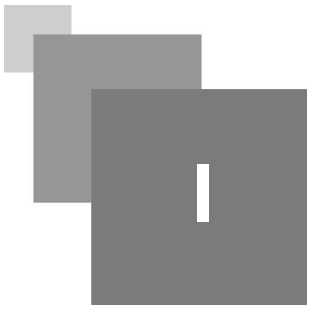
Semaine 3 : Les tableaux et les objets en JavaScript

Cours_ UVCI
PRW2103-1

DIABATE Nabègna,
Université Virtuelle de Côte d'Ivoire

Juillet 2018
1.0

Table des matières



Section 1 : Les tableaux en JavaScript

1. 1. Généralités

Un tableau (*array* en anglais) est un ensemble ordonné de valeurs auxquelles on peut faire référence avec un *nom* et un *indice*.

Par exemple, si on a un tableau `TabEmp` qui contient les noms d'employés indexés par leurs numéros d'employé, on pourrait utiliser TabEmp[1] pour accéder à l'employé n°1, TabEmp[2] pour accéder au deuxième et ainsi de suite.

À la base, un tableau n'est qu'une simple variable. Il existe deux sortes de tableaux :

1. *les tableaux à indices numériques ;*
2. *les tableaux associatifs.*

Dans cette partie du cours, nous allons nous intéresser seulement aux tableaux à indices numériques qui sont des tableaux où chaque valeur est associée à un indice (nombre entier positif). Après avoir abordé les objets en JavaScript, vous comprendrez mieux la notion de tableaux associatifs.

JavaScript ne possède pas de type particulier pour représenter un tableau de données. En revanche, il est possible d'utiliser l'objet natif `Array` ainsi que ses méthodes pour manipuler des tableaux. L'objet `Array` possède plusieurs méthodes (nous les verrons dans la suite de la leçon) qui permettent de manipuler les tableaux pour les fusionner, les inverser, les trier, etc. Il possède une propriété de longueur ainsi que d'autres propriétés qui peuvent être utilisées avec les expressions rationnelles.

2. 2. Déclaration et utilisation des tableaux

Déclaration d'un tableau

Les instructions qui suivent sont équivalentes et permettent de créer le même tableau :

```
1 var arr = new Array(élément0, élément1, ..., élémentN);
2 var arr = Array(élément0, élément1, ..., élémentN);
3 var arr = [élément0, élément1, ..., élémentN];
```

Parmi les instructions précédentes, une utilise des crochets, on appelle ceci un « littéral de tableau » ou un « *initialisateur de tableau* ». Cette notation est plus courte que les autres et est souvent préférée pour sa lisibilité.

élément0, élément1, ..., élémentN est une liste de valeurs qui formeront les éléments du tableau. Lorsque ces valeurs sont définies, le tableau initialisera la valeur des éléments correspondants. La propriété `length` du tableau permet de connaître le nombre d'arguments du tableau.

Exemple de tableau :

```
1 var TabEmp= new Array('Christophe', 'Sarah', 'Carole', 'Alex', 'Nicolas',
  'Sandrine'); // première méthode
2 var TabEmp= ['Christophe', 'Sarah', 'Carole', 'Alex', 'Nicolas', 'Sandrine']; //
```

```
seconde méthode
```

Accès aux valeurs

Pour accéder aux valeurs d'un tableau à indices numériques, la seule possibilité est de passer par l'indice de chacune des valeurs contenues dans ce tableau. La numérotation des indices commence par 0 (zéro).

```
1 document.write(TabEmp[0]); // Affiche "Christophe"
2 document.write(TabEmp[4]); // Affiche "Nicolas"
3 document.write(TabEmp[2]); // Affiche "Carole"
```

Pour lister l'intégralité du tableau, il nous faut utiliser une boucle. Il va nous être utile de connaître la "longueur" du tableau (le nombre d'indice qu'il possède). Pour cela, on fait appel à la méthode *`length`* de l'objet *`Array`*. Ainsi, on accède aux valeurs de notre tableau grâce à ses indices comme ceci :

```
1 for (i = 0; i <= TabEmp.length-1; i++)
2 {
3   document.write(i+" => "+TabEmp[i]); // On affiche chaque couples indice =>
  valeur
4 }
```

Remplir un tableau

Comme cela a été le cas dans l'exemple de la déclaration d'un tableau, il est possible de remplir un tableau directement à sa création.

Il est aussi possible de remplir un tableau en affectant des valeurs à ses éléments. Par exemple :

```
1 var emp = [];
2 emp[0] = "Casey Jones";
3 emp[1] = "Phil Lesh";
4 emp[2] = "August West";
```

Cependant, lorsqu'un tableau est déjà rempli et que nous assignons une nouvelle valeur à une cellule, l'ancienne valeur de la cellule sera écrasée et remplacée par la nouvelle valeur.

A l'aide des bucles, nous pouvons remplir de façon dynamique les tableaux de la même façon qu'on puisse les parcourir. Pour remplir un tableau avec une seule valeur (13 par exemple), on utilise une boucle.

```
1 var a = 13;
2 var long_tableau = 10;
3 var mon_tableau = new Array();
4
5 for (i = 0; i <= long_tableau; i++)
6 {
7   mon_tableau[i] = a;
8 }
```

Il existe des tableaux de tableaux de tableau... c'est a dire des tableaux qui contienne des tableaux ; on parlera alors de *tableaux multidimensionnels*. Ainsi, ceci est tout à fait faisable :

```
1 var mon_tableau = new Array('Christophe', new Array('Sarah', 'Carole', 'Alex',
  'Nicolas', 'Sandrine'));
2 document.write(mon_tableau[1][0]); // Affiche "Sarah"
```

3. Exercice

Exercice

1) Lesquelles des déclarations de tableaux suivantes sont justes ?

- ☐ var monTableau = new Array(élément0, élément1, ..., élémentN);
- ☐ var myTab = Array(élément0, élément1, ..., élémentN);
- ☐ Var new monTableau = Array(élément0, élément1, ..., élémentN);
- ☐ var maTable = [élément0, élément1, ..., élémentN];

Exercice

2) Un Tableau JavaScript peut(il être considéré comme un Objet ?

- ◯ Oui
- ◯ Non

Exercice

3) soit le code de définition tu tableau suivant :

```
1 var monTableau = new Array(élément0, élément1, ..., élémentN);
```

De combien d'éléments est constitué le tableau ? (écrire en majuscule et sans espace !)

Exercice

4) Soit le tableau défini par le code suivant :

```
1 var TabEmp= new Array('Christophe', 'Sarah', 'Carole', 'Alex',...,'Nicolas',
  'Sandrine');
```

Complétez le code suivant pour permettre d'afficher la longueur total du tableau à le fin de la saisie.

document.write(TabEmp.);

Exercice

Soit le code JavaScript suivant :

```
1 var TabEmp= new Array('Christophe', 'Sarah', 'Carole', 'Alex','Nicolas',
  'Sandrine');
2 document.write(TabEmp[4]);
```

Que fait la seconde ligne de ce code ?

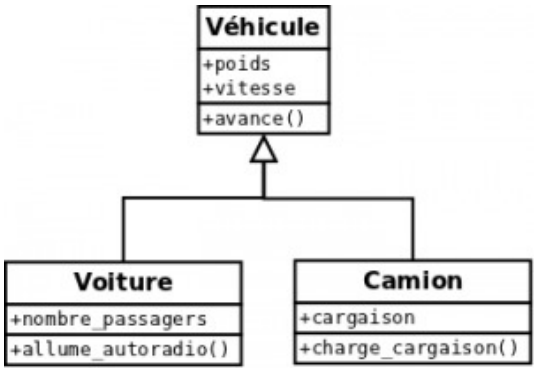
- ☐ Affiche"Nicolas"
- ☐ Affiche "Alex"
- ☐ Affiche "Sandrine"

Section 2 : Les Objets

JavaScript a été construit autour du concept d'*objets* , et son fonctionnement même se base sur l'utilisation d'objets.

En effet, en JavaScript, quasiment tout est avant tout objet :

- Les chaînes de caractères, nombres et booléens peuvent être des objets (ou des valeurs primitives traitées comme des objets) ;
- Les fonctions sont des objets ;
- Les tableaux sont des objets ;
- Les dates sont des objets ;
- Les expressions régulières sont des objets.



La compréhension de ce que sont les objets et comment ils fonctionnent est bien utile afin de bien comprendre le JavaScript.

Dans la « vraie » vie, par exemple, un crayon est *un objet*.

Il existe des crayons de différentes couleurs et de différentes tailles ou formes. La couleur, taille ou forme d'un crayon vont être *des propriétés*.

Tous les crayons vont posséder les mêmes propriétés (tous les crayons possèdent une taille) mais *les valeurs* associées à ces propriétés vont être différentes (chaque crayon peut avoir une taille différente des autres).

De plus, un crayon sert à écrire. "Ecrire" est donc ici une *fonction de l'objet* crayon ou encore ce que nous allons désormais appeler *une méthode*.

1. 1. Les Objets et les propriétés

Un objet JavaScript possède donc plusieurs propriétés qui lui sont associées. Une propriété peut être vue comme une variable attachée à l'objet.

Ainsi, les propriétés d'un objet sont variables. Les propriétés d'un objet représentent ses caractéristiques et on peut y accéder avec une notation utilisant le point « . », de la façon suivante :

```
1 nomObjet.nomPropriete
```

Les objets et les propriétés respectent les mêmes critères de nommage que les variables : Leur nom sont sensibles à la casse.

Il existe plusieurs manières de définir un objet (Cf. cours de Programmation Orientée Objets) en JavaScript (*https://www.pierre-giraud.com/javascript/cours-complet/javascript-creation-objets.php*). La plus connu que nous allons utiliser ici consiste à utiliser le mot-clé `New` et le constructeur `Object()` et en définissant par exemple plusieurs propriétés pour notre objet.

On peut définir une *propriété* en lui affectant une *valeur*. Ainsi, si on crée un objet `maVoiture` et qu'on lui donne les propriétés `fabricant`, `modèle`, et `année` :

```
var maVoiture = new Object();
maVoiture.fabricant = "Ford";
maVoiture.modèle = "Mustang";
maVoiture.année = 1969;
```

Les propriétés d'un objet qui n'ont pas été affectées auront la valeur *undefined* (et non *null*).

```
maVoiture.sansPropriete; // undefined
```

On peut aussi définir ou accéder à des propriétés JavaScript en utilisant une notation avec les crochets. Les objets sont parfois appelés « *tableaux associatifs* ». Cela peut se comprendre car chaque propriété est associée avec une chaîne de caractères (qui se comporte et qu'on peut traiter comme l'indice dans un tableau) qui permet d'y accéder. Ainsi, par exemple, on peut accéder aux propriétés de l'objet `maVoiture` de la façon suivante :

```
maVoiture["fabricant"] = "Ford";
maVoiture["modèle"] = "Mustang";
maVoiture["année"] = 1969;
```

Les initialisateurs d'objets

On peut créer des objets avec une fonction qui est un constructeur mais on peut aussi créer des objets avec des initialisateurs d'objets. On appelle parfois cette syntaxe *la notation littérale*.

La syntaxe utilisée avec les initialisateurs d'objets est la suivante :

```
var obj = { propriete_1:   valeur_1,   // propriete_# peut être un identifiant
            propriete_2:   valeur_2,   // ou un nombre
            // ...,
            "propriete n": valeur_n }; // ou une chaîne
```

où on a *obj* le nom de l'objet qu'on souhaite créer et chaque *propriété_i* un identifiant (que ce soit un nom, un nombre ou une chaîne de caractères) et chaque *valeur_i* une expression dont la valeur sera affectée à la propriété *propriété_i*.

Les initialisateurs d'objets sont des expressions et chaque initialisateur entraîne la création d'un nouvel objet dans l'instruction pour laquelle il est exécuté.

Les objets sont créés de la même façon qu'avec *new Object()*, les objets créés à partir d'une expression littérale seront des instances d'*Object*.

L'instruction suivante crée un objet et l'affecte à une variable *x* si et seulement si l'expression *cond* est vraie :

```
if (cond) var x = {emplacement: "le monde"};
```

Dans l'exemple suivant, on crée un objet *maHonda* avec trois propriétés. La propriété *moteur* est également un objet avec ses propres propriétés.

```
var maHonda = {couleur: "rouge", roue: 4, moteur: {cylindres: 4, taille: 2.2}};
```

De la même façon, on pourra utiliser des initialisateurs pour créer des tableaux

Les constructeurs

Source : Mozilla Developer (https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser_les_objets)

On peut créer des objets d'une autre façon, en suivant deux étapes :

1. On définit une fonction qui sera un constructeur définissant le type de l'objet. La convention, pour nommer les constructeurs, est d'utiliser une majuscule comme première lettre pour l'identifiant de la fonction.

2. On crée une instance de l'objet avec `new`.

Pour définir le type d'un objet, *on crée une fonction* qui définit le nom de ce type et les propriétés et méthodes des instances. Ainsi, si on souhaite créer un type d'objet pour représenter des voitures, on pourra nommer ce type *voiture*, et il pourra avoir des propriétés pour le `fabricant`, le `modèle` et l'`année`. Pour ce faire, on pourra écrire la fonction suivante :

```
function Voiture(fabricant, modèle, année) {
  this.fabricant = fabricant;
  this.modele = modele;
  this.annee = annee;
}
```

On voit ici qu'on utilise le mot-clé *this* pour affecter des valeurs aux propriétés d'un objet en fonction des valeurs passées en arguments de la fonction.

JavaScript possède un mot-clé spécial *this*, qui peut être utiliser à l'intérieur d'une méthode pour faire référence à l'objet courant.

En général, this fait référence à l'objet appelant de la méthode. Vous pouvez avoir plus d'information sur le mot-clef *this* sur *mozilla developer* (https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/L_op%C3%A

On peut désormais créer un objet `maVoiture` de la façon suivante :

```
var maVoiture = new Voiture("Eagle", "Talon TSi", 1993);
```

Cette instruction crée un objet *maVoiture* et lui affecte les valeurs fournies pour ses propriétés. On obtient donc `maVoiture.fabricant` qui sera la chaîne de caractères "Eagle", `maVoiture.année` qui sera l'entier 1993, et ainsi de suite.

Grâce à ce constructeur, on peut ensuite créer autant d'objets voiture que nécessaire. Par exemple :

```
var voitureMorgan = new Voiture("Audi", "A3", 2005);
var voitureMax = new Voiture("Mazda", "Miata", 1990);
```

Comme nous l'avons vu précédemment, un objet peut avoir une propriété qui est elle-même un objet. Ainsi, si on définit un type d'objet personne de cette façon :

```
function Personne(nom, âge, sexe) {
  this.nom = nom;
  this.age = age;
  this.sexe = sexe;
}
```

et qu'on instancie deux nouveaux objets personne avec

```
var max = new Personne("Max Gun", 33, "M");
var morguy = new Personne("Morgan Sousbrouille", 39, "M");
```

On pourra réécrire la fonction de définition pour le type voiture pour inclure une propriété *propriétaire* qui est représentée par un objet *personne* :

```
function Voiture(fabricant, modèle, année, propriétaire) {
  this.fabricant = fabricant;
  this.modele = modele;
  this.annee = annee;
  this.propriétaire = propriétaire;
}
```

Pour instancier des nouveaux objets, on pourra donc utiliser :

```
var voiture1 = new Voiture("Mazda", "Miata", 1993, max);
var voiture2 = new Voiture("Audi", "A3", 2005, morguy);
```

Le dernier argument n'est pas une chaîne de caractères ou une valeur numérique mais bien un *objet*.

Les objets *max* et *morguy* sont passés en arguments pour représenter les propriétaires. Ainsi, si on veut obtenir le nom du propriétaire pour voiture2, on peut accéder à la propriété de la façon suivante :

```
voiture2.propriétaire.nom
```

Remarque

Il est toujours possible d'ajouter une propriété à un objet défini précédemment. Par exemple, on peut ajouter une propriété à l'objet `voiture1` avec l'instruction :

```
voiture1.couleur = "noir";
```

On ajoute ainsi une propriété `couleur` à `voiture1`, et on lui affecte une valeur "*noir*". Cependant, cela n'affecte pas les autres objets `voiture`. Pour ajouter une nouvelle propriété à tous les objets, il faudra ajouter la propriété au constructeur `voiture`.

2. 2. Les méthodes

Une méthode est une fonction associée à un objet. Autrement dit, une méthode est une propriété d'un objet qui est une fonction. Les méthodes sont définies comme des fonctions normales et sont affectées à des propriétés d'un objet.

Les méthodes des objets du navigateur sont des fonctions définies à l'avance par les normes HTML, on ne peut donc pas les modifier, il est toutefois possible de créer une méthode personnelle pour un objet que l'on a créé soi-même. Prenons par exemple une page HTML, elle est composée d'un objet appelé *document*. L'objet `document` a par exemple la méthode *write()* qui lui est associée et qui permet de modifier le contenu de la page HTML en affichant du texte. Une méthode s'appelle un peu comme une propriété, c'est-à-dire de la manière suivante :

« *window.objet1.objet2.methode()* »

Dans le cas de la méthode *write()*, l'appel se fait comme suit :

```
window.document.write()
```

On peut définir nos propres méthodes de deux façons :

```
//1. Une fonction existe déjà et on veux l'utiliser comme méthode d'un Objet
nomObjet.nomMéthode = nomFonction;

//2. Définir la méthode directement dans l'Objet
var monObj = {
  maMéthode: function(params) {
    // ...faire quelque chose
  }
};
```

Dans le premier cas `nomObjet` est un objet existant, `nomMéthode` est le nom de la propriété à laquelle on souhaite affecter la méthode et `nomFonction` le nom de la fonction existante.

On peut ensuite appeler la méthode sur l'objet :

```
object.nomMéthode(paramètres);
```

Complément

On peut définir des méthodes pour un type d'objet en incluant la définition de la méthode dans le constructeur. Par exemple, on peut définir une fonction qui mettrait en forme et qui afficherait les propriétés d'un objet `voiture`. Par exemple :

```
function afficheVoiture() {
  var résultat = "Une " + this.fabricant + " " + this.modèle
    + " de cette année " + this.année;
  console.log(résultat); // nous aborderons cette syntaxe plutard
}
```

On peut ensuite ajouter cette fonction comme méthode dans le constructeur avec cette instruction :

```
this.afficheVoiture = afficheVoiture;
```

La définition complète serait donc :

```
function Voiture(fabricant, modèle, année, propriétaire) {
  this.fabricant = fabricant;
  this.modèle = modèle;
  this.année = année;
  this.propriétaire = propriétaire;
  this.afficheVoiture = afficheVoiture;
}
```

On pourra donc ensuite appeler la méthode *afficheVoiture* pour chaque objet de ce type :

```
voiture1.afficheVoiture();
voiture2.afficheVoiture();
```

3. Exercice

Exercice

1) Quelle est l'affirmation correcte ?

- ☐ Une propriété est une variable transmise à un objet
- ☐ Une méthode est une variable interne à un objet
- ☐ Une propriété est une fonction interne à un objet
- ☐ Une méthode est une fonction interne à un objet

Exercice

2) Une méthode est une fonction associée à un objet

- ○ Vrai
- ○ Faux

Exercice

3) Quelle est la déclaration syntaxiquement correcte ?

- ☐ var a = { maj: 'A': min: 'a' };
- ☐ var b = { maj: 'B'; min: 'b' };
- ☐ var c = { maj: 'C', min: 'c' };

Exercice

4) Soit le code JavaScript suivant :

```
1 var maVoiture = new Object();
2 maVoiture.fabricant = "Toyota";
3 maVoiture.modèle = "Familiale";
4 maVoiture.année = 1999;
```

Quel est alors le contenu de maVoiture.Immatriculation;

Section 3 : Les objets natifs du JavaScript

JavaScript possède un certain nombre d'objets natifs. C'est le cas des `String` , `Number` , `Math` etc. Vous trouverez la liste exhaustive des objets globaux sur *cette page du MDN*.

C'est grâce à ces objets prédéfinis que nous pouvons invoquer des méthodes sans avoir à les définir préalablement.

1. 1. L'objet String

`String` est un mot anglais qui signifie "*chaîne*". L'Objet *`String`* de JavaScript est un objet qui contient un certains nombre de *propretés* et de *méthodes* permettant de manipuler et de gérer *les chaînes de caractères* en JavaScript.

Cet objet contient une propriété fondamentale qui permet de retourner la longueur de la chaîne de caractère (string) : la propriété *`length`* (longueur en anglais).

Syntaxe :

nomVariable = nom_de_la_chaine.length;

ou

nomVariable = ('chaine de caracteres').length;

```
1   var myString = 'Apprendre le javascript';
2   document.write(myString.length);
3   // affichera 23
```

L'Objet `String` a cependant plusieurs méthodes très utiles pour la manipulation des chaînes de caractères. Pour exploiter ces différentes méthodes nous utiliserons la notation pointée : *`myString.nomMethode()`* permettant d'exécuter la méthode appelée.

Voici donc quelques-unes des méthodes de l'objet string en JavaScript :

- Les méthodes toLowerCase() et toUpperCase()

La méthode *`toLowerCase()`* permet de passer toute une chaîne de caractères en *minuscule*.

La méthode *`toUpperCase()`* permet quant à elle de passer une chaîne en *majuscule*.

```
1   var myString = 'JavaScript';
2   document.write(myString.toLowerCase());
3   // Affiche javascript
4
5   // Inversement, la méthode toUpperCase() passe la chaîne en majuscule
6
7   document.write(myString.toUpperCase());
8   // Affiche JAVASCRIPT
```

- *La méthode charAt()*

Elle permet d'accéder à un caractère dans un objet de type String. Elle prend en paramètre la position la position du caractère à atteindre.

Comme pour les tableaux, la première position d'un élément a pour valeur *0*. Ainsi donc, le dernier élément aura donc logiquement comme position *`myString.length - 1`*.

```
var myString = 'Le javascript, langage objet';
document.write(myString.charAt(5));
// affichera v
```

- *La méthode charCodeAt().*

Elle permet d'obtenir la valeur numérique décimale du code correspondant au caractère d'une chaîne. Sa table de correspondance est la *table ASCII*.

Elle prend comme argument l'indice du caractère que l'on souhaite coder ou directement le caractère le caractère lui-même !

Exemple :

```
var myString = 'Le javascript, langage objet';
document.write(myString.charCodeAt(0));
// Affiche 76

// La méthode inverse renvoie la chaîne de caractères correspondant au code
var myString = String.fromCharCode(76,76,76);
document.write(myString);
// Affiche LLL
```

- *La méthode substring()*

Si *`String`* signifie en anglais "chaîne" alors *`substring`* fera évidement penser à "sous-chaîne" !

Cette méthode permet d'extraire une chaîne de caractères dans une autre chaîne de caractère. Elle attend 2 arguments, à savoir *l'indice du premier caractère* et *l'indice du dernier*. En ne fournissant qu'*un seul argument à la méthode, la chaîne retournée sera celle commençant à l'indice donné.*

Exemples :

```
var myString = 'L\'objet String';
var LeMot= myString.substring(2,7);
document.write(LeMot);
// Affiche "objet"

var LeMot= myString.substring(2);
document.write(LeMot);
// Affiche "objet String"
```

La méthode replace()

Elle permettre de rechercher un caractère (ou une expression) dans une chaîne de caractères et de le(la) remplacer par un(e) autre.

Cette méthode a besoin de deux arguments : le caractère (ou la chaine) à remplacer et le caractère (ou la chaine) de remplacement.

```
var myString = 'Je me forme au langage JavaScript';

//On replace le mot JavaScript par PHP/MySQL
```

```
var maPhrase = myString.replace("JavaScript","PHP/MySQL");
document.write(maPhrase);
// Affichera alors 'Je me forme au langage PHP/MySQL'
```

- *Les méthodes indexOf() et lastIndexOf()*

La méthode *`indexOf()`* retourne la première position à laquelle un caractère (ou une séquence de caractères) donné a été retrouvé dans une chaîne (on parlera de première occurrence).

La méthode *`lastIndexOf()`*,elle, retourne la dernière position à laquelle un caractère (ou une séquence de caractères) donné a été retrouvé dans une chaîne.

Cette méthode peut être pratique pour récupérer l'extension de l'adresse d'un site web par exemple.

```
var myUrl = 'www.analyste-programmeur.com';

// On recherche la position de la dernière occurrence du point
var posLastPoint = myUrl.lastIndexOf('.');

// On affiche l'extension à l'aide la méthode substring()
document.write(myUrl.substring(posLastPoint));

// Affiche ".com"

// Tester l'existence d'une expression
if(myUrl.indexOf('programme') != -1){
  document.write('Chaîne trouvée');
} else {
  document.write('Chaîne non trouvée');
}
```

Si l'expression n'est pas trouvée, *la méthode renvoie -1.*

L'écriture générale de cette méthode est *`myString.lastIndexOf(expression,depart)`*. En renseignant l'argument `depart` (qui est facultatif), la recherche de l'expression commence à partir de la position `depart`. Si on ne fournit pas, la recherche commence en début de chaîne.

- *La méthode trim()*

La méthode *`trim()`* supprime les espaces superflus en début et en fin de chaîne. Cela peut être très pratique lorsqu'on veut nettoyer et mettre en forme des données envoyées par l'utilisateur par exemple.

Cette méthode n'a pas besoin d'argument pour fonctionner.

```
var myString = '      Je me forme au langage JavaScript     ';
//Contient des espace en debut et en fin

document.write(myString.trim());
// Affichera alors 'Je me forme au langage JavaScript'
```

Complément : Autres méthodes de l'objet String

Il existe plusieurs autres méthodes de l'objet string.

Le fichier suivant est a téléchargé et liste toute une panoplie de méthode de l'objet String que vous pourrez utiliser sans modération !

Aussi, vous pourrez trouver des détails sur la l'objet String sur le site *Commentçamarche (https://www.commentcamarche.com/contents/586-javascript-l-objet-string)*

2. 2. L'objet Number

L'objet *Number* sert à gérer les nombres en JavaScript. Il possède cinq propriétés non modifiables qui lui appartiennent exclusivement :

- *MAX_VALUE*, correspond au plus grand nombre enregistrable ;
- *MIN_VALUE*, correspond au plus petit nombre enregistrable ;
- *NaN*, nombre invalide, représentant une valeur qui n'est pas un nombre ;
- *NEGATIVE_INFINITY*, correspond à un mombre infiniment petit (inférieur à -MAX_VALUE),
- *POSITIVE_INFINITY*, correspond à un mombre infiniment grand (supérieur à MAX_VALUE).

L'Objet *Number* à aussi plusieurs méthodes (environ une dizaine) dont nous allons voir quelques-unes :

- La méthode toString()

Cette méthode permet de transformer un nombre en une chaîne de caractère c'est à en String ; elle n'a pas besoin d'argument pour fonctionner.

```
1 var myNombre = 2018;
2 //  MyNombre est un entier
3
4 var myString2=myNombre.toString();
5 //myString2 est une chaîne = '2018'
```

- La méthode toFixed()

Cette méthode a deux rôles : d'abord préciser le nombre de décimales d'un nombre puis ensuite le transforme en une chaîne de caractère. Il prend comme argument le nombre de décimales que le souhaite.

```
1 var myNombre = 10.4531;
2 //  MyNombre est un entier
3
4 var myNombre2=myNombre.toFixed(2);
5 //myNombre2 est une chaîne = '10.45'
```

- La méthode toPrecision()

La méthode *toPrecision()* permet de choisir la « *longueur* » du nombre (c'est à dire le nombre de chiffres) qui doit être retourné et va le retourner sous forme de chaîne de caractères.

Cette méthode va prendre en argument la taille voulue.

```
1 var myNombre = 10.4531;
2 //  MyNombre est un entier
3
4 var myNombre2=myNombre.toPrecision(3);
5 //myNombre2 = 10.4 et typeof(myNombre2)=String
```

- La méthode toExponential()

Elle permet de choisir le nombre de décimales d'un nombre et le retourne sous forme de chaîne de caractères, en utilisant une notation sous forme de nombre exponentiel. Elle prend en argument le nombre de décimales souhaitées.

```
var myNombre = 10.4531;
//  MyNombre est un entier

var myNombre2=myNombre.toExponential(2);
//myNombre2 = 1.05e+1 et typeof(myNombre2)=String
```

- Quelques méthodes globales intéressantes applicables a Number

Le JavaScript possède des fonctions globales qui peuvent être utilisées avec tous les types de données. Certaines de ces fonctions peuvent s'avérer intéressantes à utiliser en tant que méthodes avec des données de type Number :

Méthode	Description
`Number.parseFloat()`	Analyse un argument qui est une chaîne de caractères et renvoie un nombre décimal. Cette méthode est équivalente à la fonction `parseFloat()`.
`Number.parseInt()`	Analyse un argument qui est une chaîne de caractères et renvoie un entier exprimé dans une base donnée. Cette méthode est équivalente à la fonction `parseInt()`.
`Number.isFinite()`	Détermine si la valeur passée en argument est un nombre fini.
`Number.isInteger()`	Détermine si la valeur passée en argument est un nombre entier.
`Number.isNaN()`	Détermine si la valeur passée en argument est `NaN`. Cette version est plus robuste que la fonction globale `isNaN()`.
`Number.isSafeInteger()`	Détermine si la valeur fournie est un nombre qu'il est possible de représenter comme un entier sans perdre d'information.

3. 3. L'objet Array

L'objet *`Array`* possède une dizaine de méthodes qui sont pour la plupart très puissantes et permettent d'effectuer toutes sortes d'opérations sur les tableaux.

- La méthode concat()

Elle permet de regrouper deux (02) tableaux en renvoyant le résultat sous forme d'un troisième tableau.

Ce troisième tableau représente donc la fusion des deux (02) autres tableaux.

```
var monTab1 = new Array('javascript','php');
var monTab2 = new Array('html','ajax');
var monTab3 = monTab1.concat(monTab2);

document.write(monTab3.join(', '));
// affiche "javascript, php, html, ajax"
```

- La méthode join()

Déjà énoncée dans l'exemple précédent, appliquée à un tableau, la méthode *`join()`* permet de lire le contenu de ce tableau comme une chaîne de caractères. Cette méthode attend un *séparateur* comme argument et s'avère être très pratique pour lister le contenu d'un tableau.

```
var maTab = new Array('html','ajax','php','javascript');
```

```
2
3   document.write(maTab.join(' | '));
4   // affiche "html | ajax | php | javascript"
```

- *Ajoute et suppression d'éléments en fin de tableau : push() et pop()*

La méthode `pop()` supprime le dernier élément d'un tableau et ne retourne aucun résultat.

Seul le tableau sur lequel est appliquée cette méthode est affecté.

La méthode `push()` quant à elle ajoute des éléments à la fin du tableau. Elle attend en paramètre une ou plusieurs valeurs. Elle ne retourne aucun résultat mais affecte le tableau sur lequel elle est appelée.

```
1   // 1) pop()
2   var maTab = new Array('html','ajax','php','javascript');
3   maTab.pop(); // suppression du dernier élément du tableau tab
4
5   document.write(maTab.join(' | '));
6   // affiche "html | ajax | php"
7
8   // 2) push()
9   var monTab = new Array('html','ajax','php','javascript');
10  monTab.push('css','sql'); // ajoute des éléments au tableau monTab
11
12  document.write(monTab.join(' | '));
13  // affiche "html | ajax | php | javascript | css | sql"
14
```

- *La méthode unshift()*

Elle ajoute quant à elle des éléments au début du tableau et décale les indices. Elle ne retourne aucun résultat et affecte directement le tableau concerné.

```
1   var tab = new Array('html','ajax','php','javascript');
2   tab.unshift('mysql','css'); // ajoute les élément mysql et css au tableau
3
4   document.write(tab.join(' | '));
5   // affiche "mysql | css | html | ajax | javascript"
```

- *La méthode reverse()*

La méthode reverse() inverse l'ordre des éléments du tableau. Ce tableau est directement affecté et la méthode ne retourne aucun résultat.

```
1   var matab = new Array('html','ajax','php','javascript');
2   matab.reverse(); // inverse l'ordre du tableau matab
3
4   document.write(matab.join(' | '));
5   // affiche "javascript | php | ajax | html"
```

- *La méthode shift()*

Elle joue le même rôle que la méthode join() à la différence près qu'elle supprime le premier élément du tableau affecté et décale ainsi les indices du tableau. Elle ne retourne pas de résultat.

```
1 var tab = new Array('html','ajax','php','javascript');
2 tab.shift(); // suppression du premier élément du tableau tab
```

```
3
4   document.write(tab.join(' | '));
5   // affiche "ajax | php | javascript"
6
7   document.write(tab[0]);
8   // affiche "ajax"
```

- *La méthode slice()*

Elle permet d'extraire une tranche du tableau sur lequel la méthode est appliquée. Elle attend deux (02) arguments, à savoir : *le début* et *la fin* de la tranche à extraire. Cette méthode renvoie un tableau composé de la tranche extraite et s'écrit `tab.slice(début,fin)` et la longueur de ce tableau est égale à *[fin -(moins) début]*. !

```
1   var tab = new Array('html','ajax','php','javascript');
2   var tabSlice = tab.slice(1,3); // Extraction d'une tranche du tableau tab
3
4   document.write(tabSlice .join(' | '));
5   // affiche "ajax | php"
6   // On commence l'extraction à l'indice 1
7   // La longueur de la tranche est égale à 3 - 1 soit 2
8   // On s'arrête donc à l'indice 2
```

- *La méthode sort()*

La méthode sort() organise les éléments du tableau par ordre croissant ou alphabétique.

Bien qu'elle ne retourne aucun résultat, le tableau sur lequel est appliquée cette méthode est directement modifié.

La méthode `sort()` peut être associée à la méthode `reverse()` pour trier un tableau dans un ordre décroissant !

```
1  var maTab = new Array('html','ajax','php','javascript');
2  maTab.sort(); // Trie le tableau maTab
3
4   document.write(maTab.join(' | '));
5   // affiche "ajax | html | javascript | php"
6
7   maTab.reverse(); // Trie le tableau tab dans l'ordre décroissant
8
9   document.write(maTab.join(' | '));
10  // affiche "php | javascript  | html  | ajax"
```

- *La méthode splice()*

La méthode `splice()` peut affecter un tableau de deux manières différentes en fonction des arguments qu'on lui fournit.

- En lui envoyant deux (2) arguments qui seront le début et la longueur, cette méthode écrase, efface la tranche du tableau affecté par cette méthode et s'écrit `tab.splice(debut,longueur)`.
- Avec des arguments en plus, la méthode `splice()` remplacera une tranche du tableau affecté par une autre envoyée en paramètre et s'écrira `tab.splice(debut,longueur,element1, element2, ...)`. Elle décalera probablement les indices des éléments du tableau si le tranche remplacée n'est pas en fin de tableau.

```
1   var tab = new Array('html','ajax','php','javascript');
2
```

```
tab.splice(2,1); // Ecrase / efface une tranche du tableau tab
// Ici, on efface une tranche de longueur 1 à partir de l'indice 2
// Cela revient en fait à suprrimer (dans notre exemple) l'élément php

document.write(tab.join(' | '));
// affiche "html | ajax | javascript"

// Remplacement d'une tranche par une autre
tab.splice(2,2,'mysql'); // Ici, on remplace la tranche 'php','javascript'
// par la tranche 'mysql'

document.write(tab.join(' | '));
// affiche "html | ajax | mysql"
```

-La méthode constructor

La méthode `constructor` permet de vérifier si la variable `tab` est bien un objet de type `Array`. Cette propriété contient le constructeur de l'objet `Array`.

```
var tab = new Array('html','ajax','php','javascript');

 if(tab.constructor.name == 'Array'){
  document.write('La variable tab est un tableau');
}
// Vérifie si tab est bien un objet de type Array
```

4. 4. L'objet Date

Il est parfois nécessaire de manipuler les dates et les heures au sein d'une application Web. Par exemple, créer un calendrier pour pouvoir sélectionner la date de départ d'une activité plus facilement, ou afficher l'horloge sur sa page Web, ou encore calculer la durée de séjour d'un client qui compte réserver une chambre d'hôtel sur un site d'hébergement.

Les dates en JavaScript peuvent être écrites de deux manières : soit littéralement c'est-à-dire sous forme de chaînes de caractères, soit sous forme d'un Timestamp Unix multiplié par 1000. Le Timestamp Unix correspond au nombre de secondes écoulées depuis le premier janvier 1970 à minuit UTC (Universal Time), date de la naissance du premier système UNIX.

- Récupérer la date actuelle

Pour récupérer la *date actuelle* sous *forme littérale*, on va tout simplement utiliser `Date()`.

Pour afficher cette même date sous forme de nombre (le nombre de millisecondes écoulées depuis le 1er janvier 1970), on peut utiliser la méthode `now()` de l'objet Date.

Aussi, pour créer un objet Date, on utilisera la syntaxe suivante : `var monObjetDate = new Date([paramètres]);` où `monObjetDate` est le nom de l'objet à créer.

- Quelques méthodes pratiques de l'objet Date

Source : https://www.chiny.me/objet-date-6-14.php

Plusieurs méthodes sont mises à notre dispositions. Mais, comme d'habitude nous allons voir les plus utiles:

- `getYear()` **:** permet de retourner l'année sur deux chiffres. Un préfixe numérique peut s'ajouter pour indiquer le siècle. Par exemple 15 désigne 1915 et 115 désigne 2015. Cependant, cette méthode est devenue obsolète et il est préférable d'utiliser getFullYear().
- `getFullYear()` **:** permet de retourner l'année sur 4 chiffres.
- `getDate()` **:** retourne la date du mois comprise entre 1 et 31.
- `getDay()` **:** retourne un indice numérique qui représente le jour de la semaine. 0 pour

dimanche, 1 pour lundi, ... , 6 pour samedi.
- *getMonth() :* retourne un indice numérique qui représente le mois. 0 pour janvier, 1 pour février, ... , 11 pour décembre.
- *getHours() :* retourne l'heure (sans zéro initial). Si la date crée fait référence à 9h du matin, alors cette méthode retournera 9 et non pas 09.
- *getMinutes() :* retourne les minutes (sans zéro initial).
- *getSeconds() :* retourne les secondes (sans zéro initial).
- *getMilliSeconds() :* retourne les millisecondes (rarement utilisé).
- *getTime():* retourne ce que l'on appelle le `Timestamp` UNIX (ou Timestamp POSIX) en millisecondes. C'est à dire le nombre de millisecondes écoulées depuis 01/01/1970 00:00:00. C'est la date où le premier système UNIX a été lancé.

Si par exemple, on veut afficher la date d'aujourd'hui sous la forme habituelle. Par exemple: *« Lundi 1 décembre 2015 »*. Voilà une version de code long mais simple a comprendre :

```
jour=new Array(
   "Dimanche",
   "Lundi",
   "Mardi",
   "Mercredi",
   "Jeudi",
   "Vendredi",
   "Samedi"
);
mois=new Array(
   "janvier",
   "février",
   "mars",
   "avril",
   "mai",
   "juin",
   "juillet",
   "août",
   "septembre",
   "octobre",
   "novembre",
   "décembre"
);
d=new Date();
aujourdhui =jour[d.getDay()]+" "+d.getDate()+
" "+
mois[d.getMonth()]+
" "+
d.getFullYear();
```

Si on tente d'afficher la variable (chaîne de caractères) aujourd'hui on aura: *Dimanche 8 juillet 2018*

On aurait pu réussir le même résultat avec la structure *switch case*. Mais avec les tableaux c'est plus simple a comprendre et le code est intéressant.

Complément

Vous pouvez visiter le site *Commentçamarche.net* (*https://www.commentcamarche.com/contents/571-javascript-l-objet-date*) pour plus d'informations sur l'objet Date.

5. 5. L'objet Math

L'objet *Math* est, comme vous l'imaginez, un objet qui a de nombreuses méthodes et propriétés permettant de manipuler des nombres et qui contient des fonctions mathématiques courantes.

Quelque soit la méthode ou la propriété utilisée, il est indispensable de le préfixer avec *Math* car il s'agit de méthodes et propriétés statiques, par exemple :

```
1 Math.cos(30);
```

- Méthodes et propriétés standards de l'objet Math

Méthode	Description	Exemples

abs()	Cette méthode renvoie la valeur absolue d'un nombre,
	il renvoie donc le nombre s'il est positif, son opposé (positif)
	s'il est négatif
		x = Math.abs(3.26); //donne x = 3.26
		x = Math.abs(-3.26); //donne x = 3.26

ceil()	Renvoie le plus petit entier supérieur ou égal à la valeur donnée en
	paramètre
		x = Math.ceil(6.01); //donne x = 7
		x = Math.ceil(3.99); //donne x = 4

floor()	La méthode floor() retourne le plus grand entier inférieur ou égal à la
	valeur donnée en paramètre.
		x = Math.floor(6.01); //donne x = 6
		x = Math.floor(3.99); //donne x = 3

round()	Arrondit à l'entier le plus proche la valeur donnée en paramètre.
	Si la partie décimale de la valeur entrée en paramètre vaut 0.5,
	la méthode Math() arrondi à l'entier supérieur.
		x = Math.round(6.01); //donne x = 6
		x = Math.round(3.80); //donne x = 4
		x = Math.round(3.50); //donne x = 4

trunc(Nombre)	renvoie la partie entière d'un nombre.	var x = Math.trunc(6.25); //donne x = 6
		var x = Math.trunc(-3.65); //donne x = -3
		var x = Math.trunc(0,2015); //donne x = 0

max(Nombre1, Nombre2)	max() renvoie le plus grand des deux nombres donnés en paramètre	var x = Math.max(6,7.25); //donne x = 7.25
		var x = Math.max(-8.21,-3.65); //donne x = -3.65
		var x = Math.max(5,5); //donne x = 5

min(Nombre1, Nombre2)	Retourne le plus petit des deux nombres donnés en paramètre	x = Math.min(6,7.25); //donne x = 6
		x = Math.min(-8.21,-3.65); //donne x = -8.21
		x = Math.min(5,5); //donne x = 5

pow(Valeur1, Valeur2)	Retourne le nombre Valeur1 à la puissance Valeur2	x = Math.pow(3,3); //donne x = 27
		x = Math.pow(9,0.5); //(racine carrée)
		//donne x = 3

random()	La méthode random() renevoie un nombre pseudo-aléatoire compris entre 0 et 1. La valeur est générée à partir des données de l'horloge de l'ordinateur.	x = Math.random(); //donne x = 0.6489534931546957
sqrt(Valeur)	Renvoie la racine carrée du nombre passé en paramètre	x = Math.sqrt(9); //donne x = 3

Source : https://www.commentcamarche.com/contents/578-javascript-l-objet-math

- Méthodes Logarithmes et exponentielles

Méthode	description
Math.E	Propriété qui retourne le nombre d'Euler (environ 2.718).
Math.exp(valeur)	Cette méthode renvoie l'exponentielle de la valeur entrée en paramètre.
Math.LN2	La propriété LN2 fournit le logarithme népérien de 2.
Math.LN10	Propriété donne le logarithme népérien de 10.
Math.log(valeur)	La méthode log() renvoie le logarithme de la valeur entrée en paramètre.
Math.LOG2E	Propriété qui renvoie la valeur du logarithme du nombre d'Euler en base 2.
Math.SQRT1_2	Propriété qui retourne la valeur de "1 divisé par racine de 2" (0.707).
Math.SQRT2	La propriété SQRT2 (Square Root 2) donne la racine de 2 (1.414).

Source : https://www.commentcamarche.com/contents/578-javascript-l-objet-math

- Méthodes Trigonométriques

Méthode	description
`Math.PI`	Retourne la valeur du nombre PI, soit environ 3.1415927
`Math.sin(valeur)`	Retourne le sinus de la valeur entrée en paramètre (doit être donnée en radians). La valeur retourné est comprise dans l'intervalle [-1;1].
`Math.asin(valeur)`	Retourne l'arcsinus de la valeur entrée en paramètre. La valeur doit être comprise dans l'intervalle [-1;1]. Dans le cas contraire, la méthode asin() renvoie la valeur NaN (Not a Number).
`Math.cos(valeur)`	Retourne le cosinus de la valeur entrée en paramètre (doit être donnée en radians). La valeur retourné est comprise dans l'intervalle [-1;1].
`Math.acos(valeur)`	Retourne l'arccosinus de la valeur entrée en paramètre. La valeur doit être comprise dans l'intervalle [-1;1]. Dans le cas contraire, la méthode acos() renvoie la valeur NaN (Not a Number).
`Math.tan(valeur)`	Retourne la tangente de la valeur entrée en paramètre (doit être donnée en radians)
`Math.atan(valeur)`	Retourne l'arctangente de la valeur entrée en paramètre. La valeur doit être comprise dans l'intervalle [-1;1]. Dans le cas contraire, la méthode atan() renvoie la valeur NaN (Not a Number).

Source : https://www.commentcamarche.com/contents/578-javascript-l-objet-math

6. Exercice

Exercice

1) Pour ajouter un item dans un objet littéral, il faut utiliser la méthode push()

- ◯ Vrai
- ◯ Faux

Exercice

2) Quel objet JavaScript peut-on utiliser pour transformer une phrase saisie par l'utilisateur en majuscule ?

- ☐ String
- ☐ VarChar
- ☐ Var
- ☐ Array
- ☐ Math

Exercice

3) Quelle méthode permet de découper une chaîne de caractères en un tableau ?

- ☐ split()
- ☐ push()

Exercice

4) Ecrire autrement le code ci-dessous pour avoir le même résultat (Qu'est ce qui la même chose que :)

```
1 myArray.length;
```

Écrire le code sans espaces.

Exercice

5) De quel Objet `getFullYear()` est-elle une méthode ? et Quelle est son rôle ?

- ☐ L'objet String
- ☐ L'Objet Math
- ☐ L'objet Date
- ☐ Retourner l'année sur 4 chiffres
- ☐ Retourner l'année sur 2 chiffres
- ☐ Retourner le calendrier de l'année en cours