08 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. Évidemment, 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 où 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) :

```
let myTableau = [];
```

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

```
let myTableau = ["pomme", "poire", "banane"]; // Données de type chaîne
let myTableau = [123, 456, 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 :

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

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

```
let myTableau = new Array(); // Tableau vide
let myTableau = Array(); // Tableau vide
let myTableau = new Array(5); // Tableau vide qui contiendra 5 éléments
let myTableau = Array(5); // Tableau vide qui contiendra 5 éléments
let myTableau = new Array("pomme", "poire", "banane"); // Tableau avec données
let 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 : `let 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 :
`let 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 4e - 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 :

```
let 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 **propriété** `length` (= longueur) retourne le nombre d'éléments dans un tableau :

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

Parcourir un tableau

Les boucles `for` et `for...in` permettent, combinées à la propriété `length`, de parcourir un tableau : on va passer autant de fois que le tableau contient d'éléments, c'est-à-dire tant qu'on n'a pas atteint la longueur du tableau :

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

for (let 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 (let 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 ancienne variante `for each...in` qui ne fonctionnait que dans Firefox, elle est donc **à oublier**.

Bonnes pratiques modernes en JavaScript

- **Préférer la syntaxe avec crochets** pour créer un tableau : `const tab = [];` plutôt que `new Array()`.
- Utiliser `const` pour les tableaux qui ne changent pas de référence, et `let` seulement si tu réaffectes la variable.
- **Ne pas utiliser `for...in` sur les tableaux** : cette syntaxe est plutôt faite pour les objets et parcourt aussi les propriétés héritées. Sur les tableaux, elle peut réserver des surprises.
- Pour parcourir un tableau, préférer :
  - `for` classique avec un index ;
  - `for...of` pour récupérer directement les valeurs ;
  - `tab.forEach(...)` pour appliquer une fonction à chaque élément.
- Rappel : `length` est une **propriété**, pas une fonction : on écrit `tab.length` et non `tab.length()`.

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 :

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

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

indexOf	Retourne le premier indice pour lequel on trouve l'élément dans un tableau (occurrence) : `fruits.indexOf("banane")` ⇒ retourne `2`
lastIndexOf	Retourne le dernier indice de l'occurrence de l'élément dans un tableau : `let fruits = ["pomme", "poire", "banane", "fraise", "banane", "abricot"];` `fruits.lastIndexOf("banane");` ⇒ retourne `4` (position du dernier « banane »)

pop	Supprime le dernier élément d'un tableau et retourne cet élément. Cette méthode modifie la longueur du tableau : `let 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.
	`let 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	Trie le tableau en ordre alphabétique (par défaut) : `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 :
	`let ladate = "15/05/2018";`
	`let myTableau = ladate.split("/");`
	
	Le tableau `myTableau` contiendra les valeurs `"15"`, `"05"` et `"2018"`.

À 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 :

```
let tab1 = [];
```

Puis affectons-lui des éléments :

```
tab1[0] = ["poireau", "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 troisième élément (indice 2 : `"banane"`) de l'élément d'indice 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 5e tableau !).

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

```
let legumes = ["poireau", "tomate", "carotte"];
let fruits = ["pomme", "poire", "banane"];

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

Opérations 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                8   1   7   5
Après le premier passage       1   8   7   5
Après le deuxième passage      1   5   7   8
Après le troisième passage     1   5   7   8
```

Lors du premier passage, on a inversé les cases d'indices 1 et 2. Lors du deuxième passage, le minimum de (5, 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  18  14   4  26
Premier passage         5  14  18   4  26
Deuxième passage        5  14   4  18  26
Troisième passage       5   4  14  18  26
Quatrième passage       4   5  14  18  26
Cinquième passage       4   5  14  18  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 `true` si un échange est réalisé.

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

Recherche d'un élément sur un tableau trié

Une première manière de vérifier si un mot se trouve dans le dictionnaire consiste à examiner successivement 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 programme 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 20 000 tours de boucle. C'est beaucoup, même pour un ordinateur.

Or, il y a une autre manière de chercher, bien plus intelligente pourrait-on dire, 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 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 dictionnaire. Sinon, on sait maintenant qu'on devra le chercher dans la deuxième moitié.

À 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.

À 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 termes 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.

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 - Création et affichage simple

Écrivez un programme permettant de créer un tableau de nombres, dont la taille est saisie au clavier (avec `prompt`).

Ensuite, l'utilisateur doit saisir une à une les différentes valeurs du tableau (toujours avec `prompt`).

Votre programme doit ensuite :

- afficher le contenu du tableau dans la console ;
- afficher le nombre d'éléments du tableau (propriété `length`) ;
- calculer et afficher la somme de toutes les valeurs ;
- calculer et afficher la valeur minimum et la valeur maximum.

Exercice 2 - Moyenne et valeurs supérieures à la moyenne

Reprenez le principe de l'exercice 1 : création d'un tableau de nombres à partir des saisies de l'utilisateur.

À partir de ce tableau, votre programme doit :

- calculer la moyenne des valeurs ;
- afficher cette moyenne ;
- compter combien de valeurs sont strictement supérieures à la moyenne et afficher ce nombre ;
- afficher la liste des valeurs supérieures à la moyenne.

Vous ne devez pas utiliser la fonction `sort()` pour cet exercice.

Exercice 3 - Manipulation d'un tableau avec `push`, `pop` et `shift`

Créez au départ un tableau contenant quelques prénoms, par exemple :

```
let prenoms = ["Alice", "Bob", "Claire"];
```

Écrivez un programme qui :

- affiche le tableau initial ;
- demande à l'utilisateur un prénom à ajouter à la fin du tableau et l'ajoute avec `push()` ;
- supprime le dernier élément du tableau avec `pop()` et affiche le prénom supprimé ;
- supprime le premier élément du tableau avec `shift()` et affiche le prénom supprimé ;
- affiche enfin le tableau final et sa longueur.

Exercice 4 - Tableau multidimensionnel

Créez un tableau multidimensionnel contenant deux sous-tableaux :

- un tableau de légumes (par exemple : `"poireau", "tomate", "carotte"`) ;
- un tableau de fruits (par exemple : `"pomme", "poire", "banane"`).

Par exemple :

```
let tabAliments = [];
tabAliments[0] = ["poireau", "tomate", "carotte"]; // légumes
tabAliments[1] = ["pomme", "poire", "banane"];     // fruits
```

Votre programme doit :

- afficher tous les légumes (ligne 0) dans la console ;
- afficher tous les fruits (ligne 1) dans la console ;
- afficher un élément précis, par exemple le deuxième fruit et le troisième légume ;
- parcourir tout le tableau avec deux boucles imbriquées et afficher chaque élément avec son type (légume ou fruit).

Exercice 5 - Tri par recherche successive des minima

Créez un tableau de nombres (par exemple 10 valeurs au choix, ou saisies au clavier).

Écrivez une fonction qui trie ce tableau par ordre croissant en appliquant le principe du **tri par recherche successive des minima** vu dans le cours :

- on cherche le plus petit élément du tableau ;
- on l'échange avec l'élément en première position ;
- on recommence sur le reste du tableau (à partir de la deuxième position), etc.

Vous ne devez pas utiliser la fonction `sort()` de JavaScript.

Le programme doit afficher :

- le tableau initial ;
- le tableau trié.

Exercice 6 - Recherche dichotomique (optionnel)

À partir d'un tableau de nombres déjà trié par ordre croissant, écrivez une fonction qui recherche une valeur donnée en appliquant le principe de la **recherche dichotomique** (explication dans le cours).

Votre fonction doit :

- prendre en paramètre le tableau et la valeur à chercher ;
- retourner l'indice de la valeur si elle est trouvée ;
- retourner `-1` si la valeur n'est pas présente dans le tableau.

Testez votre fonction avec plusieurs valeurs :

- une valeur présente dans le tableau ;
- une valeur absente ;
- une valeur située au début, au milieu et à la fin du tableau.