#06 : Javascript (fin)

1) Les tableaux

Les tableaux vont vous permettre de stocker plusieurs valeurs (chaîne, nombre) dans une structure unique.

Pour déclarer un tableau, la syntaxe est un peu particulière : var monTableau = new Array();

Le mot clé new est utilisé pour créer des objets (au sens informatique du terme, nous aurons l'occasion de revenir sur cette notion d'objet). Un tableau est donc un objet.

Ensuite, pour remplir le tableau il faut procéder comme suit : monTableau [indice de position] = maValeur

```
monTableau [0] = "pomme";
monTableau [1] = "orange";
```

ou alors avec des nombres :

```
monTableau [0] = 15;
monTableau [1] = 20;
```

pomme aura l'indice 0, orange aura l'indice 1, etc. **L'indice de position commence toujours à zéro.**

Il est aussi possible de déclarer et de remplir le tableau en même temps :

```
monTableau = new Array ("pomme", "orange", "cerise");
```

Un exemple à essayer :

```
var mesFruits = new Array ("orange","pomme","cerise");
document.write (mesFruits[0], " : indice de position 0 </br>");
document.write (mesFruits[1], " : indice de position 1</br>");
document.write (mesFruits[2], " : indice de position 2</br>");
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat.

Il est possible de connaître la taille de votre tableau avec la méthode **length** :

```
var mesFruits = new Array ("orange","pomme","cerise");
document.write (mesFruits[0], " : indice de position 0 </br>");
document.write (mesFruits[1], " : indice de position 1</br>");
document.write (mesFruits[2], " : indice de position 2</br>");
document.write ("Ce tableau possède ", mesFruits.length, " éléments");
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et observez le résultat.

Notez bien qu'il est possible de déclarer et de "remplir" un tableau comme suit :

```
var mesFruits=["orange","pomme","cerise"];
```

La méthode typeof est aussi utilisable avec un tableau.

Il est possible de remplacer l'indice de position par une variable

```
var mesFruits = new Array ("orange","pomme","cerise");
var i=prompt("Choisir un fruit (0 une orange ; 1 une pomme ; 2 une cerise)");
document.write ("Votre choix : une ", mesFruits[i]);
```

Vous ne trouvez pas cela étrange que cela fonctionne ? Comme nous l'avons déjà vu, prompt renvoie une chaîne de caractères, donc ici, i est de type string ! Or l'indice de position doit être de type **number** et pas de type string ! Avec mesFruits[i] nous devrions avoir une erreur.

Eh non, pas d'erreur, car JavaScript est capable de faire du "**transtypage automatique**" : étudions le "raisonnement" de JavaScript : "i est de type string, mais dans mesFruits[i] il doit être de type **number**, de mon propre chef j'applique donc un "**parseInt**" à i". Voilà pourquoi nous n'avons pas d'erreur !

Pour terminer sur les tableaux, nous verrons un peu plus loin que l'instruction **for...in** nous permettra de parcourir un tableau quasi automatiquement.

2) La boucle while (tant que)

La notion de boucle est fondamentale en informatique. Une boucle permet d'exécuter plusieurs fois des instructions. La structure de la boucle while est la suivante :

```
while (condition)
{
      instructions à répéter
}
```

Tant que la **condition** reste vraie, les instructions à l'intérieur du bloc (entre les 2 accolades) seront exécutées.

```
var i=0;

while (i<=10)
{
      document.write("i vaut ",i,"</br>");
      i=i+1;
}
```

Ouvrez le fichier HTML à l'aide du navigateur Firefox et expliquez le résultat.

_ _

_ _

J'attire votre attention sur la ligne i=i+1 : Nous avons ici aussi un exemple de la non-équivalence du signe égal en mathématiques et en informatique. En mathématiques, écrire i=i+1 n'a aucun sens (cela reviendrait à dire par exemple que 5 = 6 !).

Ici i=i+1 veut dire : "Prends la valeur de i, ajoute un à cette valeur puis attribue cette nouvelle valeur à la variable i", on parle **d'affectation**. Dans le cas spécial où on ajoute une constante à une variable on parle **d'incrémentation**.

Il est tellement courant d'utiliser i=i+1 que les informaticiens ont inventé "un raccourci" : **i++**

Voici un exemple un peu plus complet (table de multiplication) :

```
function table (a)
{
      var i=1;
      document.write("TABLE DES ",a, "</br>");

      while (i<=10)
      {
            var resultat=a*i;
            document.write(a," x ",i," = ",resultat,"</br>");
            i++;
      }
}

var numS=prompt("Entrez un chiffre entre 1 et 10");
var num=parseInt(numS);

while ((num<1) || (num>10) || (isNaN(num)))
{
      numS=prompt("Entrez un chiffre entre 1 et 10");
      num=parseInt(numS);
}

table(num);
```

Petite nouveauté dans cet exemple, la fonction isNaN(num).

Si la variable num n'est pas de type nombre, cette fonction renvoie vraie (true). Si la variable num est de type nombre, la fonction **isNaN** renvoie alors faux (false).

Vous pouvez aussi constater que notre code débute par l'écriture d'une fonction. Si vous avez toute une série de fonctions à écrire, il est préférable de toutes les écrire en début de code.

Expliquez ce que fait la fonction **table** :

_ _
_ _
_ _

Expliquez le fonctionnement de la boucle **while** :

_ _
_ _
_ _

3) La boucle do while

C'est un peu la "cousine" de la boucle while, voici sa structure :

```
do
{
      instructions à répéter
}
while (condition);
```

Ici les instructions sont forcément exécutées au moins une fois (ce qui n'est pas le cas pour la boucle while). Reprenons la fin de l’exemple précédent, mais avec une boucle do while :

```
do
{
      var numS=prompt("Entrez un chiffre entre 1 et 10");
      var num=parseInt(numS);
}
while ((num<1) || (num>10) || (isNaN(num)));

document.write("le nombre choisi est" ,num);
```

Vous voyez que dans ce cas précis, l'utilisation de la boucle **do while** est plus logique car il faut acquérir la valeur avant de la tester.

4) La boucles for (boucle comptée)

Nous avons toujours ici affaire à une boucle, mais les conditions de répétitions sont un peu différentes :

```
for (début ; condition de répétition ; incrémentation)
{
      instructions à répéter
}
```

Reprenons notre exemple des tables de multiplication avec une boucle for, testez ceci :

```
function table (a)
{
      document.write("TABLE DES ",a, "</br>");

      for ( i=1 ; i<=10 ; i++ )
      {
            var resultat=a*i;
            document.write(a," x ",i," = ",resultat,"</br>");
      }
}

var num = 7 ;

table(num);
```

La boucle **commence avec i = 1**, elle va **s'exécuter tant que i<=10** reste vraie, i est augmenté d'une unité à la fin de chaque boucle (i++).

Cette boucle for est de loin la plus utilisée, il faut bien la maîtriser! Entraînez-vous !

5) *La boucle for in (énumération)*

for in va nous permettre de parcourir un tableau de façon "quasi automatique", testez cet exemple :

```
var fruit=new Array("orange", "cerise", "pomme", "pêche", "poire");
document.write("Liste des fruits </br>");

for (var i in fruit)
{
      document.write(fruit[i], " , ");
}
```

Voici un dernier exemple que je vous laisse tester :

```
var fruit=new Array();
var i=0;
do
{
      fruit[i]=prompt("Entrez un nom de fruit (pour terminer taper fin)");
      i++;
}
while (fruit[i-1]!="fin");

document.write("Voici votre liste de fruits </br>");

for (var j=0 ; j<(fruit.length-1) ; j++)
{
      if (j==(fruit.length-2))
      {
            document.write(fruit[j]);
      }
      else
      {
            document.write(fruit[j], " , ");
      }
}
```

Pourquoi le fruit[i-1] ?

_ _
_ _
_ _

Pourquoi le fruit.length-1 ?

_ _
_ _
_ _
_ _

Pourquoi le fruit.length-2 ?

_ _
_ _
_ _
_ _

6) *Exercices*

- ❑ **Exercice** : Faites un nouveau programme JavaScript qui demande de rentrer un nombre premier et qui répond **Gagné** ou **Perdu** après avoir testé si le nombre donné est bien premier.
- ❑ Pour acquérir de la pratique (**et c'est le plus important**), faites tous les exercices de cette série : https://tech.io/playgrounds/3777