javascript

(es5, es2015, es2017)

Table des matières

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 coté "front-end" , il est interprété en tant que complément des pages HTML par tous les navigateurs internet (IE/edge , firefox , Chrome , opera , ...)
- Du coté "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 "EcmaSript") 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 librairies de plus haut niveau ont été ensuite conçues et utilisées vers les années 2007 , ... , 2012, ... (ext-js , jquery , ...) .
"jquery" est la bibliothèque "javascipt" qui a été le 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 coté "serveur"

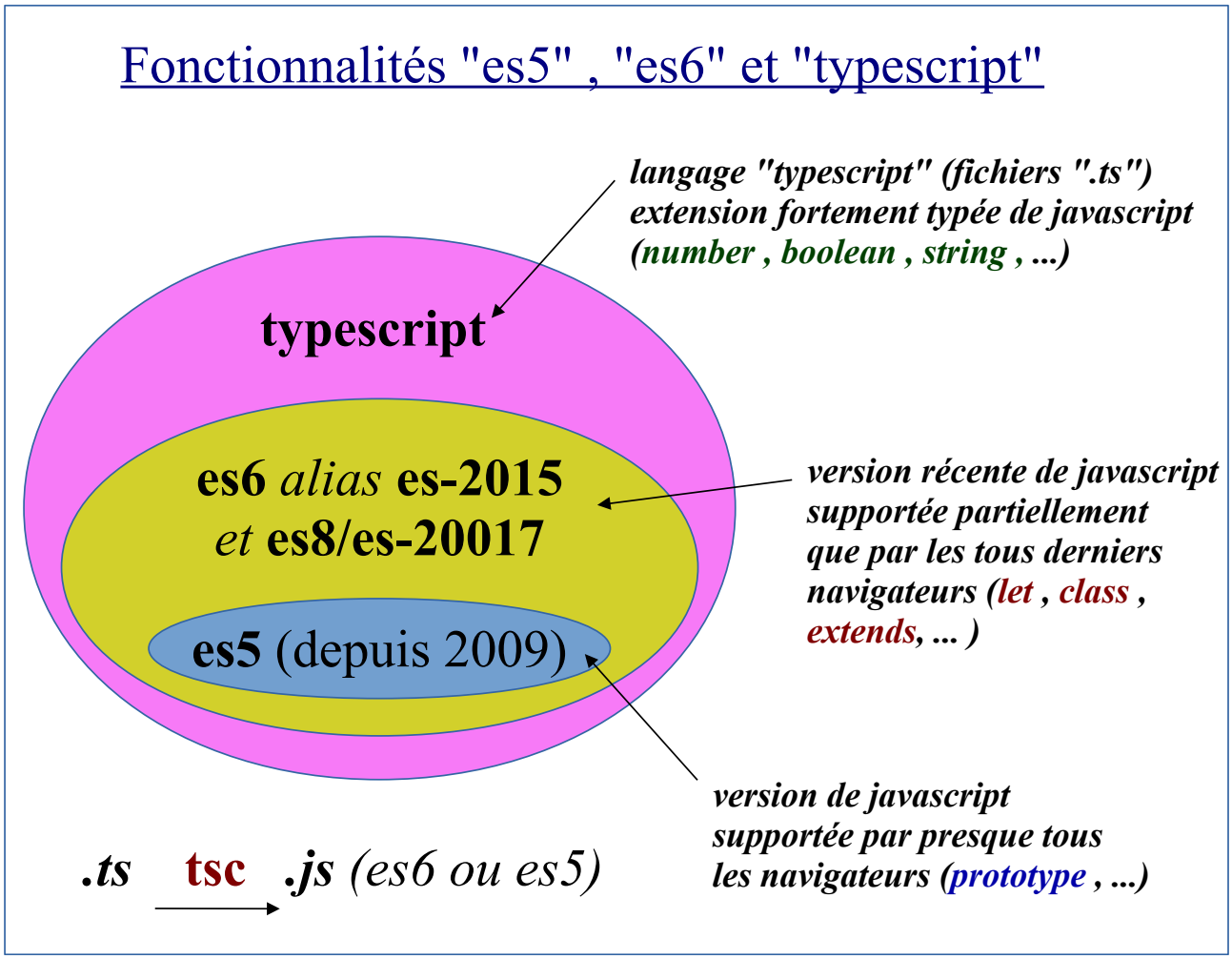
de manière très modulaire (essentiellement via l'écosystème node-js/npm) .
Son fonctionnement très "asynchrone" a permis d'obtenir de bonnes performances a peu de frais .
Beaucoup de "bonnes idées de javascript/node-js" (asynchronisme , modularité, ...) sont petit à petit reprises et intégrées dans des 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 "cordava/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** (**D**ocument **O**bject **M**odel) .

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

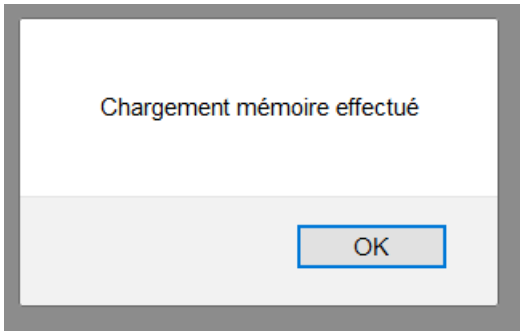
```
<script>
//instructions en JavaScript
</script>
```

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** (Mémorisation 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

```
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. Boites 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 "*developpement 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 entres 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 (affectaction)

```
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
==	Retourne la valeur true, si les opérandes sont **égales**
!=	Retourne la valeur true, si les opérandes sont **différentes**
<	Retourne la valeur true, si l'opérande gauche est **strictement inférieure** à l'opérande droite.
<=	Retourne la valeur true, si l'opérande gauche est **inférieure ou égale** à l'opérande droite.
>	Retourne la valeur true, si l'opérande gauche est **strictement supérieure** à l'opérande droite.
>=	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 (≅ 1.414)
SQR1_2	Racine carré de ½ (≅ 0.707)
E	Constante d'Euler (≅ 2.718)
LN10	Logarithme naturel de 10 (≅2.302)
LN2	Logarithme naturel de 2 (≅ 0.693)
PI	Pi (≅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 <B>
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 <FONT COLOR =couleur> et </FONT>
fontsize(taille)	Encadre la chaîne dans balise html <FONT size =taille> et </FONT>
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.
- **getYear()** 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. Eléments divers du langage javascript

8.1. Fonction à nombre d'arguments variable

```
function f1()
{
nb_arg=f1.arguments.length
if(nb_arg > 0)  { premier_arg=f1.arguments[0] ; alert(premier_arg) }
if(nb_arg > 1)  { deuxieme_arg=f1.arguments[1] ; alert(deuxieme_arg) }
}
```

f1(); f1('a1'); f1('a1','a2')

Attention : **arguments n'est pas accessible lorsque javascipt 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 JasvaScript 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 déja
{       // 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** = ***D**ocument **O**bject **M**odel*" 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)

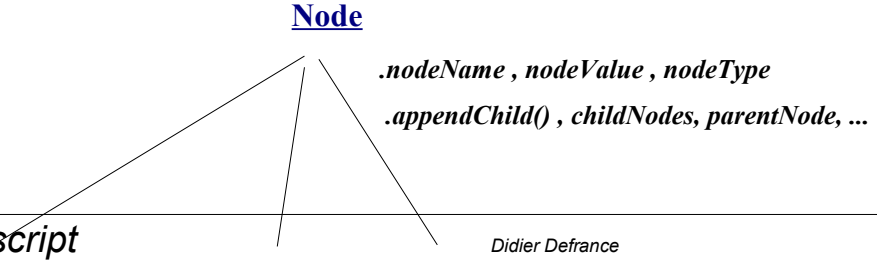
Le modèle objet normalisé par le W3C est complètement documenté au bout de l'URL suivante: ***http://www.w3.org/DOM/***
HTML peut être vu comme un cas particulier de XML . On parle de **XHTML**.
Le **DOM niveau 1 (Core)** s'applique à **XML**
Le **DOM niveau 1 (Html)** s'applique à (X)**HTML**.
Le **DOM 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**" .



Element	**Text**	**Document**	**.....**
.getAttribute(...)		***.documentElement***	
.setAttribute(...,...)		***.createElement(...), .createTextNode(...)***	

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.

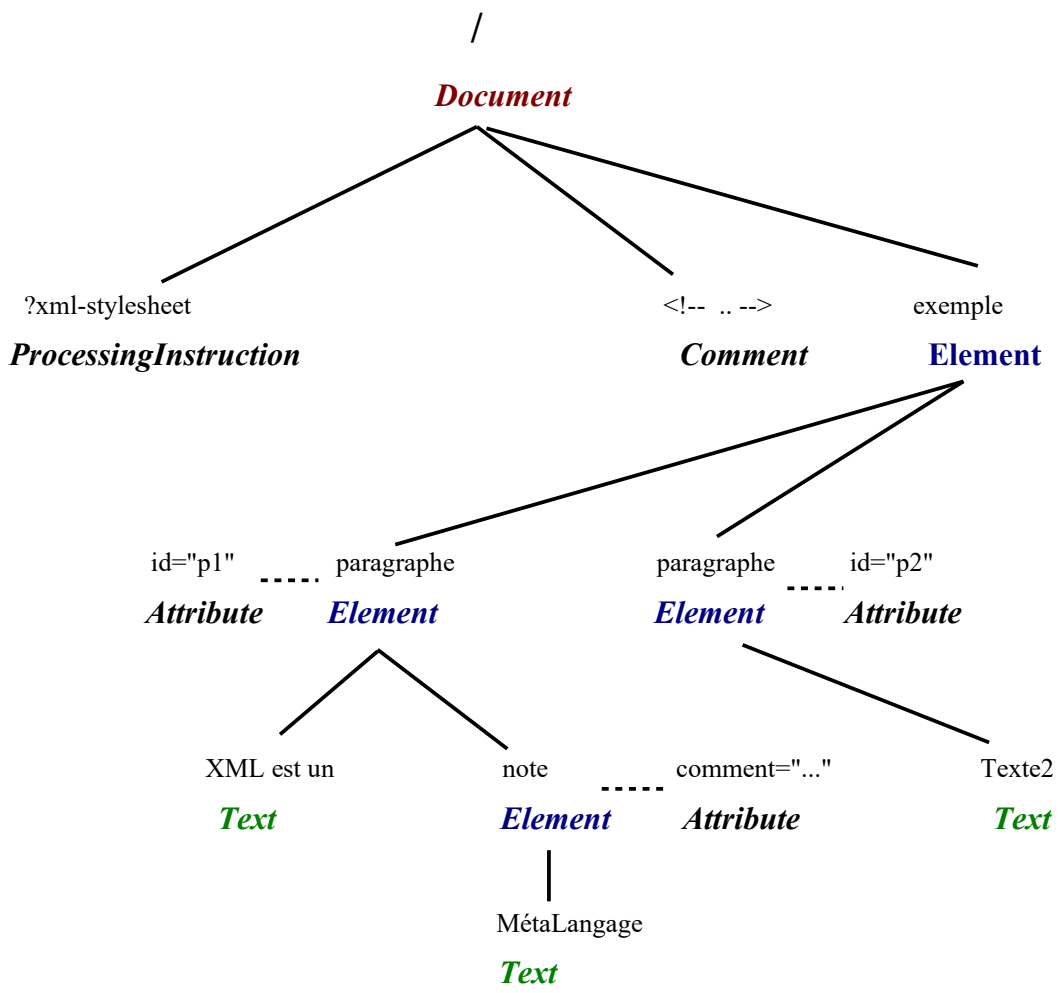
Correspondance entre fichier XML et arbre "DOM" en mémoire :

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();
e1=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 JS0N 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 "**X**ml**H**ttp**R**equest" 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'ai 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 librairies "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 ...>....) .

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. Speudo 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

d1.color=black
d1 is a Dog
d1 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 limité 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("primptemps", "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 array1Iterable = [ 123 , "abc" ];
//var iterable1 = array1Iterable;
const string1Iterable = "XYZ";
var iterable1 = string1Iterable;
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
```

VIII - Objets es2015 (class, ...)

1. Prog. orientée objet "es2015" (class, extends, ...)

1.1. Préambule ("poo via prototype es5" --> "poo es2015")

La programmation orientée objet existait déjà en "javascript/es5" mais avec une syntaxe très complexe , verbeuse et peu lisible (prototypes avancés).

Dès l'époque "es5" , le mot clef constructor existait et l'on pouvait même définir une relation d'héritage avec une syntaxe complexe et rebutante.

La nouvelle version "es6/es2015" du langage "javascript/ecmascript" a enfin apporté une nouvelle syntaxe "orientée objet" beaucoup plus claire et lisible donnant envie de programmer de nouvelles classes d'objet en javascript moderne.

1.2. Classe et instances

```
class Compte{
	constructor(numero,label,solde){
		this.numero = numero;
		this.label=label;
		this.solde=solde;
	}

	debiter(montant)  {
		this.solde -= montant; // this.solde = this.solde - montant;
	}

	crediter(montant)  {
		this.solde += montant; // this.solde = this.solde + montant;
	}
}
```

```
let c1 = new Compte(); //instance (exemplaire) 1
console.log("numero et label de c1: " + c1.numero + " " + c1.label);// undefined undefined
console.log("solde de c1: " + c1.solde); // undefined

let c2 = new Compte(); //instance (exemplaire) 2

c2.solde = 100.0;
c2.crediter(50.0);

console.log("solde de c2: " + c2.solde);  //150.0

let c3 = new Compte(3,"compte3",300); //instance (exemplaire) 3
```

```
console.log("c3: " + JSON.stringify(c3));  //{"numero":3,"label":"compte3","solde":300}
```

NB: Sans initialisation explicite (via constructeur ou autre) , les propriétés internes d'un objet sont par défaut à la valeur "**undefined**" .

1.3. "constructor" avec éventuelles valeurs par défaut

Un constructeur est une méthode qui sert à initialiser les valeurs internes d'une instance dès sa construction (dès l'appel à new) .
En langage javascript/es2015 le constructeur se programme comme la méthode spéciale "**constructor**" (mot clef du langage) :

```
class Compte{
        constructor(numero,label,solde){
            this.numero = numero; this.label=label; this.solde=solde;
        }
        //...
}
```

```
var c1 = new Compte(1,"compte 1",100.0);
c1.crediter(50.0);  console.log("solde de c1: " + c1.solde);
```

NB: contrairement au langage java (où la surcharge y est permise) , il n'est pas possible d'écrire plusieurs versions du constructeur (ou d'une fonction de même nom) en javascript/es2015:

```
constructor(numero, libelle, soldeInitial){
                this.numero = numero;
                this.label = libelle;
                this.solde = soldeInitial;
        }

constructor(){
                this.=0;
                this.label="?";
                this.=0.0;
        }
```

Il faut donc quasi systématiquement utiliser la syntaxe = ***valeur_par_defaut*** sur les arguments d'un constructeur pour pouvoir créer une nouvelle instance en précisant plus ou moins d'informations lors de la construction :

```
class Compte{
        constructor(numero=0, libelle="?", soldeInitial=0.0){
                this.numero = numero;
                this.label = libelle;
                this.solde = soldeInitial;
        }//...
}
```

```
var c1 = new Compte(1,"compte 1",100.0);
var c2 = new Compte(2,"compte 2");
var c3 = new Compte(3);  var c4 = new Compte();
```

1.4. propriétés (speudo attributs - mots clefs "get" et "set")

Bien que "es2015" ne prenne pas en charge les mots clefs "~~publie~~" , "~~private~~" et "~~protecetd~~" au niveau des membres d'une classe et que toutes les méthodes définies soient publiques , il est néanmoins possible d'utiliser les mots clefs **get** et **set** de façon à ce qu'un **couple de méthodes "get xy()" et "set xy(...)" soit vu de l'extérieur comme une propriété (speudo-attribut "xy")** de la classe .

Attention : contrairement au langage java , il ne s'agit pas de convention de nom getXy() / setXy() mais de véritables mot clefs "get" et "set" à utiliser comme préfixe (avec un espace) .

```
class Compte{
        get decouvertAutorise(){
                return (this._decouvertAutorise!=undefined)?this._decouvertAutorise:0;
        }
        set decouvertAutorise(decouvertAutorise){
                this._decouvertAutorise = decouvertAutorise;
        }
//...  }
```

```
let c4=new Compte() ;
c4.decouvertAutorise = -300;
let decouvertAutorisePourC4 = c4.decouvertAutorise;
console.log("decouvertAutorisePourC4="+decouvertAutorisePourC4);
```

NB :

- il est possible de ne coder que le "get xy()" pour une propriété en lecture seule .
- il faut un nom différent (avec par exemple un "_" en plus) au niveau du nom de l'attribut interne préfixé par "this." car sinon il y a confusion entre attribut et propriété et cela mène à des boucles infinies .
- un "getter" peut éventuellement être supprimé via le mot clef ***delete*** (ex : delete obj.dernier)
- ***Object.defineProperty()*** permet (dans le cas pointus) de définir un "getter" par "méta-programmation" (ex : dans le cadre d'un framework ou d'une api générique).

En javascript es2015, le mot clef ***get*** sert surtout à définir **une propriété dont la valeur est calculée dynamiquement** :

```
var obj = {
        get dernier() {
                if (this.arrayXy.length > 0) {
                  return this.arrayXy[this.arrayXy.length - 1];
                }
                else {
                  return null;
                }
```

```
        },
        arrayXy: ["un","deux","trois"]
      }

console.log("dernier="+obj.dernier); // "trois"
```

1.5. mot clef "static" pour méthodes de classe

Le la même façon que dans beaucoup d'autres langages orientés objets (c++, java, …) , le mot clef ***static*** permet de définir des méthodes de classe (dont l'appel s'effectue avec le préfixe "NomDeClasse." plutôt que "instancePrecise." .

ES2015 ne permet pas d'utiliser static avec un attribut mais on peut utiliser "static" sur une propriété (avec mot clef get et éventuellement set). Dans ce cas la valeur de la propriété sera partagée par toutes les instances de la classe (et l'accès se fera via le préfixe "NomDeClasse.")

Exemple:

```
class CompteEpargne  {
        //...
        static get tauxInteret(){
                return CompteEpargne.prototype._tauxInteret;
        }
        static set tauxInteret(tauxInteret){
                CompteEpargne.prototype._tauxInteret=tauxInteret;
        }

        static get plafond(){
                return CompteEpargne.prototype._plafond;
        }
        static set plafond(plafond){
                CompteEpargne.prototype._plafond=plafond;
        }

        static methodeStatiqueUtilitaire(message){
                console.log(">>>" + message + "<<<");
        }
}
CompteEpargne.prototype._tauxInteret = 1.5 ;  //1.5% par defaut
CompteEpargne.prototype._plafond = 12000; //par defaut
```

```
let cEpargne897 = new CompteEpargne() ;
//cEpargne897.solde = 250.0; // instancePrecise.proprieteOrdinairePasStatique
console.log("taux interet courant=" + CompteEpargne.tauxInteret);//1.5
console.log("plafond initial=" + CompteEpargne.plafond);//12000
CompteEpargne.plafond = 10000;
let messagePlafond= "nouveau plafond=" + CompteEpargne.plafond;//10000
CompteEpargne.methodeStatiqueUtilitaire(messagePlafond);
```

NB :

- En cas d'héritage , une sous classe peut faire référence à une méthode statique de la classe parente via le mot préfixe super.
- Une méthode statique ne peut pas être invoquée avec le préfixe ~~this~~ .

1.6. héritage et valeurs par défaut pour arguments:

```
class Animal {
    constructor(theName="default animal name") {
       this.name= theName;
    }
    move(meters = 0) {
        console.log(this.name + " moved " + meters + "m.");
    }
}
```

```
class Snake extends Animal {
    constructor(name) { super(name); }
    move(meters = 5) {
        console.log("Slithering...");
        super.move(meters);
    }
}
```

```
class Horse extends Animal {
    constructor(name) { super(name); }
    move(meters = 45) {
        console.log("Galloping...");
        super.move(meters);
    }
}
```

```
var a = new Animal(); //var a = new Animal("animal");

var sam = new Snake("Sammy the Python");  //var sam = new Snake();

var tom = new Horse("Tommy the Palomino");

a.move() ; // default animal name moved 0m.
sam.move(); // Slithering… Sammy the Python moved 5m.

tom.move(34); //avec polymorphisme  (for  Horse)
 // Galloping… Tommy the Palomino moved 34m.
```

1.7. Object.assign()

Object.assign(obj,otherObject) ; permet (selon les cas) de :

- effectuer un clonage en mode "shallow copy" (copies des références vers propriétés)
- ajouter dynamiquement un complément
- ajouter le comportement d'une classe à un pur objet de données

Exemple d'objet original:

```
const subObj = { pa : "a1" , pb : "b1" };
const obj1 = { p1: 123 , p2 : 456 , p3 : "abc" , subObj : subObj };
```

Clonage imparfait (pas toujours en profondeur) avec Object.assign(clone,original)

```
var objCloneViaShallowCopy = {}
Object.assign(objCloneViaShallowCopy,obj1); //copy of property reference
console.log("clonage via assign / shallowCopy=" + JSON.stringify(objCloneViaShallowCopy));
// {"p1":123,"p2":456,"p3":"abc","subObj":{"pa":"a1","pb":"b1"}}
objCloneViaShallowCopy.subObj.pa="a2";
//modification à la fois sur objCloneViaShallowCopy.subObj et sur obj1.subObj
console.log("obj1" + JSON.stringify(obj1));
// {"p1":123,"p2":456,"p3":"abc","subObj":{"pa":"a2","pb":"b1"}}
objCloneViaShallowCopy.subObj.pa="a1"; //restituer ancienne valeur
```

Clonage en profondeur avec obj=JSON.parse(Json.stringify(original)) :

```
var obj = JSON.parse(JSON.stringify(obj1));//clonage en profondeur
console.log("clonage en profondeur=" + JSON.stringify(obj));
obj.subObj.pa="a2"; //modification que sur obj.subObj
console.log("obj1" + JSON.stringify(obj1));//obj1 inchangé ( "a1")
```

Ajout de données via Object.assign(obj, complément) :

```
Object.assign(obj, { p4: true , p5: "def" });
console.log("après assign complement=" + JSON.stringify(obj));
//{"p1":123,"p2":456,"p3":"abc","subObj":{"pa":"a2","pb":"b1"},"p4":true,"p5":"def"}
```

Ajout comportemental via obj=Object.assign(new MyClass(), obj) :

```
class Pp {
        constructor(p1=0,p2=0,p3=0){
                this.p1=p1; this.p2=p2; this.p3=p3;
        }

        sumOfP1P2P3(){
                return this.p1+this.p2+this.p3;
        }
}

const subObj = { pa : "a1" , pb : "b1" };
const obj1 = { p1: 123 , p2 : 456 , p3 : "abc" , subObj : subObj };
obj = JSON.parse(JSON.stringify(obj1));//réinitialisation du clone "obj"
if(!(obj instanceof Pp))
    console.log("obj is not instance of Pp , no sumOfP1P2P3() method");
//console.log("obj.sumOfP1P2P3()="+obj.sumOfP1P2P3()); not working
obj = Object.assign(new Pp(),obj);
if((obj instanceof Pp)) console.log("obj is instance of Pp , with sumOfP1P2P3() method");
console.log("obj.sumOfP1P2P3()="+obj.sumOfP1P2P3()); //ok : 579abc
```

Mixin set of additional methods with Object.assign(C1.prototype , mixinXyz) :

exemple :

```
class C1 {
        constructor(id=null,label="?"){
                this.id=id; this.label=label;
        }

        displayId(){
                console.log(`id=${this.id}`);
        }
}
```

```
let myMixin = {
        //mixin object = set of additional methods (without real inheritance):

        labelToUpperCase(){
                this.label = this.label.toUpperCase();
        } ,
        displayLabel(){
                console.log(`label=${this.label}`);
        }
}

let objet = new C1(1,"abc");
//objet.displayLabel();//not a method of C1
Object.assign(C1.prototype,myMixin); //ajouter méthodes de myMixin  à C1
objet.displayId();
objet.labelToUpperCase();
objet.displayLabel(); //ABC
let objetBis = new C1(2,"def");
objetBis.displayId();
objetBis.displayLabel();//def
```

1.8. Notions "orientée objet" qui ne sont pas gérées pas es2015

Les éléments "orientés objets" suivants ne sont pas pris en charge par un moteur ~~javascript/es2015~~ mais sont pris en charge par le langage "**typescript**" (".ts" à traduire en ".js" via babel ou "tsc") :

- classes et méthodes abstraites (mot clef "~~abstract~~")
- visibilités "~~public~~" , "~~private~~" , "~~protected~~"
- interfaces (mots clefs "~~interface~~" et "~~implements~~")
- "public" , "private" ou "protected" au niveau des paramètres d'un constructeur pour définir automatiquement certaines variables d'instances (attributs)
- ...

IX - Promise (es2015)

1. Promise (es2015)

1.1. L'enfer des "callback" (sans promesses) :

Beaucoup d'api javascript ont été conçues pour fonctionner en mode asynchrone (sans blocage).
Par exemple , pour séquentiellement saisir x, saisir y et calculer x+y , le code nécessaire qui serait très simple et très lisible en C/C++ ou java est assez complexe dans l'environnement node-js :

```
var stdin = process.stdin;
var stdout = process.stdout;

function ask(question, callback) {
        stdin.resume();
        stdout.write(question + ": ");
        stdin.once('data', function(data) {
                data = data.toString().trim();
                callback(data);
        });
}

//utilisation chaînée avec callbacks imbriquées:
ask("x", function(valX){
                var x=Number(valX);
                ask("y", function(valY){
                                var y=Number(valY);
                                var res=x+y ;
                                console.log("res = (x+y)=" +res);
                                process.exit();
                        });
});
```

1.2. Principe de fonctionnement des promesses (*Promise*)

Lorsque l'on déclenche via un appel de fonction un traitement asynchrone dont le résultat ne sera prêt/connu que dans le futur , on peut retourner immédiatement un objet de type "Promise" qui encapsule l'attente d'une réponse promise.
Le résultat promis qui sera récupéré en différé dans le temps est soit une réponse positive (promesse tenue) soit une réponse négative (erreur / promesse rompue) .

A l'intérieur de la fonction asynchrone appelée, on créer et retourne une promise via l'instruction
return new Promise((resolve,reject) => { if(...) resolve(...) else reject(...) ; }) ;
//où resolve et reject sont des noms logiques de callbacks appelées dans le futur
//pour transmettre l'issue positive ou négatif du traitement asynchrone .

A l'extérieur , l'appel s'effectue via la syntaxe
.then((resolvedValue)=>{ ....} , (rejectedValue) => { ... }) ;
ou bien

```
.then((resolvedValue)=>{ ....})
.catch((rejectedValue) => { ... } ) ;
```

NB: Si à l'intérieur d'un .then(()=>{...}) on appelle et retourne une fonction asynchrone retournant à son tour une autre "Promise" , on peut alors enchaîner d'une manière lisible une séquence d'appel à d'autres .then() qui seront alors exécutés les uns après les autres au fur et à mesure de la résolution des promesses asynchrones :

```
appelAsynchrone1RetournantPromesse1(...)
.then((resPromesse1)=>{ .... ; return appelAsynchrone2RetournantPromesse2(....);})
.then((resPromesse2)=>{ .... ; return appelAsynchrone2RetournantPromesse3(....);})
.then((resPromesse3)=>{ .... ; })
.catch((premiereErreurPromesse1ou2ou3)=>{...});
```

NB : avant d'exister en version normalisée "es2015" , les "Promises" avaient été prises en charge via l'ancienne bibliothèque "**q**" *(var deferred = Q.defer(); .... deferred.resolve(data); ... return deferred.promise;) avec même utilisation .then(...).then(...).catch(...) .*
Les "Promises" étaient déjà beaucoup utilisées à l'époque de es5/angular-js (avant 2015 et es6/es2015) .

Exemple (avec Promise/es2015) :

```
var stdin = process.stdin;
var stdout = process.stdout;

function ask_(question) {
        return new Promise ((resolve,reject)=> {
                stdin.resume();
                stdout.write(question + ": ");
                stdin.once('data', function(data) {
                        data = data.toString().trim();
                        if(data=="fin")
                                reject("end/reject");
                        else
                          resolve(data);
                });
        });
}

var x,y,z;
//calcul (x+y)*z après enchaînement lisible (proche séquentiel) de "saisir x" , "saisir y", "saisir z":
ask_("x")
.then((valX)=>{ x=Number(valX); return ask_("y");})
.then((valY)=> { y=Number(valY); let res=x+y ;
                                console.log("(x+y)=" +res);
                                return ask_("z");
                        })
.then((valZ)=> { z=Number(valZ); let res=(x+y)*z ;
                                console.log("(x+y)*z=" +res);
                                process.exit();
                        })
.catch((err)=>{console.log(err);process.exit();});
```

1.3. Autre exemple simple (sans et avec "Promise"):

```
function strDateTime() {
        return (new Date()).toLocaleString();
}

const affDiffere = () => {
        setTimeout (()=> {console.log("after 2000 ms " + strDateTime());} , 2000);
};
```

Version sans "Promise" avec callbacks imbriquées :

//NB : via .setTimeout(...., delay) et return { responseJsData } ; on simule ici une récupération de //données via un appel asynchrone vers une base de données ou un WS REST.

```
const getUserInCb =
  (cbWithName) => {
        setTimeout (()=> { cbWithName({ name : "toto" });} , 2000);
  };

const getAddressFromNameInCb =
 (name , cbWithAddress)  => {
        setTimeout (()=> {  cbWithAddress({ adr : "75000 Paris for name="+name });}
                        , 1500);
  };

console.log("debut :" + strDateTime()  );
affDiffere();


getUserInCb(
   (user) => {
   console.log("username=" + user.name);
   getAddressFromNameInCb(user.name,
      (address) => { console.log("address=" + address.adr );  }
            );
   }
   );

console.log("suite :" + strDateTime()  );
```

résultats:

debut :2019-4-23 16:46:24
suite :2019-4-23 16:46:24
after 2000 ms 2019-4-23 16:46:26
username=toto

address=75000 Paris for name=toto

Même exemple avec "Promise es6/es2015" :

```
function getUserFromIdAsPromise(id){
  return new Promise (
  (resolveCbWithName,rejectCb) => {
        setTimeout (()=> { if(id) resolveCbWithName({ name : "toto" });
                           else  rejectCb("id should not be null!");
                    }
                , 2000);
  });
}

function getAddressFromNameAsPromise(name){
  return new Promise (
  (resolveCbWithAddress) => {
        setTimeout (()=> { resolveCbWithAddress({ adr : "75000 Paris for name="
                                                        +name });} , 1500);
  });
}

console.log("debut :" + strDateTime()  );  affDiffere();

getUserFromIdAsPromise(1)
//getUserFromIdAsPromise(null)
   .then( (user) => {   console.log("username=" + user.name);
                        //returning new Promise for next then() :
                        return getAddressFromNameAsPromise(user.name);
                        })
   .then( (address) => { console.log("address=" + address.adr ); } )
   .catch(error => { console.log("error:" + error); } );

console.log("suite :" + strDateTime()  );
```

Résultats avec id=1 :
debut :2019-4-23 17:18:44
suite :2019-4-23 17:18:44
after 2000 ms 2019-4-23 17:18:46
username=toto
address=75000 Paris for name=toto

Résultats avec id=null :
debut :2019-4-23 17:33:17
suite :2019-4-23 17:33:17
after 2000 ms 2019-4-23 17:33:19
error:id should not be null!

L'exemple ci-dessus montre que :

- l'on peut nommer comme on le souhaite les callbacks "resolve" et "reject" . Ce qui permet quelquefois de rendre le code plus intelligible

- la callback "reject" est facultative

1.4. Propriétés des "Promises"

```
fairePremiereChose()
.then(result1 => faireSecondeChose(result1))
.then(result2 => faireTroisiemeChose(result2))
.then(finalResult3 => {
  console.log('Résultat final : ' + finalResult3);
})
.catch(failureCallback);
```

où

```
(resultatAppelPrecedent) => faireNouvelleChose(resultatAppelPrecedent)
```

est synonyme de

```
(resultatAppelPrecedent) => { return faireNouvelleChose(resultatAppelPrecedent) ; }.
```

Si aucun catch , il est éventuellement possible de traiter l'événement "***unhandledrejection***"

```
window_or_worker.addEventListener("unhandledrejection", event => {
 // Examiner la ou les promesse(s) qui posent problème en debug
  // Nettoyer ce qui doit l'être quand ça se produit en réel
}, false);
```

Dans des cas "ultra simples" ou "triviaux" , on pourra éventuellement créer et retourner ***une promesse à résolution ou rejet immédiat*** avec une syntaxe de ce type :

argValue => **Promise.resolve**(argValue);

//version abrégée de argValue => new Promise((resolve)=>resolve(argValue))

errMsg => **Promise.reject**(errMsg);

1.5. Compositions (all , race, ...)

On peut déclencher des traitements asynchrones en parallèle et attendre que tout soit fini pour analyser globalement les résultats :

```
Promise.all([func1(), func2(), func3()])
.then(([resultat1, resultat2, resultat3]) => { /* utilisation de resultat1/2/3 */ });
```

La variante ci-après permet de déclencher des traitements asynchrones en parallèle et attendre que le premier résultat (retourné par la fonction asynchrone la plus rapide) :

```
Promise.race([func1(), func2()])
        .then( (firstReturnedValue)  => {  console.log(firstReturnedValue);});
```

Exemple :

```
function getUppercaseDataAfterDelay(data, delay){
  return new Promise (
  (resolve) => {
        setTimeout (()=> { resolve(data.toUpperCase());}, delay);
  });
}

Promise.all( [ getUppercaseDataAfterDelay("abc",2000) ,
                getUppercaseDataAfterDelay("def",1500)  ] )
     .then ( ([ res1 , res2 ]) => { console.log(">>"+res1+"--"+res2+"<<" ); });

Promise.race( [ getUppercaseDataAfterDelay("abc",2000) ,
                  getUppercaseDataAfterDelay("def",1500)  ] )
     .then ( (firstResult) => { console.log(">>>"+firstResult+"<<<" ); } );
```

>>>DEF<<<

>>ABC--DEF<<

1.6. Appel ajax via api fetch et Promise

L'api "**fetch**" supportée par certains navigateurs modernes utilise en interne l'api "Promise" de façon à déclencher des appels HTTP/ajax en mode GET ou POST ou autres.

Exemple en mode GET :

```
function myGenericJsGetFetchData(url){
  return new Promise((resolveWithJsData,reject)=>{
            fetch(url)
           .then( (response) => {
                    if (response.status !== 200) {
                              var errString = 'Problem. Status Code: ' + response.status;
                              console.log(errString);  reject(errString);   return;
                              }
                              // Examine the text in the response :
                              response.json().then(function(data) {
                                        resolveWithJsData(data);
                              })
             })
          .catch((err) =>{ console.log('Fetch Error :-S', err);   reject(err); });
         });
}
```

```
myGenericJsGetFetchData("./rest/produit/" + numProd)
        .then( (data) => {    console.log(data);
                    var jsonString = JSON.stringify(data);
                    document.querySelector('#resProd').innerHTML = jsonString;
                  })
        .catch((err) => { console.log(err); });
```

Exemple partiel en mode POST :

```
        fetch(url,{ method: 'POST' ,
                    headers: {
                              'Accept': 'application/json',
                              'Content-Type': 'application/json'
                             },
                   body : JSON.stringify(jsObj)
                  } )
           .then( (response) => {
                    if (response.status !== 200) {
                              var errString = 'Problem. Status Code: ' + response.status;
                              console.log(errString);     return;
                              }
                              response.json().then(function(data) {console.log(JSON.stringify(data));})
             })
          .catch((err) =>{ console.log('Fetch Error :-S', err); });
```

X - Modules es2015

1. Modules (es2015)

1.1. Types de modules (cjs , amd , es2015 , umd , ...)

Beaucoup de technologies javascript modernes s'exécutent dans un environnement prenant en charge des modules (bien délimités) de code (avec import/export) . Le développement d'une application "Angular2+" s'effectue à fond dans ce contexte.

Les principales technologies de "modules javascript" sont les suivantes :

- **CommonJS (cjs)** – modules "synchrones" , **syntaxe** "var xyz = **requires**('xyz')"
  NB : node (nodeJs) utilise partiellement les idées et syntaxes de CommonJS .
- **AMD** (**A**synchronous **M**odule **D**efinition) avec chargements asynchrones
- **ES2015 Modules** : syntaxiquement standardisé , mots clef "import {…} from '…' " et export pour la gestion dynamique des modules(possibilité de générer des bundles (es5 ou es6) regroupant plusieurs modules statiquement assemblés ensemble).
- **SystemJS** (très récent et pas encore complètement stabilisé) supporte en théorie les 3 technologies de modules précédentes (cjs , amd, es2015) . SystemJS nécessite certains paramétrages (quelquefois complexes) et s'utilise assez souvent avec gulp .
  --> Attention : SystemJS s'est révélé assez instable et n'est pas toujours ce qui y a de mieux .

Il existe aussi les **formats de modules suivants** :

- **umd** (**u**niversal **m**odule **d**efinition) – *fichiers xyz.umd.js*
- **iife** (**i**mmediately-**i**nvoked **f**unction **e**xpression) – *fonctions anonymes auto-exécutées*

1.2. Modules "es6/es2015" et organisation en fichiers

Les modules ES6 :

- ont une syntaxe simple et sont basés sur le découpage en fichiers (un module = un fichier),
- sont automatiquement en mode « strict » (rigoureux),
- offrent un support pour un chargement asynchrone et permet de générer des bundles "statiques" via rollup ou webpack .

Les modules doivent exposer leurs variables et méthodes de façon explicite. On dispose donc des deux mots clés :

- **export** : pour exporter tout ce qui doit être accessible en dehors du module,
- **import** : pour importer tout ce qui doit être utilisé dans le module (et qui est donc exporté par un autre module).

1.3. Exemple avec chargement direct depuis navigateur récent:

math-util.js

```
export function additionner(x , y) {
  return x + y;
}
export function multiplier(x, y) {
  return x * y;
}
```

ou bien

```
function additionner(x , y) {
  return x + y;
}
function mult(x, y) {
  return x * y;
}
export { additionner, mult as multiplier };
```

dom-util.js

```
export class DomUtil {
	static displayInDiv(divId,message){
		document.querySelector('#'+divId).innerHTML = message;
	}

	static multilineMessage(...args){
		//NB: la syntaxe ... permet de récupérer tous (ou bien les derniers) arguments (en nombre variable)
		//sous forme de tableau . Cette syntaxe est permise en mode "strict" alors que la
		//syntaxe Dom.multilineMessage.arguments est interdite en mode "strict" (dans module es6)
		let nb_arg=args.length;
		let messages=null;
		 if(nb_arg>=1) messages=args[0];
		 for(let i=1;i<nb_arg;i++)
			messages+="<br/>"+args[i];
		return messages;
	}
```

```
}
```

main.js

```
import { additionner as add, multiplier } from "./math-util.js";
import { DomUtil }  from "./dom-util.js";

function carre(x){
  return multiplier(x,x) ;
}
/*
var msg1 = "Le carre de 5 est " + carre(5);   console.log(msg1);
var msg2 = "4 * 3 vaut " + multiplier(4, 3);  console.log(msg2);
var msg3 = "5 + 6 vaut " + add(5, 6); console.log(msg3);
document.querySelector('#divA').innerHTML = msg1 + "<br/>" + msg2 + "<br/>" + msg3;
*/
DomUtil.displayInDiv('divA',
            DomUtil.multilineMessage(
                                        "Le carre de 5 est " + carre(5),
                                        "4 * 3 vaut " + multiplier(4, 3),
                                        "5 + 6 vaut " + add(5, 6)
                                    ));
```

```
<html>
    <body>
    <script type="module" src="main.js"></script>
    <div id="divA"></div>
    </body>
</html>
```

Attention :

- type="module" est indispensable mais n'est supporté que par les navigateurs récents
- la page html et les modules javascripts doivent être téléchargés via http (par exemple via http://localhost:3000/ et lite-server) .

1.4. default export (one per module)

xy.js

```
export function mult(x, y) {
   return x * y;
 }

 //export default function_or_object_or_class (ONE PER MODULE)
 export default {
        name : "xy",
        features : { x : 1 , y: 3 }
 }
```

main.js

```
import xy , { mult }  from "./xy.js";
...
let msg = xy.name + "--" + JSON.stringify(xy.features) + "--" + mult(3,4);
```

1.5. Agrégation de modules

mod1.js

```
export function f1(msg) {  return "*"+msg    }
export function f2(msg) {      return "**"+msg;    }
export function f2bis(msg) {      return "***"+msg;    }
```

mod2.js

```
export function f3(msg) {      return "#"+msg    }
export function f4(msg) {      return "##"+msg;    }
```

mod1-2.js

```
//agrégation de modules : mod1-2 = mod1 + mod2
 //importer certains éléments du module "mod2" et les ré-exporter:
 export { f1, f2 } from "./mod1.js"
 //importer tous les éléments du module "mod2" et les ré-exporter tous :
 export * from "./mod2.js"
```

main.js

```
import { f1 , f2 , f3 , f4  } from "./mod1-2.js";
let f_msg=f1('abc')+'-'+f2('abc')+'-'+f3('abc')+'-'+f4('abc');
```

ou bien

```
import * as f from "./mod1-2.js";
let f_msg=f.f1('abc')+'-'+f.f2('abc')+'-'+f.f3('abc')+'-'+f.f4('abc');
```

1.6. Technologies de "packaging" (webpack , rollup, ...) et autres

De façon à éviter le téléchargement d'une multitude de petits fichiers , il est possible de créer des gros paquets appelés "**bundles**" .

Les principales technologies de packaging "javascript" sont les suivantes :

- **webpack** (mature et supportant les modules "**csj**" , "amd" , …)
- **rollup** (récent et pour modules "**es2015**") . Rollup est une fusion intelligente de n fichiers en 1 (remplacement des imports/exports par sous contenu ajustés , prise en compte de la chaîne des dépendances en partant par exemple de main.js)
- **SystemJs-builder** (technologie assez récente et un peu moins mature)

Autres technologies annexes (proches) :

- **browserify** : technologie déjà assez ancienne permettant de faire fonctionner un module "nodeJs" dans un navigateur après transformation.
- **babel** : transformation (par exemple es2015 vers es5)
- **uglify** : minification (enlever tous les espaces et commentaires inutiles, simplifier noms des variables, …) → code beaucoup plus compact (xyz.min.js) .
- **gzip** : compression .Les fichiers bundlexy.min.js.gz sont automatiquement traités par quasiment tous les serveurs HTTP et les navigateurs : décompression automatique après transfert réseau).

1.7. Packaging web via rollup / es2015 et npm

L'un des principaux atouts de la structure des "modules es2015" tient dans les imports statiques et précis qui peuvent ainsi être analysés pour une génération optimisée des bundles à déployer en production.

Au lieu d'écrire var/const xyModule = requires('xyModule') comme en "cjs" ,
la syntaxe plus précise de es2015 permet d'écrire
import { Composant1 , … , ComposantN } from 'xyModule' .

Ainsi l'optimisation dite "**Tree-Shaking**" permet d'exclure tous les composants jamais utilisés de certaines librairies et la taille des bundles générés est plus petite.

La technologie de packaging "rollup" qui est spécialisée "es2015" peut ainsi exploiter cette optimisation via la chaîne de transformation suivante :

main[**.es2015].js**
with ***import*** → **rollup** → myBundle**.es2015.js** → myBundle**.es5.js**
subModuleXx[**.es2015].js**
subModuleYy[**.es2015].js** (*)

(*) es2015-to-es5 via ***babel*** *(presets : es2015 ou [env])* ou autres
permet d'obtenir un bundle interprétable par quasiment tous les navigateurs.

```
npm install -g rollup
npm init
npm install --save-dev babel-cli
npm install --save-dev babel-preset-env
```

package.json

```
{
 "name": "with-modules-and-rollup",
 "version": "1.0.0",
 "description": "",
 "main": "index.js",
 "scripts": {
     "es6bundle-to-es5" : "babel dist/build-es2015 -d dist/build-es5",
     "build" : "rollup --config rollup.config.js && babel dist/build-es2015 -d dist/build-es5"
 },
 "devDependencies": {
   "babel-cli": "^6.26.0",
   "babel-preset-env": "^1.7.0"
 }
}
```

```
rollup --config rollup.config.js
```

ou bien

```
npm run build
```

rollup.config.js

```
export default {
  input: 'src/main.js',
  output : {
        file: 'dist/build-es2015/main-bundle.js',
        format: 'iife'
  }
};
```

.babelrc

```
{
  "presets": ["env"]
}
```

Dans la configuration *rollup.config.js* ,

input :.../main.js correspond au point d'entrée (.js) autrement dit la racine d'un arbre import/export entre différents fichiers (es6) qui seront analysés et gérés par rollup.

Il peut quelquefois y avoir plusieurs input/output dans le fichier rollup.config.js.

le **format** peut être "**cjs**" pour une future interprétation via node/nodeJs

ou "**iife**" pour une future interprétation via html/js (navigateur)

1.8. *Packaging web via webpack*

--> voir chapitre ou annexe suivante ...

XI - async/await (es2017)

1. async/await (es2017)

async et **await** sont de nouveaux mots clefs de es2017 (inspiré de typescript et de l'univers .net/Microsoft) .
Ces nouveaux mots clefs permettent de simplifier les enchaînement de fonctions asynchrones en générant et attendant automatiquement des promesses ("Promise de es2015") .

1.1. Principes async/await :

- ***return resultat*** dans une fonction ***async***
  est (depuis es2017) équivalent à ***return new Promise.resolve(resultat);***

- **throw new Error('erreur')** dans une fonction ***async***
  est (depuis es2017) équivalent à ***return new Promise.reject(new Error('erreur'));***

- **await** permet d'**attendre la résolution d'une promesse (liée à un sous appel asynchrone) et de récupérer la valeur dans une variable** *(équivalent de .then(...) automatique)* .
  NB : ***await ne peut être utilisé qu'au sein d'une fonction préfixée par async*** .

- au sein d'une fonction préfixée par async, un bloc **try { ... } catch {...}** ordinaire permet de récupérer aussi bien certaines exceptions synchrones que certains échecs liés à des promesses non tenues par des sous appels asynchrones déclenchés via *await* .
  Autrement dit : **try** { **await** *appel_async1(...)* ;
  await *appel_async2(...)* } **catch** {...}
  peut remplacer *appel_async1().then(() =>... ; return appel_async2(...) ;)*
  .then(()=>...)
  .catch((e)=>...) ;

1.2. Exemple 1 (async/await):

Préliminaire (avec "Promise" ordinaire) :

```
function strDateTime() {
        return (new Date()).toLocaleString();
}


function myGenericTimeoutPromise(cbToDelay,delay){
 return new Promise (
 (resolve) => {
        setTimeout (()=> { resolve(cbToDelay());}
                , delay);
 });
}
```

```
const affDiffere = () => {
        myGenericTimeoutPromise(()=> {return("after 2000 ms " + strDateTime());} , 2000)
        .then((message)=> {console.log("ok - " + message); });
};

console.log("debut :" + strDateTime()  ); //debut :2019-4-30 15:50:04
affDiffere(); //ok - after 2000 ms 2019-4-30 15:50:06
```

```
async function getUserFromIdAsAutomaticPromise(id){
        const user =  await myGenericTimeoutPromise(
                        ()=> { return { name : "toto" };}, 2000);
        if(id) return user; //as Promise.resolve(user)
                else throw new Error("id should not be null!");//as Promise.reject(...)
}
```

```
async function getAddressFromNameAsAutomaticPromise(name){
        const address =  await myGenericTimeoutPromise(
                        ()=> { return { adr : "75000 Paris for name="+name }; } , 1500);
        return address; //as Promise.resolve(address)
}
```

```
function appelsClassiquesPromises(){
getUserFromIdAsAutomaticPromise(1)
//getUserFromIdAsAutomaticPromise(null)
  .then( (user) => {   console.log("username=" + user.name);
                       //returning new Promise for next then() :
                       return getAddressFromNameAsAutomaticPromise(user.name);
                       })
  .then( (address) => { console.log("address=" + address.adr ); } )
  .catch( (err) => { console.log("my error:" + err.message); } );
}

appelsClassiquesPromises();
```

```
async function appelsViaAwait(){
 try{
        const user = await getUserFromIdAsAutomaticPromise(1);
        //const user = await getUserFromIdAsAutomaticPromise(null);
        console.log("username=" + user.name);
        const address = await getAddressFromNameAsAutomaticPromise(user.name);
        console.log("address=" + address.adr );
 }
 catch(err){
    console.log("my error:" + err);
 }
}
appelsViaAwait()
//.catch( (err) => { console.log("my error:" + err); } );
```

==>

username=toto (après 2s)

address=75000 Paris for name=toto (encore après 1.5s)

1.3. Exemple 2 (async/await) :

```
var stdin = process.stdin;
var stdout = process.stdout;

function ask_(question) {
        return new Promise ((resolve,reject)=> {
                stdin.resume();
                stdout.write(question + ": ");
                stdin.once('data', function(data) {
                        data = data.toString().trim();
                        if(data=="fin")
                                reject("end/reject");
                        else
                          resolve(data);
                });
        });
}

async function ask_and_compute_x_plus_y(){
        try{
                let x,y;
                const valX = await ask_("x"); x=Number(valX);
                const valY = await ask_("y"); y=Number(valY);
                let xPlusY=x+y ;console.log("(x+y)=" +xPlusY);
                return xPlusY;
        }
        catch(e){
                console.log(e);
                throw new Error("xPlusY-error:"+e);
        }
}


async function x_plus_y_mult_z(){
        try{    /*
                const valX = await ask_("x"); let x=Number(valX);
                const valY = await ask_("y"); let y=Number(valY);
                let xPlusY=x+y ;console.log("(x+y)=" +xPlusY);
                */
                const xPlusY = await ask_and_compute_x_plus_y();
                const valZ = await ask_("z"); const z=Number(valZ);
                let res=xPlusY * z ;console.log("(x+y)*z=" +res);
        }
        catch(e){
                console.log(e);
        }
   process.exit();
}

x_plus_y_mult_z();
```

```
==>
x: 5
y: 6
(x+y)=11
z: 3
(x+y)*z=33
```

Equivalent sans async/await avec .then().then.catch() de es6/es2015 :

```
var x,y,z; //(x+y)*z
ask_("x")
.then((valX)=>{ x=Number(valX); return ask_("y");})
.then((valY)=> { y=Number(valY); let res=x+y ;
                                console.log("(x+y)=" +res);
                                return ask_("z");
                            })
.then((valZ)=> { z=Number(valZ); let res=(x+y)*z ;
                                console.log("(x+y)*z=" +res);
                                process.exit();
                            })
.catch((err)=>{console.log(err);process.exit();});
```

1.4. Combinaison de async/await avec Promise.all et Promise.race

```
function getUppercaseDataAfterDelay(data, delay){
  return new Promise (
  (resolve) => {
          setTimeout (()=> { resolve(data.toUpperCase());}, delay);
  });
}

async function test_await_Promise_all_and_race(){

  //attendre les résultats de 2 traitements asynchrones lancés en parallèle :
   const [ res1 , res2 ] = await Promise.all( [ getUppercaseDataAfterDelay("abc",2000) ,
                                                getUppercaseDataAfterDelay("def",1500)  ] );
   console.log(">>"+res1+"--"+res2+"<<" );

  //attendre le premier résultat (le plus rapidement retourné)
  // de 2 traitements asynchrones lancés en parallèle :
   const firstResult = await  Promise.race( [ getUppercaseDataAfterDelay("abc",2000) ,
                                              getUppercaseDataAfterDelay("def",1500)  ] );
   console.log(">>>"+firstResult+"<<<" );
}
test_await_Promise_all_and_race();
```

```
==>
>>ABC—DEF<<  (2 s après)
>>>DEF<<< (1.s après)
```

...

XII - Aspects divers et avances de es2015/es6

1. Aspects divers et avancés (es2015)

Attention :

Les éléments exposés dans ce chapitre sont complexes et ne sont utiles que dans certains cas très pointus . C'est pour les développeurs déjà bien expérimentés en javascript (ayant envie de programmer une librairie de code réutilisable du genre "mini framework xyz" par exemple).

1.1. WeakMap et WeakSet

Rappel : Map et Set sont de nouvelles structures de données introduites par la version es2015.

Par rapport à un simple objet javascript (déjà en interne géré comme une map entre noms et valeurs de propriétés) , une Map (de es2015) peut éventuellement comporter des clefs de types quelconques (pas obligatoirement de type string) .

Les variantes "***WeakMap***" et "***WeakSet***" (de es2015) apportent :
- beaucoup de restrictions (pas d'itération possible , moins de méthodes disponibles)
- une gestion différente de la mémoire (weak-reference permettant d'éviter quelquefois des fuites de mémoire)

```
//non primitive key for WeakMap and non primitiveValue for WeakSet
class MyObjectClass {
        constructor(v){ this.v = v; this.V = v.toUpperCase(); }
}
```

Une "WeakMap" ne peut comporter que des clefs de type "référence sur objet" (les types "primitifs sont interdits) .

Si après avoir ajouter par exemple 10 entrées de type

[copieDeRéférenceSurObjetJouantRoleDeClef , valeur] ,

du code externe à la "WeakMap" supprime (par exemple via un ***delete***) une référence (copiée) sur un des objets jouant de rôle de Clef , alors cet objet ne sera référencé que par une référence faible (weak) interne à la map .

Le ramasse-miettes (garbage collector) du moteur javascript va alors considérer que cet objet n'a plus de référence forte/ordinaire pointant vers lui et va alors supprimer l'objet clef et va indirectement supprimer l'entrée associée dans la "WeakMap" qui comportera alors un élément de moins.

```
// Weak Maps
// Weak Maps (!!! with no .size , no .forEach)
//NB: les éléments stockés dans une weakMap (dont les clefs sont obligatoirement des références
//sur des objets) ne seront conservés que si il existe encore une autre référence (externe)
//sur la même valeur "objet" d'une clef .
//Autrement dit la référence constituée par la clef d'une entrée d'une WeakMap est considérée
//comme faible et ne compte pas dans la logique de fonctionnement du "garbage collector" .
var wm = new WeakMap();
console.objKey1 = new MyObjectClass("key1");
console.objKey2 = new MyObjectClass("key2");
wm.set(console.objKey1,"val1");
wm.set(console.objKey2,"val2");
console.log("in weakMap , for console.objKey1 , value is " + wm.get(console.objKey1));
console.log("in weakMap , for console.objKey2 , value is " + wm.get(console.objKey2));
delete console.objKey2;
console.log("after delete console.objKey2 , in weakMap , for console.objKey1 , value is " +
wm.get(console.objKey1));
console.log("after delete console.objKey2 , in weakMap , for console.objKey2 , value is " +
wm.get(console.objKey2));
if(wm.has(console.objKey1))
        console.log("weakMap wm comporte encore une valeur associée à console.objKey1");
if(!wm.has(console.objKey2))
        console.log("weakMap wm ne comporte plus de valeur associée à console.objKey2");
```

```
// Weak Sets
// Weak Sets is not iterable and not very useful !!!!
var ws = new WeakSet();
console.obj1=new MyObjectClass("obj1");
console.obj2=new MyObjectClass("obj2");
ws.add(console.obj1);
ws.add(console.obj2);
delete console.obj2;
if(ws.has(console.obj1))
        console.log("weakSet ws comporte encore console.obj1");
if(!ws.has(console.obj2))
        console.log("weakSet ws ne comporte plus console.obj2");
```

1.2. nouvelles méthodes es6 sur Array, String, Number, Math

```
Number.EPSILON
Number.isInteger(Infinity) // false


Math.hypot(3, 4) // 5


"abcde".includes("cd") // true
"abc".repeat(3) // "abcabcabc"

Array.from(document.querySelectorAll("*")) // Returns a real Array
Array.of(1, 2, 3) // Similar to new Array(...), but without special one-arg behavior
[0, 0, 0].fill(7, 1) // [0,7,7]
[1,2,3].findIndex(x => x == 2) // 1
["a", "b", "c"].entries() // iterator [0, "a"], [1,"b"], [2,"c"]
["a", "b", "c"].keys() // iterator 0, 1, 2
["a", "b", "c"].values() // iterator "a", "b", "c"

Object.assign(Point, { origin: new Point(0,0) })
```

1.3. Symbols (clefs uniques)

```
const symbol1 = Symbol();
const symbol2 = Symbol(42);
const symbol3 = Symbol('foo');

console.log(typeof symbol1);
// expected output: "symbol"

console.log(symbol3.toString());
// expected output: "Symbol(foo)"

console.log(Symbol('foo') === Symbol('foo'));
// expected output: false
```

Utilisation classique n° 1 : clef pour propriété (privée ou ...)

```
let ageKey=Symbol('age'); //as speudo private key/property
let sizeKey=Symbol('size'); //as speudo private key/property
class Person {
        constructor(nom=null, age = 0 , size=0){
           this.nom=nom;
                this[ageKey]=age;
                this[sizeKey]=size;
        }

        get age(){
```

```
		return this[ageKey];
	}

	set age(newAge){
		if(newAge>=0)
			this[ageKey]=newAge;
	}

	get size(){
		return this[sizeKey];
	}

	set size(newSize){
		if(newSize>=0)
			this[sizeKey]=newSize;
	}

	logSymbolProperties(){
		for(let key of Object.getOwnPropertySymbols(this)){
			console.log(key.toString()+"_"+this[key]);
		}
	}
}

let p1 = new Person("toto",30);
console.log(p1, JSON.stringify(p1) , p1.age);
p1.nom="Toto"; p1.age=40; p1.size=160;
console.log(p1, JSON.stringify(p1) , p1.age);
p1.age=-5; //no effect , newAge invalid
p1.size=-23;
console.log(p1, JSON.stringify(p1) , p1.age);
p1.logSymbolProperties();
```

==>

```
Person { nom: 'toto', [Symbol(age)]: 30, [Symbol(size)]: 0 } '{"nom":"toto"}' 30
Person { nom: 'Toto', [Symbol(age)]: 40, [Symbol(size)]: 160 } '{"nom":"Toto"}' 40
Person { nom: 'Toto', [Symbol(age)]: 40, [Symbol(size)]: 160 } '{"nom":"Toto"}' 40
Symbol(age)_40
Symbol(size)_160
```

Utilisation classique n° 2 : constantes liées à certains concepts

`Exemples` : (couleurs , ...)

const COLOR_RED = Symbol('Red');

1.4. itérateurs et générateurs

Un itérateur est un objet technique de bas niveau ayant les principales caractéristiques suivantes :

- méthode **next**() pour itérer
- comporte une propriété **.value** (valeur quelconque du i ème élément)

- comporte une propriété booléene **.done** (true si fin de parcours/itération)
- prédéfini sur beaucoup de structure de données (String , Set, Map , Array , ...)
- utilisé en interne de façon transparente par la nouvelle boucle **for(let e of collection)** de es6/es2015 .
- utilisé en interne de façon transparente par la syntaxe **...iterateur** au sein d'un dernier paramètre d'un appel de fonction *(rest parameters **...**)* ou bien d'un paquets d'éléments d'un tableau *(spead operator **...**)*
- correspond à la fonction prototype **[Symbol.iterator]** d'une structure de données

Exemple : itérateur prédéfini sur "String" :

```
var strAbc = "abc";
console.log(typeof strAbc[Symbol.iterator]); // "function"

let itStrAbc = strAbc[Symbol.iterator]();
console.log(itStrAbc.next());  // { value: "a", done: false }
console.log(itStrAbc.next());  // { value: "b", done: false }
console.log(itStrAbc.next());  // { value: "c", done: false }
console.log(itStrAbc.next());  // { value: undefined, done: true }

let itAbc = strAbc[Symbol.iterator]();
var tabAbc = [ 'a', 'b' , 'c' ];
//let itAbc = tabAbc[Symbol.iterator]() ;
/*
let loopItem = null;
while((loopItem=itAbc.next()) && !loopItem.done) {
        console.log(loopItem.value);
}*/
for(let eltOfAbc of itAbc){
        console.log(">"+eltOfAbc);
} // >a  >b  >c
```

Construction d'un nouvel itérable élémentaire avec "function*" et "yield" :

```
var monIterable = {};
monIterable[Symbol.iterator] = function* () {
   yield 'e1';
   yield 'e2'; //yield signifie "rendre , produire , donner , générer , ..."
   yield 'e3';
};
//NB: String, Array, TypedArray, Map et Set sont des itérables natifs
//car les prototypes de chacun ont tous une méthode Symbol.iterator.

for(let elt of monIterable){
        console.log(">>"+elt);
}//>>e1  >>e2  >>e3

var myArray1 = [ 'e0' , ...monIterable , 'e4' ,'e5' ];
var myArray2 = [  ...monIterable  ];
console.log(myArray1); //[ 'e0', 'e1', 'e2', 'e3', 'e4', 'e5' ]
console.log(myArray2); //[ 'e1', 'e2', 'e3' ]
```

```
//Fonction génératrice élémentaire avec syntaxe "function*" et "yield" :
function* idMaker(){
  var index = 0;
```

```
 while(index<10)
   yield index++; //yield retourne la valeur et se met en pause (attente du futur appel)
}
//NB: le fait que la fonction génératrice (function*) soit prévue pour être appelée
//plusieurs fois via un itérateur et que yield établisse automatiquement une attente
//du prochain appel correspond à une fonctionnalité très spéciale du langage es6/es2015
//appelée PROTOCOLE d'itération .

//itérable1 basé sur générateur:
var genIt1 = idMaker();
console.log(genIt1.next().value); // 0
console.log(genIt1.next().value); // 1
console.log(genIt1.next().value); // 2
console.log("-------------");
//itérable2 basé sur même générateur:
var genIt2 = idMaker();
console.log(genIt2.next().value); // 0
console.log(genIt2.next().value); // 1
console.log("-------------");
//itérable3 basé sur même générateur:
var genIt3 = idMaker();
var myArray3 = [ ...genIt3 ];
console.log(myArray3);//[ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

NB: Un générateur peut éventuellement être codé comme une méthode spéciale d'un objet littéral ou bien d'une classe :

```
class MyClass {
//or const obj = {

     * generatorMethod() { ···
     }
 }
```

Exemple :

```
class MyBasicFifo{
        constructor(){ this.internalArray = [];
                        this[Symbol.iterator]=this.fifoIteratorGenerator;
        }

        pop(){ if(this.internalArray.length>0)
                        return this.internalArray.pop();
        }

        push(elt){ this.internalArray.push(elt);
                    let taille=this.internalArray.length;
                    for(let i=taille-1;i>0;i--){
                        this.internalArray[i]=this.internalArray[i-1];
                    }
                   this.internalArray[0]=elt;
        }

    * fifoIteratorGenerator() {
        var index = this.internalArray.length -1;
```

```
        while(index>=0)
                yield this.internalArray[index--];
    }
}
```

```
let fifo1 = new MyBasicFifo();
fifo1.push("a"); fifo1.push("b"); fifo1.push("c");
console.log(fifo1.pop());
console.log(fifo1.pop());
console.log(fifo1.pop());


fifo1.push("aa"); fifo1.push("bb"); fifo1.push("cc");
let fifo1It = fifo1.fifoIteratorGenerator();
let arr1 = [ ...fifo1It ]
console.log(arr1);

for(let e of fifo1){
        console.log(`>>${e}`);
}
```

```
 ==>
a
b
c
[ 'aa', 'bb', 'cc' ]
>>aa
>>bb
>>cc
```

liens à suivre pour approfondir le sujet :

- https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/iterateurs_et_generateurs

1.5. Proxies

Proxy permet de coder des **intercepteurs** .

Exemple simple :

```
const targetObject = {
        prenom: 'Jean',
        nom: 'Bon',
        taille: 1.75
};

const interceptorUpperCaseHandler = {
   get(target, propKey, receiver) {
        let val = target[propKey];
        // NO let val = receiver[propKey]; ==> STACK-OVERFLOW / get interceptor calling get , ...
      console.log('intercept get ' + propKey);
                if(typeof val=='string' && val!=null)
                        val=val.toUpperCase();
      return val;
   }
};

const validateNumberHandler = {
  set (target, key, value) {
    if (key === 'age' || key === 'taille') {
      if (typeof value !== 'number' || Number.isNaN(value)) {
        throw new TypeError(key +' must be a number')
      }
      if (value <= 0) {
        throw new TypeError(key +' must be a positive number')
      }
    }
    target[key]=value; //default behavior
    return true; //indicate success
  }
}

var mixedHandler = Object.assign(interceptorUpperCaseHandler,validateNumberHandler);

const proxyObject = new Proxy(targetObject, interceptorUpperCaseHandler);
const proxyValidObject = new Proxy(targetObject, /*validateNumberHandler*/ mixedHandler);
//proxyValidObject.taille="abc"; --> TypeError : taille must be a number
//proxyValidObject.taille=-56; --> TypeError: taille must be a positive number
proxyValidObject.taille=1.80;
console.log("nouvelle taille=" + proxyValidObject.taille);

console.log(JSON.stringify(proxyObject));
```

==>

intercept get taille
nouvelle taille=1.8
intercept get toJSON

intercept get prenom
intercept get nom
intercept get taille
{"prenom":"JEAN","nom":"BON","taille":1.8}

Autre exemple :

```
// Proxying a normal object :
var targetObj = {
        id : 1 ,
        label : "o1"
};
var objectDefaultValueHandler = {
  get (target, propertyName /* , receiver */) {
        /* let val = target[propertyName];
        if(val==undefined)
                return `interceptor default value for property ${propertyName}`;
        else
          return val; */
      return (propertyName in target ) ? target[propertyName]
                        : `interceptor default value for property ${propertyName}`;
  }
};

var p = new Proxy(targetObj, objectDefaultValueHandler);
console.log("p.id="+p.id); // 1
console.log("p.label="+p.label); // o1
console.log("p.p3="+p.p3); // interceptor default value for property p3
console.log("p.p4="+p.p4); // interceptor default value for property p4
```

Listes des "traps" / "intercepteurs" possibles :

- `defineProperty(target, propKey, propDesc) : boolean`
  - `Object.defineProperty(proxy, propKey, propDesc)`
- `deleteProperty(target, propKey) : boolean`
  - `delete proxy[propKey]`
  - `delete proxy.foo // propKey = 'foo'`
- `get(target, propKey, receiver) : any`
  - `receiver[propKey]`
  - `receiver.foo // propKey = 'foo'`
- `set(target, propKey, value, receiver) : boolean`
  - `receiver[propKey] = value`
  - `receiver.foo = value // propKey = 'foo'`
- `getOwnPropertyDescriptor(target, propKey) : PropDesc|Undefined`
  - `Object.getOwnPropertyDescriptor(proxy, propKey)`

- `getPrototypeOf(target) : Object|Null`
  - `Object.getPrototypeOf(proxy)`
- `setPrototypeOf(target, proto) : boolean`
  - `Object.setPrototypeOf(proxy, proto)`
- `has(target, propKey) : boolean`
  - `propKey in proxy`
- `isExtensible(target) : boolean`
  - `Object.isExtensible(proxy)`
- `ownKeys(target) : Array<PropertyKey>`
  - `Object.getOwnPropertyPropertyNames(proxy)` (only uses string keys)
  - `Object.getOwnPropertyPropertySymbols(proxy)` (only uses symbol keys)
  - `Object.keys(proxy)` (only uses enumerable string keys; enumerability is checked via `Object.getOwnPropertyDescriptor`)
- `preventExtensions(target) : boolean`
  - `Object.preventExtensions(proxy)`

Liste des "traps" / "intercepteurs" possibles pour fonctions :

- `apply(target, thisArgument, argumentsList) : any`
  - `proxy.apply(thisArgument, argumentsList)`
  - `proxy.call(thisArgument, ...argumentsList)`
  - `proxy(...argumentsList)`
- `construct(target, argumentsList, newTarget) : Object`
  - `new proxy(..argumentsList)`

Exemple (proxy fonction et methode) :

```
// Proxying a function object
var targetRepeatWordFct = function (n , word) {
  return word.repeat(n); //NB: String.repeat(n) est une nouvelle méthode de es2015
 };

var fctHandler = {
  apply(targetFct, thisArg, argsList) {
        let res=targetFct.apply(thisArg, argsList); //appel de la fonction d'origine
        return res.toUpperCase(); //retourne le résultat transformé en majuscules
  }
};


var proxyFct = new Proxy(targetRepeatWordFct, fctHandler);
console.log( targetRepeatWordFct(3,"ha")); //hahaha
console.log( proxyFct(3,"ha")); //HAHAHA
```

Etant donné que les mécanismes internes de javascript traite un appel de méthode en 2 étapes :
1) get function from methodName as propertyName
2) function call
l'interception d'une méthode peut donc se faire de la façon suivante :

```
// Proxying a object with method
var targetObj = {
        id : 1 ,
        label : "obj1",
        idAndLabel(){ return ""+this.id+","+this.label; }
};

var objectHandler = {
  get (target, propertyName /* , receiver */) {
     if((typeof target[propertyName])== "function"){
          return function(...args){
                  /* let origMethod = target[propertyName];
                  let result = origMethod.apply(this, args);
                       if(typeof result == 'string')
                              result=result.toUpperCase();
                    return result;
                     */
                 var proxyMethodFct = new Proxy(target[propertyName], fctHandler);
                  //application (réutilisation) de fctHandler (de l'exemple précédent)
                 return proxyMethodFct.apply(this, args);
          }
   }
   else { //normal attribute , not a function/method :
         let val = target[propertyName];
        if(typeof val=='string' && val!=null)
                val=val.toUpperCase();

         return val;
        }
  }//end of get()
};

var proxyObjWithMethod=new Proxy(targetObj,objectHandler);
console.log(proxyObjWithMethod.idAndLabel());  //1,OBJ1
```

1.6. Reflect Api

Etant donné qu'un objet javascript est codé en interne comme une map entre noms et valeurs de propriétés/méthodes, on pouvait déjà en es5 découvrir au runtime la structure d'un objet javacript quelconque et ajuster du code dynamique en conséquence.

La version "es6/es2015" a cependant apporté une API de "reflection" se voulant plus explicite et rigoureuse. Le coeur de cette Api est l'objet "**Reflect**" .

Pas de ~~new Reflect(....)~~ mais des appels "static" (ex : Reflect.**get**(obj,"propertyName"))

Exemples :

```
var obj ={
        id: 1,
        label : "obj1"
}

let valueOfIdProperty=Reflect.get(obj,"id"); //equivalent à obj["id"]
console.log("value of Property id = "+valueOfIdProperty);//value of Property id = 1
console.log("value of Property label = "+Reflect.get(obj,"label"));
//value of Property label = obj1

Reflect.set(obj, "label",  "labelXy");
console.log("modified obj.label = "+obj.label);//modified obj.label = labelXy

console.log("obj="+JSON.stringify(obj));//obj={"id":1,"label":"labelXy"}

let boolRes= Reflect.defineProperty(obj, "name", {value: 'nomQuiVaBien' ,
                        writable : true,  enumerable : true,   configurable : true});
//propertyDesciptor with enumerable=true to see property in loop like for (.. in) or JSON.stringify
//                  configurable=true to enable changing attribute property (delete it , ...)
console.log("obj.name="+obj.name); //nomQuiVaBien
console.log("obj="+JSON.stringify(obj));
//obj={"id":1,"label":"labelXy","name":"nomQuiVaBien"}
Reflect.deleteProperty(obj, "name");
console.log("obj.name="+obj.name); //undefined
console.log("obj="+JSON.stringify(obj));obj={"id":1,"label":"labelXy"}

console.log("obj has label property=" + Reflect.has(obj, "label")); //true
console.log("obj has name property=" + Reflect.has(obj, "name")); //false

var labelPropertyDescriptor = Reflect.getOwnPropertyDescriptor(obj, "label");
 console.log("labelPropertyDescriptor="+JSON.stringify(labelPropertyDescriptor));
//labelPropertyDescriptor={"value":"labelXy","writable":true,"enumerable":true,"configurable":true}

let arrayOfPropKeys =  Reflect.ownKeys(obj); //may ignoring inheritance in old version
console.log("arrayOfPropKeys="+arrayOfPropKeys);//arrayOfPropKeys=id,label


class Person {
   constructor(id=0,name='?'){ this.id=id; this.name=name; }
}
class Employee extends Person {
        constructor(id=0, name='emp?' , salary=0) { super(id,name); this.salary=salary; }
}
var emp1 = new Employee(1,"employee1",1000);
console.log("properties of Employee="+Reflect.ownKeys(emp1));//id,name,salary
//var allPropsIterator = Reflect.enumerate(emp1); is now obsolete : not use it !!!!
```

```
function addFct(x, y){
        return x + y;
}
console.log("10+20="+Reflect.apply(addFct, null /*thisArg*/, [10, 20]));//10+20=30

function functionForObject(x, y){
        return this.num + x + y;
}
let computeObj={
        num:30,
        methXy:functionForObject
}
var value = Reflect.apply(functionForObject, computeObj /*thisArg*/, [10, 20]);
console.log(value +" is equals to " + computeObj.methXy(10,20)); //60  is equals to 60

Reflect.preventExtensions(obj);//cannot add new property
console.log("obj is extensible after preventExtensions:"+Reflect.isExtensible(obj)); //false
obj.newAttr="newValue";//no effect
console.log("obj.newAttr="+obj.newAttr);//undefined

function constructorAB(a, b)
{
        this.a = a;
        this.b = b;
        this.fctAdd = function(){
                return this.a + this.b;
        }
}

 var builtObj = Reflect.construct(constructorAB, [10, 20]);

console.log(builtObj.fctAdd()); //30
```

1.7. Typed Arrays are an ES6 API for handling binary data.

Exemple:

```
const typedArray = new Uint8Array([0,1,2]);
console.log(typedArray.length); // 3
typedArray[0] = 5;
const normalArray = [...typedArray]; // [5,1,2]

// The elements are stored in typedArray.buffer.
// Get a different view on the same data:
const dataView = new DataView(typedArray.buffer);
console.log(dataView.getUint8(0)); // 5
```

Instances of `ArrayBuffer` store the binary data to be processed. Two kinds of *views* are used to access the data:

- Typed Arrays (`Uint8Array`, `Int16Array`, `Float32Array`, etc.) interpret the

ArrayBuffer as an indexed sequence of elements of a single type.
- Instances of `DataView` let you access data as elements of several types (`Uint8`, `Int16`, `Float32`, etc.), at any byte offset inside an ArrayBuffer.

The following browser APIs support Typed Arrays :

- File API
- XMLHttpRequest
- Fetch API
- Canvas
- WebSockets
- ...

ANNEXES

XIII - 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 à 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.
onbubmit 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 boite 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 créé 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:

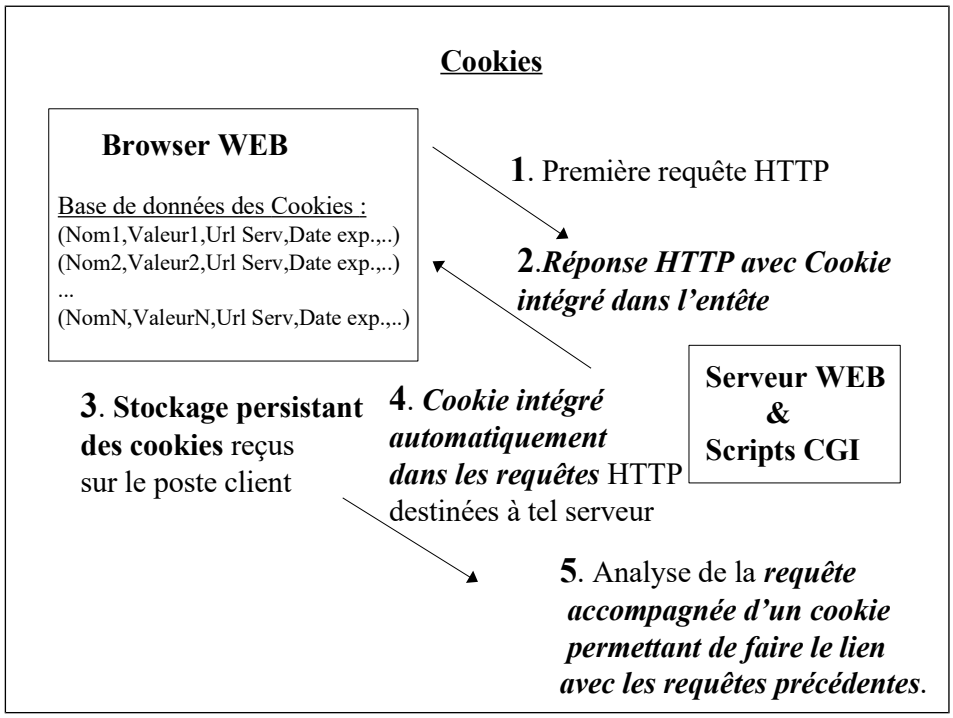
length nombre d'options dans l'objet SELECT
multiple (booléen) permet de sélectionner plusieurs éléments
size hauteur du champ select (si 1 ==> liste déroulante)
selectedIndex index de l'option sélectionnée
options liste des entrées dans la zone de liste

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

Préliminaire:
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; ***path***=/; ***expires***=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 ; ==> derniere 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 boite 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,defaut) 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 3éme 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.

XIV - Annexe – Canvas et Chart

1. Api "canvas" et bibliothèque "chart"

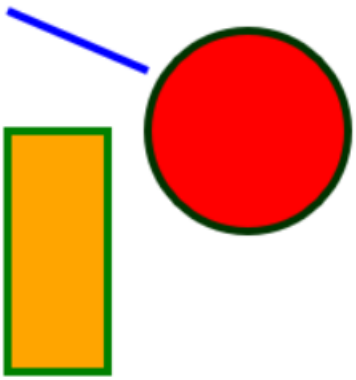
1.1. Api "canvas" (html5)

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.lineWith , 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



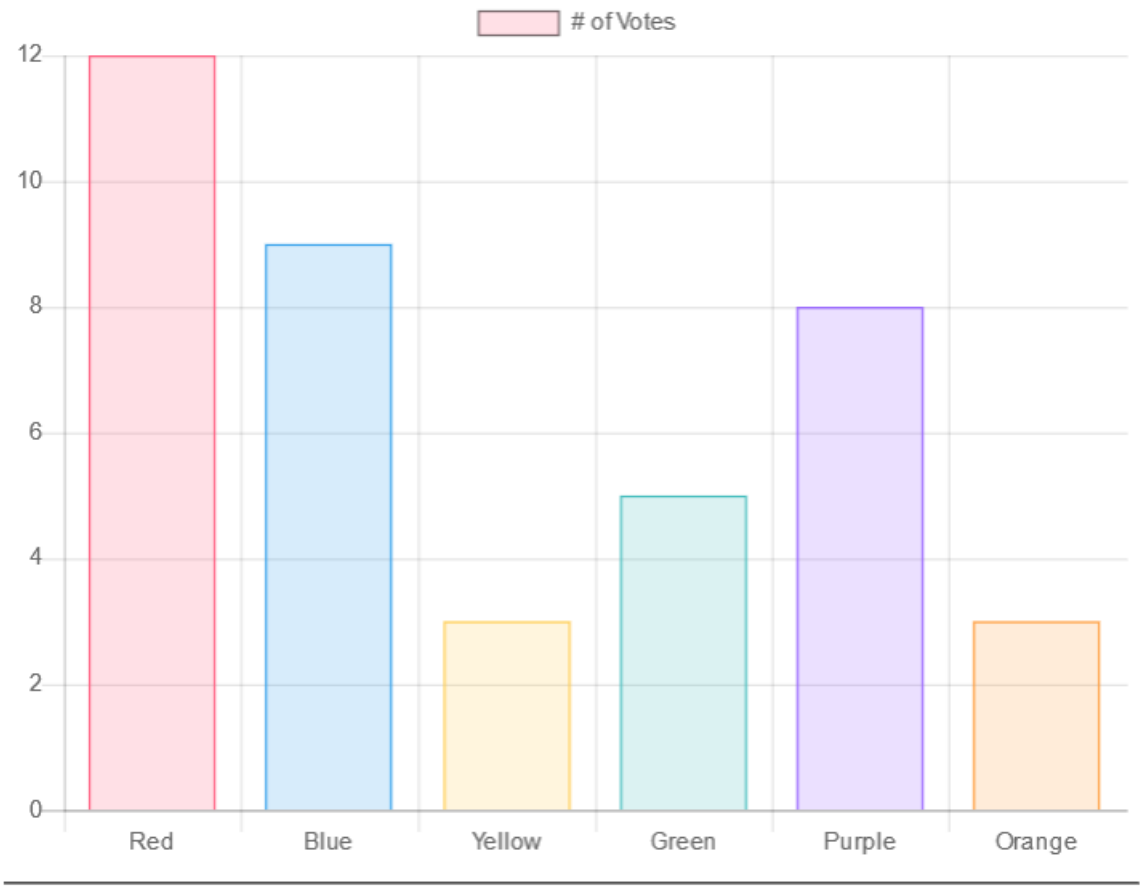
Pour approfondir le sujet --> ***https://www.w3schools.com/tags/ref_canvas.asp***

1.2. Bibliothèque "chart" (javascript)

https://www.chartjs.org/

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

bar chart



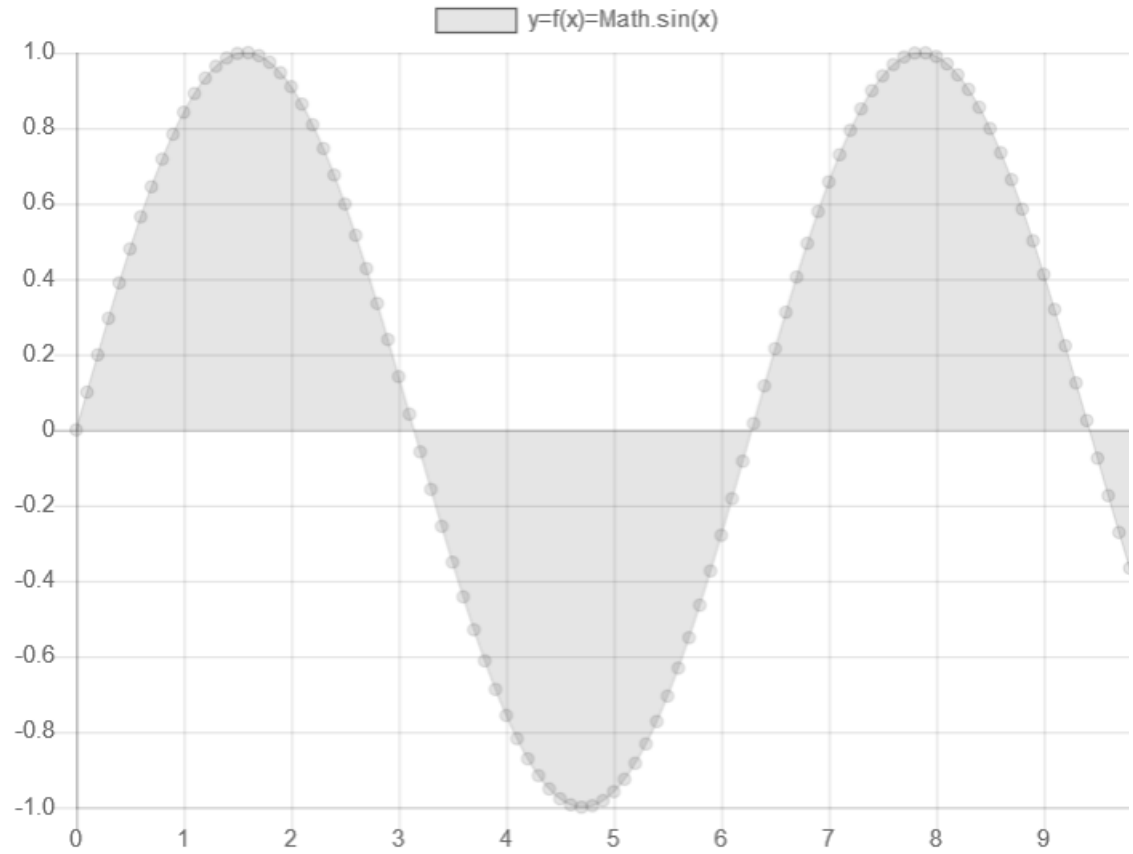

```
var canvasChart1Element = document.getElementById("myChart1");
var ctx1 = canvasChart1Element.getContext("2d");

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 0 a 10
y=f(x)=Math.sin(x)
afficher courbe




```
<h3> curve / graph / function </h3>
 pour x allant de <input id="xMin" value="0"/> a
                                  <input id="xMax" value="10"/> <br/>
```

```
 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,
        }
      }]
    }
  }
});

});
```

XV - 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 : 15

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***').**action**Zz(*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.

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

$(':**even**'); $(':**odd**');	index est pair ou impair
$(':**empty**');	Sans enfant
$(':**first**'); $(':**last**');	premier , dernier
$(':**first-child**'); $(':**last-child**);	premier , dernier enfant
$(':**focus**');	tous les éléments ayant actuellement le focus.
$(':**hidden**'); $(':**visible**');	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() **removeClass()**	ajoute ou supprime une classe à un élément.
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() *ou* **appendTo()**	Ajoute du contenu HTML ou textuel à la fin de l'élément ciblé(à l'intérieur , comme enfant , sous élément)
prepend() *ou* **prependTo()**	Ajoute du contenu HTMK ou textuel au début de l'élément ciblé (à l'intérieur , comme enfant , sous élément)
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 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 *display:none*).
show()	Fait réapparaître un élément.
fadeOut()	Fait disparaître un élément (qui aura la propriété *display:none*) 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"

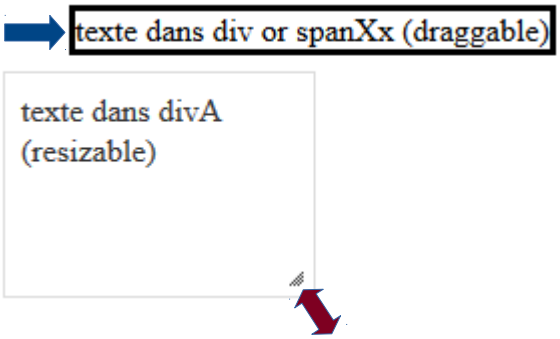
http://jqueryui.com (demo , download) est l'URL de référence pour récupérer le pack de plugins "jquery-ui" .

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>
```



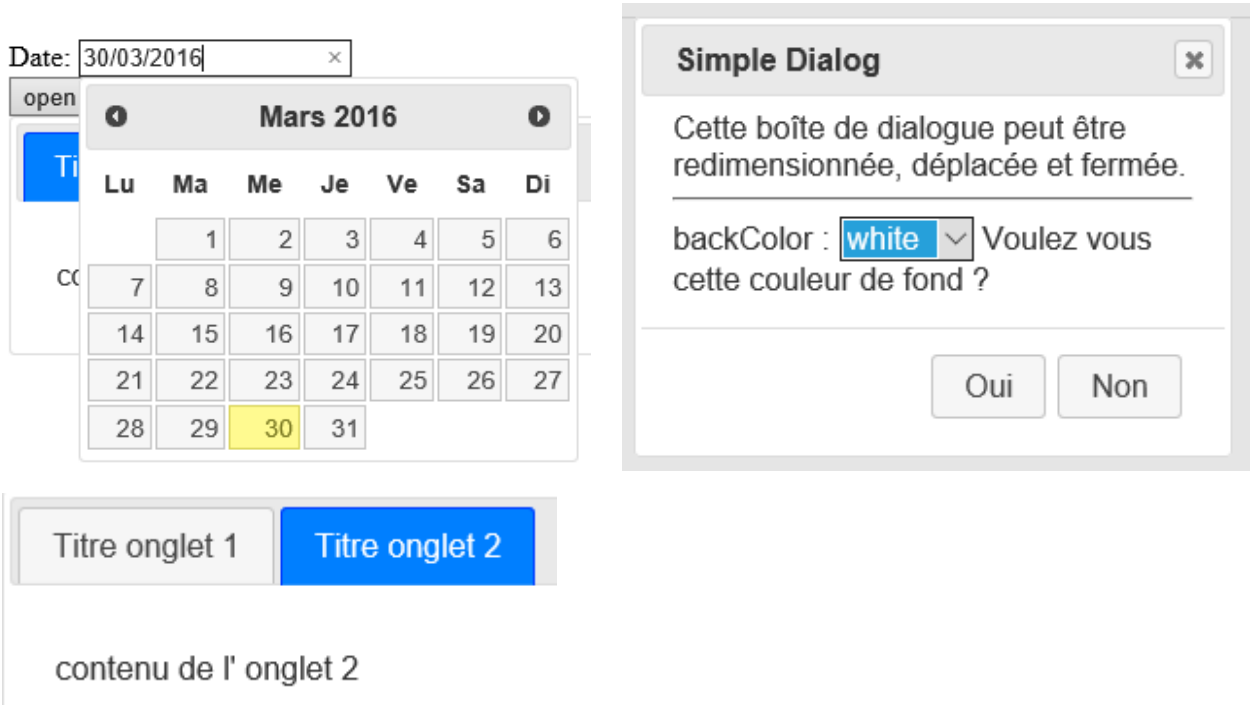
Principaux Widgets de jquery-ui :

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>
```



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>
```

texte du paragraphe1

texte du paragraphe2

```
<body>
  <div id="divA">
    <p id="p1"> texte du paragraphe1 </p>
    <p id="p2"> texte du paragraphe2 </p>
  </div>
</body>
</html>
```

XVI - Annexe – node , npm , express, ...

1. Ecosystème node+npm

node (nodeJs) est un **environnement d'exécution javascript** permettant essentiellement de :
- compartimenter le code à exécuter en **modules** (import/export)
- exécuter du code en mode "**appels asynchrones non bloquants** + callback" (sans avoir recours à une multitudes de threads)
- exécuter directement du code javascript sans avoir à utiliser un navigateur web

npm ***(node package manager)*** est une sous partie fondamentale de node qui permet de :
- **télécharger et gérer des packages utiles à une application** (librairies réutilisables)
- télécharger et utiliser des utilitaires pour la phase de développement (ex : grunt , jasmine , gulp , …)
- **prendre en compte les dépendances entre packages** (téléchargements indirects)
- générer éventuellement de nouveaux packages réutilisables (à déployer)
- ….

node est à peu près l'équivalent "javascript" d'une machine virtuelle java.
npm ressemble un peu à maven de java : téléchargement des librairies , construction d'applications.

Un **projet basé sur npm** se configure avec le fichier ***package.json*** et les packages téléchargés sont placés dans le sous répertoire **node_modules** .

Principales utilisations/applications de node :
- application "serveur" en javascript (répondant à des requêtes HTTP)
- application autonome (ex : StarUML2 = éditeur de diagrammes UML , …)
- ….

2. Express

Express correspond à un des packages téléchargeables via npm et exécutables via node.
L**a technologie "express" permet de répondre à des requêtes HTTP** et ressemble un peu à un Servlet java ou à un script CGI .

A fond **basé sur des mécanismes souples et asynchrones (avec "routes" et "callbacks")** , **"express" permet de coder assez facilement/efficacement des applications capables de** :
- **générer dynamiquement des pages HTML** (ou autres)
- mettre en œuvre des **web services "REST"** (souvent au format "JSON") .
- prendre en charge les détails du protocoles **HTTP** (authentification "basic" et/ou "bearer" , autorisations "CORS" , ….)
- ….

"express" est souvent considéré comme une technologie de bas niveau lorsque l'on la compare à d'autres technologies "web / coté serveur" telles que ASP , JSP , PHP , …

"express" permet de construire et retourner très rapidement une réponse HTTP (avec tout un tas de paramétrages fin si nécessaire) . Pour tout ce qui touche au format de la réponse à générer , il faut

utiliser des technologies complémentaires (ex : templates de pages HTML avec remplacements de valeurs) .

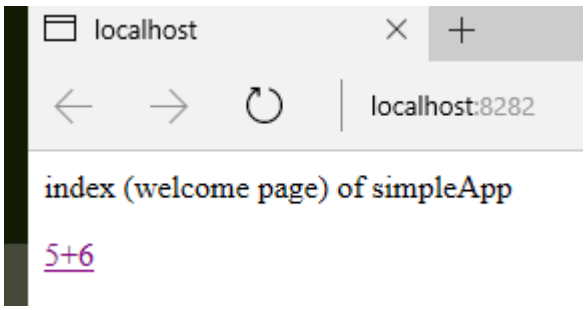
3. Exemple élémentaire "node+express"

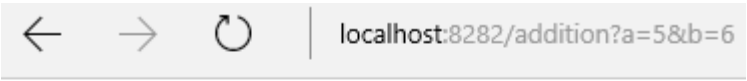
first_express_server.js

```
//modules to load:
var express = require('express');

var app = express();

app.get('/', function(req, res , next) {
     res.setHeader('Content-Type', 'text/html');
     res.write("<html> <body>");
     res.write('<p>index (welcome page) of simpleApp</p>');
     res.write('<a href="addition?a=5&b=6">5+6</a>');
     res.write("</body></html>");
     res.end();
});

//GET addition?a=5&b=6
app.get('/addition', function(req, res , next) {
     a = Number(req.query.a);   b = Number(req.query.b);
     resAdd = a+b;
     res.setHeader('Content-Type', 'text/html');
     res.write("<html> <body>");
     res.write('a=' + a + '<br/>');  res.write('b=' + b + '<br/>');
     res.write('a+b=' + resAdd + '<br/>');
     res.write("</body></html>");
     res.end();
});

app.listen(8282 , function () {
  console.log("simple express node server listening at 8282");
});
```

lancement : **node first_express_server.js**

via http://localhost:8282 au sein d'un navigateur web , on obtient le résultat suivant :



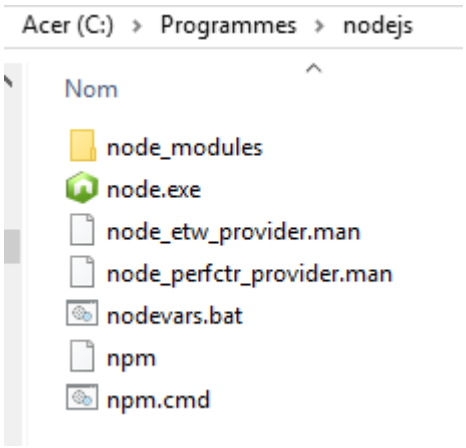


a=5
b=6
a+b=11

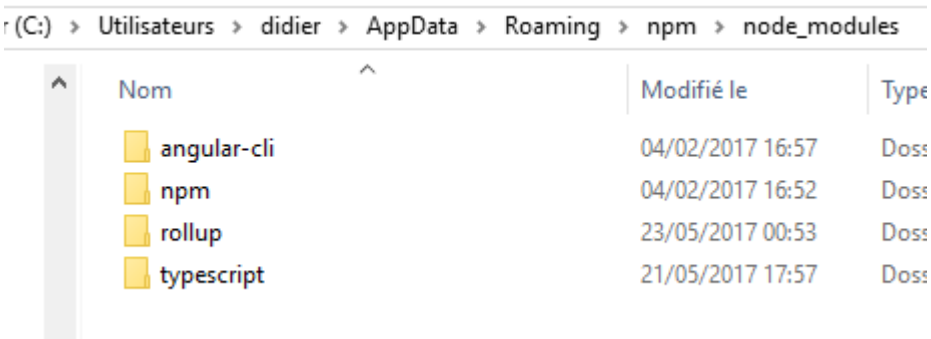
4. Installation de node et npm

Téléchargement de l'installeur **node-v6.10.3-x64.msi** (ou autre) depuis le site officiel de nodeJs (https://nodejs.org)
Lancer l'installation et se laisser guider par les menus.
Cette opération permet sous windows d'installer node et npm en même temps .

Sur une machine windows 64bits , nodejs s'installe par défaut dans ***C:\Program Files\nodejs***



Et le répertoire pour les installations de packages en mode "global" (-g) est par défaut
C:\Users\username\AppData\Roaming\npm\node_modules



Vérification de l'installation (dans un shell "CMD") :

node --version
v6.9.4 (ou autre)

npm --version
4.1.2 (ou autre)

4.1.3

5. Configuration et utilisation de npm

5.1. Initialisation d'un nouveau projet

Un développeur utilise généralement npm dans le cadre d'un projet spécifique (ex : xyz).
Après avoir créer un répertoire pour ce projet (ex : C:\tmp\temp_nodejs\xyz) et s'être placé dessus, on peut lancer la c*ommande interactive* **npm init** de façon à **générer un début de fichier "*package.json*"**

Exemple de fichier ***package.json*** généré :

```
{
  "name": "xyz",
  "version": "1.0.0",
  "description": "projet xyz",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "didier",
  "license": "ISC"
}
```

5.2. installation de nouveau package en ligne de commande :

npm install --save express
npm install --save mongoose

permet de télécharger les packages "express" et "mongoose" (ainsi que tous les packages indirectement nécessaires par analyse de dépendances) dans le sous répertoire **nodes_modules** et de mettre à jour la liste des dépendances dans le fichier **package.json** :

```
…. ,
"dependencies": {
    "express": "^4.15.3",
    "mongoose": "^4.11.0"
  },
….
```

Sans l'option **--save** , les packages sont téléchargés mais le fichier package.json n'est pas modifié.

Par défaut , c'est la dernière version du package qui est téléchargé et utilisé.
Il est possible de choisir une **version spécifique** en la précisant après le caractère @ :

npm install --save mongoose@4.10
ou bien (autre exemple) :
npm install --save mongodb@2.0.55

Autre procédure possible :

1) **éditer** le fichier ***package.json*** en y ajoutant des dépendances (au sein de la partie "dependencies") :
   exemple :
   "dependencies": {
   "express": "^4.15.3",
   "markdown": "^0.5.0",
   "mongoose": "^4.10.8"
   }
2) lancer **npm install** (ou *npm update* ultérieurement) sans argument

Ceci permet de lancer le téléchargement et installation du package "mardown" dans le sous répertoire **node_modules** .

Installation de packages utilitaires (pour le développement) :

Si l'on souhaite ensuite expliciter une dépendance de "développement" au sein d'un projet , on peut utiliser l'option **--save-dev** de **npm install** de façon ajouter celle ci dans la partie "devDependencies" de package.json :

npm install **--save-dev grunt**

```
…. ,
"devDependencies": {
   "grunt": "^1.0.1"
  }
...
```

5.3. Installation en mode global (-g)

L'option **-g** de **npm install** permet une installation en mode global : le package téléchargé sera installé dans ***C:\Users\username\AppData\Roaming\npm\node_modules*** sous windows 64bits (ou ailleurs sur d'autres systèmes) **et sera ainsi disponible (en mode partagé) par tous les projets** .

Le mode global est souvent utilisé pour installer des packages correspondant à des "utilitaires de développement" (ex : grunt) .

Exemple :

npm install -g grunt

6. Utilisation basique de node

hello_world.js

```
console.log("hello world");
```

node ***hello_world.js***

7. Modules (export , import)

Un des grands atouts de nodejs est un programmation à base de **modules bien compartimentés**.

7.1. Module avec élément(s) exporté(s)

mycomputer_module.js

```
var myAddStringFct = function(a,b) {
  result=a+b;
  resultString = "" + a + " + " + b +" = " + result;
  return resultString;
};

exports.myAddStringFct = myAddStringFct;
```

NB: ***Seuls les éléments exportés seront vus par les autres modules !!!***

7.2. Importation de module(s)

basic_exemple_with_modules.js

```
//chargement / importation des modules :
var mycomputer_module = require('./mycomputer_module'); // ./ for searching in local relative
var markdown = require('markdown').markdown; // without "./" in node_modules sub directory

//utilisation des modules importés :
var x=5;
var y=6;
var resString = mycomputer_module.myAddStringFct(x,y);

console.log(resString);
```

```
var resHtmlString = markdown.toHTML("**"+resString+"**");
//NB: "markdown" est un mini langage de balisage
// où un encadrement par ** génère un équivalent de
// <strong> HTML (proche de <bold>)
console.log(resHtmlString);
```

node basic_exemple_with_modules

résultats:

5 + 6 = 11
5 + 6 = 11

8. Essentiel de Express

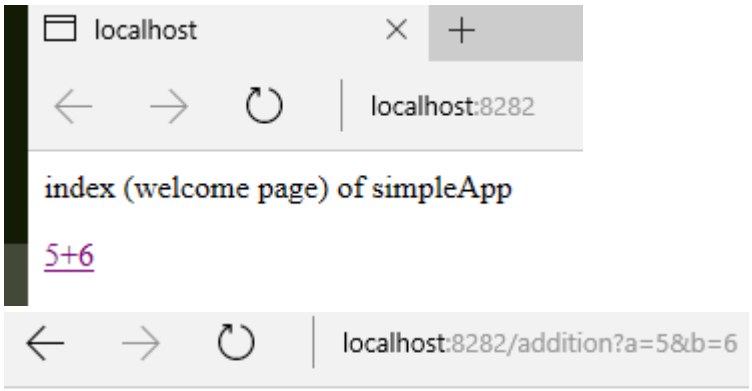
8.1. Exemple élémentaire (rappel)

first_express_server.js

```
//modules to load:
var express = require('express');

var app = express();

app.get('/', function(req, res , next) {
     res.setHeader('Content-Type', 'text/html');
     res.write("<html> <body>");
     res.write('<p>index (welcome page) of simpleApp</p>');
     res.write('<a href="addition?a=5&b=6">5+6</a>');
     res.write("</body></html>");
     res.end();
});

//GET addition?a=5&b=6
app.get('/addition', function(req, res , next) {
     a = Number(req.query.a);   b = Number(req.query.b);
     resAdd = a+b;
     res.setHeader('Content-Type', 'text/html');
     res.write("<html> <body>");
     res.write('a=' + a + '<br/>');  res.write('b=' + b + '<br/>');
     res.write('a+b=' + resAdd + '<br/>');
     res.write("</body></html>");
     res.end();
});

app.listen(8282 , function () {
  console.log("simple express node server listening at 8282");
});
```

lancement : **node first_express_server.js**

via http://localhost:8282 au sein d'un navigateur web , on obtient le résultat suivant :



9. WS REST élémentaire avec node+express

```
var express = require('express');
var myGenericMongoClient = require('./my_generic_mongo_client');
var app = express();
…
/*
//fonction inutile car équivalente à res.send(jsObject) :
var sendGenericJsonExpressResponse = function(jsObject,res){
        res.setHeader('Content-Type', 'application/json');
        res.write(JSON.stringify(jsObject));
        res.end();
}*/

function sendDataOrError(err,data,res){
        if(err==null) {
                        if(data!=null)
                                res.send(data);
                        else res.status(404).send(null);//not found
                }
                else res.status(500).send({error: err});//internal error (ex: mongo access)
}

// GET (array) /minibank/operations?numCpt=1
app.get('/minibank/operations', function(req, res,next) {
  myGenericMongoClient.genericFindList('operations', { 'compte' :
Number(req.query.numCpt) },
                function(err,tabOperations){
                        sendDataOrError(err,tabOperations,res);
                });
```

```
});

// GET /minibank/comptes/1
app.get('/minibank/comptes/:numero', function(req, res,next) {
  function(req, res,next) {
        myGenericMongoClient.genericFindOne('comptes',
                { '_id' : Number(req.params.numero) },
                function(err,compte){
                        sendDataOrError(err,compte,res);
                });
});

app.listen(8282 , function () {
 console.log("rest server listening at 8282");
});
```

NB : **req.query.**pxy récupère la valeur d'un paramètre http en fin d'URL (**?pxy=valXy**&pzz=valZz)
req.params.pxy récupère la valeur d'un paramètre logique (avec:) en fin d'URL

dans cet exemple , myGenericMongoClient.***genericFind....()*** correspond à un élément d'un module utilitaire qui récupère des données dans une base mongoDB .

10. Avec mode post et authentification minimaliste

```
var express = require('express');
var bodyParser = require('body-parser');   //dépendance indirecte de express via npm
var app = express();

var myGenericMongoClient = require('./my_generic_mongo_client');

var uuid = require('uuid'); //to generate a simple token

app.use(bodyParser.json()); // to parse JSON input data and generate js object : (req.body)
app.use(bodyParser.urlencoded({ extended: true}));

...

// POST /minibank/verifyAuth  {  "numClient" : 1 , "password" : "pwd1" }
app.post('/minibank/verifyAuth', function(req, res,next) {
        var verifAuth = req.body; // JSON input data as jsObject with ok = null
        console.log("verifAuth :" +JSON.stringify(verifAuth));
         if(verifAuth.password == ("pwd" + verifAuth.numClient) ){
                verifAuth.ok= true;
                verifAuth.token=uuid.v4();
                //éventuelle transmission parallèle via champ "x-auth-token" :
                res.header("x-auth-token", verifAuth.token);
                //+stockage dans une map pour verif ulterieure : ....
                }
        else {
                verifAuth.ok= false;
                verifAuth.token = null;
        }
```

```
   res.send(verifAuth);    // send  back with ok = true or false and token
});

// GET /minibank/comptes/1
app.get('/minibank/comptes/:numero',
   displayHeaders,  verifTokenInHeaders  /*un peu securisé*/ ,
   function(req, res,next) {
        myGenericMongoClient.genericFindOne('comptes', { '_id' : Number(req.params.numero) },
              function(err,compte){
                                    sendDataOrError(err,compte,res);
              });
});

function sendDataOrError(err,data,res){
        if(err==null) {
                        if(data!=null)
                                res.send(data);
                        else res.status(404).send(null);//not found
                }
                else res.status(500).send({error: err});//internal error (ex: mongo access)
}


//var secureMode = false;
var secureMode = true;

function extractAndVerifToken(authorizationHeader){
        if(secureMode==false) return true;
        /*else*/
        if(authorizationHeader!=null ){
                if(authorizationHeader.startsWith("Bearer")){
                        var token = authorizationHeader.substring(7);
                        console.log("extracted token:" + token);
                        //code extremement simplifié ici:
                        //idealement à comparer avec token stocké en cache (si uuid token)
                        //ou bien tester validité avec token "jwt"
                        if(token != null && token.length>0)
                                return true ;
                         else
                                return false;
                }
                else
                        return false;
        }
        else
                return false;
}

// verif bearer token in Authorization headers of request :
function verifTokenInHeaders(req, res, next) {
 if( extractAndVerifToken(req.headers.authorization))
    next();
 else
```

```
            res.status(401).send(null);//401=Unautorized or 403=Forbidden
}

// display Authorization in request (with bearer token):
function displayHeaders(req, res, next) {
  //console.log(JSON.stringify(req.headers));
  var authorization = req.headers.authorization;
  console.log("Authorization: " + authorization);
  next();
}

...
app.listen(8282 , function () {
  console.log("minibank rest server listening at 8282");
});
```

11. Autorisations "CORS"

```
var express = require('express');
var app = express();
…

// CORS enabled with express/node-js :
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
                          //ou avec  "www.xyz.com" à la place de "*" en production
  res.header("Access-Control-Allow-Headers",
             "Origin, X-Requested-With, Content-Type, Accept");
  next();
});

…
…
app.get(...) , app.post(...) , …

app.listen(8282 , function () {
  console.log("rest server with CORS enabled listening at 8282");
});
```

12. Accès à un SGBDR (SQL) via node

```
npm install mysqljs/mysql
```

...

13. Accès à MongoDB (No-SQL , JSON) via node

13.1. Via mongodb/MongoClient (sans mongoose)

my_generic_mongo_client.js

```
//myGenericMongoclient module (with MongoDB/MongoClient)
var MongoClient = require('mongodb').MongoClient;
var ObjectId = require('mongodb').ObjectID;
var assert = require('assert');

//NB: sans connexion internet le localhost est moins bien géré que
// 127.0.0.1 sur certains ordinateurs (ex: windows 8 , 10 ).
var mongoDbUrl = 'mongodb://127.0.0.1:27017/test'; //by default
var currentDb=null; //current MongoDB connection

var setMongoDbUrl = function(dbUrl){
        mongoDbUrl = dbUrl;
}

var closeCurrentMongoDBConnection = function(){
        currentDb.close();
        currentDb=null;
}

var executeInMongoDbConnection = function(callback_with_db) {
  if(currentDb==null){
   MongoClient.connect(mongoDbUrl, function(err, db) {
        if(err!=null) {
                console.log("mongoDb connection error = " + err + " for dbUrl=" + mongoDbUrl );
        }
        assert.equal(null, err);//arret de l'execution ici si err != null
        console.log("Connected correctly to mongodb database" );
        currentDb = db;
        callback_with_db(db);
        });
  }else{
        callback_with_db(currentDb);  //réutilisation de la connection.
  }
}
```

```
var genericUpdateOne = function(collectionName,id,changes,callback_with_err_and_results) {
   executeInMongoDbConnection( function(db) {
          db.collection(collectionName).updateOne( { '_id' : id }, { $set :  changes } ,
                        function(err, results) {
                              if(err!=null) {
                               console.log("genericUpdateOne error = " + err);
                              }
                              callback_with_err_and_results(err,results);
                        });
         });
};

var genericInsertOne = function(collectionName,newOne,callback_with_err_and_newId) {
  executeInMongoDbConnection( function(db) {
          db.collection(collectionName).insertOne( newOne , function(err, result) {
                  if(err!=null) {
                    console.log("genericInsertOne error = " + err);
                    newId=null;
                  }
                  else {newId=newOne._id;
                  }
                  callback_with_err_and_newId(err,newId);
                 });
   });
};

var genericFindList = function(collectionName,query,callback_with_err_and_array) {
    executeInMongoDbConnection( function(db) {
          var cursor = db.collection(collectionName).find(query);
          cursor.toArray(function(err, arr) {
                   callback_with_err_and_array(err,arr);
                 });
   });
};


var genericFindOne = function(collectionName,query, callback_with_err_and_item) {
 executeInMongoDbConnection( function(db) {
          db.collection(collectionName).findOne(query , function(err, item) {
                  if(err!=null) {
                  console.log("genericFindById error = " + err);
              }
               callback_with_err_and_item(err,item);
               });
   });
};

exports.genericUpdateOne = genericUpdateOne;
exports.genericInsertOne = genericInsertOne;
exports.genericFindList = genericFindList;
exports.genericFindOne = genericFindOne;
```

```
exports.setMongoDbUrl= setMongoDbUrl;
exports.closeCurrentMongoDBConnection=closeCurrentMongoDBConnection;
```

Utilisation :

```
var express = require('express');
var app = express();
var myGenericMongoClient = require('./my_generic_mongo_client');

// GET /minibank/comptes/1
app.get('/minibank/comptes/:numero',   function(req, res,next) {
        myGenericMongoClient.genericFindOne('comptes',
                { '_id' : Number(req.params.numero) },
                function(err,compte){
                        sendDataOrError(err,compte,res);
                });
});

// GET /minibank/clients/1
app.get('/minibank/clients/:numero',function(req, res,next) {
        myGenericMongoClient.genericFindOne('clients',
                { '_id' : Number(req.params.numero) },
                 /* anonymous callback function as lambda expression :*/
                        (err,client)=>{ sendDataOrError(err,client,res); }
                );
});

function sendDataOrError(err,data,res){
        if(err==null) {
                        if(data!=null)
                                res.send(data);
                        else res.status(404).send(null);//not found
                }
                else res.status(500).send({error: err});//internal error (ex: mongo access)
}
….
```

13.2. Via mongoose

Mongoose permet de mettre en œuvre une sorte de correspondance automatique entre un type d'objet javascript et un type document "mongoDB" .

ODM : Object Document Mapping.

Mode opératoire :

- on définit une structure de données (mongoose.Shema(....))
- on peut ensuite appeler des méthodes ".find() , .save() , .remove() , …" pour effectuer des opérations "CRUD" dans une base de données "mongoDB"

XVII - Annexe – WebPack

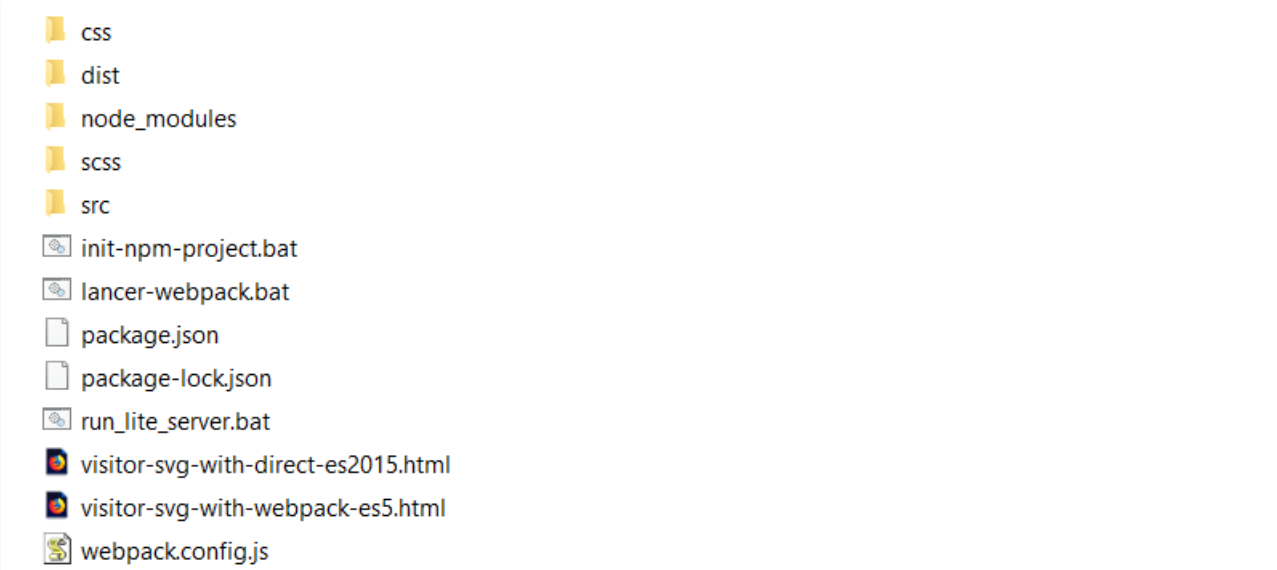
1. Webpack

webpack est une technologie javascript permettant de générer des fichiers "xyz-bundles" à partir d'un tas de petits fichiers complémentaires .

- Tout comme "rollup", webpack sait tenir compte des inter-relations entre fichiers/modules "es2015".
- webpack peut également être configuré pour déclencher des conversions "es2015 --> es5" via "**babel**".
- D'autre part , webpack peut également prendre en charge des bundles css en déclenchant si besoin un pré-processeur **saas** pour transformer ".**scss**" en ".css"

Au sein de cette présentation, c'est la version 4 (assez récente) de webpack qui sera utilisée .

1.1. Structure possible d'un projet "npm" intégrant webpack



Le répertoire **src** comportera les fichiers ".js" à assembler .
Le répertoire **dist** comportera les "bundles" générés par webpack

1.2. Configuration webpack pour des modules es2015

en mode global :
npm install -g webpack webpack-cli

dans projet "xyz" initialisé via npm init :
npm install --save-dev webpack webpack-cli

NB :

- *install -g pour lancement commande webpack*
  et install --save-dev pour accès api webpack depuis webpack.config.js
- *ajuster le fichier webpack.config.js avant de lancer webpack*

Configuration de webpack pour modules "es2015" :

webpack.config.js

```
const webpack = require("webpack");
const path = require("path");

//entry: "./src/index.js" ou entry: "./src/main.js" ou ...
let config = {
  entry: "./src/main.js",
  output: {
    path: path.resolve(__dirname, "./dist"),
    filename: "./main-bundle.js"
  }
}

module.exports = config;
```

Lancement de webpack :

lancer-webpack.bat

```
REM ajuster le fichier webpack.config.js avant de lancer webpack
webpack > statut-webpack.txt

REM NB: webpack --config ./webpack-xyz.config.js si autre config que webpack.config.js
```

ou bien

```
npm run webpack-watch
```

si **package.json** comporte (après ajout) :

```
... ,
"scripts": {
   "test": "echo \"Error: no test specified\" && exit 1",
   "webpack-watch": "webpack --watch"
  }, ...
```

NB : l'option ***--watch*** permet de surveiller les fichiers sources et de relancer automatiquement webpack dès qu'un fichier a changé .

Exemple de fichier ".html" utilisant un bundle javascript construit par webpack :

```
<html> <head>   <title>visitor-svg</title>
    <!-- <link rel="stylesheet" href="dist/styles-bundle.css"> -->
   <script src="dist/main-bundle.js"></script> </head>
<body>    <h3> visitor-svg (with es2015 modules packed in es5 bundle) <h3>
```

```
   <canvas width="500" height="400" id="myCanvas"></canvas>
</body> </html>
```

Exemple de modules es2015 (ici pour dessiner des figures géométriques) :

main.js

```
import { Fig2D , Line, Circle, Rectangle } from './fig2d-core.js' ;
import { CanvasVisitor } from './canvas-visitor-ext.js' ;

//import "../scss/styles.scss";
//import "../css/styles-ext.css";

function my_test(){
   var tabFig = new Array();
   tabFig.push(new Line(20,20,180,200,"red"));
   tabFig.push(new Circle(100,100,50,"blue", 2,"orange"));
   tabFig.push(new Circle(250,200,50,"black",1,"blue"));
   tabFig.push(new Rectangle(200,100,50,60,"green",4));
   tabFig.push(new Rectangle(20,100,50,60,"black",1,"green"));
   var visitor =  new  CanvasVisitor('myCanvas');
   for( let f  of tabFig){
     f.performVisit(visitor);
   }
}

window.addEventListener('load', function() {
        my_test();
});
```

fig2d-core.js

```
export class Fig2D {
 constructor(lineColor  = "black",    lineWidth  = 1,    fillColor  = null){
        this.lineColor = lineColor;    this.lineWidth = lineWidth;   this.fillColor = fillColor;
 }
 performVisit(visitor){}
```

```
}


export class Line  extends Fig2D{
 constructor(x1= 0 ,  y1= 0 ,   x2= 0 ,  y2= 0,   lineColor  = "black",    lineWidth  = 1){
              super(lineColor,lineWidth);
          this.x1=x1; this.y1=y1; this.x2=x2; this.y2=y2;
 }
 performVisit(visitor){
    visitor.doActionForLine(this);
  }
}

export class Circle  extends Fig2D{
 constructor(xC = 0 ,  yC = 0  ,  r = 0,  lineColor  = "black",  lineWidth  = 1,  fillColor = null){
              super(lineColor,lineWidth,fillColor);
             this.xC = xC; this.yC=yC; this.r=r;
 }
 performVisit(visitor) {
   visitor.doActionForCircle(this);
 }
}

export class Rectangle  extends Fig2D{
 constructor(x1=0, y1 =0,width =0, height =0, lineColor = "black", lineWidth =1,fillColor =null){
              super(lineColor,lineWidth,fillColor);
            this.x1=x1; this.y1=y1; this.width=width; this.height=height;
 }
 performVisit(visitor) {
   visitor.doActionForRectangle(this);
 }
}

export class FigVisitor {
 doActionForCircle(c){}
 doActionForLine(l){}
 doActionForRectangle(r){}
```

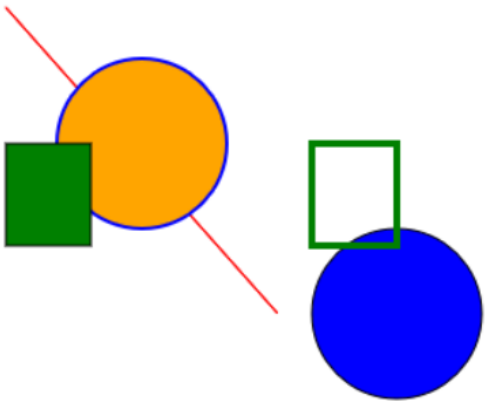
```
}
```

canvas-visitor-ext.js

```
import { Fig2D , Line, Circle, Rectangle ,  FigVisitor} from './fig2d-core.js' ;

export class CanvasVisitor  extends FigVisitor{
   constructor(canvasId){  super();
   this._canvasElement = document.getElementById(canvasId);
   this._ctx = this._canvasElement.getContext("2d");
  }

  doActionForCircle( c ) {
    this._ctx.beginPath();
    this._ctx.arc(c.xC, c.yC, c.r, 0, 2 * Math.PI, false);
    if(c.fillColor != null){
        this._ctx.fillStyle = c.fillColor;     this._ctx.fill();
      }
    this._ctx.lineWidth = c.lineWidth;
    this._ctx.strokeStyle =  c.lineColor;//'#003300';
    this._ctx.stroke();
  }
  doActionForLine( l ) {
    this._ctx.beginPath();
    this._ctx.moveTo(l.x1,l.y1);    this._ctx.lineTo(l.x2,l.y2);
    this._ctx.strokeStyle =  l.lineColor;    this._ctx.lineWidth = l.lineWidth;
    this._ctx.stroke();
  }
  doActionForRectangle( r ) {
    this._ctx.beginPath();
    this._ctx.rect(r.x1,r.y1,r.width,r.height);
    if(r.fillColor != null){
        this._ctx.fillStyle = r.fillColor;   this._ctx.fill();
      }
    this._ctx.strokeStyle =  r.lineColor;    this._ctx.lineWidth = r.lineWidth;
    this._ctx.stroke();
```

```
  }
}
```

visitor-svg (with es2015 modules packed



1.3. Intégrer babel dans webpack pour générer des bundles "es5"

npm install --save-dev babel-loader @babel/core
npm install --save-dev @babel/preset-env @babel/register

webpack.config.js

```
const webpack = require("webpack");
const path = require("path");

let config = {
  entry: "./src/main.js",
  output: {
    path: path.resolve(__dirname, "./dist"),
    filename: "./main-bundle.js"
  },
    module: {
       rules: [{
          test: /\.js/,
          exclude: /(node_modules|bower_components)/,
          use: [{
             loader: 'babel-loader',
            options: {
                    presets: ['@babel/preset-env']
                }
          }]
       }]
    }
}

module.exports = config;
```

Grâce à cette configuration le fichier généré (ici *dist/main-bundle.js*) est plus volumineux car il ne

contient plus les mots clefs "class" , "extends" , ... compréhensible que par les navigateurs récents supportant "es6/es2015" mais une traduction d'un équivalent en ancien javascript "es5" .

1.4. Gestion des css et scss dans webpack

npm install --save-dev mini-css-extract-plugin
npm install --save-dev style-loader css-loader

npm install --save-dev sass-loader node-sass

NB :

- *les modules **mini-css-extract-plugin** , **style-loader** et **css-loader** servent à générer des bundles css depuis webpack .*
- ***les modules sass-loader et node-sass servent à gérer les fichier ".scss" .***
- *autre ajout possible : postcss-loader pour générer des styles css spécifiques aux navigateurs*

*Exemple de fichier **package.json** :*

```
{
 "name": "xyz",
 "version": "1.0.0",
 "description": "",
 "main": "index.js",
 "scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "webpack-watch": "webpack --watch"
 },
 "author": "",
 "license": "ISC",
 "devDependencies": {
  "@babel/core": "^7.4.0",
  "@babel/preset-env": "^7.4.2",
  "@babel/register": "^7.4.0",
  "babel-loader": "^8.0.5",
  "css-loader": "^2.1.1",
  "mini-css-extract-plugin": "^0.5.0",
  "node-sass": "^4.11.0",
  "sass-loader": "^7.1.0",
  "style-loader": "^0.23.1",
  "webpack": "^4.29.6",
  "webpack-cli": "^3.3.0"
 }
}
```

Exemple :

dans *mains.js* :

```
//...
```

```
import "../scss/styles.scss";
import "../css/styles-ext.css";
//...
```

css/styles-ext.css

```
h3 {  color: red; }
```

*scss/**styles.scss***

```
$midnight-blue : #2c3e50;
body {  background-color: $midnight-blue ; }
```

webpack.config.js

```
const webpack = require("webpack");
const path = require("path");
const MiniCssExtractPlugin = require("mini-css-extract-plugin");//for webpack4

let config = {
  entry: "./src/main.js",
  output: {
    path: path.resolve(__dirname, "./dist"),
    filename: "./main-bundle.js"
  },
   module: {
       rules: [{
          test: /\.js/,    exclude: /(node_modules|bower_components)/,
          use: [{    loader: 'babel-loader', options: {  presets: ['@babel/preset-env']  }     }]
       },{
          test: /\.css$/,
          use:  ['style-loader' , MiniCssExtractPlugin.loader, 'css-loader' ]
          } ,{
          test: /\.scss$/,
          use:  ['style-loader' , MiniCssExtractPlugin.loader, 'css-loader' , 'sass-loader']
          }]   },
     plugins: [       new MiniCssExtractPlugin({ filename: 'styles-bundle.css'})      ]
}
module.exports = config;
```

==> ceci permet de générer le fichier *dist/**styles-bundle.css*** (référencé depuis une page html)

```
body {  background-color: #2c3e50; }
h3 {  color: red; }
```

XVIII - Annexe – Rxjs

1. introduction à RxJs

1.1. Principes de la programmation réactive

La programmation réactive consiste essentiellement à programmer un enchaînement de traitements asynchrones pour réagir à un flux de données entrantes .
Un des intérêts de la programmation réactive réside dans la souplesse et la flexibilité des traitements fonctionnels mis en place :
- En entrée, on pourra faire librement varier la source des données (jeux de données statiques , réponses http , input "web-socket" , événement DOM/js , ...)
- En sortie, on pourra éventuellement enregistrer plusieurs observateurs/consommateurs (si besoin de traitement en parallèles . par exemple : affichages multiples synchronisés) .

1.2. RxJs (présentation / évolution)

RxJs est une bibliothèque **javascript** (assimilable à un mini framework) spécialisée dans la programmation réactive .
Il existe des variantes dans d'autres langages de programmation (ex : RxJava , ...) .

RxJs s'est largement inspiré de certains éléments de programmation fonctionnelle issus du langage "**scala**" (map ,flatMap , filter , reduce , ...) et a été à son tour une source d'inspiration pour "***Reactor***" utilisable au sein de Spring 5 .

RxJs a été dès 2015/2016 mis en avant par le framework Angular qui à fait le choix d'utiliser "Observable" de RxJs plutôt que "Promise" dès la version 2.0 (Angular 2) .

Depuis, le framework "Angular" a continuer d'exploiter à fond la bibliothèque RxJs .
Cependant , les 2 frameworks ont beaucoup évolué depuis 2015/2016 .

La version **4.3** de **Angular** a apporter de grandes simplifications dans les appels de WS-REST via le service **HttpClient** (rendant *obsolète* l'ancien service *Http*) .

La version 6 de Angular a de son coté été ***restructurée*** pour intégrer les gros changements de **RxJs 6** . Heureusement, pas de bouleversement en V7 (tranquille continuité).

La version 6 de RxJs s'est restructurée en profondeur sur les points suivants :
- changement des éléments à importer (nouvelles syntaxes pour les import { } from "")
- changements au niveau des opérateurs à enchaîner (plus de préfixe , pipe() , ...) .

1.3. Principales fonctionnalités d'un "Observable"

Observable est la structure de données principale de l'api "RxJs" .

- Par certains cotés , un "Observable" ressemble beaucoup à un objet "Promise" et permet d'enregistrer élégamment une suite de traitements à effectuer de manière asynchrone .
- En plus de cela , un "Observable" peut facilement être manipulé / transformé via tout un tas d'opérateurs fonctionnels prédéfinis (ex : map , filter , sort , ...)
- En outre , comme son nom l'indique , un "Observable" correspond à une mise en oeuvre possible du design pattern "observateur" (différents observateurs synchronisés autour d'un même sujet observable).

2. Fonctionnement (sources et consommations)

```
Source_configurée_et_initialisée
.pipe(
  callback_fonctionnelle_1 ,
  callback_fonctionnelle_2 ,
  ....
) .subscribe(callback_success , callback_error , callback_terminate)
```

NB : les paramètres callback_error et callback_terminate de .subscribe() sont facultatifs et peuvent donc être omis s'ils ne sont pas utiles en fonction du contexte.

NB : il faut (en règle général , sauf cas particulier/indication contraire) appeler .subscribe() pour que la chaîne de traitement puisse commencer à s'exécuter .

3. Réorganisation de RxJs (avant et après v5,v6)

La version 6 de RxJs a été beaucoup restructurée :
-plus de préfixe "Observable." devant of() et autres fonctions
-pipe() nécessaire pour enchaîner une série d'opérateurs (ex : map , filter , ...)
-réorganisation des éléments à importer

Quelques correspondances "avant/après" pour une éventuelle migration :

anciennes versions de RxJs (ex : v4)	versions récentes de RxJs (ex : v6)
Observable.of(data) ;	of(data) ;
import { } from "" ;	import { Observable, of } from "rxjs"; import { map , flatMap ,toArray ,filter} from 'rxjs/operators';

3.1. Imports dans projet Angular / typescrit

```
import { Observable , of } from 'rxjs';
import { map , flatMap ,toArray ,filter} from 'rxjs/operators';
```

exemple d'utilisation (dans classe de Service) :

```
public rechercherProduitSimu$(prixMaxi : number) : Observable<Produit[]> {
    let tabProduit = [
      { numero : 1 , label : "produit 1" , prix : 50 } ,
      { numero : 2 , label : "produit 2" , prix : 30 } ,
      { numero : 3 , label : "produit 3" , prix : 80 } ,
      { numero : 4 , label : "produit 4" , prix : 500 }
    ]
    return of(tabProduit)
        .pipe(
                flatMap(pInTab=>pInTab) ,
                map((p : Produit)=>{p.label = p.label.toUpperCase(); return p;}) ,
                filter((p) => p.prix <= prixMaxi) ,
                toArray()
        );
  }
```

3.2. Imports "umd" dans fichier js (navigateur récent)

EssaiRxjs.html

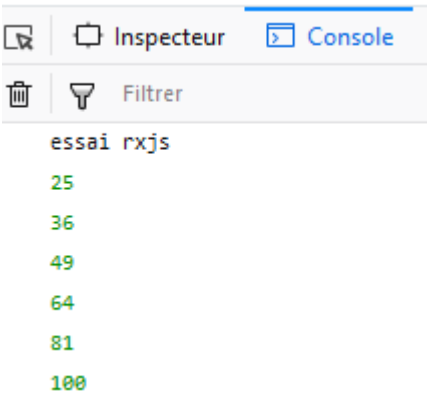
```
…
<body>
  Essai Rxjs (Observable, pipe , subscribe)
  <hr/>
  <p>ouvrir la console web du navigateur</p>
  <script src="lib/rxjs.umd.min.js"></script>
  <script src="js/essaiRxjs.js"></script>
</body>
...
```

avec rxjs.umd.min.js récupéré via https://unpkg.com/rxjs/bundles/rxjs.umd.min.js ou bien via une URL externe directe (CDN) <script src="https://unpkg.com/rxjs/bundles/rxjs.umd.min.js"></script>

NB : "The global namespace for rxjs is rxjs"

essaiRxJs.js

```
console.log('essai rxjs');

const { range } = rxjs;
const { map, filter } = rxjs.operators;

range(1, 10).pipe(
  filter(x => x ≥ 5),
  map(x => x * x)
).subscribe(x => console.log(x));
```



4. Sources classiques générant des "Observables"

4.1. Données statiques (tests temporaires , cas très simples)

```
let jsObject = { p1 : "val1" , p2 : "val2" } ;
of(jsObject)...subscribe(...) ;

let tabObj = [ { ...} , { ... } ] ;
of(tabObj)...subscribe(...) ;

of(value1 , value2 , ... , valueN)...subscribe(...) ;
```

4.2. Données numériques : range(startValue,endValue)

```
range(1, 10).subscribe(x => console.log(x)) ;
1
2
...
10
```

4.3. source périodique en tant que compteur d'occurence

```
const obsvI1 = interval(1000 /*ms*/);
//subscriptionObjsvI1 is the result of .subscribe() call
const subscriptionObjsvI1 = obsvI1.subscribe(n =>
  { console.log(`the number of interval  occurrence (starting at 0) is ${n} `);
    if(n>=5) {
          subscriptionObjsvI1.unsubscribe(); //stop if n>=5
    }
  });
```

4.4. source en tant qu'événement js/DOM

```
import { fromEvent } from 'rxjs';

const el = document.getElementById('my-element');

// Create an Observable that will publish mouse movements
const mouseMoves = fromEvent(el, 'mousemove');

// Subscribe to start listening for mouse-move events
const subscription = mouseMoves.subscribe((evt /*: MouseEvent */) => {
  // Log coords of mouse movements
  console.log(`Coords: ${evt.clientX} X ${evt.clientY}`);

  // When the mouse is over the upper-left of the screen,
  // unsubscribe to stop listening for mouse movements
  if (evt.clientX < 40 && evt.clientY < 40) {
    subscription.unsubscribe();
  }
});
```

4.5. source en tant que réponse http (sans angular HttpClient)

```
import { ajax } from 'rxjs/ajax';

// Create an Observable that will create an AJAX request
const apiData = ajax('/api/data');
// Subscribe to create the request
apiData.subscribe(res => console.log(res.status, res.response));
```

4.6. source en tant que données reçues sur canal web-socket

...

5. Principaux opérateurs (à enchaîner via pipe)

Rappel (syntaxe générale des enchaînements) :

```
Source_configurée_et_initialisée
.pipe(
  callback_fonctionnelle_1 ,
  callback_fonctionnelle_2 ,
  ....
) .subscribe(callback_success , callback_error , callback_terminate)
```

avec plein de variantes possibles

Exemple :

Principaux opérateurs :

map	Transformations quelconques (calculs , majuscules , tri , ….)
flatMap	
toArray	
filter	Filtrages (selon comparaison, ….)

5.1. map() : transformations

En sortie , résultat (retourné via return) d'une modification effectuée sur l'entrée .

Exemple 1:

```
const obsNums = of(1, 2, 3 ,4 ,5);
const squareValuesFunctionOnObs = map((val) => val * val);
const obsSquaredNums = squareValuesFunctionOnObs(obsNums);
obsSquaredNums.subscribe(x => console.log(x));
```

// affiche 1 4 9 16 25

Exemple 2:

```
const obsStrs = of("un" , "deux" , "trois");
obsStrs.pipe(
```

```
                map( (s) => s.toUpperCase() )
            )
    .subscribe(s => console.log(s));
// affiche UN   DEUX   TROIS
```

5.2. flatMap() et toArray()

De façon à itérer une séquence d'opérateurs sur chaque élément d'un tableau tout en évitant une imbrication complexe et peu lisible de ce type :

```
observableSurUnTableau
.pipe(
      map((tableau)=>{
                return tableau.map(
                  (itemInTab)=>{itemInTab.label = itemInTab.label.toUpperCase();
                              return itemInTab;}
                 );
                })
   );
```

on pourra placer une séquence d'opérateurs qui agiront sur chacun des éléments du tableau entre flatMap() et toArray() :

```
observableSurUnTableau
.pipe(
   flatMap(itemInTab=>itemInTab) ,
   map(( itemInTab )=>{ … }) ,
   filter((itemInTab) => itemInTab.prix <= 300 ) ,
    toArray()
).subscribe( (tableau) => { … } ) ;
```

5.3. filter()

```
const obsVals = of(12 , -15 , 30 , -8 , 40);
obsVals.pipe(
                filter( (v) => v >= 0)
                )
    .subscribe(v => console.log(v));
//affiche 12 30 et 40
```

```
range(1,10).pipe(
                filter( (v) => v % 2 === 0 )
                )
    .subscribe(v => console.log(v  +" est une valeur paire"));
```

5.4. map with sort on array

```
const obsTab = of([{numero:1,label:'produit1',prix:40.0},
                   {numero:2,label:'produit2',prix:30.0},
                   {numero:3,label:'produit3',prix:35.0},
```

```
                        {numero:4,label:'produit4',prix:15.0},
                        {numero:5,label:'produit5',prix:35.0}
                        ]);
obsTab.pipe(
                map( (tab) => tab.sort( (p1,p2) => (p1.prix > p2.prix) ) )
                )
  .subscribe(t => console.log( JSON.stringify(t)) );
```

6. Passerelles entre "Observable" et "Promise"

6.1. Source "Observable" initiée depuis une "Promise" :

```
import { from } from 'rxjs';

// Create an Observable out of a promise :
const observableResponse = from(fetch('/api/endpoint'));


observableResponse.subscribe({
 next(response) { console.log(response); },
 error(err) { console.error('Error: ' + err); },
 complete() { console.log('Completed'); }
});
```

6.2. Convertir un "Observable" en "Promise"

```
observableXy.toPromise()
                .then(…)
                .catch(...)
```

XIX - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

2. TP