

1. Les types de données

Il existe implicitement quatre types de données en JavaScript:

les nombres, les booléens, l'élément **null** et les chaînes de caractères.

Les nombres:

Les nombres peuvent être des *entiers* (base 10, base 8, base 16), *des nombres à virgule*, ou des *nombres avec exposant*.

La valeur spéciale **NaN** signifie "Not a Number".

Les booléens :

Les booléens peuvent avoir deux valeurs qui sont **true** pour vrai ou **false** pour faux.

L'élément null :

La valeur **null** représente la valeur *rien*.

Cette valeur est différente de 0 ou de chaîne vide ''.

Les chaînes de caractères :

Une chaîne peut contenir aucun ou plusieurs caractères délimités par des guillemets doubles ou simples.

Tableau des types de données :

nombre	Décimale (base 10)	0, 154, -17	Entier normal
	Octale (base 8)	035	Entier précédé d'un zéro
	Hexadécimale(base16)	0x4A, 0X4A	Entier précédé de 0x ou 0X
booléen	true (vrai) ou false (faux)		
null	Mot clé qui représente la valeur nulle		
Chaîne de caractères	"JavaScript" '125'		

Remarque:

En JavaScript, il est possible de convertir des chaînes en entier ou en nombre à virgule via les fonctions suivantes:

La commande **parseInt("75")** renvoie la valeur 75

et **parseFloat("41.56")** renvoie la valeur 41.56 .

Il est possible d'insérer un nombre dans une chaîne (par simple **concaténation** via l'opérateur +):

"le cours va durer " + 5 + " jours" → "le cours va durer 5 jours"

La fonction prédéfinie **isNaN(chExpr)** renvoie true si chExpr n'est pas numérique

2. Les variables (implicites ou explicites)

Le mot clef **var** permet d'explicitement déclarer une variable:

```
var v1;
v1 = "Bonjour"
v2 = "valeur" // v2 est implicitement une variable
```

3. Les opérateurs et expressions

3.1. Opérateurs arithmétiques:

Il est possible d'utiliser les opérateurs classiques : l'addition (+), la soustraction (-), la multiplication (*), la division (/) et le reste de la division entière: le modulo (%). On peut aussi effectuer une incrémentation (++), une décrémentation (--), et une négation unaire (-).

On peut utiliser les opérateurs d'incrément et de décrétement de deux manières:

++A, --A: incrémente ou décrémente (d'abord) A d'une unité et renvoie le résultat
A++, A--: renvoie le résultat et incrémente ou décrémente (ensuite) A d'une unité

Exemples:

```
12 + 7 → 19
12 % 5 → 2
A = 3 ; B = ++A → B = 4 et A = 4 ; C = A++ → C = 4 et A = 5
A = 3 ; B = --A → B = 2 et A = 2 ; C = A-- → C = 2 et A = 1
```

3.2. Les opérateurs d'attribution (affectation)

```
a += b <==> a = a + b // idem pour autres opérations
```

Exemples :

A=6, B=3

A += B → A=9, B=3

A /= B → A=2, B=3

3.3. Opérateurs logiques

Opérateur	Description
&&	"Et" logique, retourne la valeur true lorsque les deux opérandes ont pour valeur true sinon false
	"Ou" logique, retourne la valeur true lorsque l'un des opérandes a pour valeur true et false lorsque les deux opérandes ont pour valeur false .
!	"Non" logique, renvoie la valeur true si l'opérande a pour valeur false et inversement.

NB: JavaScript effectue une évaluation en court-circuit, permettant d'évaluer rapidement une expression, avec les règles suivantes:

- **false** avec && → prend toujours pour valeur **false**
- **true** avec || → prend toujours pour valeur **true**

exemple: `if(false && (a++ == 5)) ==>` le `a++` n'est jamais exécuté

3.4. Les opérations de comparaison

Les opérateurs de comparaison peuvent comparer des chaînes et des nombres. De plus, ces opérateurs de comparaison sont des opérateurs binaires.

Opérateur	Description
<code>==</code>	Retourne la valeur true, si les opérandes sont égales
<code>!=</code>	Retourne la valeur true, si les opérandes sont différentes
<code><</code>	Retourne la valeur true, si l'opérande gauche est strictement inférieure à l'opérande droite.
<code><=</code>	Retourne la valeur true, si l'opérande gauche est inférieure ou égale à l'opérande droite.
<code>></code>	Retourne la valeur true, si l'opérande gauche est strictement supérieure à l'opérande droite.
<code>>=</code>	Retourne la valeur true, si l'opérande gauche est supérieure ou égale à l'opérande droite.

Attention: Ne pas confondre l'opérateur d'affectation (`=`) avec le test d'égalité (`==`) .

3.5. Opérateur conditionnel ternaire (`? :`)

Résultat = `(condition_a_evaluer) ? valeur_si_vrai : valeur_si_faux`

exemple: `alert((jour == "lundi") ? "Bonne semaine" : "on n'est pas lundi")`

3.6. Priorités entre les opérateurs

Priorité la plus forte

Parenthèses (`()`)
 Multiplication, division, modulo (`* / %`)
 Addition, Soustraction (`+ -`)
 Opérateurs d'égalité (`== !=`)
 "Et" logique (`&&`)
 "Ou" logique (`||`)
 Opérateurs conditionnels (`? :`)
 Opérateurs d'attribution (`= += -= *= /= %=`)

Priorité la plus faible

Remarque: en cas de doute (trou de mémoire), il est fortement conseillé d'utiliser des parenthèses.

Tableau de caractères spéciaux (pour les chaînes):

Caractère	Description
<code>\t</code>	Tabulation
<code>\n</code>	Nouvelle ligne
<code>\r</code>	Retour chariot
<code>\f</code>	Saut de page

\b	Retour arrière
----	----------------

Structure de comparaison : if ... else

```
if(condition) simple_instruction_alors //; si else sur même ligne
else simple_instruction_sinon
```

Exemple:

```
if ( heure < 12 ) document.write("Good Morning")
else document.write("Good Afternoon");
```

Si on souhaite effectuer plusieurs instructions la syntaxe est la suivante:

```
if (condition)
{
    commande_1
    commande_n
}
```

4. Instruction switch/case

```
switch(variableNumerique)
{
case 1:
    commande1; commande2;
    break;
case 2:
case 3:
    commandeA; commandeB;
    break;
default:
    commandeX;
}
```

5. Les boucles

5.1. boucle "for"

```
for ( valeur_de_départ ; condition_pour_continuer ; incrémentation )
{
    bloc de commandes;
}
```

Exemple:

```
function tableau()
{
}
nom = new tableau //création d'un tableau vide

for ( i = 0 ; i < 4 ; i ++ )
{
    nom[i] = prompt ( "Donnez un nom", "" );
}
```

```
}
}
```

```
for ( i = 0 ; i < 4 ; i ++ ) alert(nom[i]);
```

for (i=10 ; i > 0; i--) → décrémentation de 1

for (i=1 ; i < 115; i+=5) → de cinq en cinq

5.2. boucle "for ... in "

La boucle **for....in** sert à parcourir automatiquement toutes les propriétés d'un objet (ou bien tous les éléments d'un tableau).

for (indice **in** tableau)

```
{
  //commande(s)/instruction(s) sur tableau[i];
}
```

5.3. boucle "while" (tant que)

```
while ( condition)
{
  //commandes_exécutées_tant_que_la_condition_est_vraie;
}
```

5.4. instructions "break" et "continue"

La commande **break** permet à tout moment d'interrompre complètement une boucle (**for** ou **while**) même si cette dernière ne s'est pas exécutée complètement.

Exemple:

```
for (i = 0 ; i < 10 ; i++)
{
  num=prompt("Donner un nombre", "");
  if (num == "0") break;
}
```

→ Si l'utilisateur saisit le nombre 0, on sort de la boucle.

La commande **continue** permet de passer à l'itération suivante dans une boucle **for** ou **while**. A la différence de la commande **break**, **continue** n'interrompt pas la boucle mais exécute la mise à jour de l'indice pour **for** et le **test** pour **while**.

Exemple:

```
for (i = 0 ; i < 10 ; i++)
{
  if (i == 5) continue;
  num=prompt("Donner un nombre", "");
}
```

→ quand i sera égal à 5, on passera directement à l'itération suivante sans exécuter la commande **prompt**.

6. Fonction eval

La fonction **eval**(chExpr) permet d'interpréter l'instruction javascript qui est dans la chaîne chExpr.

Exemples:

```
var res = eval ("3+2") // res vaudra 5  
var ch = eval("navigator.appName") // Netscape ou Internet Explorer
```

On peut ainsi écrire du code javascript qui sera interprété plus tard:

```
chExpr = "document.forms[" + nomFrm + "].reset()"  
eval (chExpr)
```

Remarque:

window.**setTimeout**(chExpr,n) permet d'interpréter l'expression chExpr en différé (n ms plus tard)

window.**setInterval**(chExpr,n) permet de lancer l'interprétation périodique de chExpr toutes les n ms;

L'objet Math

L'objet **Math** permet d'effectuer des opérations mathématiques évoluées:

Nom	Description
Propriétés:	
SQRT2	Racine carré de 2 ($\cong 1.414$)
SQR1_2	Racine carré de $\frac{1}{2}$ ($\cong 0.707$)
E	Constante d'Euler ($\cong 2.718$)
LN10	Logarithme naturel de 10 ($\cong 2.302$)
LN2	Logarithme naturel de 2 ($\cong 0.693$)
PI	Pi ($\cong 3.1415$)
Méthodes:	
acos()	Calcule l'arc cosinus en radians
asin()	Calcule l'arc sinus en radians
atan()	Calcule l'arc tangente en radians
cos()	Calcule le cosinus en radians
sin()	Calcule le sinus en radians
tan()	Calcule la tangente en radians
abs()	Calcule la valeur absolue d'un nombre
ceil()	Renvoie l'entier supérieur ou égal à un nombre
max()	Renvoie le plus grand de deux nombres
min()	Renvoie le plus petit de deux nombres
round()	Arrondit un nombre à l'entier le plus proche
random()	Renvoie un nombre aléatoire compris entre 0 et 1
exp()	Calcule e à la puissance d'un nombre
floor()	Renvoie l'entier inférieur ou égal à un nombre
log()	Calcule le logarithme naturel d'un nombre
pow()	Calcule la valeur d'un nombre à la puissance d'un autre
sqrt()	Calcule la racine carré d'un nombre

Exemples:

```
périmètre = Math.PI * 2 * rayon;
maxi = Math.max( 125, 158);
```

7. Opérations sur les chaînes de caractères

Tout objet de type **String** comporte un ensemble de méthodes permettant d'effectuer les manipulations suivantes sur des chaînes de caractères:

Nom	Description
Propriété	
length	Donne le nombre de caractères d'une chaîne
Méthode:	
anchor()	Encadre la chaîne dans une balise <A>
link(url)	Reçoit une URL et place la chaîne dans une balise <A> pour créer un lien hypertexte
big()	Encadre la chaîne dans une balise html <BIG>
small()	Encadre la chaîne dans une balise html <SMALL>
bold()	Encadre la chaîne dans balise html
fixed()	Encadre la chaîne dans balise html <TT>
italics()	Encadre la chaîne dans balise html <I>
strike()	Encadre la chaîne dans balise html <STRIKE>
sub()	Encadre la chaîne dans balise html <SUB>
sup()	Encadre la chaîne dans balise html <SUP>
blink()	Encadre la chaîne dans balise html <BLINK>
fontcolor(couleur)	Encadre la chaîne dans balise html et
fontsize(taille)	Encadre la chaîne dans balise html et
indexOf()	Reçoit une chaîne et un éventuel index initial et renvoie l'index de l'occurrence de la chaîne située après l'index initial.
lastIndexOf()	Reçoit une chaîne et un éventuel index initial et renvoie l'index de la dernière occurrence de la chaîne.
toLowerCase()	Retourne une copie de la chaîne en minuscules
toUpperCase()	Retourne une copie de la chaîne en MAJUSCULES
substring (deb,apresDernier)	Reçoit deux arguments entiers et renvoie la chaîne qui commence au premier argument et finit au niveau du caractère situé avant le second argument.
charAt (pos)	Reçoit un index pour argument et renvoie le caractère situé à cet index.

Exemple:

```
ch = "debut" + "suite" + "fin"
```

```
var chaine = "ma petite chaine";
chaine = chaine.toUpperCase(); // ==> chaine vaut maintenant "MA PETITE CHAINE".
```

```
ch="abc"
c=ch.charAt(0) // ==> c vaut "a"
chDeb=ch.substring(0,2) // ==> chDeb vaut "ab"
if(ch.indexOf("bc")<0) alert("bc" non trouvée dans : " + ch )
```


8. L'objet Date

Les objets de type **Date** permettent de travailler sur les heures (heures, minutes, secondes) et bien entendu sur les dates (mois, jours, année).

Pour définir un objet date en JavaScript on peut utiliser plusieurs constructeurs :

```
date1 = new Date(); // date et heure courantes
date2 = new Date(année, mois, jour);
date3 = new Date(année, mois, jour, heures, minutes, secondes);
```

Principales méthodes:

- **getDate()** Retourne le jour du mois sous forme d'entier compris entre 1 et 31.
- **getDay()** Retourne le jour de la semaine sous forme d'entier (0 pour dimanche, 1 pour lundi, etc.).
- **getHours()** Retourne l'heure sous forme d'entier compris entre 0 et 23.
- **getMinutes()** Retourne les minutes sous forme d'entier compris entre 0 et 59.
- **getSeconds()** Retourne les secondes sous forme d'entier compris entre 0 et 59.
- **getMonth()** Retourne le mois sous forme d'entier compris entre 0 et 11 (0 pour janvier et 11 pour décembre).
- **getTime()** Retourne le nombre de secondes qui se sont écoulées depuis le 1 janvier 1970 à 00:00:00.
- **getTimezoneOffset()** Retourne la différence existante entre l'heure locale et l'heure GMT en minutes.
- **getFullYear()** Retourne l'année sous forme d'un entier à deux chiffres 97 pour 1997.
- **getFullYear()** Retourne l'année.
- **setDate(date)** Définit le jour du mois sous forme d'entier compris entre 1 et 31.
- **setHours(heures)** Définit l'heure sous forme d'entier compris entre 0 et 23.
- **setMinutes(minutes)** Définit les minutes sous forme d'entier compris entre 0 et 59.
- **setMonth(mois)** Définit le mois sous forme d'entier compris entre 0 et 11 (0 pour janvier et 11 pour décembre).
- **setSeconds(secondes)** Définit les secondes sous forme d'entier compris entre 0 et 59.
- **setTime(l'heure)** Définit l'heure sur la base du nombre de secondes écoulées depuis le 1 janvier 1970 à 00:00:00
- **setYear(année)** Définit l'année sur la base d'un entier de quatre chiffres >1990.
- **toGMTString()** Retourne la date et l'heure en cours suivant les conventions d'Internet ("Lun 10 Jan 1997 14:45:10 GMT")
- **toLocaleString()** Retourne la date sous la forme MM/JJ/AA HH:MM:SS.

Exemple:

```
Jour_1 = new Date(1997,01, 25)
j = Jour_1.getDate() ➔ j= 25
```

9. Fonction à nombre d'arguments variable

```
function fl()
{
  nb_arg=fl.arguments.length
  if(nb_arg > 0) { premier_arg=fl.arguments[0] ; alert(premier_arg) }
  if(nb_arg > 1) { deuxieme_arg=fl.arguments[1] ; alert(deuxieme_arg) }
}
```

fl(); fl('a1'); fl('a1','a2')

10. Nouveaux type d'objets (non prédéfinis) et tableaux

Le langage JavaScript comporte un **mécanisme ultra-simple** pour **fabriquer de nouveaux type d'objet**:

Il suffit de créer une fonction qui servira à construire un nouvel objet. On désigne ce genre de fonction des "créateurs de **prototypes** d'objets"

Le mot clef **this** désigne l'objet (l'instance) courant(e).

Exemple:

```
function affVoiture()
{
  alert("marque= "+this.marque + ", modele= " + this.modele)
}

function Voiture(marque,modele)
{
  this.marque=marque // propriété 1
  this.modele=modele // propriété 2
  this.aff=affVoiture // méthode A (fonction attachée à une classe d'objet)
}
```

Création d'un ou plusieurs exemplaires de la classe Voiture:

```
v1 = new Voiture("Peugeot","306")
v2 = new Voiture("Renault","Mégane")
```

Utilisation des instances:

```
v1.modele = "206" // modification d'une propriété
v1.aff() // appel d'une méthode
```

Remarque: Il est possible d'ajouter dynamiquement une nouvelle propriété (ou des méthodes) au sein d'un objet déjà créé:

```
v1.couleur="rouge"
alert(v1.couleur)
```

Remarque très importante:

JavaScript gère les membres (propriétés ou méthodes) d'un objet sous la forme d'un tableau redimensionnable de choses quelconques (nombre, chaîne, objet,...).

Inversement, un tableau personnalisé doit être construit comme un objet.

Exemple1:

```
v1.marque <==> v1["marque"] <==> v1[0]
```

Exemple2:

```
function afficherToutesValeurs(obj)
{
  var ch=""
  for( i in obj)
  {
    m = "" + obj[i] //Remarque: "" + permet de convertir quelquechose en chaîne
    if(m.indexOf("function")<0)
    {
      // ne tenir compte que des propriétés (en écartant les fonctions)
      ch += ( obj[i] + " " )
    }
  }
  alert( ch )
}
```

afficherToutesValeurs(v1)

Exemple3 (Tableaux):

```
function tableau() // Array existe déjà
{
  // fonction pour construire un tableau vide
}
```

```
function tableau_initialise() //Array() existe déjà et fait la même chose
{
  for(i=0;i<tableau_initialise.arguments.length;i++)
    this[i]=tableau_initialise.arguments[i]
}
```

var tab1 = new tableau() // ou new **Array**()

tab1[0]="e1"

tab1[1]="e2"

afficherToutesValeurs(tab1)

//var tab2 = new tableau_initialise("hiver","printemps","ete","automne")

```
var tab2 = new Array("hiver","printemps","ete","automne")
```

afficherToutesValeurs(tab2)

Exemple basique concret (pour la syntaxe):

On peut modéliser un menu déroulant en Javascript / DHTML de la façon suivante:

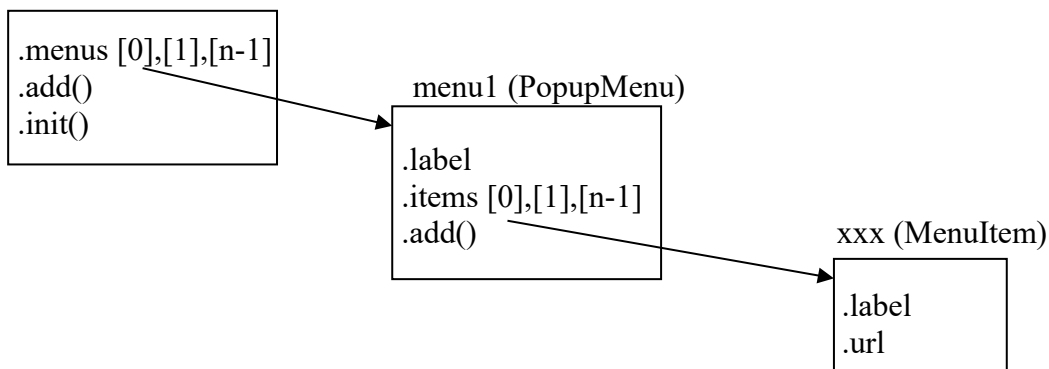
```
var menu1 = new PopupMenu("Menu 1");
menu1.add( new MenuItem("titre1A","http://www.xxx.com/page1A.html"));
menu1.add( new MenuItem("titre1B","http://www.xxx.com/page1B.html"));
```

```
var menu2 = new PopupMenu("Menu 2");
menu2.add( new MenuItem("titre2A","http://www.xxx.com/page2A.html"));
menu2.add( new MenuItem("titre2B","http://www.xxx.com/page2B.html"));
```

```
var barreMenu = new MenuBar();
barreMenu.add(menu1);
barreMenu.add(menu2);
barreMenu.init(); // initialisation et affichage.
```

⇒ on doit alors programmer les classes MenuBar, PopupMenu et MenuItem:

barreMenu (MenuBar)



```
function PopupMenu(libelle)
{
  this.label = libelle;
  this.items = new Array();
  this.add = fctAddItemInMenu;
}
```

```
function fctAddItemInMenu(item)
{
  var n = this.items.length; // position déjà occupée: de 0 à n-1
  this.items[n] = item; // ajout d'un élément dans le tableau
}
```

.....