II Les éléments de base du langage JavaScript

Comme dans tous les langages de programmation, il est important de pouvoir stocker des valeurs, d'effectuer des opérations, des tests, des boucles … Le Javascript ne déroge pas à cette règle et propose ces éléments habituels que l'on retrouve dans tous les langages de programmation un peu évolués.

II.1 Les constantes et les variables

Sans ces éléments, vous ne pouvez stocker d'informations. Javascript les utilise aussi.

a. Les constantes

Les constantes permettent une persistance de la valeur au cours de l'exécution du programme et permettent de rendre le code plus lisible.
Elles peuvent contenir des valeurs numériques ou caractères.
Pour une constante il n'y a pas de déclaration ou de typage à effectuer, il suffit d'utiliser un nom de constante et de lui affecter une valeur au moyen de l'opérateur d'affectation =.

Exemple

```
Pi = 3.14 ;
```

Remarque
Vous pouvez malgré tout utiliser le type ***const*** devant le nom de la constante que vous désirez utiliser.

Attention
Lors de la définition d'une constante, le fait d'utiliser des guillemets autour de la valeur est très important. Dans ce cas, il définit un élément chaîne de caractère et non un élément numérique.

Exemple

```
<script language="javascript">
const five = 5 ;
document.write(five + 1) ;
</script>
```

```
<script language="javascript">
const five = "5" ;
document.write(five + 1) ;
</script>
```

Résultat : **6** Résultat : **51**

Il existe des constantes prédéfinies tels :

Constante	Explication
undefined	La variable qui prend cette valeur a été déclarée mais n'a pas été initialisée. variable = ;
null	La variable n'existe pas.
infinity	Cette constante représente l'infini positif. Cela permet notamment de vérifier si il y a une division par zéro.
NaN	Equivaut à la définition de l'IEEE de « Not a Number ».

b. Les variables

Le javascript n'est pas a proprement dit un langage typé comme peut l'être le langage C par exemple. Vous pouvez donc déclarer vos variables sans indiquer leur type voir même utiliser des variables sans les déclarer.

1. Les types simples

Il en existe trois.

- String pour les chaînes de caractères
- Boolean pour les booléens (true et false)
- Number pour les numériques (entier ou flottant)

La déclaration se fait de la façon suivante :

nom_variable = **new** *type* **(***valeur***)** ;

nom_variable est le nom que vous donnez à votre variable.
new est l'opérateur qui permet de réserver la place en mémoire.
type est le type de la variable (String, Boolean ou Number).
valeur est la valeur d'initialisation de la variable.

Attention
N'oubliez pas la majuscule en première lettre du type.

Remarque

Si vous omettez la partie *valeur* lors de la déclaration, automatiquement il est affecté à la variable les valeurs :

0 si c'est un type Number
false si c'est un type Boolean
vide si c'est un type String. Dans ce dernier cas, la variable est définie et non nulle.

Exemples

```
<script language="javascript">
tva = new Number(18.6) ;
vide = new String ;
Faux = new Boolean(false) ;
</script>
```

2. Les types composés simples : les listes

Ces types permettent de stocker plusieurs valeurs dans un seul conteneur (variable). Ces valeurs peuvent être de types différents. En effet, c'est l'utilisation que vous en faites qui est importante.

- Les listes simples

A une variable on associe plusieurs valeurs. Pour bien comprendre ce que cela représente, il suffit de vous représenter cela comme une commode où chaque tiroir est numéroté de 0 à n (n est un entier positif et représente l'indice). Dans chaque tiroir se trouve ranger un objet (c'est la valeur).

La déclaration et l'initialisation peuvent se faire en utilisant trois méthodes.

```
Liste = new Array("valeur1","valeur2","valeur3");
```

Le conteneur ou variable se nomme *Liste*. Trois valeurs sont rangées en même temps que la variable *Liste* est déclarée.
La première valeur (valeur1) est rangée à l'indice 0 (tiroir 0) tandis que la deuxième valeur (valeur2) est rangée à l'indice 1 (tiroir 1) et ainsi de suite.
La ligne suivante affichera la valeur *valeur1*.

```
document.write(Liste[0]);
```

```
Liste = new Array(6);
```

La variable *Liste* est déclarée afin qu'elle puisse posséder 6 valeurs. Les indices vont alors de 0 à 5 mais aucune valeur n'est attribuée à ces indices. La valeur ***undefined*** leur est alors affectée.
La ligne suivante affichera la « valeur » *undefined*.

```
document.write(Liste[0]);
```

Remarque

Malgré la déclaration, rien ne vous empêche de stocker plus de 6 valeurs dans cette liste. Vous pouvez aussi omettre le nombre d'indices
Liste = new Array();

```
Liste[0] = "valeur1";
```

Les valeurs sont explicitement associées à leur indice.

Remarque

La dernière méthode peut être combinée avec les deux précédentes.

Remarque

Il est possible d'utiliser les raccourcis de déclaration suivants :
Liste[]="valeur1"; (valeur1 est associé au premier indice disponible)
Ou
Liste=["valeur1","valeur2", "valeur3"]; (identique à la première méthode)

Attention

Ne vous trompez pas entre les parenthèses () et les crochets [] car le résultat n'est pas du tout identique.

```
<script language="javascript">
Liste=["toto","tutu"];
document.write(Liste[1]);
</script>
```

Syntaxe correcte.
Le résultat est ***tutu***

```
<script language="javascript">
Liste=("toto","tutu");
document.write(Liste[1]);
</script>
```

Syntaxe incorrecte.
Le résultat est ***u***
On a rangé toto puis tutu dans la variable simple Liste et on demande d'afficher la 2ème lettre du mot.

Important

Une propriété très utile est celle qui compte le nombre d'éléments dans une liste. Son utilisation est la suivante :

Code	Résultat
`<script language="javascript">` `Liste=["toto","tutu"];` `document.write(Liste[1]);` `document.write(" "+Liste.length) ;` `</script>`	Le résultat est ***tutu*** **2**

- Les listes associatives

A la différence de la liste simple, dans les listes associatives possède au moins un de leurs indices qui est une chaîne de caractère (élément non numérique).
La déclaration et initialisation s'effectuent de la façon suivante :

```
Listeasso=new Array();
Listeasso["jacquenod"]="frédéric" ;
```

Après a délacration de la liste, on lui affecte une valeur *frédéric* à l'indice *jacquenod*.

Code	Résultat

`<script language="javascript">`
`Listeasso=new Array();`
`Listeasso["jacquenod"]="frédéric" ;`
`Listeasso["martin"]="jean" ;`
`document.write(Listeasso["jacquenod"]) ;`
`</script>`
	Le résultat est
	frédéric

3. Les types composés multidimensions : les tableaux

Le tableau est simplement une liste de liste. Dans un tiroir de la commode, pour reprendre l'exemple précédent, vous trouverez une autre … commode et ainsi de suite selon la profondeur de votre tableau.

Remarque

Chaque « tiroir » n'a pas forcément le même nombre de commode à l'intérieur. En clair, la profondeur peut être différente selon les indices dans un même tableau.

Il existe deux types de tableaux :

- Les tableaux à indices numériques
- Les tableaux à indices chaînes de caractères aussi appelés tableaux associatifs.

  - Les tableaux à indices numériques

Le premier niveau de cette structure est un indice numérique.
Le plus simple est de prendre un exemple.

Exemple
Le fichier shadow password (/etc/passwd) sur un système Unix contient un ensemble de login et chacun de ces logins possèdent un ensemble de valeurs qui lui sont associées (uid, gid, gecos, homedir, shell). Nous nous bornerons à l'utilisation des trois premières (uid, gid, gecos).
Pour stocker ces logins dans une structure de liste simple, vous seriez obligés de créer autant de listes que vous possédez de logins dans ce fichier.
Avec un tableau plus de problème, vous possédez une seule structure qui va permettre de tout stocker.
La déclaration et l'initialisation se font de la manière suivante :

```
<script language="javascript">
tabpasswd=new Array();
tabpasswd[0]=new Array("jacqueno",250,280,"FJ,762");
tabpasswd[1]=new Array("martin",251,280,"JM,763");
document.write(tabpasswd[0][0]+"<br>");
document.write(tabpasswd[0][1]+"<br>");
document.write(tabpasswd[1][0]);
</script>
```

Le résultat est
jacqueno
250
martin

L'exemple précédent propose un tableau à deux dimensions mais rien n'empêche de faire plus.

```
<script language="javascript">
tab3d=new Array();
tab3d[0]=new Array();
tab3d[0][0]=new Array("jacqueno",250,280,"FJ,762");
document.write(tab3d[0][0][3]);
</script>
```

Le résultat est
FJ,762

L'initialisation peut être faite de manière automatique en utilisant une boucle par exemple.

- Les tableaux associatifs

Le principe est identique à celui vu ci-dessus à ceci prêt que au moins un des indices du tableau n’est pas numérique mais est une chaîne de caractères (String).

En reprenant l’exemple précédent on remarque tout de suite que le tableau associatif est plus adapté et plus facilement utilisable par la suite si, au lieu d’insérer des indices numériques en premier niveau, on insère des indices chaînes de caractères représentant le login de l’utilisateur. En effet dans notre cas précédent, pour retrouver un login il fallait parcourir toute la structure car l’indice était non explicite. Il fallait ouvrir tous les tiroirs jusqu’à ce que le contenu soit le bon mais à priori on ne savait pas où il se situait dans la structure. L’indice était neutre.

La structure suivante permet d’avoir un indice explicite ce qui facilite grandement les recherches ensuite.

```
<script language="javascript">
tabpasswd=new Array();
tabpasswd["jacqueno"]=new Array(250,280,"FJ,762");
tabpasswd["martin"]=new Array(251,280,"JM,763");
login="jacqueno" ;
document.write(tabpasswd[login][0]+"<br>");
document.write(tabpasswd[login][1]+"<br>");
</script>
```

Le résultat est
250
280

Si on connaît le login (indice dans notre cas) il est très simple d’obtenir les informations en rapport, ce qui n’était pas le cas précédemment.

4. Autres méthodes de stockage des valeurs

Javascript est un langage « objet » ; Pour le moment rien dans l’utilisation proposée ne le montre.
Mettons donc à contribution cette caractéristique.
Il existe deux méthodes.

- Les classes

Nous allons créer une classe (fonction) qui va s’appliquer à notre objet : le tableau.

Remarque
Une fonction est aussi appelée classe ou méthode.

Reprenons l'exemple précédent. Je vais créer une classe qui contient les informations en rapport avec le login (uid, gid et gecos).

```
…
function Info(uid,gid,gecos)
{
 this.uid=uid ;
 this.gid=gid ;
 this.gecos=gecos ;
}
…
```

Rappel
this est un opérateur qui fait référence à l'objet courant. Cet opérateur n'est utilisable qu'à l'intérieur d'une classe. Il référence alors le parent de l'objet. Ici this représente le parent Info auquel appartient les éléments uid, gid et gecos.

Une fois la classe (ou fonction) créée, il faut déclarer le tableau associatif qui est la variable (plus précisément l'objet). L'objet tableau est un type complexe dont les éléments se réclament de la classe Info.

La déclaration et l'initialisation sont alors :

```
tabasso=new Array() ;
tabasso["jacqueno"]=new Info(250,280, "FJ,762") ;
tabasso["martin"]=new Info(251,280,"JM,763") ;
```

L'utilisation se fait de la façon suivante :

```
document.write(tabasso["jacqueno"].uid) ;
document.write(tabasso["martin"].gecos) ;
```

Ces termes de classe, objets, méthodes seront revus en détail plus loin.

Exemple
Nous voulons à partir d'un objet tabasso afficher les informations dans les cases text d'un formulaire dans le cas où le login demandé existe ou alors afficher un message d'alerte qui indique une erreur.

Le programme

```
<HTML>
<HEAD>
<TITLE>Les tableaux associatifs</TITLE>
<!-- Debut du script javascript -->
<script language=javascript>
// Creation d'une classe Info contenant 3 proprietes
function Info(uid,gid,gecos)
{
 this.uid = uid
 this.gid = gid
 this.gecos = gecos
}

// Declaration et initialisation de l'objet tabasso
tabasso= new Array()
tabasso["jacqueno"] = new Info(250,280,"FJ,762")
tabasso["lambda"] = new Info(56,50,"User LAMBDA")

// Creation d'une methode
function recherche(login)
{
 login = login.toLowerCase()
 // Le login demandez n'existe pas
 if(typeof tabasso[login] == "undefined")
 {
  login2 = login + ":"
  if(login2 == ":")
  {
   // Champ de saisi du login vide
   alert("Vous n'avez rien saisi !")
  }
  else
  {
   // Champ de saisi du login rempli mais non trouvé dans l'objet tabasso
   alert("L'utilisateur -->" + login + "<-- est inconnu !")
  }
 }
 else
 {
  // Champ de saisi du login rempli et trouvé dans l'objet tabasso
   // Mise a jour champ login (minuscule) 1er champ donc indice 0
  document.forms[0].elements[0].value = login
   // Mise a jour du champ uid 3eme champ donc indice 2
  document.forms[0].elements[2].value = tabasso[login].uid
   // Mise a jour du champ uid 4eme champ donc indice 3
  document.forms[0].elements[3].value = tabasso[login].gid
   // Mise a jour du champ gecos 5eme champ donc indice 4
```

```
  document.forms[0].elements[4].value = tabasso[login].gecos
 }
}
</script>
</HEAD>
<BODY BGCOLOR=#FFFFFF>
<B>
<!-- Creation du formulaire -->
<FORM>
<U>Donnez le login recherché</U><BR><BR>
<INPUT TYPE=text VALUE="" SIZE=10 MAXLENGTH=8>
<!-- Activation de la recherche et de l'affichage -->
<INPUT TYPE=button VALUE=afficher
onClick="recherche(document.forms[0].elements[0].value)">
<BR> <BR>
<HR>
<BR>
<!-- Partie affichage des infos -->
<U>Affichage des informations trouvées</U>
<BR> <BR>
uid = <INPUT TYPE=text><BR>
gid = <INPUT TYPE=text><BR>
gecos = <INPUT TYPE=text  SIZE=30><BR>
</FORM>
</B>
</BODY>
</HTML>
```

Les copies d’écran

Formulaire vide (premier chargement)

Donnez le login recherché

afficher

Affichage des informations trouvées

uid =
gid =
gecos =

L'utilisateur a entré le login *lambda* et cliquez sur le bouton **afficher**

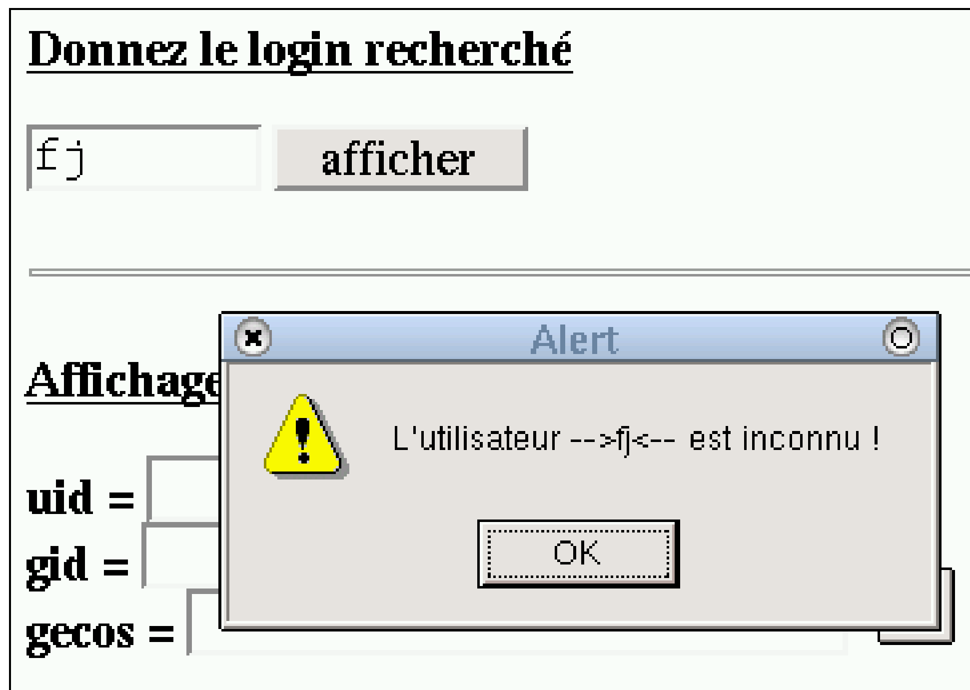
Donnez le login recherché

lambda afficher

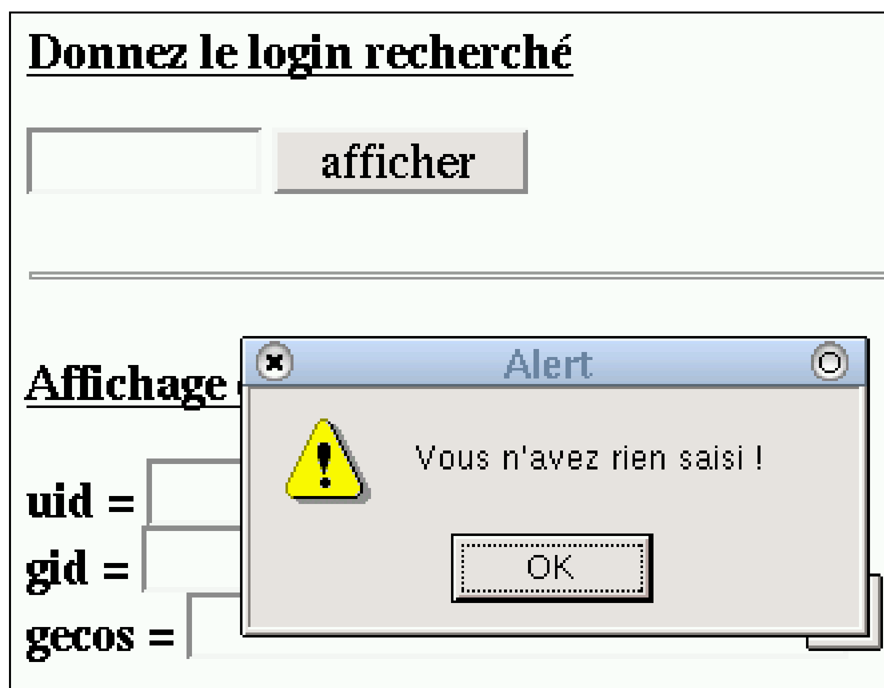
Affichage des informations trouvées

uid = 56
gid = 50
gecos = User LAMBDA

L'utilisateur a entré un login qui n'existe pas



L'utilisateur a cliqué sur le bouton **afficher** sans saisir de login



- Les méthodes

Le stockage est se fait en utilisant les propriétés d'un objet. La déclaration et l'utilisation sont spécifiques à ce mécanisme.
Vous pouvez avoir des propriétés avec des indices numériques ou des indices chaînes de caractères. L'utilisation est alors différente.

- Les indices numériques

Déclaration

logins = {
jacqueno:{0:"250",1:"280",2:"FJ,762"},
lambda:{0:"56",1:"50",2:"LAMBDA"}
}

Utilisation
document.write(logins.jacqueno[0]) ; // renvoie la valeur 250

- Les indices chaînes de caractères

Déclaration

logins = {
jacqueno:{uid:"250",gid:"280",gecos:"FJ,762"},
lambda:{uid:"56",gid:"50",gecos:"LAMBDA"}
}

Utilisation
document.write(logins.jacqueno.uid) ; // renvoie la valeur 250

II.2 Les opérateurs

Une fois les éléments de stockage mis en place, certaines opérations vont devoir être réalisées, test, ajout, …
Il est alors nécessaire d'utiliser des opérateurs.

a. Les opérateurs arithmétiques

Ces opérateurs n'ont d'intérêt que pour des opérations entre éléments dont la valeur est de type numérique.
Le tableau suivant présente les opérateurs existants, leur signification ainsi qu'un exemple d'utilisation.

Rappel
Dans une opération arithmétique, les éléments de part et d'autre de l'opérateur se nomment les opérandes.

Pour les exemples prenons deux opérandes **a** qui possède la valeur numérique 7 et **b** qui possède la valeur numérique 2.

Opérateur	Signification	Exemple
+	Addition	a + b renvoie 9
-	Soustraction	a – b renvoie 5
*	Multiplication	a * b renvoie 14
/	Division	a / b renvoie 3.5
%	Modulo (reste de la division)	a % b renvoie 1
=	Affectation	res = a + b; res renvoie 9
++	Incrémentation	a++ renvoie 8
--	Décrémentation	a-- renvoie 6

Rappel

Si une division est faite avec un diviseur égal à 0 cela provoque une erreur dans la plupart des langages. En javascript, le résultat est **Infinity**. Il vous est donc possible de tester cette éventualité pour éviter cette erreur.

```
<script language="javascript">
a = 7 ; b = 0 ;
res = a / b ;
if(res == Infinity){
 document.write("erreur le diviseur vaut 0" + res) ;
}
else{
 document.write(a + "/" + b + " vaut " + res) ;
}
</script>
```

Attention

Si vous utilisez un opérateur arithmétique et que l’un des opérandes n’est pas numérique, la réponse à l’affichage sera **NaN** sauf dans le cas de l’addition où l’opérateur **+** se transforme alors en opérateur « chaîne de caractère » de concaténation.

```
<script language="javascript">
a = "FJ" ; b = 3 ;
document.write(a + b) ; // renvoie FJ3
document.write(a – b) ; // renvoie NaN
</script>
```

Remarque

Le modulo (%) qui renvoie le reste de la division n'est pas forcément un entier.

```
<script language="javascript">
a =7 ; b = 3 ;
document.write(a % b) ; // renvoie 2.3333333333333335
</script>
```

il existe pour les opérateurs unaires ++ et -- deux utilisations.
Une est à priori et l'autre à posteriori. Le résultat n'est pas le même.
Prenons l'exemple où a vaut 7.

- A priori

document.write(--a) ; renvoie la valeur 6
La décrémentation s'effectue avant l'affichage de la valeur de a.

- A posteriori

document.write(a--) ; renvoie la valeur 7
La décrémentation s'effectue après l'affichage de la valeur de a.

Exemple

```
<script language="javascript">
c = 1;
liste = new Array(4,7,9) ;
document.write(liste[c++]) ; // renvoie 7
document.write(liste[c]) ; // renvoie 9
document.write(liste[c--]) ; // renvoie 9
</script>
```

b. Les opérateurs de comparaison

Ces opérateurs se retrouvent dans des tests permettant ainsi de comparer les valeurs des expressions.
Le résultat fourni par ces opérateurs est true (vrai) ou false (faux).

Le tableau suivant les récapitule.
Pour les exemples reprenons les deux variables **a** qui possède la valeur numérique 7 et **b** qui possède la valeur numérique 2.
Pour les deux derniers opérateurs (=== et !==) **a** vaut 7 et **b** vaut "7".

Opérateur	Signification	Exemple
<	Strictement plus petit	a < b renvoie false
>	Strictement plus grand	a > b renvoie true

<=	Plus petit ou égal	a <= b renvoie false
>=	Plus grand ou égal	a >= b renvoie true
==	Egalité des contenus	a == b renvoie false
!=	Différence des contenus	a != b renvoie true
===	Egalité des contenus et du type	a===b renvoie false
!==	Différence des contenus et du type	a!==b renvoie true

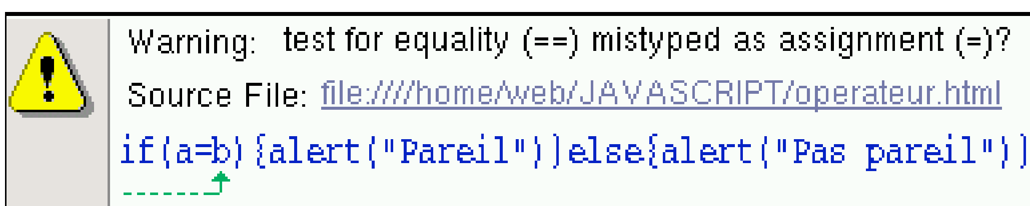
Attention

Une erreur très fréquente est d'utiliser l'opérateur **=** (affectation) lors un test en place et lieu de **==** (égalité des contenus). Ceci a pour fâcheux résultat d'avoir un test qui renverra toujours true sauf dans le cas ou l 'opérande de droite possède la valeur 0. En effet 0 est considéré comme false (if(0) renvoie false).

```
<script language="javascript">
a = 3 ;b=4 ;
//erreur on affecte a la variable a la valeur de b
// On passe toujours dans le alors alert("Pareil") ;
if(a = b){
 alert("Pareil") ; }
else{
 alert("Pas pareil") ;
}

// Cas particulier où b vaut 0
a = 3 ;b=0;
//erreur on affecte a la variable a la valeur de b
// On passe toujours dans le sinon alert("Pas pareil") ;
if(a = b){
 alert("Pareil")
}
else{
 alert("Pas pareil") ;
}
</script>
```

Vous obtiendrez alors dans la console javascript de votre navigateur le « warning » suivant :



c. Les opérateurs logiques

Ces opérateurs permettent d'associer et de combiner des expressions lors de tests. Ces opérateurs sont les suivants :

Opérateur	Signification	Table de vérité
&&	ET	Vrai && Vrai -> Vrai Vrai && Faux -> Faux Faux && Vrai -> Faux Faux && Faux -> Faux
||	OU	Vrai **||** Vrai -> Vrai Vrai **||** Faux -> Vrai Faux **||** Vrai -> Vrai Faux **||** Faux -> Faux
!	NON	! Vrai ->Faux ! Faux -> Vrai

d. Les raccourcis

Il existe des raccourcis pour utiliser certains des opérateurs vus précédemment.

Attention
La compréhension du code peut devenir plus compliquée.

- Les opérateurs standard

Pour les exemples reprenons la variable **a** qui possède la valeur numérique 7.

Raccourci	Equivalence	Valeur
a-=1	a = a – 1	6
a+=2	a = a + 2	9
a*=2	a = a * 2	14
a/=2	a = a / 2	3.5
a%=2	a = a % 2	1

- L'opérateur unaire

Il existe un autre opérateur raccourci appelé opérateur unaire. Il est représenté par le caractère **?**.
Cet opérateur permet de contracter un test if.

La syntaxe est la suivante :

```
action(expresion ? Vrai : Faux)
```

Exemple

```
<script language="javascript">
a = 7 ;b = 2 ;
document.write(a > b ? "a plus grand que b" : "a plus petit que b" ) ;
</script>
```

- Autre raccourci

Lors des tests ou des boucles, si l'action a effectuer ne contient qu'une ligne, vous pouvez omettre les accolades.

Exemple

```
<script language="javascript">
a = 7 ;b = 2 ;
if(a > b)
 document.write("a plus grand que b")
else
 document.write("a plus petit que b")
</script>
```

II.3 Les instructions conditionnelles

Dans un programme, il est souvent nécessaire d'effectuer des tests. Javascript ne propose rien de nouveau, il reprend les mêmes éléments que les langages classiques comme le C ou le Perl.

a. if

Cette instruction permet de tester le caractère Vrai ou Faux d'une expression ou d'un ensemble d'expressions lié par des opérateurs logiques pour ensuite effectuer des actions différentes en fonction du résultat.

Syntaxe

```
If(expression)
{
  // action si l'expression est Vraie
}
else
{
 // action si l'expression est Fausse
}
```

Rappel

Vous pouvez utiliser l'opérateur ternaire et omettre les accolades dans le cas d'une action sur une ligne.

Il est possible d'enchaîner les if et les else dans le cas où vous avez plusieurs éléments successifs à tester.
Dans ce cas, votre code devient vite illisible.

Il existe aussi en Javascript la possibilité d'utiliser la syntaxe else if.

Attention

La syntaxe est bien else if (un espace entre le else et le if) et non elseif.

Exemple

```
<script language="javascript">
var a = 7;
if(a < 0)
 document.write("a plus petit que 0")
else if(a == 0)
 document.write("a egal à 0")
else if(a > 0)
  document.write("a plus grand que 0")
else if(a != NaN)
 document.write("a n'est pas un nombre")
else
 document.write("Impossible !")
</script>
```

b. switch

Si vous devez tester la validité d'une variable par rapport à un ensemble de caractère ou de chaînes de caractères ou des valeurs numériques, vous pouvez utiliser le if ou le switch. Ce dernier rend votre code plus lisible. Il est donc à privilégier.

Syntaxe

```
switch(variable)
{
 case "valeur1" : action1 ;break ;
 case "valeur2" : action2 ;break ;
…
 default : action par défaut ;
}
```

Pour éviter de tester tous les cas, l'utilisation du *break* permet de sortir du switch dès l'action terminée.
Le *default* permet d'effectuer une action si la variable testée ne possède aucune des valeurs proposées dans les *case* qui précédent.

Attention

Le contenu des *case* doit être un élément déjà évalué. Vous ne pouvez y insérer des expressions.

II.4 Les boucles

Il est souvent utile afin d'automatiser des traitements d'effectuer des actions plusieurs fois. Ce « nombre de fois » peut être fixe ou non. Les instructions de boucle permettent cette répétition.

a. for

Cette boucle est utilisée lorsque le nombre de fois où vous devez exécuter une action est connu. Ce nombre est appelé itération.

Syntaxe

```
for(initialisation;condition de fin;incrémentation ou décrémentation)
{
 actions;
}
```

- **Initialisation**

Cette partie est constituée d'une ou de plusieurs expressions qui sont exécutées une seule fois au moment où le programme entre dans la boucle. Le plus souvent cette partie est l'initialisation d'un compteur.
for(**cpt=0** ; …)
Vous pouvez y insérer plusieurs initialisations. Elles sont alors séparées par le caractère **,** (virgule).
for(**cpt=0,somme=0** ; … ; …)

- **condition de fin**

Cette partie est constituée d'une expression. Lorsqu'elle est Vraie, l'exécution de la boucle est stoppée et l'interpréteur enchaîne alors avec les lignes de programme qui suivent celles de la boucle.
for(… ; **cpt <= 10** ;…)

- **incrémentation ou décrémentation**

L'incrémentation ou la décrémentation concerne la partie qui va évoluer pour tendre vers la condition de fin. Généralement, cette partie est une incrémentation du compteur d'une unité.
for(… ; … ; **cpt++)**

Remarque

Les parties initialisation et incrémentation ou décrémentation peuvent contenir plusieurs instructions.
for(**cpt=0,somme=0** ;**cpt <=10** ;**cpt++,somme=2*cpt**)

b. while

A la différence du for, le nombre d'itération à effectuer n'est à priori pas connue même si vous pouvez aussi l'utiliser lorsque ce nombre est connu. De plus, cette boucle permet de proposer des conditions de fin plus complexes et évoluées que le for.

Syntaxe

```
while(expression)
{
 actions ;
}
```

Attention

Un gros danger de cette boucle et de la boucle for dans une moindre mesure est ce que l'on appelle : la boucle infinie. Si la condition de fin ne

peut être atteinte, votre programme ne s'arrête plus et reste bloqué à l'intérieur de cette boucle. Typiquement, l'erreur est l'oubli d'incrémenter ou de décrémenter le compteur.

```
<script language="javascript">
cpt=0 ;
while(cpt <=5)
{
 document.write("cpt vaut" + cpt) ;
}
</script>
```

Remarque

L'instruction while permet de tester une première fois l'expression avant d'entrer dans la boucle et d'exécuter le bloc d'action.

c. do … while

L'utilisation de cette boucle est identique à la boucle while à un point prêt. La différence principale réside dans l'exécution du bloc d'actions. En effet, dans le while, l'expression est testée puis si elle est Vraie, le bloc d'actions est exécutée.
Le « do while » exécute le bloc d'actions puis effectue le test de l'expression.
Ceci implique que le bloc d'actions quelque soit la valeur de l'expression est exécutée au moins une fois.

Syntaxe

```
do
{
 actions ;
} while(expression);
```

Remarque

Vous pouvez utiliser l'instruction break pour sortir d'une boucle avant la fin normale de cette dernière. Vous pouvez aussi spécifier une étiquette (qui doit se trouver avant le break), l'interpréteur se positionnera alors à cet endroit.