Aide mémoire Javascript

1) Les commentaires

Les commentaires sont les même que ceux de PHP à savoir :

```
// Deux slash pour un commentaire sur une ligne
/* Un slash et une étoile pour un commentaire
   sur plusieurs lignes...
*/
```

Les commentaires ne sont pas pris en compte. Ils ne servent que de repères et d'aide pour le codeur du script.

2) Variables

Les variables en JavaScript n'ont pas nécessairement besoin d'être déclarées; mais il est plus correct de le faire :

```
// Déclaration de la variable nombre
var nombre;
```

La déclaration de variable se fait avec l'instruction var (variables en anglais (et en français)). Inutile de déclarer le type de variable (comme le feraient certains adeptes d'ActionScript). On peut immédiatement déclarer une valeur par défaut (mais cela n'est nullement obligatoire) :

```
var entier = 5;
var float = 1.999992;
var string = "J'adore wikiversity !";
var boolean = true; // ou false
var object = null;
```

3) Opérateurs

Tout comme les autres langages, JavaScript dispose d'opérateurs permettant d'effectuer les opérations informatiques de bases. Voici leur liste :

Opérateurs binaires d'affectation

Comme vu dans les exemples précédents, le symbole "=" (égal) permet d'affecter une valeur (nombre ou chaîne de caractères) à une variable.

```
var nombre = 18;
var chaine = "Bonjour tout le monde !";
// Et aussi
var a;
var b;
a = b;
// Et...
a = b = 0; // a et b valent toutes les deux 0
```

Opérateurs binaires de calcul

- L'addition s'effectue en utilisant le traditionnel symbole "+" (plus)
- La soustraction avec le symbole "-" (moins)
- La multiplication avec le symbole "*" (étoile)
- La division se sert du symbole "/" (slash)

```
var a = b = c = 2; // On initialise a, b et c
var d = a/(b+c)-a*4; // d = 2/(2+2)-2*4 = 1/2-8 = -7.5
var e = 3*(3+3); // e = 3*6 = 18
```

Il existe aussi les opérateurs +=, -=, *= et /= qui s'utilisent lorsque l'opération s'effectue sur une seule variable. Voici un exemple très simple :

```
var  a  =  3;
var b = a-2;
a -= b; // a = a-b = 3-1 = 2
a += b; // a = a+b = 3+1 = 4
b *= a; // b = b*a = 1*3 = 3
b /= b; // b = b/b = 1
```

Opérateurs de comparaisons

Les opérateurs de comparaison permettent de comparer deux valeurs et renvoient une valeur de type booléen (true, vrai ou false, faux) selon les cas. La syntaxe est simple à comprendre :

```
Valeur1 Opérateur de comparaison Valeur2
```

- > : est strictement supérieur
- < : est strictement inférieur
- >= : est supérieur ou égal
- <= : est inférieur ou égal
- == : est strictement égal
- != : est strictement différent

```
var a = b = 10;
document.write(a>b); // false
document.write(a>=b); // true
document.write((a+1)>b); // true
document.write(b!=a); // false
document.write(b==a); // true
document.write(b<=a); // true
```

Bien entendu il ne faut pas confondre l'opérateur "=" d'affectation et l'opérateur "==" de comparaison.

Opérateur de négation

Comme dans l'exemple précédent, il s'agit du point d'exclamation "!". Ainsi, true devient false et vice-versa.

Exemple :

```
var a = 3;
var b = 5;
document.write(a<b); // affiche true car 3<5
document.write(!(a<b)); // affiche false car l'expression équivaut à a>=b
```

Concaténation

La concaténation est une technique qui consiste à rassembler deux chaînes de caractères pour n'en former qu'une seule. C'est en quelque sorte le collage de ces deux chaînes. Pour ce faire, utilisez le signe "+" et "+=". C'est le même principe que précédemment.

Voici comme vous faisiez avant pour afficher deux chaînes à la suite :

```
var chaine_a = "Bonjour";
var chaine_b = "tout le monde";

// On affiche les deux chaînes à la suite
document.write(chaine_a);
document.write(chaine_b);
```

Et bien désormais, vous pouvez faire ainsi :

```
var chaine_a = "Bonjour";
var chaine_b = "tout le monde";

// On affiche les deux chaînes à la suite avec la concatenation
document.write(chaine_a+chaine_b);
```

On décline un peu...

```
var chaine_a = "Bonjour";
var chaine_b = "tout le monde";

chaine_a += chaine_b; // chaine_a vaut Bonjourtout le monde
```

Notez bien qu'il existe une ambiguïté entre l'opérateur de concaténation et le signe de l'addition en JavaScript. Cependant, l'interpréteur reconnaît s'il s'agit d'une chaîne (donc concaténation) ou bien d'un nombre (addition).

Opérateurs d'incrémentation et de décrémentation

Ils sont deux et permettent soit de diminuer de 1 la valeur d'une variable (décrémentation), soit de l'augmenter de 1 (incrémentation).

- ++ : incrémente de 1
- -- : décrémente de 1

```
var a = 0;
a++; // a = a+1 = 0+1 = 1
a--; // a = a-1 = 1-1 = 0
a--; // a = -1
```
Notez que si vous placez ces opérateurs devant la variable à incrémenter, l'action sera faite immédiatement ; voici un exemple simple :
```
var a = 0;
document.write(a++); // Affiche 0
document.write(++a); // Affiche 2
document.write(--a); // Affiche 1
```

4) Structures conditionnelles

Tests conditionnels avec `if` et `else`

Les instructions if et else mettent en place une condition dans le code. Elles permettent de différencier plusieurs cas lors de l'exécution.
```
if (condition)
{
  ...
}
else
{
  ...
}
```

Dans la condition, on se sert généralement des opérateurs de comparaison mais toute fonction renvoyant un booléen est autorisée.
```
var age = 15;
if (age >= 18)
{
  document.write("Vous pouvez entrer.");
}
else
{
  document.write("Vous ne pouvez pas entrer.");
}
```

Notez que ceci est également possible (lorsque le bloc à exécuter ne nécessite qu'une seule ligne) :
```
var age = 15;
if (age >= 18) document.write("Vous pouvez entrer.");
else document.write("Vous ne pouvez pas entrer.");
```

On peut aussi rajouter des conditions intermédiaires entre le if et le else qui permettront de tester plus de cas. Voyez plutôt :
```
var temps = "orageux";
var je_prends;

if (temps == "ensoleillé") je_prends = "mon parasol.";
else if (temps == "nuageux") je_prends = "mon chapeau.";
else if (temps == "pluvieux") je_prends = "mon parapluie.";
else je_prends = "mon paratonnerre.";

document.write("Lorsque le temps est "+temps+", je prends "+je_prends);
// Lorsque le temps est orageux, je prends mon paratonnerre.
```

Cet exemple montre comment gérer simplement les conditions multiples en JavaScript.

Ralier plusieurs conditions

Il est possible de ralier plusieurs conditions en une seule. Évidemment, la précision sera moindre mais cela se révèle vite incontournable et très pratique. Pour cela il existe deux opérateurs très simples d'utilisation.

- && : il veut dire ET.
- || : il veut dire OU.

Voyons comment cela fonctionne :

```
var temps = "neigeux";
var temperature = -5;
var je_prends;

if (temps == "neigeux" || temperature <= 0) je_prends = "ma luge ou mon blouson.";
else if (temps == "neigeux" && temperature <= 0) je_prends = "ma luge et mon blouson.";
else je_prends = "mes lunettes de soleil."; // Avec un peu de chance il fait beau :)

document.write("Lorsque le temps est "+temps+" et que la température est de
"+temperature+"{{Abbr|°C|degré Celcius}}, je prends "+je_prends);
// Lorsque le temps est orageux et que la température est de -5{{Abbr|°C|degré Celcius}},
je prends ma luge ou mon gros blouson.
```

On peut aussi séparer les ET et les OU avec des parenthèses.

L'alternative switch

Il existe une alternative intéressante aux `if` et `else`. En effet, lorsque vous avez un nombre important de cas à vérifier, il peut être intéressant de ne pas avoir à recopier une cascade de `else if`.

Pour cela, il y a le mot-clé switch :

```
switch (variable)
{
  case a: /*...*/ break;
  case b: /*...*/ break;
  case c: /*...*/ break;
  default:  /*...*/  break;
}
```

L'instruction `switch` permet de gérer plusieurs cas en fonction de la valeur d'une variable. Son utilisation, bien que pratique, est réputée être (un peu) plus lente que le `if` et surtout assez restreinte.

Exemple :

```
var temps = "soleil";
var je_prends;

switch (temps)
{
  case "soleil": je_prends = "mon parasol."; break;
  case "nuageux": je_prends = "mon chapeau."; break;
  case "pluvieux": je_prends = "mon parapluie."; break;
  default: je_prends = "mon paratonnerre."; break;
}

document.write("Lorsque le temps est "+temps+", je prends "+je_prends);
// Lorsque le temps est soleil, je prends mon parasol.
```

Structures ternaires

Derrière ce nom "barbare" se cache un procédé raisonnablement complexe mais surtout très pratique et permettant de gagner du temps, de la place et de la lisibilité.

Les ternaires sont en fait un concentré d'un groupe `if` et `else` (sans `else if` au milieu). Cela permet d'écrire une condition en une seule ligne. C'est une alternative aux structures conditionnelles vues précédemment. Voici la syntaxe :

```
(condition) ? ...bloc du if : ...bloc du else
```

Le "?" remplace donc l'accolade du `if` tandis que les " : " remplacent celle du `else`. Un code avec if et else :

```
var age = 15;
if (age >= 18) document.write("Vous pouvez entrer.");
else document.write("Vous ne pouvez pas entrer.");
```

Un code avec structure ternaire renvoyant le même résultat :

```
var age = 15;
document.write((age >= 18) ? "Vous pouvez entrer." : "Vous ne pouvez pas entrer.");
```

Pratique non ?

Ici, on a mis la condition en ternaire directement dans le `document.write` (allez faire ça avec un `if`…)

5) Structures répétitives : les boucles

Une boucle, c'est un bloc d'instructions qui sera répété tant qu'une condition sera vérifiée (= true). Une boucle peut être finie (elle finira à un moment) ou au contraire infinie (généralement il s'agit d'une erreur de code et cela finit par faire "planter" le navigateur).

La boucle `while`

L'instruction `while` est la boucle la plus simple d'utilisation puisqu'elle ne comporte qu'un seul "argument" : la condition ! Voici la syntaxe de `while` :

```
while (condition)
{
  ... (instructions, incrémentation...)
}
```

Voici un exemple simple :

```
var i = 0;
while (i < 5)
{
  document.write("Ligne "+i+"<br />");
  i++;
}
document.write("Boucle terminée !");
```

Cela va vous afficher ceci :

Ligne 0 *Ligne 1* *Ligne 2* *Ligne 3* *Ligne 4* *Boucle terminée !*

L'instruction `do... while`

La boucle `do while` est la sœur de la boucle while puisqu'il s'agit de la même boucle **SAUF** que le bloc d'instruction s'effectuera au minimum une fois et ce même si la condition de la boucle n'est pas vérifiée dès le début.

Voici un exemple simple :

```
var i = 5;
do
{
  document.write("Ligne "+i+"<br />");
  i++;
}
while (i < 5);

document.write("Boucle terminée !");
```

Ici, la condition vaut false dès le début. Un while normal aurait donc sauté la condition pour passer à la suite du code. Mais le `do while` exécute une fois le code et affiche :

Ligne 5

Boucle terminée !

L'instruction `for`

Cette structure répétitive ne porte pas bien son nom contrairement aux précédentes. Pour faire court, `for` est un *alias* de `while` à la différence qu'elle regroupe en son sein l'initialisation de(s) variable(s) utilisée(s), la condition ainsi que l'incrémentation (décrémentation) de celle(s)-ci.

Syntaxe :

```
for (initialisation; condition; incrémentation)
{
  ...
}
```

Exemple :

```
for (i = 0; i<5; i++)
{
  document.write("Ligne "+i+"<br />");
}
document.write("Boucle terminée !");
```

Le résultat est le même que celui de la boucle `while`. Ceci dit, l'utilisation d'une des trois boucles dépend du contexte et des besoins du script.

L'instruction `break`

Il arrive parfois que l'on veuille sortir d'une boucle alors même que la condition de la boucle est encore vraie. Pour cela, on peut faire appel à l'instruction `break`. Lorsque cette instruction est exécutée, la boucle se termine immédiatement et le code la suivant est exécuté.

```
var i = 1;
while (i <= 10)
{
  if (i == 5) break;
  i++;
}
document.write("La valeur de i est "+i);
```

Ce qui affiche : *La valeur de i est 5*. En effet, le programme sort de la boucle lorsque i prend la valeur de 5, donc après 4 tours.

L'instruction `continue`

Le rôle de l'instruction `continue` est de sauter le tour actuel et d'en recommencer un nouveau. Cette instruction s'utilise exactement de la même façon que `break`.

6) Tableaux à indices numériques

Ce sont des tableaux où chaque valeur est associée à un indice (nombre entier positif).

Déclaration

Voici comment déclarer un tableau à indices numériques en JavaScript et lui donner des valeurs initiales :

```
// première méthode :
var mon_tableau = new Array('Christophe', 'Sarah', 'Carole', 'Alex', 'Nicolas');
// seconde méthode :
var mon_tableau = ['Christophe', 'Sarah', 'Carole', 'Alex', 'Nicolas', 'Sandrine'];
// et pour un tableau vide :
var mon_tableau = [];
```

Accès aux valeurs

Pour accéder aux valeurs d'un tableau à indices numériques, la seule possibilité est de passer par l'indice de chacune des valeurs contenues dans ce tableau. La numérotation des indices commence par 0 (zéro).

```
mon_tableau[0] = "toto" // Range la chaîne de caractères dans la première case du tableau
tmp = mon_tableau[4] // écrit dans la variable tmp le contenu de la 5ème case du tableau
```

Listage des valeurs

Pour lister l'intégralité du tableau, il nous faut utiliser une boucle. Il va nous être utile de connaître la "longueur" du tableau (le nombre d'indice qu'il possède). Pour cela, on fait appel à la méthode **length** de l'objet Array. Ainsi, on accède aux valeurs de notre tableau grâce à ses indices comme ceci :

```
for (i = 0 ; i < mon_tableau.length ; i++)
{
  document.write(i+" => "+mon_tableau[i]); // On affiche chaque couples indice => valeur
}
```

Affection de valeurs

Pour remplir un tableau avec une seule valeur (13 par exemple), on utilise une boucle.

```
var a = 13;
var long_tableau = 10;
var mon_tableau = new Array();
for (i = 0; i <= long_tableau; i++)
{
  mon_tableau[i] = a;
}
```

Il existe des tableaux de tableaux de tableau... (ce sont des tableaux multidimensionnels) :

```
var mon_tableau = new Array('Christophe', new Array('Sarah', 'Carole', 'Nicolas'));
document.write(mon_tableau[1][0]); // Affiche "Sarah"
```

Des outils pratiques

Même si nous n'avons pas encore vu comment marchaient les fonctions en JavaScript; il est à savoir que celui-ci met à disposition des fonctions pour la gestion des tableaux. En voici la liste des principales :

- **concat**(Tableau1, Tableau2[, Tableau3, ...]) : cette méthode permet de concaténer (coller) plusieurs tableaux pour n'en former plus qu'un seul.
- **join**(Tableau) ou Tableau.**join()** ou Tableau.**toString**() : renvoie tous les éléments (valeurs) de Tableau sous forme d'une chaîne de caractères.
- **pop**(Tableau) ou Tableau.pop() : supprime le dernier élément de Tableau après avoir retourné sa valeur.
- Tableau.**shift**() : supprime le premier élément de Tableau après avoir retourné sa valeur.
- Tableau.**unshift**(valeur1[, valeur2, ...]) : permet d'ajouter des éléments au début.
- Tableau.**push**(valeur1[, valeur2, ...]) : permet d'ajouter des éléments à la fin de Tableau.
- Tableau.**sort**() : permet de trier tous les éléments de Tableau.
- Tableau.**reverse**() : inverse l'ordre des éléments de Tableau.
- Tableau.**splice**(début, fin, valeur) : écrase les valeurs des éléments dans Tableau dont l'indice est compris entre début et fin.
- Tableau.**slice**(début, fin) : retourne les éléments dont l'indice est compris entre fin et début.

7) Objets

Les objets, omniprésents dans le JavaScript, peuvent être créés afin d'obtenir l'équivalent d'un tableau associatif, c'est-à-dire lorsque chacune de ses valeurs est associée à un nom.

Déclaration

Pour créer un objet, on peut utiliser les deux syntaxes :

```
var mon_objet = new Object();
// ou : var mon_objet = {};
```

Pour attacher le nom à sa valeur, on utilise le signe mathématique égal comme ceci :

```
mon_objet['prenom_0'] = 'Christophe';
mon_objet['prenom_1'] = 'Sarah';
mon_objet['prenom_2'] = 'Carole';
mon_objet['prenom_3'] = 'Alex';
mon_objet['prenom_4'] = 'Nicolas';
mon_objet['prenom_5'] =  'Sandrine';
```

Ou on peut les attacher à la création de l'objet :

```
var mon_object = {
        "prenom_0": 'Christophe',
        'prenom_1': 'Sarah',
        'prenom_2': 'Carole'
};
```

Accès aux valeurs

Comme précédemment sauf qu'on utilise le nom associé à la variable au lieu de son indice :

```
document.write(mon_objet['prenom_3']); // Affiche "Alex"

// Ou bien, si la clé respecte la règle concernant la nomination des variables :
document.write(mon_objet.prenom_1); // Affiche "Sarah"
```

Listage des valeurs

Pour lister les valeurs de l'objet mon_objet ci-dessus, on procède ainsi :

```
for (var nom_indice in mon_object)
{
        document.write(mon_object[nom_indice]);
}
```

En effet, la boucle for(...in...) attribue à la variable *nom_indice* le nom de l'indice et passe au suivant à chaque itération. Il est à noter que cette méthode permet aussi de lister les noms d'indices si nécessaire.

Affectation de valeurs

Soit directement en utilisant le nom de l'indice:

```
mon_objet['prenom_3'] = "Alex"
// Ou comme précédemment :
mon_objet.prenom_3 = "Alex"
```

Soit via la boucle for(...in...) (ici remise à 0 de toutes les valeurs de l'objet)

```
for (var nom_indice in mon_objet)
{
        mon_objet[nom_indice] = 0;
}
```

8) Boîte d'alerte

Il existe une boîte de dialogue qui a pour principal objectif d'alerter le visiteur de quelque chose. Il s'agit de la boîte d'alerte.

Caractéristiques:

- Nom : alert()
- Boutons : "OK" (et la croix de fermeture)
- Code : window.alert (texte d'alerte);

Le **window** devant la fonction alert est **facultatif**. Exemple :

```
alert('Bonjour, bienvenue sur mon site.\nIl est tout neuf !');
```

Notez l'emploi du caractère spécial **\n** pour sauter une ligne dans une alerte JavaScript.

9) Boîte confirmation

JavaScript met aussi a disposition des codeurs une fonction nommée **confirm** qui permet d'afficher une boîte de dialogue demandant une confirmation au visiteur. Par exemple : *Voulez-vous supprimer tous les messages ?*.

Caractéristiques:

- Nom: confirm()
- Boutons : "OK", "ANNULER" (et la croix de fermeture)
- Code : window.confirm(texte de confirmation);

Exemple :

```
var choix = confirm("Voulez-vous supprimer tous les messages ?");
if (choix) alert("Vous avez cliqué sur OK");
else alert("Vous avez cliqué sur ANNULER ou vous avez fermé");
```

On comprend donc sur cet exemple que **confirm** peut renvoyer deux valeurs :

1. **TRUE** si le visiteur a cliqué sur *OK*.
2. **FALSE** si le visiteur a cliqué sur *ANNULER* ou a cliqué sur *la croix rouge* de fermeture.

10) Boîte de saisie

C'est la dernière boîte de dialogue que nous allons voir. Celle-ci permet de demander une saisie au visiteur. Elle contient donc un champs de texte.

Caractéristiques:

- Nom: prompt
- Boutons : "OK", "ANNULER" (et la croix de fermeture)
- Code : window.prompt(texte de demande, valeur par défaut du textfield);

Exemple :

```
do
{
  choix = prompt("Veuillez entrer un nombre supérieur à zéro :", 0);
}
while (isNaN(choix) || !choix || Number(choix) < 0);
document.write("Le nombre que vous avez entré est : "+choix);
```

Le code est certes plus compliqué que précédemment mais il est tout à fait compréhensible. Explication :

- En premier lieu, on utilise la boucle **do-while** car nous voulons être sûr que celle-ci sera parcourue au moins une fois (on aurait aussi pu initialiser choix à null avant une boucle while simple).
- Dans la boucle, on demande d'entrer un nombre positif via le prompt.
- Ensuite, il y a le while et sa condition : les **||** veulent dire **OU**. **isNaN** (is Not a Number) permet de vérifier si la saisie est bien un nombre. Puis on vérifie que le visiteur n'a pas cliqué sur *ANNULER* (*! choix* ou *choix == false*) et enfin, on vérifie que le nombre est supérieur à 0 (avec *Number(choix) > 0*).

11) Les fonctions

Premièrement, pour déclarer une fonction il faut le mot clé **function** suivi du nom de notre fonction sans caractères spéciaux (pas d'espace, de virgule...). Puis, entre accolades se trouvent les instructions de la fonction. Une fonction peut posséder des arguments (ou paramètres) qui sont des noms de variable à mettre entre parenthèses après le nom de la fonction et avant la première accolade et séparés par des virgules.

Pour commencer simplement, nous allons faire une petite fonction qui va simplement afficher la phrase "Bonjour tout le monde" un certain nombre de fois.

```
function afficher_phrase (phrase, n) // On déclare la fonction
{
  var term = (parseInt(n) > 1) ? '<br/>' : ''; // Si n > 1, on saute une ligne
  var i = 1; // On démarre i à 1
  do
  {
    document.write(phrase+term);
    i++;
  }
  while (i <= n);
}
```

Voilà, notre fonction est écrite. Une précision : la fonction `parseInt()` permet de convertir une chaîne en un nombre.

Maintenant, comment se servir de notre fonction ?

```
// Appel à notre fonction
afficher_phrase ("Salut tout le monde", 10); // La phrase sera affichée 10 fois de suite
```

Retourner une valeur

À présent, créons une fonction de calcul. Et... qui dit calcul dit aussi que nous allons devoir renvoyer une valeur. Pour cela, il y a le mot clé **return** qui comme son nom l'indique veut dire "retourner" en anglais. Il est simplement suivi de la valeur à retourner.

Voici un exemple assez simple se servant de la librairie Math :

```
function volume_boule (rayon)
{
  return (4/3)*Math.PI*Math.pow(parseInt(rayon), 3);
}

document.write(volume_boule (10)); // On affiche le résultat : 4188.790204786391
```

Donc, quelques explications :

- **return** : il sert à retourner le résultat de la formule suivante.
- **Math.PI** : cela retourne la constante PI (3,14...). C'est une constante.
- **Math.pow** : il s'agit de la méthode permettant de mettre un nombre à un exposant (ici, le rayon est à l'exposant 3).

Ici, la fonction `volume_boule()` est donc de type `float` puisqu'elle renvoie un nombre décimal. Cependant, on aurait pu la transformer en fonction `void` (qui ne retourne aucune valeur) en remplaçant le return par un `document.write()` ou un `alert()`.