COURS DE JAVASCRIPT 2021-2022
LICENCE 2 MIAGE/GENIE INFORMATIQUE

Table des matières

Chapitre 1 : Introduction à JS 3

1- Présentation de JS 3

2- Comment utiliser le JS ? 3

2.1 Le JS dans le corps du code HTML 3

2.2 Le JS dans une feuille totalement séparée du code HTML 4

3- Indentation et commentaire 5

Chapitre2 : Variables, type de données et constantes 6

1. Notion de variables 6
2. Les types de variables 6
3. Les constantes : 6

Chapitre 3 : Les Opérateurs 7

1- Les opérateurs Arithmétiques 7

2- La concaténation 7

3- Les opérateurs de comparaison 8

4- Les opérateurs logiques 8

5- L'opérateur ternaire 8

Chapitre 4 : Les boites de dialogue 9

1. La fonction alert 9
2. La fonction prompt 9
3. La fonction confirm 9

Chapitre 5 : Les structures de contrôle et de boucles 10

1. L'instruction if 10
2. L'instruction if…else 10
3. L'instruction switch 10
4. La boucle while 10
5. La boucle do…while 11
6. La boucle for 11

Chapitre 6 : Les fonctions 13

1. Création 13
2. Variables locales et globales 13
3. Valeur de retour 13
4. Fonction anonymes 14
5. Valeur par défaut des arguments 14

6. Paramètres du reste (Rest paramaters) 14
7. Fonction récursive 15
1. Imbrication de Fonction 15
Chapitre 7 : Les Tableaux 16
1. Création 16
2. Accès aux éléments 17
3. Parcours d'un tableau 17
4. Opération sur les tableaux 18
Chapitre 8 : Les Objets personnalisés en JS 25
1. Création d'objets 25
2. Lister les propriétés d'un objet 28
3. Modifier les propriétés d'un objet 29
4. Les setters et les getters 30
5. Supprimer les propriétés d'un objet 31
6. Comparer les objets 31

Chapitre 1 : Introduction à JS

1- Présentation de JS

Le JavaScript, créé en 1995 par Brendan Eich (pour la Netscape Communication Corporation), est un langage de programmation de scripts orienté objet. Avant il était pour rendre dynamique des pages web cote client de nos jours JavaScript est aussi utilisé cote serveur pour rendre les pages web dynamiques comme le fait le PHP. Le langage dispose de certaines bibliothèques comme JQuery et React Js contenant plusieurs fonctionnalités, il dispose également des plateformes permettant de développer des applications sans utiliser de pages web (Node JS) et des Framework prêt à l'emploie comme Angular, Vue JS, etc.

Pour coder en JS on a juste besoin d'avoir un éditeur de texte et un navigateur.

2- Comment utiliser le JS ?

Deux techniques sont disponibles pour utiliser le JavaScript :

2.1 Le JS dans le corps du code HTML

Vous pouvez écrire du JavaScript directement dans le corps du document HTML en plaçant votre code entre la balise **<script> et </script>.**

```
<!DOCTYPE html>
<html lang="en">
  <head>

    <meta charset="utf-8">
    <title>cours de JS</title>

  </head>
  <body>
  <h1> Cours de JavaScript </h1>
  <script>
    ..........
  </script>
  </body>
</html>
```

Cette approche est l'ancienne façon de d'utiliser du JS

2.2 Le JS dans une feuille totalement séparée du code HTML

La façon idéale de définir le JS consiste à les enregistrer dans un document indépendant de vos pages HTML. Et de faire appel à ce fichier dans le code html via l'attribut src de la baise <script> qu'il faudra placer avant la fermeture de la balise <body>

```
/* fichier code.js */
alert('Bonjour')
document.write('Bienvenue en JS')
```

NB :

- ✓ alert () est une instruction simple, appelée fonction, qui permet d'afficher une boîte de dialogue contenant un message. Ce message est placé entre apostrophes, elles-mêmes placées entre les parenthèses de la fonction alert ().
- ✓ document.write permet d'afficher un message sur le document en cours
- ✓ L'extension d'un fichier contenant du JavaScript est **.js**
- ✓ Contrairement à html et css js est sensible à la casse

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>
 <h1>JavaScript</h1>
 <script src = "code.js"></script>
</body>
</html>
```

NB :

- Vous pouvez tester le code JavaScript sans créer systématiquement des pages web sur le site http://jsfiddle.net
- Vous pouvez également exécuter votre code en ligne de commande via jsshell Avant de voir le **DOM (Document Object Model)** c'est cette dernière technique que nous utiliserons

```
C:\...........>js nomfichier.js
```

3- Indentation et commentaire

Pour s'y retrouver dans l'écriture du code, on peut l'indenter, c'est-à-dire hiérarchiser les lignes de code avec des tabulations. Par ailleurs, on peut intégrer des commentaires, qui ne sont pas interprétés comme du code, afin d'expliquer son code ou de mieux s'y retrouver :

```
<script>
instruction_1; // Ceci est ma première instruction
instruction_2;
  /* La troisième instruction ci-dessous,
    avec un commentaire sur deux lignes */
instruction_3;
</script>
```

Chapitre2 : Variables, type de données et constantes

1. Notion de variables

Une variable consiste en un espace de stockage, qui permet de garder en mémoire tout type de données. La variable est ensuite utilisée dans les scripts. Une variable contient seulement des caractères alphanumériques, le $ (dollar) et le _ (Under score) ; elle ne peut pas commencer par un chiffre ni prendre le nom d'une fonction existante de JavaScript. On crée la variable et on lui affecte une valeur :

```
<script>
var myVariable;
myVariable = 2;
</script>
```

Où

```
<script>
var myVariable = 2;
</script>
```

2. Les types de variables

Une variable peut être de type numérique, booléen mais aussi une chaîne de caractères en JavaScript le typage est dynamique c'est-à-dire que l'on n'a pas besoin de préciser le type au moment de la création de la variable. Le type est une fonction du contenu. Il existe néanmoins une fonction nommée **typeof** qui reçoit en paramètre le nom de la variable et renvoie le type de la variable :

```
<script>
    var a = 12;
    var b = 3.5;
    var nom = 'diallo';
    var rep = true;
    var message = 'au bas de l\'echelle';
    document.write('type de a ' + typeof(a) + "<br>");
    document.write('type de b ' + typeof b + "<br>");
    document.write('type de nom ' + typeof(nom) + "<br>");
    document.write('type de rep ' + typeof rep + "<br>");
    document.write('type de rep ' + typeof message + "<br>");
</script>
```

3. Les constantes :

En Js pour créer une constante on remplace juste le mot clé **var** par **const** dans ce cas le contenu reste inchangé pour les données de type primitifs mais pas pour les objets.

Chapitre 3 : Les Opérateurs

1- Les opérateurs Arithmétiques

On peut utiliser 5 opérateurs arithmétiques : l'addition (+), la soustraction (-), la multiplication (*), la division (/) et le modulo (%). Le modulo est le reste d'une division, l'incrémentation (++), la décrémentation (--)

```
<script>
        var number1 = 3, number2 = 2, result;
        result = number1 * number2;
        alert(result); // Affiche : « 6 »
</script>
```

```
<script>
        var number = 3;
        number += 5;
        alert(number); // Affiche : « 8 »
</script>
```

```
<script>
        var number = 3;
        number++;
        alert(number); // Affiche : « 4 »
</script>
```

Remarque :

Le JS fait belle et bien la différence entre post et pre incrémentation et entre post et pre décrémentation.

2- La concaténation

Une concaténation consiste à ajouter une chaîne de caractères à la fin d'une autre

```
<script>
        var hi = 'Diallo ', name = 'Amadou', result;
        result = hi + name;
        alert(result);
        // Affiche : « Diallo Amadou »
</script>
```

```
<script>
        var text = 'Diallo ';  text += 'Amadou';
        alert(text);
        // Affiche « Diallo Amadou ».
    </script>
```

3- Les opérateurs de comparaison

En JS il existe 8 opérateurs de comparaisons :

- ✓ == égal à
- ✓ != différent de
- ✓ === contenu et type de variable égal à
- ✓ !== contenu ou type de variable différent de
- ✓ > supérieur à
- ✓ >= supérieur ou égal à
- ✓ < inférieur à
- ✓ <= inférieur ou égal à

4- Les opérateurs logiques

Comme la plupart des langages de programmation ces opérateurs sont au nombre de 3 :

- o && qui signifie ET avec par exemple : valeur1 && valeur2 Cet opérateur vérifie la condition lorsque toutes les valeurs qui lui sont passées valent true.

- o || qui signifie OU avec par exemple : valeur1 || valeur2 Cet opérateur est plus souple car il renvoie true si une des valeurs qui lui est soumise contient true, qu'importent les autres valeurs.

- o ! qui signifie NON avec par exemple : ! valeur Cet opérateur se différencie des deux autres car il ne prend qu'une seule valeur à la fois. S'il se nomme « NON » c'est parce que sa fonction est d'inverser la valeur qui lui est passée, ainsi true deviendra false et inversement.

5- L'opérateur ternaire

```
var myvar = (condition ? expression_si_vrai : expression_si_faux);
```

Chapitre 4 : Les boites de dialogue

1. La fonction alert

Permet d'afficher un message dans une boite de dialogue avec un bouton ok.

```
alert('message');
```

2. La fonction prompt

Permet d'afficher un message dans une boite de dialogue avec un bouton ok, un bouton annuler et une zone de saisie. Un clic sur le bouton ok renvoie le contenu de la zone de saisie sous forme de chaine de caractère.

```
var age = prompt('donner votre age : ');
```

Quel que soit le contenu de la zone de saisie l'information sera renvoyée sous forme de chaine de caractère. Si vous souhaiter convertir cette information en valeur numérique utilisez les fonctions de conversion :

- ✓ parseInt : pour la conversion en entier.
- ✓ parseFloat : pour la conversion en réel.

3. La fonction confirm

Permet d'afficher un message dans une boite de dialogue avec un bouton ok, un bouton annuler. Un clic sur le bouton ok renvoie true et un clic sur le bouton annuler renvoie false.

```
confirm("etes vous marier ?");
```

Chapitre 5 : Les structures de contrôle et de boucles

1. L'instruction if

Syntaxe :

```
if(condition)
    {
      //instructions
    }
```

2. L'instruction if...else

Syntaxe :

```
if(condition)
    {
      //instructions
    }
    else
    {
      //instructions
    }
```

3. L'instruction switch

Syntaxe :

```
switch (expression)
   {
       case val1:
           //instructions
           break;
       case val2:
           //instructions
           break;
       ..............
       default:
           break ;
   }
```

4. La boucle while

Syntaxe :

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

5. La boucle do…while

Syntaxe :

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

6. La boucle for

Syntaxe :

```
for (initialisation; condition; incrémentation)
    {

        // instructions

    }
```

EXERCICES D'APPLICATION

1- Ecrire un script js permettant de fournir les trois coefficients d'une équation du second degré ensuite calcul et affiche les racines réelles si elles existent.
2- Ecrire un script js qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.
3- Ecrire un script js qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.
4- Ecrire un script js qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le script js affichera les nombres de 18 à 27 en utilisant l'instruction while.
5- Réécrire le script js précédent, en utilisant cette fois l'instruction for.
6- Ecrire un script js qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre.
7- Ecrire un script js qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le script js doit calculer : 1+ 2 + 3 + 4 + 5 = 15. On souhaite afficher uniquement le résultat, pas la décomposition du calcul.
8- Ecrire un script js qui demande un nombre de départ, et qui calcule sa factorielle. NB : la factorielle de 8, notée 8 ! vaut 1 x 2 x 3 x 4 x 5 x 6 x 7 x 8.

9- Ecrire un script js qui demande successivement 20 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 20 nombres. Modifiez ensuite le script js pour que le script js affiche de surcroît en quelle position avait été saisie ce nombre.
10- Réécrire le script js précédent, mais cette fois-ci on ne connaît pas d'avance combien l'utilisateur souhaite saisir de nombres. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.
11- Lire la suite des prix (en F entiers et terminée par zéro) des achats d'un client. Calculer la somme qu'il doit, lire la somme qu'il paye, et simuler la remise de la monnaie en affichant les textes "10 F", "5 F" et "1 F" autant de fois qu'il y a de coupures de chaque sorte à rendre.
12- Ecrire un script js qui permet à l'utilisateur de saisir une suite caractère se terminant par '*', et qui affiche à la fin le nombre d'apparition de la lettre 'A' ou 'a'.
13- Ecrire un script js permettant de convertir un entier N écrit sous forme binaire en sa valeur décimale.
14- Ecrire un script js pour résoudre chacun des problèmes suivants :
    - Calcul de la somme des N premiers nombres entiers.
    - Recherche du minimum et du maximum dans un ensemble de N nombres.
    - Calcul du quotient et reste de la division de deux entiers A et B sans utiliser l'opération de division.
    - Le calcul du produit de deux entiers en utilisant uniquement l'opération d'addition '+'.
    - Détermination si A est divisible par B. Avec A et B des entiers positifs.
    - Déterminer tous les diviseurs d'un entier X donné.
    - Déterminer si un nombre entier X est premier ou non.
    - Calcule la somme des chiffres qui composent un entier naturel N.

Chapitre 6 : Les fonctions

1. Création

```
<script>
        function nomFonction(arguments)
        {
            // Le terme "function" est obligatoire pour déclarer une fonction
            // instructions
        }
 </script>
```

2. Variables locales et globales

Toute variable déclarée dans une fonction n'est utilisable que dans cette même fonction. Ces variables spécifiques à une seule fonction se nomme variables locales. Déclarées en dehors des fonctions, on parle de variables globales.

```
<script>
        var nom = 'BAH'; // variable globale
        function salutation(prenom)
        {
            var age = 20; // variable locale ainsi que l'argument prenom
            document.write('bonjour Mr '+nom+' '+prenom+'votre age est : '+ age);
        }
        salutation('Tidiane');
    </script>
```

NB : pour déclarer une variable locale à la place du mot clé var il est conseillé d'utiliser le mot clé let

3. Valeur de retour

Une fonction peut retourner une seule valeur, stockée dans une variable dans ce cas on utilise le mot clé return.

```
<script>
        function somme(a, b)
        {
           return (a + b);
        }
        var som = somme(6, 7);
        document.write('la somme de 6 et de 7 est : ' + som);
        // affiche << la somme de 6 et de 7 est : 13 >>
</script>
```

4. Fonction anonymes

Une fonction anonyme est une fonction qui n'a pas de nom, elle peut être utilisée par le biais d'une variable.

```
<script>
        var politesse = function (prenom)
        {
           document.write('Merci' +' ' +prenom);
        };

        /*
           ici pour l'appel c'est le nom de la variable qui est considéré
          comme nom de fonction lors de l'appel
        */
        politesse("BAH");
        // affiche << Merci BAH >>
</script>
```

NB : A l'instar des fonctions anonymes il est aussi possible en JS de créer des fonctions fléchées (c'est-à-dire des fonctions créés sans le mot clé **function** et qui non pas de noms)

5. Valeur par défaut des arguments

Cette syntaxe permet d'initialiser des paramètres lors de l'appel de la fonction si aucune valeur n'est passée ou si c'est la valeur undefined qui est passée.

```
var f = (x , y = 10) => x + y
print(f(5))
```

6. Paramètres du reste (Rest paramaters)

Cette syntaxe permet de représenter un nombre indéfini d'arguments sous forme d'un tableau.

```
function f(a, b, ...lesArguments)
 {
    // ...
 }
```

```
var somme =(som = 0, ...args) =>
{ for (let index = 0; index < args.length; index++)
    { som += args[index];}
    return som
}
print('somme(7,5,8,9) = ' + somme(7,5,8,9))
```

7. Fonction récursive

Une fonction qui s'appelle elle-même est appelée une fonction récursive. Elle s'appuie sur une condition afin d'éviter une récursion infinie.

```
var factoriel = (n, index = 1, fact = 1) =>
{
    if (n === 0 || n === 1) return 1
    else return n * factoriel(n - 1)
}
print('factoriel(5) = '+factoriel(5));
```

1. Imbrication de Fonction

Il est possible d'imbriquer une fonction au sein d'une fonction. La fonction imbriquée (interne) est privée par rapport à la fonction (externe) qui la contient. Cela forme ce qu'on appelle une fermeture (*closure* en anglais).

La fonction interne est une fermeture : la fonction interne peut utiliser des arguments et des variables de la fonction externe alors que la fonction externe ne peut pas utiliser de variables et d'arguments de la fonction interne.

```
function sommecarre(a, b)
{
    function carre(x)
    {
        return (x*x);
    }
    return carre(a) + carre(b)
}
print('2² + 3² = '+ sommecarre(2,3))
```

Chapitre 7 : Les Tableaux

1. Création

Après Number, String et Boolean, Array est le 4 ème objet natif de JavaScript. Un tableau, ou plutôt un array en anglais, est une variable qui contient plusieurs valeurs, appelées items. Chaque item est accessible au moyen d'un indice (index en anglais) et dont la numérotation commence à partir de 0. Pour déclarer un tableau en JS on utilise la syntaxe suivante :

```
<script>
   var myArray = ['Diallo', 'Camara', 'Keita', 'Bilivogui'];
    // Le contenu se définit entre crochets, avec une virgule entre chaque valeur.
    // La chaîne 'Diallo' correspond à l'indice 0, 'Camara' à l'indice 1...
</script>
```

NB : il existe plusieurs autres possibilités pour la création d'un tableau :

➲ Créer un tableau en utilisant le constructeur :

```
var myArray = new Array(); // tableau vide de 0 élément
var myArray = new Array(5); // tableau vide de 5 éléments
var myArray = new Array(4,5); // tableau de 2 éléments contenant 4 et 5

            ou
var myArray = [];
```

➲ Créer un tableau avec la méthode static **from**

```
    //création et initialisation d'un tableau de 5 éléments
      var myArray = Array.from([5, 3, 7, 9, 23]);
      // création d'un tableau de 7 caractères => ['b'.'o','n','j','o'.'u'.'r']
      myArray = Array.from('bonjour');

      const double = function(item)
                    {
                       return item * 2
                    }
     /*création d'un tableau de 4 éléments en appelant la
      Fonction double pour chaque élément du premier tableau */
      myArray = Array.from([2, 3, 4, 5],double)
```

➲ Créer un tableau avec la méthode static **of**

```
var myArray = Array.of(34); // tableau contenant l'élément 34
var myArray = Array.of(5,6); // tableau contenant les éléments 5 et 6
```

2. Accès aux éléments

On peut modifier ou afficher une valeur du tableau

```
<script>
        var myArray = ['Diallo', 'Camara', 'Keita', 'Bilivogui'];
        myArray[1] = 'Sow';
        alert(myArray[1]); // Affiche : « Sow »
</script>
```

3. Parcours d'un tableau

On peut parcourir un tableau avec la boucle for

```
<script>
        var myArray = ['Diallo', 'Camara', 'Keita', 'Bilivogui'];
         // La  proprieté length indique le nombre d'éléments du tableau
        for (let index = 0; index < myArray.length; index++)
        {
         document.write(myArray[index] + '<br>');
        }
</script>
```

On peut aussi utiliser la boucle **for in** qui renvoie les clés des valeurs :

```
       var myArray = Array.from([5, 3, 7, 9, 23]);
       for (const key in myArray)
       {
        document.write(myArray[key]+ '<br>')
       }
```

On peut aussi utiliser la boucle **for of** qui renvoie les valeurs :

```
       var myArray = Array.from([5, 3, 7, 9, 23]);
       for (const item of myArray)
       {
        document.write(item + '<br>')
       }
```

4. Opération sur les tableaux

- La méthode static **isArray ()** : renvoie true si la variable passer un argument est un tableau :

```
var myArray = Array.from([5, 3, 7, 9, 23]);
if (Array.isArray(myArray))
{
  document.write('c\'est un tableau')
}
else
{
 document.write('c\'est pas un tableau')
}
```

- La méthode **push ()** : ajoute des éléments à la fin du tableau :

```
<script>
        var myArray = ['BAH', 'SOW'];
        myArray.push('CAMARA'); // Ajoute « CAMARA » à la fin du tableau
        // Ajout de « KEITA » et « MAOU » à la fin du tableau
        myArray.push('KEITA', 'MAOU');
        for (let index = 0; index < myArray.length; index++)
        {
            document.write(myArray[index] + '<br>');
        }
</script>
```

- La méthode **unshift ()** : ajoute des éléments au debut du tableau :

```
<script>
        var myArray = ['BAH', 'SOW'];
        myArray.unshift('CAMARA'); // Ajoute « CAMARA » au debut du tableau
        // Ajout de « KEITA » et « MAOU » au debut du tableau
        myArray.unshift('KEITA', 'MAOU');
        for (let index = 0; index < myArray.length; index++)
        { document.write(myArray[index] + '<br>');}
</script>
```

- La méthode **pop ()** : retire le dernier élément du tableau :
- La méthode **shift ()** : retire le premier élément du tableau :

```
<script>
        var myArray = ['BAH', 'SOW', 'CAMARA', 'BILIVOGUI', 'KABA'];
        myArray.shift(); // retire le premier élément du tableau
        myArray.pop(); // retire le dernier élément du tableau
        for (let index = 0; index < myArray.length; index++)
        {
            document.write(myArray[index] + '<br>');
        }
</script>
```

➲ La méthode **join ()** : convertit un tableau en chaine de caractère en recevant en argument un séparateur. Sans argument la virgule sera utilisée comme séparateur

```
<script>
        var myArray = ['BAH', 'SOW', 'CAMARA', 'BILIVOGUI', 'KABA'];
        var machaine = myArray.join('-');
        document.write(machaine);
    // affiche << BAH-SOW-CAMARA-BILIVOGUI-KABA >>
</script>
```

➲ La méthode **reverse ()** : permet de permuter le contenu du tableau
➲ La méthode **forEach ()** : exécute une fonction fournie une fois pour chaque élément du tableau.

```
<script>
       var myArray = ['BAH', 'SOW', 'CAMARA', 'BILIVOGUI', 'KABA'];
        myArray.reverse();
        myArray.forEach(element =>
        {
            document.write(element+'<br>');
        });
</script>
```

➲ La méthode sort () : permet de trier le tableau. Sans paramètres l'ordre de tri par défaut est croissant, basé sur la conversion des éléments en chaînes, puis sur la comparaison lexicographique.
Donc il faudra spécifier une fonction en paramètre qui définit l'ordre du trie. Cette fonction doit recevoir deux paramètres obligatoire : l'élément courant et l'élément suivant Pour effectuer la permutation.

```
<script>
       var myArray = ['BAH', 'SOW', 'CAMARA', 'BILIVOGUI', 'KABA'];
        myArray.sort();
        myArray.forEach(element =>
        {
            document.write(element+'<br>');
        });
</script>
```

➲ La méthode **at ()** : Renvoie l'élément de tableau à l'index donné. Accepte les entiers négatifs, qui comptent à partir du dernier élément.

```
for (let index = 0; index < myArray.length; index++)
    {
        const element = myArray.at(index);
        document.write(element + '<br>');
    }
document.write('le dernier element est : '+myArray.at(-1) + '<br>');
```

➲ La méthode **concat ()** : Renvoie un nouveau tableau qui est le tableau appelant joint à d'autres tableaux et/ou valeurs.

```
       var array1 = [5, 6, 7, 8, 23, 65, 88];
       var array2 = [3, 4, 9];
       var array = array1.concat(array2, 100, 200);
       for (let index = 0; index < array.length; index++)
       {
        const element = array[index];
        document.write(element + '<br>');
       }
```

➲ La méthode **every ()** : Renvoie true si tous les éléments du tableau satisfait la fonction de test. La fonction de test peut recevoir comme second argument l'indice de l'élément en cours.

```
    const est_impair = (elem => elem % 2 != 0);
    if (myArray.every(est_impair))
    {
       document.write('tous les elements sont impairs');
    }
    else
    {
       document.write('tous les elements ne sont pas impairs');
    }
```

➲ La méthode **fill ()** : change tous les éléments d'un tableau en une valeur statique, d'un index de début (default 0) à un index de fin (default array.length).

```
var myArray = [6, 7, 24, 65, 80];
myArray.fill(500);
// remplace tous les éléments du tableau par 500
myArray.fill(100,2);
/* remplace tous les éléments du tableau par 100 à partir
   de l'indice 2
*/
myArray.fill(200, 2, 4);
/* remplace tous les éléments du tableau par 200 à partir
   de l'indice 2 jusqu'à l'indice 4-1 = 3
*/
myArray.forEach(element => document.write(element+'<br>'));
```

➲ La méthode **filter ()** : crée une copie d'une partie d'un tableau donné, filtrée uniquement pour les éléments du tableau donné qui réussissent le test implémenté par la fonction de test. La fonction de test peut recevoir comme second argument l'indice de l'élément en cours.

```
var myArray = [6, 7, 24, 65, 80];
const test = function(elem){return elem % 2 === 0;};
/* ou tous simplement en lamda expression
 const test = (elem => elem % 2 === 0); */
myArray = myArray.filter(test);
myArray.forEach(element => document.write(element+'<br>'));
```

➲ La méthode **find ()** : renvoie le premier élément du tableau qui satisfait la fonction de test. Si aucune valeur ne satisfait la fonction de test, undefined est renvoyé. Le parcours se fera de gauche à droite. La fonction de test peut recevoir comme second argument l'indice de l'élément en cours.

NB :

- Si vous avez besoin de l' index de l'élément trouvé dans le tableau, utilisez **findIndex**(). Si l'élément n'existe pas -1 sera renvoyer
- Si vous faire le parcours de la droite vers la gauche utilisez les méthodes **findLast** () et **findLastIndex** ().

```
var myArray = [6, 7, 24, 65, 80];
const test = ((item,index) => item % 2 === 0 && index > 0);
document.write(myArray.find(test));
```

➲ La méthode **includes ()** : détermine si un tableau inclut une certaine valeur parmi ses entrées, en retournant true ou false selon le cas. Elle peut recevoir un deuxième paramètre qui indique l'indice de debut de recherche.

```
var myArray = [6, 7, 24, 65, 80, 60, 15, 99];
      if (myArray.includes(7, 2))
      {
         document.write('l\'élément 7 existe à partir de la position 2');
      }
      else
      {
       document.write('l\'élément 7 n\'existe pas à partir de la position 2');
      }
```

➲ La méthode **indexOf ()** : renvoie l'index de la première occurrence d'un dans le tableau, ou -1 s'il n'est pas présent. Elle peut recevoir un deuxième paramètre qui indique l'indice de debut de recherche.
NB :

- Pour rechercher la dernière occurrence de l'élément utiliser la méthode **lastIndexOf** ()

```
  var myArray = [6, 7, 24, 65, 7, 60, 15, 99];
  if (myArray.indexOf(7)  != -1)
    {
      document.write('l\'element 7 se trouve à l\'indice '+myArray.indexOf(7));
    }
  else
    {
       document.write('l\'element 7 n\'existe');
    }
```

➲ La méthode **map ()** : crée un nouveau tableau rempli avec les résultats de l'appel d'une fonction sur chaque élément du tableau. La fonction peut recevoir comme second argument l'indice de l'élément en cours.

```
      var myArray = [6, 7, 24, 65, 7, 60, 15, 7];
      var carre = myArray.map((item) => item * item );
      carre.forEach(element =>
      {
        document.write(element+'<br>');
      });
```

➲ La méthode **reduce ()** : exécute une fonction de « réducteur » fournie par l'utilisateur sur chaque élément du tableau de gauche à droite, dans l'ordre, en transmettant la valeur de retour du calcul sur l'élément précédent. La fonction réductrice reçoit deux paramètres obligatoire l'élément précèdent et l'élément courant. Un troisième paramètre facultatif est l'index de l'élément courant. La fonction reduce reçoit un second paramètre qui est la valeur initiale.

```
var myArray = [2, 5, 2, 7];
const somme = (prev, next) => (prev + next)
document.write('la somme des elements est : '+myArray.reduce(somme,0));
```

NB :

- La première fois que le rappel est exécuté, il n'y a pas de "valeur de retour du calcul précédent". Si elle est fournie, une valeur initiale peut être utilisée à sa place. Sinon, l'élément de tableau à l'index 0 est utilisé comme valeur initiale et l'itération commence à partir de l'élément suivant (index 1 au lieu de l'index 0).

- La fonction **reduceRight** () effectue la même opération mais de droite a gauche.

➲ La méthode **slice ()** renvoie d'une partie d'un tableau dans un nouvel objet tableau sélectionné de **start** à **end** (**end** non inclus) où **start** et **end** représentent l'index des éléments de ce tableau. Le tableau d'origine ne sera pas modifié.

```
var myArray = [2, 5, 2, 7, 9, 23, 56, 90];
  // copie de tous les elements de myArray dans array
  var array = myArray.slice();
// copie à partir de l'indice 3 des elements de myArray dans array
   var array = myArray.slice(3);
/*
  copie à partir de l'indice 3 à l'indice 7 exclus
  des elements de myArray dans array
*/
var array = myArray.slice(3,7);
```

➲ La méthode some () : Renvoie true si au moins un élément du tableau satisfait la fonction de test fournie false sinon.

```
var myArray = [5, 3, 7, 9, 23, 57, 91];
const lamda = item => item % 2 === 0;
if(myArray.some(lamda))
{
 document.write('il existe au moins un élément pairs dans le tableau')
}
else
{
 document.write('tous les éléments sont impairs')
}
```

➲ La méthode **splice ()** : modifie le contenu d'un tableau en remplaçant ou en insérant des éléments :

- Le premier paramètre est l'index de l'élément à remplacer ou l'index d'insertion.
- Le second paramètre est un booléen avec false on fera une insertion et avec true on fera un remplacement.
- Le troisième paramètre est la nouvelle valeur.

```
var myArray = [5, 3, 7, 9, 23];
      myArray.splice(2,false,100);//insère 100 à la deuxième position
      myArray.splice(3,true,200);//remplace l'élément de position 3 par 200
      myArray.forEach(element => document.write(element+'<br>'));
```

➲ La méthode **toString ()** : renvoie une chaîne représentant le tableau spécifié et ses éléments.

```
var myArray = [5, 3, 7, 9, 23];
document.write(myArray.toString())
```

Chapitre 8 : Les Objets personnalisés en JS

Un objet est un ensemble cohérent de propriétés et de méthodes. JavaScript dispose d'objets natifs (objets prédéfinis) qui possèdent des propriétés et des méthodes qu'on va pouvoir directement utiliser et nous permet également de définir nos propres objets.

1. Création d'objets

Nous pouvons créer des objets de 4 manières différentes en JavaScript :

- Créer un objet littéral ;
- Utiliser une fonction constructeur personnalisée ;
- Utiliser le constructeur Object () ;
- Utiliser la méthode create ().

➲ Création d'objet littéral

Pour créer un objet littéral, on utilise une paire d'accolades {…} qui indique au JavaScript que nous créons un objet.

Un objet est composé de différents couples de **« nom : valeur »** qu'on appelle membres. Chaque nom d'un membre doit être séparé de sa valeur par un caractère deux points et les différents membres d'un objet doivent être séparés les uns des autres par une virgule.

La partie « nom » de chaque membre suit les mêmes règles que le nommage d'une variable. La partie valeur d'un membre peut être n'importe quel type de valeur : une chaine de caractère, un nombre, une fonction, un tableau ou même un autre objet littéral.

Les membres d'un objet qui ne servent qu'à stocker des données sont appelés des propriétés tandis que ceux qui manipulent des données (c'est-à-dire ceux qui contiennent des fonctions en valeur) sont appelés des méthodes.

```
var Produit = {
                refprod : 'P001',
                desgprod : 'MANGUE',
                qte : 50,
                affiche : function()
                          {
                            document.write('Reference : '+this.refprod+'<br>'+
                                           'Designation :  '+this.desgprod+'<br>'+
                                           'Quantite : '+this.qte+'<br>');
                          }
              };
```

➲ Création d'objet avec une fonction de construction

Une fonction constructeur d'objets est une fonction qui va nous permettre de créer des objets semblables. En JavaScript, n'importe quelle fonction va pouvoir faire office de constructeur d'objets.

Pour construire des objets à partir d'une fonction « constructeur », nous allons devoir suivre deux étapes : définir la fonction « constructeur » et ensuite appeler ce constructeur syntaxe en utilisant le mot clé **new**.

```
<script>
      function Produit (refprod, desgprod, qte)
                    {
                      this.refprod = refprod;
                      this.desgprod = desgprod;
                      this.qte = qte;
                      this.affiche = function()
                       {
                          document.write('Reference : '+this.refprod+'<br>'+
                                         'Designation : '+this.desgprod+'<br>'+
                                         'Quantite : '+this.qte+'<br>');
                                }
                    };
    var p1 = new Produit('P001', 'ORANGE', 50)
    p1.affiche();
    </script>
```

Remarque :

- Lorsqu'on créé une fonction, le JavaScript va automatiquement lui ajouter une propriété **prototype** qui ne va être utile que lorsque la fonction est utilisée comme constructeur, c'est-à-dire lorsqu'on l'utilise avec la syntaxe **new**. L'intérêt de cette propriété est qu'il est possible d'ajouter des attributs et des méthodes partager par tous les objets créer à partir de ce constructeur.

```
<script>
      function Produit (refprod, desgprod, qte)
                  {
                    this.refprod = refprod;
                    this.desgprod = desgprod;
                    this.qte = qte;
                  };
    Produit.prototype.prix = 500;
    Produit.prototype.affiche  = function()
    {
        document.write('Reference : '+this.refprod+'<br>'+
                       'Designation : '+this.desgprod+'<br>'+
                       'Quantite : '+this.qte+'<br>');
    };
    var p1 = new Produit('P001', 'ORANGE', 50)
    p1.affiche();
    document.write(p1.prix+'<br>');
    document.write('*****************************<br>');
    var p2 = new Produit('P002', 'BANANE', 30)
    p2.affiche();
    document.write(p2.prix+'<br>');
    </script>
```

- Toutes les propriétés et méthodes doivent être précéder du mot clé **this**
- L'affectation n'est plus les : mais **=**

➲ Création d'objet avec le constructeur Object ()

```
<script>
    var p1 = new Object()
    p1.refprod = 'P001'
    p1.desgprod = 'Orange'
    p1.qte = 50;
    p1.affiche = function()
    {
        document.write('Reference : '+this.refprod+'<br>'+
                       'Designation : '+this.desgprod+'<br>'+
                       'Quantite : '+this.qte+'<br>');
    }
    p1.affiche();
</script>
```

➲ Création d'objet avec la méthode create () de Object

Les objets peuvent également être créés en utilisant la méthode Object.create(). Cette méthode peut s'avérer très utile, car elle permet de choisir le prototype pour l'objet qu'on souhaite créer, sans avoir à définir un constructeur.

```
<script>
    function Produit(ref, des, q) {
        this.refprod = ref,
        this.desgprod = des,
        this.qte = q,
        this.affiche = function()
         {
        document.write('Reference : '+this.refprod+'<br>'+
                       'Designation : '+this.desgprod+'<br>'+
                       'Quantite : '+this.qte+'<br>');
          }
    }

    let p1 = Object.create(new Produit('f001', 'fanta', 50));
    p1.affiche();
</script>
```

2. Lister les propriétés d'un objet

- Les boucles for...in qui permettent de parcourir l'ensemble des propriétés énumérables d'un objet.

```
<script>
    let p1 = Object.create(new Produit('f001', 'fanta', 50));
    for (const key in p1) {
        document.write(key+' : '+p1[key]+'<br>')
    }
</script>
```

- Object.keys(o) permet de renvoyer un tableau contenant les noms (clés ou *keys*) des propriétés propres (celles qui ne sont pas héritées via la chaîne de prototypes) d'un objet o pour les propriétés énumérables.

```
<script>
      var Rectangle =
      {
        lon : 12,
        larg : 14
      }
      var key = Object.keys(Rectangle);
      document.write(key.toString());
</script>
```

- La méthode **Object.getOwnPropertyNames()** renvoie un tableau de toutes les propriétés propres à un objet (c'est-à-dire n'étant pas héritées).

```
<script>
      var Rectangle = [12, 45]
      var key = Object.getOwnPropertyNames(Rectangle);
      document.write(key.toString());
</script>
```

3. Modifier les propriétés d'un objet

Un objet JavaScript possède plusieurs propriétés qui lui sont associées. Une propriété peut être vue comme une variable attachée à l'objet. Les propriétés d'un objet sont des variables attachées à un objet. Les propriétés d'un objet représentent ses caractéristiques et on peut y accéder avec une notation utilisant le point **« . »**, de la façon suivante :

```
let p1 = Object.create(new Produit('f001', 'fanta', 50));
    p1.affiche();
    document.write('************************<br>')
    p1.refprod = 'f002'
    p1.desgprod = 'banane'
    p1.affiche()
```

4. Les setters et les getters

Un setter est une fonction utilisé pour modifier les propriétés d'un objet cette fonction doit être précédée du mot **set**.
Un getter est une fonction utilisé pour afficher les propriétés d'un objet cette fonction doit être précédée du mot **get**.

```
var Rectangle =
{
    'lon' : 45,
    'larg' : 34,
    'affiche' : function()
    {
        print('longueur : '+this.lon+' largeur : '+this.larg)
    },
    set setLong(n){ this.lon = n},
    get getLong(){ return this.lon}
}
Rectangle.affiche()
Rectangle.setLong = 50 //appel du setter
Rectangle.affiche()
print('la largeur est : '+Rectangle.getLong) //appel du getter
```

NB : il est possible d'assurer d'ajouter des setters et des getters a un objet en faisant appel à la méthode static **defineProperties** de **Object.**

```
  let p1 = Object.create(new Produit('f001', 'fanta', 50));
    Object.defineProperties(p1,{
                                'getRefProd' : { get : function()
                                                    {
                                                      return this.refprod;
                                                    }
                                                },
                                'setRefProd' : { set : function(val)
                                                    {
                                                       this.refprod = val
                                                    }
                                                }
                            });
    p1.setRefProd = 'f003'; document.write(p1.getRefProd)
```

5. Supprimer les propriétés d'un objet

Il est possible de retirer des propriétés propres (celles qui ne sont pas héritées) grâce à l'opérateur **delete**.

```
<script>
    var Produit = {
        refprod : 'F001',
        desgprod : 'ORANGE',
        qte : 900,
        affiche : function()
         {
        document.write('Reference : '+this.refprod+'<br>'+
                       'Designation : '+this.desgprod+'<br>'+
                       'Quantite : '+this.qte+'<br>');
          }
    }
    delete Produit.qte
    let p1 = Object.create(Produit);
    if ('qte' in Produit)
    {
       p1.affiche()
    }
    else
    {
        document.write('Impossible d\'affciher la quantite')
    }
```

6. Comparer les objets

En JavaScript, les objets fonctionnent par référence. Deux objets distincts ne sont jamais égaux, même s'ils ont les mêmes valeurs pour les mêmes propriétés. On aura une équivalence uniquement si on compare deux références vers un seul et même objet donné.

```
<script>
      // Deux variables avec deux objets distincts
      // qui ont les mêmes propriétés
        let fruit1 = {nom: "diallo"};
        let fruit2 = {nom: "diallo"};
        fruit1 == fruit2  // renvoie false
        fruit1 === fruit2 // renvoie false
    </script>
```

Exercices d'application sur les Fonctions

NB : Pour chaque exercice de 1 à 8 utiliser une fonction nommée, une fonction anonyme et une fonctions fléchée

1- Ecrire une fonction vérifiant si un nombre entier naturel est un carré parfait, en utilisant seulement les opérateurs de base, et renvoie sa racine dans le cas favorable. (Indication : X est un carré parfait s'il existe un entier i tel que X = i * i.)
2- Ecrire un script JS qui, parmi N entiers naturels, calcul la somme et le produit des racines carrées des entiers carrés parfaits. Ensuite il vérifie si la somme et le produit sont des carrés parfaits.
3- Ecrire une fonction qui retourne Vrai si le caractère passé en paramètre est égal à 'o' ou 'O' (qui veut dire Oui), et Faux sinon. Ecrire une fonction qui permet d'afficher la table de multiplication de 1 à 9 d'un nombre entier positif. Puis, en utilisant la fonction précédente, écrire un script JS permettant d'afficher à l'utilisateur la table de multiplication d'un entier aussi longtemps qu'il le désire (jusqu'à ce que la réponse soit fausse).
4- Ecrire un script JS affichant tous les nombres inférieurs à 500 égaux à la somme des cubes de leurs chiffres. On utilisera une fonction UNITE, et une fonction CUBE. Exemple : 153 = 13 + 53 + 33 = 1 + 125 + 27
5- Ecrire un script JS affichant tous les nombres parfaits inférieurs à 10000. Sachant qu'un nombre entier positif (N) est parfait s'il est égal à la somme de ses diviseurs (<N). On écrira une fonction booléenne, appelée PARFAIT, pour vérifier si le nombre est parfait ou non parfait.
6- Ecrire une fonction BIN permettant de convertir un entier positif du décimal au binaire.
7- Écrire un script JS qui permette de connaître ses chances de gagner au tiercé, quarté, quinté et autres impôts volontaires. On demande à l'utilisateur le nombre de chevaux partants, et le nombre de chevaux joués. Les deux messages affichés devront être : Dans l'ordre : une chance sur X de gagner Dans le désordre : une chance sur Y de gagner X et Y nous sont donnés par la formule suivante, si n est le nombre de chevaux partants et p le nombre de chevaux joués (on rappelle que le signe ! signifie "factorielle") :

   X = n ! / (n - p) !
   Y = n ! / (p ! * (n – p) !)

Exercices d'application sur les Tableaux

1- Ecrire un script JS qui teste si tous les éléments d'un tableau sont présents dans un autre tableau.
2- Écrivez un programme JavaScript pour inverser une chaîne de caractères.
3- Ecrire un script JS qui créer une matrice contenant que des 1
4- Ecrire un script JS qui utilise la fonction filter pour renvoyer tous les nombres premiers d'un tableau.
5- Ecrire un script JS qui permet de rechercher un objet littéral dans un tableau grâce à la méthode find.
6- Ecrire un script JS qui prend un tableau de nombres et crée un nouveau tableau contenant les racines carrées des nombres du premier tableau en utilisant la fonction map.

Exercices d'application sur les Objets

EXERCICE 1

1- Définir un objet TEMPS qui contient les champs heure, minute, seconde.
2- Ecrire une fonction qui réalise la somme T de deux durées T1 et T2 de type temps.
3- Ecrire une fonction TRANSFORME qui transforme un temps T de type TEMPS en un entier S qui exprime ce temps en secondes.
4- Ecrire une fonction DECOMPOSE qui décompose un temps S exprimé en secondes en un temps T de type TEMPS.
5- Etant donnés deux temps T1 et T2 de type TEMPS, écrire un programme C qui calcule le temps T somme des temps T1 et T2 (T, T1 et T2 sont de type TEMPS) en utilisant les fonctions TRANSFORME et DECOMPOSE.

EXERCICE 2

Un nombre complexe Z est entièrement défini par ses parties réelle **a** et imaginaire **b** (Z = **a** + **bi**).
1- Créer un objet complexe
2- Ecrire les fonctions : **Reel**, **Image** et **Module** donnant les attributs d'un nombre complexe
respectivement : la partie réelle, la partie imaginaire et le module),
3- Ecrire les fonctions : **Somme**, **Difference** et **Produit** nécessaires à l'arithmétique sur les complexes, respectivement pour l'addition, la soustraction et la multiplication,
4- Ecrire une fonction **Conjugue** qui calcule le conjugué d'un nombre complexe.
5- Ecrire une fonction **Egale** qui teste l'égalité de deux nombres complexes.
6- Ecrire une fonction **Affiche** qui permet d'afficher un nombre complexe.