**Annick Lacroix**

2018-2019

**JAVASCRIPT**

**Notes de Cours Afpa**

**62014-06-08**

## **Comment indiquer les commentaires en javascript :**

// commentaire d’une ligne

/\* commentaire  
multi lignes \*/

**Du JavaScript dans un fichier séparé**On peut écrire notre code JavaScript dans un fichier séparé portant l’extension « .js ».

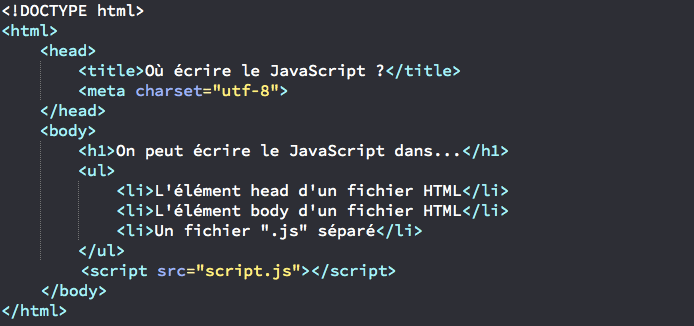
<script src="index.js"></script>

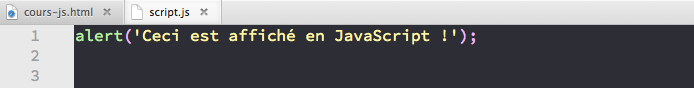
C’est très souvent la méthode recommandée dans le cas de gros projets car elle permet une meilleure maintenabilité du code grâce à la séparation des langages, et car on va pouvoir réutiliser un même code JavaScript dans plusieurs fichiers HTML.

Dans ce cas, nous allons devoir lier nos fichiers HTML et JavaScript en utilisant à nouveau un élément script et son attribut src.

En valeur de l’attribut src, nous allons indiquer le chemin relatif du fichier .js par rapport au fichier .html. Si nos deux fichiers sont dans le même dossier, par exemple, il suffira d’indiquer le nom du fichier JavaScript.

Par exemple, si l'on souhaite lier un fichier JavaScript script.js à un fichier HTML placé dans le même dossier nous écrirons :







## **Comment intégrer du code JavaScript dans une page ?**

Il y a deux manières d'ajouter du code JavaScript dans une page :

1. en liant depuis la page HTML un fichier externe, dans lequel sont placées les instructions JavaScript :

<script type="text/javascript" src="monscript.js"></script>

1. en ajoutant le code JavaScript à l'intérieur de l'élément script :
2. <script type="text/javascript">
3. // Mon code Javascript
4. ...
5. </script>

En ce qui concerne la deuxième possibilité, vous pouvez également lire [« Comment gérer les caractères spéciaux dans les éléments script et style ? »](https://www.alsacreations.com/astuce/lire/82-comment-gerer-caracteres-speciaux-elements-script-style.html)

Traditionnellement il était d'usage de placer la balise <script> entre les tags <head> et </head>, cependant il est actuellement recommandé de **la placer en fin de document juste avant </body>** pour ne pas bloquer le chargement de la page, et exécuter les scripts uniquement lorsque le DOM est prêt.

https://www.alsacreations.com/xmedia/tuto/deco/warning.gif Attention, **la balise "script" n'est pas auto-fermante**, c'est-à-dire que l'on ne peut pas écrire <script type="text/javascript" ... />

L'attribut type doit être renseigné avec la syntaxe XHTML, mais il peut être omis en HTML5.

Hello world :

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>text JS</title>

</head>

<body>

<script>

document.write("hello world.")

</script>

</body>

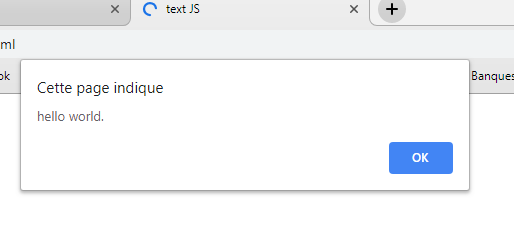
</html>

Pour créer une alerte :

<script>

alert("hello world.");

</script>



Permet de changer le contenu (ou mettre à jour suivant la personne qui se connecte) :

<body>

<p id="test">

ici le body

</p>

<script>

alert("pause ici");

document.getElementById("test").innerHTML="nouveau contenu";

</script>

</body>

</html>

## **VARIABLES :**

Pour identifier une variable on lui donne un nom unique (qui ne doit pas être attribué à une autre variable dans le même programme). Le nom de la variable peut être composé de lettres (minuscules, majuscules ou les deux), de chiffres (de 0 à 9) et les caractères **\_** et **&**. En plus, il doit impérativement commencer par une lettre ou le caractère **\_**.

Les noms ne doivent pas commencer :  
-ni par un chiffre , ni par un symbole, ni par un espace.

## **Opérateurs arithmétiques**

Ce sont des opérateurs qui effectuent des calculs mathématiques classiques à savoir: l'addition (**+**), la soustraction (**-**), la multiplication (**\***), la division (**/**) et le modulo (qui signifie le reste de la division)(**%**).

Dans le programme on peut écrire une variable des 2 façons suivantes :  
a=3  
var a=3

<body>

<p id="test">

ici le body

</p>

<script>

a=5;

b=7;

c=a+b;

document.getElementById("test").innerHTML=c;

</script>

</body>

Le résultat s’affiche sur la page html (12).

Pour montrer l’opération entière :

document.getElementById("test").innerHTML="5+7 = "+(a+b);

\*et / sont prioritaires sur + et -.  
il faut mettre l’explication avant de mettre l’opération.

## **Opérateurs d'incrémentation**

Ce sont des opérateurs qui permettent de modifier la valeurs d'une variable en l'augmentant ou la diminuant de 1. Deux opérateurs sont utilisés: incrémentation (**++**) et décrémentation (**--**).

a=10;   
a++; // a vaut donc 11   
a--; // a vaut à nouveau 10

l'**incrémentation** est l'opération qui consiste à ajouter 1 à un compteur  
a incrémentation de1 a=a+1 ; → a++ ;  
a++ ↔ a=a+1 incrément de 1

L'opération inverse, la **décrémentation**, consiste à retirer 1 au compteur  
a decrémenté de 1  
a=-1 → a - -  
a - -↔ a=a-1 décrément de 1

Exemple :  
a=1  
b=5+(a++) ; ça donne 6 et a=2  
c=5+(++a) ; ça donne 8 et a=3   
c’est une suite donc à chaque fois a prend 1 en plus.

a=5  
c= (a++) + (a++)  
 \ \  
 5 + 6  
 \ \  
 c= a=6 + a=7   
c=11 et a=7

## **Opérateurs d'assignement**

Ce sont les opérateurs qui permettent d'affecter une valeur à une variable directement ou à travers une opération (comme l'incrémentation ou la décrémentation). Les opérateurs disponibles pour cette famille sont: l'affectation directe (**=**), l'affectation avec addition (**+=**), l’affectation avec soustraction (**-=**), l'affectation avec multiplication (**\*=**), l'affectation avec division (**/=**) et l'affectation avec modulo (**%=**).   
  
Exemple:

a=10;   
b=a; // b vaut 10   
a+=5; // a vaut 15   
a-=3; // a devient 12

On peut écrire l'instruction **a++** comme ceci: **a+=1** ou encore: **a=a+1**.

## **Changement de type d'une variable (CAST)**

* **Les nombres**: Ce sont les nombres avec ou sans virgule, positifs ou négatifs.
* **Les chaines de caractères**: (ou **string**), ce sont des suites de caractères quelconques.
* **Les booléens**: Ce sont les nombres booléens qui peuvent avoir deux valeurs, soit **1** ou **0** exprimés de préférence par **vrai (true)** ou **faux (false).**
* **Les variables vides**: Ce sont des variables qui ne contiennent aucune valeur, ils ont la valeur implicite **null** (absence de valeur), **indefined** (on ne connait pas la valeur).
* **Objets**: **collection** ou H (« first\_name » : « John », « last\_name » : « Lennon »)  
   clef ↗ valeur ↗
* **Tableau**: [« John », « Paul », « ringo », « george »]   
   [ tableau ]

## **Opérateurs de comparaison**

Ce sont les opérateurs qui testent une condition si elle est vraie ou fausse. Les opérateurs disponibles sont: égal (**==**), strictement inférieur (**<**), inférieur ou égal (**<=**), strictement supérieur (**>**), supérieur ou égal (**>=**), différent (**!=**), identique (**===**), non identique (**!==**).

> sup >inf   
== égal === identique !== non identique  
>= sup ou égal <= inf ou égal  
 != différent (! → veut dire non)

## **Opérateurs logiques**

Ce sont des opérateurs qui regroupent logiquement plusieurs conditions. Les trois opérateurs utilisés sont: **ET** logique (**&&**), **OU** logique (**||**) et NON logique (**!**).

c=2  
c !=2 faux (car c est égal à 2)  
c !=3 vrai (car c est différent de 3)

=== égal niveau valeur (nombre) ET du même type (caractère ou chiffre):

C=2 2 est un entier  
c ==’2’ vrai 2 est un caractère car entre ‘ ‘  
c === ‘2’ faux compare la valeur et le type

0 5 10 15 20

x

x > 5 ET x < 10 → (x>5) && (x<10)

T F T F

A B A&&B  
V V V  
F F F  
V F F  
F V F

x > 5 OU x < 10 → (x>5) ll (x<10)

A B AllB  
V V V  
V F V  
F F V  
F V F

## **Conditions : { }**

Si (condition) alors →if   
faire un traitement

If (x>5) {   
a=3 ;  
/\*la suite du traitement\*/  
}

Si (condition) alors → if  
faire un traitement   
sinon → else  
faire un autre traitement

If (x>10) {  
 msg= « moyenne » ;  
}  
else {  
msg= « pas la moyenne » ;  
}

## **Opération ternaire (3 opérants dans la même opération pour les affectations)**

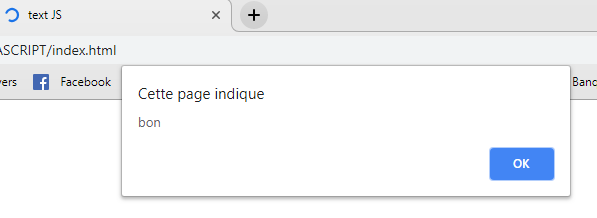
**L'opérateur ternaire** est une écriture abrégée de la structure **if else**. Il est souvent utilisé si on souhaite retourner un résultat en fonction de l'état d'une condition.

If (condition) Ou X=(condition) ? val1 : val2 ;  
msg= val1 msg= (x>10) ? « bon » : « moyen »  
else msg= « bon » ;  
msg=val2 x<10 : msg= « moyen » ;

Ce qui signifie: si la **condition** est vraie (le point d'interrogation fait allusion à une question), alors le premier bloc délimité par les parenthèses (**traitement 1**) est exécuté, sinon (représentée par **:**) alors le deuxième bloc est exécuté (dans ce cas **traitement 2**).

Msg= « bonjour\_\_» +((S==M) ?Mr :Mme) ; → bonjour Mr ou bonjour Mme

**Opérateurs de comparaison : La condition if … else if… else**

<body>

<p id="Message"></p>

<script>

var note=17;

if (note<0){

alert("valeur négative")

}

else if (note<=5){

alert("mediocre");

}

else if (note<=10){

alert("passable");

}

else if (note<=15) {

alert("bon");

}

else if (note>15){

alert("très bon");

}

else {

alert("valeur supérieure à 20")

}

//affichage d'un message

</script>

</body>

Avec les notes suivantes, on obtient les messages suivants :

-10→ valeur négative ; 3 → médiocre ; 7→ passable ; 13 → bon ; 17 → très bon ; 25→ valeur supérieure à 20.

## **Switch (comparaison)**

L'instruction **switch** évalue une expression et, selon le résultat obtenu et le cas associé, exécute les instructions correspondantes.

<script>

var mynote="A";

switch (mynote){

case "A":

document.write("Excellent travail <br>");

break;

case "B":

document.write("Bon travail <br>");

break;

case "C":

document.write("Passable <br>");

break;

default:

document.write("note inconnue<br>");

break;

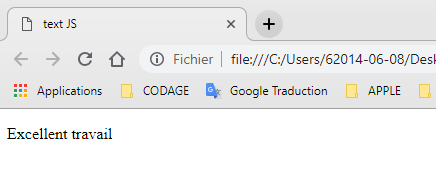
}

</script>

**Document.write** peut être abrégé par **dw**

*Break* permet de sauter à l’extérieur de la boucle.

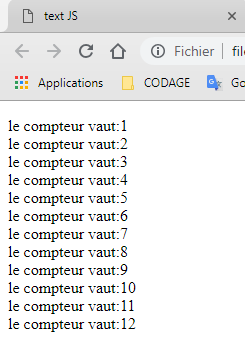
Ce qui donne dans le navigateur : (suivant la valeur donnée en variable mynote)



## **BOUCLES :** Les boucles permettent de répéter une opération plusieurs fois

## **While (condition)**

L'instruction **while** permet de créer une boucle qui s'exécute tant qu'une condition de test est vérifiée. La condition est évaluée avant d'exécuter l'instruction contenue dans la boucle.

While (condition){  
// bloc traitement  
// incrémentation  
//ou décrémentation  
//ou changement d’état  
}

Ex :  
var echec=0 ;  
while (echec>5){  
//jouer partie  
echec ++ ;  
}

Ex2 :

Var connexion=(no) ;  
while (connexion == « no »){  
//essayer de se connecter  
//au serveur  
connexion au serveur () ;  
}

script>

var compteur=1;

while (compteur <=12){

document.write("le compteur vaut:"+ compteur + " <br>");

compteur ++;

}

</script>

## **Do While (condition)**

L'instruction **while** permet de créer une boucle qui s'exécute tant qu'une condition de test est vérifiée. La condition est évaluée avant d'exécuter l'instruction contenue dans la boucle.

do {  
// bloc traitement  
// incrémentation  
//ou décrémentation  
//ou changement d’état  
}while (condition)

Ex :  
var vie=-1 ;  
do{  
//jouer partie jusqu’à  
la mort du personnage  
vie - - ;  
}while (vie>0) ;

<script>

var compteur=1;

do {

document.write("le compteur vaut:"+ compteur + " <br>");

compteur ++;

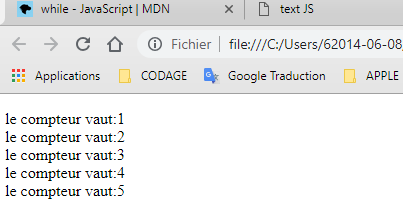
}

while (compteur<=5);

</script>

Il ajoute 1 à chaque traitement jusque 5. Ensuite il passe aux autres lignes après while.

Ce qui donne :



## **For (condition)**

L'instruction **For** crée une boucle composée de trois expressions optionnelles séparées par des points-virgules et encadrées entre des parenthèses qui sont suivies par une instruction (généralement [une instruction de bloc](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/bloc)) à exécuter dans la boucle

For (initialisation ; condition d’arrêt ; incrémentation){  
//bloc de traitement  
}

Autre exemple :

For (var i=0 ; i<students.lenght ; i +=1) {

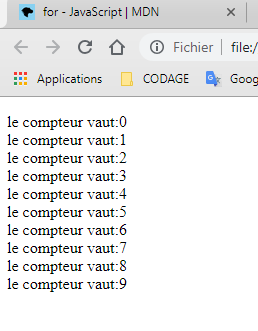
↗ ↗ ↗

Condition Condition condition

D’initiation d’arrêt finale

Console.log(students[i] ; // code exécuté

}

Ex :

Var i ;  
for (i=1 ; i≤5 ; i ++){  
//écrire valeur de « i » ;  
}

<script>

var compteur;

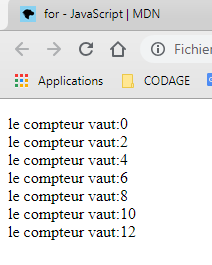
for (compteur = 0; compteur <10; compteur ++) {

document.write("le compteur vaut:"+ compteur );

document.write("<br>");

}

</script>



En augmentant le compteur de 2 en 2 : (compteur +-2)

<script>

var compteur;

for (compteur = 0; compteur <13; compteur +=2) {

document.write("le compteur vaut: "+ compteur );

document.write("<br>");

}

</script>

## **For if continue**

<script>

var compteur;

for (compteur = 0; compteur <13; compteur +=1) {

if (compteur ==5 || compteur ==6){

continue;

}

document.write("le compteur vaut: "+ compteur );

document.write("<br>");

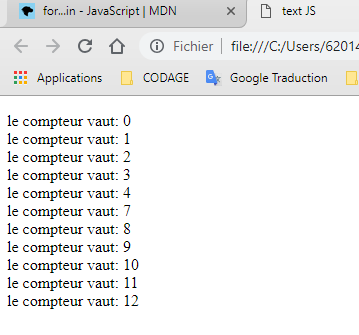
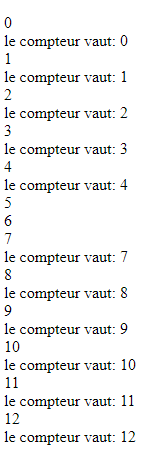
}

</script>

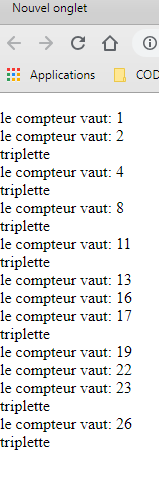
S’il trouve que le compteur vaut 5 ou 6 donc que la condition est respectée  
 if (compteur ==5 || compteur ==6)   
alors il continue la boucle et revient à « for ».  
Ensuite il reprend la boucle jusqu’au bout

Le « continue » implique que la boucle repart au début en ignorant ce qu’il y a après.

Ce qui donne :

Faire un script :

-De 1 à 36  
-si compteur multiple de 5 ou 7 ne rien imprimer (*multiple* → % 5 == 0 appelé *modulo*)  
-si compteur multiple de 3, écrire « triplette »  
-si compteur supérieur à 27, arrêter

<script>

var compteur;

for (compteur = 1; compteur < 36; compteur++) {

if (compteur % 5 == 0 || compteur % 7 == 0) {

continue;

}

if (compteur % 3 == 0) {

document.write("triplette <br>");

}

else {

document.write("le compteur vaut: " + compteur);

document.write("<br>");

}

if (compteur == 27)

break;

}

</script>

**Fonction**

Une **définition de fonction** (aussi appelée **déclaration de fonction** ou **instruction de fonction**) est construite avec le mot-clé [**function**](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/function), suivi par :

* Le nom de la fonction.
* Une liste d'arguments à passer à la fonction, entre parenthèses et séparés par des virgules.
* Les instructions JavaScript définissant la fonction, entre accolades, { }.

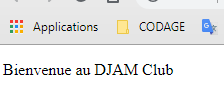
Ex :

Function nom.fonction ( ) {  
//…  
//bloc de traitement  
//  
}  
nom.fonction ( ) ;

<script>

function mafonction(){

document.write("Bienvenue au DJAM Club");

 }

</script>

<script>

mafonction();

</script>

</body>

Autre exemple :

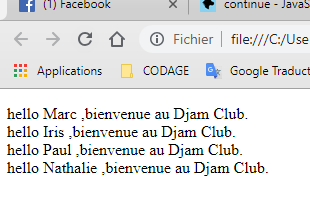
<script>

function mafonction(ttt){

document.write("hello "+ ttt + " ,bienvenue au Djam Club.<br>");

}

</script>



<script>

mafonction("Marc");

mafonction("Iris");

mafonction("Paul");

mafonction("Nathalie");

</script>

Avec plusieurs paramètres :

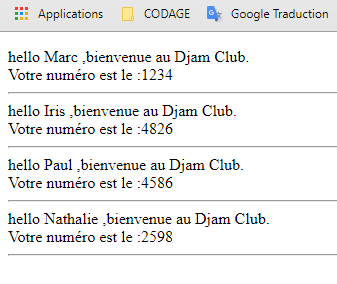
<script>

function mafonction(param1, param2){

document.write("hello "+ param1 + " ,bienvenue au Djam Club.<br>");

document.write("Votre numéro est le :" + param2+ "<br><hr>");

}

 </script>

<script>

mafonction("Marc",1234);

mafonction("Iris",4826);

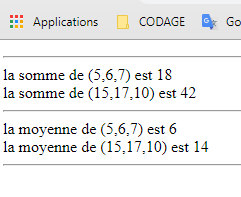
mafonction("Paul",4586);

mafonction("Nathalie",2598);

</script>

**Return**

L'instruction **return**met fin à l'exécution d'une fonction et définit une valeur à renvoyer à la fonction appelante

 <script>

function somme(x, y, z){

var som;

som = x + y + z;

return (som)

}

function moyenne(x, y, z){

return ((x + y + z)/3)

}

</script>

<hr>

<script>

var m1;

m1 = somme (5,6,7);

document.write ("la somme de (5,6,7) est " + m1 + "<br>");

m1 = somme (15,17,10);

document.write ("la somme de (15,17,10) est " + m1 + "<br>");

</script>

<hr>

<script>

var moy1;

moy1 = moyenne (5,6,7);

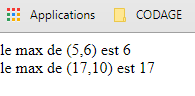
document.write ("la moyenne de (5,6,7) est " + moy1 + "<br>");

moy1 = moyenne (15,17,10);

document.write ("la moyenne de (15,17,10) est " + moy1 + "<br>");

</script>

<hr>

<head>

<script>

function max (x, y){

return ((x>y ) ? x : y);

}

</script>

</head>

<body>

<script>

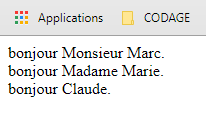
m1=max(5,6);

document.write("le max de (5,6) est " + m1 + "<br>");

m1=max(17,10);

document.write("le max de (17,10) est " + m1 + "<br>");

</script>

<head>

<script>

function salut(s){

if (s=="m" || s=="M"){

return "Monsieur";

}

else if (s=="f" || s== "F")

return "Madame";

else

return "";

}

</script>

</head>

<body>

bonjour

<script>document.write(salut("m"))</script> Marc.<br>

bonjour

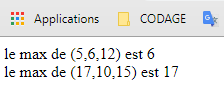
<script>document.write(salut("f"))</script> Marie.<br>

bonjour

<script>document.write(salut("x"))</script> Claude.<br>

</body>

La valeur maximale entre 3 valeurs :

<head>

<script>

function max (x, y){

return ((x>y ) ? x : y);

}

function max3(x,y,z){

return max(max(x,y),z);

}

</script>

</head>

<body>

<script>

m3=max(5,6,12);

document.write("le max de (5,6,12) est " + m3 + "<br>");

m3=max(17,10,15);

document.write("le max de (17,10,15) est " + m3 + "<br>");

</script>

//======================================

// Fonctions Utiles

//======================================

function dw(str) {

document.write(str);

}

function dwn(str) {

document.write(str+"<br>");

}

function getEBI(id) {

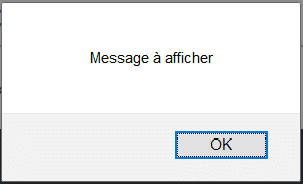
return document.getElementById(id);

}

**Boîte de dialogue en JavaScript : alert()**

La boîte de dialogue qui permet d’afficher un texte et un bouton « OK ».

alert("Message à afficher");

[](https://wprock.fr/wp-content/uploads/2018/05/wprock-boites-dialogue-javascrip-js-alert.png)

**Boîte de dialogue en JavaScript : confirm()**

La boîte de dialogue confirm() permet d’afficher un message et deux boutons : un bouton « OK » et un bouton « Annuler ». Cette fonction retourne une valeur booléenne qui vaut « true » si c’est le bouton OK qui est cliqué et retourne « false » si c’est le bouton « Annuler » qui est cliqué.

if ( confirm( "Message à afficher" ) ) {

// Code à éxécuter si l'utilisateur clique sur "OK"

} else {

// Code à éxécuter si l'utilisateur clique sur "Annuler"

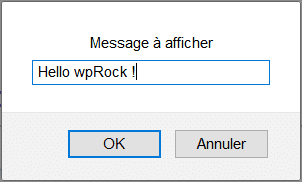
}

[](https://wprock.fr/wp-content/uploads/2018/05/wprock-boites-dialogue-javascrip-js-confirm.png)

**Boîte de dialogue en JavaScript : prompt()**

La boîte de dialogue prompt() permet d’afficher un message, un champ à remplir (imput de type texte) et un bouton « OK ». Cette fonction retourne la valeur qui a été entré dans le champ par l’utilisateur.

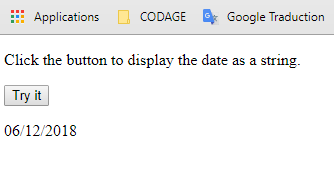
var valeur = prompt("Message à afficher");

[](https://wprock.fr/wp-content/uploads/2018/05/wprock-boites-dialogue-javascrip-js-prompt.png)

Exemple :

<body>

<p>Click the button to display the date as a string.</p>

 <button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>

function myFunction() {

var d = new Date();

var n = d.toLocaleDateString();

document.getElementById("demo").innerHTML = n;

}

</script>

</body>

**Création de tableaux :**

***Primitives*** *(exactement l’adresse de la donnée)*

Var P1= « Marc » ; → P1 Marc

Var P2=P1 ; → P1 Marc

*Primitive avec clonage de données*

P2= « Martine » → changement de la données P2

***Eléments*** *(référence vers une adresse différente de la même donnée)*

Var T1=[1,2,3] →T1= 0728

Var T2=T1 →T2= 0728

Var T3=T2

Vous devez utiliser des **objets** lorsque vous souhaitez que les noms d'éléments soient des **chaînes (texte)** .

Vous devez utiliser des **tableaux** lorsque vous souhaitez que les noms d'éléments soient des **nombres** .

Les string sont un cas particulier d’objets qui sont considérés comme une enveloppe pour les données du type chaine de caractères. Les littéraux de chaine de caractères ne doivent pas être confondus avec les objets String. Par exemple, le code suivant crée deux choses : un littéral de chaîne de caractère, s1, et l'objet String s2 :

var s1 = "truc"; //crée un littéral de chaîne de caractères

var s2 = new String("truc"); //crée un objet String

***La propriété length :***

En termes d'implémentation, les éléments d'un tableau sont en fait stockés comme des propriétés de l'objet et l'indice de l'élément est le nom de la propriété. La propriété ***length*** est spéciale : elle renvoie toujours l'indice du dernier élément plus 1. Attention : les indices d'un tableau, en JavaScript, commencent à 0 et pas à 1.

<script>

var personnes = ["Marc", "Paul", "Véronique", 25,[1, 2, 3]];

var personnes2 = new Array("Marc", "Paul", "Véronique", 25,[1, 2, 3]);

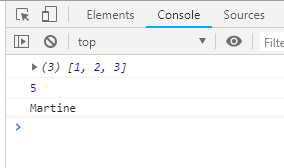
console.log(personnes[4])

console.log(personnes.length);// la propriété length renvoi la longueur d'une chaine (nombre de caractères)

personnes[1]= "Martine";

console.log(personnes[1])

</script>



Pour voir ce que ça donne dans le navigateur

(clic droit + inspecter)

***Pour placer les différents éléments :***

Personnes.splice(␣,␣,␣) ;

①,②,③

① → position départ

② → combien d’éléments à supprimer

③ → à ajouter à l’emplacement

***La propriété reverse***

La fonction**sort**trie les éléments de tableau par ordre croissant ou alphabétique croissant.  
Méthode Array.sort()

<script>

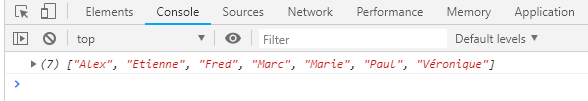
var personnes = ["Marc", "Paul", "Véronique", "Etienne", "Alex", "Marie", "Fred"];

personnes.sort(); // trie les éléments du tableau en ordre croissant

console.log(personnes);

personnes.reverse(); // replace dans l'ordre alphabétique

</script>



***La fonction forEach***( permet d'exécuter une fonction donnée sur chaque élément du tableau.).

<script>

var personnes = ["Marc", "Paul", "Véronique", "Etienne", "Alex", "Marie", "Fred"];

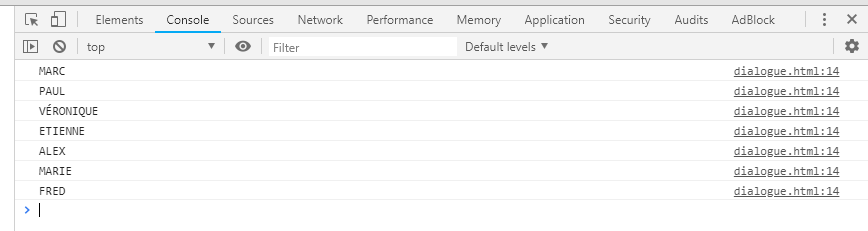
personnes.forEach(func01);

function func01(param){

console.log(param.toUpperCase());//lister en majuscules dans console

}

</script>



**Console.log** sert à voir ce qu’est la variable :

Var firstName= « John » ;

Console.log(firstName) ;  
Peut être appelé autant de fois que l’on veut.

Avec des caractères (string)

Var firstName= « John » ;

Var lastName= « Lennon » ;

//Concaténation

Var fullName= firstName + «  » + lastName ;

Console.log(fullName) ;

**ou**

Var firstName= « John » ;

Var lastName= « Lennon » ;

Console.log( firstName + «  » + lastName) ;

*Avec des chiffres :*

Var age = 30 ;

Console.log (« J’ai  » + 30 + « ans ») ;

//une année passe

//Réaffectation de la variable age

Age = age +1

Console.log (« J’ai  » + age + « ans ») ;

***Array compare → fcompare***

Comparaison de tableaux :

<script>

var res;

var tab1 = [19, 72, 33, 5, 66, 2, 14, 9, 10];

var tab2 = [19, 72, 33, 15, 16, 8, 17, 39, 50];

var tab3 = [19, 72, 33, 15, 16, 8, 39, 50];

var tab4 = [19, 72, 33, 5, 66, 2, 14, 9, 10];

res = fcompare(tab1, tab2); // compare les tab1 et tab2

document.write("resultat= " + res + "<br>"); // écrit le résultat de la comparaisipn

res = fcompare(tab1, tab3);

document.write("resultat= " + res + "<br>");

res = fcompare(tab1, tab4);

document.write("resultat= " + res + "<br>");

//Fonctions

// ////////////////////

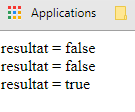
function fcompare(param1, param2) { // != différent de

if (param1.length != param2.length) { // si la longeur des tableaux est différente return false

return false;

}

for (var i = 0; i < param1.length; i++) {

 if (param1[i] != param2[i]) {

return false;

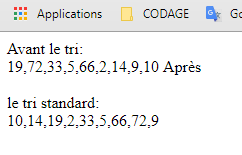
}

}

return true;

}

</script>

 ***sort :*** (classe dans l’ordre croissant)

<script>

var tab1 = [19, 72, 33, 5, 66, 2, 14, 9, 10];

document.write("Avant le tri : <br>" + tab1);

document.write("<br><br>");

document.write("Apres le tri standard : <br>"+tab1.sort());

document.write("<br><br>");

</script>

***sort avec fonction :***

<script>

var str;

var tab1 = [19, 72, 33, 5, 66, 2, 14, 9, 10];

str = tab1.join(" \_ ");

document.write("Avant le tri : <br>" + str);

document.write("<br><br>");

var tab2 = tab1.sort();

str = tab2.join(" \_ ");

document.write("Apres le tri standard : <br>" + str);

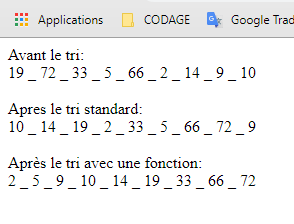
document.write("<br><br>");

var tab3 = tab1.sort(tri\_numerique);

str = tab3.join(" \_ ");

document.write("Apres le tri avec une fonction : <br>" + str);

document.write("<br><br>");

 function tri\_numerique(a, b) {

if (a < b) {

return -1;

} else if (a > b) {

return 1

} else {

return 0;

}

}

</script>

***Sort 2D avec fonction***

<script>

var str;

var tab1 = [[19,1], [72,4], [33,88], [5,25],[33,88]];

str = tab1.join(" \_ ");

document.write("Avant le tri : <br>" + str);

document.write("<br><br>");

var tab2 = tab1.sort();

str = tab2.join(" \_ ");

document.write("Apres le tri standard : <br>"+str);

document.write("<br><br>");

var tab3 = tab1.sort(tri\_numerique);

str = tab3.join(" \_ ");

document.write("Apres le tri avec une fonction : <br>" + str);

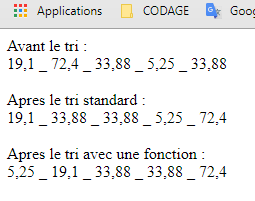
document.write("<br><br>");

function tri\_numerique(a, b) {

if (a[0] < b[0]) {

return -1;

}

 else if (a[0] > b[0]) {

return 1

}

else {

if (a[1] < b[1]) {

return -1;

}

else if (a[1] > b[1]){

return 1

}

else {

return 0;

}

}

}

</script>

***Création d’objets en JavaScript***

Outre les objets natifs, nous pouvons également créer nos propres objets en JavaScript.

Nous disposons de trois façons pour créer de nouveaux objets :

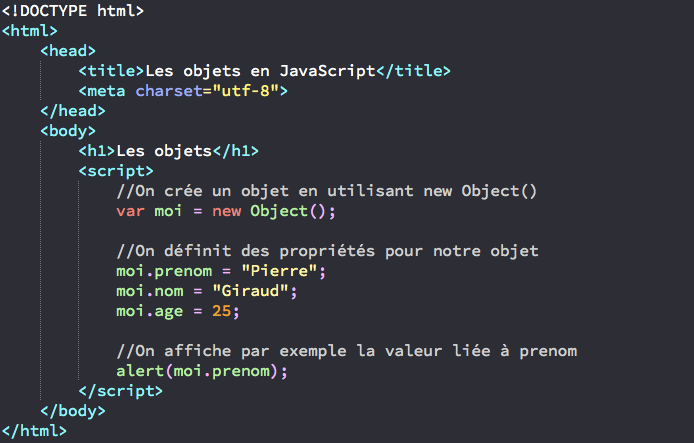
* On peut utiliser le mot clef **new** et créer un objet à partir de **Object()** ;
* On peut créer un objet littéral ;
* On peut définir un constructeur puis créer des objets à partir de celui-ci.

***Créer des objets en utilisant new Object()***

Nous avons déjà vu comment créer un nouvel objet de type string dans la partie précédente en utilisant le mot clef new.

Nous allons faire exactement pareil ici en utilisant cette fois-ci le **constructeur Object()** et en définissant par exemple plusieurs propriétés pour notre objet.

Notez qu’il est aujourd’hui généralement déconseillé de créer un objet de cette façon pour des raisons de performance du code. Cette méthode ne vous est donc présentée qu’à titre d’exemple.



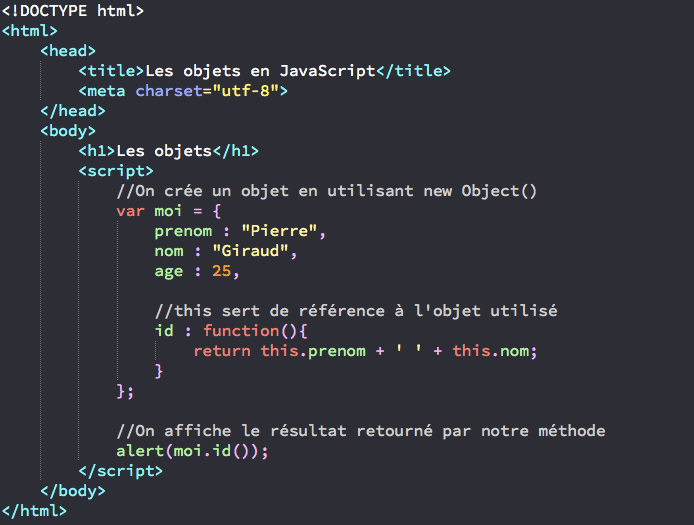


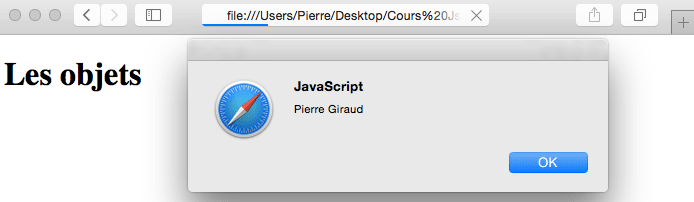
Comme vous pouvez le remarquer, pour ajouter de nouvelles propriétés (ou méthodes) à notre objet, il faut une nouvelle fois utiliser le point et ensuite définir le nom de la propriété et lui attribuer une valeur avec le signe égal.

***Créer un objet littéral***

Créer un objet littéral est la manière généralement recommandée de créer un objet, et c’est également la manière la plus simple de procéder.

Pour créer un objet littéral, nous n’avons qu’à déclarer une nouvelle variable et à utiliser une paire d’accolades en définissant les propriétés et méthodes de notre objet à l’intérieur de celles-ci.





Ici, on crée un objet littéral moi possédant trois propriétés et une méthode. On utilise cette fois-ci les deux points pour assigner les valeurs à nos marqueurs.

Notez l’utilisation du mot clef **this** qui fait ici référence à l’objet couramment utilisé.

Cela signifie que **this.prenom** va aller rechercher la valeur liée au prenom de l’objet moi, c’est-à-dire Pierre.

This est un mot clef récurrent dans les langages orientés objets et il est très important de bien en comprendre les différents sens selon les utilisations qui sont faites de celui-ci.

Notez également qu’il nous faut ajouter une paire de parenthèses afin de bien accéder au résultat retourné par notre méthode id().

Sans l’utilisation de ces parenthèses, cela retournerait la valeur de la propriété id, c’est-à-dire tout le script de notre fonction.

***Créer des objets à partir d’un constructeur***

La dernière méthode pour créer des objets en JavaScript est de créer un constructeur puis ensuite de créer des objets à partir de celui-ci.

Notre constructeur va lui-même contenir des méthodes et des propriétés et les objets que nous allons créer à partir de celui-ci vont en hériter.

Nous utiliserons cette façon de faire pour des gros projets pour lesquels nous aurons à créer beaucoup d’objets similaires.

Parmi les constructeurs natifs du JavaScript, nous avons déjà vu le constructeur String() à partir duquel nous avons créé des objets de type string.

Nous avons ensuite pu observer que l’on pouvait toujours accéder aux propriétés et aux méthodes définies dans le constructeur String() à travers nos objets de type string et même à travers les valeurs primitives.

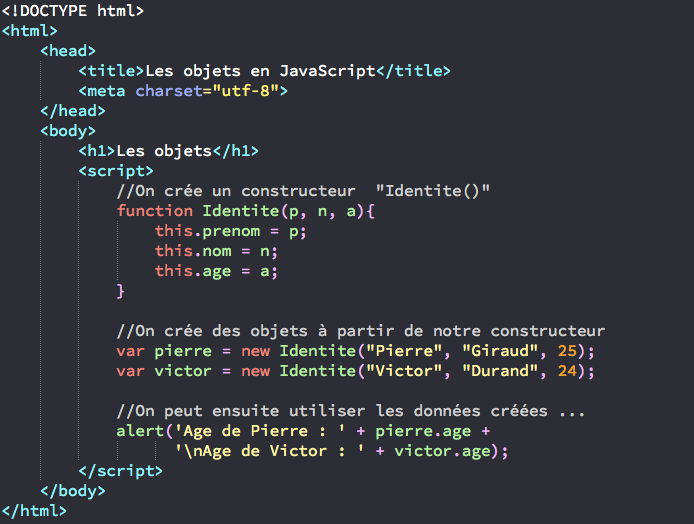
Voici ci-dessous une liste des constructeurs natifs les plus connus et utilisés en JavaScript :

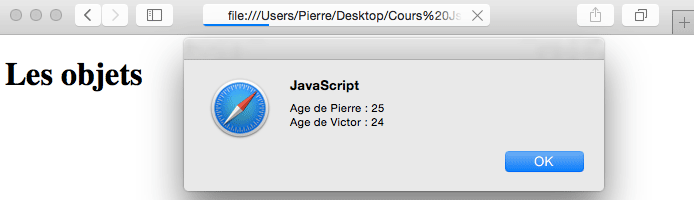
* Object() ;
* String() ;
* Number() ;
* Boolean() ;
* Array() ;
* RegExp() ;
* Function() ;
* Date() ;

Essayons de créer un nouveau constructeur ensemble. Tout d’abord, vous devez savoir qu’un constructeur est avant tout une fonction et doit donc être déclaré avec le mot clef function.

Ensuite, dans cette fonction constructeur, nous allons définir des propriétés et des méthodes.

*Plusieurs constructeurs peuvent amener au même objet.*





Ici, nous avons créé un constructeur très simple. Dans ce cas-là, le mot clef **this** va une nouvelle fois faire référence à l’objet sur lequel on travaille actuellement.

Par exemple, lorsque l’on crée l’objet pierre à partir de notre constructeur, pierre va prendre la place de **this.**

Ainsi, pour ensuite récupérer le prénom de l’objet pierre, nous n’avons qu’à y accéder via pierre.prenom.

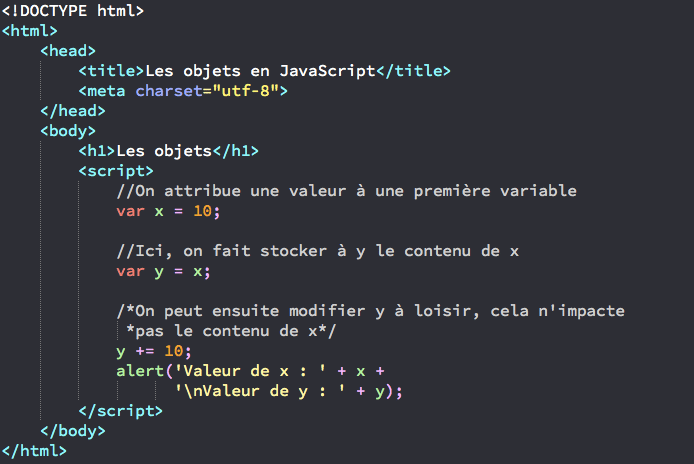
Notez que lorsqu’on crée un constructeur, nous sommes ensuite obligés d’utiliser le mot clef new pour créer de nouveaux objets à partir de celui-ci et de procéder comme je l’ai fait ci-dessus.

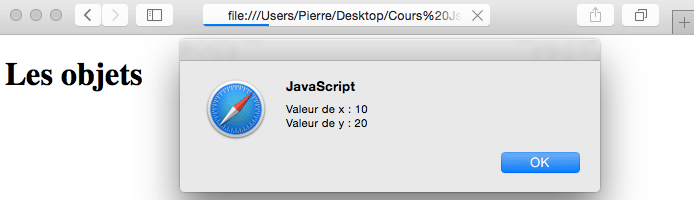
***L’identité des objets***

Faites bien attention à un point avec les objets : au contraire des valeurs primitives, on y accède par référence et non pas par valeur.

Ainsi, on ne peut pas ni « comparer » ni « copier » un objet car cela n’a tout simplement pas de sens.

Prenons un exemple concret afin de bien comprendre. Pour cela, nous allons comparer le comportement des valeurs primitives et des objets.

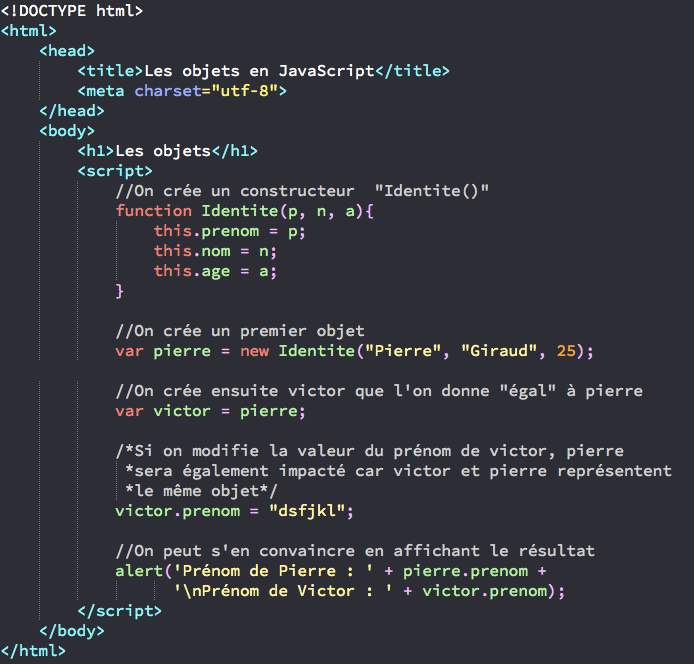




Lorsque l’on travaille avec des variables qui contiennent des valeurs primitives, on peut tout à fait créer des « copies » de ces variables puis ensuite modifier les copies sans que cela impacte la valeur contenue dans la variable originale.

Avec les objets, cela va être complètement différent puisqu’on ne va pas pouvoir créer directement de « copie » d’un objet. En fait, on ne va créer qu’une autre référence pointant vers le même objet.

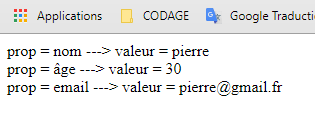
Ainsi, si l’on modifie par la suite l’une des valeurs d’une des références, on modifie également celle des autres. Regardez plutôt l’exemple suivant pour vous en convaincre :





Faîtes bien attention à ce point car beaucoup de développeurs, même expérimentés, font l’erreur.

Pour voir ou retrouver la propriété d’un objet :

 <script>

function Stagiaire(param1, param2, param3){

this.nom = param1;

this.age = param2;

this.email = param3;

}

var personne = new Stagiaire ("pierre", 30, "pierre@gmail.fr");

for(var prop in personne){

document.write("prop = " + prop + " ---> valeur = "+ personne[prop] + "<br>");

}

</script>

Exemple de Class-objet (exo)

<body>

<script>

function voiture(marque, annee, prix) {

this.marque = marque;

this.annee = annee;

this.prix = prix;

this.annonce = function () {

return ("Voiture de marque <strong>" + this.marque+"</strong> de <span class='annee'>" + this.annee + "</span> pour la modique somme de <u>" +this.prix+"</u> Euro");

//return ("Voiture de marque " + this.marque+" de " + this.annee + " pour la modique somme de " +this.prix+" Euro");

}

}

var garage = [];

garage[0] = new voiture("Renault Clio",2011, 10500);

garage[1] = new voiture("Fiat Tipo",2017, 14500);

garage[2] = new voiture("Citroen Picasso",20116, 24500);

garage[3] = new voiture("Ferrari F9",1994, 44500);

// garage[garage.length] = new voiture("Renault Clio",2011, 10500);

// garage[garage.length] = new voiture("Fiat Tipo",2017, 14500);

// garage[garage.length] = new voiture("Citroen Picasso",20116, 24500);

// garage[garage.length] = new voiture("Ferrari F9",1994, 44500);

</script>

<h1>Garage "la Joie des DWWM"</h1>

<ul>

<script>

for(var i = 0 ; i< garage.length; i++){

document.write("<li>"+garage[i].annonce()+"</li><br>");

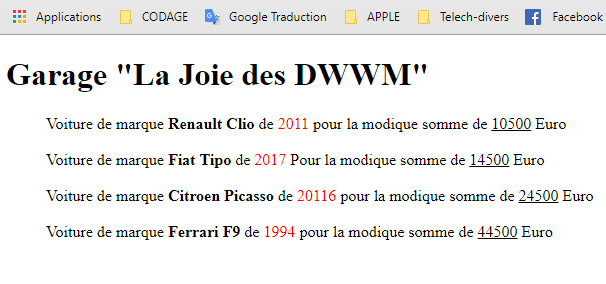
}

</script>

</ul>

</body>

Ce qui donne :



***Modification d’un lien en Javascript***

*Exemple 1 :*

<body>

<!-- Exemple 1:

On a une dic : <div id="container">. Dans cette div, on a le code html suivant:

un lien vers une page <a href="#">contact</a> avec un formulaire pour laisser vos coordonnées.

- modifier par javascript le lien pour qu'il mène vers "/page/contact.html" -->

<div id="container">

lien vers une page <a href="#"> Contact</a> avec un formulaire pour laisser vos coordonnées.

</div>

<script>

var madiv = document.getElementById('container');

var link = madiv.getElementsByTagName('a')[0];

link.href = '/page/contact.html';

</script>

</body>

On modifie le lien en allant chercher la div :

var madiv = document.getElementById('container');

puis on demande au script d’aller chercher la balise « a »

var link = madiv.getElementsByTagName('a')[0];

puis on change l’adresse du lien :

link.href = '/page/contact.html';

Pour vérifier si on a bien atteint l’élément voulu :

var madiv = document.getElementById('container');

if (madiv){

var link = madiv.getElementsByTagName('a')[0];

if (link){

link.href = '/page/contact.html';

}

}

***Modification d’un tag d’un paragraphe et ajout d’une class***

*Exemple 2 :*

*Le CSS doit être prêt et fonctionnel afin de pouvoir prendre en compte les modifications JS*

<!-- Exemple 2:

On a une div: <div id="container">. Dans cette div on a le code html suivant:

<p id="note"> ceci est un paragraphe de test </p>

Modifier par javascript le tag de paragraphe et y ajouter une class

-->

<div id="container">

<p id="note"> ceci est un paragraphe de test </p>

</div>

<script>

var p = document.getElementById('note');

if (p){

p.className = 'note.importante';

}

</script>

# ***L’element.appendChild***

La méthode **Node.appendChild()** ajoute un nœud à la fin de la liste des enfants d'un nœud parent spécifié. Si l'enfant donné est une référence à un nœud existant dans le document, appendChild() le déplace  de sa position actuelle vers une nouvelle position (il n'est pas nécessaire de supprimer le noeud sur son noeud parent avant de l'ajouter à un autre).

<!-- exemple 4:

pour vous excercer à créer des éléments html en Dom et Javascript:

recréer le bout de html suivant dans la div "container" vide:

<div id="down">

<strong> L'Hypertext Markeup Language </strong>, généralement abrégé

<a href = "https://fr.wikipedia.org/wiki/Hypertext\_Markup\_Language">

<strong>HTML</strong></a>, est le balisage conçu pour représenter

les pages web.<br>C'est un langage permettant d'écrire de l'Hypertext, d'où son nom,<br>

il est promu par la <a href="http://fr.wikipedia.org/wiki/world\_wide\_web"> World Wide Web </a>

</div>-->

<div id="container"></div>

<script>

var madiv = document.createElement('div');

madiv.id = 'dwwm';

var bold1 = document.createElement('strong');

bold1.appendChild(document.createTextNode("L'HyperText Markup Language"));

var bold2 = document.createElement('strong');

bold2.appendChild(document.createTextNode('HTML'));

var link1 = document.createElement('a');

link1.href = 'https://fr.wikipedia.org/wiki/Hypertext\_Markup\_Language';

link1.appendChild(bold2);

var link2 = document.createElement('a');

link2.href = 'http://fr.wikipedia.org/wiki/World\_Wide\_Web';

link2.appendChild(document.createTextNode('World Wide Web'));

madiv.appendChild(bold1);

madiv.appendChild(document.createTextNode(', generalement abregé '));

madiv.appendChild(link1);

madiv.appendChild(document.createTextNode(', est le langage de balisage crée pour representer les pages web.'));

madiv.appendChild(document.createElement('br'));

madiv.appendChild(document.createTextNode("C'est un langage permettant d'ecrire de l'hypertexte, d'où son nom."));

madiv.appendChild(document.createElement('br'));

madiv.appendChild(document.createTextNode('il est promu par le '));

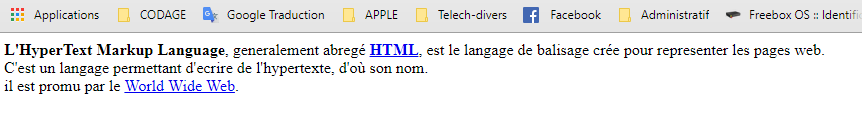
madiv.appendChild(link2);

madiv.appendChild(document.createTextNode('.'));

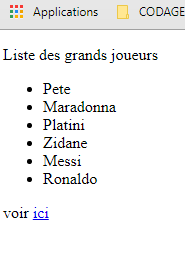
document.getElementById('container').appendChild(madiv);

</script>

Ce qui donne :



***Création d’une liste en javascript :***

<div id="container">

<p>Liste des grands joueurs</p>

<ul id="malist">

<li>Pete</li>

<li>Maradonna</li>

<li>Platini</li>

<li>Zidane</li>

<li>Messi</li>

<li>Ronaldo</li>

</ul>

<p>voir <a href="page.html">ici</a></p>

</div>

<script>

madiv = document.getElementById("container");

var NomJoueurs = [

'Pelé', 'Maradonna', 'Platini',

'Zidane', 'Messi', 'Ronaldo'

];

var tagP = document.createElement('p');

tagP.appendChild(document.createTextNode('Les grands joueurs de football:'));

var maliste = document.createElement('ul');

var joueur;

var l = NomJoueurs.length;

for (var i = 0; i < l; i++) {

joueur = document.createElement('li');

joueur.appendChild(document.createTextNode(NomJoueurs[i]));

maliste.appendChild(joueur);

}

var lien = document.createElement("a");

lien.href = "page.html";

lien.appendChild(document.createTextNode('ICI'));

var tagP2 = document.createElement('p');

tagP2.appendChild(document.createTextNode('Voir '));

tagP2.appendChild(lien);

madiv.appendChild(tagP);

madiv.appendChild(maliste);

madiv.appendChild(tagP2);

</script>

***Les Cookies :***

Les cookies vous permettent de stocker des informations sur les utilisateurs dans des pages Web.

***Créer un cookie avec JavaScript***

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

Avec JavaScript, un cookie peut être créé comme ceci:

document.cookie = "username=John Doe";

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

document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 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=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";

***Lire un cookie avec JavaScript***

Avec JavaScript, les cookies peuvent être lus comme ceci:

var x = document.cookie;

document.cookie renverra tous les cookies dans une chaîne comme ceci: cookie1 = value; cookie2 = valeur; cookie3 = valeur;

***Changer un cookie avec JavaScript***

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

document.cookie = "username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";

Le vieux cookie est écrasé.

***Supprimer un cookie avec JavaScript***

Supprimer un cookie est très simple.

Il n'est pas nécessaire de spécifier une valeur de cookie lorsque vous supprimez un cookie.

Il suffit de définir le paramètre expires sur une date passée:

document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";

Vous devez définir le chemin du cookie pour vous assurer de supprimer le bon cookie.

Certains navigateurs ne vous laisseront pas supprimer un cookie si vous ne spécifiez pas le chemin.

***La ficelle du cookie***

La propriété document.cookie ressemble à une chaîne de texte normale. Mais ce n'est pas.

Même si vous écrivez toute une chaîne de cookie dans document.cookie, lorsque vous le relisez, vous ne pouvez en voir que la paire nom-valeur.

Si vous définissez un nouveau cookie, les anciens cookies ne sont pas écrasés. Le nouveau cookie est ajouté à document.cookie. Par conséquent, si vous relisez document.cookie, vous obtiendrez quelque chose comme:

cookie1 = valeur; cookie2 = valeur;

Afficher tous les cookies    Créer un cookie 1   Créer un cookie 2    
 Supprimer le cookie 1   Supprimer le cookie 2

Si vous souhaitez rechercher la valeur d'un cookie spécifié, vous devez écrire une fonction JavaScript qui recherche la valeur du cookie dans la chaîne de cookie.

***Exemple de cookie JavaScript***

Dans l'exemple à suivre, nous allons créer un cookie qui stocke le nom d'un visiteur.

La première fois qu'un visiteur arrive sur la page Web, il lui sera demandé d'indiquer son nom. Le nom est ensuite stocké dans un cookie.

La prochaine fois que le visiteur arrivera sur la même page, il recevra un message de bienvenue.

Pour l'exemple, nous allons créer 3 fonctions JavaScript:

1. Une fonction pour définir une valeur de cookie
2. Une fonction pour obtenir une valeur de cookie
3. Une fonction pour vérifier une valeur de cookie

## ***Une fonction pour définir un cookie***

Tout d'abord, nous créons une fonction qui stocke le nom du visiteur dans une variable cookie:

function setCookie(cname, cvalue, exdays) {

var d = new Date();

d.setTime(d.getTime() + (exdays\*24\*60\*60\*1000));

var expires = "expires="+ d.toUTCString();

document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";

}

Les paramètres de la fonction ci-dessus sont le nom du cookie (cname), la valeur du cookie (cvalue) et le nombre de jours jusqu'à l'expiration du cookie (exdays).

La fonction définit un cookie en ajoutant ensemble le nom de cookie, la valeur du cookie et la La fonction définit un cookie en ajoutant ensemble le nom de cookie, la valeur du cookie et la chaîne d'expiration.

***Une fonction pour obtenir un cookie***

Ensuite, nous créons une fonction qui retourne la valeur d'un cookie spécifié:

function getCookie(cname) {

var name = cname + "=";

var decodedCookie = decodeURIComponent(document.cookie);

var ca = decodedCookie.split(';');

for(var i = 0; i <ca.length; i++) {

var c = ca[i];

while (c.charAt(0) == ' ') {

c = c.substring(1);

}

if (c.indexOf(name) == 0) {

return c.substring(name.length, c.length);

}

}

return "";

}

**Fonction expliquée:**

Prenez le cookiename en paramètre (cname).

Créez une variable (nom) avec le texte à rechercher (cname + "=").

Décoder la chaîne de cookie, pour gérer les cookies avec des caractères spéciaux, par exemple '$'

Divisez document.cookie sur des points-virgules dans un tableau appelé ca (ca = decodedCookie.split (';')).

Boucle dans le tableau ca (i = 0; i <longueur.al; i ++) et lit chaque valeur c = ca [i]).

Si le cookie est trouvé (c.indexOf (name) == 0), renvoyez la valeur du cookie (c.substring (name.length, c.length).

Si le cookie n'est pas trouvé, retournez "".

***Une fonction pour vérifier un cookie***

Enfin, nous créons la fonction qui vérifie si un cookie est défini.

Si le cookie est défini, un message d'accueil s'affiche.

Si le cookie n'est pas défini, il affichera une boîte de dialogue demandant le nom de l'utilisateur et stockera le cookie du nom d'utilisateur pendant 365 jours en appelant la fonction setCookie:

function checkCookie() {

var username = getCookie("username");

if (username != "") {

alert("Welcome again " + username);

} else {

username = prompt("Please enter your name:", "");

if (username != "" && username != null) {

setCookie("username", username, 365);

}

}

}

***Tous ensemble maintenant***

function setCookie(cname, cvalue, exdays) {

var d = new Date();

d.setTime(d.getTime() + (exdays \* 24 \* 60 \* 60 \* 1000));

var expires = "expires="+d.toUTCString();

document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";

}

function getCookie(cname) {

var name = cname + "=";

var ca = document.cookie.split(';');

for(var i = 0; i < ca.length; i++) {

var c = ca[i];

while (c.charAt(0) == ' ') {

c = c.substring(1);

}

if (c.indexOf(name) == 0) {

return c.substring(name.length, c.length);

}

}

return "";

}

function checkCookie() {

var user = getCookie("username");

if (user != "") {

alert("Welcome again " + user);

} else {

user = prompt("Please enter your name:", "");

if (user != "" && user != null) {

setCookie("username", user, 365);

}

}

}

L'exemple ci-dessus exécute la fonction **checkCookie ()** lors du chargement de la page.

# **Script pour imprimer une zone de l'écran**

Voici un script permettant d'imprimer une zone bien définie avec JavaScript. Pour cela, nous allons créer une popup est copier le contenu de cette zone définie dans ce popup. Vous trouverez le code JavaScript en lui-même ainsi que la méthode pour l'intégrer dans une page HTML.

## **Le code JavaScript**

<script language="javascript">

function imprimer\_bloc(titre, objet) {

*// Définition de la zone à imprimer*

var zone = document.getElementById(objet).innerHTML;

*// Ouverture du popup*

var fen = window.open("", "", "height=500, width=600,toolbar=0, menubar=0, scrollbars=1, resizable=1,status=0, location=0, left=10, top=10");

*// style du popup*

fen.document.body.style.color = '#000000';

fen.document.body.style.backgroundColor = '#FFFFFF';

fen.document.body.style.padding = "20px";

*// Ajout des données a imprimer*

fen.document.title = titre;

fen.document.body.innerHTML += " " + zone + " ";

*// Impression du popup*

fen.window.print();

*//Fermeture du popup*

fen.window.close();

return **true**;

}

</script>

On colle ce code dans la page HTML, avant le </header> si possible.

## **Appel de la fonction JavaScript**

Maintenant que la fonction est créer nous allons vous expliquer comment l'intégrer dans votre page HTML.

Pour définir la zone a imprimer, nous allons l'intégrer (l'encapsuler) entre une balise

et lui donner un nom (dans notre exemple : imprime\_moi).

Avec cette exemple vous pouvez intégrer plusieurs liens (ou boutons) dans une même page, il suffit de changer le nom de chaque zone a imprimer.

<[**div**](http://december.com/html/4/element/div.html) id="imprime\_moi">

Ceci est le texte contenu dans la zone a imprimer

</[**div**](http://december.com/html/4/element/div.html)>

// Lien

<[**a**](http://december.com/html/4/element/a.html) href="#" onclick="javascript:imprime\_bloc('titre', 'imprime\_moi');">Cliquez ici pour imprimer la zone</[**a**](http://december.com/html/4/element/a.html)>