# JAVASCRIPT
# VERSION « MODERNE » : POST ES6
**1 - Bases du langage**

## SOMMAIRE


**Sommaire ..... 1**

**JS - 1 : INTRODUCTION ET tour complet ..... 4**

**JavaScript – Introduction ..... 4**

1 - Intro ..... 4
2 - Environnement de travail ..... 10
3 - Tester JavaScript en ligne : à éviter ! ..... 11
3 - Tutoriels et ressources références ..... 12
4 - Balise <script> ..... 13
5 – Exemples : 01 – 02 - 03 ..... 15
6 - Organisation du cours ..... 17

**JavaScript – Tour complet ..... 18**

Console Javascript ..... 18
04 - variable, addition, concaténation, prompt() – A tester ..... 21
05 – premières fonctions – A tester ..... 22
06 – premières listes – A tester et exercices ..... 24
07 - document.write() et test – A tester ..... 26
08 – document. getElementById() et boucle for – A tester ..... 27
09 – Button : programmation événementielle – A tester ..... 28
10 – onclick, onmouseover, on mouseout – A tester ..... 29
11 – fonctions et paramètres en sortie – Point théorique (facultatif) ..... 31
12 – Exercices ..... 33

**JS - 2 : Bases du code ..... 34**

**Référence complète ..... 34**

**Ré-introduction ..... 34**

Ré-introduction ..... 34
Le langage ..... 35
Principes ..... 36
Les interpréteurs JavaScript ..... 37

**Types et opérations de base ..... 38**

Les 3 + 1 types ..... 38
opérateur typeof ..... 40
Opérations de base ..... 42

**Variable ..... 44**

Principes et usages modernes : ES6+ ..... 44
Visibilité ..... 45
Variable globale : à éviter ! ..... 45
Utilisation d'une variable ..... 46
Conversions de types ..... 47
Expression et évaluation d'une expression ..... 48

**Affichage - Saisie ..... 49**

JavaScript output ..... 49
Console.log ..... 49

Affichage d'une fenêtre d'alerte, avec ou sans confirmation ..... 49
Affichage d'une fenêtre avec texte et champs de saisie ..... 49
textContent et innerHTML ..... 49
Write ..... 49
**Tests ..... 50**
**Boucles ..... 51**
**Fonctions ..... 52**
Principes ..... 52
Fonctions mathématiques prédéfinies : Math ..... 53
Fonctions de manipulation de chaines prédéfinies : String ..... 54
Fonctions de manipulation de date prédéfinies : Date ..... 55
Paramètres en sortie des fonctions : exemple 11 ..... 56
**Exercices – Série 1 ..... 59**
1 – Calculs sur des figures ..... 59
2 – Jour de la semaine ..... 59
3 – Table de multiplication ..... 60
4 : Compteur de clics ..... 60

**JS - 3 : Tableaux et objets - JSON ..... 61**
**Installation des fichiers de tests ..... 61**
**Bases du code – 2 – tableaux et objets (ou structures) ..... 62**
1 - Tableaux : exemple 1 ..... 62
2 - Objet ( = structure) en JS : exemple 2 ..... 65
3 - Boucles spéciales : exemples 1 et 2 ..... 67
4 – Tableau de d'objets ( = structures) : exemple 3 ..... 70
5 - JSON ..... 71
6 – Objet (= structure) avec fonctions : exemple 4 ..... 73
**Exercices – Série 2 ..... 74**
0 – JSON ..... 74
1 – Tableau de notes ..... 76
2 - Tableau d'élèves avec des notes – Tri d'un objet (= structure) ..... 78
3 - Tableau d'élèves avec des notes – creerEleve ..... 79
4 – Jeu de grammaire ..... 80
5 – Pipotron, Poétron ..... 81

**JS - 4 : JS moderne : ES6/2015 ..... 82**
**JS moderne : ES6/2015 - Standard ECMAScript ..... 82**
**L'essentiel de ES6/2015 ..... 82**
**let ..... 83**
Principes ..... 83
Portée ..... 83
Utilité ..... 83
Usage ..... 83
Variables globales : rappels ..... 83
**const ..... 84**
Principes ..... 84
Portée ..... 84
Utilité ..... 84
Usage ..... 84
**POO ES6 ..... 85**
Principes ..... 85
Exemple de classe ..... 87
**Fonction fléchée : fat arrow : => ..... 88**
Exemple -> ici ..... 88

**Fonction fléchée : fat arrow : => ....89**
Exemple -> ici....89
**this et =>....90**
Exemple -> ici....90
**Le mode strict ....91**
**Destructuration ....92**
Exemples -> ici....92
Sur un objet ....92
Sur un tableau....93
**Exemples : un composant compteur ....94**
TP2-1 : création d'un composant compteur ....94
Présentation de 4 versions d'un petit compteur : ....94
TP2-1 : mise à jour du composant compteur ....94

**JS - 5 : Anciennes versions de POO ....95**
**Anciennes versions de POO - facultatif ....95**
**Bases : objet (=structure) avec méthode – exemple 1....95**
Objet (=structure) avec méthode....95
Utilisation....96
Limitation....96
**POO - Version 1....97**
**POO - Version 2 : avec attribut prototype ....98**
ajout d'une fonction avec prototype en cours de code ....98
ajout d'une fonction avec prototype sur des classes natives ....98
Array.prototype – méthodes filter, every, some....99
**POO – Version 3 : avec un objet prototype et Object.create() – exemple 2....100**
Création de Classe : notion de prototype – technique 1 ....100
Création de Classe : fonction d'initialisation – technique 2....101
Utilisation du new....101
**POO – Version 3 : Héritage - exemple 2....102**
Principes ....102
Application....102
Exercice 1 : une IHM pour l'exemple 2....103
**POO – tableau d'objets ....104**
Exemple 3 ....104
Exercice 2 : une IHM pour l'exemple 3....105
Exercice 3 : tableau d'élèves avec notes et photos en POO ....105

Edition : octobre 2019 / ES6-2015 – MAJ fév 2024

# JS - 1 : INTRODUCTION ET TOUR COMPLET

**Dans le cours :**
Les exemples sont présentés dans un chapitre en vert.
Les exercices à faire sont présentés dans un chapitre en jaune.
Les phrases importantes sont en gras ou surlignés en bleu ciel

## JavaScript – Introduction

### 1 - Intro

**Qu'est-ce que c'est ?**

Le JavaScript (JS) est un langage de script (comme le HTML ou le PHP).
Le code JS qui s'ajoute à la page HTML dans une balise <script>.
Il est interprété par le navigateur. Il permet de **rendre plus dynamique et interactive la page HTML**, sans passer par le serveur.
Il peut aussi communiquer avec le serveur en utilisant les technologies AJAX.
C'est un langage objet non typé, comme le python !
**Rien à voir avec Java !**

**Historique**

Wiki : https://fr.wikipedia.org/wiki/JavaScript
Inventé en **1995** par Brendan Eich pour **Netscape** premier navigateur Web populaire (l'ancêtre de Firefox).
Aujourd'hui, tous les navigateurs comprennent le JavaScript aujourd'hui.
Standardisé par l'ECMA International sous le nom d'ECMAScript.
ECMAScript 5, version standardisée sortie en 2009.
**ECMAScript 6 : dernière version standardisée, sortie en 2015.**

**Désactiver JavaScript**

On peut sur chaque navigateur, désactiver JS. Le JavaScript ne se substitue donc pas aux vérifications qu'il faut faire côté serveur.
Beaucoup de sites ne pourront pas fonctionner sans JS.

**Bonnes pratiques**

JS a évolué depuis sa création. Les premiers usages peuvent être aujourd'hui considéré comme obsolètes et relevant de mauvaises pratiques.
Donc il faut faire attention à ne pas copier-coller n'importe quel exemple récupéré sur le web !

## JS moderne : ES6/2015 - Standard ECMAScript

- ES6 = ES2015 = ES6/2015 : une révolution pour JavaScript.
- **ES6/2015 : « sucre syntaxique » pour les Classes**. JavaScript n'a pas de classes. La fonctionnalité des classes est reprise par les prototypes d'objet et le « sucre syntaxique pour les Classes » apparu avec ES6.
- Pour les prototypes, voir la fin de ce document.
- Pour une introduction à l'objet, voir : http://bliaudet.free.fr/article.php3?id_article=108 : on trouve un pdf et des exemples JavaScript ES6, Python et Java.
- Les grands framework JS s'appuient sur ES6 (React, Angular, …)

## Quelques références

- Historique : https://apprendre-a-coder.com/es6/
- Petite intro : https://www.wanadev.fr/21-introduction-a-ecmascript-6-le-javascript-de-demain/
- W3schools : https://www.w3schools.com/js/js_es6.asp
- Différences 5 et 6 : http://es6-features.org/#StringInterpolation
- Mozilla : https://developer.mozilla.org/fr/docs/Web/JavaScript/Nouveaut%C3%A9s_et_historique_de_JavaScript/Support_ECMAScript_2015_par_Mozilla
- Spécifications complètes : https://www.ecma-international.org/ecma-262/6.0/index.html

**Côté serveur : Node.js**

Apparition en 2009 de la plate-forme Node.js, qui permet d'écrire en JavaScript des applications Web très rapides.
L'environnement Node.js exécute du JavaScript côté serveur pour générer du HTML dans lequel du JavaScript pourra toujours être exécuté côté client.
https://openclassrooms.com/courses/des-applications-ultra-rapides-avec-node-js/node-js-mais-a-quoi-ca-sert

**Développement front – Jquery – React – Vues.js**

Le JavaScript sert à améliorer le visuel de la page web. C'est un complément au CSS. Il participe à la spécialisation du travail entre Frontend et Backend.

Des bibliothèques existent pour faciliter le travail et découpler les usages (la logique métier) de la technique (le DOM).

**Jquery** : la plus bibliothèque la plus populaire jusque vers 2015. Lancée en 2006. Pour dynamiser les pages web => **devenue obsolète en 2019 !**
**Mais ça reste utilisé par Bootstrap**
https://www.w3schools.com/jquery/

**React et Vues.js** sont des bibliothèques Node.js pour du développement front.

⇒ **React :** facebook – 2013 : le plus utilisé
https://www.w3schools.com/react/default.asp

⇒ VueJS : 2014 : en progrès

React et/ou Vue.js sont un bon choix, quoiqu'il arrive.

- Leur architecture et philosophie sont proches. Découvrir l'un permet de prendre la main plus rapidement avec l'autre.
- La syntaxe de templating en quasi-HTML de VueJS est plus "naturelle" que le JSX de React

**Développement d'application : Framework JavaScript**

Avec du HTML et du JavaScript on peut développer des applications côté client. Des bibliothèques et/ou frameworks permettent de faciliter le travail.

- **Node.js** : google-chrome – 2009 : la base des framework avec le **npm** (node package manager).
  ⇒ Angular : google – 2010-2012 : beaucoup de gens pas intéressé
  ⇒ Express : pour la réalisation d'API.
- Les méthodes d'écriture de code JavaScript "modernes" seront nécessaires pour les apprivoiser (POO, structuré, MVC, EcmaScript, etc.)

➢ ***AngularJS***

C'est une autre bibliothèque-framework JavaScript créé en **2009 chez Google**. Pour développer des applications côté client.
https://www.w3schools.com/angular/angular_intro.asp

➢ ***JointJS***

C'est un site qui propose des bibliothèques payantes permettant de développer des applications comme par exemple un modeleur UML :
http://resources.jointjs.com/demos/kitchensink
L'intérêt ici est d'avoir un outil en ligne. On peut tester le modeleur UML.
Il y a là un champ de développement commercial ou libre considérable !
https://www.jointjs.com/opensource

➢ ***draw.io***

Application JS en ligne : modeleur UML et autres syntaxes. Gratuit !
https://www.draw.io

## 2 - Environnement de travail

**Editeur orienté front-end**

- ***VS Code ou Sublime Text*** ***(ou Notepad ++ ou  Bracket ou autre)***

**Navigateur**

- ***Chrome ou Firefox***

**Inutile : environnement WAMP**

On peut installer les fichiers de test dans un environnement WAMP (dans le répertoire www du serveur local).
Ce n'est pas obligatoire puisqu'on fonctionne, dans un premier temps (sans AJAX), uniquement côté client.

## 3 - Tester JavaScript en ligne : à éviter !

**« Bac à sable » (coder en ligne) : à l'occasion pour tester ses codes**

Pour faire des tests éventuellement.
Il faut aussi bien prendre le contrôle sur les fichiers.
http://codepen.io/ : « start coding »
https://jsfiddle.net/
http://jsbin.com/

**Exemple**

- https://codepen.io/wlabarron/pen/yYrPRQ?editors=0011

Dans la zone « your code » de JS, écrivez :

```
let output = "hello world"
console.log(output)
```

Dans la zone HTML, écrivez :

```
<h1Test</h1>
```

Ensuite, cliquez sur « run »

## 3 - Tutoriels et ressources références

**Mozilla**

**Les bases :**
https://developer.mozilla.org/fr/docs/Apprendre/Commencer_avec_le_web/Les_bases_JavaScript
**Intro et tutos :** https://developer.mozilla.org/fr/docs/Web/JavaScript
**Syntaxe :** https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference#Les_instructions

**w3school**

**JS de base :** https://www.w3schools.com/js/
**ES6 :** https://www.w3schools.com/js/js_es6.asp

**Eloquent JavaScript**

Le site est moins à jour par rapport à l'ES2015
*** sommaire : http://fr.eloquentjavascript.net/contents.html
En anglais : http://eloquentjavascript.net
En français : http://fr.eloquentjavascript.net

## 4 - Balise <script>

Le code JS se place entre les balises <script> et </script>

### Où mettre la balise <script> ?

On peut placer des balises <script> dans le <head> ou dans le <body>.
De préférence on ne met qu'une balise <script> par fichier HTML.

Aujourd'hui, on préfère placer **la balise <script> à la fin du <body>, juste avant le </body>** : en effet pour bien s'exécuter, le JS doit d'abord avoir chargé la page HTML (mais l'affichage se fait après le chargement complet, donc après l'exécution du code JavaScript).

### Ce qu'il faut faire : code JS avant le </body>

```
<body>
  code de la page HTML

  <script>
    code JS
  </script>
</body>
```

**Bonne pratique ultime : inclusion d'un fichier externe, avant le </body>**

```
<body>
  code de la page HTML

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

On peut inclure un fichier contenant du code JS avec l'attribut « src » dans la balise <script>.
On fera ça dans tous nos exemples.

**Si on met la balise <script> dans le <head> :**

On utilise le « window.onload » pour dire que le code JS sera exécuté après que la page ait été chargée. On met le code dans la fonction, entre les accolades.

```
<script>
  window.onload=fonction(){
    code JS
  }
</script>
```

## 5 – Exemples : 01 – 02 - 03

### Chargez les fichiers

Les exemples du cours sont dans un fichier zip fournis avec l'article du cours.

- JavaScript_01_exemples_01_bases_et_tour.zip

Téléchargez le zip et désippez-le. On obtient un répertoire :
tour JavaScript_01_exemples_01_bases_et_tour
Mettez ce répertoire : dans un dossier « Partie_1 » que vous aurez mis dans un dossier JavaScript.
Ce dossier JavaScript peut être mis où vous voulez sur votre machine. Vous pouvez le mettre dans le répertoire web « www » du serveur WAMP mais ce n'est pas utile. Les fichiers HTML contenant du JavaScript peuvent être exécutés sans environnement serveur.
Ce dossier contient les codes des exemples de 01 à 11.

## 01 - Hello world !

- ***HTML***

```
<body>
    Ce qu'on veut dans la page
    A la fin, une balise script
    <script>
        alert('Hello world!');
    </script>
</body>
```

La partie JavaScript est écrite dans la balise script.
On utilise la fonction alert.

## 02 - balise script seule

- ***HTML***

```
<script>
    alert('Hello world!');
</script>
```

Ca fonctionne aussi sans HTML, évidemment ! C'est du HTML !

## 03 - Hello world ! fichier JS

- ***HTML***

```
<body>
    <script src="hello.js"></script>
</body>
```

- ***JavaScript : hello.js***

```
alert('Hello world!');
```

On peut aussi coder le JavaScript dans un fichier séparé.
C'est plus lisible.

## Exercice : testez ces trois fichiers

## 6 - Organisation du cours

**Programmation impérative**

Le JS permet de faire de la programmation impérative classique : variables, tests, boucles, fonctions, etc.

**DOM**

Il permet ensuite d'interagir avec la page HTML : il utilise pour cela une API : le DOM.

**Programmation évènementielle**

Il permet aussi de gérer des événements : c'est de la programmation événementielle.

**AJAX – XML - JSON**

Enfin, il permet de communiquer avec le serveur : c'est la partie AJAX avec les formats de communication XML et surtout JSON aujourd'hui.

## Console Javascript

### Le problème

Le JavaScript permet d'écrire dans la page HTML.
Comment tester notre code ? Comment afficher le contenu des variables qu'on va manipuler sans modifier pour autant la page web ?

### Accès à la console JavaScript

Dans le navigateur, on peut afficher une console JavaScript.

- ***Firefox***

Outils / Développement Web / Console Web : onglet Console

- ***Chrome***

Afficher / Option pour les développeurs / Console JavaScript : onglet Console

- ***Safari***

Développement / Afficher la console JavaScript : onglet Console

- ***Etc.***

### Raccourcis clavier : alt-cmd-i / onglet console

Outils de développement : alt-cmd-i
Onglet console.

## Taper du code directement

On peut taper du code directement, comme en python, ou dans une console SQL

```
> a=3
> a
> b=a+a
> b
> console.log("b = "+b)
> alert("b = ", b)
> document.write("<h1>bonjour</h1>")
> document.write("<h2>resultat = "+b+"</h2>")
> document.querySelector("h1").style.backgroundColor="aqua";

etc.
```

- ***console.log()***

L'instruction « console.log() » permet d'afficher des variables et du texte dans le mode console à partir de code JavaScript.

- ***alert(b)***

L'instruction « alert » permet d'ouvrir une fenêtre avec un message.

- ***document.write(« … »)***

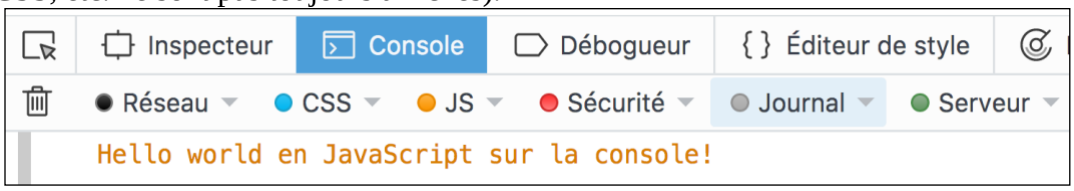
L'instruction « document.write » permet d'afficher d'ajouter du code HTML dans la page.

- ***document.querySelector(« … »)***

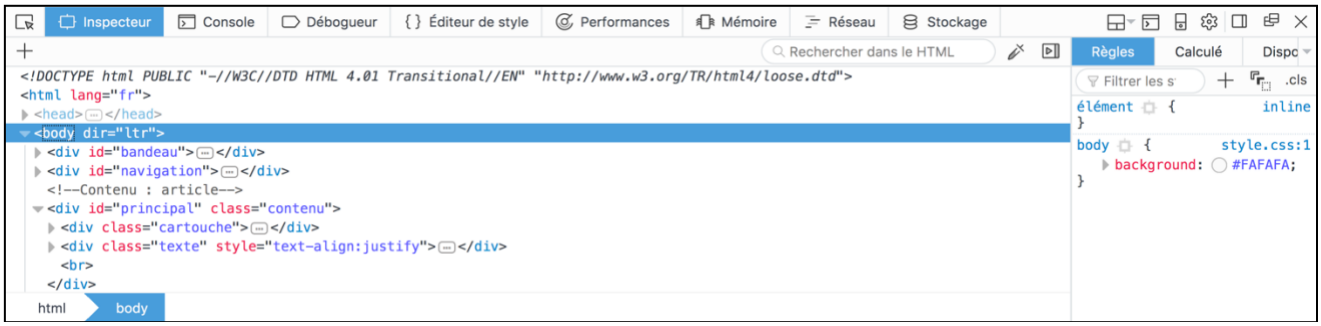
Ici, on récupère une balise et on modifie son style.

➢ ***Mode console / Journal***

Pour voir l'affichage des instructions console.log, on ouvre le mode console / Journal (les onglet Réseau, CSS, etc. ne sont pas toujours affichés).




## Fonctionnalités de l'environnement dans Firefox




- **Inspecteur** : pour parcourir le code HTML
- **Console** : pour avoir les erreurs et le « **mode console** »
- **Débogueur** : pour mettre des points d'arrêt dans le code
- **Editeur de style** : pour voir le CSS. On peut le changer pour voir les résultats
-  : pour faire apparaître la console dans l'inspecteur, par exemple.
-  : pour présenter les outils verticalement.
-  : pour présenter les outils dans une fenêtre à part.
-  : pour accéder aux configuration : par exemple le thème sombre.

## 04 - variable, addition, concaténation, prompt() – A tester

- ***HTML : index.html***

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

- ***JavaScript : script.js***

```
// Déclaration des variables
// mieux vaut déclarer les variables au début
// on préfère let à var ou rien
// on checke l'usage des variables en cliquant dessus

let v1 = 5
let v2 = 10
let resultat = v1+v2
let message = ''

alert('v1+v2: '+v1 + ' + ' + v2 + ' = ' + resultat)
message = '1 : Le double du résultat est : ' + resultat*2
alert(message);
console.log(message)

v1 = prompt('Entrez le premier chiffre :')
v2 = prompt('Entrez le second chiffre :')
resultat = parseInt(v1) + parseInt(v2);

alert(v1 + ' + ' + v2 + ' = ' + resultat);
console.log(v1 + ' + ' + v2 + ' = ' + resultat);

message = '2 : Le double du résultat est : ' + resultat*2
alert(message)
console.log(message)
```

- alert pour afficher une variable dans une fenêtre
- prompt pour lire une variable dans une fenêtre
- parseInt() permet de transformer du texte en entier. Il existe aussi une fonction parseFloat(). On peut aussi ajouter parseFloat(… ).toFixed(2) pour limiter à deux chiffres après la virgule.
- On peut additionner avec le signe « + » ou faire une concaténation.

http://www.w3schools.com/js/js_output.asp

## 05 – premières fonctions – A tester

### 05-fonction

- ***HTML : index.html***

```
<script src="cours2.js"></script>
```

- ***JavaScript : script.js***

```
function fonction_1(){
  let v1 = 5, v2 = 10
  let resultat = v1+v2
  let texte = '' // mieux peut commencer par déclarer texte
  alert(v1 + ' + ' + v2 + ' = ' + resultat)
  texte = 'f_1: Le double du résultat est : ' + resultat*2
  alert(texte)
  console.log(texte)
}

function fonction_2(v1, v2){
  let resultat = v1 + v2
  let texte = ''  // mieux peut commencer par déclarer texte
  alert(v1 + ' + ' + v2 + ' = ' + resultat)
  texte = 'f_2: Le double du résultat est : ' + resultat*2; //
sans let
  alert(texte)
  return texte
}

// MAIN
// le code s'exécute dans l'ordre : d'abord on déclare les
fonctions
// puis on les utilise dans le main
//
fonction_1(); // appel à la fonction

let w1 = prompt('Entrez le premier chiffre :')
let w2 = prompt('Entrez le second chiffre :')

let message = fonction_2(parseInt(w1), parseInt(w2)) // appel à
la fonction
console.log(message) // affichage de la variable globale de f_v2
```

On reprend le calcul précédent et on écrit des fonctions.
Une première fonction sans paramètre.
Une deuxième fonction avec paramètre.
La variable « texte » dans f_v2 est globale : on peut l'afficher après l'appel à la fonction. C'est plutôt à éviter.

**05-fonction-2**

- ***HTML : index.html***

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

- ***JavaScript : script.js***

```
console.log("Tests de console.log")

let a=5, b=3
console.log(a + ' x ' +b + ' = ' + a*b)

function direBonjour(prenom) {
    let message = "Bonjour " + prenom + " !"
    return message
}

console.log(direBonjour("Baptiste"))
console.log(direBonjour("Sophie"))

console.log("Au revoir !")
```

console.log permet d'écrire dans la console de log. C'est utile pour déboguer son programme.

## 06 – premières listes – A tester et exercices

- ***HTML : index.html***

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

- ***JavaScript : script.js***

```
console.log("Tests de liste, pop, push, for, sort et forEach")

let liste=[10, 5, 20, 15, 25]
console.log(liste)

let elt = liste.pop()
console.log(elt)

liste.push(35)
console.log(liste)

for(let i=0; i<liste.length; i++){
    console.log(liste[i]*2)
}

liste.sort((a, b) => a - b)
console.log(liste)

liste.forEach((elt)=>{
    console.log(elt*100)
})

console.log("Au revoir !")
```

- ***Exercice***

- Installez node.js et tester le code avec node.js
  ⇒ Ouvrez un terminal dans le dossier de script.js
  ⇒ Tapez : node script.js

## ➢ *Exercice*

```
/*
    INTERLUDE :

    exercice : écrire une fonction qui mélange une liste
    on l'appellera : shuffle()

    pour faire ça :
        1 : on récupère un élément de la liste de départ "l" au
hasard
                Math.floor(Math.random() * array.length)
                permet de récupérer un indice au hasard
        2 : on ajoute l'élément dans une nouvelle liste : "l2"
        3 : on supprime l'élément trouvé au hasard de "l"
                la fonction splice permet de faire ça
                cherchez son fonctionnement
        => on répète ça tant que l n'est pas vide.
        4 : à la fin, on vide l2 dans l

        Testez la fonction dans une page HTML
        Testez la fonction avec Node
        Pour tester :
            on fabrique une liste de 10 éléments entre 0 et 100
au hasard
            on l'affiche
            on la trie, on l'affiche
            on la mélange, on l'affiche
            on la re-mélange, on l'affiche
            on la re-trie, on l'affiche
 */
```

## 07 - document.write() et test – A tester

- ***HTML : index.html***

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

- ***JavaScript : script.js***

```
document.write('<h1>Partie JavaScript : document.write écrit à la
fin de la page</h1>');

let prenom = prompt('Entrez votre prénom :');
let age = prompt('Entrez votre age :');

age = parseInt(age);

if(age>25){
  document.write('Désolé '+prenom+'<br>');
  document.write('Vous n\'avez pas droit à la carte Jeune');
}
else{
  document.write('Bonjour '+prenom+'<br>');
  document.write('Vous pouvez bénéficier de la carte Jeune');
}
```

- document.write permet d'écrire à la fin de la page HTML

## 08 – document. getElementById() et boucle for – A tester

- ***HTML : index.html***

```
    <body>
      <h1>Début de la page</h1>
      <div id="resultats"></div>
      <h1>Fin de la page</h1>

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

- ***JavaScript : cours.js***

```
// on fabrique le code HTML qu'on va ajouter
let innerHTML = '<h3>Table de 5 : </h3>'
for(let i=1; i<11; i++){
  innerHTML += '5 * '+i+' = '+5*i+'<br>';
}
let balise=document.getElementById('resultats_1');
balise.innerHTML = innerHTML

//Version compacte :
document.getElementById('resultats_2').innerHTML = innerHTML
```

- balise=document.getElementById('resultats') permet de récupérer la balise dont l'id vaut "resultat".
- balise.innerHTML est le texte qu'on place dans la balise
- Le « += » permet d'ajouter du texte dans la balise
- La boucle for est une boucle standard.
- Version compacte :

```
document.getElementById('resultats').innerHTML = '<h3>Table de 5
: </h3>';
```

## 09 – Button : programmation événementielle – A tester

L'objectif est de déclencher du code JavaScript en cliquant sur un bouton.

- ***HTML : index.html***

```
<body>
  <h1>Début de la page</h1>
    <fieldset>
      <p>Zone de démonstration</p>
      <p id="demo"></p>
    </fieldset>

      <button onclick="document.getElementById('demo').innerHTML
= 'bien cliqué sur le bouton 1'">Cliquez moi 1!!!</button>

      <button onclick="boutonTest()">Cliquez moi 2 !!!</button>
  <h1>Fin de la page</h1>

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

- ***JavaScript***

```
function boutonTest(){
  document.getElementById('demo').innerHTML =
    '<p style="background-color:aqua"> bien cliqué sur le bouton
2</p>'
}
```

- La balise <button> sert à créer un bouton qui actionnera du code JavaScript.
- On peut ajouter des attributs dans la balise <button> qui définisse un événement à l'origine de l'exécution d'un code JavaScript. Ici l'événement « onclick ». La valeur de l'attribut, c'est du code JavaScript à exécuter quand l'événement est déclenché.
- Pour la première balise <button>, le code JavaScript est un « document.getElementById »
- Pour la deuxième balise <button>, le code JavaScript est l'appel à la fonction boutonTest()
- La fonction boutonTest() est définit dans le fichier JavaScript qui est inclus dans la balise <script>
- Le bilan est que la page HTML contient du code JavaScript directement dans la balise <button>

## 10 – onclick, onmouseover, on mouseout – A tester

L'objectif est de déclencher du code JavaScript en fonction de certains événements (onclick, onmouseover, etc.), ces événements pouvant s'appliquer à n'importe quelle balise.
L'objectif est aussi d'ajouter ces événements dans le code JavaScript et pas dans la page HTML.

- ***HTML : index.html***

```
<body>
  <h1>Début de la page</h1>

  <p id="p1"> Premier paragraphe de test : cliquez moi pour
changer la couleur de fond</p>

  <p id="p2"> Deuxième paragraphe de test : passez sur moi pour
changer la couleur de fond</p>

  <p id="p3"
onclick="document.getElementById('p3').style.backgroundColor='yel
low';">
    Troisième paragraphe de test : cliquez moi pour changer la
couleur de fond
  </p>

  <h1>Fin de la page</h1>

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

- ***JavaScript***

```
// on préfère éviter le JavaScript dans le HTML

// Paragraphe p1 :
let baliseP1 = document.getElementById('p1');
baliseP1.onclick = function(){
  baliseP1.style.backgroundColor='aqua';
}

document.getElementById('p1').addEventListener("dblclick",
function (event) {
  baliseP1.style.backgroundColor='yellow';
})

// Paragraphe p2 :
let baliseP2 = document.getElementById('p2');
baliseP2.onmouseover = function(){
  baliseP2.style.backgroundColor='yellow';
}
baliseP2.onmouseout=function(){
  baliseP2.style.backgroundColor='';
}
```

- Dans le HTML, dans le premier <p> est repris en JavaScript.
- Dans le JavaScritpt : on récupère la balise p1.
- Sur cette balise on met dans l'attribut « onclick » une fonction qui définit l'action à réaliser.
- L'action à réaliser consiste à modifier le style.backgroundColor de la balise.

- Notez que le nom reprend celui du CSS : background-color, mais en « Camel Case » (pas de tiret, majuscule sur le deuxième mot).
- On fait la même chose sur la balise p2, mais cette fois sur les événements « onmouseover » et « onmouseout »
- Dans le code HTML, sur un paragraphe p3, on fait directement ce qu'on a fait sur le paragraphe p1.

## 11 – fonctions et paramètres en sortie – Point théorique (facultatif)

Présentation du problème des paramètres en sortie.
Seules les listes et les objets peuvent être modifiés.

```
// présentation théorique

////////////////////////////////////////////////////////////////
/////////
console.log("Inversion d'entiers passés en paramètre :
impossible");
function inverser(a, b){
  console.log("Dans la fonction inverser : entrée : a="+a+" -
b="+b);
  let tmp=a;
  a=b;
  b=tmp;
  console.log("Dans la fonction inverser : sortie : a="+a+" -
b="+b);
}

let a=5, b=10;
console.log("Avant inverser : a="+a+" - b="+b);
inverser(a,b);
console.log("Après inverser : a="+a+" - b="+b);

console.log("Avant inverser, en 'dur' : a=5, b=10");
inverser(5,10);

////////////////////////////////////////////////////////////////
/////////
console.log("Inversion d'un tableau de 2 éléments : possible");
function inverserTableauDe2(tab){
  console.log("Dans la fonction inverserTableauDe2 : entrée :
"+tab);
  let tmp=tab[0];
  tab[0]=tab[1];
  tab[1]=tmp;
  console.log("Dans la fonction inverserCouple : sortie : "+tab);
}

let tab = [2,4];
console.log("Tableau de départ : "+tab);
inverserTableauDe2(tab)
console.log("Tableau après inversion : "+tab);

////////////////////////////////////////////////////////////////
/////////
console.log("Inversion de 2 champs d'un objet : possible");
function inverserCouple(couple){
  console.log("Dans la fonction inverserCouple : entrée :
a="+couple.a+" / b="+couple.b);
  let tmp=couple.a;
  couple.a=couple.b;
  couple.b=tmp;
  console.log("Dans la fonction inverserCouple : sortie :
a="+couple.a+" / b="+couple.b);
```

```
}

let couple = {
  a:2,
  b:4
}
console.log("Structure de départ : a="+couple.a+" /
b="+couple.b);
inverserCouple(couple)
console.log("Structure après inversion : a="+couple.a+" /
b="+couple.b);


////////////////////////////////////////////////////////////
/////////
console.log("Inversion d'objet avec méthode");
let objet={
  a:5,
  b:10,
  toString:function(){
    return "a="+this.a+" - b="+this.b;
  },
  inverserCouple:function(){
    console.log("Dans la fonction objet inverserCouple : entrée :
a="+this.a+" / b="+this.b);
    tmp=this.a;
    this.a=this.b;
    this.b=tmp;
    console.log("Dans la fonction objet inverserCouple : sortie :
a="+this.a+" / b="+this.b);
  }
}
console.log("Objet de départ : "+objet.toString());

inverserCouple(objet)
console.log("Objet après inverserCouple(objet):
"+objet.toString());

objet.inverserCouple();
console.log("Objet après objet.inverserCouple():
"+objet.toString());
```

## 12 – Exercices

A ce stade, on peut faire les exercices JS-2 - série 1
Ils sont environ page 62 du poly de cours.
Pour ces exercices, on peut s'appuyer sur les codes du tour complet et sur les éléments de syntaxe du chapitre « Bases du code ».

1 – Calculs sur des figures
2 – Jour de la semaine
3 – Table de multiplication
4 : Compteur de clics

# JS - 2 : BASES DU CODE

## Référence complète

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions

## Ré-introduction

### Ré-introduction

- Pourquoi une ré-introduction ? Parce que JavaScript : le langage de programmation le plus incompris au monde.
- Souvent raillé comme étant un simple jouet mais langage très puissant.
- Il y a une première vie du JavaScript avec JQuery, côté client et de l'AJAX
- Puis une deuxième vie du JavaScript côté serveur, avec Node.js, Angular, React, Express, etc.
- Avec cette deuxième vie, le JavaScript s'est développé : le code est devenu plus propre et naturellement objet. De nouvelles bibliothèques sont apparues côté client comme React ou Vue.js et des framework sont apparus côté serveur comme Angular ou Express. Ainsi, on trouve de nombreuses applications JavaScript de premier plan : https://www.draw.io
- Une connaissance approfondie de cette technologie est une **compétence importante pour tout développeur Web**.

## Le langage

- Créé en **1995** par Brendan Eich, un ingénieur de **Netscape**.
- Rapidement soumis à l'Ecma International, organisation de normalisation européenne => première édition du **standard ECMAScript en 1997** (ES1 = ES1997). Aujourd'hui ES6.
- ES6=ES2015 : sixième édition qui apporte des nouveautés majeures, publié en juin 2015.
- Conçu pour s'exécuter comme un langage de script dans un environnement hôte : c'est à cet environnement de fournir des mécanismes de communication avec le monde extérieur.
- L'environnement hôte le plus commun est un **navigateur**, mais il en existe bien d'autres.
- D'autres interpréteurs JS existent :
  - ✓ dans Adobe Acrobat, Photoshop, les images SVG, le moteur de widgets de Yahoo!,
  - ✓ des environnements côté serveur tels que **Node.js,**
  - ✓ les bases de données NoSQL telles que Apache CouchDB,
  - ✓ les ordinateurs embarqués
  - ✓ des environnements de bureaux comme GNOME (interface graphique très populaire des systèmes d'exploitation GNU/Linux).
- Le développement d'application s'est développé avec JS. On pense à React ou Angular. Mais aussi au développement mobile avec React-Native.

**Principes**

- JavaScript est un **langage dynamique multi-paradigmes :**
  ⇒ procédural,
  ⇒ objet,
  ⇒ événementiel.
- Cf : http://bliaudet.free.fr/IMG/pdf/Introduction-a-la-POO-Premiers-diagrammes-de-classes-UML.pdf
- Il dispose de :
  ⇒ types,
  ⇒ opérateurs,
  ⇒ objets natifs
  ⇒ méthodes.
- Sa syntaxe s'inspire des langages Java et C.
- Le JavaScript d'origine n'a pas de classes. Mais la fonctionnalité des classes est reprise par les prototypes d'objet puis directement avec l'ES6/ES2015.
- **Spécificité du JavaScript : les fonctions sont des objets**. On peut donc stocker ces fonctions dans des variables et les transmettre comme n'importe quel objet.

## Les interpréteurs JavaScript

**Les navigateurs :**

Tous les navigateurs offrent un mode console qui permet de faire du JavaScript

**Node.js**

- ***Node permet de démarrer un interpréteur.***

Pour vérifier si Node est installé :

```
C:>node -v
```

Pour installer Node : **Node.js**

- ***Démarrer l'interpréteur node***

```
C:>node
> a=3
3
>
```

## Types et opérations de base

### Les 3 + 1 types

#### 3 types simples :number, string, boolean

Le type d'une valeur détermine son rôle et les opérations qui lui sont applicables.
Les principaux types de bases du JS sont : nombre, chaîne de caractères, booléen

Type number : entier ou réel. Les réels s'écrivent avec un « . »
Type string : chaîne de caractère : entre guillemets ou apostrophes
Type boolean : true et false, en minuscules

#### 1 type complexe : object – Le JSON

JS permet de définir des **tableaux** et des **objets** (= structure)
Tous les types complexes sont des « object » (les tableaux et les objets).
Le JSON : JavaScript Object Notation, c'est la syntaxe des objets en JavaScript. C'est une syntaxe standard qu'on utilise pour les API et qu'on retrouve en Python.

➢ ***Les tableaux***

- const tabAnimaux = ["chien", "chat", "poule"]; // on utilise un const : on peut ajouter, modifier ou supprimer dans le tableau, mais on ne peut pas mettre un nouveau tableau.
- a.length; // 3

➢ ***Les objets***

- let obj = {};

```
const obj = {
  "nom": "Carotte",
  "for": "Max",
  "details" : {
    "couleur" : "orange",
    "taille" : 12
  }
};
obj.details.couleur; // orange
obj["details"]["taille"]; // 12
```

- On peut ne pas mettre de guillemets sur pour le nom des propriétés, mais ce n'est pas l'usage :

```
const obj = {
  nom: "Carotte",
  for: "Max",
  details: {
    couleur: "orange",
    taille: 12
  }
};
obj.details.couleur; // orange
obj["details"]["taille"]; // 12
```

## opérateur typeof

### typeof de number, de string, de boolean

```
typeof 1;              typeof(1)          // number
typeof 1.1;            typeof(1.1)        // number

a=5;
typeof a ;             typeof (a)         // number

typeof "hello";                           // string
typeof true;                              // boolean
typeof (1==1);                            // boolean
```

**typeof d' « object »**

les tableaux et les objets (= structure) sont de type « object »

- ***tableau***

```
typeof [1, 2]          // object // tableau de 2 entiers
tab=[1, 2]
typeof tab             // object // tableau de 2 entiers
typeof tab[0]          // number
```

- ***objet ( on parlait de « structure » en langage C )***

```
typeof {nom :'toto', age :15}        // object à 2 attributes

personne= {nom :'toto', age :15}
typeof personne                      // object à 2 attributs

typeof personne.nom                  // string
```

## Opérations de base

### Principe

Chaque type permet d'accéder à des opérateurs et à des méthodes.

### Exemple

4*3  // affiche le résultat
la division par 0 renvoie « Infinity »

« bonjour » ou 'bonjour' : affiche « bonjour »
« bonjour \n tout le monde » : le \n est un passage à la ligne
« bon »+  «jour » : affiche « bonjour »

« bonjour »[0] : vaut « b »
« bonjour »[3] : vaut « j »
« bonjour ».length : vaut 7

« bonjour ».toUpperCase : vaut « BONJOUR »

**Type Number**

toFixed(x) Formats a number with x numbers of digits after the decimal point
toString() Converts a number to a string

a.toFixed(2)

etc.
https://www.w3schools.com/jsref/jsref_obj_number.asp

**Type String**

substr(debut, lgr) Extrait les caractères à partir d'une position de début sur une longueur donnée
concat() Joins two or more strings, and returns a new joined strings
etc.

"hello world".substr(4, 3) // "o w"

Etc.
https://www.w3schools.com/jsref/jsref_obj_string.asp

**Type Boolean**

toString() Converts a boolean value to a string, and returns the result

(1==1).toString() // "true"
https://www.w3schools.com/jsref/jsref_obj_boolean.asp

# Variable

## Principes et usages modernes : ES6+

### Présentation

Nom, valeur, type, adresse, signification
Le nom de la variable est constitué de maj, min, chiffre, $, _ (underscore)
Le type est défini à l'usage

### Déclarer une variable : let ou const

Quand on déclare une variable, on utilise le mot clé « let », « const » (ou « var » pour les globales : à éviter).
On peut écrire : let a ;
Ou écrire let a = 5 ;

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/let

### Initialisation d'une variable

L'initialisation, c'est le moment où on donne la première valeur à une variable.
On peut le faire au moment de la déclaration : let a = 5 ;
On peut le faire après la déclaration : let a ; a=5 ;

### Constante

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/const

```
const a = 3
```

Une constante ne pourra pas être modifiée, MAIS :

On pourra modifier le contenu d'un tableau ou d'un objet déclaré en constante,
mais pas toute la variable : on modifie le contenu de la référence, pas la référence elle-même.

### Le bon usage ES6+

Le bon usage consiste à n'utiliser que des let et des const.

Le var peut se justifier pour simuler de la programmation objet : mieux vaut alors programmer des classes !

## Visibilité

Le code d'un fichier JavaScript s'exécute de la première à la dernière ligne.
Quand on définit une fonction, il ne se passe rien tant qu'elle n'est pas appelée.

Une variable déclarée avec un let ou un const est visible partout en dessous d'elle **dans le bloc où elle a été déclarée** : elle est donc locale au bloc.

Le bloc principal, c'est le fichier lui-même.

Une variable sera visible dans une fonction si on appelle la fonction après la déclaration de la variable, même si la variable a été créée après la fonction. C'est une variable globale : à éviter !

## Variable globale : à éviter !

**Variable globale : var a=3; ou b=5 directement, sans déclaration.**

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/var
**L'instruction var** permet de déclarer une variable qui sera visible partout dans le code à partir de sa déclaration : c'est une **variable globale, déclarée dans le « tas »**.
**A éviter.**

**Variable déclarée sant mot-clé : équivalent de « var » : à éviter !**

Si on initialise une variable qui n'a pas été déclarée, c'est comme si c'était une « var » : b=10 ;

**Variable locale : let a=3;**

**L'instruction let** permet de déclarer une variable qui ne sera visible **que dans le bloc** où elle est déclarée : c'est une **variable locale, déclarée dans la « pile »** (stack)..

## Utilisation d'une variable

### Affectation

```
a=5
```

### Incrémentation

```
a=a+1
a+=1
++a
a++ // post incrémentation : si on affiche en même temps, c'est la valeur de a avant l'incrémentation qui s'affiche.
a=10
console.log(a++)  // affiche 10
console.log(a)  // affiche 11
```

### Cas d'erreur

si on utilise une **variable déclarée mais sans valeur** : undefined
si on utilise une **variable non déclarée** : reference error, can't find variable

## Conversions de types

On peut changer le type d'une variable en lui donnant une nouvelle valeur.

```
a=5 ; …
a=«texte »
```

## Expression et évaluation d'une expression

### Principes d'évaluation

a=expression ;
L'expression est évaluée : elle produit une valeur qui a un certain type. Ici ce quelque chose est affectée à « a ».
console.log(expression) ;
L'expression la plus simple, c'est une valeur ou une simple variable.
Les expressions peuvent être complexes en intégrant des opérateurs, des parenthèses, des fonctions.

### Évaluation en fonction du contexte

Les expressions sont évaluées en fonction du type.
Le type est donné en fonction du contexte :

```
a=3;
b=a+2 // b vaut 5 : a est un entier
c="resultat="+a+2 // c vaut : "resultat=32" : a est une string
```

Ici c vaut : "resultat = 32"
En effet, le + est un + de concaténation : a est alors considéré comme une string

## Affichage - Saisie

### JavaScript output

http://www.w3schools.com/js/js_output.asp
Toutes les possibilités d'affichage dans la page sont présentées :

### Console.log

console.log(3)
console.log(a) : affiche a
console.log(a, b)
console.log(st1, st2)
console.log(st1+st2) : concaténation
a=5 ; b=3 ;
console.log(a + ' x ' +b + ' = ' + a*b) ;
etc.

### Affichage d'une fenêtre d'alerte, avec ou sans confirmation

alert() ou window.alert() sont équivalent : pas de confirmation

confirm() : permet d'annuler

### Affichage d'une fenêtre avec texte et champs de saisie

let a = prompt(« entrez a : )

### textContent et innerHTML

let demo = document.**getElementById("demo")**
demo.innerHTML = « coucou<br> » // la balise est prise en compte

document.**getElementById("demo").**textContent = « coucou<br> » // la balise <br> sera du texte

### Write

Write n'est pas pratique : il écrit à la fin de la page, voir ouvre une nouvelle page.
https://developer.mozilla.org/fr/docs/Web/API/Document/write

document.write(5 + 6);

- ***Affichage dans une page HTML vierge sur clic d'un bouton***

<button onclick="**document.write**(5 + 6)">Try it</button>

## Tests

### Type et expression booléenne

&&, ||, true, false
==, !=, <, <=, >, >=
https://developer.mozilla.org/fr/docs/Web/JavaScript/Les_différents_tests_d_égalité
===, !==
L'égalité faible == fait une conversion de type.
L'égalité forte === n'en fait pas.
« 17 » == 17 est true
« 17 » === 17 est false

**if, else**

```
if (condition) {

}
else {

}
```

**else if**

```
if (condition) {

}
else if (condition) {

}
else {

}
```

**switch**

```
switch (variable) {
   case valeur :
      instructions
      break
   case valeur :
      …
   default :
      instructions
}
```

## Boucles

**while**

```
while(condition){
   instructions
}

i=0
while(i<5){
   console.log("i: "+i)
   i+= 1
}
```

**for**

```
for (let i = 0; i < 5; i++) {
   console.log(i)
}
```

# Fonctions

## Principes

https://www.w3schools.com/js/js_scope.asp

### Déclaration d'une fonction

Une fonction est un regroupement d'instructions qui réalisent une tâche donnée.
Une fonction rend le code plus modulaire.
**Une fonction est constituée d'une en-tête et d'un corps.**
Une fois écrite, une fonction peut être appelée depuis n'importe quel emplacement du programme.
Une fonction peut recevoir des informations sous la forme de paramètres.
Une fonction peut renvoyer ou non une valeur de retour.

### Variables dans les fonctions

**Toutes les variables déclarées avec un « let » (ou un « const ») dans les fonctions sont locales aux fonctions** : elles ne sont pas utilisables en dehors des fonctions.

Les variables déclarées avec un var ou sans mot clé sont globales. Elles seront utilisables en dehors de la fonction : c'est à éviter !!!

### Variable locale ayant le nom d'une variable globale

Si une fonction déclare une variable locale qui existait déjà comme variable globale, la variable globale n'est plus visible dans la fonction. La variable globale est alors indépendante de la variable locale qui a le même nom.

- ***Le bon usage***

N'utiliser que des variables locales dans les fonctions : il faut donc toutes les déclarer avec un let ou un const.
Il faut éviter de déclarer des variables globales dans les fonctions.
Il faut éviter, autant que possible, d'utiliser des variables globales dans les fonctions.

## Fonctions mathématiques prédéfinies : Math

- ***Exemples***

```
console.log(Math.min(4.5, 5));   // Affiche 4.5
console.log(Math.random());      // Affiche un nombre aléatoire entre 0 et 1
```

- ***Constantes prédéfinies***

Math.PI

- ***Toutes les méthodes***

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Math
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux

http://www.w3schools.com/jsref/jsref_obj_math.asp
http://www.w3schools.com/js/js_math.asp

## Fonctions de manipulation de chaines prédéfinies : String

- ***Présentation de toutes les méthodes :***

- String

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

- ***Exemples***

http://www.w3schools.com/js/js_strings.asp

mot= « test » ;
mot.length ;
mot.toLowerCase() ; mot.toUpperCase() ;
mot.charAt(0) ; mot[0] ;
mot.indexOf(«A »)
etc.

## Fonctions de manipulation de date prédéfinies : Date

**Présentation de toutes les méthodes :**

- Date

https://www.w3schools.com/jsref/jsref_obj_date.asp
let d = new Date();
d.getDate() : retourne le jour du mois
d.getDay() : retourne le jour de la semaine : 0 pour dimanche, 1 pour lundi
getHour
setDate, setHours,
etc.

➢ ***Exemples***

http://www.w3schools.com/js/js_dates.asp
etc.

## Paramètres en sortie des fonctions : exemple 11

**Principes**

Une fonction retourne une valeur de n'importe quel type : number, string, boolean ou object (tableau, objet ( = structure) et composés)
Les paramètres de type **number**, **string**, **boolean** sont **toujours en entrée**, jamais en sortie. Ils sont passés par valeur.
Les paramètres de type **object** toujours **en entrée et en sortie**. Ils sont passés par référence.

**exemple**

La fonction ci-dessous à 2 entiers en sortie.
C'est impossible. Donc l'inversion n'est pas effective après l'appel de la fonction.

```
function inverser(a, b){
  let tmp=a;
  a=b;
  b=tmp;
}
a=5;b=10;
inverser(a,b); // ne fait rien !
console.log(a) // vaut 5
console.log(b) // vaut 10
```

La fonction ci-dessous à une objet ( = structure) à 2 entiers en paramètre : c'est un objet donc il est en entrée-sortie.

```
function inverserCouple(couple){
  tmp=couple.a;
  couple.a=couple.b;
  couple.b=tmp;
}
couple={a:5,b:10};
inverserCouple(couple);
console.log(couple.a) // vaut 10
console.log(couple.b) // vaut 5
```

## Exemple 11 – Paramètres en sortie : fonction d'inversion – A tester

Les paramètres de type **number**, **string**, **boolean** sont **toujours en entrée**, jamais en sortie. Ils sont passés par valeur.
Les paramètres de type **object** toujours **en entrée et en sortie**. Ils sont passés par référence.

- ***HTML : index.html***

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

- ***JavaScript : script.js***

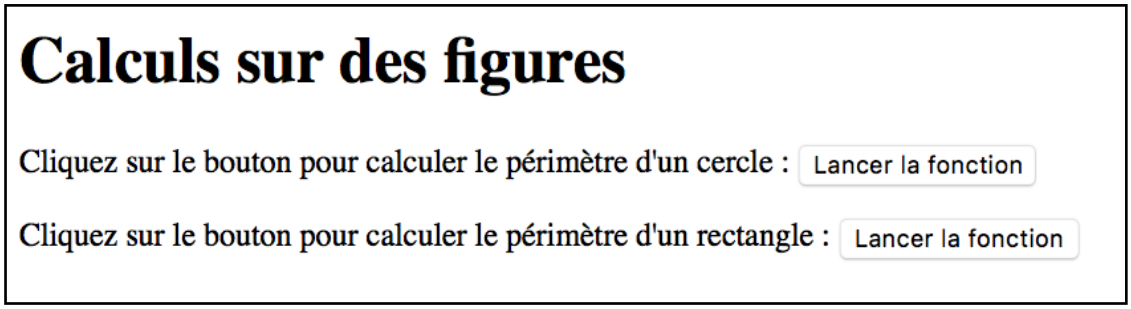
```
////////////////////////////////////////////////////////////////
////////
console.log("INVERSION DE NUMBERS");
function inverser(a, b){
  console.log("Dans la fonction inverser : entrée : a="+a+" -
b="+b);
  let tmp=a;
  a=b;
  b=tmp;
  console.log("Dans la fonction inverser : sortie : a="+a+" -
b="+b);
}

let a=5, b=10;
console.log("Avant inverser : a="+a+" - b="+b);
inverser(a,b);
console.log("Après inverser : a="+a+" - b="+b);


console.log("Avant inverser, en 'dur' : a=5, b=10");
inverser(5,10);

////////////////////////////////////////////////////////////////
////////
////////////////////////////////////////////////////////////////
////////
console.log("INVERSION DE TABLEAU");
function inverserTableauDe2(tab){
  console.log("Dans la fonction inverserTableauDe2 : entrée :
"+tab);
  let tmp=tab[0];
  tab[0]=tab[1];
  tab[1]=tmp;
  console.log("Dans la fonction inverserCouple : sortie : "+tab);
}

const tab = [2,4];
console.log("Tableau de départ : "+tab);
inverserTableauDe2(tab)
console.log("Tableau après inversion : "+tab);

////////////////////////////////////////////////////////////////
////////
////////////////////////////////////////////////////////////////
////////
console.log("INVERSION DE STRUCTURE");
```

```
function inverserCouple(couple){
  console.log("Dans la fonction inverserCouple : entrée :
a="+couple.a+" / b="+couple.b);
  let tmp=couple.a;
  couple.a=couple.b;
  couple.b=tmp;
  console.log("Dans la fonction inverserCouple : sortie :
a="+couple.a+" / b="+couple.b);
}

const couple = {
  a:2,
  b:4
}
console.log("Structure de départ : a="+couple.a+" /
b="+couple.b);
inverserCouple(couple)
console.log("Structure après inversion : a="+couple.a+" /
b="+couple.b);


////////////////////////////////////////////////////////////////
/////////
////////////////////////////////////////////////////////////////
/////////
console.log("INVERSION D'OBJET");
const objet={
  a:5,
  b:10,
  toString:function(){
    return "a="+this.a+" - b="+this.b;
  },
  inverserCouple:function(){
    console.log("Dans la fonction objet inverserCouple : entrée :
a="+this.a+" / b="+this.b);
    tmp=this.a;
    this.a=this.b;
    this.b=tmp;
    console.log("Dans la fonction objet inverserCouple : sortie :
a="+this.a+" / b="+this.b);
  }
}
console.log("Objet de départ : "+objet.toString());

inverserCouple(objet)
console.log("Objet après inverserCouple(objet):
"+objet.toString());

objet.inverserCouple();
console.log("Objet après objet.inverserCouple():
"+objet.toString());
```

## Exercices – Série 1

### 1 – Calculs sur des figures

Écrire une page qui permet de saisir la largeur et la longueur d'un rectangle puis qui affiche son périmètre et sa surface et qui qui permette de saisir le rayon d'un cercle et qui affiche le périmètre du cercle. Le résultat doit avoir 2 chiffres après la virgule.

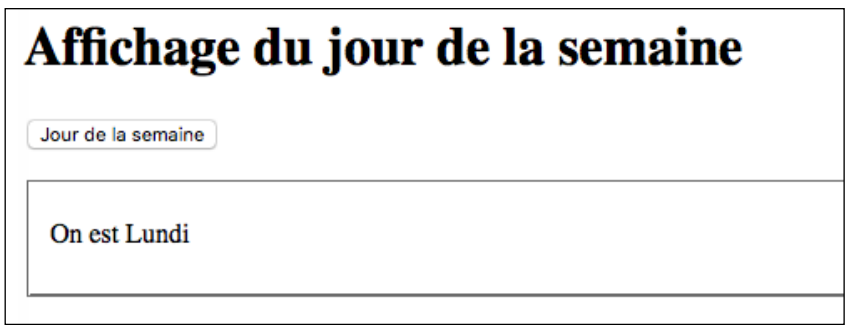



On fournit 2 boutons à l'utilisateur. Il peut saisir les valeurs. Le résultat s'affiche en dernière ligne de la page, dans une fenêtre d'alerte et dans la console de log.
Vous devez vous appuyer sur les exemples.

### 2 – Jour de la semaine

Sur le même principe que l'exercice précédent, écrire une fonction qui affiche le jour de la semaine.
Regardez ici le fonctionnement de la fonction getDay :
https://www.w3schools.com/jsref/jsref_getday.asp
Le retour d'un getDay sur une date vaut 0 pour dimanche, 1 pour lundi, etc.
Pour cela, on se dotera d'une fonction qui renvoie le jour de la semaine à partir du chiffre correspondant au résultat du getDay().

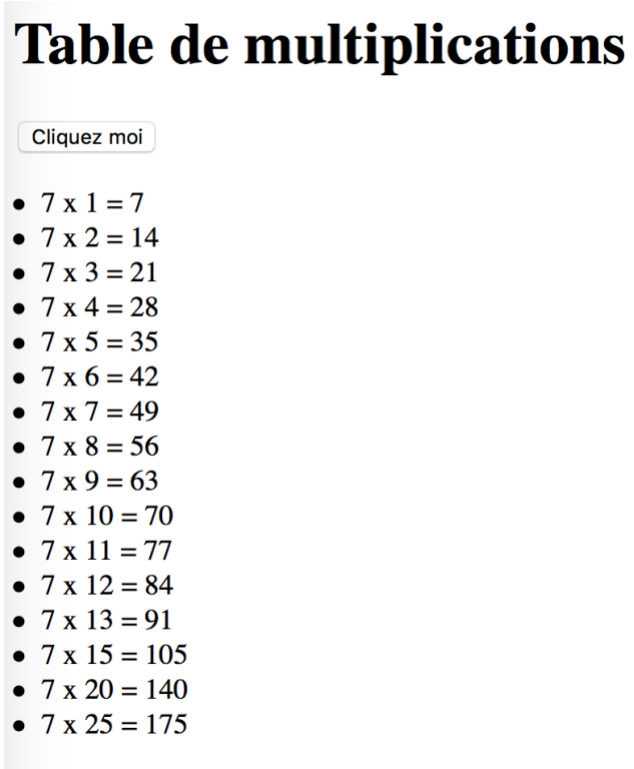
On affiche les résultats ainsi :




L'encadrement est une balise <fieldset>

## 3 – Table de multiplication

1. Ecrire une page HTML avec du JS qui permet d'obtenir le résultat suivant en cliquant sur le bouton :



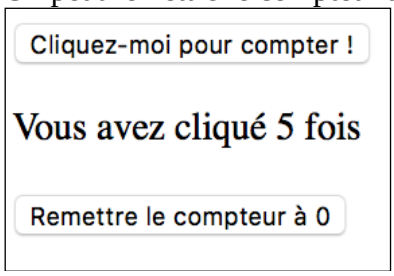

On pourra saisir la valeur 7 ou bien n'importe quelle autre valeur.
Principe de résolution :
On va construire le code HTML à produire dans une variable appelée : innerHTML
Quand elle est entièrement construite : <ul><li> etc. </ul> on écrira une instruction du type de :

```
balise.innerHTML=innerHTML;
```

## 4 : Compteur de clics

**Objectif**

Avec l'interface ci-dessous, on peut compter les clics et modifier l'affichage chaque fois qu'on clique sur « cliquez-moi  pour compter ».
On peut remettre le compteur à zéro.



# JS - 3 : TABLEAUX ET OBJETS - JSON

## Installation des fichiers de tests

Dans le cours :
Les exemples sont présentés dans un chapitre en vert.
Les exercices à faire sont présentés dans un chapitre en jaune.

Les exemples du cours sont dans un fichier zip fournis avec l'article du cours.

- JavaScript_01_exemples_02_tableaux_structures.zip

Chargez ces fichiers et mettez-les :
Soit dans un dossier « Partie_1 » que vous aurez mis dans un dossier JavaScript.
Ce dossier JavaScript peut être mis soit où vous voulez sur votre machine, soit dans le répertoire web « www » du serveur WAMP.

## Bases du code – 2 – tableaux et objets (ou structures)

### 1 - Tableaux : exemple 1

**Présentation**

http://www.w3schools.com/jsref/jsref_obj_array.asp
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array

- Les tableaux permettent de regrouper des données
- On peut mettre tout ce qu'on veut dans un tableau : number, string, booléen, objet
- On peut mélanger toutes sortes de types de données dans un même tableau.

**Déclaration**

- const tab = []; // Création d'un tableau vide
- const tab = [5, 10, 12]; // Création d'un tableau plein
- tab=new Array()
- tab=new Array(5, 10, 12) ; // Création d'un tableau plein
- tab=new Array(10) ; // Création d'un tableau de 10 cases vides

**Accès aux éléments**

- Comme les caractères d'une chaîne, les éléments d'un tableau sont identifiés par un indice débutant à zéro : tab[0], tab[1]
- tab[4] =10 ;
- On ne peut pas écrire tab[0]=5 si le tableau n'a pas préalablement été déclaré.

**length**

- tab.length : retourne la position + 1 du dernier élément : le nombre d'éléments. Les éléments d'indice négatifs ne sont pas pris en compte.

**Tous les éléments du tableau**

- tab : liste toutes les valeurs, sauf les éléments d'indice négatif.
- for(let i=0 ; i< tab.length ; i++) console.log(tab[i]) ;
- for(let key in tab) console.log(tab[key]) ;
- for(let value of tab) console.log(value) ;

**Trous dans le tableau !**

- On peut mettre l'index qu'on veut : on n'est pas obligé de tout remplir. Par exemple si le contenu de tab s'affiche ainsi : [5, 3, 5: 9, 8 :2], c'est que tab[0]=5, tab[1]=3, tab[5]=9, tab[8]=2.
- const tab=[5, 3] ; tab[5]=9 ; tab[8]=2 ; produit le tableau précédent.
- Attention : tab.length, ne donne pas le nombre d'éléments. Dans le tableau précédent, c'est 9.

**Indice négatif**

- On peut mettre des indices négatifs dans le tableau.
- On y accède avec le «  for key in » mais pas avec le « for value of ».

**Fonctions de manipulation du tableau**

https://www.w3schools.com/jsref/jsref_obj_array.asp

- tab.includes(5) // true
- tab.indexOf(5) // 0 : la position de 5 dans le tableau
- delete tab[0];  // supprime l'élément 0 qui est alors undefined.
- tab.push(valeur) : ajoute un élément après celui de l'index le plus élevé.
- elt = tab.pop() : sort du tableau l'élément ayant l'index le plus élevé. Sa valeur passe dans elt.
- tab.sort() : attention, c'est un tri alphabétique : 10 est avant 2 !

https://www.w3schools.com/js/js_array_sort.asp

**Type**

- typeof tab : retourne « object » : un tableau est un objet (au sens POO)
- Array.isArray(tab) : retourne « true » si c'est un tableau
- tab instanceof(Array) : retourne « true » si c'est un tableau

**Application**

Testez l'exemple 1

## 2 - Objet ( = structure) en JS : exemple 2

**Présentation**

- Un objet ( = structure), c'est une variable qui contient des champs (ou propriétés ou attributs) qui peuvent contenir toute sorte de valeurs.
- Les objets sont utiles pour décrire des objets du monde réel avec leurs caractéristiques.
- On accède aux champs avec l'opérateur « . »

```
let eleve = {
  id:1,
  nom: "toto",
  note: 15
};
eleve.nom; // vaut toto
eleve.note; // vaut 1
eleve.note=18; // on modifie la note
eleve.note; // vaut 18

eleve['note'] = 15 // on modifie la note : autre écriture
```

**Tous les composants d'un objets**

- eleve : liste tous les champs de l'objet ( = structure), avec les valeurs
- for( key in eleve) console.log(eleve[key]) ;

**Méthodes applicables**

- typeof eleve retourne object
- eleve.hasOwnProperty("nom") retourne true : pour savoir si une propriété appartient à un objet ( = structure).

**Application**

Testez l'exemple 2
Affichez l'objet ( = structure) dans la page HTML

## 3 - Boucles spéciales : exemples 1 et 2

**Boucle « for », rappel :**

```
for(i=0 ; i< tab.length ; i++) console.log(tab[i]) ;
```

- Attention, ça passe par tous les trous.

**boucle « for value of» : rappel**

- La **boucle « for value of »** permet de récupérer chaque valeur d'un tableau (pas d'un objet) dans une variable.

```
for(let value of tab) console.log(value) ;
```

- Attention, ça passe par tous les trous, comme une boucle for, et aussi les clés non entières.

**boucle « for key in» : rappel**

- La **boucle « for key in »** permet de récupérer chaque élément d'un objet ( = structure) ou d'un tableau dans une variable.

```
for(let key in tab) console.log(tab[key]) ;
```

- Ca ne retourne que les key existantes, négatives comprises.
- Exemple avec une structure :

```
const person = {fname:"John", lname:"Doe", age:25};

let text = "";
let x;
for (x in person) {
    text += person[x] + " ";
}

// text vaut : "John Doe 25"
```

https://www.w3schools.com/jsref/jsref_forin.asp

**méthode forEach**

- La **méthode forEach()** permet d'exécuter une fonction sur chaque élément du tableau. On code la fonction directement dans le forEach :

```
const tab = ['a', 'b', 'c'];

tab.forEach(function(element) {
  console.log(element);
});
```

- On peut aussi passer une fonction en paramètre et définir la fonction ailleurs.

```
const tab = ['a', 'b', 'c'];

tab.forEach(maFonction(element));

function maFonction(element){
  console.log(element);
}
```

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/forEach

**methode map**

- La **méthode map()** retourne un tableau avec les valeurs retournées par le fonction.

```
const tab = [1, 2, 3, 4]
const tabCarre = tab.map(function (elt) {
  return elt*elt
})
console.log(tabCarre) // [ 1, 4, 9, 16 ]
```

- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/map

```
const personnes=[{id: 1, nom:'toto'}, {id: 2, nom: 'titi'}]
tabNoms = personnes.map(function (personne) {
  return personne.nom
})
console.log(tabNoms)  // [ 'toto', 'titi' ]
```

➢ ***map et index***

```
const tab = [1, 2, 3, 4]
tab.map(function (elt, index) {
  console.log('tabCarre['+index+']='+elt+' -> carré='+elt*elt)
})
```

➢ ***max et fonction fléchée***

```
const tabCarre = tab.map(function (elt) {
  return elt*elt
})

devient :
const tabCarre = tab.map(elt => {
  return elt*elt
})

devient :
const tabCarre = tab.map(elt => elt*elt)
```

**méthode filter**

- La **méthode filter()** retourne un tableau avec uniquement certaines valeurs du tableau de départ.

```
const entiers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

const pairs = entiers.filter(entier => entier%2 == 0);

console.log(pairs);  // [2, 4, 6, 8, 10];
```

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

## 4 – Tableau de d'objets ( = structures) : exemple 3

### Présentation

On peut mettre des objets dans des tableaux :

```
const eleve1 = {
    id:1,
    nom: "toto",
    note: 15
};
const eleve2 = {
    id:2,
    nom: "tata",
    note: 16
};
const tabEleves = [eleve1, eleve2];
```

### Accès aux données

tabEleves[1].nom vaut « tata ».

### Tri d'un tableau d'objets (syntaxe ES6 : fléchée)

tabEleves.sort( (a,b) => a.nom -b.nom)

## 5 - JSON

### Format JSON : JavaScript Object Notation

On peut aussi déclarer directement un tableau d'objets ( = structures) :

```
const tabEleves = [
  {
    id:1,
    nom: "toto",
    note: 15
  },
  {
    id:2,
    nom: "tata",
    note: 16
  }
]
```

Ou bien, compacté :

```
const tabEleves = [
  { "id":1, "nom": "toto", "note": 15 },
  { "id":2, "nom": "tata", "note": 16 }
]
```

Ou encore plus compacté :

```
const tabEleves = [ { "id":1, "nom": "toto", "note": 15 }, {
"id":2, "nom": "tata", "note": 16 } ]
```

Le format [ ] et {} avec les attributs entre guillemets, c'est du JSON.

### Affichage d'un fichier JSON dans un navigateur

- On peut afficher un fichier JSON dans un navigateur comme n'importe quel fichier texte : en mode CLI.
- On peut aussi avoir un affichage formatté en mode GUI.

- Firefox propose un affichage formatté par défaut.
- Dans Chrome ou Brave, il faut installer une extension : par exemple : JSON formatter.

**Lecture d'un format JSON dans Firefox (ou avec une extension formater JSON)**

Le fichier eleve.json contient :
[{"id":1 ,"nom":"toto", "note": 15}, {"id":2, "nom":"tata", "note": 16}]

Quand on ouvre un fichier directement dans Firefox, on obtient :


**Exemples**

Prenez le fichier JSON de l'exemple 3 et affichez le dans Firefox

```
const eleve1 = {
    id:1,
    nom: "toto",
    note: 15
};
const eleve2 = {
    id:2,
    nom: "tata",
    note: 16
};
const tabEleves = [eleve1, eleve2];
```

## 6 – Objet (= structure) avec fonctions : exemple 4

### Présentation

On peut déclarer une ou plusieurs fonctions en même temps qu'un objet (= structure).
C'est une première approche de la programmation objet.

### Exemple

```
couple={
  a:5,
  b:10,
  toString:function(){return "a="+this.a+" - b="+this.b;}
}
```

### Usages

```
console.log(couple.a) vaut 5
couple.a = 3
console.log(couple.toString) vaut "a=3 - b=10"
```

### Exemple

exemple 4 – Tableau d'objets (= structures)
Affichez l'objet dans la page HTML

## Exercices – Série 2

### 0 – JSON

**Installation de l'extension JSON Formatter dans Chrome**

- Installez JSON Formatter dans Chrome.
- Paramétrez JSON Formatter dans Chrome : gérer extensions / détails / Autoriser l'accès aux URL de fichier.

- tests de fichiers :
  - bliaudet.free.fr/IMG/json/eleves.json : ici
  - bliaudet.free.fr/IMG/json/films.json : ici
  - bliaudet.free.fr/IMG/json/les_films.json : ici
  - bliaudet.free.fr/IMG/json/voitures.json : ici
  - bliaudet.free.fr/IMG/json/frequentation-dans-les-salles-de-cinema.json : ici

**Exercice 1 :**

- Chargez ces fichiers directement sur votre ordi à partir des liens. Regardez le résultat : cherchez à comprendre les données.

**Exercice 2 :**

- Enregistrez le fichier « les_films.json » dans un fichier. Lisez ce fichier dans votre navigateur.
- Dans une page HTML, afficher ce fichier en format <table>
- Ajoutez 3 boutons qui permettent de trier : par année, par titre et par id.
- Ajoutez un bouton qui permette de sélectionner les films uniquement après une année donnée.

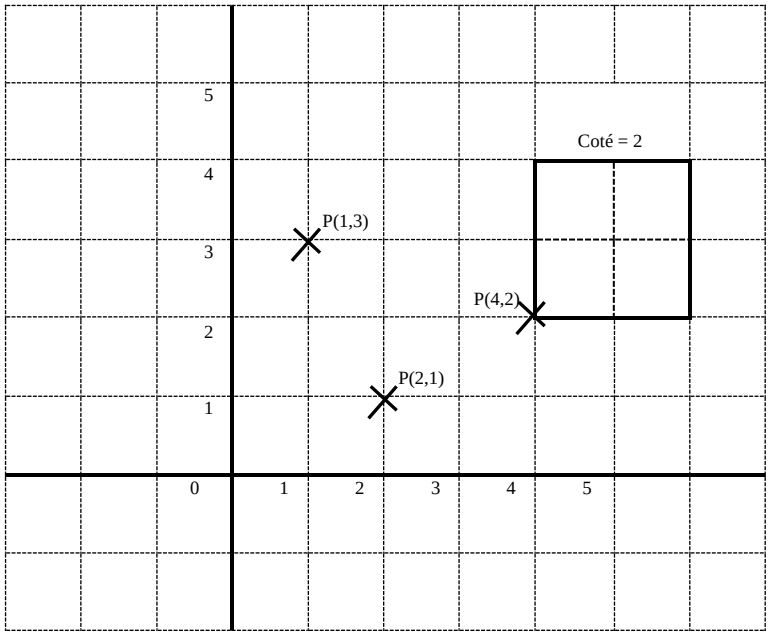
**Exercice 3 :**

- Enregistrez le dernier exemple (fréquentation) dans un fichier. Lisez ce fichier dans votre navigateur.
- Écrivez un code qui affiche dans une page HTML les recettes pour chaque année entre 2000 et 2009, dans l'ordre des années. D'abord vous affichez une liste textuelle de valeurs. Puis vous affichez les résultats dans un tableau HTML.
- Pour faire ça, vous pouvez commencer par afficher les résultats pour toutes les valeurs.

## Exercice 4

Un point est caractérisé par ses coordonnées x et y. Un carré à base horizontale est défini par les coordonnées de son point en bas à gauche et par son côté. Les coordonnées et le côté sont des réels.
On travaille sur un ensemble de carrés.

- Définir les objets qui permettent de gérer ce problème. Créer le carré du schéma ci-dessous.
- Créez un fichier JSON qui regroupe les objets du schéma ci-dessous. Vous rajouterez 2 carrés aux choix.




## Exercice 5

Un élève est caractérisé par son nom, son prénom, sa date de naissance. Il y a 3 matières d'informatique : algo, C++ et SQL. Chaque matière donne lieu à 2 QCM et à 1 examen. Les QCM comptent pour 25%. L'examen pour 50%. On connaît les dates d'examen et de QCM. Chaque élève porte toutes les informations le concernant. On connaît les notes pour chaque examen, la note finale pour la matière et la moyenne des 3 matières.

- Définir la ou les objets JSON permettant de gérer un élève.
- Définir l'objet JSON permettant de gérer une classe. On enregistre aussi la moyenne générale de la classe dans la structure de la classe.

## 1 – Tableau de notes

Définir en JavaScript un tableau contenant des notes d'étudiants comprises entre 0 et 20.

- Implémenter en JavaScript les fonctions qui permettent de :
  - ✓ afficher un tel tableau de façon standard HTML
  - ✓ savoir combien d'étudiants ont eu plus de 10
  - ✓ calculer la moyenne des notes
  - ✓ connaître la note maximale
  - ✓ rechercher une note particulière : on donne la note, on récupère l'indice.
  - ✓ trier le tableau
- Remarque de méthode :
  - ✓ Vous pouvez vous aider de : https://www.w3schools.com/jsref/jsref_obj_array.asp. Pour par exemple : indexOf ou la fonction de tri.
  - ✓ dans les fonctions, on ne fera aucun affichage.
  - ✓ On utilise un fichier HTML et un fichier JavaScript.
  - ✓ On affiche les résultats dans une div prévue pour dans le HTML et aussi en fin de fichier avec un document.write pour afficher dans la page HTML
  - ✓ Dans le fichier JavaScript, à la fin, on a d'abord les fonctions, puis la création du tableau, puis les appels aux document.write. Le fichier JS est comme un contrôleur qui include le modèle (les fonctions), puis « met la colle du contrôleur » (définit le tableau), puis include la vue (les document.write).

- ***Objectifs à atteindre en termes de présentation (il manque quelques fonctionnalités)***

# Tableau

## Dans la div

Tableau de départ :

| 8 | 12 | 9 | 12 | 17 | 18 | 15 | 13 |
|---|---|---|---|---|---|---|---|

Nombre de notes supérieures à 10 : 6

Moyenne des notes : 11.555555555555555

Tableau trié :

| 8 | 9 | 12 | 12 | 13 | 15 | 17 | 18 |
|---|---|---|---|---|---|---|---|

## Ajout avec des document.write en dessous

Tableau de départ :

| 8 | 9 | 12 | 12 | 13 | 15 | 17 | 18 |
|---|---|---|---|---|---|---|---|

Nombre de notes supérieures à 10 : 6

Moyenne des notes : 11.555555555555555

Tableau trié :

| 8 | 9 | 12 | 12 | 13 | 15 | 17 | 18 |
|---|---|---|---|---|---|---|---|

## 2 - Tableau d'élèves avec des notes – Tri d'un objet (= structure)

Dupliquer le travail de l'exercice précédent.
Dans l'exercice précédent, ajouter un prénom pour chaque note. On utilise forcément une fonction.

- Mettez à jour toutes les fonctions.
- Ajoutez une fonction de tri par nom et une fonction de tri par note (googler trier un tableau json). Afficher le tableau trié par nom puis trié par note.

➢ ***Objectifs à atteindre en termes de présentation :***

# Tableau de structures

Tableau de départ :

| Prénom | Note |
|---|---|
| tata | 8 |
| tete | 12 |
| titi | 9 |
| toto | 12 |
| tutu | 17 |
| tate | 18 |
| tati | 15 |
| tato | 13 |

plus de 10 : 6

moyenne : 13.00

max : 18

9 est présent dans le tableau en position 3

Tableau trié par notes :

| Prénom | Note |
|---|---|
| tata | 8 |
| titi | 9 |
| toto | 12 |

etc.

## 3 - Tableau d'élèves avec des notes – creerEleve

Dupliquer le travail de l'exercice précédent

- Créer une fonction qui permet de créer un élève : creerEleve(nom, note, photo)
- Remplir désormais le tableau complet en utilisant cette fonction.
- Créer une fonction qui permet d'afficher un élève de telle sorte qu'elle puisse être utilisée dans la fonction qui affiche tout le tableau (afficher veut dire : retourner un texte avec le code HTML). Mettez à jour la fonction.
- Afficher un élève au choix après les tris.
- Le but est de gérer chaque fonctionnalité avec un bouton.

➢ ***Objectifs à atteindre en terme de présentation :***

# Tableau de structures avec images

Retour au départ | Retour au départ - JSON | Tri par nom | Tri par notes | Infos | Est présent ?

| Prénom | Note | Photo |
|---|---|---|
| tati | 8 |  |
| titi | 9 |  |
| TaTa | 11 |  |

## 4 – Jeu de grammaire

**Etape 1**

- Mettez votre travail dans un dossier appelé : JS_bases_exo10_1
- Définir une variable contenant le verbe chanter.
- Créer une variable qui contient le « nom d'agent » du verbe chanter. Le nom d'agent, c'est celui qui fait l'action. Pour chanter, il s'agit de « chanteur » : le chanteur chante. Pour créer cette variable, supprimer les deux dernières lettres de ce mot et les remplacer par les lettres eur.
- Produire le texte suivant : « chanter : le chanteur chante »

**Etape 2**

Dupliquer le travail de l'exercice précédent dans un dossier appelé : JS_bases_exo10_2

- Généraliser le code précédent en écrivant une fonction traitant n'importe quel verbe du premier groupe.
- Créer un tableau avec des verbes du premier groupe.
- Appliquer la fonction à tous les verbes du tableau.

**Etape 3**

Dupliquer le travail de l'exercice précédent dans un dossier appelé : JS_bases_exo10_3

- Mettez des verbes du premier et du deuxième groupe dans le tableau (deuxième groupe : verbe finissant en « ir » à l'infinitif et en « issons » à la première personne du pluriel au présent.
- Définir un tableau à six cases contenant les pronoms personnels.
- Définir un tableau contenant les marques de conjugaison au présent de l'indicatif pour les verbes du premier groupe, et un autre pour les terminaisons des verbes du deuxième groupe.
- Écrire une fonction qui décide si le verbe appartient au premier ou au deuxième groupe.
- Mettre un verbe du tableau dans une variable appelée : « verbe »
- Afficher les formes conjuguées de « verbe » au présent de l'indicatif, chaque forme étant précédée du ou des pronom(s) personnel(s) adéquat(s).
- Produire une fonction qui affichage les formes conjuguées de n'importe quel verbe au présent de l'indicatif, chaque forme étant précédée du ou des pronom(s) personnel(s) adéquat(s).
  - ✓ On affichera au début : le nom du verbe et son groupe.
  - ✓ Si ce n'est pas un verbe du premier ou du deuxième groupe, on n'affiche pas la conjugaison.
  - ✓ Si c'est un verbe du premier groupe, on affiche, par exemple : le mangeur mange.
  - ✓ Pour les verbes du premier et du deuxième groupe, on affiche la conjugaison.

## 5 – Pipotron, Poétron

Pipotron : http://www.pipotron.free.fr
Poétron : http://www.vudansvotreemail.com/poetron-sourire-doc-htm.htm?pos=4&

Dupliquer le travail du jeu de grammaire dans un dossier appelé : JS_bases_exo10

- Créer un tableau de noms avec un article. Par exemple : le chat, les feuilles, l'arbre, un animal, Bertrand, etc.
- Avec le tableau de verbe, fabriquez une phrase aléatoirement en choisissant un nom pour le sujet, un verbe et un nom pour le complément.
- Attention à la conjugaison !

# JS - 4 : JS MODERNE : ES6/2015

## JS moderne : ES6/2015 - Standard ECMAScript

- https://apprendre-a-coder.com/es6/
- ES6 = ES2015 = ES6/2015 : une révolution pour JavaScript.
- **ES6/2015 : sucre syntaxique pour les Classes**. JavaScript n'a pas de classes. La fonctionnalité des classes est reprise par les prototypes d'objet et le « sucre syntaxique pour les Classes » apparu avec ES6.
- Pour les prototypes, voir le chapitre précédent.
- Pour une introduction à l'objet, voir : http://bliaudet.free.fr/article.php3?id_article=108 : on trouve un pdf et des exemples JavaScript ES6, Python et Java.

## L'essentiel de ES6/2015

- Il y a essentiellement 4 points dans ES6/2015 :
- https://gist.github.com/gaearon/683e676101005de0add59e8bb345340c
- "**let**" et "**const**" : mieux contrôler ses variables. Bon usage : on définit les variables avec des let et des const.
- **Classe** : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes
- **arrow function** : fonction fléchée. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
- **destructuration**

## let

### Principes

- let : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/let

### Portée

- let permet de déclarer une variable dont la portée est celle du bloc courant

### Utilité

- Avoir des variables locales explicites.

### Usage

- On va déclarer toutes les variables en let : une variable est locale par défaut. On évite les variables globales.

### Variables globales : rappels

- D'une façon générale, il vaut mieux éviter les globales.
- Les variables globales de niveau fichier (ou module) peuvent être déclarées avec un var ou sans mot clé.
- Les variables globales qui apparaissent dans les fonctions sont déclarées sans mot clé. Elles peuvent être utile pour conserver un état pour un retour dans la fonction à condition de ne pas les utiliser en dehors de la fonction.

## const

### Principes

- const : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/const
- Pour un non objet : on ne peut plus le modifier. C'est une constante classique.
- Pour un objet (donc les tableaux), on ne peut pas le redéfinir, mais on peut modifier son contenu ;
- const obj={ … }
- const tab=[]

### Portée

- const permet de déclarer une variable dont la portée est celle du bloc courant : une constante déclarée dans une fonction n'est visible que dans la fonction.

### Utilité

- Quand un objet est déclaré en constante, on ne peut plus le modifier => on ne peut plus le transformer en entier, le passer à null, etc => c'est une protection syntaxique

### Usage

- On va déclarer tous les objets en constante !

## POO ES6

### Principes

ES6 permet de faire de la programmation objet « standard » alors que JavaScript permettait de faire de la POO spécifique.
Aujourd'hui (en 2023), il faut s'intéresser uniquement à la façon « standard » de faire de la POO en ES6.

```
class MaClasse extends MonParent {
  constructor (p1, p2) { // le constructeur
    super(p1) // par exemple avec p1 en paramètre
    this.p1 = …
    this.p2 = …
    this.p3 = 0
    …
  }

  methode () {  // les méthodes qu'on veut
    …
  }
}
```

- L'appel au constructeur parent « super(param) » est obligatoire quand il y a héritage « extends ».
- Cet appel doit être fait avant tout usage du « this » qui est obligatoire.
- Sinon, on aura des SyntaxError ou des ReferenceError.
- Les méthodes s'écrivent directement « maMethode(paramètres) { … } » et non plus « maMethode : function(paramètres) { … }

**Héritage**

```
class MaClasse extends MonParent {
  constructor (p1, p2) { // le constructeur
    super(p1) // par exemple avec p1 en paramètre
    this.p1 = …
    this.p2 = …
    this.p3 = 0
    …
  }

  methode () {  // les méthodes qu'on veut
    …
  }
}
```

- L'appel au constructeur parent « super(param) » est obligatoire quand il y a héritage « extends ».
- Cet appel doit être fait avant tout usage du « this » qui est obligatoire.
- Sinon, on aura des SyntaxError ou des ReferenceError.

**Exemple de classe**

## Déclaration de la classe

```
class Personne {
    // le constructeur et ses attributs
    constructor(prenom, nom, salaire) {
        this.prenom = prenom
        this.nom = nom
        this.salaire = salaire
    }
    // les méthodes
    toString() {
        return this.prenom + ' ' + this.nom.toUpperCase() + ' :
'+this.salaire;
    }
    setSalaire(salaire) {
        this.salaire = (salaire)
    }
}
```

## Usage

```
let personne = new  Personne('Toto', 'Leprogrammeur', 3000)
console.log(personne.toString())
personne.setSalaire(3500)
console.log(personne.toString())
```

## Inclusion dans un fichier HTML

```
<!DOCTYPE html>
<html lang="fr">
     <head>
         <meta charset="utf-8">
         <title>Classe ES6</title>
     </head>

     <body>
          <h1>Test POO : regardez en console</h1>
          <script src="Personne.js"></script>
          <script src="main.js"></script>
     </body>
</html>
```

## Fonction fléchée : fat arrow : =>

https://openclassrooms.com/fr/courses/4664381-realisez-une-application-web-avec-react-js/4664806-modernisez-votre-javascript-avec-es2015#/id/r-4718486

**Exemple -> ici**

On se donne :

```
const people = []
people[0] = { name: "adulte", age: 30 }
people[1] = { name: "enfant", age: 10 }
```

Ancienne version :

```
console.log('ancienne version')
const adults = [], minors = []
people.forEach(function(person) {
  if (person.age >= 18) {
    adults.push(person)
  } else {
    minors.push(person)
  }
})
console.log(adults)
console.log(minors)
```

Version ES6 :

```
console.log('nouvelle version avec =>')
const adults2 = [], minors2 = []
people.forEach((person) => {
  if (person.age >= 18) {
    adults2.push(person)
  } else {
    minors2.push(person)
  }
})
console.log(adults2)
console.log(minors2)
```

## Fonction fléchée : fat arrow : =>

https://openclassrooms.com/fr/courses/4664381-realisez-une-application-web-avec-react-js/4664806-modernisez-votre-javascript-avec-es2015#/id/r-4718486

**Exemple -> ici**

On se donne :

```
const people = []
people[0] = { name: "adulte", age: 30 }
people[1] = { name: "enfant", age: 10 }
```

Ancienne version :

```
console.log('ancienne version')
const tab1 = people.map(function (person) {
 return person.name
})
console.log(tab1)
```

Version ES6 :

```
console.log('nouvelle version avec =>')
const tab2 = people.map(person => person.age)
console.log(tab2)
```

https://openclassrooms.com/fr/courses/4664381-realisez-une-application-web-avec-react-js/4664806-modernisez-votre-javascript-avec-es2015#/id/r-4718486

- map : la fonction map() retourne un tableau avec les valeurs passées en paramètres.
- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/map

## this et =>

https://openclassrooms.com/fr/courses/4664381-realisez-une-application-web-avec-react-js/4664806-modernisez-votre-javascript-avec-es2015#/id/r-4718486

- En JS classique, le this dans une fonction fait référence à l'objet qui appelle la fonction.
- Dans une fonction anonyme classique, le this fait référence à l'objet global (la fenêtre).
- Dans les fonctions fléchées, le this fait référence à l'objet le plus proche.

**Exemple -> ici**

```
const name = 'name extérieur'
const test = {
    name: 'name intérieur',
    testerThis() {
        let tab = [11 , 22]
        console.log("testerThis : this et name")
        console.log(this)      // this c'est l'objet test
        console.log(this.name)
        console.log(name)

        console.log('ancienne version')
        tab.forEach(function (elt) { // 1 seul élément dans tab
            console.log(this)        // this c'est la fenêtre
            console.log(elt + ':' + this.name) // name n'existe pas
        })

        console.log('nouvelle version avec =>')
        tab.forEach((elt) => {       // 1 seul élément dans tab
            console.log(this)        // this c'est l'objet test
            console.log(elt + ':' + this.name)
        })
    }
}

test.testerThis()
```

## Le mode strict

- **mode laxiste JS, mode strict JS**
- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Strict_mode
- Le mode strict rend explicite certaines erreurs silencieuses. Il améliore ainsi la qualité et la maintenabilité du code.
- Dans l'exemple précédent, le cas du mileu :

```
tab.forEach(function(elt) {          // 1 seul élément dans tab
  console.log("TAB : forEach")
  console.log(this)                  // this c'est la fenêtre
  console.log(elt+':'+this.name)     // name n'existe pas
})
```

- affiche

```
TAB : forEach
Window
1:
```

- il n'y a pas de this.name : c'est une erreur silencieuse.
- Si on ajoute au début du code :

```
'use strict';
```

- On obtiendra une erreur :

```
TypeError: this is undefined
```

## Destructuration

https://openclassrooms.com/fr/courses/4664381-realisez-une-application-web-avec-react-js/4664806-modernisez-votre-javascript-avec-es2015#/id/r-4718497

- La déstructuration nous permet de récupérer plus facilement plusieurs informations au sein d'un objet ou d'un tableau.

**Exemples -> ici**

### Sur un objet

On se donne :

```
obj = {
    firstName: 'Bertrand',
    lastName: 'Liaudet'
}
```

Ancienne version :

```
console.log('ancienne version')
console.log(obj)
const firstName1 = obj.firstName
const lastName1 = obj.lastName
console.log(firstName1, lastName1)
```

Syntaxe ES6 :

```
console.log('nouvelle version avec destructuration')
console.log(obj)
// destructuration : on sort chaque attribut
// on reprend forcément le nom de l'attribut
const {firstName, lastName} = obj
console.log(firstName, lastName)
```

**Sur un tableau**

On se donne :

```
const fullName = "Bertrand Liaudet"
```

Ancienne version :

```
console.log('ancienne version')
const names = fullName.split(' ')
console.log(names) // ["Bertrand", "Liaudet"]
const firstName1 = names[0]
const lastName1 = names[1]
console.log(firstName1, lastName1)
```

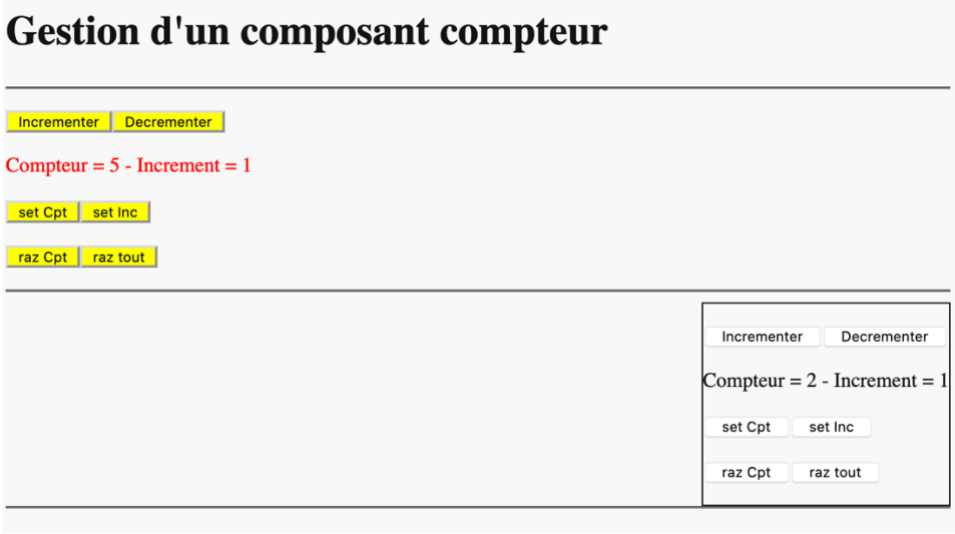
Syntaxe ES6 :

```
console.log('nouvelle version avec destructuration')
const [firstName, lastName] = fullName.split(' ')
console.log(firstName, lastName)
```

## Exemples : un composant compteur

### TP2-1 : création d'un composant compteur

Le but est de créer un composant qui permette d'afficher ces deux résultats.
On utilise la programmation objet.




### Présentation de 4 versions d'un petit compteur :

- Version sans variable globale
- Version avec variable globale
- Version avec Classe-function et prototype
- Version avec Classe

On peut ne s'intéresser qu'à la version avec classe.
Les versions avec classes permettent de construire un composant HTML avec son HTML, son CSS et son JavaScript.
Les versions composant distinguent 2 fichiers JavaScript : le « main » et la classe « Compteur ».

### TP2-1 : mise à jour du composant compteur

- Dans la version avec classe, ajoutez des fonctionnalités pour arriver aux résultats demandés pour le TP2-1.

# JS - 5 : ANCIENNES VERSIONS DE POO

## Anciennes versions de POO - facultatif

JavaScript permet de faire de la programmation objet.
Il y a plusieurs façons de faire de la POO en JavaScript classique.
https://developer.mozilla.org/fr/docs/Web/JavaScript/Introduction_à_JavaScript_orienté_objet

Dans le cours :
Les exemples sont présentés dans un chapitre en vert.
Les exercices à faire sont présentés dans un chapitre en jaune.

Les exemples du cours sont dans un fichier zip fournis avec l'article du cours.

- JavaScript_01_exemples_03_objets.zip

Chargez ces fichiers et mettez-les :
Soit dans un dossier « Partie_1 » que vous aurez mis dans un dossier JavaScript.
Ce dossier JavaScript peut être mis soit où vous voulez sur votre machine, soit dans le répertoire web « www » du serveur WAMP.

## Bases : objet (=structure) avec méthode – exemple 1

### Objet (=structure) avec méthode

Un objet ressemble à :

```
let eleve = {
  id:1,
  nom: "toto",
  note: 15
};
```

Pour un attribut, à la place d'une valeur on peut mettre une méthode :

```
let eleve = {
  id:1,
  nom: "toto",
  note: 15
  afficher : function(){
    console.log("nom de l\'élève : ", this.nom)
    console.log("note de l\'élève : ", this.note)
  }
};
```

A partir de là, l'objet (=structure) ressemble à une classe de la programmation objet.
On va se doter d'une variable de type structure qui servira de prototype pour créer d'autres objets grâce à une fonction spéciale de JS.

**Utilisation**

```
console.log("Affichage de l\'élève : ")
eleve.afficher();

eleve.note=18;
console.log("Affichage de l\'élève après modification de la note
: ")
eleve.afficher();
```

**Limitation**

Avec cette technique, on doit redéfinir les attributs à chaque fois qu'on crée une nouvelle variable de structure identique

## POO - Version 1

```
function Personne(prenom, nom) {
  this.prenom = prenom;
  this.nom = nom;
  this.nomComplet = function() {
    return this.prenom + ' ' + this.nom;
  }
  this.nomCompletInverse = function() {
    return this.nom + ' ' + this.prenom;
  }
}
Const s = new Personne("Simon", "Willison");
console.log(s.nomComplet())
console.log(s.nomCompletInverse())
```

- Défaut : chaque objet porte les fonctions. Elles ne sont pas partagées.

## POO - Version 2 : avec attribut prototype

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

```
function Personne(prenom, nom) {
  this.prenom = prenom;
  this.nom = nom;
}
Personne.prototype.nomComplet = function() {
  return this.prenom + ' ' + this.nom;
}
Personne.prototype.nomCompletInverse = function nomCompletInverse()
{
  return this.nom + ' ' + this.prenom;
}
const s = new Personne("Simon", "Willison");
console.log(s.nomComplet())          //  Simon Willison
console.log(s.nomCompletInverse())   //  Willison Simon
```

- Le prototype permet d'ajouter la fonction et qu'elle soit partagée.

### ajout d'une fonction avec prototype en cours de code

```
Personne.prototype.nomEnMajuscules = function() {
  return this.nom.toUpperCase()
}
console.log(s.nomEnMajuscules());  //  WILLISON
```

### ajout d'une fonction avec prototype sur des classes natives

```
let s = "Simon"; // s est une String
String.prototype.inverse = function() {
  let r = "";
  for (let i = this.length - 1; i >= 0; i--) {
    r += this[i];
  }
  return r;
}
console.log(s.inverse()); // "nomiS"
```

- On pourrait aussi redéfinir la méthode « toString » qui est déjà présente sur les chaines de caractères.

**Array.prototype – méthodes filter, every, some**

- Array utilise la technique du prototype : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/prototype
- Associé au prototype on trouve de nombreuses méthodes qu'on peut redéfinir, comme par exemple filter(), every() et some()
- **filter() :** https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/filter
- **every() :** https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/every
- **some() :** https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/some
-

## POO – Version 3 : avec un objet prototype et Object.create() – exemple 2

### Création de Classe : notion de prototype – technique 1

**Classe : création d'une variable de type structure qui sert de prototype**

Le prototype est l'équivalent de la classe.
Toutefois, c'est en réalité une variable de type structure : c'est donc un type « object »
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Object
Cette variable est appelée « prototype » : elle servira de référence pour la création des objets.

```
const Personnage = { // variable qui sert de Prototype
    nom: "", // valeur par défaut
    sante: 0, // valeur par défaut
    force: 0, // valeur par défaut
    xp: 0, // expérience, valeur par défaut

    // On peut mettre des fonction
    decrire: function () { // retourne la description à afficher
        let description = this.nom + " a " + this.sante +
              "points de vie, " + this.force +
              " en force et " + this.xp + " points d'expérience";
        return description;
    }
};
```

A noter qu'on écrit la variable avec une majuscule : Personnage. C'est pour dire que c'est le prototype, l'équivalent de la Classe.

**Objet : création d'une variable à partir d'un prototype : fonction static Object.create**

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Object/create
On utilise la fonction static Object.create pour créer un nouvel objet à partir du prototype.
Ensuite, on peut donner des valeurs à chaque attribut.

```
const perso1 = Object.create(Personnage);
perso1.nom = "Aurora";
perso1.sante = 150;
perso1.force = 25;
```

```
const perso2 = Object.create(Personnage);
perso2.nom = "Glacius";
perso2.sante = 130;
perso2.force = 35;
```

## Création de Classe : fonction d'initialisation – technique 2

### Ajout de méthodes dans le prototype

On définit une fonction d'initialisation dans la structure. Cette fonction va initialiser les attributs de la structure.
De ce fait, il n'est plus nécessaire de les déclarer dans le prototype.
On peut aussi ajouter d'autres fonctions.
Notez le mot clé « this ».

```
const Personnage = {
    init: function (nom, sante, force) {
        this.nom = nom;
        this.sante = sante;
        this.force = force;
        this.xp = 0;
    },

    decrire: function () { // retourne la description à afficher
        let description = this.nom + " a " + this.sante +
              "points de vie, " + this.force +
              " en force et " + this.xp + " points d'expérience";
        return description;
    }
};

const perso1 = Object.create(Personnage);
perso1.init("Aurora", 150, 25);
console.log(perso1.decrire());
```

## Utilisation du new

On peut aussi créer des objets avec un new. Mais il vaut mieux éviter.
https://openclassrooms.com/courses/dynamisez-vos-sites-web-avec-javascript/les-objets-5

## POO – Version 3 : Héritage - exemple 2

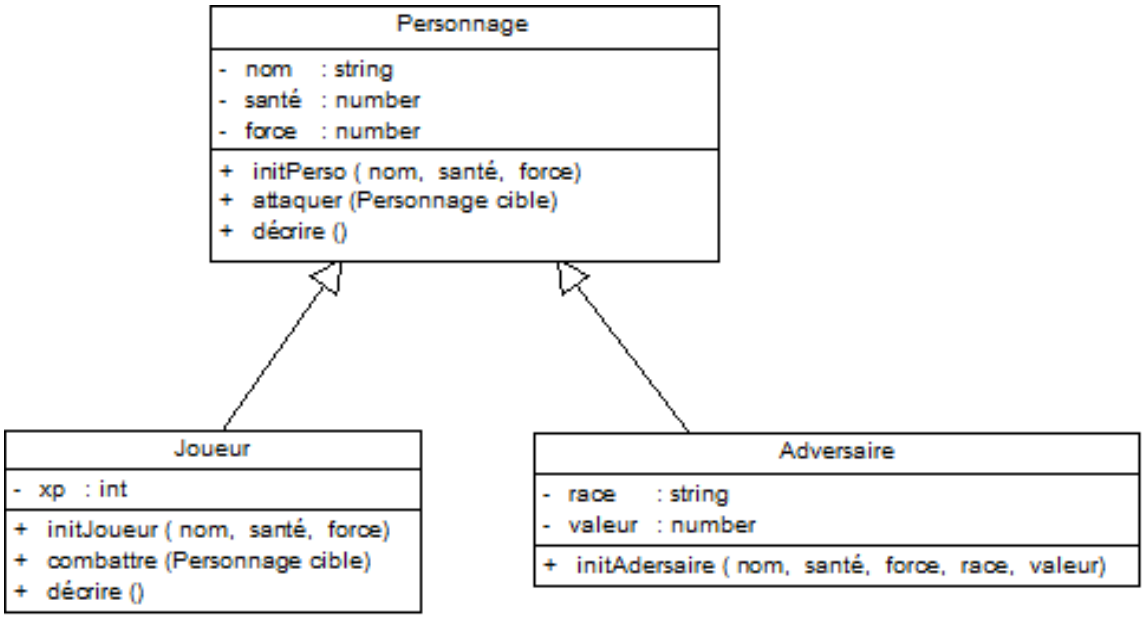
### Principes

On peut simuler l'héritage avec les prototypes.
Une classe dérivée sera créée à partir d'un prototype. Ensuite, on ajoute à la variable créée des attributs et des méthodes. Elle devient un prototype pour des objets.

### Application

Joueurs et Adversaires sont des Personnages. Seuls les joueurs ont de l'expérience. Les adversaires ont une race et une valeur.

Le modèle des classes est le suivant :




Bien noter que le modèle est très utile pour comprendre l'organisation des classes. Sans lui, on se perd rapidement dans le code.

**Code**

A partir du Personnage, on crée un Joueur et on met à jour la fonction d'initialisation.
Idem pour l'adversaire.
A noter la syntaxe de la création de la fonction initJoueur

```
// On crée le prototype Personnage
const Personnage = {
    initPerso: function (nom, sante, force) {
        this.nom = nom;
        this.sante = sante;
        this.force = force;
    }
};

//on crée le prototype Joueur
const Joueur = Object.create(Personnage); // Prototype du Joueur

// on ajoute la function initJoueur au prototype
Joueur.initJoueur = function (nom, sante, force) {
    this.initPerso(nom, sante, force); // appelle initPerso
    this.xp = 0;                       // init la partie joueur
};

// on ajoute la function décrire au prototype
Joueur.decrire = function () {
    Let description = this.nom + " a " + this.sante + " points de
vie, " + this.force + " en force et " + this.xp + " points
d'expérience";
    return description;
};

//on crée le prototype Adversaire
const Adversaire = Object.create(Personnage);

// on ajoute la function initAdversaire au prototype
Adversaire.initAdversaire = function (nom, sante, force, race,
valeur) {
    this.initPerso(nom, sante, force);
    this.race = race;
    this.valeur = valeur;
};
```

http://www.pompage.net/traduction/classe-et-heritage-en-javascript

**Exercice 1 : une IHM pour l'exemple 2**

L'exemple 2 travaille uniquement en console.
Produisez une IHM qui permette d'afficher les personnages et d'exécuter des méthodes.

## POO – tableau d'objets

### Exemple 3

On peut ranger les objets dans un tableau.

```
let Film = { // Prototype-Classe Film
    init: function (titre, annee) {
        this.titre = titre;
        this.annee = annee;
    },
    // Renvoie la description du film
    decrire: function () {
        let description = this.titre + " (" + this.annee + ")";
        return description;
    }
};

// création d'un tableau de films
let films = [];

// creation d'un film et rangement dans le tableau
let film = Object.create(Film);
film.init("Ta'ang", 2016);
films.push(film);

// creation d'un film et rangement dans le tableau
film = Object.create(Film);
film.init("Divines", 2016);
films.push(film);

// creation d'un film et rangement dans le tableau
film = Object.create(Film);
film.init("Juste la fin du monde", 2016);
films.push(film);

// affichage des films du tableau
films.forEach(function (film) {
    console.log(film.decrire());
});
```

## Exercice 2 : une IHM pour l'exemple 3

L'exemple 3 travaille uniquement en console.
Produisez une IHM qui permette d'afficher les personnages et d'exécuter des méthodes.

## Exercice 3 : tableau d'élèves avec notes et photos en POO

A partir de l'exercice 3 de l'étape 2 : 03-exercice-creer-eleve-correction (tableau d'élève avec photo), faites une version en POO :

- Coder l'étape 3 en programmation objet :
  - ✓ On se dote d'un prototype Classe pour le tableau complet et d'un prototype Eleve pour les élèves.
  - ✓ On se dotera aussi d'une fonction qui permette d'ajouter un nouvel élève.