JS 10 : Tableaux

Objectifs

- Découvrir et comprendre la notion de tableau
- Créer et manipuler des tableaux et les données qu'ils contiennent.
- Connaître les fonctions courantes de manipulation de tableaux

Définition

Imaginons que dans un programme, nous ayons besoin simultanément de 12 valeurs, par exemple des notes pour calculer une moyenne. Evidemment, la seule solution dont nous disposons à l'heure actuelle consiste à déclarer douze variables, appelées par exemple N1, N2, N3 ... mais cela ne change pas fondamentalement le problème, car arrivé au calcul, et après une succession de douze instructions de lecture distinctes, cela donnera obligatoirement quelque chose comme :

Moy = (N1 + N2 + N3 + N4 + N5 + N6 + N7 + N8 + N9 + N10 + N11 + N12) / 12

Ce qui est laborieux. Et pour un peu que nous soyons dans un programme de gestion avec quelques centaines ou quelques milliers de valeurs à traiter, cela se révèle fortement problématique.

Si, de plus, on est dans une situation on l'on ne peut pas savoir d'avance combien il y aura de valeurs à traiter, on se retrouve là face à un mur.

C'est pourquoi la programmation nous permet **de rassembler toutes ces variables en une seule**, au sein de laquelle chaque valeur sera désignée par un numéro.

En bon français, cela donnerait donc quelque chose du genre « la note numéro 1 », « la note numéro 2 », « la note numéro 8 »...

Un tableau est une **suite d'éléments (une liste), de variables**. On peut accéder à un élément d'un tableau en utilisant sa position : l'index, ou encore « indice », « clé ».

Un tableau s'utilise comme une variable et peut donc être passé en argument à une fonction (y compris méthode d'une classe) ou être retourné comme résultat d'une fonction.

Déclaration et création

En Javascript, un tableau peut se déclarer de plusieurs façons :

Initialisation d'un tableau vide (sans données) :

```
var myTableau = [];
```

Initialisation d'un tableau avec des données de type chaîne :

```
var myTableau = ["pomme", "poire", "banane"]; // données de type chaine
var myTableau = [15, 496, 789]; // données de type entier
```

Notez qu'un tableau Javascript peut contenir des éléments de différents types (chaînes, entiers) à la fois, ce qui n'est pas le cas dans certains langages (par exemple en C#).

On pourrait donc avoir :

```
var myTableau = ["pomme", 123, "poire", 456];
```

Initialisation d'un tableau avec l'objet « Array » :

```
var myTableau = new Array(); // Tableau vide
var myTableau = Array(); // Tableau vide
var myTableau = new Array(5); // Tableau vide qui contiendra 5 éléments
var myTableau = Array(5); // Tableau vide qui contiendra 5 éléments
var myTableau = new Array("pomme", "poire", "banane"); Tableau avec données
var myTableau = Array("pomme", "poire", "banane"); // Tableau avec données
```

La traduction du mot « **tableau** » en **anglais** est « **array** ». On retrouve ce terme dans la plupart des langages informatiques.

Attention

Les syntaxes new Array(5) et Array(5) qui permettent de déclarer le nombre d'éléments d'un tableau font qu'il est impossible de les utiliser pour créer un tableau qui ne contiendrait qu'un seul élément de type entier. Dans ce cas, seule la syntaxe avec crochets doit être employée : var myTableau = [5]

Utilisation

Pour accéder à un élément du tableau, on appelle le tableau suivi entre crochets [] de la position de l'élément souhaité.

Mais attention : en Javascript (comme dans beaucoup d'autres langages), **le premier élément d'un tableau se trouve à l'indice 0**.

Donc, **le deuxième élément d'un tableau se trouve lui à la position 1**, le troisième à la position 2 et ainsi de suite avec toujours un décalage de -1. Par exemple dans le tableau suivant :
var myTableau = ["pomme", "poire", "banane", "fraise", "abricot"];

Si on veut accéder à l'**élément « pomme »** - le premier - on écrira **myTableau[0]**, pour l'élément « fraise » - le 4ème - on écrira myTableau[3].

Remplir un tableau

Lorsqu'on a déclaré un tableau vide, on le remplit en assignant une valeur à la position souhaitée :

```
var myTableau = [];
myTableau[0] = "pomme";
myTableau[1] = "poire";
```

Fonctions courantes sur les tableaux

Dans les langages informatiques, de nombreuses fonctions natives spécifiques à chacun d'entre eux permettent d'exploiter les tableaux et leurs données : tris, calculs, extraction de données, connaître la longueur (c'est-à-dire le nombre d'éléments d'un tableau) etc.

Voici quelques fonctions utiles en Javascript.

Connaître le nombre d'éléments dans un tableau

La fonction `length` (= longueur) retourne le nombre d'éléments dans un tableau :

```
var myTableau = ["pomme", "poire", "banane", "fraise", "abricot"];
var nb = myTableau.length ;
console.log("Le tableau contient "+nb+ "elements"); // Affiche : 5
```

Parcourir un tableau

Les boucles `for` et `foreach` permettent, combinées à la fonction `length()` de parcourir un ta-bleau : on va passer autant de fois que le tableau contient des éléments, c'est-à-dire tant qu'on n'a pas atteint la longueur du tableau :

```
var myTableau = ["pomme", "poire", "banane", "fraise", "abricot"];

for (var i = 0; i < myTableau.length; i++)
{
  console.log("Fruit : "+myTableau[i]);
}
```

Il s'agit ici d'une boucle for tout à fait banale mais il existe une autre syntaxe plus simple d'écriture mais plus lente en exécution : `for ... in`

```
for (var fruit in myTableau)
{
  console.log("Fruit : "+myTableau[fruit]);
}
```

Notez bien qu'avec `for..in` la variable extraite (dans notre cas la variable `fruit`) contient l'indice et non pas la valeur, il faut donc écrire `tableau[indice]` pour afficher la valeur.

`for..in` est l'équivalent de l'instruction foreach` qui existe dans d'autres langages (notamment en PHP).

Il existe une variante `for..each..in` qui ne fonctionne que dans Firefox, elle est donc **à oublier**.

Fonctions de manipulation de données d'un tableau

Certaines de ces fonctions ont un nom que l'on pourra retrouver dans d'autres langages (PHP...) pour des usages identiques mais parfois avec des syntaxes/arguments différents.

Les exemples des fonctions ci-dessous sont donnés à partir du tableau suivant :

```
var fruits = ["pomme", "poire", "banane", "fraise", "abricot"];
```

concat	Réunit deux tableaux.
	Exemple :
	var fruits = ["pomme", "poire", "banane", "fraise", "abricot"];
	var autres = ["sucre", "farine", "oeufs"];
	var ingredients = fruits.concat(autres);

indexOf	Retourne le premier indice pour lequel on trouve l'élément dans un ta-bleau (occurrence) : fruits.indexOf('banane') => retourne 2
lastIndexOf	Retourne le dernier indice de l'occurrence de l'élément dans un tableau : var fruits = ["pomme", "poire", "banane", "fraise", "banane", "abricot"]; fruits.lastIndexOf("banane"); => retourne 4 (position du dernier banane, celui en rouge)
pop	supprime le dernier élément d'un tableau et retourne cet élément. Cette méthode modifie la longueur du tableau : last = fruits.pop(); console.log(last) => retourne 'abricot' et le supprime du tableau

push	ajoute un ou plusieurs éléments à la fin d'un tableau et retourne la nouvelle taille du tableau.
	var nb = fruits.push("mangue", "prune");
	
	Le nouveau tableau sera le suivant :
	
	fruits = ["pomme", "poire", "banane", "fraise", "abricot", 'mangue', 'prune"];
	
	et la variable nb vaudra 7.

shift	Retourne le 1er élément d'un tableau et le supprime. fruits.shift(); => retourne 'pomme'
sort	Tri le tableau en ordre ascendant : fruits.sort(); => retourne abricot, banane, fraise, poire, pomme
split	Découpe une chaîne selon un caractère passé en argument, le résultat de cette « découpe » sera un tableau : Exemple : var ladate = '15/05/2018'; var myTableau = ladate.split('/'); Le tableau myTableau contiendra les valeurs 15, 05 et 2018.

A noter que l'on peut utiliser ces fonctions sans forcément affecter leur retour à une variable.

Il existe beaucoup d'autres fonctions Javascript pour les tableaux : vous pouvez les consulter sur cette page (liste des fonctions dans la colonne de gauche).

Tableaux multidimensionnels

On l'a déjà dit, un tableau peut contenir des chaînes, des entiers, voire les deux, et peut aussi contenir des... tableaux, ce qui donne donc des « tableaux de tableaux » : on appelle ça des tableaux à plusieurs dimensions ou multidimensionnels.

Déclarons un tableau vide :

```
var tab1 = [];
```

puis affectons lui des éléments :

```
tab1[0] = ["pommes", "tomate", "carotte"];
tab1[1] = ["pomme", "poire", "banane"];
```

Vous remarquerez que les éléments ajoutés sont eux-mêmes des tableaux !

Puis, nous pouvons par exemple écrire :

```
console.log(tab1[1][2]);
```

Ce qui affiche «banane » : en effet, on a demandé le second élément (donc la 3ème valeur : banane) de l'élément 1 (donc le tableau des fruits) du tableau tab1.

Dans cet exemple, il n'y a que 2 niveaux de tableaux, mais il peut y avoir 3, 4, 5 niveaux ou plus, c'est illimité sauf que cela devient vite très compliqué à gérer (imaginez pour accéder aux données du 58ème tableau !).

Notez qu'on aurait très bien pu utiliser la forme suivante :

```
var legumes = ["pâtisson", "tomate", "carotte"];
var fruits = ["pomme", "poire", "banane"];

tab1[0] = legumes;
tab1[1] = tableau;
```

Opération et tri sur les tableaux

Opérations diverses sur un tableau non trié

- Sur une structure de données de type tableau, il est possible de faire plusieurs types de traitement, comme par exemple la recherche du minimum ou du maximum, la somme ou le produit ou la moyenne des postes du tableau.
- On peut également vouloir rechercher un élément donné dans un tableau.
- Par exemple, sur un tableau de 35 postes numériques, on désire calculer la somme des postes, et rechercher le plus petit élément.
- Pour le calcul de la somme, on ajoutera le contenu des cases une à une, depuis la première jusqu'à la trente cinquième.
- Pour la recherche du minimum :
- On va supposer que la première case contient le minimum relatif
- On va comparer le contenu de la deuxième case avec le minimum relatif : si celui-ci est inférieur, il deviendra le minimum relatif.
- On recommencera l'opération avec les postes restants

Tri d'un tableau

Un tableau est ordonné lorsqu'il existe une relation d'ordre entre les différentes cases :

On parle de :

- tri croissant si le contenu de la case d'indice i est inférieur ou égal au contenu de la case d'indice i + 1
- tri décroissant si le contenu de la case d'indice i est supérieur ou égal au contenu de la case d'indice i + 1

Plusieurs méthodes de tri existent ; en voici deux exemples sur la base d'un tableau de 30 valeurs numériques VALNUM

Tri croissant par recherche successive des minima

Le principe est le suivant :

- Recherche du minimum dans le tableau de 30 valeurs, et échange du contenu des cases d'indice 1 et d'indice correspondant à la valeur du minimum.
- Application du même principe sur 29 valeurs (30 - première), puis sur 28, puis 27 .... jusqu'au tableau de deux cases.

Visualisation du traitement sur 4 valeurs :

```
  Tableau initial          3   1   4   2
  Après le premier passage     1   3   4   2
  Après le deuxième passage    1   2   4   3
  Après le troisième passage   1   2   3   4
```

Lors du premier passage, on a inversé les cases d'indices 1 et 2. Lors du deuxième passage, le minimum de (3, 7,8) étant 5, on a inversé les cases d'indices 2 et 4. Lors du troisième passage, rien ne s'est passé.

Tri à bulle

Le principe est le suivant :

Le tableau est parcouru en comparant les éléments consécutifs.

S'ils sont mal ordonnés, ces deux éléments sont permutés. On recommence jusqu'à ce qu'il n'y ait plus d'échange.

Visualisation du traitement sur 5 valeurs :

```
  Tableau initial          5   16  14   4   26
  Premier passage      5   14  16   4   26
                           14   4   16  26
  Deuxième passage         5   14  4   16  26
  Troisième passage        5   4   14  16  26
  Quatrième passage        4   5   14  16  26
```

Comme aucune permutation n'a été réalisée, l'algorithme s'arrête.

Méthode : Les éléments sont comparés deux à deux, et on affecte une variable booléenne à vrae si un échange est réalisé.

La condition d'arrêt du traitement est que la variable booléenne soit restée à faux.

Recherche d'un élément sur un tableau trié

Une première manière de vérifier si un mot se trouve dans le dictionnaire consiste à examiner suc-cessivement tous les mots du dictionnaire, du premier au dernier, et à les comparer avec le mot à vérifier.

Ça marche, mais cela risque d'être long : si le mot ne se trouve pas dans le dictionnaire, le pro-gramme ne le saura qu'après 40 000 tours de boucle ! Et même si le mot figure dans le dictionnaire la réponse exigera tout de même en moyenne 20000 tours de boucle. C'est beaucoup, même pour un ordinateur.

Or, il y a une autre manière de chercher, bien plus intelligente journal-ier des, et qui met à profit le fait que dans un dictionnaire, les mots sont triés par ordre alphabétique. D'ailleurs, un être humain qui cherche un mot dans le dictionnaire ne lit jamais tous les mots, du premier au dernier : il utilise lui aussi le fait que les mots soient triés.

Pour une machine, quelle est la manière la plus rationnelle de chercher dans un dictionnaire ? C'est de comparer le mot à vérifier avec le mot qui se trouve pile poil au milieu du dictionnaire. Si le mot à vérifier est antérieur dans l'ordre alphabétique, on sait qu'on devra le chercher dorénavant dans la première moitié du dico. Sinon, on sait maintenant qu'on devra le chercher dans la deuxième moitié.

A partir de là, on prend la moitié de dictionnaire qui nous reste, et on recommence : on compare le mot à chercher avec celui qui se trouve au milieu du morceau de dictionnaire restant. On écarte la mauvaise moitié, et on recommence, et ainsi de suite.

A force de couper notre dictionnaire en deux, puis encore en deux, etc. on va finir par se retrouver avec des morceaux qui ne contiennent plus qu'un seul mot.

Et si on n'est pas tombé sur le bon mot à un moment ou à un autre, c'est que le mot à vérifier ne fait pas partie du dictionnaire.

Regardons ce que cela donne en terme de nombre d'opérations à effectuer, en choisissant le pire cas, celui où le mot est absent du dictionnaire :

Au départ, on cherche le mot parmi 40 000.

Après le test n°1, on ne le cherche plus que parmi 20 000.

Après le test n°2, on ne le cherche plus que parmi 10 000.

Après le test n°3, on ne le cherche plus que parmi 5 000,etc.

Après le test n°15, on ne le cherche plus que parmi 2.

Après le test n°16, on ne le cherche plus que parmi 1.

Et là, on sait que le mot n'existe pas. Moralité : on a obtenu notre réponse en 16 opérations contre 40 000 précédemment !

Il n'y a pas photo sur l'écart de performances entre la technique barbare et la technique futée.

Attention, toutefois, même si c'est évident, **la recherche dichotomique ne peut s'effectuer que sur des éléments préalablement triés**.

Exercices

Exercice 1

Ecrivez un programme permettant de créer un tableau, dont la taille est saisie au clavier.

Ensuite l'utilisateur doit rentrer les différentes valeurs du tableau.

Puis votre programme doit afficher le contenu du tableau.

Exercice 2

Créer le programme qui fournira un menu permettant d'obtenir les fonctionnalités suivantes à partir d'un tableau à une dimension :

- Affichage du contenu de tous les postes du tableau,
- Affichage du contenu d'un poste dont l'index est saisi au clavier,
- Affichage du maximum et de la moyenne des postes du tableau

Ce programme sera structuré de la manière suivante :

- une fonction Gettnlenger pour lire un entier au clavier,
- une fonction InitTab pour créer et initialiser l'instance de tableau : le nombre de postes souhaité sera entré au clavier,
- une fonction SaisieTab pour permettre la saisie des différents postes du tableau,
- une fonction AfficheTab pour afficher tous les postes du tableau,
- une fonction RechercheTab pour afficher le contenu d'un poste de tableau dont le rang est saisi au clavier
- une fonction InfoTab qui affichera le maximum et la moyenne des postes.

Les fonctions seront appelées successivement.

Exercice 3 : Tri d'un tableau

Ecrire le programme qui réalise le tri à bulles.