document et ses éléments, comme une structure à arbre ; dans le navigateur cette structure est contenue dans l'objet `window` (qui coïncide avec l'objet `global`), voir la figure ci-dessous.

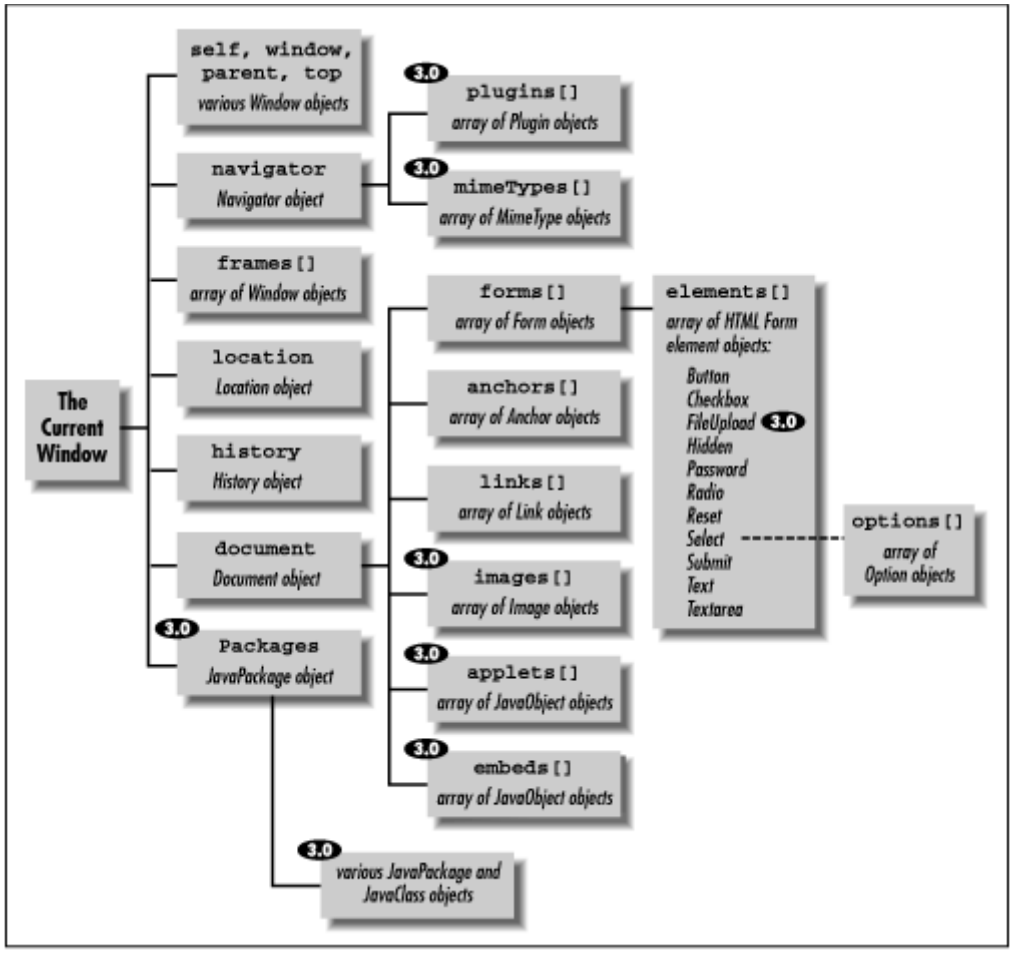


Figure 4: L'objet `window`

Les crochets qui suivent certains objets indiquent qu'il y a une matrice (*array*) d'objets.

Le nom de chaque objet est préfacé par les noms de tous les objets qui le contiennent, à partir de la racine, séparés par des points. Par exemple, l'objet `window` contient une matrice d'objets `form`ulaires qui peuvent contenir des objets de saisie, des boutons, etc :

```
window.document.forms[0].elements[3]  // premier formulaire quatrième objet
```

Tous les noms d'objets commencent par le nom de l'objet racine, c'est-à-dire `window`, mais JavaScript permet de l'omettre :

```
document.forms[0].elements[3]          // premier formulaire quatrième objet
```

Propriétés, méthodes et objets de `window` sont globales et accessibles dans tout code JavaScript.

6.1 Méthodes de l'objet window

6.1.1 Interaction avec l'utilisateur

- `alert(`*`message`*`)` signal à l'utilisateur
- `confirm(`*`message`*`)` requête de confirmer (réponse `true`) ou annuler
- `prompt(`*`message,défautTexte`*`)` affiche une boîte de dialogue pour insérer des données

6.1.2 Interaction avec le navigateur

Il y a plusieurs commandes pour créer et gérer une fenêtre, toutefois la création peut être bloqué par choix de l'utilisateur.

- *window*.open(*URL*,*name*,*specs*,*replace*) ouvre une page (et crée un nouveau objet window)
- *window*.close() ferme la fenêtre
- *window*.document.write(*données*) écrit dans la fenêtre
- *window*.resizeTo(*largeur*,*hauteur*) dimensionne la fenêtre
- *window*.moveBy(*x*,*y*),moveTo(*x*,*y*) déplacement de la fenêtre relative à sa position ou absolut

☞ La propriété opener de l'objet document permet d'accéder à la fenêtre mère.

```
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 19: Création d'une fenêtre

☞ Au lieu de ces commandes, qui posent des problèmes sur différents navigateurs, il est mieux utiliser la création d'une « fenêtre » par tags de HTML (comme) qui peuvent être rendues visibles (et cachées).

6.2 L'objet document

Quand un document HTML est chargé dans le navigateur, il devient l'objet document ; l'objet document est le nœud racine du document HTML et l'ancêtre de tous les autres nœuds ; avec JavaScript on peut accéder à les propriétés et à les méthodes de tous les nœuds.

6.2.1 Accéder aux éléments

L'objet document contient le nœud body et des collections des nœuds (éléments) comme forms (*formulaires*) et images (tous celui de la 4 où il y a les crochets), toutefois tous nœuds sont accessible par des méthodes :

- *element*.getElementsByTagName(*tagType*) crée une matrice d'objets ; si *tagType* est * la matrice contiendra tous les nœuds du document.
  ☞ *element* peut etre de document à nœud, pour exemple a form nœud.
- document.getElementById(*id*) accède a l'élément par son id, ☞ naturellement *id* doit être unique.
- document.getElementsByClassName(*nomDeClasse*) fournit une matrice d'éléments d'une class voulue,
- document.getElementsByName(*nom*) collection d'objet des formulaires qu'ils ont le même *nom*.

☞ Il y a des propriétés avec ...Element... et ...Element**s**... ceci signifie que la valeur obtenue est, dans un cas un objet, dans l'autre une matrice, en particulier l'attribut *id* doit être univoque dans la page (mais il n'est pas contrôlé), au contraire le même *nom* peut être dans plusieurs formulaires.

La propriété parentNode d'un élément, référence son parent ; la propriété this dans le contexte de la gestion des événements d'un formulaire référence l'objet qui a engendré l'événement et this.form est le formulaire même.

```
var tags = document.getElementsByTagName("*");        // tous les 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 != "")? "Id repété\n"+str:"Pas id multiples")
```

Script 20: Contrôle pour id multiples

6.2.2 Ajouter un nœud

On doit créer le nœud et l'éventuel texte contenu, ou « cloner » un nœud présent, par les méthodes :

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

- *element*.cloneNode(*boolean*) si *boolean* est true ils sont copiés les éventuels nœuds fils.

On insére le nœud au point voulu par les méthodes :

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

parentNode peut être un nœud qui peut avoir des descendants comme body, form, etc.

parentNode et *node* peuvent être :

- un nœud individué par son *id*,
- un nœud individué par sa position relative (*voir Figure 5: Position relative au nœud (me)*)
- un nœud appartenant à une collection.

Aussi *place* est un nœud (pas body évidemment).

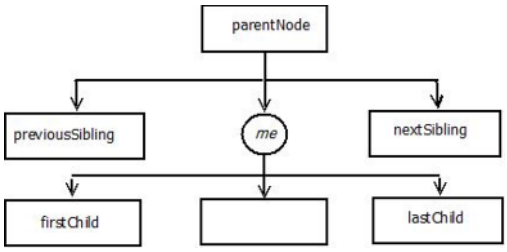


Figure 5: Position relative au nœud (me)

☞ Ces instructions doivent être exécutées après que le document a été bâti, voir l'exemple ici dessous.

```
...
<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 21: Ajouter un nœud dans le body et après un élément

☞ Pour certaines TAGs, comme TABLE et listes il y a des méthodes ad hoc.

6.2.3 Déplacer ou supprimer un nœud

Les méthodes *parentNoeud*.appendChild et *parentNoeud*.insertBefore appliquées à un nœud existant, ont l'effet de le déplacer :

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

Pour supprimer des nœuds on doit connaître le parent de l'élément et utiliser la méthode :

parentNoeud.removeChild(*nœud*)

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

Figure 6: Effacement d'un nœud

6.2.4 Modifier un nœud

6.2.4.1 Accéder aux propriétés

Le nœud peut avoir des attributs (celui qui sont contenus dans le TAG) et des propriétés. Propriétés et attributs peuvent être confondus, mais en réalité ils sont deux choses différentes bien que, en général, les propriétés sont synchronisées avec les attributs.

	Nom attribut	Attribut	Propriété
Input type text	`value`	La valeur initial	La valeur actuel
Input type check box	`checked`	Chaîne vide	`true` ou `false`
Ancre (A TAG)	`href`	La valeur initial	L'URL complète

Figure 7: Quelques différences entre attributs et propriétés

Autres différences sont :

- les noms des attributs sont insensibles à la casse,
- l'attribut `class` corresponde à la propriété `className`,
- dans les tags on peut insérer des attributs « propriétaires », qui ne deviennent pas des propriétés du nœud,
- avec JavaScript on peut ajouter des propriétés au nœud, qu'ils ne sont pas des attributs.

L'accès aux attributs HTML d'un nœud est possible à l'aide des méthodes suivants :

- *nœud*`.hasAttribute(`*nom*`)` contrôle s'il y a l'attribut
- *nœud*`.getAttribute(`*nom*`)` prélève l'attribut
- *nœud*`.setAttribute(`*nom*`,`*valeur*`)` modifie l'attribut
- *nœud*`.removeAttribute(`*nom*`)` élimine l'attribut

16. Exercice

préparez une page HTML avec un formulaire pour évaluer attributs et propriétés

6.2.4.2 Les propriétés innerHTML, innerText et textContent

Ces propriétés permettent de lire ou modifier le contenu d'un TAG : `innerHTML` insère une structure HTML qui devient immédiatement opérative (sauf éventuels scripts pour raisons de sécurité), `innerText` (IE) et `textContent` (tous navigateurs) insèrent un contenu textuel.

Avec `innerHTML` on peut créer une structure, même complexe, plus aisément que par les méthodes vues au paragraphe 6.2.2, ceux-ci sont adaptés pour une utilisation interactive tandis que `innerHTML`, en général, est utilisé par ajouter un contenu venant du serveur.

```
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 22: innerHTML et horloge brut

6.2.4.3 Changer le style d'un nœud

L'objet style d'un élément contient les propriétés attribuées à un élément, le nom de la propriété coïncide, normalement, avec le nom équivalent CSS, sauf pour les noms avec le tiret qui sont traduits selon la règle dite *CamelCase* : le tiret est supprimé et la lettre suivante est changée en majuscule.

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

ex. `elem.setAttribute("style","width: 500px; background-color: yellow;");`

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

Script 23: Changer le nom du style CSS en nom JavaScript (Camel function)

element`.setAttribute(`*nomAttribute*`,`*valeur*`)` peut être utilisée pour définir l'attribut `style`.

valeur doit être une chaîne d'attributs avec la syntaxe CSS :

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

car `setAttribute` remplace les styles existants les éventuelles modifications de la propriété `style` doivent être exécutées après `setAttribute`.

6.2.5 Évents

L'objet `document` peut associer du code JavaScript à des événements qui ont lieu sur un de ces composants, le code

sera exécuté seulement quand l'évent aura lieu ; le mot `this` est la référence à l'objet où l'événement a eu lieu (voire 8). La capture d'un évent engendre l'objet `event` dont les principales propriétés sont :

- `type` le nom de l'évent,
- `target` l'objet où il y a eu l'évent (FireFox),
- `srcElement` l'objet où il y a eu l'évent (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>
```

Dans les tags de HTML les noms des événements ne sont pas sensible à la casse, mais il n'est pas le cas de JavaScript.

☞ La gestion de l'événement est différent dans IE et Firefox. Dans IE, il y a l'objet `window.event`, alors que dans Firefox et d'autres W3C navigateurs conformes, l'événement est le premier paramètre de la fonction associée à la gestionnaire d'événements ; en outre il y a des différences aussi dans certes propriétés.

Voir dessous comme écrire une fonction « *cross browser* ».

```
document.getElementById("div1").onclick = function(evt){
    var event=window.event || evt;
    var target=event.srcElement || event.target;     // srcElement IE, target Firefox
    alert(event.type+" "+target+" "+target.id)
};
```

Évents de la souris

`onclick`, `ondblclick` quand on pousse le bouton une ou deux fois,
`onmousedown`, `onmouseup` capture le début et la fin de l'action pousse bouton,
`onmousemove` quand la souris passe sur un objet,
`onmouseover`, `onmouseout` quand la souris entre et sort d'un objet.

```
<script language="Javascript" type="text/javascript">
essai = {
 pi : 3.141592653589,
 perimetre: function(r) {return 2*this.pi*r;}
}
</script>
...
P&eacute;rim&egrave;tre et Pi grec
<br><input type="button" value="Pi grec" onClick='alert(essai["pi"])' />
<br><input type="button" value="P&eacute;rim&egrave;tre"
      onClick='alert(essai.perimetre(10))' />
<img src="images/condor.gif"
                      onMouseOver='this.src="images/logopace.gif"'
      onMouseOut='this.src="images/condor.gif"' />
```

Figure 8 : Événements onClick, onMauseOver, onMauseOut et l'objet `this`

Évents du clavier

`onkeydown`, `onkeypress`, `onkeyup`

Évents des objets

`onload` quand un objet (image, page) a été chargé sur le navigateur.
`onerror` quand un objet il n'a pas pu être chargé, par exemple une image incorrecte.
`onabort`, `onresize`, `onscroll`, `onunload`

Évents des formulaires

`onblur` quand un élément perd le focus.
`onchange` l'évent se produit quand un élément a changé, par exemple une sélection d'une liste déroulante, un change sur un check box, etc.
`onreset` Quand le bouton `reset` a été cliqué.
`onsubmit` Avant que le formulaire soit envoyé au serveur.

7 Ajax

Ajax (*Asynchronous JavaScript and XML*), se base sur l'objet `XMLHttpRequest` qu'il permet un moyen facile de récupérer des données à partir d'une URL[6] pour mettre à jour seulement une partie de la page.

7.1 Propriétés et méthodes

L'objet a des méthodes :

- `open(`*`paramétrés`*`)` détermine URL, mode et type de communication, en particulier les paramètres sont :
  - *`méthode`* `GET` ou `POST` pour envoyer et recevoir des données au WEB serveur[7],
  - *`URL`* le script serveur qui élabore la requête, si la méthode est `GET` il peut être suivi des données dans le format `?`*`nom=valeur`*`[&`*`nom=valeur`*`[...]]`
  - *`asyncFlag`* optionnel : `true`, le défaut, indique que la requête est asynchrone, dans ce cas on doit indiquer une fonction qui gérera la réponse.
  - *`userName`* *et* *`password`* à indiquer seulement si sont nécessaires pour accéder au site.
- `send(`*`content`*`)` Transmit la requête, content est présente seulement si la méthode est `POST`, il peut être une simple chaîne de caractères avec syntaxe comme dans la méthode `GET` (sans `?`), ou an objet par exemple `FormData` qui permet d'envoyer le contenu d'un formulaire.

et propriétés :

- `onreadystatechange = `*`fonction`* est une propriété évent, *`fonction`* est la fonction qui gère le changement d'état de la requête, elle est nécessaire si *`asyncFlag`*`=true`,
- `readyState` propriété qui indique l'état de la requête, la valeur 4 indique la fin,
- `status` est le résultat de la requête `404` pour "Not Found" ou 200 pour "OK",
- `statusText` description du résultat,
- `responseText` les données.

7.2 Utilisation

Utilisation synchrone (déconseillée)	Utilisation asynchrone
Création de l'objet `XMLHttpRequest`	
	Activation de la propriété `onreadystatechange`
Début de la communication avec la méthode `open`	
	Éventuel envoi de header(s) par la méthode `setRequestHeader`
Envoi des donnes par la méthode `send`	
Collecte de données par la propriété `responseText`	

7.2.1 Utilisation de la méthode HTTP `GET`

Les donnés éventuels sont la part de l'URL nommée *query string ;* ils ont la forme :

`champ=valeur[&champ=valeur[…]]`

et ils sont séparés de l'adresse par le caractère **?,** par exemple :

`www.sitenom/scriptnom.php?poids=76&nom=El+Condor`

Dans la *query string* sont acceptées seulement les caractères alphanumériques et `-` `_` `.` `~` ; tous les autres caractères doivent être codés avec `%`*`nn`*, où *nn* est la codifie hexadécimale du caractère ; l'espace est `%20`, il peut être remplacé aussi par +, toutefois il est mieux utiliser la fonction `encodeURIComponent`.

La méthode `send` doit être utilisée avec paramètre `null` ; dans l'exemple dessous il y a une requête d'un fichier et l'envoie d'une expression à évaluer :

```
function getData(url) {
    AJAX=new XMLHttpRequest();
    AJAX.open("GET", url, false);              // false = requête synchrone
```

6 URL (*Uniform Resource Locator*) ou, informellement, **adresse web**.

7 Il y a aussi des autres méthodes.

```
    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 24: Ajax : la méthode GET et communication synchrone

7.2.2 Utilisation de la méthode HTTP `POST`

Avec la méthode `POST` les données à envoyer sont indiqué dans la méthode `send` (`null` si dans `open` la méthode est `GET`) ; les données peuvent avoir différentes formes : ensemble de caractères dans le format *query string* ou certains objets entre les quels a particulière importance l'objet `FormData` ; dans le premier cas, avant la méthode `send`, il faut indiquer le type de données qui doivent être envoyés, par la méthode `setRequestHeader`.
L'objet `FormData` permets d'engendrer un ensemble de couples clé-valeur ; les données sont transmises dans le même format utilisé par les formulaires quand on pousse le bouton de soumission ; des couples clé-valeur peuvent être ajouté à l'objet par la méthode `append`.

☞ `FormData` engendré d'un formulaire, il ne contient pas les valeurs de boutons radio, case à cocher et listes déroulantes si ces objets n'ont pas été choisis.

```
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 25: Ajax : exemple d'envoi de caractères ou de formulaire avec la méthode POST

8 Graphiques

HTML5 introduit l'élément de balisage de toile `<canvas ...>`, qui est un conteneur utilisé pour réaliser des graphiques à l'aide de JavaScript.

Le tag `canvas` veut essentiellement les attributs `width` et `height`, qui donnent la dimension en pixel de l'objet graphique ; l'attribut `id` est utile pour l'individuer.

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

L'objet DOM `canvas` dispose (indirectement) de plusieurs méthode pour créer et manipuler lignes, rectangles, arcs, textes et images ; en fait, ces méthodes appartiennent à un objet HTML5 accessible via la méthode `getContext` :

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

☞ Au début l'origine des coordonnées, le point `(0,0)`, est dans le coin supérieur gauche ; les coordonnées de l'axe `y` croissent vers le bas de l'écran.

La surface du dessin peut être limité par la méthode `clip()` qui coupe une région de n'importe quelle forme et grandeur de l'original.

8.1 Couleurs

Les méthodes `fillStyle`, `strokeStyle` et `shadowColor` définissent les couleurs utilisée respectivement pour le remplissage, les traits et les sombres. Les valeurs de *`couleur`* sont les mêmes que dans CSS:

Valeur hexadécimal	`#rrggbb` ou `#rgb`	`fillStyle = '#800000'  // rouge` `strokeStyle = '#0f0'  // vert`
Nom du couleur	16 mots-clés prises de la palette VGA Windows : **aqua**, **black**, **blue**, **fuchsia**, **gray**, **green**, **lime**, **maroon**, **navy**, **olive**, **purple**, **red**, **silver**, **teal**, **white**, et **yellow**.	`shadowColor = 'silver'` `strokeStyle = 'fuchsia'`
fonctions	`rgb(`*`rouge`*`,`*`vert`*`,`*`bleu`*`)` ou `rgba(`*`rouge`*`,`*`vert`*`,`*`bleu`*`,`*`transparence`*`)`	`fillStyle = 'rgb(0,0,255)' // bleu`

Figure 9: Couleurs

☞ *`couleur`* est un ensemble de caractères ou un objet gradient (voire parag. 8.6.3 Gradients).

`rgba(`*`rouge`*`,`*`vert`*`,`*`bleu`*`,`*`transparence`*`)` les valeurs des couleurs sont des nombres entiers entre 0 et 255 inclus, la valeur de *`transparence`* varie de 0 (transparent) à 1 (opaque).

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

Script 26: Remplissage couleur de fond du canevas

8.2 Textes

Il y a deux méthodes pour dessiner du texte : `fillText(`*`text`*`,`*`x`*`,`*`y`*`)` et `strokeText(`*`text`*`,`*`x`*`,`*`y`*`)`. Le premier dessine la forme de texte remplie, le deuxième dessine le contour. Chaque méthode prévoit aussi un quatrième argument optionnel pour la largeur maximale.

La propriété `font` spécifie la police du texte selon la syntaxe de la propriété CSS *font-family*.

Les propriétés `textAlign` et `textBaseline` servent pour aligner le texte, par exemple :

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

dessinent le texte centré sur les coordonnées données.

```
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 27: Canevas: Textes

La méthode `ctx.measureText(text)` expose la propriété `width` de *`text`*:

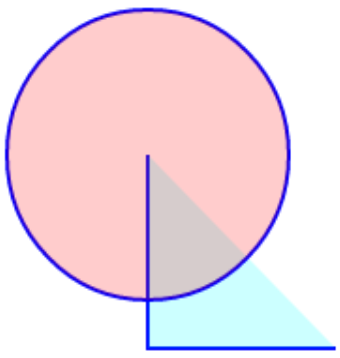
```
var textWidth = ctx.measureText(title).width
```

8.3 Lignes et formes

Le dessin de lignes commence par la méthode `beginPath()` et se termine par la méthode `closePath()`, les méthodes `stroke` et `fill` dessinent sur le canvas. Le départ du dessin est individué par la méthode `moveTo(x,y)` ou implicitement par la méthode `arc` ; la fin de la ligne devient le point de début de l'éventuelle ligne successive.

- `lineWidth(dimension)` *`dimension`* est la dimension des lignes,
- `setLineDash(pattern)` *`pattern`* est une matrice qui spécifies la longueur de l'alternance entre ligne et vide ex. `setLineDash([2,2])` ; une matrice vide restaure la ligne continue,
- `fill()` remplisse une forme qui éventuellement ferme,
- `stroke()` dessein,
- **`arc`**`(x,y,radius,debutAngle,finAngle,direction)` prépare un arc ; les angles sont en radiants, si *`finAngle - debutAngle`* est égal à **`2Π (2*Math.PI)`** on obtient un cercle ; le dessin de l'arc est selon le mouvement des lancettes de la montre si ***`direction`*** est `false`.
- `lineTo(x,y)` dessine un ligne jusqu'à la coordonné *`x,y`*.
- **`fillRect`**`(x,y,width,height)` dessine et emplisse un rectangle
- **`strokeRect`**`(x,y,width,height)` dessine un rectangle
- **`clearRect`**`(x,y,width,height)` efface une surface rectangulaire
- `quadraticCurveTo(cp1x,cp1y,x,y)` et `bezierCurveTo(cp1x,cp1y,cp2x,cp2y,x,y)` dessinent une courbe qui relie le point de début jusqu'au point (*x, y,*) selon un ou deux points de contrôle (dit intuitivement des points d'attraction).

```
function essayFormes(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='Formes'
onClick='essayFormes("canv","rgba(0,255,255,0.2)")'/>
```



Script 28: Canevas : triangle et cercle

☞ L'omission de la méthode `closePath()` a évité le dessin du bord diagonal du triangle, mais la méthode `fill()` considère la forme comme fermée.

8.4 Images

Les images en format PNG, GIF ou JPEG peuvent être utilisées dans le canevas, par la méthode `drawImage()` qui a trois variantes (***`dx`***, ***`dy`*** sont les coordonnée du point d'insertion de l'image).

`drawImage(image,dx,dy)`	l'image est insérée inchangée
`drawImage(image,dx,dy,dw,dh)`	l'image est insérée modifié
`drawImage(image,sx,sy,sw,sh,dx,dy,dw,dh)`	l'image ou une partie est insérée et modifié

Le premier paramètre est une image : un TAG `canvas` ou `img`, qu'ils peuvent être accédé par son `ID`, ou un objet `Image`, dont la propriété `src` peut être aussi une image sur le serveur et, dans ce cas, il faut attendre la fin de son chargement (voir l'exemple).

```
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 29: Les trois formes de drawImage

8.4.1 Sauver l'image

La méthode `toDataURL` rend le contenu du canevas comme une image (codifiée en base64) qu'on peut utiliser comme source d'une autre `canvas` ou `img` tag ou être envoyée au web serveur. La valeur de défaut de *`imageType`* est `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 30: Upload (synchrones) de l'image du canevas en format jpeg

8.5 Transformation géométriques

L'espace du dessin est contrôlé par une matrice de transformation `3x3` qui au début est unitaire (matrice à gauche dans la Figure 10), le changement des valeurs dans les premières deux lignes de la matrice permettent de modifier les desseins successifs :

- **a** modifie la taille du dessin horizontalement
- **b** incline le dessin horizontalement, la valeur est la tangente de l'angle
- **c** incline le dessin verticalement, la valeur est la tangente de l'angle
- **d** modifie la taille du dessin verticalement
- **e** déplace le dessin horizontalement
- **f** déplace le dessin verticalement

1	0	0
0	1	0
0	0	1

a	c	e
b	d	f
0	0	1

Figure 10: Matrice de transformation

Il y a des méthodes, qui modifient la matrice, pour déplacer, tourner et changer la taille ; la méthode `transform` et `setTransform` modifient la matrice ce qui permet, avec une seule instruction, à effectuer plusieurs transformations ; les deux méthodes ont six paramètres :

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

`setTransform` crée une nouvelle matrice, `transform` modifie la matrice existante en la multipliant par les paramètres.

☞ Toutes transformations affectent seulement les dessins crée après les méthodes de transformation.

8.5.1 Sauvegarde et restauration

Les méthodes `save()` et `restore()` sont utiles dans la création de graphiques complexes ; `save()` mémorise l'état actuel (matrice de transformations, couleurs, etc), et `restore()` récupéré l'état sauvé.

8.5.2 Déplacement de l'origine

La méthode `translate(x,y)` déplace la position de l'origine, c'est-à-dire les coordonnées du coin supérieur gauche deviennent (-*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 31: Canevas: déplacement de l'origine

8.5.3 Modification de l'échelle

La méthode `scale(w,h)` modifie le ratio du graphique ; des valeurs négatifs obtiennent un changement de direction des 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 32: Canevas: ellipse et effet miroir

8.5.4 Rotation

La méthode `rotate(angle)` exécute une rotation du graphique, *`angle`* est en radiants. Cette méthode correspond à une transformation qui agisse sur l'échelle et l'inclinaison contemporainement.

8.5.5 Transformation complexes

La méthode `setTransform()` peut effectuer translation, rotation et changement d'échelle contemporainement ; la méthode est nécessaire si on veut dessiner, dans un espace non orthogonal :

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

la transformation ci-dessus dilate l'axe x, incline l'axe y de 45 dégrées, les coordonnées de l'axe `y` croissent vers le haut de l'écran et le point (0,0) est déplacé.

8.5.6 Travailler avec les classique coordonnés cartésiennes

Par les transformations géométriques on déplace l'origine dans le centre du canvas et les coordonnées croissent de gauche à droite et du bas en haute, il s'agit d'une translation et une inversion de la direction de l'ordonnée `y;` les deux fragments suivants sont équivalents :

```
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);
```

Dans le logiciel ici dessous il y a une fonction pour dessiner des textes.

```
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 est greque alpha
```

Script 33: Dessiner des textes dans les coordonnées cartésiennes

8.6 Effets spéciaux

8.6.1 Composition d'images

La propriété `globalCompositeOperation()` définit ou renvoie la manière dont une image source (nouvelle) est dessinées sur une image, existante, de destination ; entres ses valeurs il y a :

- `source-over` l'image source est dessinée au-dessus de l'image de destination, il est la composition initial,
- `destination-over` l'image source est dessinée au-dessous de l'image de destination,
- `source-atop` est visible seul la partie de l'image source à l'intérieur de l'image de destination,
- `destination-atop` est visible seul la partie de l'image destination à l'intérieur de l'image source.

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

8.6.2 Ombres

La taille de l'ombre est la taille de l'objet, plus la taille de la zone de flou ; il y a quatre propriétés pour créer l'effet d'ombrage :

`shadowOffsetX`	distances entre l'objet et l'ombre en pixel	`5`
`shadowOffsetY`		`-5`
`shadowBlur`	taille de la zone de flou en pixel	`15`
`shadowColor`	couleur de l'ombre	`"red"`

Figure 11: Propriétés pour l'ombrage

17. Exercice

Essayez de dessiner l'image de la Figure 11: Propriétés pour l'ombrage.

8.6.3 Gradients

Le gradient est une transition graduelle entre au moins deux couleurs, qui peut être appliqué à lignes et formes par les méthodes `strokeStyle()` et `fillStyle()`. Le gradient est un objet qu'il peut être créée linéaire ou radial.

Les couleurs sont indiquées par la méthode de l'objet gradient `addColorStop(`*`position`*`,` *`couleur`*`)`; où *`position`* varie entre `0` et `1` et indique où, dans le gradient, s'arrête la *`couleur`*, de ce point la couleur se transforme dans la prochaine couleur qu'il est atteint à son point d'arrêt :

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

Dans l'exemple il y a rouge au début pour `3/10` du gradient, suivi d'une transition au vert qui termine a `7/10` du gradient ; le reste est rempli par la dernière couleur (le vert).

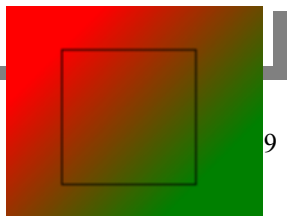
8.6.3.1 Gradients linéaires

Le gradient est obtenu par lignes de couler parallèles.

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

Les paramétrés de la méthode sont les coordonnées de deux points qui individuent la direction des lignes de transition et aussi la portion d'espace du canevas où il s'applique ; au dehors la couleur est la couleur des extrêmes:

```
var id = document.getElementById("canvas1");
var ctx = id.getContext("2d");
```



```
ctx.fillStyle="#00ff00";
var grd=ctx.createLinearGradient(50,50,150,150); // gradient diagonal
grd.addColorStop(0.0,'red');
grd.addColorStop(1,'green');
ctx.fillStyle=grd;
ctx.fillRect(0,0,id.width,id.height);   // tous le canevas
ctx.strokeRect(50,50,100,100)
```

Script 34: Canevas: gradient linéaire

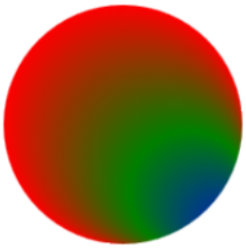
8.6.3.2 Gradients radials

Dans le gradient radial la transition est par un tronc de cône individué par deux cercles et la transition est du bord du premier cercle jusqu'à bord du deuxième.

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

Si est $r_1 > 0$ la couleur de l'intérieur du premier cercle est celle du premier couleur ; l'espace où la transition à lieu est celle de la projection du cône.

```
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 35: Canevas: gradient radial

8.6.4 Motifs

La méthode `createPattern(`*`image`*`, `*`répétition`*`)` utilise l'*`image`* pour créer un objet motif. Le second argument peut être une chaîne avec une des valeurs `repeat` (la valeur de défaut), `repeat-x`, `repeat-y`, et `no-repeat`.

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

8.6.5 Manipulation de l'image

La méthode `getImageData` crée un `imageData` objet dont la propriété `data` est l'image ou une partie sous forme d'une matrice de pixels, où chaque pixel est représentée par quatre bytes: rouge, vert, bleu et transparence et il peut être utilisée pour manipuler l'image.

```
...
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 36: getImageData

L' insertion d'une portion de l'image est possible par la méthode `putImageData` que, dans sa forme complète, elle permet d'insérer seulement une portion de l'objet `imageData`, ce qu'il est appelée *sale* (*dirty*) rectangle, en effet l'entier image est virtuellement copiée mais non pas ce qu'il est dehors du *sale* 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 37: putImageData

9 Annexes

9.1 Notes sur les expressions régulières

Une expression régulière est une chaîne de caractères utilisées pour rechercher, contrôler, extraire des textes dans un texte ; elle a une syntaxe cryptique et ici il y a un esquisse avec quelques exemples.

L'expression régulière est renfermé entre `//` et peut être suivie par des modificateurs comme **`i`** pour ignorer la casse.

L'expression est formée avec les caractères à rechercher dans le texte et éventuels caractères de contrôle, entre celui-ci il y a **`\`** dit *escape* utilisée pour introduire les caractères de contrôle ou des catégories de caractères :

- **`\`** escape caractère
  - **`\w`** quelconque caractère alphabétique,
  - **`\s`** des *white space* i.e. tabulation, line feed, form feed, carriage return, et espace,
  - **`\d`** quelconque chiffre,
- quantificateurs, s'appliquent au(x) caractère(s) qui le(s) précèd(ent)
  - **`*`** zéro ou plus caractères
  - **`+`** un ou plus caractères
  - **`?`** 0 ou 1 caractère (éventuel)
- **`{n}`, `{n,}` et `{n,m}`** respectivement exactement *n* caractères, au moins *n* caractères et entre *n* et *m* caractères.
- **`.`** (point) quelconque caractère

`( )` renferment un groupe de caractères ;ce qui est renfermé est mémorisé dans les variables `$1`, `$2`, ...

`[a-z]` quelconque lettre entre `a` et `z` comprises.

`$` (au fond de l'expression)

`^` (au début de l'expression)

9.1.1 Exemples

`/^\s*$/`	ensemble vide ou avec (white) espaces
`/aa+/`	trouve une séquence de deux ou plus a, donc il intercepte `aa`, `aaa`, `....`
`/(\w+)\s+(\w+)\s+(\w+)/`	trouve et mémorise trois mots
`/(.+)\s+(\w+)$/;`	`$1` tous les mots sauf le dernier, `$2` le dernier mot
`/(\-[a-z])/`	trouve et mémorise tiret suivi d'une lettre
`/\.(jpg|jpeg)$/i`	contrôles le type du fichier (ici jpg et jpeg), sans tenir compte de la casse
`/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/`	contrôle l'adresse courriel
`/^\d+$/`	seulement nombres entiers
`/^[-+]?\d{1,2}(\.\d{1,2})?$/`	**Valeur numériques** **`[-+]?`** le signe est possible **`\d{1,2}`** une ou deux chiffres **`(\.\d{1,2})?`** il est possibles le point décimal, suivi par une ou deux chiffres

```
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 38: Contrôle de l'adresse courriel

9.2 Tables

9.2.1 Figures

9.2.2 Scripts

9.2.3 Réponses

Question 1 63
Questions 2 1) 9, 2) Août, 3) 6
Question 3 1 opérateur d'assignation, 1 opérateur arithmétique.
Question 7 Quelques nombres pairs sont signalées comme premiers.
Question 8 Dans `do ... while` les instructions sont toujours exécutées une fois.

Question 9 1) le nom il n'est pas un nom valide. 2) le nom est contenu dans une variable.

Questions 11 1) moyenne arithmétique 2) `object` array

Question 12 Efface le tiret et change en majuscule
la lettre qui suit (*Camel function*).