**JS**

**Dure totale du module : 21H00**

Table des matières

[Objectifs Pédagogiques 4](#_Toc119402286)

[Présentation de JS 5](#_Toc119402287)

[Langage de programmation 6](#_Toc119402288)

[Multi Environnement 6](#_Toc119402289)

[Orienté Objet 7](#_Toc119402290)

[Web Dynamique 8](#_Toc119402291)

[Algo / Syntaxe JS moderne (ES6 et +) 10](#_Toc119402292)

[1 - Setup de JS 10](#_Toc119402293)

[Applications 10](#_Toc119402294)

[Setup Classique 10](#_Toc119402295)

[Côté Template 10](#_Toc119402296)

[async ou defer 12](#_Toc119402297)

[2 – Les Variables 14](#_Toc119402298)

[Anatomie d’une variable 14](#_Toc119402299)

[Déclarer une variable & Assigner une valeur 14](#_Toc119402300)

[Puiseurs Types de variable 15](#_Toc119402301)

[3 – Nombres / Calculs /Modulo ☺/ Incrémenter / Assignement Composé 17](#_Toc119402302)

[Calculs 18](#_Toc119402303)

[Incrémenter / assignement composé / puissance (calcul) 19](#_Toc119402304)

[4 – Chaines de caractères : double simple quote, template strings 20](#_Toc119402305)

[5 – Tableaux array 22](#_Toc119402306)

[7 – Fonctions / Fonctions Fléchée / return / param / param par defaut / scope 26](#_Toc119402307)

[function 26](#_Toc119402308)

[Paramètre 26](#_Toc119402309)

[return 28](#_Toc119402310)

[Paramètre par défaut 28](#_Toc119402311)

[scope 29](#_Toc119402312)

[fonctions fléchées 32](#_Toc119402313)

[8 – Opérateurs condition / condition ternaire 33](#_Toc119402314)

[Opérateurs de comparaison 33](#_Toc119402315)

[Conditions /opérateur ternaire 34](#_Toc119402316)

[9 – Condition if else 35](#_Toc119402317)

[If else if else 35](#_Toc119402318)

[10 – Objets 37](#_Toc119402319)

[Syntaxe / ajout / delete 37](#_Toc119402320)

[hasOwnProperty 38](#_Toc119402321)

[11 – Boucle While / For / ForEach / For …of / For …in / map 40](#_Toc119402322)

[While 40](#_Toc119402323)

[For 41](#_Toc119402324)

[forEach 42](#_Toc119402325)

[for…of 43](#_Toc119402326)

[for…in 44](#_Toc119402327)

[Convertir des objets en tableaux 45](#_Toc119402328)

[12 – Aléatoire (Random) 47](#_Toc119402329)

[14 – ES6 Le Problème de var (let & const) 47](#_Toc119402330)

[15 – ES6 / 7 : déstructuring Objet / array / spread operator 47](#_Toc119402331)

[16 – Classes 48](#_Toc119402332)

[Déclarer une classe / construire une nouvelle instance de classe 49](#_Toc119402333)

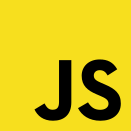
[Classes : Méthodes d’instances 51](#_Toc119402334)

[DOM (Document Object Model) 54](#_Toc119402335)

# Objectifs Pédagogiques

TODO

# Présentation de JS



Voici Brendan Eich :



Nous sommes à l’été 1995, dans les bureaux de la [Netscape Communications Corporation](https://fr.wikipedia.org/wiki/Netscape_Communications_Corporation), le web est en effervescence grâce aux langages HTML et CSS, en parallèle les applications de bureau sont très populaires pour les différents OS (Windows, MacOs, Linux …) et le langage de programmation Java propose une machine virtuelle (très populaire) permettant de s’affranchir des barrières liées au langages spécifique des OS.

Brendan va donc travailler sur la problématique de créer un nouveau langage qui devra répondre aux contraintes suivantes :

* Un langage de programmation
* Multi Environnement
* Orienté Objet
* Permettant de rendre des pages web dynamiques

10 Jours plus tard naissait Javascript…

## Langage de programmation

Brendan s’est inspiré de nombreux [langages de programmation](https://fr.wikipedia.org/wiki/Langage_de_programmation), notamment de [Java](https://fr.wikipedia.org/wiki/Java_(langage)) mais en simplifiant la [syntaxe](https://fr.wikipedia.org/wiki/Syntaxe_JavaScript) pour les débutants.

## Multi Environnement

L’aspect multi environnement réside dans le fait que JS s’exécute au sein du navigateur web de l’utilisateur, et tout le monde utilise un navigateur (encore + aujourd’hui) et on retrouve les navigateurs partout (Desktop, Mobile,Tablettes, TV, Montres …)

(\*Il existe des frameworks JS comme [electron](https://www.electronjs.org/) permettant de créer des apps desktop à partir d’un app web.)

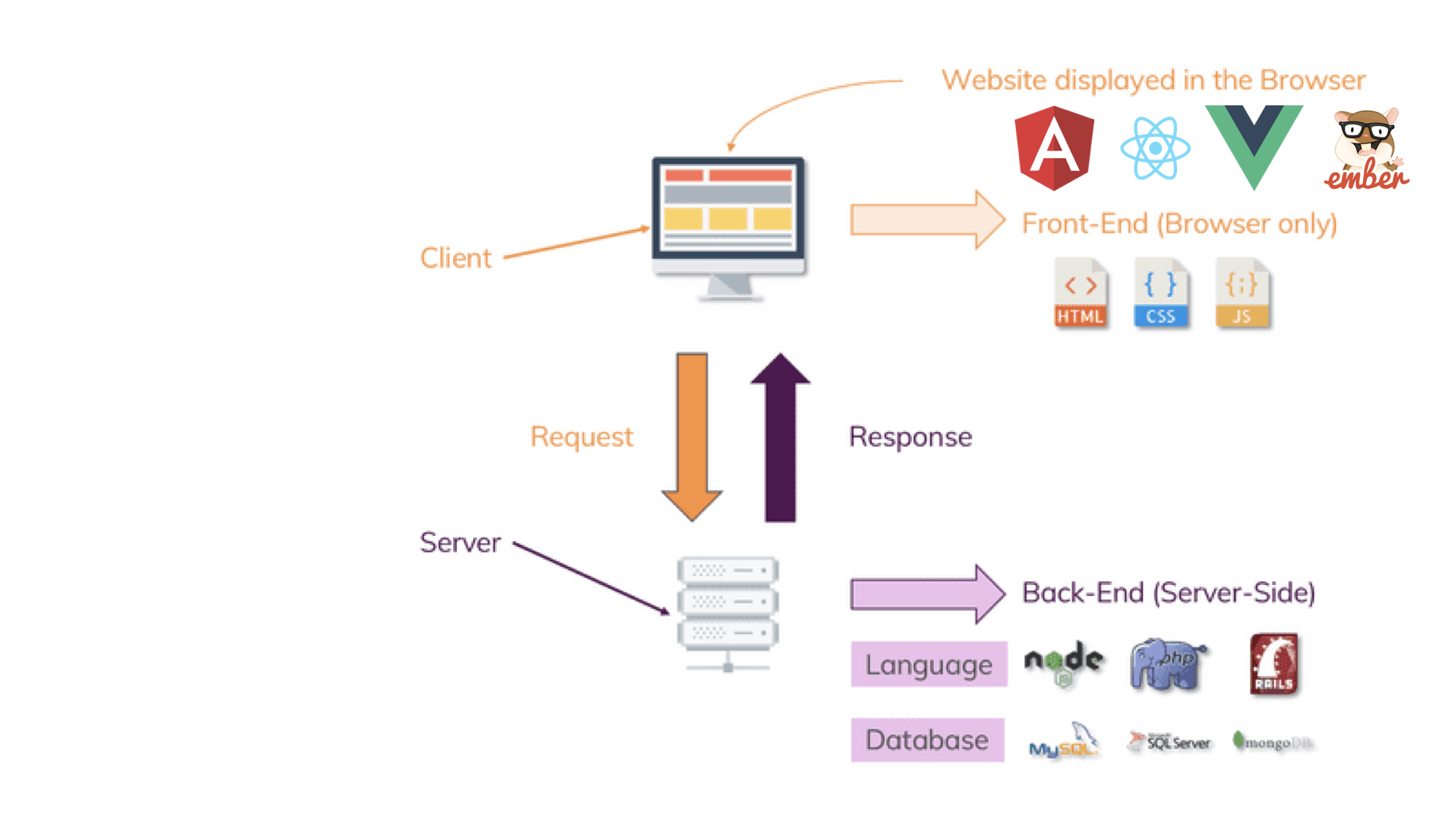
En effet les navigateurs sont composés de 2 moteurs : un moteur de rendu (Render Engine) ainsi qu’un moteur JS (Javascript Engine).



Render Engine (Gauche) / Javascript Engine (Droite)

JS s’exécute dans le navigateur certes mais coté client, Front-End, sur la page HTML qui est déjà chargée.

Par la suite après la version ES6 de JS, on peut exécuter du JS côté serveur (Back-End) via des technologies comme Node.js ou encore Deno.



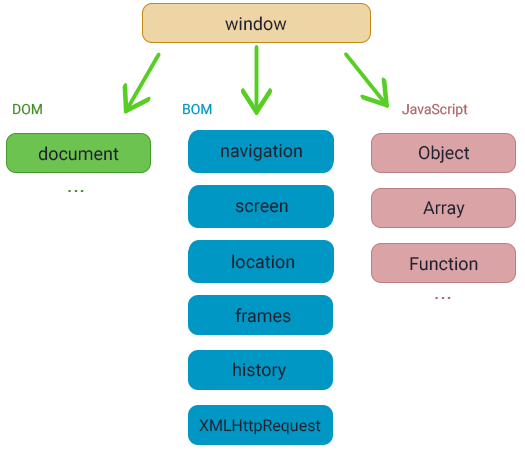
## Orienté Objet

Javascript est un langage conçu pour être orienté objet, à tel point que l’on peut entendre dire « en JS tout est objet », c’est en ce point que JS diffère par rapport à d’autres langage comme Java, pour être plus précis, JS est un langage de programmation orienté objet à prototype, cela signifie que JS va fournir les outils de base du langage (créer une chaine de caractères, une fonction etc..) via un système d’objet là ou Java utilise des classes (en JS une chaine de caractères, une fonction et même une classe sont des objets).

## Web Dynamique

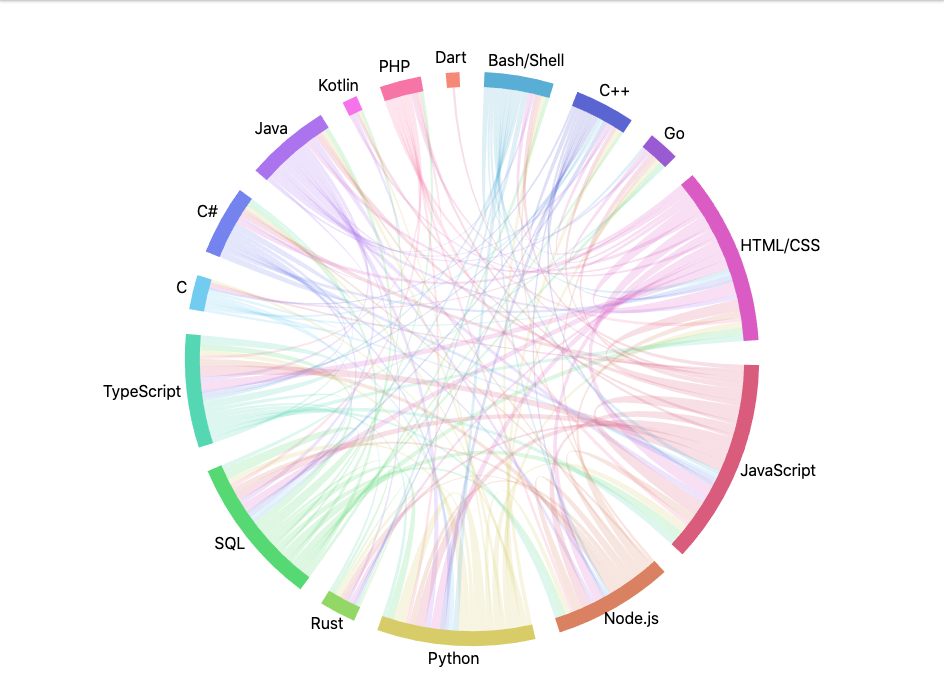
JS va apporter un ensemble d’outils pour manipuler ce qu’il se passe dans la page web, au sein du navigateur de l’utilisateur.

Techniquement JS contient déjà des objets (avec des propriétés et des fonctions) pour tous les éléments affichés dans la fenêtre de l’utilisateur, l’objet window dans lequel va être exécuté notre code JavaScript (nos variables, nos fonctions, nos classes, nos tableaux, etc.) qui pourra manipuler les objets du BOM (Browser Object Model) pour contrôler l’historique ou la navigation et manipuler les objets du DOM (Document Object Model), les éléments HTML et CSS chargés sur la page.



JS : Actuellement

Très vite JS s’est imposé comme un langage incontournable du Web, en combinaison à HTML et CSS dans un premier temps pour améliorer le design des site web, puis au travers de librairies populaires comme JQuery et les évolutions majeures de ES6 qui ont amené de nouvelles technologies côté serveur (Node / Deno) mais aussi les Frameworks (React, Angular, Vue, Svelte…)



<https://stackoverflow.blog/2021/08/02/2021-stack-overflow-developer-survey-results/>

# Algo / Syntaxe JS moderne (ES6 et +)

## 1 - Setup de JS

### Applications

De base nous avons besoin :

* Éditeur de code : [VSCODE](https://code.visualstudio.com/) (au choix) + Extension [Live Server](https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer)
* Navigateurs : Chromium (Google Chrome, Edge, Opera, Brave …) / Firefox, safari ? (Pour tester avec des navigateurs qui ont des moteurs différents)

### Setup Classique

Lier une page HTML avec un fichier JS en utilisant la balise script à la fin du body

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

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

<title>Intro JS</title>

</head>

<body>

<h1>SetUp</h1>

<!--! Les script JS à la fin du body après le chargement de tous les éléments Html -->

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

</body>

</html>

Le fichier JS :

console.log('Bienvenue dans Javascript');

### Côté Template

On peut aussi écrire du code JS dans le template HTML directement dans la balise <script>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

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

<title>Intro JS</title>

</head>

<body>

<h1>SetUp</h1>

<!--! Les script JS à la fin du body après le chargement de tous les éléments Html -->

<script>console.log('Hello World')</script>

</body>

</html>

### async ou defer

On peut aussi placer la balise script pour relier le fichier JS dans le haut du HTML, dans la balise <head> MAIS il faut ajouter l’attribut async ou defer

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

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

<title>Intro JS</title>

<!--! defer : différer l'exécution à la fin du chargement du document -->

<!-- <script src="app.js" defer></script> -->

<!--! async : charger/exécuter les scripts de façon asynchrone. -->

<script src="app.js" async></script>

</head>

<body>

<h1>SetUp</h1>

</body>

</html>

<https://www.alsacreations.com/astuce/lire/1562-script-attribut-async-defer.html>

Defer : Antérieur à la vague HTML5, l'attribut defer existait déjà dans les "anciennes" versions d'Internet Explorer. Il signifie que le navigateur peut charger le(s) script(s) en **parallèle**, sans stopper le rendu de la page HTML. Contrairement à async, l'ordre d'exécution des scripts est préservé, en fonction de leur apparition dans le code source HTML. Il est par ailleurs reporté à la fin du *parsing* du DOM (avant l'événement DOMContentLoaded). De nos jours, cet attribut présente moins d'intérêt car les navigateurs disposent par défaut de techniques internes pour télécharger les ressources en parallèle sans nécessairement attendre les autres.

Async : Nouveau venu dans HTML5, async signifie que le script pourra être exécuté de façon **asynchrone**, dès qu'il sera disponible (téléchargé). Cela signifie aussi que le navigateur n'attendra pas de suivre un ordre particulier si plusieurs balises <script> sont présentes, et ne bloquera pas le chargement du reste des ressources, notamment la page HTML. L'exécution aura lieu avant l'événement load lancé sur window et ne sera valable que pour les scripts externes au document, c'est-à-dire ceux dont l'attribut src mentionne l'adresse.

Ce comportement est bien pratique pour gagner en temps de chargement, il faut cependant l'utiliser avec prudence : si l'ordre n'est pas respecté, un fichier exécuté de façon asynchrone ne pourra *attendre*le chargement d'un précédent, par exemple s'il en utilise des fonctions voire un framework. Il ne faudra pas non plus compter appeler document.write() pour écrire dans le document HTML puisqu'il sera impossible de savoir à quel moment les actions seront déclenchées.

## 2 – Les Variables

Pour rendre une application web dynamique, il nous faut manipuler, interagir avec des données, par exemple le nombre de billets restant pour un concert, une commande avec un numéro de produit, un prix et une date, une adresse mail, etc. Un langage de programmation va donc utiliser des variables pour enregistrer ces données (pour l’ordinateur c’est un espace mémoire.

### Anatomie d’une variable

Une variable va se définir par 3 aspects :

* Un NOM (pour identifier ce que confient la variable)
* Un TYPE (un nombre, une chaine de caractère, etc…)
* Une VALEUR (c’est le contenu de la variable)

JavaScript est un langage dont le typage est *faible* et *dynamique*. Cela signifie qu’il n’est pas nécessaire de déclarer le type d’une variable avant de l’utiliser. Le type de la variable sera automatiquement déterminé lorsque le programme sera exécuté. Cela signifie également que la même variable pourra avoir différents types au cours de son existence

[https ://developer.mozilla.org/fr/docs/Web/JavaScript/Data\_structures](https://developer.mozilla.org/fr/docs/Web/JavaScript/Data_structures)

### Déclarer une variable & Assigner une valeur

⚠️ : Pour assigner un VALEUR à une Variable il faut au préalable que cette variable soit initialisée (avec le mot clé let ou const suivi du NOM de la variable).

PS : Prendre aussi l’habitude de finir une instruction JS avec ;

let maVariable;

Ensuite nous pouvons lui assigner une VALEUR

maVariable = 'Hello World';

C’est crucial car cela permet à cette variable d’être utilisée dans le programme, auquel cas notre code va provoquer des erreurs et donc entrainer le crash de notre application.

99% du temps nous allons déclarer une variable et lui assigner une valeur directement.

let maVariable = 'Hello World';

### Puiseurs Types de variable

Le dernier standard ECMAScript définit 8 types de données :

* Sept types de données [primitifs](https://developer.mozilla.org/en-US/docs/Glossary/Primitive):
  + [Booléen](https://developer.mozilla.org/en-US/docs/Glossary/Boolean) (true / false)
  + [Null](https://developer.mozilla.org/en-US/docs/Glossary/Null)
  + [Undefined](https://developer.mozilla.org/en-US/docs/Glossary/undefined)
  + [Number](https://developer.mozilla.org/en-US/docs/Glossary/Number)
  + [BigInt](https://developer.mozilla.org/en-US/docs/Glossary/BigInt) (proposition pour ES2020)
  + [String](https://developer.mozilla.org/en-US/docs/Glossary/String) (chaine de caractère)
  + [Symbole](https://developer.mozilla.org/en-US/docs/Glossary/Symbol) (type introduit avec ECMAScript 6)
  + et le type [Objet](https://developer.mozilla.org/en-US/docs/Glossary/Object)

/\*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 2-VARIABLES

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

//! On déclare une variable avec let ou const (ou var dans les anciennes verisons de JS)

let maVariable;

//! On assigne une valeur à une variable avec le signe =

maVariable = 'Hello World';

console.log(maVariable);

//? Les types de variables (JS utilise un typage dynamique)

let uneString = 'MDR';

let unNombre = 11;

let unBooleen = true;

let unTableau = [1,2,3,'Hello'];

let unObjet = {

propriete1 : 22,

propriete2:'LOL'

};

let uneFonction = function maFonction (){

console.log('Fonction qui affiche HelloWorld');

}

console.log(uneString);

console.log(unNombre);

console.log(unBooleen);

console.log(unTableau);

console.log(unObjet);

console.log(uneFonction);

<https://github.com/jefff404/cours-javascript/tree/1-variables>

## 3 – Nombres / Calculs /Modulo ☺/ Incrémenter / Assignement Composé

Bien entendu, une des utilités principales des langage de programmation c’est de pouvoir faire des calculs (+ ou - complexes) pour cela nous allons utiliser des opérateurs de calculs (+ , - , \*, /) comme en Mathématiques.

Une autre technique très largement utilisée dans les applications web consiste à cumuler des chiffres.

Imaginons une variable qui nous sert de compteur de (vues, likes, lectures, etc…) que nous allons incrémenter (c’est-à-dire rajouter une valeur à ce compteur).

### Calculs

//! Un calcul simple

console.log(42\*675);

//! Un calcul simple avec des variables

let unChiffre = 9;

let unNombre = 33;

console.log(unChiffre\*unNombre);

//! Attention pour les décimaux on utilise la notation avec le point

//! 2.9

console.log(2,9+1,3);

console.log(2.9+1.3);

//! On peut prioriser les calculs avec des parenthèses

console.log((1+1)-(2\*3)/2);

//! Modulo pour calculer le reste d'une division euclidienne

console.log(10%2);

//! L'incrémentation

let unCompteur = 0;

console.log(unCompteur+1);

unCompteur = 0;

unCompteur = unCompteur+1;

console.log(unCompteur);

unCompteur = 0;

### Incrémenter / assignement composé / puissance (calcul)

//! Assignement composé (idéal pour cumuler)

//! ci dessous équivalent à unCompteur = unCompteur+1

unCompteur +=1;

console.log(unCompteur);

//! pour ajouter 1, on peut incrémenter avec ++

unCompteur = 0;

unCompteur ++;

console.log(unCompteur);

//! on peut aussi décrémenter avec --

unCompteur --;

console.log(unCompteur);

//! Assignement composé avec +10

unCompteur +=10;

console.log(unCompteur);

//! Assignement composé avec la multiplication

//!(ça marche avec tous les opérateurs de calculs)

unCompteur \*=10;

console.log(unCompteur);

unCompteur /=10;

console.log(unCompteur);

//! nouvelle syntaxe pour faire les puissance

//!(anciennement on utilisait la fonction Math.pow())

console.log(unChiffre\*\*9);

<https://github.com/jefff404/cours-javascript/tree/2-calculs>

## 4 – Chaines de caractères : double simple quote, template strings

En javascript nous pouvons tout aussi bien gérer des chaines de caractères, cela peut constituer juste un mot, une phrase ou tout un paragraphe par exemple pour cela ci-dessous nous allons voir les différentes syntaxes permettant de gérer des strings (chaines de caractères).

Nous allons donc voir les simple quote, les double quote, (guillemets simple ou double) ainsi que les littéraux de gabarits (ou template strings) qui facilite l’affiche d’une variable au sein d’un texte.

Dans certains cas il peut être utile d’assembler plusieurs chaines de caractères, c’est le principe de la concaténation, pour cela on utilise l’opérateur +

//! pour les chaines de caractères on utilise les simple ou double quote

let bonjour = 'Bonjour';

let unMessage = "Bienvenue";

console.log(bonjour,unMessage);

// si un caractère gène la syntaxe de la string,

//! on utilise l'antislash pour échapper un caractère

let unTexte = "Bonjour \"MR Gingle\"";

let unTxt = 'Aujourd\'hui';

console.log(unTexte,unTxt);

//! concaténation : avec l'opérateur + on assemble plusieurs strings

let concatenation = bonjour + " et " + unMessage +', il fait beau temps' + unTxt;

console.log(concatenation);

//! Autre solution : les Template Strings, on fait notre chaine de caractère

//! avec les backticks `` (altgr +7)

//! On est déjà dans une strings et on veut isoler les variables

//! avec cette syntaxe : ${maVariable}

let uneTemplateString = `Hello ! ${bonjour} et ${unMessage}

on retourne à la "ligne" plus besoin des 'antislash'`;

console.log(uneTemplateString);

//! sur une string on peut accéder à la propriété length

//! Ce qui nous retourne le nombre de caractères dans la string

//! warning : un espace est compté comme un caractère

let uneVille = 'Vene zuela';

console.log(uneVille.length);

//! Du coup on peut aussi accéder à un caractère en particulier

//! avec la syntaxe crochet []

console.log(uneVille[3]);

//! Astuce pour trouver automatiquement la dernière lettre d'une string

console.log(uneVille[uneVille.length-1]);

<https://github.com/jefff404/cours-javascript/tree/3-strings>

## 5 – Tableaux array

Dans des applications qui gèrent plusieurs données il peut s’avérer judicieux de les ranger dans un tableau (dans une seule donnée on aura plusieurs informations).

Pour la syntaxe des tableaux nous assignons à une variable, des crochets [ ], c’est à l’intérieur de ces [ ] que l’on peut ranger des données (des strings, des numbers, des variables … bref tout type de données)

Ci-dessous un exemple d’un tableau qui contient des nombres :

let unTablo = [11,90,88888];

Les tableaux on cet aspect pratique qu’ils ont un système d’indexation (on peut accéder à chaque case du tableau) en utilisant cette syntaxe :

unTablo[2]

Ci-dessus on accède au contenu de la case numéro 2 du tableau qui contient le nombre…88888

(⚠️ le système d’indexation des cases d’un tableau commence à 0, donc la case numéro 0 est en fait la première case du tableau).

//! EXO 4 ARRAYS

//TODO: Créer 1 variable pour un nom,

//TODO: Créer une variable pour un age,

//TODO: Créer une variable passions qui est un tableau qui contient 2 chaines de caractères (au choix)

//TODO: Puis créer une variable tabUser qui est un tableau qui contient les variable du nom, de l'age et passions

//TODO: en Console on affiche le tabUser

//TODO : en passant par tabUser on veut afficher en console uniquement les passions

//TODO : en passant par tabUser on veut afficher en console uniquement la 2ème passion

let nameUser = 'Dong Rodrigue';

let ageUser = 65;

let passions = ['Le Drift','Jonquilles'];

let tabUser = [nameUser,ageUser,passions];

console.log('tabUser : ',tabUser);

console.log(tabUser[2]);// c'est le tableau passions

console.log(tabUser[2][1]);// dans passions (tabUser[2]) on accède à la case [1]

💡 Dans un tableau, les données (le contenu des case) ne sont pas immutable, c’est-à-dire que l’on peut leur réassigner une valeur comme ceci :

tabUser[0] = 111;

tabUser[1] = 'DonDiegoDelavega';

tabUser[2][1] = 'Le Cinéma';

console.log('tab modifié : ',tabUser);

💡Comme pour les chaines de caractères sur les tableaux il existe nativement une propriété length, qui renseigne le nombre de case que contient un tableau :

console.log(tabUser.length);

⚠️ Attention à ne pas confondre lorsque l’on fait ceci :

console.log(tabUser[1].length);

(Ici c’est la propriété length des chaines de caractères qui est utilisé, car dans cette case se trouve une chaine de caractères).

//!EXO 4.3 Ajout ƒ° .push()

//TODO : créer un nouveau tableau qui contient des trucs

//TODO : allez se renseigner la ƒ° push()

//TODO : utiliser push pour ajouter un nouveau truc au tableau

//TODO: On affiche en console ce tableau

Une solution :

(la fonction push(),peut accepter plusieurs paramètres en les séparant par des virgule pour ajouter plusieurs données en utilisant une seule fois la fonction.

let testTabAjout = [10,10,10];

console.log('Avant Ajout :',testTabAjout);

testTabAjout.push('Cortex',92,'Les Pyramides');

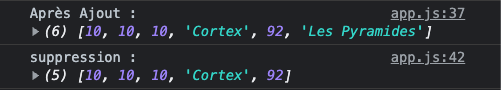
console.log('Après Ajout : ',testTabAjout);

Pour les tableaux il existe également la fonction .pop(), qui va permettre de supprimer le dernier élément du tableau :

// pop pour suppr le dernier element du tableau

testTabAjout.pop();

console.log('suppression : ',testTabAjout);



//! EXO 4.4 ARRAYS Récap

// TODO: Créer 2 variables leNom et lePrénom

//TODO: Créer un tableau laPhrase et on y ajoute via (push)Le nom ,Le prénom Les initiales

//TODO: Afficher le tableau dans la console le nom le prénom et les initiales

Une Solution :

let leNom = 'Garcia';

let lePrenom = 'José';

let initiales = lePrenom[0] + leNom[0];

let laPhrase = [];

laPhrase.push(leNom,lePrenom,initiales)

console.log(laPhrase);

<https://github.com/jefff404/cours-javascript/tree/4-tableaux>

## 7 – Fonctions / Fonctions Fléchée / return / param / param par defaut / scope

Dans tous les langages de programmation on retrouve le concept de fonctions, il s’agit d’un bloc de code (un sous-programme dans notre programme si l’on veut) qui va contenir plusieurs instructions, de cette manière plutôt que d’écrire plusieurs fois certaines lignes de code, on va les regrouper au sein d’une fonction, de cette manière nous pourrons exécuter ailleurs dans le programme toutes les lignes de code de la fonction

### function

Syntaxe : comment déclarer une fonction :

// On déclare la fonction sous le nom maSuperFonction et le travail qu'elle doit faire

function maSuperFonction(){

// ICI on code ce que doit faire la ƒ°

console.log('Ma fonction fonctionne');

console.log(10+2);

};

Ensuite quelque part dans le programme il va falloir exécuter la fonction :

// On va exec au moins 1 fois notre ƒ°

maSuperFonction();

### Paramètre

Les fonctions ont aussi un concept de paramètre, dans le cas où les instructions au sein de la fonction ont besoin d’une variable extérieure. Ci-dessous une fonction qui va afficher en console une variable unNom

// Quand une ƒ° a besoin de paramètres ???

function disMonNom(unNom){

console.log(unNom);

};

disMonNom('JACKO');

La variable unNom est définie directement lors de l’utilisation de la fonction.

//! EXO 5 : Function

// TODO : créer une fonction qui prend un nombre en paramètre

// TODO : La ƒ° doit afficher en console: 33 + le nombre reçu en paramètre

// TODO : créer une autre fonction qui prend 2 nombres en paramètre

// TODO : Cette seconde ƒ° doit afficher en console l'ADDITION des 2 nombres reçus en paramètre

Une Solution :

function fonctionUn(unTruc){

console.log(33+unTruc);

}

fonctionUn(7);

💡 On peut aussi prévoir des fonctions nécessitant plusieurs paramètres comme ceci

function fonctionDeux(unTruc,unBidule){

console.log(unBidule+unTruc);

}

fonctionDeux(10,88);

### return

Les fonctions gèrent également le concept de return, c’est-à-dire que nous pouvons écrire des fonctions qui vont retourner quelque chose (une variable, un résultat, etc..). Dans les exemples précédents, si on analyse bien, nos fonctions font deux choses techniquement : un calcul ET l’affichage du résultat de ce calcul, mais admettons, nous voulons garder un code clair et précis, on veut une fonction qui fait Uniquement du calcul, l’affichage du résultat sera géré ailleurs dans le programme.  
Il faut donc adapter notre fonction pour qu’elle retourne un résultat que l’on stockera dans une variable.

Exemple avec une fonction qui a pour but de soustraire 2 nombres :

//? Quand une fonction doit return quelquechose

function moins(a,b){

return a-b;

};

// à l'execution de la ƒ° le resultat du calcul (ce que la fonction renvoit/return)

// sera stocké dans une variable monResultat

let monResultat = moins(99,20);

// Ou alors, le résultat est direct return dans un console.log()

console.log('Monresultat:',moins(99,20));

console.log('exec de la fonction direct dans console.log() : ',moins(99,20));

### Paramètre par défaut

Une bonne pratique lorsque l’on écrit des fonction (surtout en travail collaboratif), consiste à renseigner un paramètre par défaut dans le cas où on oublie de renseigner un paramètre quand on exécute une fonction.

// PARAMETRE PAR DEFAUT

// Si à l'utilisation de la ƒ° on oubli le paramètre

// Pas d'erreur car le param par défaut sera 'OuiOui'

function sayTheName(unNomBidon='OuiOui'){

console.log(unNomBidon);

};

sayTheName();

### scope

⚠️ Quand on code des fonctions (quand les scripts se complexifient), il est nécessaire de prendre en compte la notion de scope soit la portée des variables.   
Une fonction va pouvoir utiliser des variables déclarées globalement mais le programme global ne peut pas utiliser une variable déclarée dans une fonction.

// ? La notion de scope (la portée d'une variable)

// ? Dans l'exemple ci-dessous on a 2 fois la même variable testScope1 qui est déclarée ?????

// ? En fait même si elles ont le même nom ce ne sont pas les même espaces mémoires qui sont alloués

// ? let testScope1 = 99; est dans le scope global de notre programme

// ? let testScope1 = 12; est dans le scope de la fonction

let testScope1 = 99;

function maFonctionTestScope(){

let testScope1 = 12;

console.log('scope de la fonction :',testScope1);

};

maFonctionTestScope();

console.log('scope hors de la fonction :',testScope1);

Quizz : Trouver le Bug

//! EXO 5.1 : Quizz Function

//TODO : Pourquoi ca beug ?

function buggyFunction() {

let wtf = 9;

console.log(wtf);

};

buggyFunction();

console.log(wtf);



Quizz 2 : Trouver le Bug

//! EXO 5.1.2 : Quizz Function

//TODO : Pourquoi ca beug / Pourquoi ca marche pas ?

let something = 44;

function functionBugParent() {

let something = 9;

console.log(something);

console.log(lesNews);

function functionBugEnfant() {

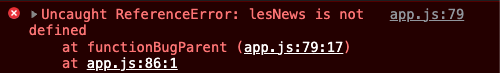
let lesNews = `il est 99h67`;

}

};

functionBugParent();

console.log(something);



//! EXO 5.2 : La moyenne de 2 notes

//TODO: Créer une fonction qui calcule la moyenne de 2 notes

//TODO: Afficher le résultat en console

Une Solution :

let noteSport = 8;

let notePhilo = 2;

let laMoyenne = moyenne2notes(notePhilo,noteSport);

// On peut executer la ƒ° AVANT de la définir (pas d'ordre pour décrire les fonctions)

function moyenne2notes(a,b){

return (a+b)/2;

};

console.log('La moyenne des 2 notes : ',laMoyenne);

### fonctions fléchées

💡 Autre syntaxe, les fonctions fléchées : une autre manière d’écrire des fonctions, plus besoin du mot clé function ni du mot clé return

Exemple avec une fonction de calcul qui sont anonyme (elle n’ont pas de nom), mais on peut les stocker dans une variable, (la variable s’utilise alors comme une fonction).

Fonction avec syntaxe classique :

let somme = function(a, b) {

return a + b;

};

Mode Fonction fléchées :

let leCalcul = (a, b) => a + b;

console.log(leCalcul(1, 2));

Avant la => ce sont les paramètres Après la => c’est ce que return / exécute la fonction

<https://github.com/jefff404/cours-javascript/tree/5-fonctions>

## 8 – Opérateurs condition / condition ternaire

### Opérateurs de comparaison

Autre concept qui sera surtout utiliser pour les conditions, ce sont les opérateurs de comparaison cela renvoi un booléen (true ou false).

//!-------OPERATEURS DE COMPARAISON-------

// ? on peut utiliser == (égalité) & != (différent de) pour comparer 2 variables

// ? Cela nous renvoit un BOOLEAN (true ou false)

let a=1;

let b=2;

let c='2';

console.log(a==a);

console.log(a==b);

console.log(a!=a);

console.log('⚠️ Type',b==c);

console.log('⚠️ Type avec ===',b===c);

// ? Plus d'autres opérateurs de comparaison (on connait déjà)

console.log(a>b);

console.log(a<b);

console.log(a>=b);

console.log(a<=b);

à noter la différence pour vérifier une égalité entre l’opérateur == et ===, le triple égale va permettre de vérifier aussi si les variables comparées sont du même type.

### Conditions /opérateur ternaire

Une syntaxe pour écrire des conditions simples qui renvoi quelque chose ou quelque chose d’autre si une condition est remplie ou non. Grace à l’opérateur « ? »

Avant le ? on décrit notre condition, Après le ? nous avons ce qui est return quand la condition est remplie puis « : » et ce qui est return quand la condition n’est pas remplie

//!-------CONDITIONS TERNAIRES-------

// ? on combine un opérateur de comparaison et l'opérateur ? pour établir une condition qui return une chose ou une autre chose

// ? cela permet de faire une condition if (simple) avec une syntaxe racourcie

let whatIsYourAge = 6;

console.log(whatIsYourAge >18 ? '🍹':'👮‍♂️');

// Astuce pour check si une variable est définie (si ya QQchose dans son espace mémoire)

let userPremium;

// On check si une variable est définie la condition c'est juste uneVariable ?

console.log(userPremium?'OK 🤙':'Not OK 👺');

// ↑ c'est l'équivalent de ↓

console.log(userPremium ==true?'OK 🤙':'Not OK 👺');

// on doit lui assigner QQCHOSE

userPremium = 'YES';

console.log(userPremium?'OK 🤙':'Not OK 👺');

On peut aussi combiner plusieurs conditions avec les opérateurs   
|| (une condition OU une autre condition), && (une condition ET une autre condition)

// ? On peut utiliser des operateur aussi pour combiner des conditions && (pour ET) || (pour OU)

console.log(3==3&&3<4);

let typeUtilisateur = 'Extra';

console.log(typeUtilisateur == 'Extra' || typeUtilisateur == 'Premium');

<https://github.com/jefff404/cours-javascript/tree/6-expressions-operateurs>

## 9 – Condition if else

### If else if else

Avec if, nous allons pouvoir exécuter du code seulement si une condition est remplie, on peut combiner if avec else qui correspond à SINON

Dans l’exemple ci-dessous nous avons une fonction qui prend un nombre en paramètre et à l’interieur de cette fonction nous avons plusieurs conditions :

SI le nombre est supérieur ou = à 30 alors on return une phrase

SINON SI le nombre est inférieur à 10 alors on return une autre phrase

SINON on return une troisième phrase

//!-------CONDITION avec IF ELSE-------

// ? Avec if on va pouvoir créer un bloc de code qui s'exécute si une condition est remplie

function calculTableResto(nombreDeReservation) {

if (nombreDeReservation>=30){

return 'il nous reste pas beaucoup de tables, ca serait pour combien de personnes ?';

}

else if(nombreDeReservation<10){

return 'Il nous reste une table'

}

else{

return 'On est fermé !'

}

};

console.log(calculTableResto(50));

//! EXO 7 - IF ELSE

// TODO: Créer une fonction reçoit un tableau de 3 notes et qui calcule une moyenne entre ces 3 notes

// TODO: Dans cette ƒ°, SI la moyenne est suppérieur ou égale à 15 on renvoi une string (très Bien)

// TODO: Dans cette ƒ°, SINON SI la moyenne est suppérieur ou égale à 10 on renvoi une string (assez Bien)

// TODO: Dans cette ƒ°, SINON renvoi une string (refus)

Une Solution :

function msgMentionBacOfficiel(tabNotes) {

let moyenneCalc = (tabNotes[0]+tabNotes[1]+tabNotes[2])/tabNotes.length;

console.log('la Moyenne au Bac : ',moyenneCalc);

if (moyenneCalc>=16) {

return "Tu as Gagné !"

} else if (moyenneCalc >=10 && moyenneCalc<16) {

return 'Assez Bien'

} else {

return 'YO T NUL GRO'

}

};

console.log(msgMentionBacOfficiel([13,6,3]));

<https://github.com/jefff404/cours-javascript/tree/7-if-else>

## 10 – Objets

### Syntaxe / ajout / delete

Autre type de variable utile pour stocker dans une variable plusieurs information, les objets avec la syntaxe en accolades { }, assez similaires aux tableaux (mais les objet n’ont pas de système d’indexation), c’est à nous de définir les propriétés (les clé) et leur assigner avec « : » une valeur.

// ? syntaxe { uneProprieté:uneValeur }

// ? dans un objet on assigne avec : plutot qu'avec =

let user = {

id:3657826,

'name':'Seagal',

firstname:'Steven',

badges:['🎥','👮‍♂️','🎸','🥋','🎤']

};

console.log(user);

On peut accéder aux propriétés d’un objet avec la notation en point

console.log(user.id);

console.log(user.name);

Ou avec la notation en tableau associatif

console.log(user['id']);

Pour ajouter une propriété on fait une assignation de valeur (si la propriété existe sa valeur est écrasée par la nouvelle, sinon cela créer la propriété)

// ? On peut ajouter simplement des propriétés dans un objet avec une assignation de valeur

// ? Si on assigne à une propriété déjà existante cela écrase la valeur

// ? Mais Si on assigne à une propriété non existante cela créee la propriété

user.dps = 9999;

On peut également effacer une propriété d’un objet avec delete

// ? On peut supprimer une propriété

delete user.badges;

### hasOwnProperty

JS propose plusieurs fonctions natives utilisables sur des objets notamment. hasOwnProperty(), qui renvoi true ou false pour vérifier si la propriété d’un objet existe

Ci-dessous on utilise la fonction dans un console.log() directement.

// ? une ƒ° native de JS poour connaitre les propriétés d'un objet, hasOwnProperty()

let menuDuJour={

entree:"Bassine d'Aioli",

plat:"Rat Adulte",

dessert:'île Fidji'

};

console.log(menuDuJour);

console.log(menuDuJour.hasOwnProperty('entree'));



// ! EXO 8 OBJECTS

// TODO : Refaire l'exo avec les passions en mode objet

Une Solution :

let nameUser = 'Dong Rodrigue';

let ageUser = 65;

let objetUser = {

'nom' : nameUser,

'age' : ageUser,

'passions': {

passion1:'Le Drift',

passion2:'Jonquilles'

}

};

console.log('objet de utilisateur : ',objetUser);

console.log(objetUser.nom);

console.log(objetUser['passions']);// c le taleau passions

console.log(objetUser.passions.passion2);

Comme pour les tableaux, dans un objet les propriétés peuvent être réassignées à une valeur

objetUser.name = 111;

objetUser.age = 'DonDiegoDelavega';

objetUser.passions.passion2 = 'Le Cinéma';

<https://github.com/jefff404/cours-javascript/tree/8-objets>

## 11 – Boucle While / For / ForEach / For …of / For …in / map

Comme dans tous les langages de programmation Javascript a un système de boucle, de base cela va nous permettre de répéter des instructions de code selon une condition. Les boucles vont également s’avérer utile par la suite, pour parcourir des itérables comme des tableaux ou des objets.

### While

Correspond à répéter une ou plusieurs instructions TANT QUE une condition est vraie.

let unIndex = 0;

while (unIndex < 10) {

console.log("Le Nombre : " + unIndex);

unIndex++;

};

Ci-dessus on a un index initialisé à 0 et TANT QUE cet index est strictement inférieur à 10 ALORS, on va faire un console.log(), puis ne pas oublier d’incrémenter l’index pour pouvoir passer à une itération de boucle suivante.



### For

Autre manière de créer des boucles, avec for(), dans les paramètre on va pouvoir directement initialiser un index, définir une condition et incrémenter l’index dans l’exemple ci-dessous nous allons faire une boucle visant à parcourir chaque case d’un tableau pour l’afficher en console.

let listeFilm = ['Ultime Décision','Mission Alcatraz','Attack Force'];

//? Boucle for, on définit un index (ici c'est i),

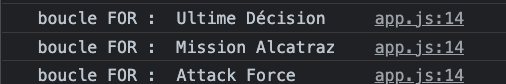
//? puis on définit une condition qui va définir le nombre de fois que le code dans la boucle sera éxecuter

//? puis on définit comment on passe à la prochaine itération de la boucle (ici i++, on augmente de 1 le numero de la case que l'on console.log)

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

console.log('boucle FOR : ',listeFilm[i]);

};



### forEach

Une autre alternative, la fonction forEach de JS automatise le parcours d’un tableau ou objet (sans que l’on ait à gérer un système d’indexation (i++))  
forEach va prendre en paramètre une fonction, cette même fonction pourra avoir un paramètre qui correspondra à chaque case parcourue. (Généralement dans la parenthèse de forEach on passe une fonction fléchée).

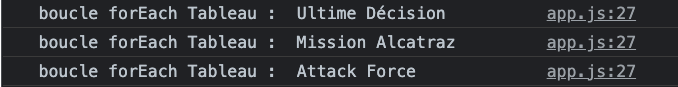
let listeFilm = ['Ultime Décision','Mission Alcatraz','Attack Force'];

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

// ? On va choisir une variable temporaire pour parcourir les elements du tableau

listeFilm.forEach(unFilm => console.log('boucle forEach Tableau : ',unFilm));

Ici chaque case du tableau sera stockée temporairement dans unFilm.



### for…of

Encore une alternative pour parcourir des variables tableaux (et autre) c’est la boucle for … of, qui de la même manière que dans l’exemple précèdent dans lequel on va définir une variable temporaire pour parcourir chaque case du tableau :

//? L'instruction for...of permet de créer une boucle Array qui parcourt un objet itérable

//? (ce qui inclut les objets Array, Map, Set, String, TypedArray, l'objet arguments, etc.)

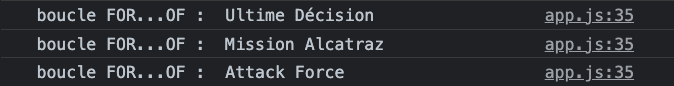
//? et qui permet d'exécuter une ou plusieurs instructions pour la valeur de chaque propriété.

// on définit une variable temporaire pour parcourir le tableau

for(let unElementTablo of listeFilm){

console.log('boucle FOR...OF : ',unElementTablo);

};



### for…in

Si l’on prend le cas des objets JS propose aussi un équivalent à for of (pour les variables de type Array), les boucles for in qui ont exactement la même utilisation que l’exemple précédent

const userData = {

name: 'John Doe',

email: 'john.doe@example.com',

age: 25,

dob: '08/02/1989',

active: true

};

Il faut définir une variable temporaire qui stockera les clés (propriétés) de l’objet

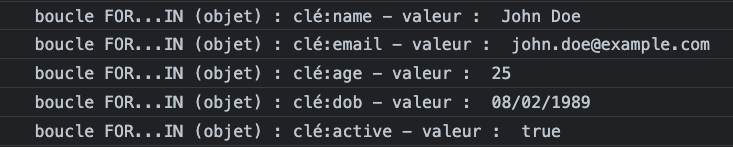
// on définit une variable temporaire pour parcourir le objet :)

for(let cleObjet in userData){

console.log(`boucle FOR...IN (objet) : clé:${cleObjet} - valeur : ${userData[cleObjet]} `);

};

Ici durant le parcourt de l’objet chaque propriété ou clé seront stockées temporairement dans la variable cleObjet.  
Rappel : pour accéder aux propriétés d’un objet on la notation en tableau associatif unObjet[quelquechose]



### Convertir des objets en tableaux

Depuis sa version ES8 JS propose des fonctions utilisables sur les Objets qui vont pouvoir transformer leurs clés et ou leurs valeurs sous forme de tableau (pour utiliser les ƒ° forEach ou map par exemple).

//? Parcourir les Objet (Depuis javaScript ES8)

//? La Method .keys() qui convertit les clés d'un objet en tableau

//? La Method .values() qui convertit les valeurs d'un objet en tableau

//? La Method .entries() qui renvoit un tableau dans un tableau pour combiner clé - valeur

const keyUser = Object.keys(userData);

console.log("les clé de l'objet converties en array : ",keyUser);

const valuesUser = Object.values(userData);

console.log("les valeur de l'objet converties en array : ",valuesUser);

const convertedDataUser = Object.entries(userData);

console.log("les entrées de l'objet converties en array : ",convertedDataUser);

// De fait, une fois les objets convertis en tableau on peut ruser et utiliser forEach par exemple :

valuesUser.forEach((lesValeurs)=>{

console.log(`FOREACH avec objet converti en tableau chaque valeurs : ${lesValeurs}`);

});

convertedDataUser.forEach(([key, value])=>{

console.log(`FOREACH avec objet converti en tableau : ${key}: ${value}`);

});

Exo Array et fonction map

// TODO :JS map phase 1

// TODO : côté template html rajouter plein de <p></p>

// TODO :On va récupérer TOUS les <p> de notre page dans une variable lesTxt via getElementsByTagName

// TODO :On va faire un console log de lesTxt

//TODO JS map Phase 2

//TODO Avec la methode Array.from(), dans une nouvelle variable textesTab on va transformer notre htmlCollection en array

//TODO On console log la variables textesTab

//TODO JS Map Phase 3 (on peut travailler sur un Array)

//TODO Sur textesTab on va utiliser la ƒ° map(),

//TODO dans map(), on va lui passer en param une fonction fléchée qui elle a en parametre une variable temporaire (auChoix)

//TODO cette fonction fléchée elle va modifier le innerHTML ou innerText de la variable temporaire

<https://github.com/jefff404/cours-javascript/tree/9-loops>

## 12 – Aléatoire (Random)

## 14 – ES6 Le Problème de var (let & const)

## 15 – ES6 / 7 : déstructuring Objet / array / spread operator

## 16 – Classes

A ce stade là nous avons des notions de base en JS (les objets ☺), cela nous permet de construire des pages / des applications web Dynamique.

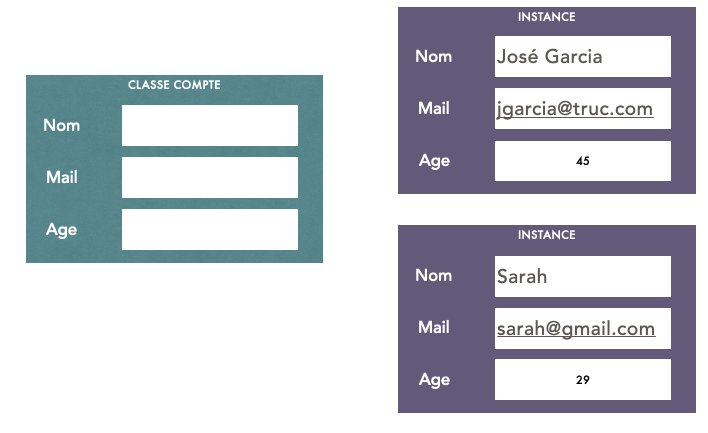
A fur et à mesure les projets vont se complexifier et on va vouloir gérer plusieurs objets.  
Mais par exemple si l’on veut créer une app de messagerie on voudra gérer des objets utilisateur et des objets messages… Mais en fait même PLUSIEURS objets utilisateur et PLUSIEURS objets messages.

Nous n’allons pas déclarer littéralement chaque message et chaque utilisateur de l’application (il peut y en avoir à l’infini).

Pour résoudre cette impasse de programmation, il nous faut un système de classes pour répondre à cette problématique de créer des objets du même type de manière plus facile, rapide et fiable.  
Programmation orientée Objet (P.O.O)

Dans l’illustration ci-dessous sont représentés en vert la classe Compte, les classe utilise une fonction constructor qui reçoivent en paramètre les données nécessaires pour créer des nouveaux objets, ici pour créer des nouveaux Comptes il nous faut renseigner un Nom un Age et un Mail.

En mauve il s’agit de 2 instances de la classe Compte (deux exemplaires de Compte)



### Déclarer une classe / construire une nouvelle instance de classe

//? Le nom des classes commence toujours par une majuscule (par convention)

//? On utilise la fonction constructor qui aura en paramètre des données que l'on juge indispensable

//? Pour créer des nouveaux Film

class Film {

constructor(title, author, duration){

// Dans la classe on va déclarer des variables qui recevront chaque paramètres

// Ces variables de classe sont déclarées et manipulées avec this

this.title = title;//\*attribut en IN MODE (passés en paramètres)

this.author = author;

this.duration = duration;

this.nombreLikes = 0; //\*\*On peut aussi avoir des attributs en OUT mode (pas besoin de les passer en paramètre)

}

};

//? Une fois notre classe créée on peut créer + facilement + rapidement des nouveaux films

//?(on va les stocker dans des variables)

let taxi3 = new Film('Taxi III','Luk Bresson',120);

console.log(taxi3);



Autre notation (pour bien différencier les variables this et les variables reçues en paramètre du constructor)

class User {

//\* Constructor -> initialise les données

constructor(nom, prenom, mail) {

this.\_nom = nom;

this.\_prenom = prenom;

this.\_mail = mail;

}

}

### Classes : Méthodes d’instances

Maintenant que nous maitrisons une façon de déclarer des classes et construire des nouvelles instances, nous allons pouvoir également définir des fonctions à l’intérieur d’une classe, de cette manière ces fonctions ne seront utilisables QUE sur des objets de cette classe (techniquement cela s’appelle des Méthodes d’instances), c’est aussi une meilleure organisation du code de l’application (cf. Diagramme de Classe).

class Voiture{

// On peut aussi avoir des classes qui n'ont rien dans le constructor

constructor(){

this.moteurAllumé = false;

};

// Chaque class peut avoir des fonction (plutot méthodes d'instances)

// Qui ne pourrons être utilisées QUE sur des objets de cette classe

demarrerMoteur(){

this.moteurAllumé = true;

console.log('On va Faire des Drift sur le Périph !');

};

};

let superCar = new Voiture;

superCar.demarrerMoteur();

console.log(superCar);

Ci-dessus il n’y aura QUE des objets Voiture qui pourront utiliser la méthodes demarrerMoteur

Exo Class IMC

Exo Class PME

Exo Class COMPTES BANCAIRES.

# DOM (Document Object Model)

18- Présentation DOM

19 – Organisation / organisation Modulaire

20 – Sélectionner éléments Html

21-Déplacer/ Créer / Supprimer des éléments Html

22 – Supprimer élément Html

23 – Créer TextNode / éléments Html

24 – attributs Html : consulter / modifier – ajouter

25 – Travailler avec listes éléments HTMLCollection / NodeList

26 – Travailler avec la taille des éléments Html

27 – Gérer le style des éléments Html

28 – Gérer les classes Css des éléments Html

Exo Scroll

Exo rebondir

Évènement

API

BDD : Firebase

Conclusion :

