Introduction au JavaScript

Massinissa CHAOUCHI

Sommaire

I- Les langages du Web

Maintenant que l'environnement matériel et logiciel du web vous est familier nous allons pouvoir rentrer dans le cœur du sujet. Et commencer par un bref résumé des différents langages de programmation du web, ainsi que leurs positions dans l'architecture client-serveur. Ici nous allons répondre à la fameuse question : « *Ce langage est-il front-end ou back-end ?*».

Si un code est executé sur le navigateur web alors c'est un langage Front-End.(Client)

Si un code est executé par le serveur ou un programme du serveur alors c'est un langage Back-End.

Front-End est back-end est avant tout une affaire du « lieu » d'exécution du code. On parle d'environnement d'exécution ou « runtime » typiquement :

- L'environnement d'exécution de apache2 (un serveur web codé en C) est Linux ; Windows ou MacOS.
- L'environnement d'exécution du « Jeu du Dinosaure » de Google Chrome en cas d'erreur 500 est Google Chrome soit un navigateur web.
- L'environnement d'exécution d'un site WordPress codé en PHP est (le plus souvent) un serveur sous Linux.

Un Environnement d'exécution peut donc est un OS, un autre programme qui tourne lui-même sur un OS.

Si le client exécute le code c'est du front-end, si c'est le serveur alors c'est du back-end. C'est aussi simple que ça.

Vous verrez d'autant plus l'importance de l'environnement d'exécution pour définir si un code est front ou back. Le JavaScript par exemple connaît deux environnements d'exécution possible : Node JS (serveur, back-end) ou le navigateur web (client,front-end). Le JavaScript est donc à la fois Front-End et Back-End, comme beaucoup de langage de programmation généraliste.

Le HTML – Hyper Text Markup Langage

Le HTML est un langage de balisage du web qui permet de créer des pages web. Après analyse du code HTML par le navigateur web, ce dernier affiche une page web en fonction des balises écrites dans le code source HTML.

Le code HTML est exécuté par le navigateur web, soit le client. **Le HTML est donc un langage front-end**.

1- Le CSS – Cascade Sytle Sheet

Le CSS est un langage informatique qui permet de styliser le code HTML pour en améliorer l'aspect visuel. Le CSS est composé de « *sélecteur css*» qui permettent de sélectionner des balises HTML et modifier leurs apparences. Les modifications peuvent aller de la simple police de

caractère à l'animation en passant par l'adaptation de l'affichage pour les écrans mobile, tablette et PC.

Les *sélecteur css* sont très utilisés en JavaScript *front-end* pour sélectionner une balise précise et modifier son comportement grâce à la programmation.

Le code CSS est exécuté par le navigateur web, soit le client. ***Le CSS est donc un langage front-end****.*

2- Le PHP – PHP HyperText Preprocessor

PHP est un langage de programmation back-end. Il permet de modifier le contenu du page web de façon dynamique au moyen de condition, variable et tout les autre outils que l'on retrouve dans tout langage de programation convenable. Sans le PHP le web serait rester statique, la majeur partie des site web du web fonctionne grâce à PHP. Voici quelque exemple des fonctionnalités de PHP :

- Accès à une base de données
- Hashage de mot de passe
- Accéder à des fichiers
- Rend fonctionnel les formulaires HTML.

Chose importante, contrairement au HTML et au CSS, le PHP n'est pas interpréter par le navigateur web mais par le serveur web entre la réception de la requête HTTP du client et l'envoie du code HTML.

Le code PHP est exécuté par le serveur on peut donc dire que ***PHP est un langage back-end****.*

3- Le JavaScript – ECMAScript ou JS

Le JavaScript est un langage de programmation crée par l'entreprise Oracle dans les années 90, il permet à l'origine de rendre dynamiquement le contenu d'une page web sans charger une nouvelle page web du serveur.

Le HTML deviens dynamique avec JS

A la différence du dynamisme offert par le PHP le JavaScript est interprété dans le navigateur coté-client. Ce qui signifie que le navigateur n'a pas besoin d'envoyer une requête HTTP au serveur pour changer le contenu de la page.

Au même titre que le PHP à permis l'essor des blogs et des premiers réseau sociaux comme MySpace, Facebook et WordPress ; JavaScript à permis l'essor des applications web comme *netflix.com*, les services Google : *drive, doc, calc, meet ou* plus récemment encore les versions web de uber-eats, uber, tiktok.

Si vous avez déjà parcouru un site web en ayant l'impression d'utiliser un logiciel ou une application vous pouvez dire merci à JavaScript et à son moteur inclus coté client dans les navigateurs web.

JavaScript, *front-end* ou *back-end* ?

Pour fonctionner le code JavaScript à besoin d'être interprété par un programme appelé *moteur JS*. Un moteur JS se trouve dans tout les navigateurs moderne, voici le nom des plus connus :

- Le moteur *V8* contenu dans Google Chrome et Chromium.
- *Rhino* contenu dans le navigateur Firefox de la fondation Mozilla.
- *JavaScriptCore* contenu dans Safari.

JavaScript est un langage utilisable coté client et coté serveur. En effet l'utilisation classique de JavaScript sous-entend son exécution dans le navigateur web, coté *front-end* donc ; cependant le JS est devenu un langage coté serveur grâce à NodeJS.

NodeJS est un programme qui contient le *moteur V8*. Ce programme est le plus souvent installé sur un serveur et permets donc d'exécuter du JavaScript en *back-end* sur un serveur.

Astuce - Reconnaître une technologie *front-end* d'une technologie *back-end* :

Une technologie est dite *front-end* si le programme qui permet son fonctionnement est contenu sur le client, une technologie est dite *back-end* si le programme qui permet son fonctionnement est contenu dans le serveur. C'est tout. Peux importe que l'on parle de développement web, mobile ou logiciel. Si à l'avenir vous vous demandez si un programme est *back-end* ou *front-end*, regardez où est ce programme est exécuté. Gardez cette astuce en tête.

L’environnement d’exécution de JavaScript

Les navigateurs et *NodeJS* contiennent un moteur JS, il peuvent donc exécuté du JS. Mais pour faire un vrai programme il faut plus qu’un langage de programmation : l’accès au entrée et sortie utilisateur, le clavier, le scrolling, l’écran, les cookies et les systèmes de stockage du navigateur ; pour le serveur l’accès au réseau, le système de fichier, la console. Rien de tout ça n’est contenu dans le langage JavaScript car ces choses la dépendes de **l’environnement d’exécution** du JavaScript (*Runtime environnement*).

Définition – l’environnement d’exécution : c’est la machine ainsi que tout les logiciels sur lesquels le moteur JavaScript fonctionne : système d’exploitation, programme connexes disponible dans le navigateur ou *NodeJS*.

Le navigateur web et *NodeJS* sont les environnements d’exécution les plus communs[1] de JavaScript.

Pensez l’environnement d’exécution comme le corps humain il est fait de muscle et d’organes à disposition du contrôle du cerveau. Ici le cerveau est JavaScript et les organes et muscles sont tout les programmes à disposition du JS pour interagir avec le monde.

1 Electron par exemple est un environnement d’exécution de JavaScript pensé pour les logiciels dit lourd. Discord et Visual Studio Code sont des programmes JavaScript exécutés dans l’environnement d’exécution Electron sur une machine Windows, Linux ou Mac.

II- JavaScript – les prérequis technique.

Vous avez maintenant toute les bases pour aborder sereinement l'apprentissage de JavaScript.

Les prérequis technique à la création de programme JavaScript, aussi appelé *script, sont :*

- Un éditeur de code pour écrire le code source (VSCode)
- Un interpréteur JavaScript (Navigateur ou NodeJS)

1- Installation de VSCode

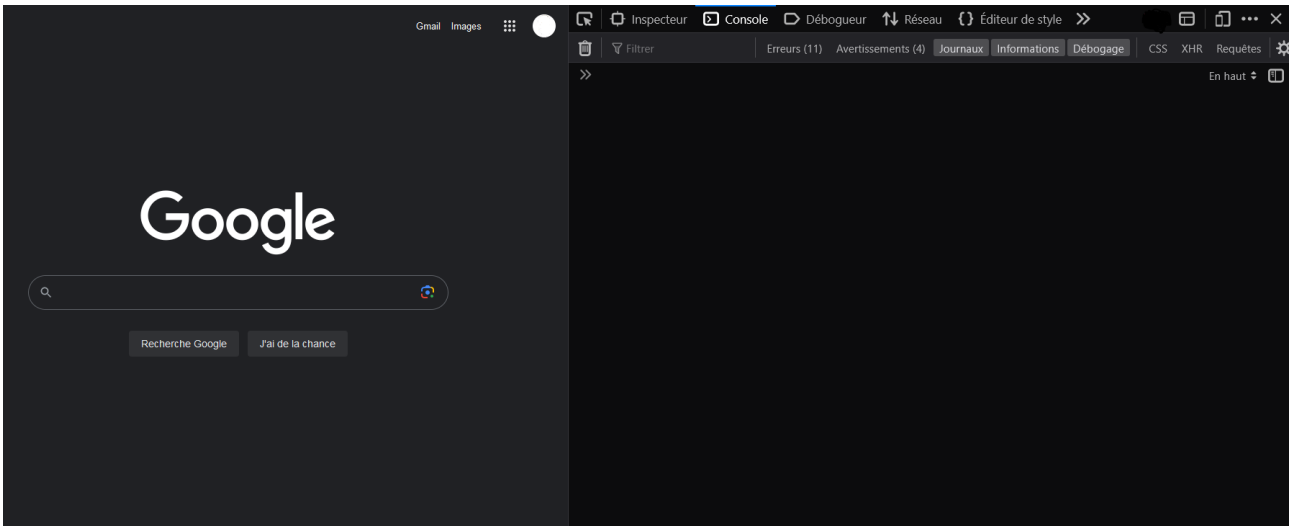
Sous Windows : https://code.visualstudio.com/download

Sous Linux (Ubuntu) :

```
$ wget https://code.visualstudio.com/sha/download?build=stable&os=linux-deb-x64
$ sudo apt install ./code_1.81_amd64.deb
```

2- L'interpréteur JavaScript

Il n'y a rien à installer votre navigateur possède déjà un interpréteur JS. La **console** de cette interpréteur est disponible en appuyant sur la touche **f12** lorsque vous êtes sur votre navigateur.



C'est dans cette console que les résultats de nos premiers programmes apparaîtrons. Vous pouvez également ecrire directement le code dans la console ligne par ligne pour effectuer des tests rapides.

III- Votre premier programme Hello World

Le navigateur interprète le code *JS* lorsqu'il le trouve dans un fichier HTML.

Voici le code minimal pour interpréter du JavaScript dans un navigateur :

index.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Apprendre JavaScript</title>
</head>
<body>
  <script>
    // Écrire du JavaScript ici
  </script>
</body>
</html>
```

La nouveauté est la balise HTML `<script></script>`. Tout ce qui se trouve dans cette balise ne sera pas vue comme du HTML mais comme du JavaScript.

1- console.log - Hello World !

Le « Hello World !» est une tradition dans la programmation, c'est souvent la première chose que l'on tente d'afficher lorsque l'on test quelque chose de nouveau.

Pour afficher « Hello World » dans la console de commande il faut utiliser la fonction *log* de l'objet *console*, comme ceci :

```
console.log("Hello World !");
```

Voici le code complet à enregistrer dans un fichier nommé *index.html*:

index.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Apprendre JavaScript</title>
</head>
<body>
  <script>
    console.log("Hello World !");
  </script>
</body>
</html>
```

Ouvrez le fichier *index.html* dans un navigateur et le message *« Hello World ! »* devrait apparaître, sans les guillemets, dans la console du navigateur (*F12*).

Nous définirons fonction et objet plus tard mais notez bien que la console du navigateur est représenté sous la forme d'un **objet** nommé « ***console*** » et que cet objet possède un fonction nommé « ***log*** » qui permet d'afficher un texte dans la console.

La console fais partie des outils (objets) fournit par l'environnement d'exécution. Nativement en JavaScript l'objet *console* n'existe pas, mais le navigateur nous offre l'objet *console* pour accéder à sa console.

A partir de maintenant j'omettrais la partie HTML du fichier index.html pour se concentrer uniquement sur le JavaScript qui se trouve entre les balises `<script></script>`.

2- Les commentaires

Lorsque l'on code il est d'usage de commenter ce que l'on fait pour que tout le monde puissent comprendre notre programme. En JavaScript les commentaires se font grâce au double slash « // ».

```
// Début du programme
console.log("Hello World !");   // Affiche "Hello World !"
// Fin du programme
```

Les commentaires sont complètement ignoré par l'interpréteur JS et vous pouvez écrire se que vous voulez dedans.

Les commentaires sur une ligne

```
// Je suis un commentaire sur une ligne
// Je suis un autre commentaire sur une ligne
// Tout ce qui se trouve après les doubles slashs est ignoré
```

Les commentaires sur plusieurs lignes

```
/* Je suis un
 commentaire sur
 plusieurs
 lignes */
```

Lorsque vous ouvrez un commentaire avec `/*` tout ce qui se trouve jusqu'à `*/` sera complètement ignoré lors de la compilation du code.

Pour plus de lisibilité et pour vous habituer j'expliquerai souvent le code du cours à l'aide de commentaires.

IV- Un rapide tour des possibilités de JavaScript

Cette partie présente une rapide introduction des possibilités de JavaScript sous la forme de lignes de code commentées. Ainsi vous aurez pour la suite du cours une vue globale des objectifs de connaissances et du chemin à parcourir.

1- Les commentaires

```
// Tout ce qui suit un double slashes est un commentaire
// écrivez y en Français ou en Anglais pour expliquer votre code
```

2- Les variables

```
// Les variables sont des espaces mémoire auxquelles on affecte une donnée
let x;              // Décalaration de la variable nommé x
x = 10;             // Affectation de la valeur 10 à la variable x
x                   // Une variable toute seul sera évaluer par JavaScript
x++;                // Incrémentation de 1, equivaut à x = x + 1
x--;                // Décrémentation de 1, equivaut à x = x – 1

// Le plus souvent on evalue un variable dans une fonction
console.log(x)      // => 10
// Ou dans un calcul lors d'une affectation
let age = x + 5;

Les variables en JavaScript existent sous trois types : les variables globales(var),
locales(let) et les constantes(const).
```

3- Les types de données

```
// JavaScript supporte de nombreux type de donnée

// Le type Number
x = 1;              // Un Number peut être un entier
x = 0.5;            // Ou un nombre décimal
x = -20;            // Voir même un nombre négatif

// le type String - le texte
x = "hello world";  // Un String est toujours contenu entre guillements
x = 'hello world';  // Ou entre simple quotes.

// Boolean
x = true;           // Un Boolean peut être vrai
x = false;          // Ou faux

// Undefined
x = undefined;      // undefined est la valeur d'une variable non déclaré
```

4- Connaître le type d’une variable

```
// La fonction typeof() permet de connaître le type d'une donnée :
typeof(5);                  // => Number
typeof("bonjour");          // => String
let age = 22;
typeof(age);                // => Number
```

5- L’objet

```
// Le type le plus important en JavaScript est Object

// Un objet contient des variables appelées attribut de l'objet
```

```
let livre = {
      titre : "Le seigneur des anneaux",      // l'attribut titre contient une string
      auteur : "J.R.R Tolkien"                // l'attribut auteur contient une string
};
// L'accès au attributs ce fait via l' opérateur . ou [] :
livre.titre;        // => "Le seigneur des anneaux"
livre["auteur"];    // => "J.R.R Tolkien" : Une autre manière d'accéder au attribut
livre.annee = 1967; // Création d'un nouvel attribut
```

6- Les tableaux *Array*

```
// Les tableaux (Array) sont des variables qui contiennes plusieurs valeurs.
let eleves = ["Mathieu","John","Billy","Joe","Rémi"];      // Création d'un tableau

// Accès
eleves[0] // => "Mathieu" : On accède à un element du tableau par un index numérique
eleves[1] // => "John"

// Modification
eleves[0] = "Mathias";     // Affectation de la valeur "Mathias" à l'element 0 du tableau
eleves[0]                  // => "Mathias"
eleves.length;             /* L'attribut length(taille) permet de connaitre le nombre
                              d'élément du tableau */
```

7- Les opérateurs arithmétique et logique

Opérateurs arithmétique

```
3 + 6        // => 9 : (Number)
5- 2         // => 3
3 * 3        // => 9
10 / 3       // => 3.33333
10 % 3       // => 1
```

Opérateurs logique

```
let x = 2; let y = 3;
x === y                    // => false : egalité
x !== y                    // => true : inégalité
x < y                      // => true : inférieur à
x <= y                     // => true : inférieur ou égal à
x > y                      // => false : supérieur à
x >= y                     // => false : supérieur ou égal à

"albert" === "roger"       // => false : égalité de String
(x !== y ) === true        // => true

(x === 2) && (y === 3)     /* => true : && signfie ET, les deux comparaisons sont
                                        VRAI(true) */

(x > 3) || (y < 2)         /* => false : || signifie OU, aucune des deux comparaisons
                                         n'est VRAI(true) */

!true                      // => false : ! signfie NOT, il inverse la valeur d'un Boolean
!(x === 2)                 /* => false : x est égal à 2 est true, l'inverse de true est
                                         false */
```

8- Les fonctions

```
// Les fonctions sont des bloc d'instructions réutilisables et paramétrables qui peuvent
renvoyer une valeur de retour.
function moyenne(a,b,c,d)        // La fonction nommée "moyenne" à 4 paramètres
{
      let somme = a+b+c+d;
      return somme/4;            // La fonction retourne un Number
}
```

```
let resultat = moyenne(10,11,12,15);
console.log(resultat);           // => 12 : Appel de la fonction moyenne
```

Une fonction est défini par son nom, le type de sa valeur de retour et ses paramètres.

9- Les méthodes

```
let hero = {
	nom : "Link",
	pv : 100,
	pointAttaque : 2,
	parler(message)	// Une fonction dans un objet est appelé une méthode
	{
		console.log(nom +": "+ message);
	}
};
hero.parler("A l'attaque compagnons !"); // => "Link: A l'attaque compagnons !"
```

10- Les structures conditionnelles

Si Sinon, le *if else*

```
// La condition SI SINON
let age = 23;
if(age < 18)
{
	// Si x est inférieur à y
	console.log("Tu es mineur");
}
else
{
	// Si x est supérieur ou égal à y
	console.log("Tu es majeur");
}
```

Boucle pour, le *for*

```
// La boucle POUR
let eleves = ["Mathieu","John","Billy","Joe","Rémi"]; // Création d'un tableau
for(let eleve of eleves)  // Pour chaque eleve du tableau d'eleves
{
	console.log(eleve); // J'affiche l'eleve via une variable "eleve"
}
/** Résultat
* => "Mathieu"
* => "John"
* => "Billy"
* => "Joe"
* => "Rémi"
*/
```

La boucle tant que, le *while*

```
// La boucle TANT QUE
let coef = 1;
while(coef <= 5)
{
	console.log(coef*5);
	coef++;
}
/** Résultat
* => 5
* => 10
* => 15
* => 20
* => 25
*/
```

11- Class d’objet

```
// Les classes sont des types d'objet réutilisable
class Personnage {
	constructor(nom,pv,pointAttaque) {
		this.nom = nom;		// Déclaration de 3 nouveaux attributs d'objet
		this.pv = pv;		// this signifie "lui-même"
		this.pointAttaque = pointAttaque; // Et affectation des valeurs
	}

	Attaquer(ennemie) {
		ennemie.pv = ennemie.pv - this.pointAttaque;  // this, c'est l'objet qui
                                                              // appel la méthode
	}

}

let hero = new Personnage("Link",100,10);  /* Instanciation d'un objet de la classe
                                              Personnage */
let squelette = new Personnage("Squelette",30,2);

console.log(squelette.pv) // => 30
hero.Attaquer(squelette); // La méthode Attaquer prend en paramètre l'objet squelette
console.log(squelette.pv) // => 20
```

V- Les Variables

1- Vue global

La déclaration d'une variable

La déclaration d'une variable se fait à l'aide du mot clé « *let* », en précisant l'identifiant (le nom) de la variable. A la déclaration la variable à pour type *undefined* et pour valeur *undefined*[2].

```
let identifiant ;
let prenom ;
let age;
```

La déclaration d'une constante

Une constante fonctionne comme une variable à la différence près que aucune modification de sa valeur n'est possible.

```
const pi = 3.14159265359 ;
```

La constante ne pouvant être modifiée il est obligatoire de lui affecter une valeur à sa création.

```
const pi ;   // => Uncaught SyntaxError: missing = in const declaration
             // => Erreur de syntaxe : Il manque = dans la déclaration de la constante
```

L'affectation

L'affection s'effectue grâce au opérateurs d'affectation et permet de définir la valeur et le type de la variable.

```
prenom = "Vincent";
console.log(typeof(prenom));           // => "String"
age = 23;
console.log(typeof(age));        // => "Number"
```

Notez bien que le mot clé *let* n'est pas présent à l'affectation car les variables prenom et age ont déjà été déclarées.

VI- Les types de données et le transtypages

Il existent deux catégories de types de données : les types primitifs et les objets. Les Numbers, les String (texte) et les Boolean (valeur binaire) sont des types primitifs. On retrouvent également undefined et null qui sont à la fois des types primitifs et des valeurs. Le type undefined est un ensemble de un element nommée undefined qui est en faite la valeur d'une variable vide ou d'un attribut d'objet inconnu. Le type null est également un ensemble de un element, null est une valeur que l'on donne volontairement à une variable ou en valeur de retour d'une fonction pour signifier l'absence de valeur. La différence entre undefined et null se trouve dans la provencence de la valeur : undefined est le plus souvent involontaire et provient d'une variable inexistante alors que null est volontaire et represente « 0 » sans pour antant utiliser le Number 0.

2 *undefined* désigne à la fois le type et la valeur de la variable, en effet le type *undefined* est un ensemble de 1 élément : « *undefined* ».

1- Les types primitifs et littéral

String, Texte

Un texte en JavaScript est un suite de caractère nommé String.

Une string se déclarer de trois façon :

```
"Hello World !"                         // Avec des doubles quotes
'Hello World !'                         // Avec des simples quotes

let prenom = "Massinissa";
`Hello ${prenom}.`                  // Avec des backtick, ce qui permet d'utiliser des
                                    // variables dans la string sans opérateur +
```

Les doubles quotes permettent d'utiliser des simples quotes à l'intérieur de la string, là ou les simples quotes permettent d'utiliser des double quotes à l'intérieur de la string.

```
"Je m'appelle Massinissa"
'Guy : "Odile Jacques! Quel plaisir de vous voir tous les deux !".'
```

La taille d'une string

Le nombre de caractère dans une string se recupère via la variable length, qui signifie longeur en anglais.

```
const ville = "Narbonne";
ville.length                // 8
```

Fonctions usuelles d'une string.

Les strings possèdes des fonctions qui facilite leurs manipulations , par exemples.

```
let phrase = "Salut à tous !";
phrase.toLowerCase();               // => "salut à tous !"
phrase.toUpperCase();               // => "SALUT À TOUS !"

// Split permet de découper une string en un tableau à partir d'un caratère de
séparation, le plus souvent un espace vide ou une virgule.
phrase.split(" ");                  // => ["Salut","à","tous","!"]

// match permet l'utilisation d'une regex
phrase.match("Bonjour");            // => null
phrase.match("Salut");              // => ["Salut"]
```

Voir la doc *String.prototype* sur la MDN.

Parcourir les caractères d'une string.

Une chaines de caractères peut être parcouru comme un tableau grâce à l'opérateur d'indexation.

```
let prenom = "Louis";
console.log(prenom[2]);             // => "u"
```

Comme les tableaux les *strings* sont des *itérables* et peuvent et peuvent être utilisés dans une boucle *for*.

Number

Un Number est un nombre réel.

Il se déclare comme suit :

```
let x = 4
let y = 4.09
let z = -3908.6
```

A propos de NaN

NaN signifie *Not a Number.* C'est une valeur spécial du type Number qui se produit lors ce que le résultat d'un calcul n'a pas de sens et ne produit donc *pas un nombre*. Comme lors de la soustraction de deux chaines par exemple. L'on peu utiliser la fonction *isNaN()* pour tester si une valeur est un nombre ou non, la fonction renvoi un *Boolean*.

Faire des mathématiques avec les Number, la classe Math

La classe Math contient des fonction statiques permettent d'effectuer des opérations mathématiques plus ou moins complexe parmis les plus pratiques on retrouve :

- Math.random(), qui permet de générer un réel entre 0 et 1.
- Math.round(), qui arrondi à l'entier le plus proche.
- Math.trunc(), qui retire le parti décimal d'un réel
- Math.abs, qui rend la valeur absolue d'un nombre (il retire le – d'une nombre négatif)

Retrouvez tout les détails sur la doc de la MDN.

Boolean

Un boolean est une valeur binaire soit vraie (*true*) soit fausse (*false*). On peut la déclarer comme suit :

```
let isSend = true ;
```

Même si le plus souvent on fabrique un boolean depuis un test logique.

```
let isSuperior = 8 > 10
console.log(isSuperior);                // => false
```

Le transtypage

Parfois il est neccessaire de transformer un texte en Number pour effectuer des calculs.

```
let a = "3";
let b = "5";
console.log(a+b);          //=>"35"
```

Ici on obtient la string *« 35 »* alors que l'on veut le Number *8*.
La solution est donc la fonction Number() qui permet de transtyper un texte en Number.

```
let a = Number("3");
let b = Number("5");
console.log(a+b);          //=>8
```

Cette méthode est tout particulièrement utile quand on traite une entrée utilisateur d'une balise **<input>** ou d'un **prompt()**.

Undefined

```
let produit = {
```

```
        taille: 34,
        prix : 100,
};

console.log(produit.marque);            //=> undefined l'attribut marque n'existe pas
console.log(table);                     //=> undefined la variable table n'existe pas
```

Null

Null permet de définir explicitement une variable comme vide. Il est utile dans le cas ou vous souhaitez créer un variable mais que vous n'avez pas encore de donnée à lui fournir.

```
let product = null ;
searchButton.addEventListener("click",()=>{ product = searchProduct() });
```

Ici on attend que l'utilisateur clic sur le bouton pour recherche un produit. Au chargement de la page le produit n'existe pas mais plus tard lors de l'evenement « click » le produit sera recherche puis affecter à la varaible *product*.

VII- Les opérateurs

+ operator

L'opérateur + permet l'addition de deux Number ou la concaténation de deux String.

```
console.log(3+4) ;                      // Addition
console.log("Bonjour"+" à tous");       // Concatenation
```

Il également possible d'effectuer une conversion du type avec l'opérateur +. Ici le la variable *age* est un *Number* que je souhaite concaténer avec une String. La valeur 23 sera convertir en une string conctenant le caractère "2" puis le caractère "3" pour former "23".

```
let age = 23;       // Number
console.log("Bonjour Alan, tu as "+age+" ans");      // Concatenation et conversion de type
```

L'opérateur + permet de fabriquer des String ou des Number suivant que l'on effectue une concaténation de string ou une addition de number.

- operator

L'opérateur – permet de faire des soustraction. Si il est utilisé avec des valeurs qui ne se soustrait pas l'opérateur produit la valeur NaN qui signifie « Not a Number ».

```
console.log(3-10);                      //=> -7
console.log("Salut"-"toi");             //=>NaN
```

L'opérateur - permet de fabriquer des Number.

= operator

L'opérateur = affecte l'opérande de droite à l'opérande de gauche et défini le type de la variable en fonction de l'opérande de droite. L'opérande de gauche doit être une variable ou un attribut.

```
age = 23;           // Number
prenom = "Vincent"  // String
produits = [ {prix:100,nom:"Nike"}, {prix:120,nom:"Adidas"} ];      // Tableau (Array)
```

L'opérateur = permet de fabriquer des variables de n'importe quelle types.

== operator et != operator

L'opérateur == ou «double égal » test l'egalité entre deux opérandes et renvoi un Boolean (une valeur binaire) VRAI ou FAUX. L'opérateur != permet de tester la non-égalité entre deux opérandes.

```
console.log(3 == 3);            // true
console.log(3 != 3);            // false
```

L'opérateur == et **!=** permet de fabriquer des Boolean.

++ operator

L'opérateur ++ ajoute 1 à une variable

```
// Ces deux lignes sont similaires
age++;
age = age + 1;
```

L'opérateur ++ permet de fabriquer des Number.

-- operator

L'opérateur -- soustrait 1 à une variable

```
// Ces deux lignes sont similaires
age--;
age = age – 1;
```

L'opérateur ++ permet de fabriquer des Number.

+= operator

L'opérateur += ajoute l'opérande de droite à la valeur de la variable

```
// Ces deux lignes sont similaires
age = age + 5;
age+=5;
```

L'opérateur ++ permet de fabriquer des Number.

-= operator

L'opérateur -= soustrait l'opérande de droite à la valeur de la variable

```
// Ces deux lignes sont similaires
age = age - 2;
age-=2;
```

L'opérateur ++ permet de fabriquer des Number.

*= operator

L'opérateur *= multiple l'opérande de droite à la valeur de la variable

```
// Ces deux lignes sont similaires
age = age * 5;
age*= 5;
```

L'opérateur ++ permet de fabriquer des Number.

/= operator

L’opérateur /= divise la valeur de la variable par l’opérande de droite puis affecte la nouvelle valeur.

```
// Ces deux lignes sont similaires
age = age / 4;
age/=4;
```

L'opérateur ++ permet de fabriquer des Number.

L’évaluation d’une variable

Si une variable est placée seule, sa valeur sera évaluée par JavaScript et utilisable par des opérateurs ou des fonctions.

```
let annee = 5 ;
annee               // Ici JavaScript va évaluer la variable et retourner un resultat
console.log(annee)  // => 5 : Ici Le resulat de l'évaluation est passé en paramètre
typeof(annee)       // Number
let mois = age *12; // Ici le résultat de l'evalution est utiliser dans la muliplication
                    // avant l'affectation
```

L’évaluation d’une fonction

Si une fonction renvoi une valeur, il est courant d'evaluer sa valeur de retour. La fonction typeof par exemple renvoi une String, un texte qui décrit le type de la variable.

```
let age = 23;
console.log( "La variable age est du type : " + typeof(age) ) //=>La variable age est du
type : Number
```

Le code précedent provocera le même résultat que le suivant :

```
let age = 23;
let typeDeAge = typeof(age);
console.log( "La variable age est du type : " + typeDeAge ) //=> La variable age est du
type : Number
```

La modification

Les opérateurs d’affections permettent également de modifier le contenu d’une variable existante.

```
let nom = "Massinissa";
nom = "Mathieu";

let produit = {};
produit.reference = "Nike 42";
produit.reference = "Adidas 43";
```

VIII- Les conditions

Les conditions permettent d'interpréter une suite d'instructions si un test est vrai. Les conditions sont composées d'un mot-clé, d'un bloc d'instructions et d'un test logique.

1- Le bloc d'instruction

Un bloc d'instruction est délimité par des accolades, toutes les variables déclarés dans le bloc ne sont connu que du bloc, on appel cela un contexte d'exécution.

```
1  let a = 12;
2  {
3   // Début du bloc d'instruction
4   let texte = "bonjour";
5   // Fin du bloc d'instruction
6  }
7  console.log(a)          // => 12
8  console.log(texte)      // => undefined
```

Gestion de la mémoire

Un programme à accès à deux types de mémoires :

- La RAM qui stocke les variables globales du programme et les détruits quand le programme s'arrête (pour une page web quand l'onglet est fermée).
- La PILE qui stocke les variables du bloc d'instruction et les détruits à la fin du bloc d'instructions.

Dans l'exemple précédent la variable *a* est crée en dehors de tout bloc d'instruction, *a* est donc une variable globale et sera détruite à la fin du programme. La variable *texte* cependant est crée dans un bloc d'instruction et est donc détruite à la fin du bloc (ligne 5).

2- Le test logique

Un test logique renvoi un Boolean (*true* ou *false*). Ce Boolean est le résultat de l'utilisation des opérateurs logique : >, <, <=, >=, ==, !=.

```
let age = 24 ;
age > 17 && age < 26             // => true
typeof(age > 17 && age < 26)     // => Boolean
```

3- Le mot clé *if*

If se traduit en français par « *si* ».

« Si l'age est entre 18 et 25 inclus, affiche 'Réduction tarif jeune' ».

```
if(age > 17 && age < 26){
      console.log("Réduction tarif jeune");
}
```

Tout est là pour constituer une **condition**, le **test** avec ses opérateurs logique, le **bloc d'instruction** avec ses accolades et le mot clé **if** pour définir le type de condition.

Le else – sinon

Pour exécuter un bloc d'instruction dans le cas ou l'instruction est fausse, le mot clé *else* est à coupler au mot clé *if* comme ceci :

```
if(age > 17 && age < 26){
	console.log("Réduction tarif jeune");
}
else{
	console.log("Tarif normal");
}
```

4- Les boucles conditionnels

La boucle *while* – tant que

La boucle *while* évalue le résultat du test au début du bloc d'instructions, si le test est vrai (*true)* il exécute le bloc. Une fois le bloc exécuté, il évalue à nouveau le résultat du test et ainsi de suite.

Tant que le test est vrai le bloc d'instructions sera exécuté.

Voici un programme qui affiche la table de 5 de 1 à 10 :

```
let coef = 1;
while(coef <= 10)   // évaluation du test
{
	// Si le test renvoi true
	console.log(coef*5);
	coef++;
}
// Le programme continue quand le test renvoi false
// Suite du programme ...
```

La boucle infini

Si le test d'une boucle est toujours vrai, alors on se retrouve avec une boucle infini qui va *frezzer* (geler) le programme.

```
while(true){
	// Le programme est bloqué ici
}
```

Si je reprend l'exemple pris plus haut et que je retire l'incrementation je vais prvoquer une boucle infini qui va geler mon programme y compris l'affiche de la page.

```
let coef = 1;
while(coef <= 10)   // évaluation du test
{
	// Si le test renvoi true
	console.log(coef*5);
	// Boucle infini coef est toujours inférieur à 10
}
```

La boucle *for* – Pour

La boucle *for* est une version syntaxiquement plus élégante de la boucle *while*. Elle est composée de 3 instructions qui vont s'exécuter à des moments différents de la boucle :

- L'instruction d'initialisation, s'exécute une fois avant le premier test
- L'instruction de début de bloc, s'exécute en tout premier à chaque passage de la boucle
- L'instruction de la fin du bloc, s'exécute en tout dernier à chaque passage de la boucle

```
for(initialisation ; debut de bloc ; fin de bloc){
	// instructions ...
}
```

```
for(let i = 1 ; i <=10; i++){
	console.log(i) ;
}
```

La boucle *for* ci-dessus est sensiblement la même que la boucle *while* suivante :

```
let i = 1;		// Initialisation
while(i <= 10)		// Début de bloc
{
	console.log(i);
	i++;		// Fin de bloc
}
```

La boucle *for* est donc une alternative plus élégante à la boucle *while*. Voilà pourquoi le *for* est la condition la plus utilisé en JavaScript avec le *if*.

Syntaxe alternative de la boucle for

Il existent d'autre syntaxes encore plus simple pour la boucle *for* notamment pour parcourir les éléments d'un tableau ou les attributs d'un objet.

For of

```
let eleves = ["Mathieu","Billy","Laura","Clément"];
for( eleve of eleves){
	console.log(eleve);
}
```

Cette syntaxe permet de parcourir les elements d'un tableau Lors de la boucle l'element courant est disponible via la variable eleve. Cette variable est appelée iterateur et son nom est défini par le programmeur. Ici vous auriez pu nommé la variable *eleve*, *student*, *iterateur* ou bien *it* peu importe.

For in

```
Let produit = {
	reference : "Nike air",
	prix : 99,
	categorie : "Sneakers"
}
for( attribut in produit){
	console.log(attribut);
}
```

Cette syntaxe permet de parcourir les attributs d'un objet.

IX- Les fonctions

Les fonctions sont des blocs d'instruction paramètrables et réutilisables. Comment les variables elle possèdent un identifiant et un type de valeur de retour. Une fonction peut également posséder des paramètres qui vont influencer le résultat de la fonction.

Vous connaissez déjà deux fonctions : *log* et *typeof*.

1- Syntaxe

Les paramètres d'une fonction peuvent aller de zéro à l'infini et chaque paramètre est séparé par une virgule.

Le mot-clé *return* renvoi une valeur et arrête la fonction.

```
function identifiant(parametre1, parametre2, parametreN, ...){
	// instructions ...
	return valeur_optionnel ;
}
```

2- Déclarer une fonction

```
function salut(prenom){
	return "Salut "+prenom+" !";
}

function somme(a,b){
	return a+b;
}

function moyenne(a,b,c){
	let somme = a+b+c;
	return somme/3;
}
```

3- Appeler une fonction

L'appel d'une fonction se fait via l'opérateur *()*.

```
salut("Thomas");		// => "Salut thomas !" : Evaluation de la valeur de retour de
				// la fonction
```

Comme pour les variables le résultat de l'évaluation d'une fonction peut être stocké dans une variable ou utilisé en paramètre d'une autre fonction.

```
let message = salut("Thomas");	// Affectation de la valeur de retour de la fonction
				// salut dans la variable message
console.log(message);
// ou
console.log(salut("Thomas"));
```

4- Le mot clé *return* - l'arrêt de la fonction

Le mot clé *return* met fin à l'exécution de la fonction et renvoi une valeur de retour si elle est précisée ou *undefined* si aucune valeur n'est précisé.

```
function est_majeur(age){
	if(age >= 18){
		return true;
	}
	else{
		return false;
	}
}
est_majeur(15);	// => false : 15 est inférieur à 18
```

Ici on aurait pu directement écrire :

```
function est_majeur(age){
	return age >= 18;
}
```

Omettre le *return*

Si le return est omit ou si le return ne renvoi par de valeur, la valeur de retour sera *undefined.*

```
function affiche_carre(x){
	console.log(x*x);
	// le return n'est pas précisé la valeur de retour est donc undefined
}
// equivalent à
function affiche_carre(x){
	console.log(x*x);
	return; // Le return ne renvoi rien, donc undefined
}

let retour = affiche_carre(2);		//=> "4"
console.log(retour);			//=> undefined
```

X- Les tableaux – Array

L'*array* est une variable qui contient une suite d'éléments identifiées par un index numérique et accessible via l'opérateur d'indexation *[]*.

L'*array* est une des classes d'objet les plus utilisée en JavaScript, dites-vous que si vous devez manipuler une suite de données, vous avez besoin d'un *Array*.

1- Intérêt d'un *Array*

- Contenir une liste de données
- Possède des méthodes qui facilite l'accès à ces données.
- Stocker des données provenant d'une base de données.

2- Déclaration

Syntaxe

```
let tableau = [elementUn, elementDeux, elementN, ...];

let eleves = ["Mathieu","Bob","Billy"];
let notes = [12,15,20,13,8];
```

3- Lecture des éléments

La lecture se fait via l'opérateur d'indexation, avec comme premier index 0 et comme dernier index le nombre d'élément du tableau moins un.

```
notes[0]            // => 12
notes[2]            // => 20
eleves[0]           // => "Mathieu"
eleves[2]           // => "Billy"
```

4- Ajout d'un élément

En JavaScript un *array* est une instance de la classe *Array*. Ces attributs ont des identifiants numérique d'où l'obligation d'y accéder par l'opérateur d'indexation[3].

L'ajout d'un élément se fait via la méthode *push* de la classe *Array*.

```
let eleves = ["Mathieu","Bob","Billy"];
eleves.push("Massinissa");       //  La String est ajoutée en dernier élément de l'array
```

5- Supprimer des éléments

La méthode *splice* de la classe *Array* permet de supprimer un certain nombre d'élément d'un *array* à partie d'un index de départ.

Syntaxe

```
tableau.splice(index_debut,nombreElementASupprimer);
```

3 Si l'on écrit *tableau.0* au lieu de *tableau[0]*, JavaScript va comprendre la valeur littéral *0* et non l'attribut ayant l'identifiant *0*, voilà pourquoi l'on est obligé d'accéder au élément d'un *array* via l'opérateur d'indexation.

```
let tableau = [10,11,12,13,14,15];
tableau.splice(2,3);      // Supprime 3 éléments à partir du 2 : 12,13,14
tableau                   // => [10,11,15]
```

Supprimer un seul élément

```
let tableau = [10,11,12,13,14,15];
tableau.splice(4,1);      // Supprime 1 élément à partir du 4 : 14
tableau                   // => [10,11,12,13,15]
```

XI- Fonctions usuelles des Tableaux

1- ForEach - parcourir

La fonction forEach permet de remplacer la syntaxe du for classique par une syntaxe utilisant une fonction callback, c'est à dire une fonction qui est passée en paramètre d'une autre fonction.

```
let eleves = ["louis","massi"];

eleves.forEach(function(eleve){
      console.log(eleve);
});
// ou
function showEleve(eleve){
      console.log(eleve);
}
eleves.forEach(showEleve);       // Cette syntaxe se rapproche beaucoup de l'anglais et
                                 // est très lisible.
```

Voir la doc de la MDN.

2- Filter - extraire

La fonction filter est similaire à forEach à la différence qu'elle permet de filtrer un tableau pour en extraire uniquement les elements qui valide un certain test.

Voir la doc de la MDN.

3- Map - transformer

La fonction map permet de fabriquer un tableau à partir d'un autre. Il est très utiliser en JS pour formater des données. Si vous recevez par exemple un tableau de produits pour une boutique en ligne et que vous souhaitez utilisez les données de ces produits pour fabriquer un tableau d'objet de la classe Produit.

XII- Les Objets

Le JavaScript est un langage dit « Orientée objet »ce qui signifie qu'il est possible de créer des structure de donnée appelées objet.

Un objet est composé d'attributs et de méthodes.

Un attribut c'est le nom que l'on donne à une variable contenu dans un objet.

La méthode c'est le nom que l'on donne à une fonction contenu dans un objet.

Le type *Object* est le type le plus important en JavaScript vous allez en rencontrer constamment. L'objet console est d'ailleurs du type *Object.*

```
typeof(console) ;   // Object
```

1- Le paradigme de *programmation orientée objet(POO)*

La POO rajoute une couche d'abstraction supplémentaire à la programmation classique et permet de structurer l'architecture d'un programme d'une manière similaire à la façon dont les Hommes structures leurs pensées.

2- La pensée Objet

L'Homme se représente sont environnement sous la forme d'objets, chaque objet étant défini par un nom, des caractéristiques ainsi que par les interactions possible avec cet objet.

Un produit à vendre par exemple est composé : d'un prix, d'un nom, d'une description, d'une image et du nombre de produit en stock. Toutes ses caractéristiques sont les *attributs* de l'objet *produit*.

Un produit c'est aussi des interactions : acheter, retirer, appliquer un promotion, changer le stock. C'est interactions sont autant de fonction essentiel à l'utilisation de notre produit. Ces fonctions sont appelées les *méthodes* de l'objet.

En résumé la programmation orientée objet permet d'organiser tout son code sous la forme d'objet avec lesquelles on peut interagir via des méthodes.

3- Instanciation d'un objet

La déclaration d'un objet se dit « instanciation » et se fait comme suit :

```
let mon_objet = { };      // Déclaration d'un objet vide
```

4- Déclaration des attributs d'un objet

La déclaration des attributs d'un objet se fait de deux manières : à l'instanciation ou dynamiquement après l'instanciation.

```
// Déclaration des attributs à l'instanciation
let produit = {
      nom : "Banane",
      prix : 3
} ;
// Déclaration dynamique d'un attribut
produit.stock = 200; // L'attribut stock est maintenant présent en plus de nom et prix
```

L'opérateur d'accession point '.'

L'accès au attributs et méthodes d'un objet se fait via l'opérateur point : '.'

```
console.log(produit.nom);        // => "Banane"
```

L'opérateur point permet d'accéder à l'attribut *stock* de l'objet *produit*. Vous remarquerez que vous vous servez de l'opérateur point pour accéder à la méthode *log* de l'objet console.

L'opérateur d'indexation *[]*

L'opérateur d'indexation [] permet également d'accéder au attributs et méthode d'un objet.

```
produit["nom"];
// équivalent à
produit.nom;

console.log("salut");
// équivalent à
console["log"]("salut");
```

L'opérateur d'indexation n'est pas très utilisé pour l'accès des objets sauf dans le cas ou ces objets sont des tableaux (*Array*).

5- Déclaration des méthodes d'un objet

```
let personnage = {
    nom : "Richard Coeur de Lion",
    titre : "Roi d'Angleterre",
    saluer(invite){
        console.log("Salutation " + invite + " !");
    }
}
```

La déclaration d'une méthode se fait comme pour une fonction classique à la différence que l'on omet le mot clé *function*, JavaScript comprend la présence d'un méthode par les parenthèses et les accolades.

6- Appel d'une méthode

L'appel d'une méthode se fait via l'opérateur d'accession, comme pour *console.log*.

```
personnage.saluer(« Billy») ;    // => "Salutation Billy !"
```

7- Le mot clé *this*

Le mot-clé *this* permet d'accéder à l'objet de l'intérieur.

```
let personnage = {
    nom : "Richard Coeur de Lion",
    titre : "Roi d'Angleterre",
    saluer(invite){
        console.log(this.nom+ " dit Salutation " + invite + " !");
    }
}
personnage.saluer("Billy");      // =>  "Richard Coeur de Lion dit Salutation Billy !
```

Si l'on omet le *this* JavaScript va penser que l'on parle d'une variable globale précédemment déclarée et va utiliser sa valeur ou renvoyer *undefined.*

```
let nom = "Banane";
let personnage = {
    nom : "Richard Coeur de Lion",
    titre : "Roi d'Angleterre",
    saluer(invite){
        console.log(nom+ " dit Salutation " + invite + " !");   // Oublie du this !
    }
}
personnage.saluer("Billy");  // =>  "Banane dit Salutation Billy !
```

8- Passage par valeur

Si l'on passe en paramètre d'une fonction une variable d'un type primitifs comme : *Number*, *Boolean* ou *String*, la variable est évaluée et la valeur de la variable est passée en paramètre à la fonction. Ce qui signifie que l'on ne peux pas modifier une variable passée en paramètre car à l'intérieur de la fonction c'est seulement une copie au-quelle on a accès.

```
function change_variable(a){
       a = 25;
}

let a = 10;
change_variable(a);
a             // => 10 : a est toujours égal à 10 car c'est seulement une copie de a dans
              // la mémoire PILE qui à été utilisé dans la fonction
```

9- Passage par adresse

Dans la cadre du passage d'un objet c'est un peu différent. En effet on ne peut pas passer la valeur d'un objet car un objet est composé de donnée trop complexe. Alors lorsque l'on passe un objet en paramètre d'une fonction c'est enfaîte son adresse dans la mémoire RAM que l'on passe. Ce qui nous permet de modifier ses attributs à l'intérieur d'une fonction.

```
function change_variable(objet){
    objet.a = 25;
}

let objet = { a : 10 };
change_variable(objet);
objet.a;      // => 25 : l'attribut a à bien été modifier à l'intérieur d'une fonction
```

Voyez l'adresse de l'objet comme un raccourcis vers l'espace mémoire où il se trouve, un peu comme le raccourcis d'un logiciel présent sur le bureau de votre ordinateur.

XIII- Les classes – créer ses propres type de variable

Une classe est un type d'objet permettent d'instancier plusieurs objets à partir d'un même patron de conception appelé *class*. Les classes sont crées par les développeurs pour instancier des objets.

1- Créer un classe

Un classe possède une méthode appelé « *construtor* » cette méthode est appelée à l'instanciation de l'objet et permet de définir les attributs de l'objet.

Syntaxe

```
class identifiant{
	constructor(attribut1,attribut2,attributN,...){
		this.attribut1 = attribut1;
		this.attribut2 = attribut2;
		this.attributN = attributN;
		...
	}

	methodN(){
		...
	},
	...
}

class Produit{
	constructor(nom,prix,stock){
		this.nom = nom;
		this.prix = prix;
		this.stock = stock;
	}
	AfficherProduit(){
		console.log(`${this.nom}, ${this.stock} en stock - ${this.prix}€/u`[4]);
	}
}
```

Ici les attributs *nom*, *prix* et *stock* sont définis dans le *constructor.*

2- Instanciation d'un objet à partir d'une classe

L'opérateur *new* permet d'instancier un objet à partir d'une classe.

```
let produit = new Produit("Banane",3,120);    // Instance de la classe Produit
produit.AfficherProduit();                    // => Banane, 120 en stock – 3€/u
```

Astuce: Pour connaître la classe d'un objet il faut lire la valeur de l'attribut *Object.constructor.name*.

```
console.constructor.name         // Object
produit.constructor.name         // Produit
```

3- Précision sur la classe *Object*

La classe *Object* est la classe mère à partir duquel toutes les autres classe hérites. Les attributs et méthodes disponibles sur la classe *Object* sont donc disponibles sur une classe comme *Produit* car « *Produit* hérite de *Object* ».

4 Ceci est une *structured string*, elle évalue l'expression placé entre *${}*. Elle est très utilisée pour éviter l'utilisation intempestive de l'opérateur +.

XIV- Développement JavaScript *Front-end*

Comme dit précédemment c'est l'environnement d'exécution (Runtime) qui permet à JavaScript d'accéder au entrées et sorties de l'utilisateur. L'objet console est par exemple inexistant du langage JavaScript et c'est le navigateur qui va vous créer cette objet et ainsi vous ouvrir l'accès à la console du navigateur.

Dans ce chapitre nous allons aborder l'organisation des différents outils (API) offerts par le navigateur pour vous permettre de créer des sites web dynamiques avec JavaScript.

Dans un premier temps faisons un inventaire des APIs disponible dans notre navigateur.

1- Qu'est ce qu'une API - Application Programming Interface

Une api est un programme qui fait l'interface entre deux « programmes », par exemple l'objet console est une interface entre le script écrit par le programmeur et la console du navigateur. L'objet console est donc une API.

2- La programmation évènementiel

Le JavaScript est un langage événementiel, ce qui signifie que vous allez pouvoir attacher des fonctions à des événements.

Par exemple : modifier le CSS et passer le site en « *dark mode* » au clic de la souris sur un bouton. Vous écrirez une fonction qui va changer les couleurs du site puis vous attacherez cette fonction à *l'event* « *click* » d'un bouton HTML comme ceci :

```
// Recupère le bouton HTML grace au nom de sa classe HTML
const boutonHTML = document.querySelector(".btn_dark_mode") ;

// Attache la fonction darkMode à l'event click du bouton
boutonHTML.addEventListener("click",darkMode) ;

// Déclaration de la fonction darkMode
function darkMode(){
	// Passe les elements HTML en dark mode en modifiant leurs CSS
}
```

Voir la documentation de la MDN sur les événement en JS :
https://developer.mozilla.org/fr/docs/Learn/JavaScript/Building_blocks/Events

3- L'objet *Window*

window est l'objet qui «encapsule » tout les autres objets manipulables en JavaScript *front-end*. Par exemple l'objet *console* est enfaîte un attribut de l'objet *window*.

```
console.log("Hello world !");
// équivaux à
window.console.log("Hello world !");
```

Pour améliorer la lisibilité du code il n'est pas obligatoire de rajouter l'objet *window* avant l'appel ou l'accès à ses attributs et méthodes.

L'objet w*indow* est appelé l'objet globale.

4- L'objet window.console

L'objet *window.console* ou *console* permet, grâce à ces méthodes, d'accéder à la console du navigateur.

> *Voir la documentation de la MDN sur l'objet console :*
> *https://developer.mozilla.org/fr/docs/Web/API/console*

5- L'objet window.document – le DOM

L'objet *window.document* permet d'accéder au document HTML, concrètement la page HTML. C'est l'objet le plus important à connaître en JavaScript, il est appelé le **DOM** pour *Document Object Model*.

Le DOM est la représentation objet du document HTML c'est dans cet objet et dans ses enfants que vous pourrez modifier le contenu de votre page ou encore réagir à des événements comme le clic, la molette ou la soumission d'un formulaire.

```
const titre = document.querySelector("h1") ;
titre.innerText = "Nouveau titre de la page";
```

> *Voir la documentation de la MDN sur le DOM:*
> *https://developer.mozilla.org/fr/docs/Web/API/Document_Object_Model/Introduction*

6- L'objet *window.localStorage* & l'objet *window.sessionStorage*

Les objets *localStorage* et *sessionStorage* permettent de stocker des données dans le navigateur du client à la manière des «cookies ». Ces objets bénéficie de jeu de méthodes qui permettent le stockage d'informations très simplement.

Si vous avez besoin de stocker des données après le passage de votre utilisateur, comme un panier d'achat ou des préférences d'utilisation le *localStorage* est fait pour vous.

Si vous avez besoin de stocker des données pendant la durée de la session de votre utilisateur, le *sessionStorage* est fait pour vous.

> *Voir la documentation de la MDN sur les WebStorage:*
> *https://developer.mozilla.org/fr/docs/Web/API/Web_Storage_API*

7- L'objet *window.fetch* – le client HTTP

L'objet fetch est un client HTTP. Il permet de faire des requêtes HTTP à un serveur et pour y récupérer des données.

Entre autre, si vous avez besoin d'afficher des données provenant d'un serveur après le chargement de la page comme des recommandation dans la barre de recherche ou des produits e-commerce mis à jours en temps réel en fonction de filtres de recherche. L'API fetch est là pour vous.

Attention : je recommande une connaissance minimale de l'asynchrone, des Promises et la lecture de la RFC HTTP pour apprendre *fetch* dans les meilleurs conditions.

> *Voir la documentation de la MDN sur les API fetch:*
> *https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch*