

essentiel javascript (es5, es6)

Table des matières

I - Javascript (présentation générale).....	4
1. Présentation (générale) de javascript.....	4
2. Présentation des principales bibliothèques.....	8
II - Bases du langage javascript (toutes versions).....	10
1. Variables et types.....	10
2. Les variables (implicites ou explicites).....	12
3. Les opérateurs et expressions.....	12
4. Instruction switch/case.....	14
5. Les boucles.....	14
6. Fonction eval.....	16

7. Objets Math , String , Date ,	17
8. Eléments divers du langage javascript.....	20
9. Nouveaux type d'objets (non prédéfinis) et tableaux.....	20
III - DOM et gestion des événements.....	23
1. DOM ET DHTML.....	23
2. Le modèle normalisé (DOM du W3C) (depuis IE5).....	23
IV - JSON , localStorage,	30
1. Format JSON.....	30
2. Manipulations JSON en javascript.....	30
3. Suppression/remplacement d'attribut javascript.....	30
4. LocalStorage et SessionStorage.....	31
V - Ajax (xhr , fetch , ...).....	32
1. Appels de WS REST (HTTP) depuis js/ajax.....	32
VI - Prototype.....	38
1. Prototype (javascript).....	38
VII - Essentiel es2015 (arrow function, ...).....	41
1. Mots clefs "let" et "const" (es2015).....	41
2. "Arrow function" et "lexical this" (es2015).....	42
3. for...of (es2015) utilisant itérateurs internes.....	43
VIII - Annexe – navigator , window , document (js).....	50
1. document , form , events , cookies (js).....	50
2. objets navigator , window, ... (js).....	57
IX - Annexe – Canvas et Chart.....	60
1. Api "canvas" et bibliothèque "chart".....	60
X - Annexe – JQuery.....	66
1. Présentation de JQuery.....	66
2. Essentiel de JQuery.....	69
3. Plugins JQuery.....	76
XI - Annexe – Bibliographie, Liens WEB + TP.....	81

1. Bibliographie et liens vers sites "internet".....	81
2. TP.....	81

I - Javascript (présentation générale)

1. Présentation (générale) de javascript

1.1. Généralité sur le langage javascript

Javascript est un langage interprété et sans typage fort qui est essentiellement utilisé dans le développement d'application web :

- Du côté "front-end", il est interprété en tant que complément des pages HTML par tous les navigateurs internet (IE/edge , firefox , Chrome , opera , ...)
- Du côté "back-end", il est pris en charge par le moteur "node-js" et peut par exemple service à gérer des web-services "REST" via le framework "express" .

"Javascript" est le nom commun/générique d'un langage dont les versions normalisées/standardisées sont appelées "EcmaScript" .

Bref Historique :

"Javascript" (alias "EcmaScript") est un langage qui a été conçu vers les années 1995 par les entreprises "Sun" et "NetScape" . "javascript" s'est inspiré du langage "java" au niveau de la syntaxe (boucles , mot clefs , bloc délimités par { } et objets ressemblants "String" , "Math") mais doit être considéré comme un langage très différent (pas compilé , pas fortement typé , ...)

Ce langage a été ensuite rapidement utilisé par "Microsoft" au sein de son navigateur "internet explorer" .

Vers 1997,...1999, 2000,, 2004, ... il y avait à l'époque d'assez grandes différences dans l'interprétation du langage javascript au sein des différents navigateurs existants (pas la même façon de différencier minuscules/majuscules , différentes façons de gérer les événements et quelques parties des documents HTML,) .

Pour masquer ces disparités, certaines bibliothèques de plus haut niveau ont été ensuite conçues et utilisées vers les années 2007 , ... , 2012, (ext-js , jquery ,) .

"jquery" est la bibliothèque "javascript" qui a été la plus utilisée (et qui est encore un peu utilisé aujourd'hui) .

Les navigateurs modernes ont heureusement fait des efforts pour interpréter l'essentiel de javascript/ecmascript de la même façon et aujourd'hui n'y a plus beaucoup de différence dans l'interprétation de javascript (en version "es5")entre les versions modernes de IE/Edge , Firefox , Chrome , Opera, Safari , ...

En 2015 et en 2017, le langage "javascript / ecmascript" a beaucoup évolué .

Les nouvelles syntaxes de "es2015" / "es2017" n'étant supportées (en 2017, 2018, 2019, 2020) que par des navigateurs très modernes , on a pour l'instant souvent besoin de transformer du code source "es2015/es2017" en du code "es5" (via "babel" par exemple) de façon à ce que les instructions soient correctement interprétées par la majorité des navigateurs utilisés .

Depuis, 2012, ..., 2014, ... le langage "javascript" est maintenant également utilisé côté "serveur"

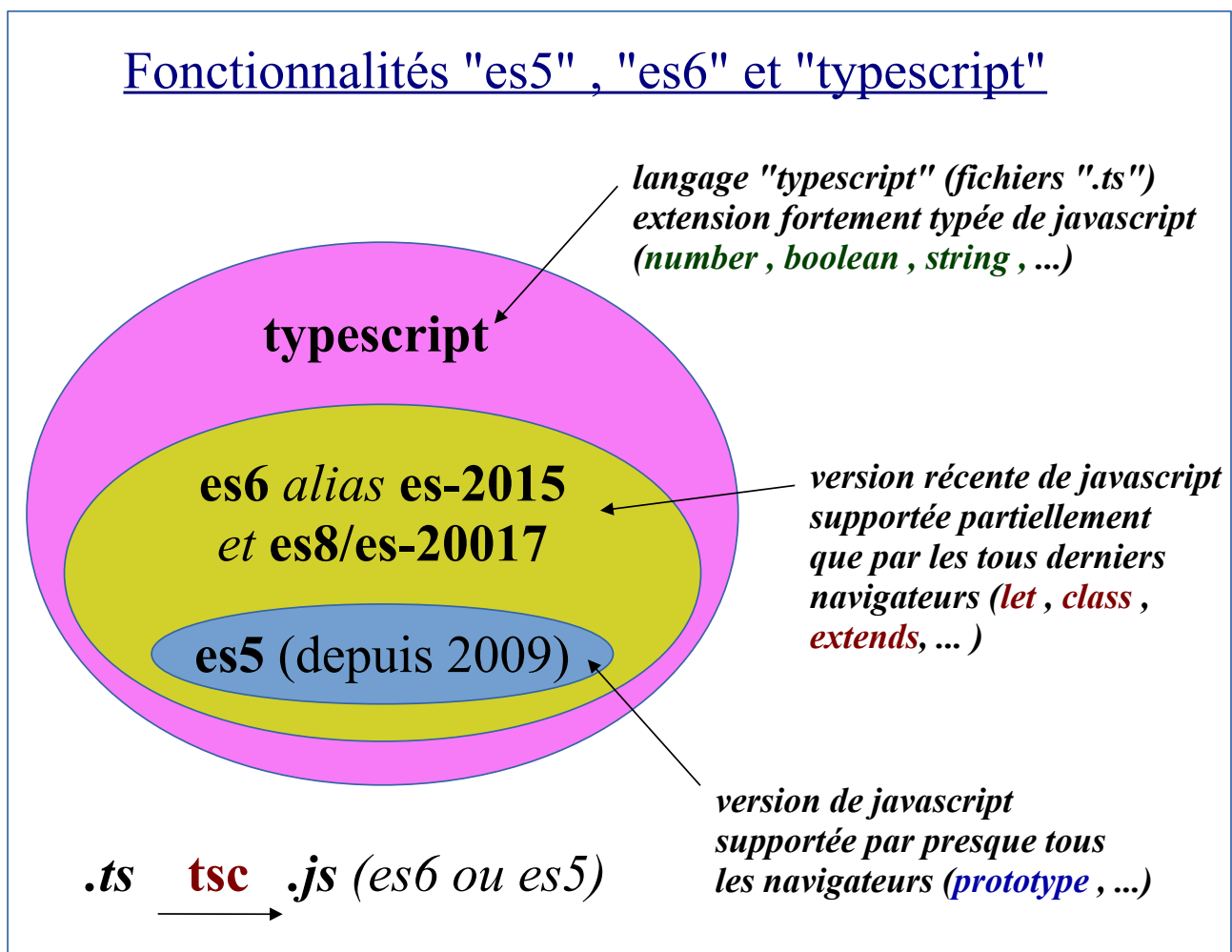
de manière très modulaire (essentiellement via l'écosystème node-js/npm) .
Son fonctionnement très "asynchrone" a permis d'obtenir de bonnes performances a peu de frais .
Beaucoup de "bonnes idées de javascript/node-js" (asynchronisme , modularité, ...) sont petit à petit reprises et intégrées dans des langages et écosystèmes concurrents (ex: java 10 , spring >=5 , ...) .

Le langage javascript/ecmascript (et sa variante fortement typée "typescript") est aujourd'hui beaucoup utilisé (coté "front-end") au sein de framework "Single Page Application" (ex : "VueJs" , "react" , "angular" , ...) et également au sein d'application pour mobile (via "cordova/ionic" ou "PWA/service-worker") .

Javascript et DOM (pour HTML/CSS) :

Des instructions "javascript" sont très souvent utilisées pour manipuler dynamiquement des parties d'une page HTML (contenu textuel , composants , styles css) via l'API **DOM** (Document Object Model) .

1.2. Différentes versions et variantes/extensions de "EcmaScript"



es5 (de 2009) est supporté par quasiment tous les navigateurs
es6 (alias es2015) sera petit à petit supporté (partiellement ou complètement) par de plus en plus de navigateur .
es2017 (avec nouveaux mots clefs `async` , `await`) n'est pour l'instant quelque-chose d'assez récent .

Beaucoup de développements modernes peuvent néanmoins s'appuyer sur des syntaxes récentes de es2015/es2017 (voir typescript) en se reposant sur une étape de traduction/transformation du code :

scrXy.ts (ou .es2015.js ou .es2017.js) ==> (via babel ou ...) ==> scrXy.es5.js

1.3 Insertion du code JavaScript dans une page HTML

```
<script>
//instructions en JavaScript
</script>
```

Remarque:

Pour créer des **commentaires** en **HTML** on utilise la structure `<!-- Commentaire -->` .
En JavaScript les commentaires commencent par un double slash (`//`) situé au début du commentaire et continuent jusqu'à la fin de la ligne.

Exemple simple (pour la syntaxe uniquement) :

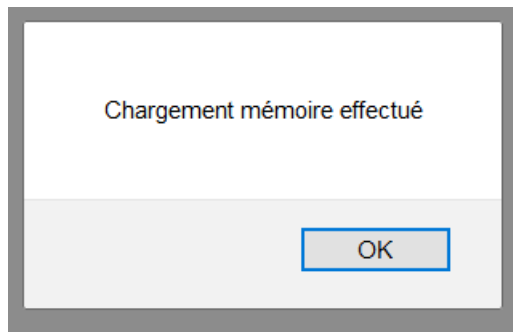
```
<html>
<head>
<title> exemple JavaScript simple </title>
<script>
    function affDlg(msg){
        alert(msg) ;
    }

    function affDoc(msg){
        document.write(msg)
    }
</script>
</head>
<body onload="affDlg('Chargement mémoire effectué')">

<script>
affDoc("Ligne1 <br/> Ligne2 générée via JavaScript <br/>")
var chaine="Javascript"
affDoc(chaine.toUpperCase().bold()) ; affDoc("<br/>")
</script>

<input type="button" value="b1" onclick="affDlg('Click bouton')" >
</body>
</html>
```

Ligne1
Ligne2 générée via JavaScript
JAVASCRIPT
b1



1.5 Référence vers un fichier de script séparé

```
...  
<script src="util.js"></script>  
...
```

Le *fichier texte* util.js pourra ainsi comporter un ensemble de fonctions JavaScript prêtes à être réutilisées dans différentes pages HTML.

1.6 Principales fonctionnalités de JavaScript

JavaScript est principalement utilisé pour :

- **ajouter différents gadgets** (de bon ou de mauvais goût): message défilant , animations, ...
- Fenêtre secondaire que l'on n'a pas demandée (harcèlement publicitaire)
-
- **effectuer des contrôles de saisie** (véritablement utiles pour soulager le réseau et le serveur).
- **rendre le document interactif** (prompt, boîtes de dialogues , menus déroulants,).
- **relier entre eux différents composants d'une même page HTML**. Par exemple, un click sur un bouton poussoir va permettre de déclencher une fonction JavaScript qui va activer une opération sur un autre composant (démarrer ou stopper une animation au sein d'un composant multimédia ,).
- **gérer des Cookies** (Mémoire des préférences de l'utilisateur,...).
- gérer certains effets graphiques (visibilité ,changement d'image, styles CSS, ...).
- Déclencher des **requêtes ajax** pour récupérer (ou ...) des données depuis le serveur
- **modifier le contenu de la page courante** (ajout de données dans un tableau , ...)
- ...

1.7 affichage au sein d'une <div>

```
function affInDivA(msg){  
    var divA=document.getElementById('divA');  
    divA.innerHTML="message=<b>"+msg+"</b>";  
}
```

```
<body onload="affInDivA('Chargement mémoire effectué')">
```

```
blabla<br/>
<div id="divA"></div>
....
</body>
```

1.3. Boîtes de dialogue (alert, prompt, confirm)

alert("message") affiche une simple boîte l'alerte pour informer l'utilisateur.

réponse = **prompt**("question", "valeur par défaut") permet d'afficher une boîte de dialogue au sein de laquelle l'utilisateur peut fixer la valeur d'un certain paramètre.

if(**confirm**("voulez vous ... ?")) ... permet de demander une confirmation de façon à effectuer un certain traitement avec l'approbation de l'utilisateur.

1.4. Affichage dans la console du navigateur (pratique pour debug)

console.log("message qui va bien");

La console du navigateur internet ne s'affiche que via le menu "développement web" (plus d'outils) / "console web" (Ctrl-Maj-I "Chrome" ou Ctrl-Maj-K "FireFox")

Remarque importante : au sein du langage javascript , une chaîne de caractère peut être délimitée par des simples ou doubles quotes : "message qui va bien" ou 'message qui va bien' .

2. Présentation des principales bibliothèques

Une bibliothèque "javascript" peut généralement être vue comme :

- * une extension au langage de base (nouvelles fonctionnalités)
- * un mini framework qui automatise certains points
- * une couche d'abstraction qui permet d'écrire moins de lignes de "bas niveau"
- * une couche supplémentaire qui masque certaines différences entre navigateurs

Bibliothèques "javascript"	Principales caractéristiques/fonctionnalités
dojo	Une des premières extensions "web2" maintenant un peu passée de mode
ext-js	Bonne extension (assez complète) mais cependant moins populaire que jquery.
jquery	Extension qui est la plus populaire (bon graphisme , assez simple , standard de fait)
handlebars	templates HTML réutilisables et paramétrables en "pur javascript" (coté client / navigateur)
canvas , chart	Api pour dessin vectoriel et graphique

RxJs	Programmation réactive en javascript (Observable , subscribe)
Vue / Vue-Js	Framework SPA (Single Page Application) gérant modèle MVVM (proche MVC) coté navigateur
React (facebook)	Framework SPA à base composants "javascript" (.jsx)
Angular (de google)	Framework SPA (concurrent de VueJs et React)

II - Bases du langage javascript (toutes versions)

1. Variables et types

1.1. Les types (implicites) de données

Il existe implicitement quatre grands types de données en JavaScript:
les nombres, les booléens, l'élément **null** (object) et les chaînes de caractères.

Les nombres (number):

Les nombres peuvent être des *entiers* (base 10, base 8, base 16), *des nombres à virgule*, ou des *nombres avec exposant*.

La valeur spéciale NaN signifie "Not a Number" .

Les booléens (boolean):

Les booléens peuvent avoir deux valeurs qui sont **true** pour vrai ou **false** pour faux.

L'élément null (de type "object"):

La valeur **null** représente la valeur *rien* (au sens pas d'objet / pas d'instance référencée).

Cette valeur est différente de 0 ou de chaîne vide ''.

Les chaînes de caractères (string):

Une chaîne peut contenir aucun ou plusieurs caractères délimités par des guillemets doubles ou simples.

Tableau des types de données :

number	Décimale (base 10)	0, 154, -17	Entier normal
	Octale (base 8)	035	Entier précédé d'un zéro
	Hexadécimale(base16)	0x4A, 0X4A	Entier précédé de 0x ou 0X
boolean	true (vrai) ou false (faux)		
null (object)	Mot clé qui représente la valeur nulle		
string	"JavaScript" '125'		
undefined	Variable déclarée mais jamais initialisée (considérée comme équivalent proche de null mais de type undefined)		
function	Variable référençant une fonction (ex : callback) .		

Remarque:

En JavaScript, il est possible de convertir des chaînes en entier ou en nombre à virgule via les fonctions suivantes:

La commande **parseInt("75")** ou bien **Number("75")** renvoie la valeur 75
et **parseFloat("41.56")** ou bien **Number("41.56")** renvoie la valeur 41.56 .

Number("123px") retourne NaN tandis que **parseInt("123px")** retourne 123 .

Il est possible d'insérer un nombre dans une chaîne (par simple **concaténation** via l'opérateur +):

"le cours va durer " + 5 + " jours" → "le cours va durer 5 jours"

La fonction prédéfinie **isNaN**(chExpr) renvoie true si chExpr n'est pas numérique

Remarque importante : une **variable non initialisée** est considérée comme "**undefined**" (notion proche de "null") et ne peut pas être utilisée en tant qu'objet préfixe .

1.2. Opérateurs typeof , == , === et tests/comparaisons

L'opérateur **typeof** *variable* retourne une chaîne de caractère de type "string" , "number" , "boolean" , "undefined" , selon le type du contenu de la variable à l'instant t .

```
var vv ;  
if( typeof vv == "undefined" ) {  
    console.log("la variable vv n'est pas initialisée") ;  
}  
  
if( vv == null ) {  
    console.log("la variable vv est soit null(e) soit non initialisée") ;  
}
```

L'opérateur **==** (d'origine c/c++/java) retourne true si les 2 expressions ont des valeurs à peu près équivalente (ex 25 est une valeur considérée équivalente à "25") .

L'opérateur **===** (spécifique à javascript) retourne true si les 2 expressions ont à la fois les mêmes valeurs et le même type ("25" et 25 ne sont pas de même type) .

2. Les variables (implicites ou explicites)

Le mot clef **var** permet d'explicitement déclarer une variable:

```
var v1;  
v1 = "Bonjour"  
v2 = "valeur" // v2 est implicitement une variable
```

3. Les opérateurs et expressions

3.1. Opérateurs arithmétiques:

Il est possible d'utiliser les opérateurs classiques : l'addition (+), la soustraction (-), la multiplication (*), la division (/) et le reste de la division entière: le modulo (%). On peut aussi effectuer une incrémentation (++), une décrémentation (--) et une négation unaire (-).

On peut utiliser les opérateurs d'incrément et de décrément de deux manières:

++A, --A: incrémente ou décrémente (d'abord) A d'une unité et renvoie le résultat
A++, A--: renvoie le résultat et incrémente ou décrémente (ensuite) A d'une unité

Exemples:

```
12 + 7 → 19  
12 % 5 → 2  
A = 3 ; B = ++A → B = 4 et A = 4 ; C = A++ → C = 4 et A = 5  
A = 3 ; B = --A → B = 2 et A = 2 ; C = A-- → C = 2 et A = 1
```

3.2. Les opérateurs d'attribution (affectation)

`a += b <==> a = a + b // idem pour autres opérations`

Exemples :

```
A=6, B=3  
A += B → A=9 , B=3  
A /= B → A=2 , B=3
```

3.3. Opérateurs logiques

Opérateur	Description
&&	"Et" logique, retourne la valeur true lorsque les deux opérandes ont pour valeur true sinon false
	"Ou" logique, retourne la valeur true lorsque l'un des opérandes a pour valeur true et false lorsque les deux opérandes ont pour valeur false .
!	"Non" logique, renvoie la valeur true si l'opérande a pour valeur false et inversement.

NB: JavaScript effectue une évaluation en court-circuit, permettant d'évaluer rapidement une

expression, avec les règles suivantes:

- false avec **&&** → prend toujours pour valeur **false**
- true avec **||** → prend toujours pour valeur **true**

exemple: `if(false && (a++ == 5))` ==> le `a++` n'est jamais exécuté

3.4. Les opérations de comparaison

Les opérateurs de comparaison peuvent comparer des chaînes et des nombres. De plus, ces opérateurs de comparaison sont des opérateurs binaires.

Opérateur	Description
<code>==</code>	Retourne la valeur true, si les opérandes sont égales
<code>!=</code>	Retourne la valeur true, si les opérandes sont différentes
<code><</code>	Retourne la valeur true, si l'opérande gauche est strictement inférieure à l'opérande droite.
<code><=</code>	Retourne la valeur true, si l'opérande gauche est inférieure ou égale à l'opérande droite.
<code>></code>	Retourne la valeur true, si l'opérande gauche est strictement supérieure à l'opérande droite.
<code>>=</code>	Retourne la valeur true, si l'opérande gauche est supérieure ou égale à l'opérande droite.

Attention: Ne pas confondre l'opérateur d'affectation (`=`) avec le test d'égalité (`==`) .

3.5. Opérateur conditionnel ternaire (`? :`)

Résultat = `(condition_a_evaluer) ? valeur_si_vrai : valeur_si_faux`

exemple: `alert((jour == "lundi") ? "Bonne semaine" : "on n'est pas lundi")`

3.6. Priorités entre les opérateurs

Priorité la plus forte

Parenthèses (`()`)
Multiplication, division, modulo (`*` / `%`)
Addition, Soustraction (`+` -)
Opérateurs d'égalité (`==` `!=`)
"Et" logique (`&&`)
"Ou" logique (`||`)
Opérateurs conditionnels (`? :`)
Opérateurs d'attribution (`=` `+=` `-=` `*=` `/=` `%=`)

Priorité la plus faible

Remarque: en cas de doute (trou de mémoire), il est fortement conseillé d'utiliser des parenthèses.

Tableau de caractères spéciaux (pour les chaînes):

Caractère	Description
\t	Tabulation
\n	Nouvelle ligne
\r	Retour chariot
\f	Saut de page
\b	Retour arrière

Structure de comparaison : if ... else

```
if(condition) simple_instruction_alors //; si else sur même ligne
else simple_instruction_sinon
```

Exemple:

```
if ( heure < 12 ) document.write("Good Morning")
else document.write("Good Afternoon");
```

Si on souhaite effectuer plusieurs instructions la syntaxe est la suivante:

```
if (condition)
{
    commande_1
    commande_n
}
```

4. Instruction switch/case

```
switch(variableNumerique)
{
case 1:
    commande1; commande2;
    break;
case 2:
case 3:
    commandeA; commandeB;
    break;
default:
    commandeX;
}
```

5. Les boucles

5.1. boucle "for"

```
for ( valeur_de_départ ; condition_pour_continuer ; incrémentation )
{
    bloc de commandes;
}
```

Exemple:

```
function tableau()
{
}
nom = new tableau //création d'un tableau vide

for ( i = 0 ; i < 4 ; i ++ )
{
  nom[i] = prompt ( "Donnez un nom", "" );
}

for ( i = 0 ; i < 4 ; i ++ ) alert(nom[i]);

for (i=10 ; i > 0; i--) ➔ décrémentation de 1
for (i=1 ; i < 115; i+=5 ) ➔ de cinq en cinq
```

5.2. boucle "for ... in "

La boucle **for....in** sert à parcourir automatiquement toutes les propriétés d'un objet (ou bien tous les éléments d'un tableau).

```
for ( indice in tableau )
{
  //commande(s)/instruction(s) sur tableau[indice];
}
```

NB: Les indices (ou clefs) de valeur(s) "undefined" sont automatiquement écartés dans la boucle for ... in mais pas dans la boucle for(i=0;i<tab.length;i++) .

5.3. boucle "while" (tant que)

```
while ( condition)
{
  //commandes_exécutées_tant_que_la_condition_est_vraie;
}
```

5.4. instructions "break" et "continue"

La commande **break** permet à tout moment d'interrompre complètement une boucle (**for** ou **while**) même si cette dernière ne s'est pas exécutée complètement.

Exemple:

```
for (i = 0 ; i < 10 ; i++)
{
  num=prompt("Donner un nombre", "" );
  if (num == "0") break;
}
```

➔ Si l'utilisateur saisit le nombre 0, on sort de la boucle.

La commande **continue** permet de passer à l'itération suivante dans une boucle **for** ou **while**. A la différence de la commande **break**, **continue** n'interrompt pas la boucle mais exécute la mise à jour de l'indice pour **for** et le **test** pour **while**.

Exemple:

```
for (i = 0 ; i < 10 ; i++)  
{  
  if (i == 5) continue;  
  num=prompt("Donner un nombre", "");  
}
```

➔ quand i sera égal à 5, on passera directement à l'itération suivante sans exécuter la commande **prompt**.

6. Fonction eval

La fonction **eval**(chExpr) permet d'interpréter l'instruction javascript qui est dans la chaîne chExpr.

Exemples:

```
var res = eval("3+2") // res vaudra 5  
var ch = eval("navigator.appName") // Netscape ou Internet Explorer
```

On peut ainsi écrire du code javascript qui sera interprété plus tard:

```
chExpr = "document.forms[" + nomFrm + "].reset()"  
eval (chExpr)
```

Remarque:

window.**setTimeout**(chExpr,n) permet d'interpréter l'expression chExpr en différé (n ms plus tard)

window.**setInterval**(chExpr,n) permet de lancer l'interprétation périodique de chExpr toutes les n ms;

7. Objets Math , String , Date , ...

7.1. L'objet Math

L'objet **Math** permet d'effectuer des opérations mathématiques évoluées:

Nom	Description
Propriétés:	
SQRT2	Racine carré de 2 ($\cong 1.414$)
SQR1_2	Racine carré de $\frac{1}{2}$ ($\cong 0.707$)
E	Constante d'Euler ($\cong 2.718$)
LN10	Logarithme naturel de 10 ($\cong 2.302$)
LN2	Logarithme naturel de 2 ($\cong 0.693$)
PI	Pi ($\cong 3.1415$)
Méthodes:	
acos()	Calcule l'arc cosinus en radians
asin()	Calcule l'arc sinus en radians
atan()	Calcule l'arc tangente en radians
cos()	Calcule le cosinus en radians
sin()	Calcule le sinus en radians
tan()	Calcule la tangente en radians
abs()	Calcule la valeur absolue d'un nombre
ceil()	Renvoie l'entier supérieur ou égal à un nombre
max()	Renvoie le plus grand de deux nombres
min()	Renvoie le plus petit de deux nombres
round()	Arrondit un nombre à l'entier le plus proche
random()	Renvoie un nombre aléatoire compris entre 0 et 1
exp()	Calcule e à la puissance d'un nombre
floor()	Renvoie l'entier inférieur ou égal à un nombre
log()	Calcule le logarithme naturel d'un nombre
pow()	Calcule la valeur d'un nombre à la puissance d'un autre
sqrt()	Calcule la racine carré d'un nombre

Exemples:

```
périmètre = Math.PI * 2 * rayon;
maxi = Math.max( 125, 158);
```

7.2. Opérations sur les chaînes de caractères (String)

Tout objet de type **String** comporte un ensemble de méthodes permettant d'effectuer les manipulations suivantes sur des chaînes de caractères:

Nom	Description
Propriété	
length	Donne le nombre de caractères d'une chaîne
Méthode:	
anchor()	Encadre la chaîne dans une balise <A>
link(url)	Reçoit une URL et place la chaîne dans une balise <A> pour créer un lien hypertexte
big()	Encadre la chaîne dans une balise html <BIG>
small()	Encadre la chaîne dans une balise html <SMALL>
bold()	Encadre la chaîne dans balise html
fixed()	Encadre la chaîne dans balise html <TT>
italics()	Encadre la chaîne dans balise html <I>
strike()	Encadre la chaîne dans balise html <STRIKE>
sub()	Encadre la chaîne dans balise html <SUB>
sup()	Encadre la chaîne dans balise html <SUP>
blink()	Encadre la chaîne dans balise html <BLINK>
fontcolor(couleur)	Encadre la chaîne dans balise html et
fontsize(taille)	Encadre la chaîne dans balise html et
indexOf()	Reçoit une chaîne et un éventuel index initial et renvoie l'index de l'occurrence de la chaîne située après l'index initial.
lastIndexOf()	Reçoit une chaîne et un éventuel index initial et renvoie l'index de la dernière occurrence de la chaîne.
toLowerCase()	Retourne une copie de la chaîne en minuscules
toUpperCase()	Retourne une copie de la chaîne en MAJUSCULES
substring (deb,apresDernier)	Reçoit deux arguments entiers et renvoie la chaîne qui commence au premier argument et finit au niveau du caractère situé avant le second argument.
charAt (pos)	Reçoit un index pour argument et renvoie le caractère situé à cet index.

Exemple:

```
ch = "debut" + "suite" + "fin"
```

```
var chaine = "ma petite chaine";
chaine = chaine.toUpperCase(); // ==> chaine vaut maintenant "MA PETITE CHAINE".
```

```
ch="abc"
c=ch.charAt(0) // ==> c vaut "a"
chDeb=ch.substring(0,2) // ==> chDeb vaut "ab"
if(ch.indexOf("bc")<0) alert("bc" non trouvée dans : " + ch )
```

7.3. L'objet Date

Les objets de type **Date** permettent de travailler sur les heures (heures, minutes, secondes) et bien entendu sur les dates (mois, jours, année).

Pour définir un objet date en JavaScript on peut utiliser plusieurs constructeurs :

```
date1 = new Date(); // date et heure courantes
date2 = new Date(année, mois, jour);
date3 = new Date(année, mois, jour, heures, minutes, secondes);
```

Principales méthodes:

- **getDate()** Retourne le jour du mois sous forme d'entier compris entre 1 et 31.
- **getDay()** Retourne le jour de la semaine sous forme d'entier (0 pour dimanche, 1 pour lundi, etc.).
- **getHours()** Retourne l'heure sous forme d'entier compris entre 0 et 23.
- **getMinutes()** Retourne les minutes sous forme d'entier compris entre 0 et 59.
- **getSeconds()** Retourne les secondes sous forme d'entier compris entre 0 et 59.
- **getMonth()** Retourne le mois sous forme d'entier compris entre 0 et 11 (0 pour janvier et 11 pour décembre).
- **getTime()** Retourne le nombre de secondes qui se sont écoulées depuis le 1 janvier 1970 à 00:00:00.
- **getTimezoneOffset()** Retourne la différence existante entre l'heure locale et l'heure GMT en minutes.
- **getFullYear()** Retourne l'année sous forme d'un entier à deux chiffres 97 pour 1997.
- **getFullYear()** Retourne l'année.
- **setDate(date)** Définit le jour du mois sous forme d'entier compris entre 1 et 31.
- **setHours(heures)** Définit l'heure sous forme d'entier compris entre 0 et 23.
- **setMinutes(minutes)** Définit les minutes sous forme d'entier compris entre 0 et 59.
- **setMonth(mois)** Définit le mois sous forme d'entier compris entre 0 et 11 (0 pour janvier et 11 pour décembre).
- **setSeconds(secondes)** Définit les secondes sous forme d'entier compris entre 0 et 59.
- **setTime(l'heure)** Définit l'heure sur la base du nombre de secondes écoulées depuis le 1 janvier 1970 à 00:00:00
- **setYear(année)** Définit l'année sur la base d'un entier de quatre chiffres >1990.
- **toGMTString()** Retourne la date et l'heure en cours suivant les conventions d'Internet ("Lun 10 Jan 1997 14:45:10 GMT")
- **toLocaleString()** Retourne la date sous la forme MM/JJ/AA HH:MM:SS.

Exemple:

```
Jour_1 = new Date(1997,01, 25)
j = Jour_1.getDate() → j= 25
```

8. Éléments divers du langage javascript

8.1. Fonction à nombre d'arguments variable

```
function fl()
{
  nb_arg=fl.arguments.length
  if(nb_arg > 0) { premier_arg=fl.arguments[0] ; alert(premier_arg) }
  if(nb_arg > 1) { deuxieme_arg=fl.arguments[1] ; alert(deuxieme_arg) }
}
```

fl(); *fl*('a1'); *fl*('a1','a2')

Attention : **arguments** n'est pas accessible lorsque javascript est utilisé en mode **strict** .

8.2. Conversion au format JSON :

var jsonString = **JSON.stringify**(jsObject) ;

var jsObject2 = **JSON.parse**(jsonString) ;

8.3. Opérateur "in"

propertyNameXy **in** *objectZzz* renvoie true ou false selon que l'objet *objectZzz* comporte ou pas la propriété *propertyNameXy* .

9. Nouveaux type d'objets (non prédéfinis) et tableaux

9.1. fonction faisant office de constructeur et mot clef this

Le langage JavaScript comporte un **mécanisme ultra-simple** pour **fabriquer de nouveaux type d'objet**:

Il suffit de créer une fonction qui servira à construire un nouvel objet. On désigne ce genre de fonction des "créateurs de **prototypes** d'objets"

Le mot clef **this** désigne l'objet (l'instance) courant(e).

Exemple:

```
function affVoiture()
{
  alert("marque= "+this.marque + ", modele= " + this.modele)
}

function Voiture(marque,modele)
{
  this.marque=marque // propriété 1
  this.modele=modele // propriété 2
  this.aff=affVoiture // méthode A (fonction attachée à une classe d'objet)
}
```

Création d'un ou plusieurs exemplaires de la classe Voiture:

```
v1 = new Voiture("Peugeot","306")
v2 = new Voiture("Renault","Mégane")
```

Utilisation des instances:

```
v1.modele = "206" // modification d'une propriété
v1.aff() // appel d'une méthode
```

Remarque: Il est possible d'ajouter dynamiquement une nouvelle propriété (ou des méthodes) au sein d'un objet déjà créé:

```
v1.couleur="rouge"
alert(v1.couleur)
```

Remarque très importante:

JavaScript gère les membres (propriétés ou méthodes) d'un objet sous la forme d'un tableau redimensionnable de choses quelconques (nombre,chaîne,objet,...).

Inversement , un tableau personnalisé doit être construit comme un objet.

Exemple1:

```
v1.marque <==> v1["marque"] <==> v1[0]
```

Exemple2:

```
function afficherToutesValeurs(obj)
{
  var ch=""
  for( i in obj)
  {
    m = "" + obj[i] //Remarque: "" + permet de convertir quelquechose en chaine
    if(m.indexOf("function")<0)
    {
      // ne tenir compte que des propriétés (en écartant les fonctions)
      ch += ( obj[i] + " " )
    }
  }
}
```

```
    }  
  }  
  alert( ch )  
}
```

afficherToutesValeurs(v1)

Exemple3 (Tableaux):

```
function tableau() // Array existe déjà  
{ // fonction pour construire un tableau vide  
}
```

```
function tableau_initialise() //Array() existe déjà et fait la même chose  
{  
  for(i=0;i<tableau_initialise.arguments.length;i++)  
    this[i]=tableau_initialise.arguments[i]  
}
```

```
var tab1 = new tableau() // ou new Array()  
tab1[0]="e1"  
tab1[1]="e2"  
afficherToutesValeurs(tab1)  
//var tab2 = new tableau_initialise("hiver","printemps","ete","automne")  
var tab2 = new Array("hiver","printemps","ete","automne")  
afficherToutesValeurs(tab2)
```

9.2. Array

Quelques opérations classiques prédéfinis sur les tableaux javascripts :

tab.push(elt) ; //ajoute à la fin
dernier_elt = tab.pop() ; //retourne et retire la valeur du dernier élément du tableau

delete tab[i] ; //supprime la valeur de tab[i] qui devient **undefined** .

tab.splice(i , 2 , val1 , val2) ; //remplace tab[i] par val1 et tab[i+1] par val2 , etc
tab.splice(i, 1) ; //remplace tab[i] par rien et donc supprime la case tab[i]
//sans trou , certains autres éléments sont déplacés (changement d'indice)

il existe aussi **.sort()** ,

III - DOM et gestion des événements

1. DOM ET DHTML

1.1. Présentation

DHTML (dynamic html) était un ancien **terme médiatique** qui indique simplement que certains éléments de la page peuvent être dynamiquement modifiés par des morceaux de scripts (JavaScript, ...). Aujourd'hui le terme "**DOM = Document Object Model**" désigne l'api (souvent utilisé en langage javascript) pour manipuler les différentes parties de la page HTML courante (ainsi que les styles css associés) .

Standard à suivre :

Le **W3C (World Wide Web Consortium)** a petit à petit élaboré un **modèle objet normalisé** (DOM de Niveaux 1,2 et 3) avec entre autres "**document.getElementById()**". Ce modèle objet normalisé s'appuie sur l'arborescence générique d'XML et sur les feuilles de style CSS.

2. Le modèle normalisé (DOM du W3C) (depuis IE5)

Le modèle objet normalisé par le W3C est complètement documenté au bout de l'URL suivante: <http://www.w3.org/DOM/>

HTML peut être vu comme un cas particulier de XML . On parle de **XHTML**.

Le **DOM niveau 1 (Core)** s'applique à **XML**

Le **DOM niveau 1 (Html)** s'applique à (X)**HTML**.

Le **DOM niveau 2** s'applique essentiellement aux **styles (CSS)** et aux **événements**.

2.1. DOM Niveau 1 (Core)

XML DOM est une **API normalisée** au niveau du W3C.

L'api XML DOM permet de **construire un arbre en mémoire**. Chaque noeud de l'arbre correspond à un élément XML quelconque (Balise, Zone Textuelle, Instruction de traitement, ...). Dans un arbre DOM, les noeuds ne sont pas tous du même type:

- Le noeud racine est de type **Document**
- Les noeuds liés aux balises sont de type **Element**
- Les noeuds comportant un morceau de texte sont de type **Text**

Cependant , ces différents types de noeuds héritent tous d'un type générique : "**Node**".

Node

.nodeName , nodeValue , nodeType
.appendChild() , childNodes, parentNode, ...

<u>Element</u>	<u>Text</u>	<u>Document</u>
<code>.getAttribute(...)</code>		<code>.documentElement</code>	
<code>.setAttribute(...,...)</code>		<code>.createElement(...), .createTextNode(...)</code>	

Certaines propriétés (associées au type générique *Node*) sont accessibles depuis n'importe quel noeud :

- **nodeName** retourne le nom d'une balise ou null .
- **nodeValue** retourne la valeur d'un texte ou null .
- **nodeType** retourne une constante indiquant le type de noeud (ELEMENT_NODE , TEXT_NODE,)
- **childNodes** retourne une liste de noeuds fils sous la forme d'un objet ensembliste de type **NodeList** .
- **parentNode** retourne une référence sur le noeud père

D'autres fonctions ne sont disponibles que sur certains types précis de noeuds:

- La fonction **documentElement()** que l'on appelle sur le noeud racine du document retourne **l'unique noeud de type Element qui correspond à la balise de premier niveau** .
- Seul le noeud racine (de type *Document*) comporte des fonctions [`createElement("nombalise")` , `createTextNode("valeurTexte")`] permettant de créer de nouveaux noeuds qui devront ultérieurement être accrochés sous un des noeuds de l'arbre via la méthode `appendChild()` .
- Les méthodes `setAttribute("nomAttribut","valeur")` et `getAttribute("nomAttribut")` doivent être appelées sur un noeud de type *Element* .

L'interface **Attr(ibute)** correspond à un attribut. *Les noeuds de type Attr(ibute) ne sont pas directement placés sous un noeud de type Element.*

Les différents attributs d'un noeuds sont accessibles depuis la collection attributes d'un élément.

L'interface **NodeList** représente un ensemble ordonné de noeuds.

Sa propriété **length** retourne le nombre d'éléments (pour boucle for de 0 à n-1).

Sa méthode **item(i)** retourne le i éme noeud de la liste.

Correspondance entre fichier XML et arbre "DOM" en mémoire :

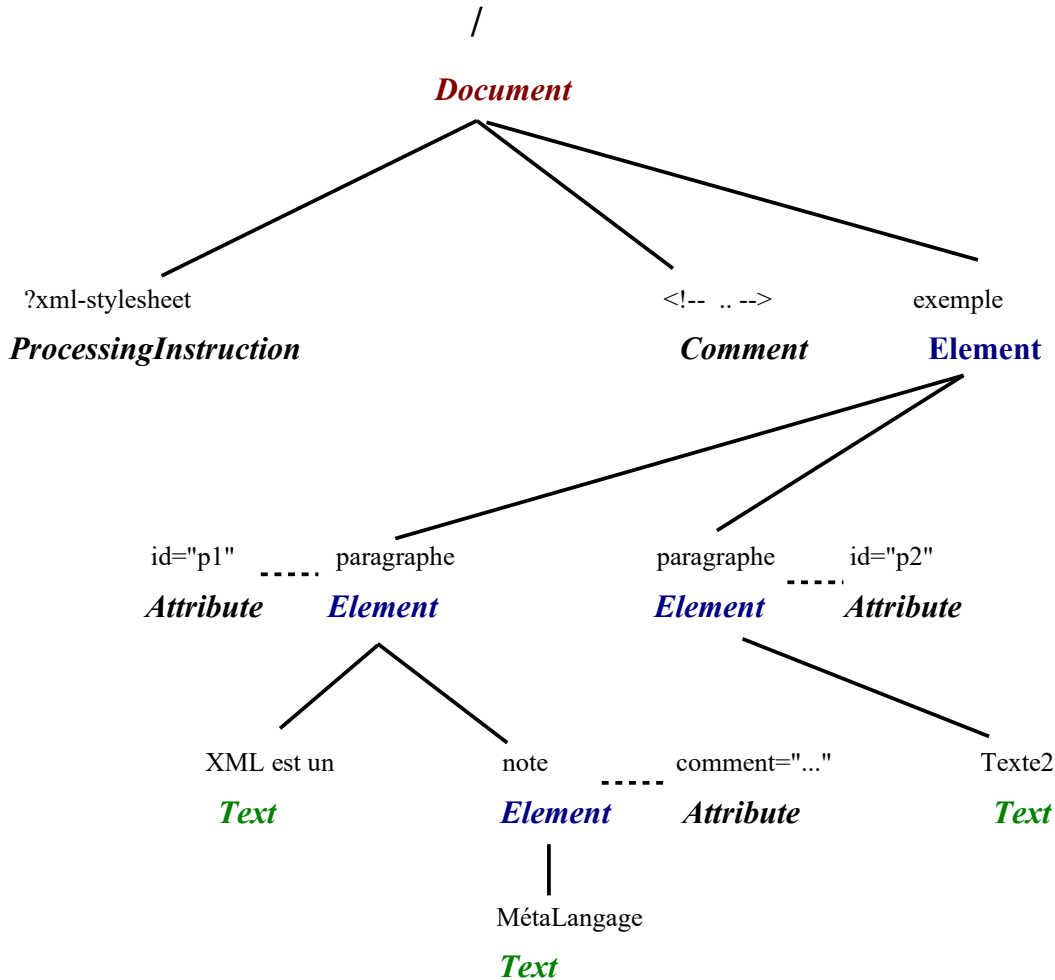
Le fichier xml suivant

```
<?xml version="1.0" ?>
<?xml-stylesheet href="/style1.css" type="text/css" ?>
<!-- commentaire -->
<exemple>
  <paragraphe id="p1">XML est un
    <note comment="important"> M&eacute;taLangage</note>
  </paragraphe>
```



```
<paragraphe id="p2">texte2</paragraphe>
</exemple>
```

est représenté via un arbre DOM de ce type:



2.2. DOM Niveau 1 (HTML)

Le DOM niveau 1 (HTML) est une extension du DOM (Core) prévue pour HTML. Une liste d'éléments sera souvent gérée au travers d'une collection :

```
interface HTMLCollection {
  readonly attribute unsigned long length;
  Node item(in unsigned long index);
  Node namedItem(in DOMString name);
};
```

Un **HTMLDocument** hérite des spécificités d'un **Document** et possède en plus les propriétés et méthodes suivantes:

```
interface HTMLDocument : Document {
  attribute DOMString title;
  readonly attribute DOMString referrer;
```

```
readonly attribute DOMString domain;  
readonly attribute DOMString URL;  
attribute HTMLElement body;  
readonly attribute HTMLCollection images;  
readonly attribute HTMLCollection applets;  
readonly attribute HTMLCollection links;  
readonly attribute HTMLCollection forms;  
readonly attribute HTMLCollection anchors;  
attribute DOMString cookie;  
void open();  
void close();  
void write(in DOMString text);  
void writeln(in DOMString text);  
Element getElementById(in DOMString elementId);  
NodeList getElementsByName(in DOMString elementName);  
};
```

La principale fonction est **getElementById()** . Celle-ci renvoie une référence sur un élément dont on connaît l'id .

Cette fonction est supportée depuis **IE5** et **Netscape 6** .

Un élément quelconque d'une page HTML sera de type **HTMLElement** (héritant de *Element* héritant de *Node*).

```
interface HTMLElement : Element {  
  attribute DOMString id;  
  attribute DOMString title;  
  attribute DOMString lang;  
  attribute DOMString dir;  
  attribute DOMString className;  
};
```

Les différentes interfaces suivantes correspondent à des éléments particuliers d'une page HTML:

HTMLHtmlElement racine de la page <html>

HTMLHeadElement partie <head>

HTMLLinkElement <link>

HTMLTitleElement <title>

HTMLMetaElement <meta>

HTMLStyleElement <style>

```
interface HTMLBodyElement : HTMLElement {  
  attribute DOMString aLink;  
  attribute DOMString background;  
  attribute DOMString bgColor;  
  attribute DOMString link;  
  attribute DOMString text;  
  attribute DOMString vLink;  
};
```

```
interface HTMLFormElement : HTMLElement {  
  readonly attribute HTMLCollection elements;  
  readonly attribute long length;  
  attribute DOMString name;  
  attribute DOMString acceptCharset;  
  attribute DOMString action;  
  attribute DOMString enctype;  
  attribute DOMString method;  
  attribute DOMString target;  
  void submit();  
  void reset();  
};  
HTMLSelectElement  
HTMLOptionElement
```

HTMLInputElement
HTMLTextAreaElement
HTMLButtonElement
 ...

Un tableau HTML est vu comme un élément du type suivant:

```

interface HTMLTableElement : HTMLElement {
...
readonly attribute HTMLCollection rows;
...
attribute DOMString bgColor;
attribute DOMString border;
...
HTMLElement insertRow(in long index);
void deleteRow(in long index);
};
  
```

Ligne d'un tableau:

```

interface HTMLTableRowElement : HTMLElement {
attribute long rowIndex;
...
attribute HTMLCollection cells;
...
HTMLElement insertCell(in long index);
void deleteCell(in long index);
};
  
```

Cellule d'un tableau: **HTMLTableCellElement**

2.3. Exemples (depuis IE5 et NS6)

```

document.getElementById("xxx").style.visibility = "hidden"
document.getElementById("xxx").style.color = "green"
document.getElementById("xxx").innerHTML = "yyy" // non normalisé.
document.getElementById("xxx").onmouseover= "...."
  
```

Equivalent de l'ancien document.all de IE4:

```
var docContents= document.getElementsByTagName("*");
```

Ajouter un élément dans la page:

```

var txtNode = document.createTextNode("blabla");
var link = document.createElement('a');
link.setAttribute('href','mypage.html');
link.appendChild(txt);
document.getElementById("xxx").appendChild(link);
  
```

Remplacement du contenu d'un élément (avec NS6):

*// équivalent de **document.getElementById(eltId).innerHTML=content***

```

function dynamicContentNS6(elementId,content)
{ if(document.getElementById)
{
rng=document.createRange();
el=document.getElementById(eltId);
  
```

```

rng.setStartBefore(e1);
htmlFrag = rng.createContextualFragment(content);
while(e1.hasChildNodes())
e1.removeChild(e1.lastChild());
e1.appendChild(htmlFrag);
}
}

```

2.4. Initialisation dès le chargement de la page

```

function initAfterLoad() {
...
}

window.addEventListener('load', initAfterLoad);

```

2.5. Traitement des événements (Normalisé par DOM Niveau 2):

==> ne fonctionne pas toujours avec d'anciennes versions de IE .

<http://www.w3.org/TR/DOM-Level-2-Events/events.html>

```

var e= document.getElementById("xxx");
e.addEventListener("mouseover",fctShowMsg,false);
// le booléen "capture" est rarement à true (exécution dès la phase de capture)
//il est souvent à false (valeur par défaut) (exécution durant la phase "bubbling" de remontée des
// événements des composants "enfants" vers les composants "parents" ).

```

// (annuler action par défaut).

e.removeEventListener(-,-,-);

La fonction de traitement est du type suivant:

```

function showHomeMsg(evt) // evt est de type W3C Event Object (DOM 2)
{ //....
}

```

// Introduced in DOM Level 2:

```

interface Event {
// PhaseType
const unsigned short CAPTURING_PHASE = 1;
const unsigned short AT_TARGET = 2;
const unsigned short BUBBLING_PHASE = 3;
readonly attribute DOMString type;
readonly attribute EventTarget target;
readonly attribute EventTarget currentTarget;
readonly attribute unsigned short eventPhase;

```

```
readonly attribute boolean bubbles;  
readonly attribute boolean cancelable;  
readonly attribute DOMTimeStamp timeStamp;  
void stopPropagation();  
void preventDefault();  
void initEvent(in DOMString eventTypeArg,  
in boolean canBubbleArg,  
in boolean cancelableArg);  
};
```

2.6. innerHTML

```
function affInDivA(msg){  
    var divA=document.getElementById('divA');  
    divA.innerHTML="message=<b>"+msg+"</b>";  
}
```

2.7. querySelector (depuis IE 8 , Fx 3.5 , ...)

De façon à utiliser une syntaxe proche de jQuery et des sélecteurs css , on peut utiliser `querySelector()` à la place de `getElementById()` :

```
//var myP=document.getElementById('myP');  
var myP=document.querySelector('#myP');
```

2.8. insertRow() , insertCell()

```
var newRow = tableElt.insertRow(-1 ou ...);  
var newCell = newRow.insertCell(0 ou ...);  
newCell.innerHTML="Paris ou autre";
```

C'est un équivalent plus rapide de `document.createElement("tr") ; + appendChild(...)`

IV - JSON , localStorage, ...

1. Format JSON

Format JSON (JSON = *JavaScript Object Notation*)

Les 2 principales caractéristiques de JSON sont :

- Le principe de clé / valeur (map)
- L'organisation des données sous forme de tableau

Les types de données valables sont :

- tableau
- objet
- chaîne de caractères
- valeur numérique (entier, double)
- booléen (true/false)
- null

```
[
  {
    "nom": "article a",
    "prix": 3.05,
    "disponible": false,
    "descriptif": "article1"
  },
  {
    "nom": "article b",
    "prix": 13.05,
    "disponible": true,
    "descriptif": null
  }
]
```

une liste d'articles

une personne

```
{
  "nom": "xxxx",
  "prenom": "yyyy",
  "age": 25
}
```

2. Manipulations JSON en javascript

```
var objDonneesJs = JSON.parse(string_donnees_json);
```

```
var jsonString = JSON.stringify(jsDataObject);
```

3. Suppression/remplacement d'attribut javascript

```
var obj = { prenom:"jean", nom:"Bon" , age:25 };
console.log(JSON.stringify(obj));
obj.name=obj.nom; //ajout de la propriété .name (par copie de la valeur de .nom)
console.log(JSON.stringify(obj));
```

```
delete obj.nom; //supression de la propriété .nom  
console.log(JSON.stringify(obj));
```

4. LocalStorage et SessionStorage

localStorage et **sessionStorage** sont des objets prédéfinis des navigateurs modernes qui permettent de **stocker et récupérer des informations depuis une ou plusieurs pages html** différentes si nécessaire.

```
localStorage.setItem("clefXy","valeur qui va bien") ;  
var stringValue= localStorage.getItem("clefXy") ;
```

NB : sessionStorage est lié à une session utilisateur et les informations qui y sont stockées sont généralement conservées sur une plus courte durée .

Selon le navigateur , les informations stockées en localStorage sont (ou pas) conservées suite à un redémarrage du navigateur.

Les informations stockées dans localStorage sont conservées suite à un "refresh" d'une page html .

NB : via JSON.stringify(jsObject) (et inversement JSON.parse(jsonString)) on peut transformer des objets "javascript" en chaîne de caractères au format JSON puis stocker ensuite cette chaîne en tant qu'item de localStorage ou sessionStorage .

V - Ajax (xhr , fetch , ...)

1. Appels de WS REST (HTTP) depuis js/ajax

Avec ajax , ça va briller!

1.1. Cadre des appels

Lorsqu'une requête http est initiée depuis du code javascript s'exécutant dans le contexte d'une page html , on dit que l'on effectue un appel "ajax" .

Le sigle Ajax correspondant à peu près à "asynchronous javascript activation framework" indique :

- le déclenchement non bloquant d'une requête http
- l'enregistrement d'une fonction "callback" qui sera automatiquement appelée en différé lorsque la réponse reviendra

NB : l'appel non bloquant peut être considéré comme asynchrone mais le protocole HTTP est un protocole de transport synchrone (avec timeout si la réponse ne revient pas dans un délai raisonnable).

Techniquement, un appel ajax s'effectue en s'appuyant sur un objet technique "XmlHttpRequest" fourni par tous les navigateurs pas trop anciens .

La partie Xml de XmlHttpRequest tient au fait qu'historiquement les premiers webServices normalisés (SOAP) étaient au format Xml. Bien que le terme "XmlHttpRequest" n'ait pas été changé pour des raisons de compatibilité ascendante du code javascript, il est possible de déclencher n'importe quelle requête HTTP depuis XHR , y compris des requêtes au format "JSON" .

Pour simplifier la syntaxe d'un appel ajax on peut éventuellement s'appuyer sur des bibliothèques "javascript" complémentaires ("jquery" , "fetch" , "RxJs" , ...) .

Le principe des appels "ajax" sert essentiellement à déclencher une requête HTTP suite à un événement utilisateur en vue d'obtenir des données permettant de réactualiser une partie de la page HTML courante . La page courante n'est pas entièrement remplacée par une autre. Seule une sous partie de celle ci est ajustée par code js/DOM (Document Object Model).

Certaines applications , dites "SPA: Single Page Application" sont entièrement bâties sur ce principe . Au lieu de switcher de pages html on switch de sous pages (<div ...>....</div>) .

1.2. XHR (XmlHttpRequest) dans tout navigateur récent

Exemple :

```
function makeAjaxRequest(callback) {
    var xhr = new XMLHttpRequest();

    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && (xhr.status == 200 || xhr.status == 0)) {
            callback(xhr.responseText);
        }
    };

    xhr.open("GET", "handlingData.php", true);
    xhr.send(null);
}

function readData(sData) {
    if (sData!=null) {
        alert("good response");
        // + display data with DOM
    } else {
        alert("empty response");
    }
}

makeAjaxRequest(readData);
```

variations en mode "POST" :

```
xhr.open("POST", "handlingData.php", true);
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

xhr.send("param1=xx&param2=yy");
```

```
xhr.open("POST", "/json-handler");
xhr.setRequestHeader("Content-Type", "application/json");
xhr.send(JSON.stringify({prenom:"Jean", nom:"Bon"}));
```

1.3. Appel ajax via jQuery

La syntaxe des appels "ajax" est un peu plus explicite en s'appuyant sur la bibliothèque "jquery".

Exemple :

```
<html>
<head>
  <meta charset="ISO-8859-1">
  <title>browse-spectacles</title>
  <script src="lib/jquery-3.3.1.min.js"></script>
  <script src="js/my-jq-ajax-util.js"></script>
</script>
$(function() {

  //appel ajax pour récupérer la liste des catégories et remplir le <select>
  $.ajax({
    type: "GET",
    url: "spectacle-api/public/spectacle/allCategories",
    contentType : "application/json",
    success: function (data,status,xhr) {
      if (data) {
        var categoryList = data;
        for(categoryIndex in categoryList){
          var category=categoryList[categoryIndex];
          $('#selectCategory').append('<option value="'+ category.id +"'>'+
            category.id + ' (' + category.title + ')</option>');
        }
        //$("#spanMsg").html(JSON.stringify(data));
      }
    },
    error: function( jqXHR, textStatus, errorThrown ){
      $("#spanMsg").html( xhrStatusToErrorMessage(jqXHR) );
    }
  }); //end $.ajax
});

</script>
</head>
<body>
  <h3> BROWSE Spectacles </h3>
  categorie : <select id="selectCategory"> </select><br/>
  ... <span id="spanMsg"></span> <br/>...
</body>
</html>
```

fonctions utilitaires dans [js/my-jq-ajax-util.js](#)

```
function setSecurityTokenForAjax(){

  var authToken = sessionStorage.getItem("authToken");
  //localStorage.getItem("authToken");

  $(document).ajaxSend(function(e, xhr, options) {
```

```

        //retransmission du jeton d'authentification dans l'entête http de la requete ajax
        xhr.setRequestHeader('Authorization','Bearer '+ authToken);
    });
}

function xhrStatusToErrorMessage(jqXHR){
    var errMsg = "ajax error";//by default
    var detailsMsg=""; //by default
    console.log("jqXHR.status="+jqXHR.status);
    switch(jqXHR.status){
        case 400 :
            errMsg = "Server understood the request, but request content was invalid.";
            if(jqXHR.responseText!=null)
                detailsMsg = jqXHR.responseText;
            break;
        case 401 :
            errMsg = "Unauthorized access (401)"; break;
        case 403 :
            errMsg = "Forbidden resource can't be accessed (403)"; break;
        case 404 :
            errMsg = "resource not found (404)"; break;
        case 500 :
            errMsg = "Internal server error (500)"; break;
        case 503 :
            errMsg = "Service unavailable (503)"; break;
    }
    return errMsg+" "+detailsMsg;
}

```

Variation en mode "POST" et "[application/x-www-form-urlencoded](#)"

```

var dataJsObject = { prenom : "jean", nom : "Bon", taille: 175 } ;
$.ajax({
    type: "POST",
    url: "./my-api/person",
    contentType : "application/x-www-form-urlencoded; charset=utf-8",
    data: $.param(dataJsObject),
    dataType : 'json_or_text_or_...',
    beforeSend: function (xhr) {
        xhr.setRequestHeader ("Authorization",
                                "Basic " + btoa("usernameXy" + ":" + "passwordXy"));
    },
    success: ... , error: ....
}); //end $.ajax

```

Variation en mode "POST" et "JSON" in/out :

```
//setSecurityTokenForAjax();//js/my-jq-ajax-util.js
$.ajax({
    type: "POST",
    url: "spectacle-api/spectacle",
    data : JSON.stringify(spectacleAdditionJsObject),
    dataType : "json",
    contentType : "application/json",
    success: function (data,status,xhr) {
        if (data) {
            $("#spanMsg").html(JSON.stringify(data));
        }
    },
    error: function( jqXHR, textStatus, errorThrown ){
        $("#spanMsg").html( xhrStatusToErrorMessage(jqXHR) );
    }
});//end $.ajax
```

1.4. Api fetch

Api récente (syntaxe concise basé sur enchaînement asynchrone et "**Promise**") mais pas encore supporté par tous les navigateurs.

Exemple :

```
fetch('./api/some.json')
    .then(
        function(response) {
            if (response.status !== 200) {
                console.log('Problem. Status Code: ' + response.status);
                return;
            }
            // Examine the text in the response :
            response.json().then(function(data) {
                console.log(data);
            });
        }
    )
    .catch(function(err) {
        console.log('Fetch Error :-S', err);
    });
```

1.5. Appel ajax via RxJs (api réactive)

Le framework "RxJs" lié au concept de "programmation asynchrone réactive" est assez sophistiqué et permet de déclencher une série de traitements d'une façon assez indépendante de la source de données (ex : données statiques , réponse ajax , push web-socket , ...) .

RxJs peut soit être directement utilisé en tant que bibliothèque javascript dans une page HTML , soit être utilisé via "typescript et le framework Angular 2,4,5,6 ou autre) .

Attention à la version utilisée (différences significatives dans la version récente de RxJs accompagnant Angular 6) .

--> une bonne présentation de RxJS est accessible au bout de l'URL suivante

<https://www.julienpradet.fr/tutoriels/introduction-a-rxjs/>

--> exemples d'appel ajax via RxJs et de config proxy dans le support de cours "Angular 4,5,6"

VI - Prototype

1. Prototype (javascript)

Depuis longtemps (es4, es5) , le langage javascript prend en charge la notion de **prototype** ayant une sémantique proche de "**default static** (*function, property,*)" pour un certain type d'objet (ex : String , ...).

1.1. Prototype (exemple)

```
function Client(name){
    this.name =name;
    this.direBonjour = methDireBonjour;
    this.showInternal = methShowInternal;
}

Client.prototype.address="1, rue elle"; //as static default address
Client.prototype.country="France"; //as static default country
//Client.prototype.liveIn = function(otherContry) { Client.prototype.country = otherContry; }
//change static default property
Client.prototype.liveIn = function(otherContry) { this.country = otherContry; } //change property

function methDireBonjour(){
    console.log("Bonjour, mon nom est "+this.name);
    console.log("my address is "+this.address);
    console.log("my country is "+this.country);
};
function methShowInternal(){
    console.log("this.constructor.toString()" + this.constructor.toString());
    console.log("this.constructor.prototype.address=" + this.constructor.prototype.address);
    console.log("this.constructor.prototype.country=" + this.constructor.prototype.country);
};

var c0=new Client();
c0.direBonjour();
//Bonjour, mon nom est undefined ,my address is 1, rue elle, my country is France

var c1 = new Client("c1"); c1.liveIn("USA");
c1.direBonjour();//Bonjour, mon nom est c1, my address is 1, rue elle, my country is USA
c1.showInternal();
//this.constructor.toString()=function Client(name){
//    this.name =name;
//    this.direBonjour = methDireBonjour;
//    this.showInternal = methShowInternal;
//}
//this.constructor.prototype.address=1, rue elle
//this.constructor.prototype.country=France
```

```
var c2 = new Client("c2"); c2.liveIn("UK");
c2.direBonjour();//Bonjour, mon nom est c2 , my address is 1, rue elle , my country is UK
```

1.2. Spseudo héritage via prototype

```
console.log("***** SPEUDO HERITAGE (via prototype es5) entre Dog et Animal *****");

function Animal(name){
    this.name=name;
}
Animal.prototype.color="black";//default color
Animal.prototype.weight=0;//default weight

var a1=new Animal("animal 1");
console.log("a1="+JSON.stringify(a1)); //a1={"name":"animal 1"}
console.log("Animal.prototype as jsonString="+JSON.stringify(Animal.prototype));
//Animal.prototype as jsonString={"color":"black","weight":0}
console.log("a1.color="+a1.color); //a1.color=black

//Expression d'un heritage entre Dog et Animal
Dog.prototype = Object.create(Animal.prototype, {
    constructor: { value: Dog,
        enumerable: false,
        writable: true,
        configurable: true }
});

//constructeur de Dog(...) :
function Dog(type,name){
    //NB: methodeXy.call(this, args) permet de préciser this en plus des arguments
    Client.call(this,name); //appel du constructeur de Client(...)
    this.type=type;//new attribute/property
}
Dog.prototype.height="40"; // Après HERITAGE !!!

var d1 = new Dog("berger allemand", "medor");
console.log("d1.type="+ d1.type);
console.log("d1.name="+ d1.name);
console.log("d1.height="+ d1.height);
console.log("d1.color="+ d1.color);
if(d1 instanceof Dog)
    console.log("d1 is a Dog");
if(d1 instanceof Animal)
    console.log("d1 is a Animal");
```

==>

```
d1.type=berger allemand
d1.name=medor
d1.height=40
```

dl.color=black
dl is a Dog
dl is a Animal

VII - Essentiel es2015 (arrow function, ...)

1. Mots clefs "let" et "const" (es2015)

Depuis longtemps (en javascript) , le mot clef "**var**" permet de déclarer explicitement une variable dont la portée dépend de l'endroit de sa déclaration (globale ou dans une fonction).

Sans aucune déclaration, une variable (affectée à la volée) est globale et cela risque d'engendrer des effets de bords (incontrôlés) .

Introduits depuis es6/es2015 et typescript 1.4 , les mots clefs **let** et **const** apportent de nouveaux comportements :

- Une variable déclarée via le mot clef **let** a une *portée limitée au bloc local* (exemple boucle for) . Il n'y a alors pas de collision avec une éventuelle autre variable de même nom déclarée quelques ligne au dessus du bloc d'instructions (entre {} , de la boucle).
- Une variable déclarée via le mot clef **const** *ne peut plus changer de valeur après la première affectation*. Il s'agit d'une **constante** .

Exemple :

```
const PISur2 = Math.PI / 2;
//PISur2=2; // Error, can't assign to a `const`
console.log("PISur2 = " + PISur2);

var tableau = new Array();
tableau[0] = "abc";
tableau[1] = "def";

var i = 5;
var j = 5;

//for(let i in tableau) {
for(let i=0; i<tableau.length; i++) {
    console.log("*** at index " + i + " value = " + tableau[i] );
}

//for(j=0; j<tableau.length; j++) {
for(var j=0; j<tableau.length; j++) {
    console.log("### at index " + j + " value = " + tableau[j] );
}

console.log("i=" + i); //affiche i=5
console.log("j=" + j); //affiche j=2
```

2. "Arrow function" et "lexical this" (es2015)

Rappels (2 syntaxes "javascript" ordinaires) valables en "javascript/es5" :

```
//Named function:
function add(x, y) {
    return x+y;
}

//Anonymous function:
var myAdd = function(x, y) { return x+y; };
```

Arrow functions (es2015) (alias "Lambda expressions")

Une "**Arrow function**" en javascript/es2015 (à peu près équivalent à une "lambda expression" de java >8) est syntaxiquement introduite via **() => { }**

Il s'agit d'une syntaxe épurée/simplifiée d'une fonction anonyme où les parenthèses englobent d'éventuels paramètres et les accolades englobent le code.

Subtilité du "lexical this" :

La valeur du mot clef "this" est habituellement évaluée lors de l'invocation d'une fonction .
Dans le cas d'une "lambda expression" , le mot clef this est évalué dès la création de la fonction et correspond au this du niveau englobant (classe ou fonction).

Exemples de "lambda expressions" :

```
myFct = (tab) => { var taille = tab.length; return taille; }
//ou plus simplement:
myFct = (tab) => { return tab.length; }
//ou encore plus simplement:
myFct = (tab) => tab.length;
//ou encore plus simplement:
myFct = tab => tab.length;
```

```
var numRes = myFct([12,58,69]);
console.log("numRes=" + numRes); //affiche 3
```

```
myFct2 = (x,y) => { return (x+y) / 2; } //with statement body in { }
//ou plus simplement:
myFct2 = (x,y) => (x+y) / 2; //with simple expression
```

NB : les "**Promise**" (es2015) et la technologie "**RxJs**" utilisée par angular>=2 utilisent beaucoup de "Arrow Functions" .

Exemple de "arrow function" combiné ici avec `arrayXy.forEach(callback)` de "javascript/es5" :

```
let array1 = [ 1 , 2 , 3 , 4 , 5 , 6 ];
let eltPairs = [];
array1.forEach( (e) => { if( (e % 2) === 0 )
                        eltPairs.push(e);
                      }
                );
console.log(eltPairs); // affiche [2,4,6]
```

Suite de l'exemple utilisant `nouveauTableau = arrayXy.map(transform_callback)` de es5 :

```
let eltImpairs = eltPairs.map( v => v-1 );
console.log(eltImpairs); // affiche [1,3,5]
```

Exemple montrant "lexical this" (arrow function utilisant `this` de niveau englobant) :

```
var toto = {
  _name: "toto",
  _friends: [ 'titi' , 'tata'],
  printFriends() {
    this._friends.forEach(f =>
      console.log(this._name + " est ami avec " + f));
  }
};
toto.printFriends();
```

Autre comportement à connaître:

Si une "fonction fléchée" / "arrow fonction" est à l'intérieur d'une autre fonction, elle partage alors les arguments/paramètres de la fonction parente .

3. for...of (es2015) utilisant itérateurs internes

```
var tableau = new Array();
```

```
//tableau.push("abc");
```

```
//tableau.push("def");
```

```
tableau[0] = "abc";
tableau[1] = "def";
```

Au moins 3 parcours possibles via boucle **for**:

```
var n = tableau.length;
for(let i = 0; i < n; i++) {
  console.log(">> at index " + i + " value = " + tableau[i] );
}
```

```
for(let i in tableau) {
  console.log("** at index " + i + " value = " + tableau[i] );
}
```

//for(index in ...) existait déjà en es5

//for(...of ...) au sens "for each ... of ..." est une nouveauté de es2015

```
for( let s of tableau){
  console.log("## val = " + s );
}
```

NB : la boucle for...of est prédéfinie sur un tableau . il est cependant possible de personnaliser son comportement si l'on souhaite la déclencher sur une structure de données personnalisée. On peut pour cela mettre en oeuvre des itérateurs (et éventuels générateurs de bas niveaux) ---> dans chapitre ou annexe "éléments divers et avancés de es2015" .

3.1. "template string" es2015 (avec quotes inverses et \${})

```
var name = "toto";
var year=2015;
// ES5
//var message = "Hello " + name + ", happy " + year; // Hello toto , happy 2015
// ES6/ES2015 :
const message = `Hello ${name} , happy ${year}`; // Hello toto , happy 2015
//attention: exception "ReferenceError: name is not defined" si name est undefined
console.log(message);
```

\${} peut éventuellement englober des expressions mathématiques ou bien des appels de fonctions.

```
let x=5 , y=6;
let carre = (x) => x*x ;
console.log(`pour x=${x} et y=${y} , x*y=${x*y} et x*x=${carre(x)}`);
//affiche pour x=5 et y=6 , x*y=30 et x*x=25
```

template-string multi-lignes :

```
/*
//ES5
let htmlPart=
"<select> \
  <option>1</option> \
  <option>2</option> \
</select>";
*/
```

```
//template multi-lignes ES2015:
let htmlPart=
`<select>
  <option>1</option>
  <option>2</option>
</select> `;
console.log(htmlPart);
```

3.2. Map , Set

```
// Sets (ensembles sans doublon)
var s = new Set();
s.add("hello").add("goodbye").add("hello");
if(s.size === 2)
  console.log("s comporte 2 elements");
if(s.has("hello"))
  console.log("s comporte hello");
```

// List ==> Array ordinaire (déjà en es5 , à remplir via .push()).

```
// Maps (table d'association (clef,valeur))
var m = new Map();
m.set("hiver", "froid , neige");
m.set("printemps", "fleur , vert");
m.set("ete", "soleil , plage");
m.set("ete", "chaud , plage"); //la nouvelle valeur remplace l'ancienne .
m.set("automne", "feuilles mortes");
let carateristique_ete = m.get("ete");
console.log("carateristique_ete="+carateristique_ete); //chaud , plage
if(m.has("ete"))
    console.log("Map m comporte une valeur associée à ete");
for(saison of m.keys()){
    console.log("saison "+ saison + " - " + m.get(saison));
}
//m.values() permettrait d'effectuer une boucle sur les valeurs (peu importe les clefs)
for([k,v] of m.entries()){
    console.log("saison "+ k + " -- " + v);
}
m.forEach((val,key)=> console.log("saison "+ key + " --- " + val));
m.clear();
if(m.size===0)
    console.log("map m is empty");

//Bien que ce code soit lisible et explicite, un vieil objet javascript en faisait autant :
var objectMap = {
    hiver : "froid , neige",
    printemps : "fleur , vert",
};
objectMap["ete"]="chaud, plage";
console.log("carateristique_hiver="+ objectMap["hiver"]); // froid , neige

//Une des valeurs ajoutées par "Map" (es2015) est la possibilité d'avoir des clefs de n'importe
//quelle sorte possible (ex : window , document , element_arbre_DOM, ...).
```

NB : es2015 a également introduit les variantes "WeakMap" et "WeakSet" mais celles-ci ne sont utilisables et utiles que dans des cas très pointus (ex : programmation de "cache"). "WeakMap" et "WeakSet" sont exposés dans le chapitre (ou annexe) "Aspects divers et avancés" .

3.3. "Destructuring" (affectation multiple avec perte de structure)

Destructuring objet : extract object parts in several variables :

```
const p = { nom : 'Allemagne' , capitale : 'Berlin' , population : 83000000, superficie : 357386};
const { nom , capitale } = p;
console.log("nom="+nom+" capitale="+capitale);
//nom="?" ; interdit car nom et capitale sont considérées comme des variables "const"

//NB: les noms "population" et "superficie" doivent correspondre à des propriétés de l'objet
//dont il faut (partiellement) extraire certaines valeurs (sinon "undefined")
//l'ordre n'est pas important
const { superficie , population } = p;
console.log("population="+population+" superficie="+superficie);
==>
```

nom=Allemagne capitale=Berlin
population=83000001 superficie=357386

utilité concrète (parmi d'autres) : *fonction avec paramètres nommés* :

```
function fxabc_with_named_param( { paramX=0 , a=0 , b=0 , c=0 } = {} ){
    //return ax^2+bx+c
    return a * Math.pow(paramX,2) + b * paramX + c;
}

let troisFois4 = fxabc_with_named_param( { paramX :4 , b : 3 } );
console.log("troisFois4="+troisFois4 );//12
let deuxFois4AuCarreplus6 = fxabc_with_named_param( { paramX :4 , a : 2 , c :6 } );
console.log("deuxFois4AuCarreplus6="+deuxFois4AuCarreplus6 );//38
```

Destructuring iterable (array or ...) :

```
const [ id , label ] = [ 123 , "abc" ];
console.log("id="+id+" label="+label);

//const arrayIterable = [ 123 , "abc" ];
//var iterable1 = arrayIterable;
const stringIterable = "XYZ";
var iterable1 = stringIterable;
const [ partie1 , partie2 ] = iterable1;
console.log("partie1="+partie1+" partie2="+partie2);
```

==>
id=123 label=abc
partie1=X partie2=Y



Autre exemple plus artistique (Picasso) :

3.4. for (..of ..) with destructuring on Array , Map, ...

```
const dayArray = ['lundi', 'mardi', 'mercredi'];
for (const entry of dayArray.entries()) {
  console.log(entry);
}
// [ 0, 'lundi' ]
// [ 1, 'mardi' ]
// [ 2, 'mercredi' ]

for (const [index, element] of dayArray.entries()) {
  console.log(`${index}. ${element}`);
}
// 0. lundi
// 1. mardi
// 2. mardi
```

```
const mapBoolNoYes = new Map([
  [false, 'no'],
  [true, 'yes'],
]);
for (const [key, value] of mapBoolNoYes) {
  console.log(`${key} => ${value}`);
}
// false => no
// true => yes
```


ANNEXES

VIII - Annexe – navigator , window , document (js)

1. document , form , events , cookies (js)

1.1. Vue d'ensemble sur les objets d'un document HTML

document

forms

elements (text fields, textarea, checkbox, password ,radio, select, button, submit, reset)

links

anchors

1.2. Les événements (html)

Un **événement** est un signal automatiquement généré par suite d'une action spécifique de l'utilisateur sur une certaine partie du document. Le langage JavaScript permet de spécifier des scripts (fonctions) qui seront alors automatiquement déclenché(e)s pour réagir d'une façon ou d'une autre.

Pour spécifier que la fonction f1 sera déclenchée quand un utilisateur appuiera sur un bouton, on utilise la syntaxe suivante:

```
<input type="button" value="B1" onclick="f1()" > <!-- ou f1(this.form) -->
```

événement se produit quand ...

onblur une zone perd la main sur les entrées clavier (perte du focus).

onchange le texte d'une zone a changé ou nouvelle sélection dans une liste

onclick on clique sur un bouton ou sur un élément qui gère le click .

onfocus une zone prend la main sur les entrées clavier (focus).

onload le navigateur a fini de charger une page (dans un onglet du navigateur). L'événement onload se positionne dans la balise body.

onmouseover la souris passe (survole) sur une zone.

onsubmit un formulaire est soumis au serveur par l'appui du bouton Submit.

onunload on quitte un document (symétrique de onLoad)

onerror le chargement d'une page ou d'une image produit une erreur.

onmouseout la souris quitte une zone.

onreset on clique sur le bouton reset d'un formulaire.

Nb: les événements sont liés à certains types d'élément html (certains éléments sont à l'origine d'un nombre très limité d'événement(s)).

Exemple:

```
<body onload="alert('Bienvenue')" onunload="alert('Au revoir..')">
```

1.3. L'objet document (html)

Principales Propriétés (autres que celles qui seront développées ultérieurement):

document.body.style.backgroundColor (*anciennement bgColor*) Couleur de l'arrière plan définie par le triplet hexadécimal RGB

document.body.style.color (*anciennement fgColor*) Couleur du premier plan définie par le triplet hexadécimal RGB.

forms Matrice d'objets correspondant à chaque formulaire contenu dans le document. forms.length permet d'obtenir le nombre de formulaire

lastModified Contient la date à laquelle le document a été modifié pour la dernière fois

links Matrice d'objets correspondant à chaque lien dans un document. links.length permet d'obtenir le nombre de liens dans un document

title Contient le titre du document

1.4. L'objet FORM

Principales propriétés:

action URL du script serveur ou boîte aux lettres (si mailto:).

elements tableau des éléments du formulaire (Zones de saisie,)

name nom du formulaire

encoding type MIME utilisé pour coder les données de formulaire soumises au serveur, correspond au paramètre ENCTYPE de la balise FORM

method méthode à utiliser pour dialogue HTTP (GET ou POST)

target Nom de fenêtre dans laquelle doit s'afficher le résultat renvoyé par le serveur

Nb: La méthode **submit()** déclenche l'envoi du formulaire au serveur.

1.5. Vérification (contrôle) des saisies

NB : Dès que l'on clique sur le bouton Submit, les différentes données saisies dans le formulaire associé sont alors automatiquement envoyées vers le script serveur identifié par l'attribut action (ex : page php , asp , jsp , servlet java , script cgi ,).

Etapes :

1. L'utilisateur **rempli un formulaire HTML** et click sur "**Submit**".
2. le navigateur crée et envoie la requête HTTP (en mode post ou get)
3. Lancement d'un code coté serveur (identifié par l 'URL précisée par l'attribut **action** de <form>) qui récupère une copie des valeurs saisies.
4. **Traitement des données reçues coté serveur** (recherche,archivage, ...)
5. Création (coté serveur php/jsp/asp/cgi) d'une **page de réponse HTML** et envoi de celle-ci en tant que réponse HTTP.

6. Dès que le navigateur reçoit la réponse , l'ancienne page (avec formulaire de saisie) est remplacée par la page de réponse fabriquée par le serveur.

Pour effectuer un **contrôle de saisie en JavaScript** avant les étapes 2 à 5, il faut traiter l'événement **onsubmit** de la façon suivante:

```
<form method="post" action="cgi-bin/scriptX" onsubmit="return verifForm(this)" >
```

Attention: le **return** est indispensable !!!

La fonction **verifForm** (que l'on doit écrire) doit effectuer des contrôles de saisie sur chacun des champs du formulaire et doit **retourner une valeur booléenne** qui sera interprétée comme suit:

- **true** (indiquant que les données sont correctement saisies) va permettre l'action par défaut (envoi des données au script serveur).
- **false** (indiquant qu'une des données est mal saisie) va annuler l'envoi des données au serveur.

Exemple de sous fonctions utilitaires (dans [verif.js](#)) pour vérifier les saisies:

```
function verifNum(chExpr,zone)
{
  var res=true ; //par défaut
  /* var i, c // ancienne Solution 1
  if(chExpr.length==0) res=false
  for(i=0;i<chExpr.length;i++)
  {
    c=chExpr.charAt(i)
    if( c < '0' || c > '9') res=false
  } */
  if(isNaN(chExpr)) res=false // solution 2 (mieux)

  if(res==false)
    if(zone != null)
    {
      alert("erreur saisie, " + zone.name + " non numérique");
      zone.select(); zone.focus()
    } else alert("erreur saisie, zone non numérique");
  return res
}
```

```
function verifEntre(val,v1,v2)
{
  if(val < v1 || val > v2)
  {
    alert(val + " n'est pas compris entre " + v1 + " et " + v2 )
    return false
  }
  /*else*/ return true
}
```

...

```
<script src="verif.js">
```

```

</script>
...
<script>
function verifForm(frm)
{ if(!verifNum(frm.age.value,frm.age)) return false
  if(!verifEntre(frm.age.value,0,120)) return false
  if(!verifNum(frm.dep.value,frm.dep)) return false
  if(!verifEntre(frm.dep.value,1,95)) return false
  /*else*/ return true
}
....

```

1.6. Les éléments d'un formulaire (INPUT,)

1.6.a. Propriétés communes (en général)

En général, les différentes zones d'un formulaire partagent les principales **propriétés** suivantes:

name nom du champ (de la zone)

value valeur de la zone

1.6.b. Méthodes générales:

Action des principales méthodes:

- **focus()** donne le focus sur un champ.
- **blur()** enlève le focus de l'objet.
- **select()** sélectionne le contenu d'un objet.
- **click()** émule un clic sur un bouton (bouton poussoir, cache à cocher, radio, ...).

1.6.c. Spécificité des éléments text et password

defaultValue valeur par défaut du champ (propriété supporté aussi par textarea)

size Indique la taille de la zone de saisie (sans scrolling)

1.6.d. Spécificités des cases à cocher (checkbox)

checked booléen indiquant l'état (coché , décoché)

defaultChecked état par défaut

1.6.e. Spécificités des boutons "radio"

Un bouton radio reprend toutes les propriétés d'une case à cocher, et supporte les propriétés supplémentaires suivantes:

length nombre de boutons radio dans le groupe (identifié par name commun)

index index du bouton radio sélectionné

1.6.f. Spécificités des zones de liste (select)

Les principales propriétés spécifiques à l'élément **select** sont les suivantes:

length nombre d'options dans l'objet SELECT
multiple (booléen) permet de sélectionner plusieurs éléments
size hauteur du champ select (si 1 ==> liste déroulante)
selectedIndex index de l'option sélectionnée
options liste des entrées dans la zone de liste

L'attribut **options** comporte les **propriétés** suivantes:

defaultSelected Booléen indiquant si l'option est sélectionnée par défaut.

name Attribut donnant un nom à l'option.

selected Booléen indiquant si l'option est sélectionnée.

text Contient la valeur de texte affiché dans le menu déroulant.

value Indique la valeur de l'élément de la liste.

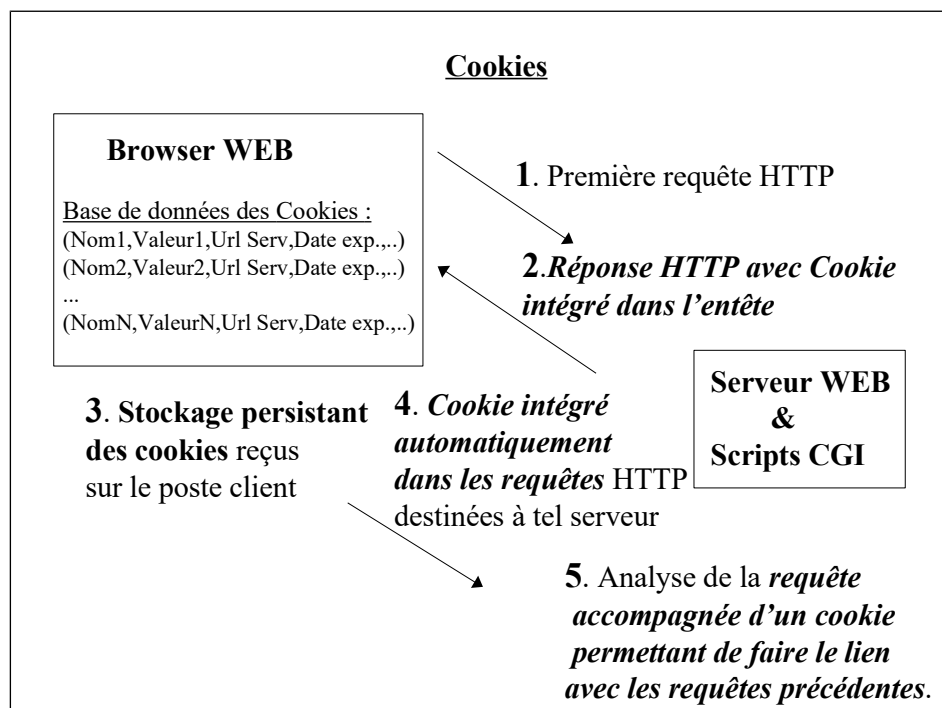
1.7. gestion des cookies (coté navigateur) en JavaScript

Preliminaire:

escape(chaine) encode une chaîne sous une forme indépendante de la plate-forme et standard vis à vis de HTTP (caractères étendus convertis en %xx, où xx est le code ASCII).

unescape(chaine) décode une chaîne de caractères.

1.7.a. principes des cookies



Un cookie est une association (**Nom ,Valeur, Url site serveur, Date d'expiration**)

Les cookies sont générés et relus (interprétés) par les scripts CGI (ou pages jsp/php/asp).

Dès qu'un browser reçoit un cookie (stocké dans l'entête d'une réponse HTTP) , **il stocke celui-ci de façon permanente dans une base de données locale.**

Lorsque le navigateur Web va émettre une nouvelle requête vers un site Web déjà consulté, les

cookies correspondants sont alors automatiquement intégrés dans les requêtes HTTP.

Le script CGI ainsi déclenché peut ainsi analyser les cookies pour y récupérer les paramètres anciennement choisis.

Le grand avantage d'un cookie est le fait d'être persistant.

Un script CGI (ou servlet java ou page php, asp, jsp) peut ainsi récupérer la valeur d'un paramètre dont la valeur a été défini plusieurs jours auparavant.

Applications possibles:

- Gestion des contextes et des sessions (lien entre requêtes successives)
- Mémorisation des préférences d'un utilisateur

1.7.b. Mise en oeuvre des cookies (Détails)

Une réponse HTTP peut éventuellement contenir (dans l'entête) un cookie formaté de la façon suivante:

Set-Cookie: NomCookie=Valeur; <i>path</i> =/; <i>expires</i> =Wednesday, 09-Nov-99 23:12:40 GMT

Lorsque que client ré-émettra une requête vers le même serveur, la liste de cookie suivante sera alors automatiquement intégrée dans celle-ci:

Cookie: NomCookie1=Valeur1; NomCookie2=Valeur2

Le script CGI ainsi déclenché pourra alors récupérer cette liste via la variable d'environnement **HTTP_COOKIE** .

...

NB: La **date d'expiration** (au format WeekDay, DD-Month-YY HH:MM:SS GMT) est très importante, elle régit les règles suivantes:

- Le cookie sera effacé au niveau du poste client lorsque cette date sera dépassée
- Si aucune date d'expiration n'est précisée par le serveur (champ expires optionnel), le cookie sera éphémère (non stocké sur le disque et effacé en fin de session)
- Si un script CGI veut effacer un cookie au niveau d'un client, il doit envoyé de nouveau celui-ci (avec les mêmes noms,valeurs) mais avec une date d'expiration aujourd'hui dépassée.

1.7.c. Manipulation d'un cookie en javascript (coté client)

NB: Certains navigateurs imposent un téléchargement des pages html via http pour que le Tp sur les cookies puisse fonctionner.

document.cookie est une **propriété** qui:

- en lecture, correspond à la liste des cookies associés au site http courant.
- en écriture, correspond à un seul cookie (avec ses différents paramètres).

Généralement , on gère les cookies via des fonctions utilitaires (**GetCookie** , **SetCookie**) que l'on programme une fois pour toute comme suit :

```

function SetCookie(nom,valeur)
{ // arguments optionnels: expires,path,domain,secure
var argc=SetCookie.arguments.length
var argv=SetCookie.arguments
var expires=(argc > 2) ? argv[2] : null
var path=(argc > 3) ? argv[3] : null
var domain=(argc > 4) ? argv[4] : null
var secure=(argc > 5) ? argv[5] : null
var ch = nom + "=" + escape(valeur)
if(expires != null) ch = ch + "; expires=" + expires.toGMTString()
if(path != null) ch = ch + "; path=" + path
if(domain != null) ch = ch + "; domain=" + domain
if(secure == true) ch = ch + "; secure"
document.cookie=ch
}

```

```

function GetCookie(nom)
{
var res=""
var prop=nom+"="
var allCookies=document.cookie
var longTotale=allCookies.length
var longProp=prop.length
var posProp=0
var posPtVirgule=0
if(longTotale>0) { // cookie non vide
posProp=allCookies.indexOf(prop,0)
if(posProp != -1) { // Propriété trouvée
posPtVirgule=allCookies.indexOf(";",posProp+longProp)
if(posPtVirgule != -1)
res=unescape(allCookies.substring(posProp+longProp,posPtVirgule))
else // pas de ; ==> dernière valeur de la liste
res=unescape(allCookies.substring(posProp+longProp,longTotale))
}
}
return res
}

```

Exemple:

```

...
<script>
var dateExp = new Date(2020,12,31)
</script>
</head>
<body bgcolor="#FFFFFF"
onload="document.form1.txtCouleur.value=GetCookie('CoulPref');
    document.bgColor=document.form1.txtCouleur.value">
<form method="POST" name="form1">
<p>Couleur préférée: <input type="text" size="20" name="txtCouleur"

```



```
onblur="SetCookie('CoulPref',this.value,dateExp);document.bgColor=this.value"></p>
</form>
</body>
</html>
```

1.8. Traitement des erreurs (javascript)

```
function afferror(msg, url , line_number )
{
var txt="Erreur: "+msg + " at line number "+line_number;
//txt = txt + " ,url="+url;
window.status = txt;
console.log(txt) ;
return true; // pour dire que l'on a traiter nous même l'erreur (pas trait. par défaut).
}
window.onerror=afferror;
```

2. objets navigator , window, ... (js)

2.1. Vue d'ensemble sur les objets du navigateur

navigator
window
 location
 history
 document (DOM)

2.2. L'objet navigator

L'objet **navigator** reflète les informations (pas toujours précises) sur la version de Navigateur utilisé. Il vaut mieux ne pas s'appuyer dessus!! (en grande partie "obsolète" ou bien "pas standard").

2.3. L'objet window

L'objet **window** est l'objet de niveau le plus élevé pour chaque fenêtre de premier niveau du navigateur. Il est l'objet parent des objets document, location, history.

Propriétés:

name Nom de la fenêtre ou du cadre

status Servait à afficher un message dans la barre d'état en attribuant des valeurs à cette propriété

Méthodes:

alert(message) affiche *message* dans une boîte de dialogue

close() ferme la fenêtre

confirm(message) affiche *message* dans une boîte de dialogue avec les boutons **OK** et **CANCEL**.
Retourne **true** si on clique sur **OK** sinon **false**

open(url,nom,fonction) ouvre l'*url* d'une page dans une fenêtre nommée **nom**. Si le **nom** n'existe pas, une nouvelle fenêtre est créée avec ce **nom**.

prompt(msg,default) affiche le **message** dans une boîte de dialogue comportant une zone de saisie qui prend la valeur par défaut indiquée. La réponse est renvoyée sous forme de chaîne.

setTimeout(expr,delay) évalue l'expression de façon asynchrone et en différée dans le temps après un certain délai exprimé en ms

clearTimeout(id) annule le déclenchement de `id=SetTimeout(...)`

Exemples:

`window.alert("Affichage d'un message")`

`netscapeWin=window.open("http://www.netscape.com", "netscapeHomePage")`

NB: la fonction **open** comporte un 3ème paramètre (fonction) facultatif sous forme d'une chaîne contenant les caractéristiques pour la nouvelle fenêtre.

Les caractéristiques peut contenir les couples suivants, **séparés par des virgules** et ne comportant pas d'espace :

toolbar=[yes,no,1,0] Indique si la fenêtre doit comporter une barre d'outils

location=[yes,no,1,0] Indique si la fenêtre doit comporter un champ d'URL

status=[yes,no,1,0] Indique si la fenêtre doit comporter une barre d'état

menubar=[yes,no,1,0] Indique si la fenêtre doit comporter des menus

scrollbars=[yes,no,1,0] Indique si la fenêtre doit comporter des barres de défilement.

resizable=[yes,no,1,0] Indique si l'utilisateur doit pouvoir redimensionner la fenêtre.

width = pixels Indique la largeur de la fenêtre en pixels

height = pixels Indique la hauteur de la fenêtre en pixels

2.3.a. Exemple1: harcèlement publicitaire

```
<html> <head>
<script>
<!--
var f
function newWin()
{f=window.open("pub.htm","fenetre_pub",
               "toolbar=no,location=no,width=200,height=200")
}
-->
</script> </head>
<body onload='newWin()' onunload='f.close()>
Document principal
</body> </html>
```

2.4. L'objet window.location (emplacement / url)

L'objet **location** reflète des informations sur l'URL en cours.

Exemple:

window.location.href="<http://www.netscape.com/>"

Propriétés:

host la partie hostname et le port d'une URL.

hostname nom et le domaine de l'URL spécifié.

href URL complète du document

pathname la partie de l'URL correspondant au chemin (sans nom de machine mais avec nom de fichier)

port partie port d'un URL (si différent de 80) ex: 8080

protocol partie protocole d'un URL (avec les :, mais sans les slash). exemples: http: ftp:

search partie search (recherche) d'un URL (c'est-à-dire les informations situées après le point d'interrogation).

2.5. L'objet history

L'objet qui permet d'accéder à la liste contenant l'historique des URL visitées en JavaScript est l'objet **history**.

Les méthodes de l'objet **history** sont les suivantes :

length (*propriété*) Donne la longueur de l'historique

back() Charge l'URL précédente de l'historique

forward() Charge l'URL suivante de l'historique

go("url") Charge l'URL indiquée par une adresse relative dans l'historique.

La méthode **history.go()** admet pour argument une chaîne plutôt qu'un entier. Dans ce cas, elle charge l'entrée de l'historique la plus proche qui contient cette chaîne dans son URL.

Exemple:

history.back() retour à la page précédente.

IX - Annexe – Canvas et Chart

1. Api "canvas" et bibliothèque "chart"

1.1. Api "canvas" (html5)

Exemple :

```
<html>
<head>
<title>canvas api</title>
<script>
function init() {

var canvasElement = document.getElementById("myCanvas");
var ctx = canvasElement.getContext("2d");

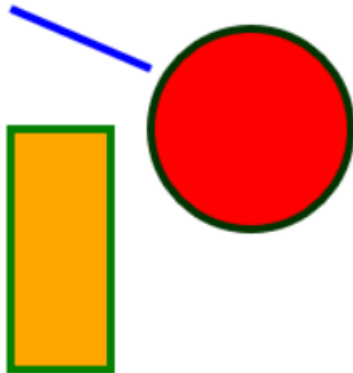
//cercle
var c={ xC:200, yC:100, r:50,
  fillColor:'red', lineWidth:4, lineColor:'#003300'
};
ctx.beginPath(); //start path with upcoming attributes (ctx.lineWidth , ctx.strokeStyle, ...)
ctx.arc(c.xC, c.yC, c.r, 0 /*startAngle*/, 2 * Math.PI /*endAngle*/, false);
  if(c.fillColor != null){
    ctx.fillStyle = c.fillColor;
    ctx.fill();
  }
ctx.lineWidth = c.lineWidth;
ctx.strokeStyle = c.lineColor;
ctx.stroke(); //draw path (arc or ...)

//line
var l={ x1:80, y1:40, x2:150, y2:70,
  lineWidth:4, lineColor:'blue'
};
ctx.beginPath(); //start or reset path
ctx.moveTo(l.x1,l.y1);
ctx.lineTo(l.x2,l.y2);
ctx.strokeStyle = l.lineColor;
ctx.lineWidth = l.lineWidth;
ctx.stroke(); //draw path

//rectangle
var r={ x1:80, y1:100, width:50, height:120,
  fillColor:'orange', lineWidth:4, lineColor:'green'
};
ctx.beginPath();
ctx.rect(r.x1,r.y1,r.width,r.height);
  if(r.fillColor != null){
    ctx.fillStyle = r.fillColor;
    ctx.fill();
  }
}
```

```
ctx.strokeStyle = r.lineColor;  
ctx.lineWidth = r.lineWidth;  
ctx.stroke(); //draw path  
  
}  
</script>  
</head>  
<body onload="init()">  
<h2>canvas api</h2>  
<canvas id="myCanvas" width="400" height="400"></canvas>  
  
</body>  
</html>
```

canvas api



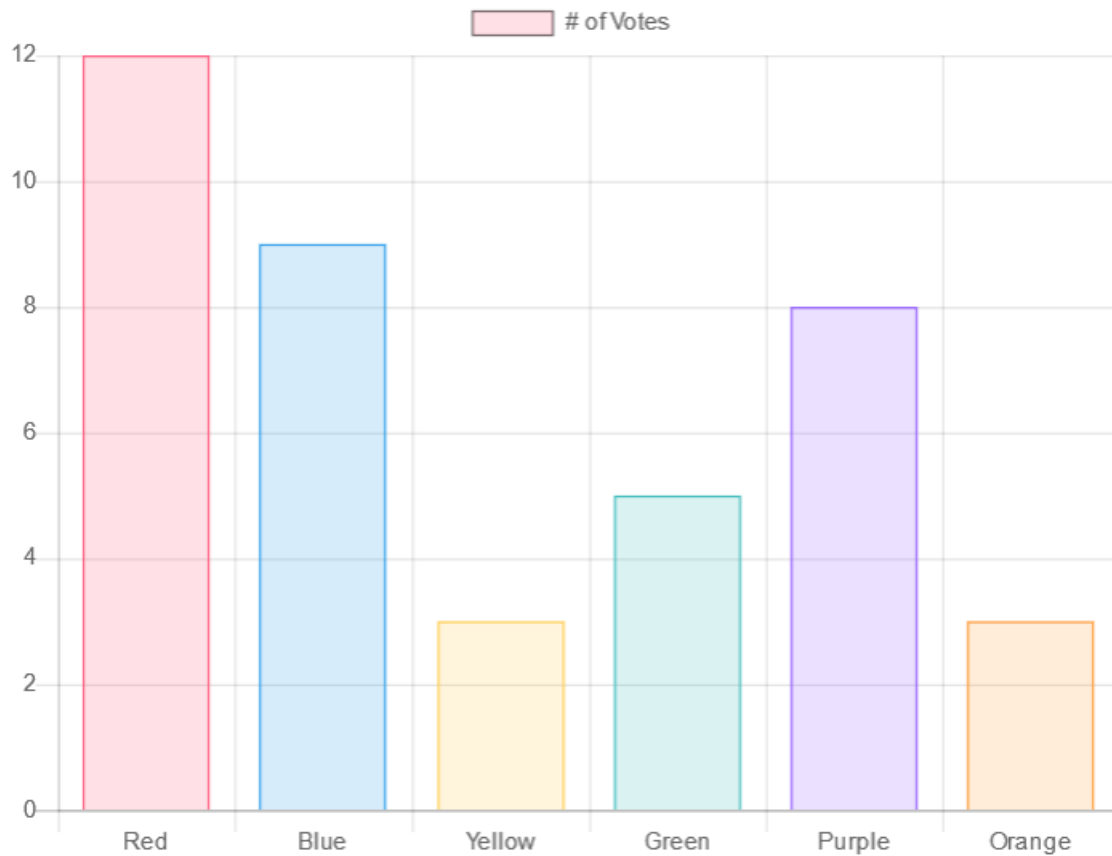
Pour approfondir le sujet --> https://www.w3schools.com/tags/ref_canvas.asp

1.2. Bibliothèque "chart" (javascript)

<https://www.chartjs.org/>

```
<script src="lib/chart.2.7.3.min.js"></script>
```

bar chart



```
var canvasChart1Element = document.getElementById("myChart1");
var ctx1 = canvasChart1Element.getContext("2d");

var myChart1 = new Chart(ctx1, {
  type: 'bar',
  data: {
    labels: ["Red", "Blue", "Yellow", "Green", "Purple", "Orange"],
    datasets: [{
      label: '# of Votes',
      data: [12, 9, 3, 5, 8, 3],
      backgroundColor: [
        'rgba(255, 99, 132, 0.2)',      'rgba(54, 162, 235, 0.2)',
        'rgba(255, 206, 86, 0.2)',    'rgba(75, 192, 192, 0.2)',
        'rgba(153, 102, 255, 0.2)',   'rgba(255, 159, 64, 0.2)'
      ],
      borderColor: [
        'rgba(255,99,132,1)',          'rgba(54, 162, 235, 1)',
        'rgba(255, 206, 86, 1)',      'rgba(75, 192, 192, 1)',
      ]
    }]
  }
});
```

```

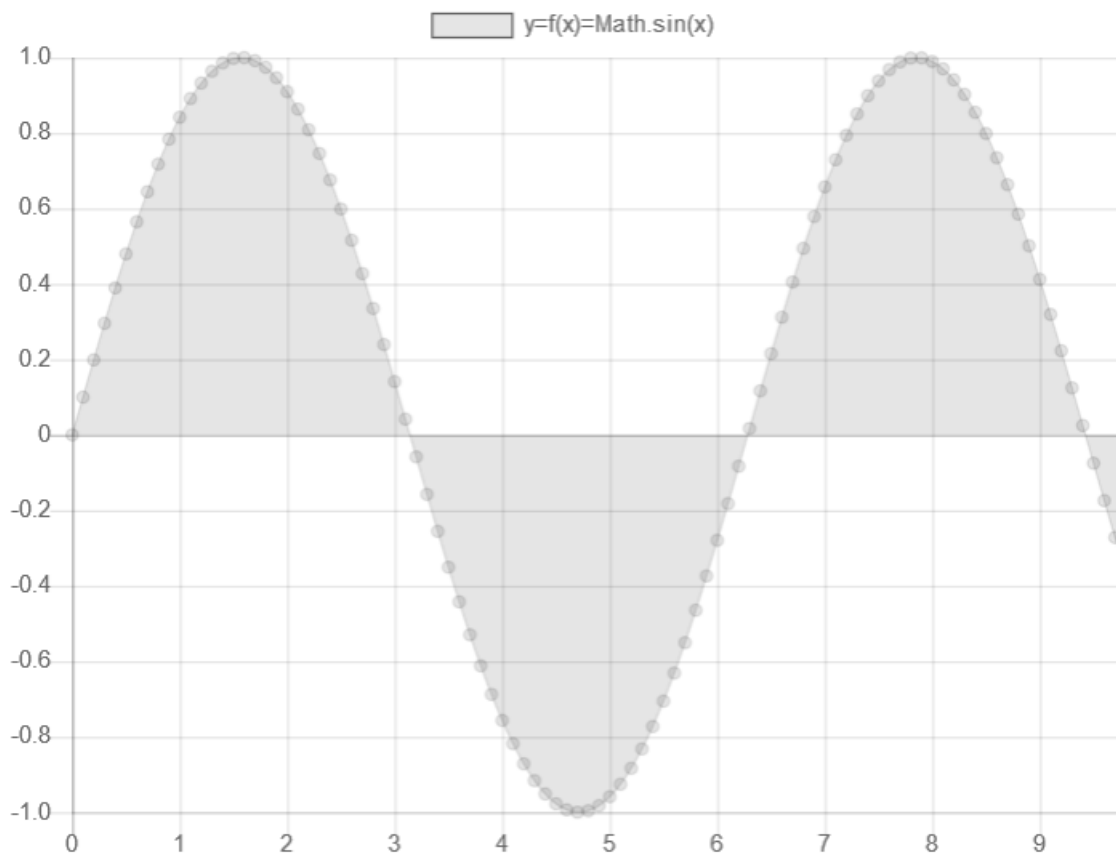
        'rgba(153, 102, 255, 1)',          'rgba(255, 159, 64, 1)'
    ],
    borderWidth: 1
  }
},
options: {
  scales: {
    yAxes: [{
      ticks: {
        beginAtZero:true
      }
    }]
  }
}
});

```

Autre exemple :

curve / graph / function

pour x allant de a
 $y=f(x)=\text{Math.sin}(x)$



<h3> curve / graph / function </h3>

pour x allant de <input id="xMin" value="0"/> a

<input id="xMax" value="10"/>


```
y=f(x)=<input id="fx" value="Math.sin(x)" /><br/>
<input type="button" id="btnDraw" value="afficher courbe" /><br/>
<canvas id="myChart2" width="400" height="300"></canvas>
```

```
var btnDraw = document.getElementById("btnDraw");
btnDraw.addEventListener("click",function(event){

    var ctx2 = document.getElementById("myChart2").getContext('2d');
    var fx = document.getElementById("fx").value;
    var xMin = document.getElementById("xMin").value;
    var xMax = document.getElementById("xMax").value;
    var yMin=0;   var yMax=0;

    var x,y;
    pointValues=[];
    xMin=Number(xMin)*1.0;xMax=Number(xMax)*1.0;
    var n=100;
    var dx=(xMax-xMin)/n;
    for(x=xMin;x<=xMax;x+=dx){
        y=eval(fx);
        if(y<=yMin) yMin=y;
        if(y>=yMax) yMax=y;
        pointValues.push( {x:x,y:y} );
    }
    var dy=(yMax-yMin)/100;
    console.log(pointValues);

    var myChart2 = new Chart(ctx2, {
type: 'line',
data: {
    datasets: [{
        label: 'y=f(x)='+fx,
        data: pointValues,
        borderColor: [
            'rgba(33, 232, 234, 1)',    'rgba(33, 232, 234, 1)',
            'rgba(33, 232, 234, 1)',    'rgba(33, 232, 234, 1)',
            'rgba(33, 232, 234, 1)',    'rgba(33, 232, 234, 1)'
        ],
        borderWidth: 1
    }]
```



```
    }],  
  },  
  options: {  
    scales: {  
      xAxes: [{  
        type: 'linear',  
        position: 'bottom',  
        ticks: {  
          min: xMin,  
          max: xMax,  
          stepSize: dx*10,  
          fixedStepSize: dx*10,  
        }  
      }],  
      yAxes: [{  
        ticks: {  
          min: yMin,  
          max: yMax,  
          stepSize: dy*10,  
          fixedStepSize: dy*10,  
        }  
      }]  
    }  
  }  
});  
  
});
```

X - Annexe – JQuery

1. Présentation de JQuery

JQuery est une **bibliothèque javascript** très populaire qui permet de simplifier considérablement la programmation de l'aspect dynamique des pages HTML qui s'affichent dans un navigateur.



1.1. Principales fonctionnalités de JQuery

- **Syntaxe** très **compacte** et cohérente vis à vis de HTML_DOM et CSS
- Fonctionne avec presque tous les navigateurs (**masque les différences entre IE, Firefox et Chrome**)
- **Sélections** et **modifications** efficaces des **éléments d'une page HTML** (styles css , valeurs , formulaire , div , ...).
- Permet de coder simplement (et de façon portable) des **gestionnaires d'événement** en javascript .
- Simplifie la prise en charge d'**AJAX**
- Extensibilité via **plugins** (nouveaux composants , nouveaux effets , ...)
- Se combine bien avec la librairie "**bootstrap CSS**" (via des plugins jquery pour bootstrap) .
- Tellement utilisé que c'est devenu un "*standard de fait*" .

1.2. Utilisation de JQuery (exemples)

2 versions possibles (**à télécharger** et placer par exemple dans un répertoire relatif "**lib**" :

- **jquery.js** (avec code javascript lisible pour faciliter le développement)
- **jquery.min.js** (avec code compressé pour optimiser les futurs téléchargements vers les navigateurs des clients en mode "production").

NB : il est éventuellement possible de faire référence à une version distante de la librairie "jquery.js" (hébergée par un serveur "**CDN**" / "Content Delivery Network" :) sans l'incorporer dans le projet en utilisant une URL absolue telle que:

<http://code.jquery.com/jquery.min.js>

Ou bien

<http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js>

PageXy.html

```

<!DOCTYPE html>
<html>
  <head> <meta charset="UTF-8"> <title>Le titre du document</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
    <script src="lib/jquery.js"></script>
    <script>
      $(function() {
        $('#p1').css('border-width', '3px').css('border-style','solid');
        $('#p2').html("nouveau contenu html du paragraphe p2");
        $('p').css('color', 'blue'); // tous les éléments de type p en bleu
        $('#btnEuroToFranc').on('click',function(){
          $('#txtFranc').html( 6.55957 * Number( $('#txtEuro').val() ) );
        });
      });
    </script>
  </head>
  <body>
    <p id="p1"> texte du paragraphe1 </p>
    <p id="p2"> texte du paragraphe2 </p>
    <form>
      somme en euro : <input type="text" id="txtEuro" /> <br/>
      <input type="button" value="euroToFranc" id="btnEuroToFranc" /> <br/>
      montant equivalent en franc : <span id="txtFranc"/>
    </form>
  </body>
</html>

```

texte du paragraphe1

nouveau contenu html du paragraphe p2

somme en euro :

euroToFranc

montant equivalent en franc : 98.39355

2. Essentiel de JQuery

2.1. Syntaxe générale et comportement

\$(selecteurXy) est un équivalent compact de **jQuery**(selecteurXy) .

Et, en particulier, on a les équivalences suivantes au niveau du point de démarrage :

```
jQuery(document).ready( function() {
    // ici, le DOM de la page est entièrement défini et peut être manipulé via jQuery
});
```

```
$(document).ready( function() {
    // ici, le DOM de la page est entièrement défini et peut être manipulé via jQuery
});
```

```
$( function() {
    // ici, le DOM de la page est entièrement défini et peut être manipulé via jQuery
});
```

La syntaxe générale d'une expression jQuery est **\$('selecteurXy').actionZz(parametres);**

La syntaxe des sélecteurs est la même que pour les styles CSS.

Les actions sont quelquefois en mode "lecture" , d'autres fois en mode "modification" et peuvent être enchaînées : **\$(selecteurXy).actionA('param1a' , 'param2a').actionB('paramB') ;**

2.2. Principaux sélecteurs (idem CSS)

\$('*');	tous les éléments (inclus <html>, <head> et <body>).
\$('element');	tous les éléments étant de type correspondant.
\$('#element');	l'élément ayant l'id donné (unique).
\$('.element');	tous les éléments ayant la classe donnée.
\$('element1, element2');	les éléments 1 et 2 et ... (selon # , . ou autre)
\$('parent enfant');	tous les enfants directs ou indirects de l'élément parent.
\$('parent > enfant');	tous les enfants directs de l'élément parent.
\$('frere + element');	tous les éléments précédés directement d'un frère.
\$('frere ~ element');	les éléments précédés directement ou indirectement d'un frère.
\$('element[attr]');	tous les éléments possédant l'attribut donné.

<code>\$(element[attr="val"]);</code>	tous les éléments possédant l'attribut donné et dont la valeur est égale à celle spécifiée.
<code>\$(element[attr!="val"]);</code>	tous les éléments possédant l'attribut donné et dont la valeur est différente de celle spécifiée.
<code>\$(element[attr*="val"]);</code>	tous les éléments possédant l'attribut donné et dont la valeur contient , entre autres, la chaîne spécifiée.
<code>\$(element[attr^="val"]);</code>	tous les éléments possédant l'attribut donné et dont la valeur commence par la chaîne spécifiée.

`$('a')[0]` retourne le premier lien hypertexte de la page.

`$('p.redClass')[3]` retourne la quatrième balise `<p>` de classe `redClass`.

`$('[src]')` sélectionne tous les éléments qui possèdent un attribut `src` ;

`$('[width="100"]')` sélectionne tous les éléments qui ont un attribut `width` égal à 100.

Quelques filtres :

<code>\$(':even');</code> <code>\$(':odd');</code>	index est pair ou impair
<code>\$(':empty');</code>	Sans enfant
<code>\$(':first');</code> <code>\$(':last');</code>	premier , dernier
<code>\$(':first-child');</code> <code>\$(':last-child');</code>	premier , dernier enfant
<code>\$(':focus');</code>	tous les éléments ayant actuellement le focus.
<code>\$(':hidden');</code> <code>\$(':visible');</code>	caché , visible

Exemples :

`$('tr:odd')` sélectionne les lignes impaires d'un tableau.

`$('td:first')` sélectionne la première cellule

Sélection selon type des éléments d'un formulaire :

`$(':input');` `$(':button');` `$(':checkbox');` `$(':checked');` `$(':file');` `$(':password');`
`$(':radio');` `$(':submit');` `$(':text');`

Conversions (js/dom , jquery):

```
var variableJQ = $(variableJS);
var variableJS = $(selecteurJQ).get();
```

2.3. Principales actions

Méthodes de parcours:

find()	Permet de trouver un enfant particulier.
children()	Trouve l'enfant direct de l'élément ciblé.
parent()	Trouve le premier parent de l'élément ciblé.
parents()	Trouve tous les ancêtres de l'élément ciblé.
siblings()	Trouve tous les éléments "frères" (de même niveau)
each()	Boucle sur chaque élément.

```
$( 'sel' ).each( function(index) {
    //Une ou plusieurs instructions JavaScript
    //NB: index vaudra automatiquement 0,1,2,..., n-1
});
```

Récupération de positions et de tailles:

width(), height()	largeur, hauteur
innerWidth(), innerHeight()	Largeur, hauteur en prenant en compte les marges intérieures mais pas les bordures.
outerWidth(), outerHeight()	Largeur, hauteur en prenant en compte ses marges intérieures, extérieures, et ses bordures.
offset()	Récupère les coordonnées absolues de l'élément ciblé.
position()	Récupère les coordonnées relatives de l'élément ciblé.
scrollLeft(), scrollTop()	Positions horizontale et verticale de la barre de défilement par rapport à la page.

Manipulation d'attributs:

attr()	Récupère ou modifie l'attribut d'un élément.
prop()	Gère une propriété d'un élément (ex: disabled)
removeAttr()	Supprime l'attribut d'un élément.
addClass()	ajoute ou supprime une classe à un élément.
removeClass()	
hasClass()	Vérifie si un élément a ou pas telle classe css.

Exemples:

```
$('#itemXy').attr('src','logo.gif'); $('#btnHide').prop('disabled',true);
$('#itemXy').attr({ src: 'logo.gif', alt: 'société Xy', width: '250px'});
$('a').attr('target', function() {
    if(this.host == location.host) return '_self'
    else return '_blank'
});
```

Manipulations HTML :

html()	Récupère ou modifie le contenu HTML de l'élément ciblé.
text()	Récupère ou modifie le contenu textuel de l'élément ciblé.
val()	Récupère ou modifie la valeur d'un élément de formulaire.
append() <i>ou</i>	Ajoute du contenu HTML ou textuel à la fin de l'élément ciblé(à l'intérieur , comme enfant , sous élément)
appendTo()	
prepend() <i>ou</i>	Ajoute du contenu HTML ou textuel au début de l'élément ciblé (à l'intérieur , comme enfant , sous élément)
prependTo()	
before()	Ajoute du contenu HTML ou textuel avant l'élément ciblé.
after()	Ajoute du contenu HTML ou textuel après l'élément ciblé.
empty()	Vide un élément.
remove()	Supprime un élément.
wrap()	Enveloppe un élément.

2.4. Principaux gestionnaires d'événement

La source d'un événement est **evt.target** et l'on peut commencer des instructions/actions utiles avec **evt.target.id** ou bien **\$(evt.target)** .

Bien que l'on puisse directement gérer des événements javascript via des méthodes jquery reprenant spécifiquement les noms des événements, il est également possible d'utiliser la fonction générique **.on('type_evenement', function() { ...})** qui est "universelle", "plus paramétrable" et "utilisable sur de nouveaux événements personnalisés" .

Exemple simple:

```
$('#btnEuroToFranc').on('click',function(){
    $('#txtFranc').html( 6.55957 * Number( $('#txtEuro').val() ) );
});
```

Principaux événements "souris":

Événements (evt.type)	circonstances	paramètres importants
click	click gauche	
dblclick	double click	
mousedown	Appui sur un bouton (gauche, droit, ...) de la souris	evt.which valant 1 pour bouton gauche 2 pour bouton central 3 pour bouton droit
mouseover ou mouseenter	début de survol de l'élément	
mouseout ou mouseleave	fin de survol de l'élément	
mousemove	Déplacement du pointeur de souris au dessus de l'élément	
mouseup	Relachement d'un bouton (gauche, droit, ...) de la souris	idem mousedown
scroll	scroll via molette de la souris	

Exemple:

```
$('#target').on('mousedown', function(evt){
    $('#message').html('Événement : ' + evt.type + '. Bouton pressé : ' + evt.which );
});
```

Position de la souris (véhiculée par l'événement source) :

'x='+ **evt.pageX**+' y='+ **evt.pageY** // position absolue par rapport à la page

'x='+ evt.clientX+' y='+ evt.clientY // position absolue par rapport à la zone cliente (partie visible
 // de la page dans le navigateur selon position des ascenseurs)
 'x='+ (evt.pageX - this.offsetLeft) +' y='+ (evt.pageY - this.offsetTop) // % élément courant

Principaux événements "clavier":

Événements (evt.type)	circonstances	paramètres importants
keydown	Appui sur une touche	evt.which valant le code touche (sans différence min et MAJ et avec code pour F1, F2 , ...)
keyup	Relachement d'une touche	idem keydown
keypress	(Appui + relachement) effectué	evt.which valant le code ASCII du caractère (ex : 65 pour 'A' 97 pour 'a')

NB: var c = **String.fromCharCode**(e.which); permet de convertir le code ASCII en caractère correspondant ; ce qui est pratique pour coder 'keypress' .

Principaux événements applicables sur "contrôle d'un formulaire ou ...":

Événements (evt.type)	circonstances
focus	Réception du focus
blur	Perte du focus
focusin	Réception du focus par l'élément ou un de ses enfants
focusout	Perte du focus par l'élément ou un de ses enfants
resize	Redimensionnement
change	Changement d'état ou de sélection

```
function my_generic_event_handler(evt){
  var rapport_evt = "evt.type=" + evt.type + " and evt.which=" + evt.which + " on #" + evt.target.id;
  var new_option = "<option>" + msg + "</option>" ;
  $("#listboxEvents").append(new_option) ;
}
```

```
$('#select1').on('click change focus blur',my_generic_event_handler);
```

```
$('#radioCelibataire,#radioEnCouple').on('click change focus blur',my_generic_event_handler);
```

Les événements "load" et "unload" sont d'autre part déclenchés lors du téléchargement des pages et des images .

Eventuelle désactivation de certains gestionnaires d'événements :

`$('#txt1').off('click dblclick mousedown mouseup mouseover mouseout scroll');`

2.5. Principaux effets (transition , animation)

animate()	Anime une ou plusieurs propriété(s) CSS, à l'aide d'arguments tels que la durée ou l'accélération de l'animation.
hide()	Fait disparaître un élément (et lui donne la propriété <i>display:none</i>).
show()	Fait réapparaître un élément.
fadeOut()	Fait disparaître un élément (qui aura la propriété <i>display:none</i>) avec un effet de fondu.
fadeIn()	Fait réapparaître un élément avec un effet de fondu.
slideUp()	Fait disparaître un élément avec un effet de glissement.
slideDown()	Fait réapparaître un élément avec un effet de glissement.
stop()	Arrête l'animation en cours.

2.6. Ajax via jquery

\$.ajax()	Exécute une requête AJAX de type GET ou POST ou PUT ou DELETE ou ...
\$.post()	Exécute une requête AJAX de type POST (raccourci de \$.ajax()).
\$.get()	Exécute une requête AJAX de type GET (raccourci de \$.ajax()).
load()	Charge du contenu HTML de manière asynchrone.

3. Plugins JQuery

Utilisation de plugins prédéfinis

<script src="....js"></script> et lire la documentation du plugin .

Pack "jquery-ui"

<http://jqueryui.com> (demo , download) est l'URL de référence pour récupérer le pack de plugins "jquery-ui" .

Interactions (automatiques) prises en charge par jquery-ui :

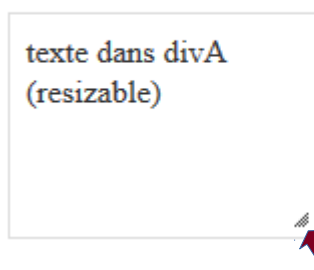
Draggable , **Droppable** , **Resizable** , **Selectable** , **Sortable**

Exemple :

```
<html>
<head> <meta charset="UTF-8"> <title>draggable , resizable with jquery-ui</title>
  <link rel="stylesheet" type="text/css" href="css/jquery-ui.css" />
  <script src="lib/jquery-2.2.1.js"></script>
  <script src="lib/jquery-ui.min.js"></script>
  <style>
    #divA { width: 150px; height: 150px; padding: 0.5em; }
  </style>
  <script>
    $(function() {
      $('#spanXx').css('border-width', '3px').css('border-style','solid');
      $('#spanXx').draggable();
      $('#divA').resizable();
    });
  </script>
</head>
<body>
  <span id="spanXx"> texte dans div or spanXx (draggable)</span>
  <br/><br/>
  <div id="divA" class="ui-widget-content"> texte dans divA (resizable)</div>
</body>
```

→ texte dans div or spanXx (draggable)

texte dans divA
(resizable)



Principaux Widgets de jquery-ui :

[Accordion](#) , [Autocomplete](#), [Button](#), [Datepicker](#) , [Dialog](#) , [Menu](#)
[Progressbar](#) , [Selectmenu](#) , [Slider](#) , [Spinner](#) , [Tabs](#) , [Tooltip](#)

Exemples :

```
<html>
<head> <meta charset="UTF-8"> <title>jquery-ui widget</title>
  <link rel="stylesheet" type="text/css" href="css/jquery-ui.css" />
  <script src="lib/jquery-2.2.1.js"></script>
  <script src="lib/jquery-ui.min.js"></script>
  <style> #myDialog { display: none;}</style>
  <script>
    $(function() {
      $.datepicker.regional['fr'] = {
monthNames:['Janvier','F&eacute;vrier','Mars','Avril','Mai','Juin','Juillet',
  'Ao&ucirc;t','Septembre','Octobre','Novembre','D&eacute;cembre'],
dayNames: ['Dimanche','Lundi','Mardi','Mercredi','Jeudi','Vendredi','Samedi'],
dayNamesMin: ['Di','Lu','Ma','Me','Je','Ve','Sa'],
dateFormat: 'dd/mm/yy', firstDay: 1, isRTL: false,
showMonthAfterYear: false, yearSuffix: "};
      $.datepicker.setDefaults($.datepicker.regional['fr']);
      $( "#datepicker1").datepicker();
      $('#btnOpenDlg').on('click', function(){
        $("#myDialog" ).dialog( {modal: true,
          buttons: { "Oui": function() {
            $('body').css('background', $('#sBackColor').val());
            $( this ).dialog( "close" );},
            "Non": function() { $( this ).dialog( "close" );}
          }
        });
      } //end of evt function
    });
  //end of on_click
  $('#serieOnglets').tabs(); //look et comportement sous forme d'onglets
});
</script>
</head>
<body> Date: <input type="text" id="datepicker1" /> <br/>
```

```

<input type='button' id="btnOpenDlg" value="open myDialog" /> <br/>
<div id="myDialog" title="Simple Dialog">
    Cette boîte de dialogue peut être redimensionnée, déplacée et fermée. <hr/>
    backColor : <select id="sBackColor">
        <option checked='true' >white</option>
        <option value='#eeee00'>yellow</option>
    </select> Voulez vous cette couleur de fond ?
</div>

<div id="serieOnglets">
<ul>
    <li><a href="#onglet-1">Titre onglet 1</a></li>
    <li><a href="#onglet-2">Titre onglet 2</a></li>
    <!-- <li><a href="...">recup possible contenu via ajax</a></li> -->
</ul>
<div id="onglet-1"> <p>contenu onglet1</p> </div>
<div id="onglet-2"> <p>contenu de l' onglet 2</p> </div>
</div> </body> </html>

```

Date: 30/03/2016

open

Titre onglet 1

cc

Mars 2016						
Lu	Ma	Me	Je	Ve	Sa	Di
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Simple Dialog

Cette boîte de dialogue peut être redimensionnée, déplacée et fermée.

backColor : white Voulez vous cette couleur de fond ?

Oui

Non

Titre onglet 1

Titre onglet 2

contenu de l' onglet 2

3.1. Programmation (facile) de plugins personnalisés

Squelette d'un plugin jquery :

```
(function($) {  
  $.fn.myFct = function(éventuels_paramètres)  
  {  
    this.each(function() {  
      // les instructions du plugin  
      $(this).wrap('<b><i></i></b>'); //exemple simple  
    });  
    return this; //pour pouvoir enchaîner (si nécessaire)  
  };  
})(jQuery);
```

Utilisation (habituelle) :

\$('#idXy').myFct('param1', 'param2');

Exemple :

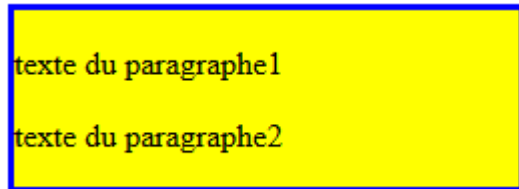
js/my-jquery-plugin.js

```
(function($) {  
  $.fn.withBorder = function(myBorderColorParam)  
  {  
    var borderColor = myBorderColorParam;  
    if(borderColor == undefined){  
      borderColor = 'black';  
    }  
  
    this.each( function() {  
      $(this).css('border-width', '3px')  
        .css('border-color', borderColor).css('border-style','solid');  
    });  
    return this;  
  };  
})(jQuery);
```

Utilisation :

```
<html>  
<head> <meta charset="UTF-8"> <title>test custom jquery plugin</title>  
  <script src="lib/jquery-2.2.1.js"></script>  
  <script src="js/my-jquery-plugin.js"></script>  
</script>
```

```
$(function() {  
    //$('#divA').withBorder().css('background','yellow');  
    $('#divA').withBorder('blue').css('background','yellow');  
});  
</script>  
</head>  
  
  
  
  
  
  
  
  
  
<body>  
    <div id="divA">  
        <p id="p1"> texte du paragraphe1 </p>  
        <p id="p2"> texte du paragraphe2 </p>  
    </div>  
</body>  
</html>
```



XI - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

2. TP