JavaScript v3.0 12/12/2015

Lier un fichier CSS:
<link rel='stylesheet' type='text/css' href='stylesheet.css'/>
<link rel="stylesheet" href="design.css" /> *//HTML5*
Lier un fichier JS:
<script type="text/javascript" src="script.js"></script>
<script src="script.js"></script> *//HTML5*

Déclaration de variable:
var nomVariable = "Chaîne";
var nomVariable = 1234;
var person = "John Doe", carName = "Volvo";
var $test1 = "lol", _test2="lol2";

Ecrire dans la console:
console.log(2 * 5) ;
console.log("Salut") ;

Commentaire
//Commentaire
/ Commentaire */*

In JavaScript there are 5 different data types that can contain values:
- ✓ string
- ✓ number
- ✓ boolean
- ✓ object
- ✓ function

There are 3 types of objects:
- ✓ Object
- ✓ Date
- ✓ Array

And 2 data types that cannot contain values:
- ✓ null
- ✓ undefined

Infinity = 1 .79769313486231570e+308
« A » === « \u0041 »
'c' + 'a' + 't' === 'cat'
\b -> retour arrière
\f -> nouvelle page
\n -> nouvelle ligne
\r -> retour chariot
\t -> tab
\u -> 4 chiffres héxadécimaux

Instruction d'interruption :
Instruction break -> L'instruction break permet de terminer la boucle en cours ou l'instruction switch ou

label en cours et de passer le contrôle du programme à l'instruction suivant l'instruction terminée.

Instruction return -> L'instruction return met fin à l'exécution d'une fonction et définit une valeur à renvoyer à la fonction appelante.
Instruction throw -> L'instruction throw permet de lever une exception définie par l'utilisateur. L'exécution de la fonction courante sera stopée (les instructions situées après l'instruction throw ne seront pas exécutées) et le contrôle sera passé au premier bloc catch de la pile d'appels. Si aucun bloc catch ne se trouve dans les fonctions de la pile d'appels, le programme sera terminé.

Valeurs équivalentes à false :
- false
- null
- undefined
- la chaîne vide ''
- le chiffre 0
- le nombre NaN

Priorité des opérateurs :
1. . [] ()
2. delete new typeof + - !
3. $ / %
4. + -
5. >= <= > <
6. === !==
7. &&
8. | |
9. ?:

Variable Globale Réduite :
var VARGLOBALE = {} ;
VARGLOBALE.profile = {
"nom" :"John",
"login" :"coco"
} ;

<BEST PRACTICE>

- ✓ Ne PAS utiliser: new Object()
  - var x1 = {}; *// new object*
  - var x2 = ""; *// new primitive string*
  - var x3 = 0; *// new primitive number*
  - var x4 = false; *// new primitive boolean*
  - var x5 = []; *// new array object*
  - var x6 = /()/; *// new regexp object*
  - var x7 = function(){}; *// new function object*
- ✓ Réduire les accès au DOM:
  - obj = document.getElementById("demo");
  - obj.innerHTML = "Hello";

<STRING>

Longueur d'une chaîne :
"Chaîne de caractères".length;
Extraire des lettres d'une chaîne :
"Chaîne de caractères".substring(3,7);

Mettre une chaîne en MAJUSCULE:
"Chaîne de caractères ".toUpperCase();
Mettre une chaîne en minuscule:
"Chaîne de caractères ".toLowerCase();

charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a copy of the joined strings
fromCharCode()	Converts Unicode values to characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string
localeCompare()	Compares two strings in the current locale
match()	Searches a string for a match against a regular expression, and returns the matches
replace("àremplacer","remplacerpar")	Searches a string for a value and returns a new string with the value replaced
search()	Searches a string for a value and returns the position of the match
slice(start, end)	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
substr(start, length)	Extracts a part of a string from a start position through a number of characters
substring(start, end)	Extracts a part of a string between two specified positions
toLocaleLowerCase()	Converts a string to lowercase letters, according to the host's locale
toLocaleUpperCase()	Converts a string to uppercase letters, according to the host's locale
toString()	Returns the value of a String object
trim()	Removes whitespace from both ends of a string
valueOf()	Returns the primitive value of a String object

<NUMBER>

Méthodes globales:

Number()	Returns a number, converted from its argument.
parseFloat()	Parses its argument and returns a floating point number
parseInt()	Parses its argument and returns an integer

Méthodes des nombres:

toString()	Returns a number as a string
toExponential()	Returns a string, with a number rounded and written using exponential notation.
toFixed()	Returns a string, with a number rounded and written with a specified number of decimals.
toPrecision()	Returns a string, with a number written with a specified length
valueOf()	Returns a number as a number
Math.random();	Returns a random number

<DATE>

Méthodes de dates:

getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

//Tous les éléments peuvent être SET également

<DISPLAY>

Afficher en JavaScript
Writing into an alert box, using **window.alert()**.
Writing into the HTML output using **document.write()**.
Writing into an HTML element, using **innerHTML**.
Writing into the browser console, using **console.log()**.

Boite de dialogue de confirmation :
confirm("message") ;

Boite de dialogue de saisie :
prompt("Qui es-tu ?");

Boite de dialogue d'affichage :
alert("Qui es-tu ?");

<LOOP & CONDITIONAL>

Syntaxe WHILE:
```
compris = true;
while(compris){
        console.log("J'apprends les boucles while !");
        compris = false;
}
```

Syntaxe DO WHILE :
```
conditionBoucle = false;
do { // Instruction(s) }
while (conditionBoucle);
```

Syntaxe for:
```
for (var i = 5; i < 51; i+=5) {
        console.log(i);
}
```

Syntaxe ternaire:
variablename = (condition) ? value1:value2
var voteable = (age < 18) ? "Too young":"Old enough";

<TABLE>

Déclaration de tableau :
var mixed = [34, "bonbon", "bleu", 11];

Afficher un élément du tableau :

```
console.log(mixed[3]);
// Affiche la 4ème cellule du tableau mixed
```

Parcourir un tableau:
```
var langages = ["HTML", "CSS", "JavaScript"];
for(i = 0 ; i < langages.length ; i++)
{
    console.log(langages[i]);
}
```

Passer certains éléments dans un tableau:
```
nomTableau.push(text[i]);
```

Tableau multi-dimensions:
```
var nouveauTableau = [  [1,1,1] , [1,1,1], [1,1,1]  ];
```

Conversion Array>String:
```
var fruits = ["Banana", "Orange","Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join("
* ");
```

Ajouter et/ou supprimer un élément d'un tableau:
```
tableau.concat();
// Produit un nouveau tableau contenant une copie superficielle de ce
tableauà laquelle sont ajoutés les elements.
tableau.join();
// Cela crée une chaine à partir d'un tableau et le séparateur par défaut
est ",". Pour effectuer une jointure sans séparateur, faire un tableau.join('');
 fruits.pop();
// Removes the last element ("Mango") from fruits
fruits.push("Kiwi");
 // Adds a new element ("Kiwi") at the end of fruits
fruits.reverse();
// Sorts the elements of fruits and
 // Reverses the order of the elements
fruits.shift();
// Removes the first element "Banana" from fruits
delete montableau[0];
// Supprime la première valeur de mon tableau (la remplace par
undefined)
tableau.slice(0, 1);
 // Crée une copie superficielle d'une portion d'un tableau
fruits.sort();
 // Sorts the elements of fruits
fruits.splice(2, 0, "Lemon", "Kiwi");
//Ajouter les éléments donnés à la case 2 mais n'éffacer aucun élément
déjà present (Décale les autres).
fruits.unshift("Lemon");
// Adds a new element "Lemon" to the beginning of fruits
fruits[0] = "Kiwi";
 // Changes the element [0] of fruits to "Kiwi"
fruits[fruits.length] = "Kiwi";
  // Appends "Kiwi" to fruit (ajout à la fin)
delete fruits[0];
// Changes the first element in fruits to undefined
```

<OBJECT>

Création d'objet (Méthode littérale):
```
var monObj = {
   clef1: valeur,
   clef2: valeur
}
var monObj = {};
```

Exemple:
```
var amis = {};
amis.bill = { prenom: "Bill", nom: "Gates",
numero: "(206) 555-5555"};
```

Donner une valeur à une clef (Méthode littérale):
```
myObj.clef1 = valeur;
// OU
myObj['clef2'] = valeur;
```

Création d'objet (Méthode objet):
```
var monObjet = new Object();
```

Donner une valeur à une clef (Méthode objet):
```
myObj.clef1= valeur;
// OU
myObj['clef2'] = valeur;
```

Créer un objet via une classe:
```
function Personne(nom,age) {
  this.nom = nom;
  this.age = age;
}
```

var bob = new Personne("Bob Smith", 30);
```
// Recréons bob, en utilisant notre constructeur
```

Lister les propriétés:
```
for (var prop in object){
    console.log(prop);
}
```
Lister les valeurs:
```
for (var prop in object){
    console.log(object.prop);
}
```

Afficher le type d'un élément:
```
console.log( typeof nomElement );
```

Afficher si l'objet possède la propriété :
```
var monObj = {
   nom: "Robert"
};
console.log( monObj.hasOwnProperty('nom') );
 // devrait afficher true
console.log( monObj.hasOwnProperty('surnom') );
// devrait afficher false
```

<FUNCTION>

Syntaxe fonction:
```
function disBonjour(nom) {
   console.log('Bonjour ' + nom);
};
```

Prototype:
```
Chat.prototype.miauler = function()
{
   console.log("Meow !");
};
```

Accéder au variables privées:
```
function Personne(prenom) {
   this.prenom = prenom;
```

```
  var compteEnBanque = 7500;

  this.getCompte = function() {
    return compteEnBanque;
  };
}
```

Variable privée et publique:

```
function Personne(prenom,nom,age) {
  this.age = age;
// Variable publique
  var compteEnBanque = 7500;
// Variable privée
}
```

Utilisation des variables

```
// code here can not use carName
function myFunction() {
   var carName = "Volvo";
   // code here can use carName
}
```

Variables Globales

```
// code here can use window.carName in HTML
// code here can use carName
function myFunction() {
   carName = "Volvo";
   // code here can use carName
}
```

Fonction anonyme:

```
var myAnonymousFunction = function() {
    console.log('myAnonymousFunction is called');
}
```

<CLOSURE>

Une closure est un contexte crée par la déclaration d'une fonction, et qui permet à cette dite fonction d'accéder et manipuler des variables se trouvant en dehors de la portée de cette fonction.

Exemple :

```
var luke = "luke";
var jedi;
function foo(){
        var vador = "je suis ton père";
        function bar(){
                console.log(luke, vador);
        }
        jedi = bar;
}
foo();
jedi();
// Affiche luke je suis ton père
```

<u>Cas d'usages :</u>

1) Variables privées (encapsulation)

```
function Jedi(){
        var jedi = "";
        this.luke = function(){
                jedi = "Luke";
                return this;
        }
        this.imYourFather = function(){
                return jedi + ", Je suis ton père";
        };
};
var jedi = new Jedi();
jedi.luke().imYourFather(); // Luke, Je suis ton père
```

2) Fonction de callback

```
<div id="content"></div>
<button id="load">Vador says...</button>
<script>
//
var content = document.querySelector('#content');
var click = function(){
        content.innerHTML = 'Luke, ...';
        var xhr = new XMLHttpRequest();
        xhr.open('GET', '/api/users/luke', false);
        xhr.onload = function(data){
        // data.response : ... I'm your father!
        content.innerHTML = data.response;
        };
        xhr.send();
};
document.querySelector('#load').addEventListener('click',
click);
//
</script>
```

3) Fonction partielle

```
function curry(fn) {
 var restArgs = Array.prototype.slice.call(arguments, 1);
 return function() {
   var arg = Array.prototype.slice.call(arguments);
   return fn.apply(null, arg.concat(restArgs));
 };
};
```

Immediately Invoked Execution Fonction:

```
var test_1 = 'Hello';
(function(){
  console.log(test_1); // Hello
  var test_2 = 'World';
  console.log(test_2); // World
})();
console.log(test_2); // not defined
```

<ERROR>

```
try {
   //Block of code to try
}
catch(err) {
   //Bloc de code pour gérer les erreurs
}
finally {
        //Bloc de code doit être exécuté indépendamment du résultat de
        try / catch
        }
```

<HTML DOM>

Document Object Model
Le DOM HTML est un standard pour savoir comment obtenir, modifier, ajouter ou supprimer des éléments HTML.
Il définit:

- ✓ Les éléments HTML comme des objets
- ✓ Les propriétés de tous les éléments HTML
- ✓ Les méthodes d'accéder à tous les éléments HTML
- ✓ Les événements pour tous les éléments HTML

Trouver des éléments HTML:
```
document.getElementById()
//Find an element by element id
document.getElementsName()
//Find elements by name
document.getElementsByTagName()
//Find elements by tag name
document.getElementsByClassName()
//Find elements by class name
element.children()
//Renvoit un tableau de tous les elements enfants de l'élément scpécifié

element.childNodes ()
//Renvoit un tableau de tous les Noeuds enfants de l'élément scpécifié
document.querySelectorAll("p.intro");
// Finding HTML Elements by CSS Selectors
document.forms["frm1"];
// Finding HTML Elements by HTML Object Collections
```

Modifier des éléments HTML:
```
element.innerHTML=
//Change the inner HTML of an element
element.innerText=
//Change the inner Text of an element
element.attribute=
//Change the attribute of an HTML element
element.setAttribute(attribute,value)
//Change the attribute of an HTML element
element.style.property=
//Change the style of an HTML element
```

Ajouter et supprimer des éléments HTML:
```
document.createElement()
//Create an HTML element
document.removeChild()
//Remove an HTML element
document.appendChild()
//Add an HTML element
document.html()
//Add an HTML element
document.replaceChild()
//Replace an HTML element
document.write(text)
//Write into the HTML output stream
element.insertBefore(para,child);
//Insérer avant
```

Ajout d'éléments de gestion:
```
document.getElementById(id).onclick=function(){code}//Add ing event handler code to an onclick event
```

<MAIN ACTIONS>

Action qui attend une confirmation :
```
<a onclick="confirm()" style="cursor:pointer;">Cliquez ici</a> function Confirm(){
        if confirm("Tu es sur?"){
                //code
}}
```

Changer un contenu HTML
```
document.getElementById("nom de l'id").innerHTML = "Texte";
```

Changer un attribut HTML (ex : une image)
```
var image = document.getElementById('myImage');
  if (image.src.match("bulbon")) {
    image.src = "pic_bulboff.gif";
  } else {
    image.src = "pic_bulbon.gif";
  }
```

Changer la valeur d'un attribut:
```
document.getElementById(id).attribute=new value
//EX:
document.getElementById("myImage").src = "landscape.jpg";
```

Modifier le CSS
```
document.getElementById("demo").style.fontSize = "25px";
```

Récuperer un élément précis :
```
document.getElementByTagName("a")[0].href;
//Récupère la valeur du href de la 1ère balise <a> dans le tableau des balise <a> récupérées.
```

Valider des données
```
function myFunction() {
  var x, text;

  // Get the value of the input field with id="numb"
  x = document.getElementById("numb").value;

  // If x is Not a Number or less than one or greater than 10
  if (isNaN(x) || x < 1 || x > 10) {
    text = "Input not valid";
  } else {
    text = "Input OK";
  }
  document.getElementById("demo").innerHTML = text;
}
```

Mots clés

break>	Terminates a switch or a loop
continue>	Jumps out of a loop and starts at the top
debugger>	Stops the execution of JavaScript, and calls (if available) the debugging function

<EVENT>

http://www.w3schools.com/jsref/dom_obj_event.asp

Evènement HTML
```
<some-HTML-element some-event="some JavaScript">
//Exemple
<button onclick='getElementById("demo").innerHTML=Date()'>The time is?</button>
//OU
```

```
<button onclick="this.innerHTML=Date()">The time is?</button>
//OU
<button onclick="displayDate()">The time is?</button>
//Appel d'une fonction
```

Evènements HTML communs

onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

Mais aussi: onblur, onerror, onfocus, onkeypress, onkeyup, onmouseup, onsubmit.

```
//Exemple:
<div
onmousedown="mDown(this)"
onmouseup="mUp(this)"
onmouseover="mOver(this)"
onmouseout="mOut(this)"
style="background-color:#D94A38;width:90px;height:20px;padding:40px;">
Test Me</div>

<script>
function mDown(obj) {
    obj.style.backgroundColor = "#1ec5e5";
    obj.innerHTML = "Tu cliques";
}
function mUp(obj) {
    obj.style.backgroundColor="#D94A38";
    obj.innerHTML="Tu relaches";
}
function mOver(obj) {
    obj.innerHTML = "Over"
}
function mOut(obj) {
    obj.innerHTML = "Out"
}
```

Syntaxe addEventListener()

```
element.addEventListener(event, function, useCapture);
//EX:
document.getElementById("myBtn").addEventListener("click", function(){
    alert("Hello World!");
});
```

Plusieurs gestionnaires d'évènements pour le même élément:

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
removeEventListener() //Effacer un EventListener
```

< JS Browser BOM >

JS Window (Document Object Model

JS Screen
JS Location
JS History
JS Navigator
JS Popup Alert
JS Timing
JS Cookies