JavaScript

JavaScript

Pr. Aimad QAZDAR

Table des matières

Introduction

Le langage **JavaScript**, à ne pas confondre avec Java, est un langage de **script** ou **langage de programmation dynamique** très simple. Il permet de rendre un site web développé en HTML **dynamique**.

Parmi ces aspects dynamiques, je peux donner comme exemple :

- **Changer le contenu d'une page HTML** , *Cliquer ici pour voir (cf. exemple1.htm)*
- **Modifier les valeurs d'attribut HTML** ,*Cliquer ici pour voir (cf. exemple2.html)*
- **Changer le style d'un élément HTML** ,*Cliquer ici pour voir (cf. exemple3.html)*
- **Masquer des éléments HTML**, *Cliquer ici pour voir (cf. exemple4.html)*
- **Afficher des éléments HTML**, *Cliquer ici pour voir (cf. exemple5.html)*
- et d'autre exemple qu'on va découvrir durant ce chapitre.

I Script, c'est quoi?

1. **Qu'est qu'un Script ?**

   Un **script** désigne un programme (ou un mini-programme) chargé d'exécuter un traitement pré-définie quand un utilisateur réalise une action ou qu'une page web est en cours d'affichage sur un écran. Il s'agit généralement d'une suite de commandes simples et souvent peu structurées qui permettent l'automatisation de certaines tâches successives dans un ordre donné.

   Un script peut donc par exemple ouvrir un répertoire et crypter des fichiers qui s'y trouvent, ou modifier à la volée la taille d'une image à l'ouverture d'une page, etc.

2. **Type de Script**

   Nous distingons entre deux types de langages de script :

   1. **les scripts interprétés côté serveur** (c'est le cas des langages utilisés pour la création de sites web dynamiques comme PHP, Python, ASP, etc.). Le code n'est alors pas visible sur le code source de la page web car il est lu et interprété par la machine qui héberge le site (le serveur), puis envoyé vers l'appareil de l'utilisateur en format HTML.
   2. **Les scripts interprétés côté client**, envoyés sous leur forme brute à la machine cliente (celle de l'utilisateur). C'est le cas de JavaScript compris par le navigateur web et exécuté directement par celui-ci.

3. **Quelque langage de Script**

   Les langages de script sont très nombreux. Certains sont utilisés plus souvent que d'autres, voici quelques exemples parmi les plus courants :

   - Script shell : sh ; bash ; ksh ; zsh ; csh ; tcsh ; fish pour GNU/Linux et Unix ;
   - Cmd (anciennement Command), Windows PowerShell pour le monde Windows ;
   - Javascript,
   - Work Flow language,
   - Modern pascal,
   - Rebol,
   - PwerShell,
   - etc.

II Un peu d'histoire...

La première version du JavaScript a été crée en 1995 par Brendan Eich[1] au sein de la société Netscape Communications en association avec Sun Microsystem.

C'est le premier langage dit de "**scripting**", complètement conçu pour le développement Web et les navigateurs afin de fournir de l'interactivité et de la dynamique tout en augmentant le HTML.

Plus tard, Microsoft développera son propre langage Javascript officiellement connu sous le nom de JScript

En bref, l'histoire de JavaScript peut être raconter selon deux périodes :

- Avant le Web 2.0 : langage est peu utilisé, il est considéré comme un pseudo-langage à éviter pour des développements « sérieux » ;
- Après le Web 2.0 : la renaissance du langage qui est employé pour ajouter des fonctionnalités interactives aux pages web, grâce notamment à son intégration avec HTML5.

Le Javascript est standardisé par un comité spécialisé, l'ECMA - European Computer Manufacturers Association.

Ces dernières années, nous assistons à une nouvelle vision pour employer le JavaScript en dehors des pages web, notamment dans le développement côté-serveur ou dans le développement d'application desktop et mobile. Cette vision est bien résumé par la citation de Atwood: « any application that can be written in JavaScript, will eventually be written in JavaScript ».

III Versions de JavaScript

ECMAScript est le nom officiel du langage JavaScript.

Depuis 2015, ECMAScript est nommé par année (ECMAScript 2015)

Le tableau suivant représente une synthèse des versions du JavaScript

Ver	Official Name	Description
1	ECMAScript 1 (1997)	First Edition.
2	ECMAScript 2 (1998)	Editorial changes only.
3	ECMAScript 3 (1999)	Added Regular Expressions. Added try/catch.
4	ECMAScript 4	Never released.
5	ECMAScript 5 (2009)	Added "strict mode". Added JSON support. Added String.trim(). Added Array.isArray(). Added Array Iteration Methods.
5.1	ECMAScript 5.1 (2011)	Editorial changes.
6	ECMAScript 2015	Added let and const. Added default parameter values. Added Array.find(). Added Array.findIndex().
7	ECMAScript 2016	Added exponential operator (**). Added Array.prototype.includes.
8	ECMAScript 2017	Added string padding. Added new Object properties. Added Async functions. Added Shared Memory.
9	ECMAScript 2018	Added rest / spread properties. Added Asynchronous iteration. Added Promise.finally(). Additions to RegExp.

IV HTML et JavaScript

Le code JavaScript s'intègre de deux manières avec le code HTML :

1. **Insertion directe dans le code HTML**
2. **Insertion comme un module externe**

- **Insertion directe dans le code HTML**

Dans cette méthode le code JavaScript s'insère dans la page HTML elle même. C'est la méthode la plus simple et la plus fréquemment utilisée par les développeurs. Exemple :

```
<html>
  <head>
    <title>..... </title>
  </head>
  <body>
    <script language="JavaScript">
      alert('bonjour');
    </script>
  </body>
</html>
```

Le code JavaScript est placé dans le corps même de la page HTML, entre les balises <body> .... </body>. Et il est écrit entre les deux balises spécifiques **<SCRIPT>....</SCRIPT>**

Ce script sera exécuté automatiquement lors du chargement de la page HTML dans le navigateur en même temps que le contenu de la page HTML s'affiche à l'écran.

- **Insertion comme un module externe**

Dans cette méthode, on peut insérer du code JavaScript en faisant appel à un module externe se forme d'un fichier .js se trouvant à n'importe quelle adresse (URL).

Le fichier .js ne doit contenir que du JavaScript (pas d'HTML) comme suit :

```
alert("bonjour");
```

Pour appeler ce fichier, il suffit d'ajouter la balise **script** avec l'attribut **src** pour indiquer le chemin du script.

```
<html>
  <head>
    <title>Exemple1 </title>
  </head>
  <body>

    <script src='./script1.js'></script>
```

```
 8
 9   </body>
10 </html>
```

Les deux balises de JavaScript doivent être placés entre :

- Les Tags <body> ... </body> dans le cas d'une exécution directe
- Les Tags <head> ... </head> de la page HTML pour une exécution différée.

L'avantage de cette méthode c'est au lieu de modifier toutes les pages contenant le code en question, il n'y a qu'à modifier le code du module externe pour que toutes les pages faisant appel à lui bénéficient de la modification sans risque d'erreur.

Par contre, l'appel d'un code externe génère une requête supplémentaire vers le serveurs ce qui peut encombrer le réseau.

V Structurer votre script

La syntaxe du langage JavaScript s'appuie sur le modèle de Java et C (que vous connaissez déjà).

Voici quelques règles générales pour bien structurer votre script :

- Chaque commande doit être **terminée** par un **point-virgule** (;).
- Un **nombre à virgule** est séparé par un **point** (.) et non par une virgule.
- Le langage **Javascript est sensible à la casse**, par exemple la variable MonPrenom est différente de Monprenom
- Pour intégrer des commentaires à vos scripts, Il existe deux méthodes :
    - Placer un **double slash** (//) devant le texte à commenter
    - Encadrer le texte par **un slash suivi d'une étoile** (/*) et la même séquence inversée (*/)

VI Input / Output

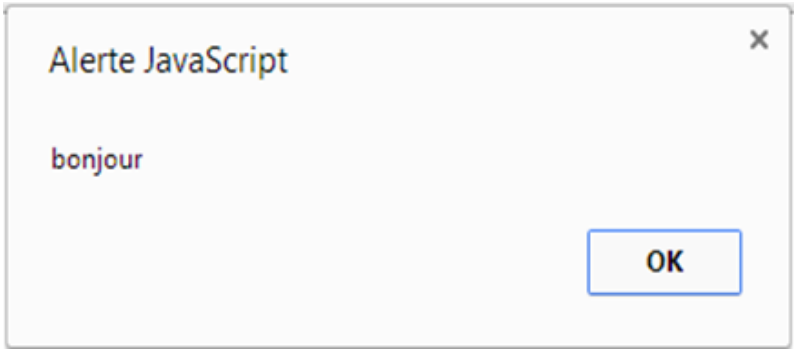
Comme dans toute langage de programmation le JavaScript contient des fonction qui permettent d'entrer ou de sortie des données, ce que nous appelons des boites de messages.

Il existe 3 types de boites de messages peuvent être affichés en utilisant Javascript : **Alerte**, **Confirmation** et **Invite**

- **Méthode alert()** : sert à afficher à l'utilisateur des informations simples de type texte. Une fois que l'internaute a lu le message, il doit cliquer sur OK pour faire disparaître la boîte.

  L'exemple suivant permet d'afficher une alerte contenant le texte Bonjour pour illustré dans la figure ci-dessus.

```
<html>
  <head>
    <title> une page simple </title>
  </head>
  <body>
    <script language='javascript'>
      alert('bonjour');
    </script>
  </body>
</html>
```



- **Méthode confirm()**

Cette méthode permet à l'utilisateur de choisir entre les boutons OK ou Annuler. Comme présenté dans l'exemple suivant :

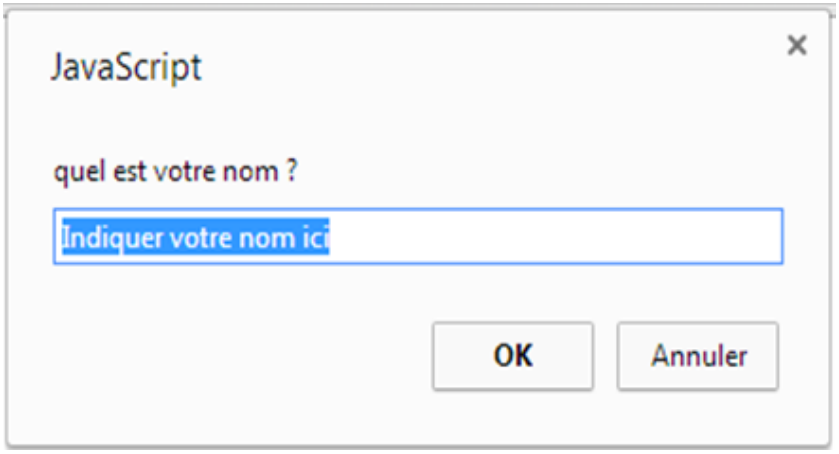
```
<html>
  <head>
    <title> une page simple </title>
  </head>
  <body>
    Bonjour
    <script language='javascript'>
      confirm('Etes vous sur de vouloire Quitter ce menu?');
    </script>
  </body>
</html>
```

- **Méthode prompt()**

La méthode prompt() permet d'inviter l'utilisateur à saisir son propre message en réponse à la question posée.

Exemple :

```
<html>
  <head>
    <title> une page simple </title>
  </head>
  <body>
    Bonjour
    <script language='javascript'>
        prompt('Quel est votre nom ?',
        'Indiquer votre nom ici');
    </script>
  </body>
</html>

```



Pour afficher le texte saisie par l'utilisateur dans la page web on utilise la méthode **document.write**.

La méthode document.write() permet d'écrire du code HTML dans la page WEB.

Un exemple d'utilisation est présenté dans le code suivant :

```
<script language='javascript'>
  document.write (
    '<b>',
    prompt('Quel est votre nom ?',
        'Indiquer votre nom ici'),
    '</b>');
</script>
```

Dans cet exemple, on récupère le texte saisir par l'utilisateur via la méthode **prompt()**puis on l'affiche dans la page HTML en **gras**. La méthode prompt() est placée entre les balise ....

VII Variables en JS

1. Introduction

Avant de commencer à parler sur les variables en JavaScript, nous allons faire un rappel sur la notions des variables dans les langage de programmation.

2. Qu'est ce qu'une variable?

Le but d'un programme informatique est de faire quelque un traitement avec des données. Votre programme utilise des variables pour enregistrer et manipuler ces données. Un nom d'utilisateur, le nombre de billets restants pour un vol, la disponibilité ou non d'un certain produit en stock, toutes ces données sont enregistrées dans des variables.

Généralement, une variable est définie par : son **Nom**, son **Type**, sa **Valeur** et sa **Porté**.

3. Déclaration et Affectation

Le mot-clé **var** permet de déclarer une ou plusieurs variables. Après la déclaration de la variable, il est possible de lui affecter une valeur par l'intermédiaire du signe d'égalité (**=**).

Si une valeur est affectée à une variable sans que cette dernière ne soit déclarée, alors JavaScript la déclare **automatiquement**. Mais notez bien que la lecture d'une variable non déclarée provoque une erreur.

Une variable est dite **indéfinie** (undefined), si elle est correctement déclarée mais dont aucune valeur n'est affectée.

Voici quelque exemple de déclaration des variables en JavaScript :

```
1 //Déclaration de i, de j et de k.
2 var i, j, k;
3
4 //Affectation de 1 à i.
5 i = 1;
6
7 //Déclaration et affectation de prix.
8 var prix = 0;
9
10 //Déclaration et affectation de caractère
11 var caractere = ["a", "b", "c"];
```

Pour la déclaration de variable en JavaScript, il existe quelque contraintes concernant les noms, par exemple:

- Les noms de variables ne peuvent contenir que des **lettres**, **chiffres**, ou le caractère "_" (underscore)

  Mon_Prenom est un nom valide

- Les caractères **spéciaux** et **accentués** sont interdits (é, à, ç, ï, etc..)

  Mon_Prénom n'est pas un nom valide. Il y a un caractère accentué.

- Les **majuscules** et les **minuscules** ont leur importance (JS est sensible à la casse, vous vous rappeler de ça).

  MonPrenom est différent de Monprenom.

- Un nom de variable ne peut contenir d'**espaces**.

  Mon Prenom n'est pas un nom de variable correct. Il y a un espace.

- Les mots **réservés JavaScript** ne peuvent être utilisés comme noms de variable.

4. Types de variables

En JavaScript, le **type** d'une variable dépend de la **valeur stockée** dans cette variable. Pour cela on n 'a pas besoin de preciser le type.

Prenons le script suivant :

```
<script language='javascript'>
      var maVariable = 'mohamed';
      document.write(maVariable);
      document.write('<br>');
      maVariable =10;
      document.write(maVariable);
</script>
```

Après avoir tester ce script, quel est le résultat afficher sur la page HTML ?

Le JavaScript présente trois principaux types de valeurs :

- **String**: chaîne de caractère ;
- **Number** : qui admet trois (03) valeurs spéciales
  - Positive Infinity ou +Infinity (valeur infini positive)
  - Negative Infinity ou –Infinity (valeur infinie négative)
  - Nan (Not a Number) habituellement générée comme résultat d'une opération mathématique incohérente
- **Boolean** : Deux valeurs littérales : true (vrai) et false (faux).

5. Portée de variables

Selon l'endroit où une variable est déclarée, celle-ci pourra être accessible de tout endroit dans le code ou bien uniquement dans une fonction dans le script, on parle de « **portée** » d'une variable (Scope)

Les variables en JavaScipt, comme en langage C, peuvent être **globales** ou **locales**.

- Une **variable globale** est **déclarée en début de script** et est **accessible à n'importe quel endroit du programme.**
- Une **variable locale** est **déclarée à l'intérieur d'une fonction** et **n'est utilisable que dans la fonction elle-même**.

Exemple :

```
<script type="text/javascript">
  //Variable globale.
  var varGlobal = 0;

  function Affiche() {
    //Variable locale
      var varLocal = 1;
  }
</script>
```

Dans cet exemple, la variable **varGlobal** est une variable **globale**. C-à-d, elle est reconnue dans **tout le script** y compris la fonction Affiche().

Par contre, la variable nommée varLocal est une variable **locale**, elle n'est pas reconnue que à **l'intérieur de la fonction où elle est déclarée**, la fonction Affiche() dans l'exemple.

ES2015 a introduit un nouveau mot clé JavaScript important: **let**.

Ce mot clé permet de déclarer des des variables **portée block** (Block Scope) reconnue dans un bloc de code.

Exemple :

```
var x = 10;
alert('x ='+x);
{ // Debut du bloc
  let x = 2;
  alert('x ='+x);
} // Fin du bloc
alert('x ='+x);
```

L'alert() en **ligne 2** va afficher **x =10** puisqu'on a déclaré x et lui affecté la valeur 10;

L'alert() de la **ligne 5** va afficher une nouvelle valeur de x (**x =2**); la variable x est déclarée avec le mot clé **let** donc elle reconnue seulement à l'intérieur du bloc (entre les accolades **{**.... **}**).

finalement le dernier alert() en **ligne 7** va afficher la premier valeur de x (**x =10**) puisque le deuxième x déclaré en ligne 4 est porté bloc.

VIII Les opérateurs

Nous distinguons trois types d'opérateurs : **Arithmétiques** , **Logiques**.

Opérateurs Arithmétiques

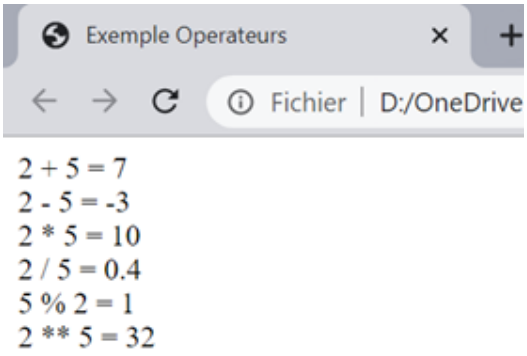
Les opérateurs arithmétiques utilisent des opérandes numériques et renvoie une valeur numérique.

- **+** : l'opérateur d'addition.
- **-** : l'opérateur de soustraction.
- **/** : l'opérateur de division.
- ***** : l'opérateur de multiplication.
- **%** : l'opérateur du reste.
- ****** : Opérateur de puissance (exponentiation).
- **+=** ou **-=** : si on écrit a+=b équivalent à a=a+b de même pour a-=b;

  ++ ou **--**: l'opérateur incrémentation / décrémentation , a++ équivalent à écrire : a=a+1. (a++) est différent de (++a)

Exemple :

```
var a=2;
var b=5;
document.write (
        a, ' + ', b, ' = ', a+b ,'</br>',
        a, ' - ', b, ' = ', a-b ,'</br>',
        a, ' * ', b, ' = ', a*b ,'</br>',
        a, ' / ', b, ' = ', a/b ,'</br>',
        b, ' % ', a, ' = ', b%a ,'</br>',
        a, ' ** ', b, ' = ', a**b ,'</br>',
    );
```



Recopier le code précédent est tester les autres opérateurs.

Voici les règles de précédence pour ces opérateurs du plus prioritaire au moins prioritaire : **() ➔ *** , **/, **, % ➔ -, + ➔ =**

Opérateurs Logiques

Un opérateur logique permet de comparer deux opérandes et de renvoyer une valeur booléenne selon le résultat de cette comparaison.

< : Opérateur inférieur strict.

> : Opérateur supérieur strict.

<= : Opérateur inférieur ou égal.

>= : Opérateur supérieur ou égal.

== : Opérateur d’égalité.

!= : Opérateur d'inégalité .

& : ET binaire (AND).

| : OU binaire (OR).

^ :OU exclusif binaire (XOR).

```
var a=2;
var b=5;
document.write (
        a, ' < ', b, ' = ', a<b ,'</br>',
        a, ' > ', b, ' = ', a>b ,'</br>',
        a, ' <= ', b, ' = ', a<=b ,'</br>',
        a, ' >= ', b, ' = ', a>=b ,'</br>',
        b, ' == ', a, ' = ', b==a ,'</br>',
        a, ' != ', b, ' = ', a!=b ,'</br>',
    );
```

Je vous laisse recopie et exécuter ce script pour voir le résultat qu'il va produire et n'oublie pas de tester tous les opérateurs.

IX Les opérateurs de chaîne

En JavaScript, il exit des opérateurs sur les variable de type «String» (chaîne de caractère).

Parmi ces opérateurs on trouve :

- **La concaténation**

L'opérateur (**+**) permet de concaténer deux chaînes de caractère.

Exemple :

```
1 var chaine = 'bonjour'+" "+ "les"+ " "+ 'SMI-3';
2 alert(chaine);
```

Ce script permet d'afficher une alert contenant le texte : bonjour les SMI-3

- **Déterminer la longueur d'une chaîne**

Pour déterminer la longueur d'une chaîne on utilise l'opérateur **length** comme suit :

```
1 var ch1 = "bonjour ";
2 var longueur = ch1.length;
3 alert(longueur);
```

Lorsque vous exécutez ce script vous aurez comme résultat la valeur 8. Pouvez-vous expliquer pourquoi ?

- **Identifier le nième caractère d'une chaîne**

```
1 var ch1 ="Rebonjour !";
2 alert(ch1.charAt(2));
```

Ce script permet d'afficher le caractère qui se trouve en 2 ieme position. Notez bien 0 est l'indice de la première position..

- **Extraction d'une partie de la chaîne**

```
1 var dateDuJour = '04/12/2020"
2 var mois = dateDuJour.substring(3, 5);
```

Le 3 dans **substring** représente l'indice du **premier caractère** de la sous-chaîne à extraire, c'est le 1. Et le 5 c'est l'indice du **dernier caractère** à prendre en considération c'est le deuxième slash (/) ; ce caractère ne fera pas partie de la sous-chaîne à extraire. Notez bien que les indices des chaînes commence de 0.

Donc la valeur affectée à la variable mois est 12.

X Les tableaux

Un **tableau** est une collection homogène de données, ordonnée et de taille statique. Chaque élément est repéré par un indice (son rang dans le tableau).

- **Créer un tableau**

L'utilisation du mot clé **array** est le moyen le plus simple de créer un tableau JavaScript.

Exemple :

```
1 var voitures = new Array('Toyota','Dacia', 'Skoda');
```

Ce code est équivalent à écrire :

```
1 var voitures = ['Toyota','Dacia', 'Skoda'];
```

- **Accéder aux éléments d'un tableau**

Vous accédez à un élément du tableau en vous référant à son numéro d'index.

L'instruction en ligne 2 du script suivant permet d'accéder à la valeur du deuxième élément dans les voitures.

```
1 var voitures = ['Toyota','Dacia', 'Skoda'];
2 var nom = voitures[1];
3 document.write(nom);
```

- **Modification d'un élément de tableau**

La ligne 03 du script suivant modifie la valeur du premier élément dans les voitures:

```
1 var voitures = ['Toyota','Dacia', 'Skoda'];
2 document.write("<p> <b> Avant </b> ", voitures[0], "</p>");
3 voitures [0] = "Opel";
4 document.write("<p> <b> Apres </b>  ", voitures[0], "</p>");
```

XI Instructions conditionnelles

1. Introduction

Très souvent, lorsque vous écrivez du code, vous souhaitez effectuer différentes actions pour différentes conditions. Pour cela vous faites appel aux instructions conditionnelles pour ce faire.

En JavaScript, nous avons les instructions conditionnelles suivantes :

- **if** pour spécifier un bloc de code à exécuter, si une condition spécifiée est vraie
- **else** pour spécifier un bloc de code à exécuter, si la même condition est fausse
- **else if** pour spécifier une nouvelle condition à tester, si la première condition est fausse
- **switch** pour spécifier de nombreux blocs de code alternatifs à exécuter

2. L'instruction <if>

L'instruction <**if**> a pour spécifier un bloc de code JavaScript à exécuter si une condition est vraie.

Exemple :

Afficher un message "Bonne journée" si l'heure est inférieure à 18h00

```
1 var heure =10;
2 if (heure < 18) {
3   message = "Bonne journée";
4 }
5 alert(message)
```

- **L'instruction <if-else>**

L'instruction <**if-else**> permet de spécifier un bloc de code à exécuter si la condition est fausse.

Exemple :

Si l'heure est inférieure à 18, afficher un message d'accueil «Bonne journée», sinon «Bonsoir»:

```
1 var heure =19;
2 if (heure < 18) {
3   message = "Bonne journée";
4 }else{
5   message = "Bonne soir";
6 }
7 alert(message);
```

- **L'instruction <else- if>**

Cette instruction permet de spécifier une nouvelle condition si la première condition est fausse.

Par exemple : Si l'heure est inférieure à 10h00, créez un message d'accueil «Bonjour», sinon, mais l'heure est inférieure à 20h00, créez un message d'accueil «Bonjour», sinon un «Bonsoir»:

```
1 var heure =14;
2 if (heure < 12) {
3   message = "Bonne journée";
4 }else if (heure < 18) {
5   message = "Bonne apres-midi";
6 }else{
7   message = "Bonne soir";
8 }
9 alert(message);
```

3. L'instruction <switch>

L'instruction <**switch**> est utilisée pour effectuer différentes actions en fonction de différentes conditions.

Elle permet de sélectionner l'un des nombreux blocs de code à exécuter.

Exemple :

L'exemple utilise le numéro du jour de la semaine pour afficher le nom du jour de la semaine.

```
switch (new Date().getDay()) {
  case 0:
    day = "Dimanche";
    break;
  case 1:
    day = "Lundi";
    break;
  case 2:
     day = "Mardi";
    break;
  case 3:
    day = "Mercredi";
    break;
  case 4:
    day = "Jeudi";
    break;
  case 5:
    day = "Vendredi";
    break;
  case 6:
    day = "Samedi";
}
alert(day);
```

La méthode **getDay ()**, de l'objet **Date()**, utilisée dans cet exemple renvoie le jour de la semaine sous la forme d'un nombre compris entre 0 et 6.

(Dimanche = 0, lundi = 1, mardi = 2, etc)

XII Instructions répétitives

1. Introduction

Les instructions répétitives appelé aussi des boucles sont utilisées pour exécuter un bloc de code plusieurs fois.

C'est souvent le cas lorsque vous travaillez avec des tableaux:

Au lieu d'écrire:

```
var voitures = ['Toyota','Dacia', 'Skoda','Mercedes', 'Renault','BMW'];
var text='';
text += voitures[0] + "<br>";
text += voitures[1] + "<br>";
text += voitures[2] + "<br>";
text += voitures[3] + "<br>";
text += voitures[4] + "<br>";
text += voitures[5] + "<br>";
document.write(text);
```

Vous écrivez :

```
var i;
for (i = 0; i < voitures.length; i++) {
  text += voitures[i] + "<br>";
}
document.write(text);
```

Notez bien que **voitures.length** permet de récupérer la taille du tableau.

JavaScript prend en charge différents types de boucles :

- **for**
- **while**
- **do - while**

2. Boucle <for>

En JavaScript nous avons trois (03) types de boucles <for> :

- **for** qui parcourt un bloc de code plusieurs fois
- **for - in** qui boucle les propriétés d'un objet
- **for - of** qui parcourt sur les valeurs d'un objet itérable

La boucle<for> :

Comme dans tous les langages de programmation, la boucle <for>permet de répéter un bloc de code plusieurs fois. La syntaxe de la boucle <for> est la suivante :

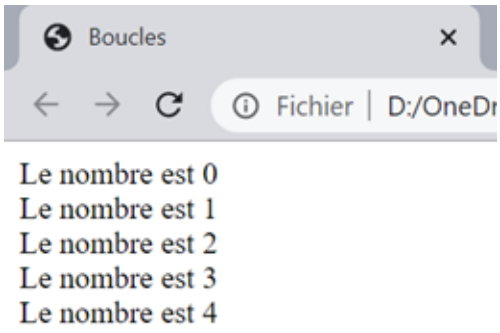
```
for (expression 1; expression 2; expression 3) {
  // bloc de code à executer
}
```

- L'expression 1 est exécutée (une fois) avant l'exécution du bloc de code.
- L'expression 2 définit la condition d'exécution du bloc de code.
- L'expression 3 est exécutée (à chaque fois) après l'exécution du bloc de code.

Exemple :

```
var text;
var i;
for (i = 0; i < 5; i++) {
  text='';
  text += "Le nombre est " + i + "<br>";
  document.write(text);
}
```

Le résultat de ce script est affiché dans la figure suivante :



La boucle <for-in> et <for-of> :

Exemple d'utilisation de <**for-in**> et <**for-of**>

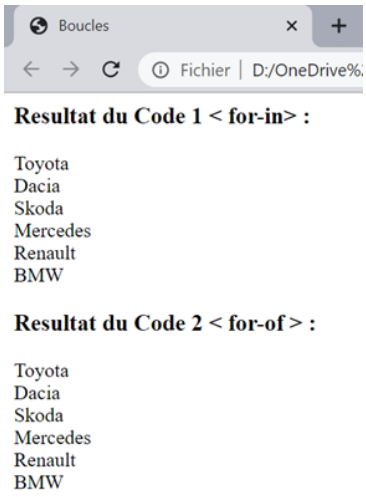
Code 1 :

```
var voitures = ['Toyota','Dacia', 'Skoda','Mercedes', 'Renault','BMW'];
var x;
for (x in voitures) {
  document.write(voitures[x],"<br>");
}
```

Code 2 :

```
var voitures = ['Toyota','Dacia', 'Skoda','Mercedes', 'Renault','BMW'];
var x;
document.write("<b> for-of :</b><br>");
for (x of voitures) {
document.write(x,"<br>");
}

```

La figure suivante présente le résultat de ces deux codes :



Comme vous remarquer les deux code produit le même résultat. Pour mieux comprendre pourquoi, la variable **x** dans <**for-in**> contient l'**indice** de la voiture, tandis que la variable **x** dans <**for-of**> contient la **voiture** elle même.

3. Boucle <while>

La boucle **<while>** permet de répéter un bloc de code alors qu'une condition spécifiée est vraie.

Dans l'exemple suivant, le code de la boucle s'exécutera, encore et encore, tant qu'une variable (i) est inférieure à 10:

```
var i=0;
var text;
while (i < 10) {
  text='';
  text += "Le nombre est " + i;
  i++;
  document.write('<p>', text);
}
```

Recopier ce script et tester le pour voir le résultat.

La boucle < **do- while**> est une variante de la boucle while. Cette boucle exécutera le bloc de code une fois, avant de vérifier si la condition est vraie, puis elle répétera la boucle tant que la condition est vraie.

L'exemple ci-dessous utilise une boucle <do- while>. La boucle sera toujours exécutée au moins une fois, même si la condition est fausse, car le bloc de code est exécuté avant que la condition ne soit testée:

```
do {
  text='';
  text += "le nombre est " + i;
  i++;
  document.write('<p>', text);
}
while (i < 10);
```

Recopier le deuxième code, exécuter le et comparer son résultat avec le résultat du premier.

4. <Break> et <Continue>

L'instruction <**break**> permet de faire un "saute" d'une boucle.

L'instruction <**continue**> "saute par dessus" d'une itération dans la boucle.

- **L'instruction <break>** :

Vous avez déjà vu l'instruction <break> utilisée dans *la partie* *(cf. p.21)* précédente de ce cours. Il était utilisé pour "sortir" de l'instruction switch ().

L'instruction <break> peut également être utilisée pour sortir d'une boucle.

Exemple :

```
var i;
var text;
for (i = 0; i < 10; i++) {
  text='';
  if (i == 6) { break; }
  text += "Le nombre est " + i + "<br>";
   document.write('<p>', text);
}
```

Dans cet exemple, break permet de sortir de la boucle quand (i==6) même si la boucle doit se répéter tant que (i<10).

- **L'instruction <continue>** :

On va reprendre le même exemple précédent et on va changer <break> par <continue> comme suit :

```
var i;
var text;
for (i = 0; i < 10; i++) {
  text='';
  if (i == 6) { continue; }
  text += "le nombre est " + i + "<br>";
   document.write('<p>', text);
}
```

Cet exemple permet l'affichage du texte contenant la valeur de (i) de 0 à 9 mais en **sautant** la valeur 6.

XIII Les fonctions

1. Introduction

Une fonction JavaScript est un **bloc de code** conçu pour effectuer une **tâche particulière**.

Une fonction JavaScript est exécutée lorsque "quelque chose" l'invoque (l'**appelle**).

Un fonction reçoit une ou plusieurs informations à partir desquelles elle retourne une ou plusieurs informations.

Les informations fournies à la fonction sont appelées **arguments** ou **paramètres** de la fonction.

Les informations renvoyées par la fonction sont appelées **résultat**.

Pourquoi Les fonctions?

Les fonctions permettent la **réutilisation du code**. C-à-d, vous définissez le code une fois et utilisez-le plusieurs fois.

Vous pouvez utiliser le même code plusieurs fois avec des arguments différents, pour produire des résultats différents.

Type de fonctions

En JavaScript, il existe deux types de fonctions :

- Les fonctions que vous **définissez vous-même**.
- Les fonctions propres à JavaScript, appelées **méthodes**.

2. Définition et Appel de fonction

- **Définition de fonction**

Une fonction JavaScript est définie avec le mot-clé **function**, suivi d'un nom, suivi de parenthèses ().

Les noms de fonction peuvent contenir des lettres, des chiffres, des traits de soulignement et des signes dollar (mêmes règles que les variables).

Les parenthèses peuvent inclure des noms de **paramètres** séparés par des virgules: **(** paramètre1**,** paramètre2**, ...)**

Le **code à exécuter**, par la fonction, est placé entre accolades: **{ }**

La valeur de retour est placée après le mot clé **return**, (cette ligne est facultative)

```
function nom_de_la_fonction (parametre1, parametre2, parametre3) {
   // code à exécuter
 return(valeur_de_retour); //Facultatif
}
```

- **Appel de fonction**

Le code à l'intérieur de la fonction s'exécutera lorsque "quelque chose" invoquera (appellera) la fonction:

Lorsqu'un événement se produit (lorsqu'un utilisateur clique sur un bouton)

Lorsqu'il est appelé (appelé) à partir du code JavaScript

Automatiquement (auto-invoqué)

Pratiquement, voici comment appeler une fonction :

```
nom_de_la_fonction (parametre1, parametre2, parametre3);
```

Exemple de fonction pour calculer le produit de deux nombres et retourner résultat :

```
var x = myFunction(4, 3);   // La fonction est appelée, la valeur de retour sera stocké dans x
document.write(x);

//Définition de la fonction
function myFunction(a, b) {
  return a * b;             // La fonction renvoie le produit de a et b
}
```

3. Les fonctions prédéfinies

Une fonction prédéfinie est un "mini-Script" autonome auquel vous pouvez demander de faire un traitement par passage d'un ou de plusieurs paramètres. Une fois le traitement est terminé, la fonction renvoie une valeur de retour (la réponse) au demandeur.

Le JavaScript dispose de plusieurs fonctions prédéfinies, je vous présente ici quelque exemple :

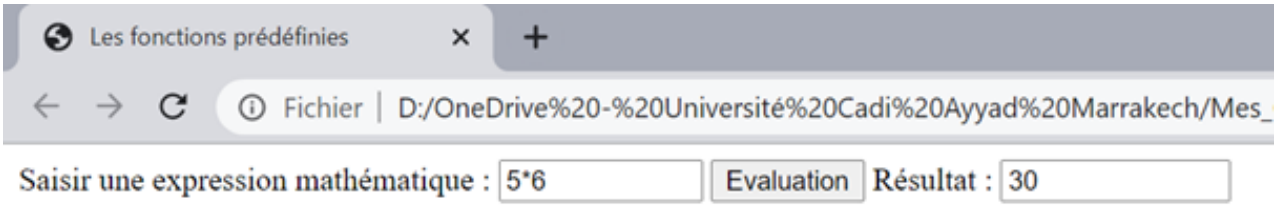
- **eval**

  Cette fonction permet de calculer une formule mathématique à partir d'une chaîne de caractères.

  Exemple :

```
<html>
  <head>
    <meta charset="utf-8">
    <title> Les fonctions prédéfinies </title>
  </head>
  <body>
    <SCRIPT LANGUAGE="JavaScript">
      function evaluation()  {
            document.formulaire.calcul.value=eval(document.formulaire.saisie.value);
      }
    </SCRIPT>
    <FORM NAME="formulaire">
      Saisir une expression mathématique :
      <INPUT TYPE="text" NAME="saisie" MAXLENGTH=10 SIZE=10>
      <INPUT TYPE="button" VALUE="Evaluation" onClick="evaluation()">
      Résultat : <INPUT TYPE="text" NAME="calcul" MAXLENGTH=12 SIZE=12>
    </FORM>
  </body>
</html>
```

Ce code permet de générer la page suivante. Sur cette page, vous pouvez saisir une formule mathématique sous forme d'une chaîne de caractère et en cliquant sur le bouton "Evaluation" le script va récupérer la formule depuis la chaîne saisie, l'exécuter puis retourner le résultat.



Noter bien que l'appel de la fonction se fait suite au clique de utilisateur sur le bouton "Evaluation" (l'événement **onClick**).

- **isFinite**

  Détermine si le paramètre est un nombre fini. Cette fonction renvoie false si ce n'est pas un nombre ou l'infini positif ou infini négatif.

  Exemple :

```
isFinite(240) //retourne true
isFinite("Un nombre") //retourne false
```

- **isNaN**

  Détermine si le paramètre n'est pas un nombre (NaN : Not a Number).

  Exemple :

```
isNaN("un nombre") //retourne true
isNaN(20) //retourne false
```

- **parseFloat**

  Analyse une chaîne de caractères et retourne un nombre décimal. Si l'argument évalué n'est pas un nombre, renvoie NaN (Not a Number).

```
var numero="125";
var nombre= parseFloat(numero); //retourne le nombre 125
```

- **parseInt**

  Analyse une chaîne de caractères et retourne un nombre entier de la base spécifiée. La base peut prendre les valeurs **16** (hexadécimal) **10** (décimal), **8** (octal), **2** (binaire).

  L'exemple suivant permet de retourner un nombre entier de la base décimale :

```
var nbr=30.75;
var arrondi = parseInt(nbr, 10); //retourne 30
```

- **Escape**

  Retourne la valeur hexadécimale à partir d'une chaîne de caractère codée en ISO-Latin-1.

  Exemple :

```
escape("$&") //retourne %24%26
```

Notez bien que les fonctions prédéfinies (méthodes) sont associées à un **objet** en particulier. Nous allons voir par la suite les différents objets en JavaScript et leurs méthodes.

XIV Les Objets

1. Introduction

Les **objets** de JavaScript, sont soit des entités **pré-définies** du langage, soit **créés par le programmeur**.

Dans cette partie, nous allons focaliser sur les objets prédéfinies .

Parmi les objets prédéfinies de JavaScript, on trouve :

- Le navigateur est un objet qui s'appelle "**navigator**".
- La fenêtre du navigateur se nomme "**window**"
- La page HTML est un autre objet, que l'on appelle "**document**".
- Un formulaire "**Form**" à l'intérieur d'un "document", est aussi un objet.
- Un lien hypertexte dans une page HTML, est encore un autre objet. Il s'appelle "**link**".
- etc...

2. Création d'un objet

L'opérateur <**New**> est utilisé pour créer :

- Une nouvelle instance
- Un **nouveau objet** l'un des **types d'objets** prédéfinis : Array, Number, String, Boolean, Date, Function, Image ou Object.
- Un nouveau type d'objet défini par l'utilisateur

syntaxe :

```
nouvel_objet = new type_objet(parametres);
```

Pour connaître le type d'un objet vous pouvez utiliser l'opérateur <**Typeof**>.

En effet, l'opérateur typeof renvoie une chaîne de caractères indiquant quel est le type de l'opérande.

Exemples :

```
var i = 1;
typeof i; //retourne number

var titre="La Gloire de mon père";
typeof titre; //retourne string

var jour = new Date();
typeof jour;  //retourne object

var choix = true;
typeof choix; //retourne boolean

var cas = null;
typeof cas; //retourne object

typeof parseFloat; //retourne function

typeof Math; //retourne object
```

Les objets prédéfinis disposent d'un ensemble de méthodes qui permettent de faciliter la manipulation des ces objets.

Dans les sections qui suivent nous allons voir quelques objets et leurs méthodes avec des exemples d'utilisation.

3. L'objet <String>

Voici quelques méthodes de "**String**" qui vous aident à travailler avec des chaînes :

- **indexOf()**

  Cette méthode admet en paramètre un texte et retourne l'index (la position) de la première occurrence du texte spécifié dans la chaîne à partie du début.

- **lastIndexOf()**

  Renvoie l'index du texte spécifié dans une chaîne à partir de la fin. Les deux premières méthodes acceptent un deuxième paramètre comme position de départ de la recherche.

- **search()**

  Recherche un texte et renvoie sa position dans la chaîne.

- **toUpperCase()**

  Convertir une chaîne en majuscules.

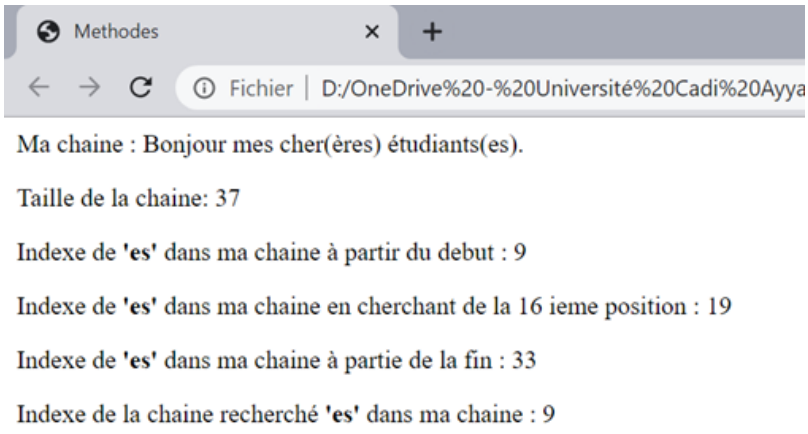
Si la chaîne passée en paramètre n'existe pas ces méthodes renvoient la valeur **-1**.

D'autres méthode pour **String** telles que : **slice()**, **substring()**, **replace()**, etc. sont disponible sur cette page[2].

Exemple d'utilisation de ces méthodes :

```
1  var str = 'Bonjour mes cher(ères) étudiants(es).';
2
3  const chainRech = 'es';
4  var i=16;
5
6  var taille = str.length;
7
8  var posF = str.indexOf(chainRech);
9  var posFi = str.indexOf(chainRech,i);
10 var posL = str.lastIndexOf(chainRech);
11 var ser = str.search(chainRech);
12
13 document.write('<p> Ma chaine : ',str);
14 document.write('<p> Taille de la chaine: ',taille);
15 document.write("<p> Indexe de <b>'", chainRech ,"'</b> dans ma chaine à partir du
   debut  : ",posF);
16 document.write("<p> Indexe de <b>'", chainRech ,"'</b> dans ma chaine en cherchant de
   la ", i ," ieme position  : ",posFi);
17 document.write("<p> Indexe de <b>'", chainRech ,"'</b> dans ma chaine à partie de la
   fin : ",posL);
18 document.write("<p> Indexe de la chaine recherché <b>'", chainRech ,"'</b> dans ma
   chaine : ",ser);
19 document.write("<p> La chaine  <b>'", chainRech ,"'</b> en majiscule : ",chainRech
   .toUpperCase());
```

Le résultat de cet exemple est illustré dans la figure suivante :



4. L'objet <Number>

Les méthodes "**Number**" vous aident à travailler avec les nombres et voici quelque-unes :

- **toString()**

  Permet de retourner un nombre sous forme de chaîne.
- **toExponential ()**

  Renvoie une chaîne, avec un nombre arrondi et écrit en notation exponentielle.
- **toFixed ()**

  Renvoie une chaîne, avec le nombre écrit avec le nombre de chiffre après la virgule spécifié.
- **toPrecision ()**

  Renvoie une chaîne, avec un nombre écrit en respectant la longueur spécifiée.
- **Number ()**

  Peut être utilisé pour convertir des variables JavaScript en nombres.
- **parseInt()**

  Analyse une chaîne de caractères et retourne un nombre entier de la base spécifiée. La base peut prendre les valeurs 16 (hexadécimal) 10 (décimal), 8 (octal), 2 (binaire).
- **isFinite()**

  Détermine si le paramètre est un nombre fini. Renvoie false si ce n'est pas un nombre ou l'infini positif ou infini négatif.
- **isNaN()**

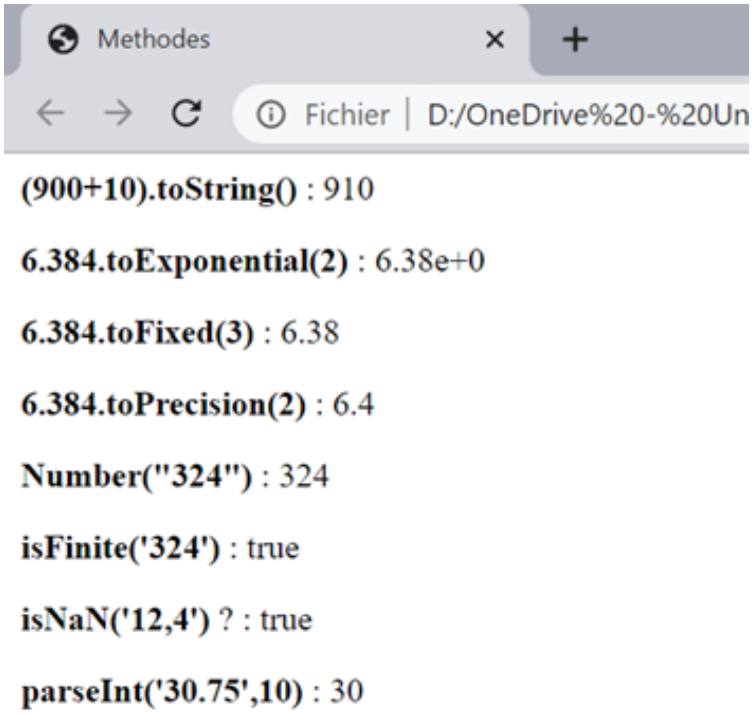
  Détermine si le paramètre n'est pas un nombre (NaN : Not a Number).

Exemple :

```
var x = 900;
document.write('<p>(',x,'+10).toString() : ',(x+10).toString());

x = 6.3840;
document.write('<p>',x,'.toExponential(2) : ', x.toExponential(2));
document.write('<p>',x,'.toFixed(3) : ', x.toFixed(2));
document.write('<p>',x,'.toPrecision(2) : ', x.toPrecision(2));

var str="324";
document.write('<p>Number("',str,'") : ', Number(str));

var prix=30.75;
document.write("<p> Le prix  <b>'", prix ,"'</b> arrindi : ",parseInt(prix, 10));
```

```
14
15 var str="
```

La figure suivant présente le résultat de ce code :



Autres méthode pour **Number** sont disponible sur cette page[3].

5. L'objet <Array>

Les méthodes "**Array**" vous seront très utiles pour manipuler les tableaux.

Parmi ces méthodes on a :

- **toString()** :

  Convertit un tableau en une chaîne de valeurs de tableau (séparées par des virgules).
- **pop()** :

  Supprime le dernier élément d'un tableau.
- **shift()** :

  Supprime le premier élément du tableau et "décale" tous les autres éléments vers un index inférieur.
- **splice()** :

  Utiliser pour ajouter de nouveaux éléments à un tableau.

Exemples :

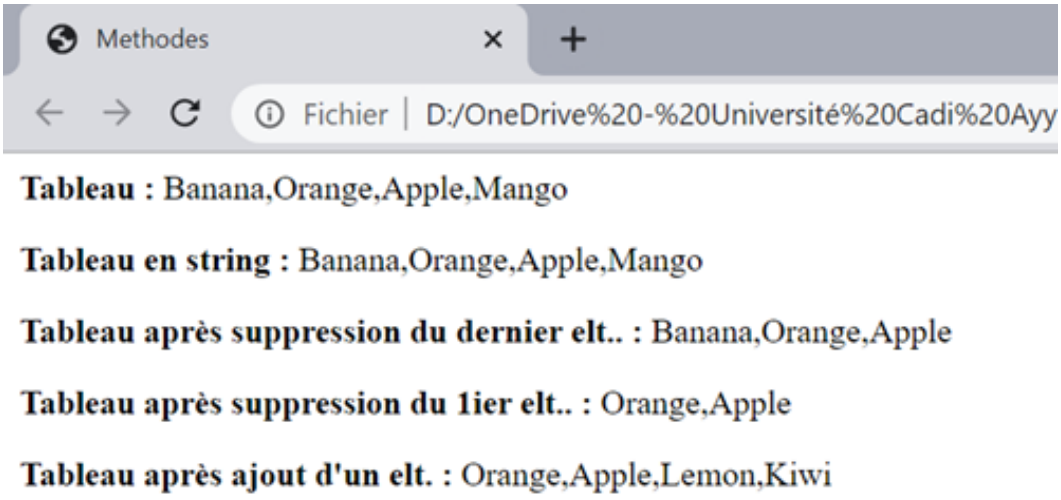
```
1 var fruits = ["Banana", "Orange", "Apple", "Mango"];
2 document.write("<p><b>tableau : </b>",fruits );
3
4 document.write("<p><b>tableau en string : </b>", fruits.toString());
5
6 fruits.pop();              // Removes the last element ("Mango") from fruits
7 document.write("<p><b>tableau après suppression du dernier elt.. : </b>", fruits);
8
9 fruits.shift();            // Removes the first element "Banana" from fruits
10 document.write("<p><b>tableau après suppression du 1ier elt.. : </b>", fruits);
11
12 fruits.splice(2, 0, "Lemon", "Kiwi");
```

```
13 document.write("<p><b>tableau après ajout d'un elt. : </b>", fruits);
```

NB : Pour cet exemple **fruits.splice(2, 0, "Lemon", "Kiwi")**

- Le premier paramètre (2) définit la position où les nouveaux éléments doivent être ajoutés (épissés).
- Le deuxième paramètre (0) définit le nombre d'éléments à supprimer.
- Le reste des paramètres ("Lemon", "Kiwi") définissent les nouveaux éléments à ajouter.

La figure suivante présente le résultat de ces methodes :



Plus de méthode <Array> sont disponible sur ce lien[4].

6. L'objet <Date>

Les méthodes "Date" permettent la manipulation des dates.

Dans ces méthodes nous distinguos entre deux types les **Getter** et les **Setter**.

- **Getter** ou appelé les methodes Get permettent de récupérer la date.
- **Setter** ou appelé les methodes Set sont utilisée pour modifier la valeur de la date.

Quelques exemples de méthode "Date" :

- **getDate ()**

  Renvoie le jour d'une date sous forme de nombre (1-31)
- **getHours ()**

  Renvoie les heures d'une date sous forme de nombre (0-23):
- **getMinutes ()**

  Renvoie les minutes d'une date sous forme de nombre (0-59):
- **setFullYear ()**

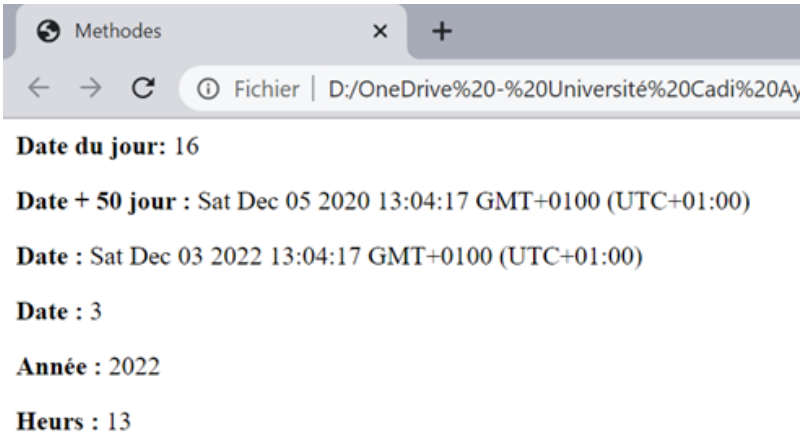
  Permet de définir l'année, le mois et le jour d'un objet Date.
- **setDate ()**

  Est utilisée pour ajouter des jours à une date.

Exemples :

```
1 var d = new Date();
2
3 document.write("<p><b>Date du jour: </b>", d.getDate());
4
5 d.setDate(d.getDate() + 50);
6 document.write("<p><b>Date + 50 jour : </b>",d);
```

```
7
8 d.setFullYear(2022, 11, 3);
9
10 document.write("<p><b>Date : </b>", d);
11 document.write("<p><b>Date : </b>", d.getDate());
12 document.write("<p><b>Année : </b>", d.getFullYear());
13 document.write("<p><b>Heurs : </b>", d.getHours());
```



Plus de méthode de Get et Set Date sont disponible respectivement sur ces pages : Getter Date[5] et Setter Date.

NB : Les dates peuvent être facilement comparées.

L'exemple suivant compare la date du jour au 14 janvier 2023:

```
1 var today, someday, text;
2
3 today = new Date();
4 someday = new Date();
5 someday.setFullYear(2023, 0, 14);
6
7 if (someday > today) {
8   text = "Aujourd'hui est avant le  14 Janvier 2023.";
9 } else {
10   text = "Aujourd'hui est après le  14 Janvier 2023.";
11 }
12 document.write("<p><b>", text,"</b>");
13
```

Vous pouvez recopier et tester ce script.

XV Les objets du navigateur

1. Introduction

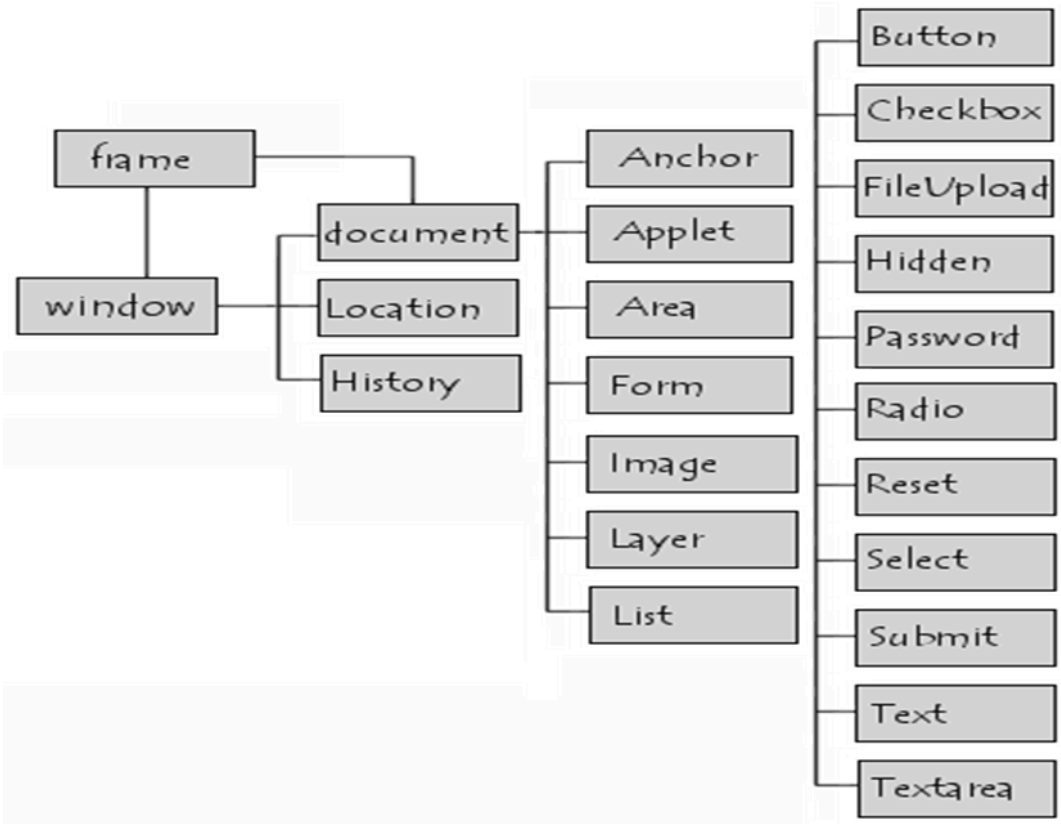
Quand vous lancer une page Web, le navigateur crée des objets prédéfinis correspondant à la page Web, à l'état du navigateur. Les objets crées permettent d'extraire beaucoup d'informations qui vous seront utiles.

Les objets de base du navigateur sont :

- **navigator** : qui contient des informations sur le navigateur de l'internaute.
- **window** : c'est l'objet où s'affiche la page, il contient donc des propriétés qui concerne la fenêtre et les objets-enfants contenus dans la fenetre
- **location** : contient des informations relatives à l'adresse de la page affichée
- **history**: c'est l'historique de liens qui ont été visités précédemment
- **document** : il contient les propriétés sur le contenu du document (titre, couleur d'arrière plan, etc.)

2. Hiérarchie des objets <Navigator>

La figure suivante représente la structure hiérarchique des objets du navigateur. L'accès à n'importe quel objet est possible grâce à une désignation dépendant de la hiérarchie, c-à-d, on part du sommet puis on descend l'arborescence.



Selon cette hiérarchie, les descendants d'objets représentent des propriétés de ces objets, mais ils peuvent aussi être des objets qui contiennent eux-mêmes des propriétés , etc.

L'objet le plus haut dans la hiérarchie est **window** qui correspond à la fenêtre même du navigateur.

L'objet document fait référence au contenu de la fenêtre. Cet objet regroupe l'ensemble des éléments HTML présents sur la page.

Pour atteindre ces différents éléments, nous utiliserons :

1. soit des **méthodes propres** à l'objet document, comme la méthode getElementById( ), qui permet de trouver l'élément en fonction de son identifiant (ID);
2. soit des **collections d'objets** qui regroupent sous forme de tableaux JavaScript tous les éléments de type déterminé.

Dans l'exemple suivant nous accédons à l'objet document via sont **id** qu'est égal à "demo" :

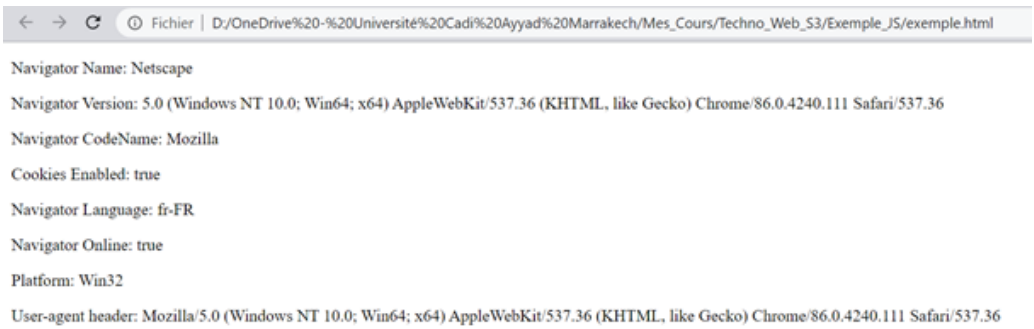
```
<body>
    <p id="demo">Cliquer sur le bouton pour changer le texte de ce paragraphe.</p>

    <button onclick="myFunction()">Essayer-le</button>

    <script>
      function myFunction() {
        document.getElementById("demo").innerHTML = "Hello World";
      }
    </script>
</body>
```

Quelques propriétés de l'objet <navigator> :

- **appName** : application (Netscape, Internet Explorer)
- **appVersion** : numéro de version.
- **platform** : système d'exploitation (Win32)
- **language** : langage du navigateur

D'autre propriete sont presenté dans l'exemple suivant :

```
 <body>

   <div id="demo"></div>

   <script>
     var txt = "";
     txt += "<p>Navigator Name: " + navigator.appName + "</p>";
     txt += "<p>Navigator Version: " + navigator.appVersion + "</p>";
     txt += "<p>Navigator CodeName: " + navigator.appCodeName + "</p>";
     txt += "<p>Cookies Enabled: " + navigator.cookieEnabled + "</p>";
     txt += "<p>Navigator Language: " + navigator.language + "</p>";
     txt += "<p>Navigator Online: " + navigator.onLine + "</p>";
     txt += "<p>Platform: " + navigator.platform + "</p>";
     txt += "<p>User-agent header: " + navigator.userAgent + "</p>";

     document.getElementById("demo").innerHTML = txt;
   </script>
 </body>
```



Remarque :

La valeur renvoyée par "appName" varie selon les navigateurs:

- IE11, Firefox, Chrome et Safari renvoie "Netscape"
- IE 10 et les versions antérieures renvoient «Microsoft Internet Explorer»
- Opera retourne "Opera"

3. <window>

L'objet <**Window**> représente l'objet où s'affiche la page, il contient donc des propriétés qui concerne la fenêtre et les objets-enfants contenus dans la fenêtre ainsi que des méthodes qui facilitent la manipulation de l'objet <window> .

Parmi les propriétés de l'objet window, nous avons :

- **closed** : Indique que la fenêtre a été fermée;
- **defaultStatus** : Indique le message par défaut dans la barre de statuts;
- **document** : Retourne l'objet document de la fenêtre;
- **frames** : Retourne la collection de cadres dans la fenêtre;
- **history** : Retourne l'historique de la session de navigation;
- **location** : Retourne l'adresse actuellement visitée;
- **name** : Indique le nom de la fenêtre;
- **navigator** : Retourne le navigateur utilisé;
- **opener** : Retourne l'objet window qui a créé la fenêtre en cours;
- **parent** : Retourne l'objet window immédiatement supérieur dans la hiérarchie;
- **self** : Retourne l'objet window correspondant à la fenêtre en cours;
- **status** : Indique le message affiché dans la barre de status;
- **top** : Retourne l'objet window le plus haut dans la hiérarchie.

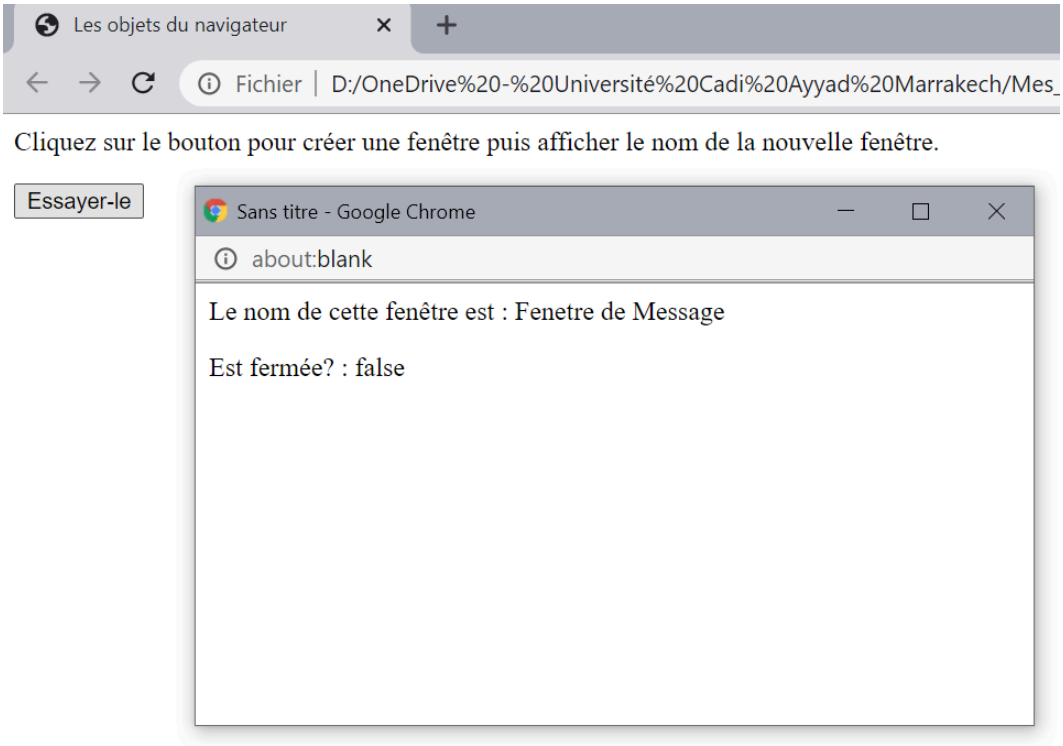
Concernant les méthodes de cet objet, nous pouvons citer :

- **blur( )** : Enlève le focus de la fenêtre;
- **close( )** : Ferme la fenêtre;
- **focus( )** : Place le focus sur la fenêtre;
- **moveBy( )** : Déplace d'une distance;
- **moveTo( )** : Déplace la fenêtre vers un point spécifié;
- **open( )** : Ouvre une nouvelle fenêtre;
- **print( )** : Imprime le contenu de la fenêtre;
- **resizeBy( )** : Redimensionne d'un certain rapport;
- **resizeTo( )** : Redimensionne la fenêtre;
- **setTimeout( )** : Évalue une chaîne de caractère après un certain laps de temps.

L'exemple suivant permet de créer une nouvelle fenêtre puis d'afficher son nom et si elle est fermée ainsi de l'imprimer.

```
1 <body>
2     <p>Cliquez sur le bouton pour créer une fenêtre puis afficher le nom de la
  nouvelle fenêtre.</p>
3     <button onclick="myFunction()">Essayer-le</button>
4     <script>
5       function myFunction() {
```

```
6           var myWindow = window.open("", "Fenetre de Message", "width=500,height=250");
7           myWindow.document.write("<p>Le nom de cette fenêtre est : " + myWindow.name +
"</p>"
8             +"<p>Est fermée?st : " + myWindow.closed    + "</p>"
9             );
10           myWindow.print();
11        }
12     </script>
13  </body>
```



4. <document>

L'objet <**document**> est le propriétaire de tous les autres objets de votre page Web. Il représente votre page Web.

De même l'objet <document> dispose des propriétés et des méthodes.

Propriétés de l'objet document :

- **applets** : retourne la collection d'applets java présente dans le document;
- **cookie** : permet de stocker un cookie;
- **domain** : indique le nom de domaine du serveur ayant apporté le document;
- **forms** : retourne la collection de formulaires présents dans le document;
- **images** : retourne la collection d'images présentes dans le document;
- **links** : retourne la collection de liens présents dans le document;
- **referrer** : indique l'adresse de la page précédente;
- **title** : indique le titre du document.

Méthodes de l'objet document :

- **close( )** : ferme le document en écriture;
- **open( )** : ouvre le document en écriture;
- **write( )** : écrit dans le document;
- **writeln( )** : écrit dans le document et effectue un retour à la ligne

Exemple de code qui permet d'écrire sur une ligne le titre du document sur la page en cliquant sur un bouton.

```
<body>
    <p>Cliquez sur le bouton pour écrire le titre du dcoument sur la page.</p>
    <button onclick="myFunction()">Essayer-le</button>
    <script>
      function myFunction() {
        document.writeln("Titre  : " + document.title);
    }
    </script>
</body>
```

1. https://fr.wikipedia.org/wiki/Brendan_Eich
2. https://www.w3schools.com/js/js_string_methods.asp
3. https://www.w3schools.com/js/js_number_methods.asp
4. https://www.w3schools.com/js/js_array_methods.asp
5. https://www.w3schools.com/js/js_date_methods.asp

XVI Les événements

JavaScript est un langage de programmation appelé aussi **langage événementiel**, c-à-d. que le script sera exécuté suite à un événement.

La majorité de ses scripts sont associés à des événements qui se produisent sur le navigateur.

Les événements peuvent être des **actions du navigateur** ou des **actions d'un utilisateur** qui vont pouvoir donner lieu à une **interactivité**.

Voici quelques exemples d'événements HTML: une page Web HTML a fini de se charger, un champ de saisie HTML a été modifié, un bouton HTML a été cliqué, etc. mais l'événement par excellence est le clic de la souris.

Grâce à JavaScript il est possible d'associer des fonctions ou des méthodes à des événements tel que: le passage de la souris au dessus d'une zone, le changement d'une valeur, le clique de la souris, etc.

HTML permet d'ajouter des attributs de gestionnaire d'événements, avec du code JavaScript, aux éléments HTML.

La syntaxe d'un gestionnaire d'événement est la suivante:

```
<element event="action_JavaScript_ou_fonction()">
```

Dans l'exemple suivant, un attribut onclick (avec code) est ajouté à un élément <button>:

```
<body>
  <button onclick="this.innerHTML=Date()">Le temps est ?</button>
</body>
```

Dans cet exemple le code modifie le contenu de son propre élément pour cela nous avons utilisé **this**.innerHTML.

Chaque événement ne peut pas être associé à n'importe quel objet. Par exemple on ne pas appliquer l'événement onChange à un lien hypertexte.

Ci-dessous un résumé des événements qui peuvent être associés aux objets les plus courants:

- **Lien hypertexte** : onClick, onMouseOver, onMouseOut
- **Page du navigateur** : onLoad, onUnload
- **Boutons** : onClick
- **Liste de selection** : onBlur, onFocus, onChange
- **Champs de texte** : onBlur, onFocus, onChange, onSelect, onKeyUp, onKeyDown
- **Images** : onMouseOver, onMouseOut

Voici une liste de quelques événements HTML courants:

- **Click**: se produit lorsque l'utilisateur clique sur un élément associé à l'événement.
- **Load**: se produit lorsque le navigateur de l'utilisateur charge la page en cours.
- **Unload**: se produit lorsque le navigateur de l'utilisateur quitte la page en cours.
- **MouseOver**: c'est lorsque l'utilisateur survole (met le curseur de la souris sur) l'élément associé à l'événement.
- **MouseOut**: c'est lorsque le curseur de la souris quitte l'élément.
- **Focus**: se produit lorsque l'utilisateur donne le focus à un élément, c'est-à-dire que l'élément devient actif.
- **Blur**: c'est lorsque l'élément perd le focus (il était actif et il ne l'est plus, c'est à ce moment que cet événement est déclanché).

- **Change**: se produit lorsque l'utilisateur modifie le contenu d'un champs donné.
- **Select**: se produit lorsque l'utilisateur sélectionne un texte (ou une partie d'un texte) dans un champs de type ‘text' ou ‘textarea'.
- **Submit**: se produit lorsque l'utilisateur clique sur le bouton de la soumission d'un formulaire (bouton ‘submit').
- **KeyDown**: se produit lorsque l'utilisateur enfonce une touche du clavier.
- **KeyUp**: c'est lorsque l'utilisateur relâche une touche du clavier.
- **DblClick**: se produit lorsque l'utilisateur clique deux fois (double clique) sur un élément du navigateur.
- **Resize**: quand l'utilisateur redimensionne le navigateur.

XVII Les cookies

1. Introduction

Les **Cookies**, je ne parle pas des cookies sur l'image ci-dessous, sont des données, stockées dans de petits fichiers texte, sur votre ordinateur.

2. Pourquoi les cookies ?

Lorsqu'un serveur Web a envoyé une page Web à un navigateur, la connexion est interrompue et le serveur oublie tout ce qui concerne l'utilisateur.

Les cookies ont été inventés pour résoudre le problème "**comment se souvenir des informations sur l'utilisateur**".

Lorsqu'un utilisateur visite une page Web, son nom peut être stocké dans un cookie. La prochaine fois que l'utilisateur visite la page, le cookie "se souvient" de son nom.

Les cookies qui vont enregistrer la valeur du nom de l'utilisateur comme:

```
1 username = Ahmed DAOUD
```

Lorsqu'un navigateur demande une page Web à un serveur, des cookies appartenant à la page sont ajoutés à la demande. De cette façon, le serveur obtient les données nécessaires pour «se souvenir» des informations sur les utilisateurs.

Un cookie permet de :

- **Transmettre des valeurs** (contenu de variables) d'une page HTML à une autre.

  Par exemple, créer un site marchand et constituer un "caddie" ou un "panier" pour le client. Caddie qui restera sur son poste et vous permettra d'évaluer la facture finale au bout de la commande. Sans faire appel à quelque serveur que ce soit.

- **Personnaliser les pages** présentées à l'utilisateur en reprenant par exemple son nom en haut de chaque page

Limite des cookies :

On ne peut pas écrire autant de cookies que l'on veut sur le poste de l'utilisateur (client d'une page).

Il y a des limites :

- **Limites en nombre** : Un seul serveur (ou domaine) ne peut pas être autorisé à écrire plus de 20 cookies.
- **Limites en taille** : un cookie ne peut excéder 4 Ko.
- **Limites du poste client** : Un poste client ne peut stocker plus de 300 cookies en tout.

Où sont stockées les cookies

Le répertoire de stockage des cookies dépend largement du navigateur utilisé Chrome, Firfoxe, Edge, etc.

Google Chrome héberge tous les cookies dans un fichier appelé "cookies", qui se trouve dans le chemin suivant:

C:\Users\Admin\AppData\Local\Google\Chrome\User Data\Default

3. Structure d'un cookie

La structure générale d'un cookie est comme suit :

```
Nom=Contenu; expires=expdate; path=Chemin; domain=NomDeDomaine;  secure;
```

Avec :

- **Nom=Contenu;**

  Sont deux variables suivies d'un ";" . Elles représentent l'en-tête du cookie.

  La variable **Nom** contient le nom à donner au cookie.

  La variable **Contenu** contient le contenu du cookie

  Exemple : ma_cookie=«oui:visite»

- **Expires= expdate;**

  Le mot réservé expires suivi du signe "=" (égal). Derrière ce signe, vous mettrez une date d'expiration représentant la date à laquelle le cookie sera supprimé du disque dur du client.

  La date d'expiration doit être au format : Wdy, DD-Mon-YYYY HH:MM:SS GMT

- **path=Chemin;**

  Path représente le chemin de la page qui a créé le cookie.

- **domain=NomDeDomaine;**

  domain représente le nom du domaine de cette même page

- **secure**

  secure prend les valeurs "true" ou "false"

  Il permet de spécifier que le cookie sera envoyé uniquement si la connexion est sécurisée selon que le cookie doit utiliser des protocoles HTTP simples (non sécurisés) ou HTTPS (sécurisés).

Notez bien que les arguments path, domain et secure sont facultatifs. lorsque ces arguments sont omis, les valeurs par défaut sont prises. Pour secure, la valeur est "False" par défaut

4. CRUD un cookie

CRUD signifie **C**reate (Creer) , **R**ead (Lire) , **U**pdate (Modifier) , **D**elete (Supprimer)

JavaScript peut créer, lire , modifier et supprimer des cookies avec la propriété **document.cookie**.

- **Créer un cookie**

La syntaxe pour créer un cookie en JavaScript est comme ceci :

```
document.cookie = "username=Ahmed DAOUD";
```

Vous pouvez également ajouter une date d'expiration (en heure UTC). Par défaut, le cookie est supprimé lorsque le navigateur est fermé :

```
document.cookie = "username=Ahmed DAOUD; expires=Thu, 31 Dec 2020 12:00:00 UTC";
```

Avec un paramètre de chemin, vous pouvez indiquer au navigateur à quel chemin appartient le cookie. Par défaut, le cookie appartient à la page en cours.

```
document.cookie = "username=Ahmed DAOUD; expires=Thu, 31 Dec 2020 12:00:00 UTC;
path=/";
```

- **Lire Cookie**

En JavaScript, les cookies peuvent être lus comme ceci:

```
var x = document.cookie;
```

document.cookie renverra tous les cookies dans une seule chaîne, un peu comme : cookie1 = valeur; cookie2 = valeur; cookie3 = valeur;

Modifier un cookie

Avec JavaScript, vous pouvez modifier un cookie de la même manière que vous le créez:

```
document.cookie = "username=Mohammed ALI; expires=Thu, 31 Dec 2022 12:00:00 UTC;
path=/";
```

L'ancien cookie est écrasé.

- **Supprimer un cookie**

La suppression d'un cookie est très simple.

Vous n'êtes pas obligé de spécifier une valeur de cookie lorsque vous le supprimez.

Définissez simplement le paramètre expires sur une date inférieure à celle du moment où on l'écrit sur le disque.

```
document.cookie = "username=Mohammed ALI; expires=Thu, 31 Dec 2000 12:00:00 UTC;
path=/";
```