Votre Formation Sur Mesure

JAVASCRIPT

ELAN
202 avenue de Colmar - 67100 STRASBOURG
☎ 03 88 30 78 30 🖷 03 88 28 30 69
elan@elan-formation.fr
www.elan-formation.fr

SAS ELAN au capital de 37 000 € -
RCS Strasbourg B 390758241 – SIRET 39075824100041 – Code APE : 8559A
N° déclaration DRTEFP 42670182967 - Cet enregistrement ne vaut pas agrément de l'Etat

SOMMAIRE

I. Introduction 3

II. La pratique 3

1. Avant de commencer 3

2. Les variables 5

3. Les fonctions 6

4. Les conditionnelles 7

a) If … else 7
b) Opérateur ternaire 8
c) Switch … case 8

5. Les boucles 11

a) For 11
b) Do … While 12
c) While 12

6. Les évènements 13

7. Les opérateurs 15

a) Affectation 15
b) Concaténation 15
c) Arithmétique 15
d) Opérateurs logiques 17
e) Opérateurs de comparaison 17

8. Les chaînes de caractères 19

9. Les tableaux 20

10. Les tableaux associatifs 22

I. Introduction

Depuis un certain temps maintenant, les sites web profitent d'une interface de plus en plus dynamique, voire interactive. Un élément apparaît ici lorsqu'on passe la souris, un autre disparaît par là. De plus, avec l'envahissement des mobiles, vos sites préférés s'habillent de fonctionnalités supplémentaires permettant de nouvelles interactions tactiles. Tout ceci est devenu possible grâce aux avancées et évolutions des langages web, dont un en particulier : JavaScript (souvent abrégé JS).

Initialement, JavaScript est un langage qui a été créé en dix jours seulement par Brendan EICH pour la compagnie NETSCAPE. Il s'est inspiré de nombreux langages, notamment de JAVA et C++ mais en simplifiant la syntaxe pour les débutants. JavaScript a pour la première fois été implémenté dans le navigateur web « Netscape Navigator » durant les années 90. Le succès du navigateur a permis l'adoption en masse de JavaScript. Il n'a pourtant été standardisé que deux plus tard ! Aujourd'hui, JavaScript est utilisé pour les pages web interactives mais aussi côté serveur avec l'utilisation par exemple de Node.js.

Pour décrire JavaScript, il s'agit d'un langage de programmation de script qui peut être utilisé de façon procédurale (une série d'étapes à réaliser) ou orienté objet.

Ce support de cours a pour objectif de vous familiariser avec la syntaxe de JavaScript. Vous pouvez tester les exemples suivant dans un fichier JavaScript lié à une page HTML, utiliser une plateforme telle que https://jsfiddle.net ou encore écrire le code directement dans la console de votre navigateur. Chaque navigateur est pourvu d'une console dans laquelle vous pouvez écrire vos instructions JavaScript et elles seront interprétées en direct. Il vous faudra appuyer généralement sur la touche « F12 » de votre clavier pour la faire apparaître.

II. La pratique

1. Avant de commencer

Avant de se lancer, il y a certaines règles, telles les commandements du codeur JavaScript :

- Le code JavaScript doit être écrit dans un fichier à part au format « .js » et intégré dans votre fichier HTML, généralement avant la fermeture de la balise <body>. Écrire son script dans une page HTML alourdit considérablement le poids de cette dernière et donc amènera à un chargement plus long.

```
<!DOCTYPE html>
<html lang="fr">
        ...
            <p>Hello world!</p>
            <script src="mon_script.js"></script>
        </body>
</html>
```

- Ajouter des espaces et des sauts de lignes dans votre code ! Ils ne sont pas interprétés et augmentent grandement la lisibilité du code.

```
//cecodeestindigeste
var a=56,b=25;
if(a<b){console.log(b)}else{console.log(a)};

// ce code est plus agréable à lire
var a = 56, b = 25;
if ( a < b ) {
        console.log( a );
} else {
        console.log( b );
}
```

- Pensez au futur, commentez votre code. Un code qui n'est pas commenté devient très compliqué à reprendre six mois plus tard donc soyez généreux et commentez un maximum.

```
//ceci est un commentaire sur une ligne
var a = 56; // il peut même être placé après du code !

/* ceci est un bloc de commentaire
et il peut continuer sur autant de lignes que vous le souhaitez, jusqu'à sa balise fermante */
var a = 56;
```

- Donnez des noms significatifs à vos variables et fonctions. Cela améliore la compréhension du code. Vous pouvez utiliser toutes les lettres de l'alphabet, majuscules ou minuscules, les nombres de 0 à 9 et l'underscore (le tiret du huit)

```
// Exemples de noms compréhensibles
function afficherImage();
var maVariable01 = 'du texte';

// Mauvais exemples, à ne pas faire
var i = 'hello M. Todd';
function xyz();
```

- Chaque ligne comporte généralement une étape et chaque étape se termine par un point-virgule.

2. Les variables

Les variables sont des symboles qui associent un nom à une valeur. Cette valeur peut être un nombre, du texte, une fonction, etc. Une variable peut voir sa valeur évoluer au fil d'un fichier, d'où son nom. Les variables sont les premiers éléments à être traités avant que le code ne soit exécuté, peu importe où elles sont placées dans le code.

Attention : utilisez toujours le mot clé « var » pour déclarer vos variables la première fois.

```
var nomVar = valeur1, nomVar2 = valeur2 ...,
nomVarN = valeurN;
```

nomvarN
Le nom de la variable, cela peut être n'importe quel identifiant valide.

valeurN
La valeur initiale à affecter à la variable, cela peut être n'importe quelle expression valide. S'il n'y a aucune valeur fournie, la variable vaudra « undefined » (non définie).

Attention : Il n'y a pas d'espace dans le nom d'une variable. Pour simplifier la compréhension, la syntaxe « camelCase » est privilégiée : les mots sont liés sans espaces mais avec une majuscule pour différencier chaque nouveau mot.

```
// Le « CamelCase » ressemble à ceci
var unPeuDeTexte = 'Un peu de texte';
var bonjourToutLeMonde = 'Bonjour tout le monde';
var number = 56;
```

Les variables existent sous différents types :

- Les variables de type numérique - INT ou NUMBER
- Les variables de contenant une chaîne de caractère - STRING
- Les variables contenant soit *true* (vrai) ou *false* (faux) – BOOLEAN

3. Les fonctions

Une fonction est un bloc contenant des instructions, généralement liées à des paramètres. Une fonction permet de préparer une série d'actions, qui se déclencheront une fois la fonction appelée.

```
function nom ( param1, param2,..., paramN ) {
        [instructions]
}
```

nom
Le nom de la fonction.

paramN
Le nom d'un argument à passer à la fonction. Une fonction peut avoir jusqu'à 255 arguments (cela peut varier en fonction des moteurs).

instructions
Les instructions qui constituent le corps de la fonction.

Attention : Les variables déclarées dans une fonction n'existeront plus en dehors de cette dernière. Si vous avez besoin d'utiliser une variable en dehors de la fonction, pensez à la déclarer à l'extérieur de la fonction.

```
var lundi = "Des patates", mardi = "Des pâtes", mercredi = "Risotto aux cèpes", jeudi = "Des frites", vendredi = "Des sushis"; //On déclare toutes les variables


function afficherRepas(jour){ //On déclare la fonction avec un paramètre
        console.log('Le repas d'aujourd'hui est le suivant: '+jour); //la commande « console.log » permet d'afficher ce qui est entre les parenthèses dans la console du navigateur
//On veut que la fonction affiche dans la console une phrase, suivie du contenu du paramètre 'jour' passé en fonction
}
afficherRepas (mercredi); //On exécute la fonction avec une des variables placée en paramètre. Essayez de varier le paramètre avec les différents noms de variables!
```

4. Les conditionnelles

Une expression conditionnelle est une fonction qui effectue différents calculs ou actions, en fonction de l'évaluation d'une condition *booléenne*, à savoir qui renvoie *vrai* ou *faux*.

a) If … else

L'instruction « if » exécute une action si une condition est vraie. Si la condition n'est pas vérifiée, il est possible d'utiliser une autre instruction : « else » pour exécuter alors une autre suite d'instructions.

```
if (condition) {
        instruction1
} else {
        instruction2
}
```

condition
Une expression qui est évaluée à true (vrai) ou false (faux).

instruction1
L'instruction qui est exécutée si la condition est vérifiée (est évaluée à true). Cette instruction peut être n'importe quelle instruction valide, y compris une imbrication d'autres instructions « if ». Pour exécuter plusieurs instructions, on pourra utiliser un bloc d'instructions ({ ... }) qui permet de les regrouper.

instruction2
Si la clause « else » existe, l'instruction qui est exécutée si la condition est évaluée à false. Comme pour la première, cette instruction peut être n'importe quelle instruction valide : une autre instruction « if » imbriquée, un bloc d'instruction, une instruction vide, etc.

```
// un exemple

var nombre1 = 156;
var nombre2 = 350;

if ( nombre1 > nombre2 ) { //On teste si « nombre1 » est plus grand que « nombre2 »
        console.log(nombre1); //Si c'est le cas, on affiche dans la console la valeur de « nombre1 »
} else {
        console.log(nombre2); //Sinon on affiche dans la console la valeur de « nombre2 »
}
```

b) Opérateur ternaire

L'opérateur ternaire est fréquemment utilisé comme raccourci pour la déclaration de l'instruction « if … else », lorsque les instructions sont courtes.

```
condition ? expr1 : expr2
```

condition
Une expression qui est évaluée en un booléen (*true* ou *false*).

expr1, expr2
Des expressions dont la valeur peut être de n'importe quel type.

```
var estMembre = false; //On définit la variable estMembre avec la valeur booléenne "False"
var texte = "Le prix est : " //On définit une phrase qui apparaîtra en console
var resultat = (estMembre ? "15 €" : "30 €") //On teste si estMembre est vrai ou faux, puis on stocke la réponse dans la variable "résultat". "estMembre" étant faux, la valeur stockée sera "30 €".
console.log(texte+resultat); //On affiche le résultat
```

La console affichera alors *"Le prix est : 30 €"*.

c) Switch … case

Une instruction switch commence par évaluer l'expression fournie (cette évaluation ne se produit qu'une fois). Si une correspondance est trouvée, le programme exécutera les instructions associées. Si plusieurs cas de figure correspondent, le premier sera sélectionné (même si les cas sont différents les uns des autres).

Si aucune clause case n'est trouvée, le programme recherche la clause optionnelle « default » et si elle existe, les instructions correspondantes sont exécutées. Si cette clause optionnelle n'est pas utilisée, le programme continuera l'exécution du code présent après l'instruction switch.

L'instruction break peut optionnellement être utilisée pour chaque cas et permet de s'assurer que seules les instructions associées à ce cas seront exécutées. Si break n'est pas utilisé, le programme continuera son exécution avec les instructions suivantes (des autres cas de l'instruction switch).

```
switch (expression) {
        case valeur1:
          // Instructions à exécuter lorsque le résultat de l'expression correspond à valeur1
          instructions1;
          break;
        case valeur2:
          // Instructions à exécuter lorsque le résultat de l'expression correspond à valeur2
          instructions 2;
          break;
        ...
        case valeurN:
          // Instructions à exécuter lorsque le résultat de l'expression à valeurN
          instructionsN;
          break;
        default:
          // Instructions à exécuter lorsqu'aucune des valeurs ne correspond
          instructions_def;
          break;]
}
```

expression
Une expression à comparer avec chacune des clauses « case ».

case expressionN (facultatif)
Une clause qu'on compare avec expression.

default (facultatif)
Une clause exécutée si aucune correspondance n'est trouvée avec les clauses case (et/ou s'il n'y a pas de break pour les clauses case précédentes).

instructionsN
Les instructions à exécuter lorsque l'expression correspond au cas présenté pour cette clause.

instructions_def
Les instructions à exécuter si l'expression ne correspond à aucun cas de figure précédemment décrit.

Exemple:

```
var estCoupable = "coupable" //Testez aussi avec la valeur "innocent" à la place de "coupable".
switch(estCoupable){
        case "coupable":
            console.log("Vous allez sur la case prison, ne passez pas par la case départ, ne touchez pas 20 000 €");
            break;
        case "innocent":
            console.log("Tirez une carte chance !");
            break;
        default:
            console.log("Vous échappez à la justice.");
            break;
}
```

5. Les boucles

Les boucles permettent de répéter des actions simplement et rapidement. Il y a différents types de boucles mais elles se ressemblent toutes au sens où elles répètent une action un certain nombre de fois (ce nombre peut éventuellement être zéro).

Les différents types de boucles permettent d'utiliser différentes façons de commencer et de terminer une boucle. Chaque type de boucle pourra être utilisé en fonction de la situation et du problème que l'on cherche à résoudre.

a) For

Une boucle « for » répète des instructions jusqu'à ce qu'une condition donnée ne soit plus vérifiée.

Elle s'utilise de la façon suivante :

```
for ([expressionInitiale]; [condition]; [Incrément]){
        instruction
}
```

Voici ce qui se passe quand une boucle for s'exécute :

L'expression initiale « expressionInitiale » est exécutée. Généralement, on utilise cette expression pour initialiser une variable « compteur » qui sera utilisée dans la boucle.

L'expression **condition** est évaluée, si elle vaut *true*, les instructions contenues dans la boucle sont exécutées, puis l'expression de mise à jour « **Incrément** » est exécutée.

On recommence ensuite l'évaluation de la condition, etc. Si la valeur de condition est *false*, la boucle se termine.

Attention : si la condition est absente, elle est considérée comme *true*.

```
//Dans la boucle qui suit, on souhaite afficher une table de multiplication
var table = 7; //On cherchera à afficher la table de 7

for ( var i = 0; i < 11; i++ ) {
        //On définit un compteur, une limite afin de pouvoir stopper la boucle et enfin une incrémentation
        console.log(i+' fois '+table+' est égale à '+ i*table);
        //On affiche dans la console les variable, le calcul et du texte pour formater joliment la réponse.
        //Le symbole « + » sert ici à la concaténation mais cela est abordé plus loin.
}
```

b) Do … While

Ce modèle de boucle permet répéter un ensemble d'instructions jusqu'à ce qu'une condition ne soit plus verifiée. Littéralement, *do … while* signifie *faire … tant que*.

```
do {
        instructions
} while (condition)
```

Exemple :

```
var compteur = 10; // On définit un compteur
do { //On indique la liste d'instructions
        console.log('Il reste '+compteur+' étapes avant la fin du compte à rebours') //Ici, on lui demande d'afficher la variable « compteur » accompagnée de texte
        compteur --; // On décrémente « compteur »
} while (compteur > 0); // La boucle continuera jusqu'à ce que « compteur » arrive à 0
```

c) While

« While » permet d'exécuter une instruction tant qu'une condition est vérifiée. Si la condition n'est pas vérifiée, l'instruction n'est pas exécutée.

```
while (condition) {
        instruction
}
```

Exemple :

```
var nbRepas = 0, repas = "biscuit"; //On définit les variables
while(nbRepas < 13) {
        //Tant que « nbRepas » est plus petit que 13
        if (nbRepas > 1) {
          //Le but ici est de définit si « biscuit » prend un « s » ou pas, en fonction du nombre présent
          repas = "biscuits";
        }
        console.log('Vous avez mangé '+nbRepas+' '+repas);
        //On affiche le résultat
        nbRepas++; //On incrémente la variable
}
```

6. Les évènements

On appelle « évènement » toute action effectuée par un utilisateur, qui donnera lieu à une interactivité. L'évènement le plus connu est le « clic », qui est d'ailleurs le seul évènement géré par HTML. Mais grâce à JavaScript, il est possible de lier des fonctions à divers évènements, tels que le passage de la souris sur une zone, l'appui sur une touche du clavier, etc.

Il existe plusieurs méthodes pour déclarer un évènement. La première et la plus ancienne de ces méthodes consiste à pointer vers l'élément cible, à lui associer l'évènement et à lui adjoindre le nom de la fonction sans les parenthèses.

```
document.getElementById('mon_element').onclick=nom_fonction
```

Dans le cas ci-dessus, à chaque clic sur « mon_element », la fonction « nom_fonction » sera exécutée. On peut cependant lui adjoindre une fonction anonyme comme suivant :

```
document.getElementById('mon_element').onclick = function("un_parametre"){
        //du code, du code, etc.
}
```

La seconde méthode s'appelle « addEventListener » et se déclare ainsi :

```
document.getElementById('mon_element').addEventListener('click', nom_de_la_fonction, false);
```

Avec cette méthode, on peut adjoindre plusieurs fonctions pour le même évènement. Vous noterez que par rapport à la méthode précédente, il n'est plus question d'utiliser « onclick » mais « click ».

Il est possible de retirer l'évènement en utilisant « removeEventListener », en précisant la fonction et au cas où plusieurs fonctions sont lancées pour l'élément auquel est rattaché l'évènement, les autres fonctions seront toujours exécutées.

```
document.getElementById('mon_element').removeEventListener('click', nom_de_la_fonction_1, false);
```

Avec cette méthode, il est possible de mettre une fonction anonyme mais comme cette dernière ne contient pas de nom, il ne sera pas possible de retirer l'évènement avec « removeEventListener ».

Pour être utilisé, « removeEventListener » doit comporter l'évènement ainsi que la fonction à retirer.

Exemple d'un évènement affecté à un bouton :

```
//CODE HTML
<button id="bouton">Click me</button>
<p id="demo"></p>

//CODE JAVASCRIPT
var bouton = document.getElementById("bouton");
//On prépare une variable qui cible le bouton
function myFunction() {
        document.getElementById("demo").innerHTML = "Hello World";
        //On prépare la fonction qui va permettre d'insérer du texte dans la balise « p »
}
bouton.addEventListener('click', myFunction, false);
//On met en place le « listener » qui va attendre le clic sur le bouton pour déclencher la fonction
```

7. Les opérateurs

a) Affectation

Le principal opérateur d'affectation est le signe égal (=). Il permet d'affecter une valeur à une variable.

JavaScript reprend les raccourcis sténographiques du langage C en proposant les autres opérateurs d'affectation qui font précéder le signe égal par un opérateur arithmétique (+, -, *, / ou %) ou de concaténation (+).

```
var a = 5;
var b = 4;
console.log(a+b); //Renvoie 9
a += b;
console.log(a); //Renvoie 9

var a = 5;
var b = 4;
console.log(a*b); //Renvoie 20
a *= b;
console.log(a); //Renvoie 20

var a = 5;
var b = 'du texte';
console.log(a+b); //Renvoie « 5du texte »
```

b) Concaténation

Le terme concaténation désigne l'action de mettre bout à bout au moins deux chaînes de caractères. L'opérateur de concaténation « + » permet de joindre ces chaînes, ou des chaînes avec d'autres objets comme des nombres qui sont alors automatiquement convertis en chaîne. Le résultat d'une concaténation est de type *STRING*.

```
var a = 'du texte à concaténer';
var b = ' et encore du texte à concaténer';

console.log(a+b);
//Renvoie une chaîne de caractère unique « du texte à concaténer et encore du texte à concaténer »
```

c) Arithmétique

JavaScript permet aussi de faire de l'arithmétique ! Les opérateurs classiques d'addition « + », de soustraction « - », de multiplication « * », de division « / » fonctionnent uniquement entre deux nombres ou variables de type *NUMBER*.

Attention : dans le cas de l'opérateur « + », si une des parties n'est pas un nombre mais une valeur de type *STRING*, il fonctionnera alors comme un opérateur de concaténation !

```
var a = 5;
var b = 23;
var c ='du texte';

console.log(a+b); // Renvoie 28
console.log(a*b); // Renvoie 115
console.log(b/a); // Renvoie 4,6

console.log(a+c); // Attention ceci renverra du texte !
```

Il existe un dernier opérateur arithmétique qui se nomme modulo « % ». Ce dernier calcul le reste d'une division. Pour rappel, diviser par exemple 7 par 3 revient à faire ce calcul : 7 = 2 x 3 + 1.

Le chiffre sept est composé de deux fois le chiffre trois. Il reste un. Donc sept modulo trois rendra un.

```
var a = 5;
var b = 23;

console.log(a%b);// Renvoie 5
console.log(b%a); // Renvoie 3
```

d) Opérateurs logiques

Les opérateurs logiques sont généralement utilisés avec les valeurs *BOOLEAN*. Vous les trouverez dans des expressions de conditions comme dans « if … else » et sont au nombre de trois :

- expr1 && (et) expr2 : renvoie true si expr1 et expr2 renvoient true. Si l'une d'elles renvoie false alors le tout renverra false.
- expr1 || (ou) expr2 : renvoie true si au moins l'une des deux expressions renvoie true.
- !expr (NON) : renvoie false si son unique opérande peut être converti en true, sinon il renvoie true.

```
var nombre1 = 10;
var nombre2 = 100;
var nombre3 = 50;

if ( nombre2 > nombre3 && nombre1 > nombre3 ){ // Renvoie faux car seule une des conditions est vraie.
        ...
}
if ( nombre2 > nombre3 || nombre1 > nombre3 ){ // Renvoie vrai car au moins une des conditions est vraie.
        ...
}

console.log(!nombre2); // Renvoie « false »
console.log(!nombre2 == !nombre2); // Renvoie « true »
```

e) Opérateurs de comparaison

- Opérateurs d'égalité (==, ===) :

L'opérateur == renvoie la valeur « vrai » si les deux termes à comparer sont équivalents. L'opérateur === compare les valeurs et leur type. Le résultat est donc vrai si les deux valeurs sont égales et de même type.

```
var nombre1 = 10; // Ceci est bien un nombre
var nombre2 = '10'; // Ceci est du texte

console.log(nombre1 == nombre2); // Renvoie true car on ne vérifie pas le type de la variable
console.log(nombre1 === nombre2) ; // Renvoie false

// Attention à ne pas juste mettre un seul symbole '=' sinon vous attribuerez à 'nombre1' la valeur 'de nombre2'
console.log(nombre1 = nombre2); // Renvoie 10
```

- Opérateur d'inégalité (!=, !==) :

L'opérateur != renvoie la valeur *true* si les deux termes à comparer sont différents. L'opérateur !== compare les valeurs et leur type. Le résultat renvoie *true* si les deux valeurs sont différentes ou de type différents.

```
var nombre1 = 10;
var nombre2 = '10';
var nombre3 = 50;

console.log(nombre1 != nombre3); // Renvoie true
console.log(nombre1 != nombre2); // Renvoie false car on ne compare pas le type des variables
console.log(nombre1 !== nombre2); // Renvoie true, car le type des variables n'est pas le même
```

- Opérateur d'ordre (<, <=, >, >=) :

Respectivement inférieur, inférieur ou égal, supérieur, supérieur ou égal. À l'instar de l'opérateur +, les opérateurs de comparaison peuvent comparer numériquement ou alphabétiquement.

```
var nombre1 = 10;
var nombre2 = 50;
var nombre3 = 50;

console.log(nombre1 > nombre2); // Renvoie false
console.log(nombre1 < nombre2); // Renvoie true
console.log(nombre2 >= nombre3); // Renvoie true
```

- Incrémentation et décrémentation (++, --) :

Ces deux opérateurs sont des moyens simples d'incrémenter ou de décrémenter des variables de type *NUMBER*. Si elle n'est pas de type *NUMBER*, elle est automatiquement convertie.

```
var nombre1 = 10;
nombre1++;
console.log(nombre1); // Renvoie 11

var nombre2 = 5;
nombre2--;
console.log(nombre2--); // Renvoie 4
```

8. Les chaînes de caractères

Nous avons précédemment parlé de *STRING* comme étant le nom donné aux variables de type texte. « String » est aussi le nom donné à l'objet global JavaScript permettant de construire des chaînes de caractère.

Les chaînes de caractères possèdent des méthodes permettant de les manipuler telles que « length » pour connaître la longueur de la chaîne ou « slice » permettant d'extraire une partie d'une chaîne de caractères et renvoie donc une nouvelle chaîne. Ces différentes méthodes vous permettront aussi de comparer des chaînes de caractère.

```
var texte = 'Ceci est une chaîne de caractère';
console.log(texte.length); // Renvoie 32

var extrait = texte.slice(5,20);
console.log(extrait); // Renvoie 'est une chaîne'
```

La particularité des chaînes de caractères dans JavaScript, c'est que ce dernier n'est pas un traitement de texte. Malgré ça, il existe une possibilité de gérer les retours à la ligne et autres tabulations si nécessaire. Ces caractères spéciaux sont possible grâce à l'échappement, symbolisé par l'antislash « \ ».

```
//CODE HTML
<div id="zone_texte"></div>

//CODE JAVASCRIPT
var paragraphe = 'Je me sentais si cruellement démuni que j\'envisageai de chercher du \"secours\" au village.\n\r Puis je me souvins que j\'avais une sorte d\'aide très particulière à portée de main.'

document.getElementById("zone_texte").innerHTML = paragraphe;
```

9. Les tableaux

Un tableau, ou « ARRAY » en anglais, est une variable qui contient plusieurs valeurs appelés « items ». Chaque « item » est rangé à la façon d'une liste et est donc accessible par le biais d'un « index » dont la numérotation commence à partir de zéro.

Voici un tableau stockant cinq « items » :

Index	0	1	2	3	4
Item	Valeur1	Valeur2	Valeur3	Valeur4	Valeur5

Un tableau est initialisé dans une variable avec les éléments donnés mis entre crochets et séparés avec une virgule :

```
var tableau = ['pomme', 'banane', 'abricot', 'fraise'];
```

Les tableaux JavaScript sont des objets possédant plusieurs méthodes incorporées pour exécuter des opérations de parcours et de modification. Ni la taille d'un tableau ni les types de ses éléments ne sont fixés car ils peuvent être modifiés à tout moment.

Les éléments de tableau sont simplement des propriétés d'objet. L'exemple suivant va nous permettre de lire l'élément précis d'un tableau, d'en connaître sa longueur ainsi que d'utiliser une boucle pour parcourir un tableau de façon complète.

```
var tableau = ['pomme', 'banane', 'abricot', 'fraise'];


console.log(tableau[1]); // Renvoie 'banane'
console.log(tableau[3]); // Renvoie 'fraise'
console.log(tableau.length); // Renvoie 4

for (var i = 0; i < 4; i++){
        console.log(tableau[i]);
}
// Renvoie 'pomme', 'banane', 'abricot', 'fraise'
```

Tout comme les objets « STRING », les tableaux ont des méthodes permettant leur modification telles que « sort » pour trier les éléments d'un tableau, « push » pour rajouter un élément au tableau.

```
var fruit = ["pommes", "bananes", "Cerises"];
fruit.sort();
console.log(fruit); // ["Cerises", "bananes", "pommes"];

var scores = [1, 2, 10, 21];
scores.sort();
console.log(scores); // [1, 10, 2, 21]
// Attention 10 vient avant 2 selon l'ordre Unicode

var choses = ["mot", "Mot", "1 Mot", "2 Mots"];
choses.sort();
console.log(choses); // ["1 Mot", "2 Mots", "Mot", "mot"]
// En Unicode, les nombres arrivent avant les majuscules, qui elles-mêmes arrivent avant les minuscules.
```

Attention : sachez que si dans un tableau, tous les types d 'éléments sont acceptés, il peut donc aussi contenir un autre tableau et donc devenir un tableau à deux dimensions !

```
var tableau = [['brocoli', 'chou-fleur', 'pates'], ['Recettes', 'Repas', 'Diner'], [1, 2, 3]]; // Ceci est un tableau à plusieurs dimensions

for (var i = 0; i < 3; i++) { // On parcourt le tableau
        for (var j = 0; j < 3; j++) { // On parcourt chaque tableau
            console.log(tableau[i][j]); // On affiche chaque résultat
        }
}
```

10. Les tableaux associatifs

Contrairement aux tableaux vus précédemment, ces derniers se déclarent autrement et ne sont plus soumis à un index numéroté. A la place, choisissez l'index qui devient plutôt un identifiant. Les valeurs quant à elles peuvent toujours être de type « STRING », « ARRAY », « NUMBER », etc.

Les valeurs de ces tableaux se consultent de façon tout aussi différente. L'exemple suivant vous le montre.

```
var association = {
        "prenom" : "Jean Claude",
        "nom" : "Van Damme",
        10 : "Mawashi",
        "catchphrase" : ["Je suis aware", "Un serpent, c'est gentil"]
}

for (var key in association){
        var valeur = association[key];
        console.log(key + " = " + valeur + '<br>');
}
```