Javascript ES 6 > les 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 etc. 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 :

```
Moyenne = (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, mais d'abord retenez qu'on utilise `const` comme déclaration de variable.

Initialisation d'un tableau vide (sans données) :

```
const aTab = [];
```

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

```
const aFruits = ["pomme", "poire", "banane"]; // Données de type chaîne
const aEntiers = [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# ou Java).

On pourrait donc avoir :

```
const aTab = ["pomme", 123, "poire", 456];
```

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

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

```
const aFruits = ["pomme", "poire", "banane", "fraise", "abricot"];
```

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

Remplir un tableau

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

```
const aFruits = [];
aFruits[0] = ["pomme"];
aFruits[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 :

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

Parcourir un tableau

Les boucles `for` et `foreach` permettent, combinées à la fonction `length()` de parcourir un tableau : 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 :

```
const aFruits = ["pomme", "poire", "banane", "fraise", "abricot"];

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

Exemple

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

Notez bien qu'avec `for ... in` la variable extraite (dans notre cas la variable `sFruit`) 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 :

```
const aFruits = ["pomme", "poire", "banane", "fraise", "abricot"];
```

`concat()`	Réunit deux tableaux.
	Exemple :
	`const aAutres = ["sucre", "farine", "oeufs"];`
	`const aIngredients = aFruits.concat(aAutres);`

`indexOf()`	Retourne le premier indice pour lequel on trouve l'élément dans un tableau (occurrence) : `aFruits.indexOf('banane') => retourne 2`
`lastIndexOf()`	Retourne le dernier indice de l'occurrence de l'élément dans un tableau : `aFruits.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 :
	`const iLast = aFruits.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. `const nb = aFruits.push('mangue', 'prune');` Le nouveau tableau contiendra : *pomme, poire, banane, fraise, abricot, mangue, prune* et la variable nb vaudra 7.
`shift()`	Retourne le 1er élément d'un tableau et le supprime. `aFruits.shift(); => retourne 'pomme'`
`sort()`	Tri le tableau en ordre ascendant : `aFruits.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** : `const ladate = '15/05/2018'; const aTableau = ladate.split('/'); Le tableau` `aTableau` `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 :

```
const a1 = [];
```

puis affectons lui des éléments :

```
a1[0] = ["poireau", "tomate", "carotte"];
a1[1] = ["pomme", "poire", "banane"];
```

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

Puis, nous pouvons par exemple écrire :

```
console.log(a1[1][2]);
```

Ce qui affiche *banane* : en effet, on a demandé le second élément (donc la 3ème valeur) de l'élément 1 (donc le tableau des fruits) du tableau `a1`.

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 9ème tableau !).

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

```
const aLegumes = ["poireau", "tomate", "carotte"];
const aFruits = ["pomme", "poire", "banane"];

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

La fonction `map()`

La fonction `map()` (méthode de l'objet `Array()`) permet de parcourir et d'effectuer des opérations ur un tableau multidimensionnel plus simplement qu'avec une boucle.

Exemple

> Déterminez au passage ce que fait cet algo !

Cas utilisant une boucle :

```
const aNumbers = [1, 2, 3, 4];
const aNumbers2 = [];

for (let i = 0; i < aNumbers.length; i++)
{
    aNumbers2[i] = aNumbers[i] * 2;
}

console.log(aNumbers2); // affiche [2, 4, 6, 8]
```

La même chose avec la fonction `map()` :

```
const aNumbers = [1, 2, 3, 4];

const aNumbers = aNumbers.map(function(n) {
   return n * 2;
});

console.log(aNumbers2); // [2, 4, 6, 8]
```

Au besoin ces fonctions peuvent être enchaînées :

```
const aNumbers = [1, 2, 3, 4];
const aNumbers2 = aNumbers.map(function(n){
      return n* 2;
}).map(function(n) {
      return n++;
});

console.log(aNumbers2); // [3, 5, 7, 9]
```

La fonction `filter()`

La fonction `filter()` (méthode de l'objet `Array()`) permet de filtrer un ensemble de données.

Exemple

```
const aNumbers = [1, 2, 3, 4];
const aNumbers2 = [];

for (let i = 0; i < aNumbers.length; i++) {

    if (aNumbers[i] % 2 !== 0) {
          aNumbers2[i] = aNumbers[i] * 2;
    }
}

console.log(aNumbers2); // [2, 6]
```

Avec l'utilisation de `filter()` :

```
const aNumbers = [1, 2, 3, 4];

const aNumbers2 = aNumbers.filter(function(n) {
  return (n % 2 !== 0);
}).map(function(n) {
     return aNumber * 2;
});

console.log(aNumbers2); // [2, 6]
```

Il y a un certain nombre de différences entre les fonctions `map()` et `filter()`. Spécifiquement, au niveau des valeurs de retour, `filter()` ne retourne pas de données modifiées, mais doit retourner une valeur booléenne(vrai/faux). Ceci a un impact sur si les données testées seront prises en compte ou non.

Les fonctions `map()` et `filter()` peuvent être enchaînées les une après les autres, infiniment. Conséquemment, n'hésitez pas à en faire usage afin d'optimiser votre code.

La fonction `reduce()`

+++ TODO +++

La fonction `reduce()` (méthode de l'objet `Array()`) permet d'obtenir un sous-produit des données d'un tableau.

La fonction `forEach ${var}`

+++ TODO : dans tableaux ou dans boucles ? +++

Exercices

Exercice 1

+++ TODO : reprendre exercice sur les tableaux en PHP : mettre aux stagiiares les tableaux en Javascript +++

Exercice 2

Vous n'aimez pas le tableur Excel ? Pas grave, on va le reprogrammer ... en Javascript !

- Créer un tableau comportant 4 lignes et 5 colonnes.
- La dernière ligne et la dernière colonne serviront à afficher les totaux des cellules de chaque ligne et de chaque colonne.
- Saisir au clavier les valeurs des 3 premières lignes.
- Saisir au clavier les valeurs des 4 premières colonnes.
- Calculer le total de chaque ligne et de chaque colonne.

> N'oubliez pas de réfléchir d'abord, à l'aide d'un papier, d'un crayon et de pseudo-code...