

*@Utopios Consulting*



# Javascript

## La Logique du Web

*@Utopios Consulting*

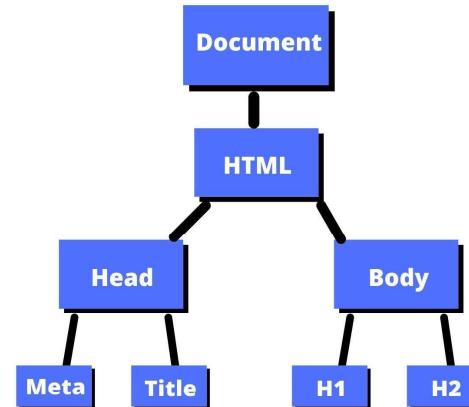
# Introduction au Javascript

# Qu'est-ce que le Javascript ?

Alors que l'HTML crée la structure de notre page web (ses fondations) et que le CSS permet d'avoir un style personnalisé (la décoration intérieure et l'arrangement des pièces), le Javascript permet à notre site de posséder une intéactivité et des fonctionnalités (la lumière, la plomberie, etc...).

Grâce au Javascript, il est possible de modifier facilement la composition de notre page web en changeant par exemple nos différents **<span>** en fonction des différentes entrées utilisateurs. De plus, grâce au Javascript (qu'on nomme communément JS pour réduire l'usure de nos doigts) il est possible de déclencher des évènements lors du clic d'un bouton, de l'appui d'une touche ou de toute autre interaction entre l'utilisateur et notre site internet.

Le JS se base sur ce que l'on appelle le DOM (Document Objet Model) pour fonctionner. Lorsque l'on crée un site internet avec des balises HTML, ces balises peuvent contenir d'autres balises – leurs enfants, et créer ainsi une hiérarchie d'objet HTML que le Javascript peut consulter et parcourir à loisir.



# A quoi ça sert ?

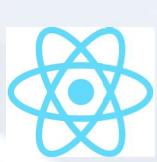
*Utopios Consulting*

Le Javascript est un langage utilisé dans plusieurs domaines, et qui peut être couplé à ce que l'on appelle des « **frameworks** » dans le but d'élargir ses possibilités rapidement. Ces frameworks ont été développées par des pionniers du JS, et permettent de faire sortir le Javascript de son but principal – le web.

Désormais, via l'utilisation du Javascript et de ses frameworks, il est possible de faire des applications :

- **Web** : Via l'utilisation des frameworks React, Vue.js ou Angular
- **Serveur** : Via l'utilisation de Node.js
- **Mobile** : Via l'utilisation de Ionic ou de React Native
- **Desktop** : Via l'utilisation d'Electron

Le Javascript est donc l'un des langages les plus populaires de part sa facilité d'apprentissage et de ses possibilités. En apprenant le Javascript, vous pourrez, à terme, développer plusieurs projets dans des environnements divers et variés.



React Native

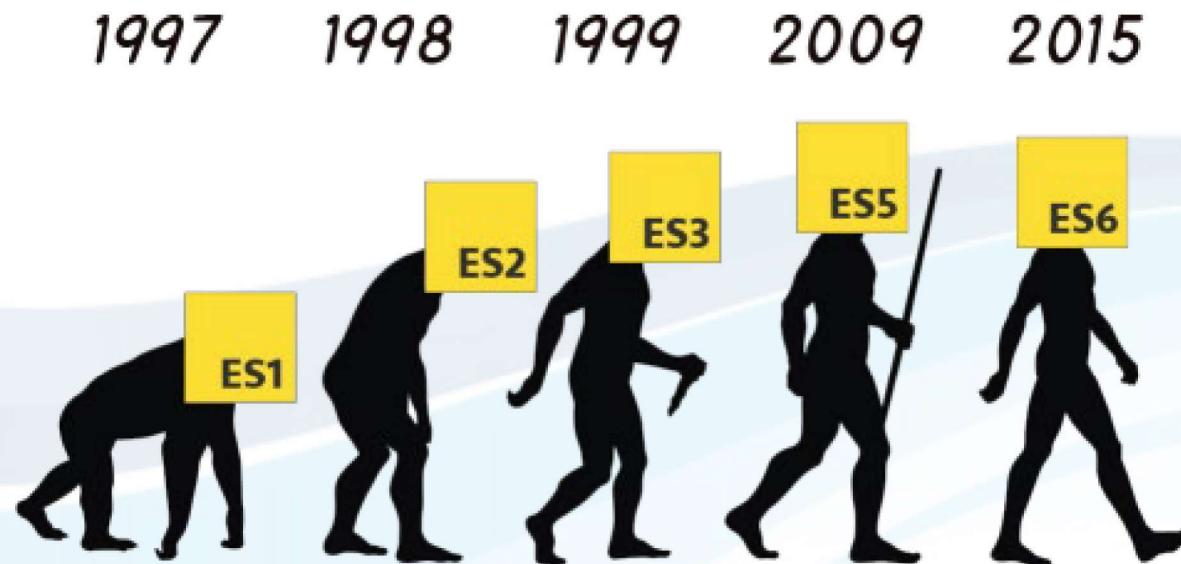


# Un peu d'Histoire...

*OpenSpirits Consulting*

Le Javascript a subit beaucoup d'évolution au court de sa vie (qui a débutée en 1996). Son nom réel est en fait ECMAScript, mais il fut nommé Javascript en raison de la prédominance du langage Java lors de son lancement (par soucis de marketing).

Une fois lancé, il a évolué en plusieurs versions, donnant par exemple l'ES1, l'ES2, etc... Pour en venir en 2015 à l'ES6 (ECMAScript 6). A partir de cette version, de nombreuses fonctionnalités supplémentaires ont fait leur apparition. Pour cette raison, il est désormais convenu d'appeler le Javascript post-ES6 le Javascript « moderne ».



# Notre premier programme

Lorsque l'on commence la programmation, il est très fréquent que notre premier exercice soit de créer un programme permettant d'écrire le texte « Hello World », en souvenir des débuts du développement informatique.

Pour réaliser la chose en Javascript, il suffit simplement d'ouvrir l'inspecteur de code de notre navigateur préféré puis de passer dans la section « Console ». Une fois fait, la ligne de code qu'il nous faut entrer est :

```
> console.log("Hello World")
Hello World
```

En effet, la console de notre navigateur possède ce que l'on appelle une « fonction » (nous reviendrons plus tard sur les fonctions). Cette fonction, nommée `log()`, prend comme paramètre une chaîne de caractère (représenté en langage informatique par des caractères délimités par des guillemets). Le but de cette fonction est simplement de permettre de loguer (dans un soucis principalement de suivi) les informations que l'on demande. Ici, on demande donc à notre console de loguer la suite de mots « Hello World », et c'est ce qu'elle fait à la ligne du dessous.

Bravo, nous avons ainsi notre premier programme !

# Où placer nos scripts ?

A la manière du style de notre site, il est possible d'inclure du Javascript dans notre page HTML de plusieurs façon.

La première méthode consiste en l'utilisation de la balise `<script></script>` qui possédera en son sein notre code Javascript, de façon à obtenir un résultat similaire à l'exemple ci-dessous :

```
</body>
<script>
    console.log("Hello World");
</script>
</html>
```

Une autre façon plus pratique d'inclure du Javascript dans notre site est de le placer dans un fichier portant l'extension `.js` qui sera lié à notre HTML par utilisation de l'attribut `src` de notre balise `<script>`.

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

Dans les deux cas, il sera possible d'écrire autant de lignes de Javascript que l'on veut, mais bien évidemment, la seconde méthode permet de séparer les parties de notre site selon leur thématiques :

(HTML => index.html / CSS => style.css / JS => main.js)

*@Utopios Consulting*

# **Les Valeurs et Variables**

# Qu'est-ce qu'une variable ?

Une variable en informatique est tout simplement un conteneur qui permet de stocker une ou plusieurs valeurs. On pourrait assimiler une variable à une boîte qui serait remplie à la volée de différentes choses.

En langage informatique, on peut différencier plusieurs types de langages informatiques. Le Javascript fait partie des langages que l'on désigne du terme de « faiblement typé ». Ce type de langage se définit par la façon dont on définit une variable. Dans les langages typés, il nous faut définir le type de la variable (par exemple un nombre ou un texte) et ce type ne peut alors plus changer (il n'est pas possible de placer du texte dans une variable de type nombre).

Dans le cas d'un langage faiblement typé comme le Javascript, on déclare une variable simplement en lui donnant un nom, comme dans l'exemple ci-dessous :

```
1 var maVariable = "Contenu de ma variable";
2 var monNombre = 42;
3 var maVerification = true;
4
```

La déclaration de variables en Javascript se fait donc par l'utilisation du mot-clé « **var** » , « **let** » ou « **const** » suivit du nom de notre variable. Ce nom doit être commencé par une lettre, ne pas posséder d'accent et suivre la convention de nommage du « **camel casing** ». L'affectation de variable se fait quant à elle par l'utilisation de l'opérateur « **=** ».

Comme son nom l'indique, il s'agit d'écrire en collant nos mots et en ajoutant une majuscule à chaque nouveau mot.

Ex : **onEcritDoncDeLaSorte**

# Var, Let et Const

*Topios Consulting*

Pour déclarer une variable, il existe donc trois mot-clés qu'il faut placer avant notre nom de variable. Cette instruction de code s'appelle la « déclaration » d'une variable. Malgré tout, les trois mots-clés n'ont pas le même effet sur la façon dont notre variable fonctionnera.

Le mot clé « **var** » est l'ancien mot-clé de déclaration d'une variable. Il permet la déclaration d'une variable à n'importe quel endroit du code, même APRES son utilisation ou son affectation.

Le mot-clé « **let** » est désormais le mot-clé convenu pour la déclaration d'une variable. Contrairement à var, il protège de l'affectation ou de l'utilisation d'une variable AVANT sa déclaration.

Le mot-clé « **const** » permet la création d'une constante. Une constante est une variable qui ne peut pas changer (elle est immuable, contrairement à une variable classique qui est dite mutable ou muable).

A côté de la déclaration, il faut avoir donc recours à l'affectation pour pouvoir réellement profiter des avantages d'une variable. Pour affecter une variable, il faut avoir recours à la syntaxe **nomVariable = valeur**. Il est également possible d'affecter une variable en même temps que de la déclarer (c'est d'ailleurs nécessaire à la création d'une constante) comme dans l'exemple ci-dessous :

```
let maVariable = "Contenu de ma variable";
let monNombre = 42;
```

# Les Types de Données

En Javascript, une valeur est soit un objet, soit une valeur primitive. Dans un premier temps, nous allons nous attaquer aux différents types primitifs.

- **Les Nombres** : Ce sont des nombres à valeur flottante. En effet, en Javascript, les nombres sont comptés par défaut comme des nombres possédant une virgule (qu'elle soit affichée ou non)
- **Les Strings** : Les Chaines de Caractères permettent de stocker des séquences de lettres, de mots ou de lignes de texte.
- **Les Booléens** : Ces valeurs sont de deux types : Vrai ou Faux. Ils sont utilisés dans le cas de prises de décision au niveau de notre code (par exemple dans des structures conditionnelles)
- **Undefined** : En Javascript, une variable ne possédant pas encore de valeur est définie comme « non définie ».
- **Null** : Cette valeur est également synonyme d'absence de valeur, mais est utilisée dans d'autres cas.
- **Symbol (ES2015)** : Concerne les valeurs unique qui ne peuvent changer.
- **BigInt (ES2020)** : Concernent les nombres trop grands pour être contenus dans une valeur de type « nombre » classique.

Suite au fait que le Javascript est un langage faiblement typé, le typage de nos valeurs se fait à la volée lors de leur affectation. On parle ainsi également de typage dynamique. A cause de cela, il est important de bien faire attention à ce que l'on place dans nos variables et de bien les nommer afin que notre code soit clair et facilement compréhensible par autrui. Il ne faudrait pas que l'on donne comme nom « prénom » à une variable destinée à contenir l'âge d'une personne par exemple.

Il est également possible de connaître le type d'une variable via l'utilisation de **typeof**, comme dans l'exemple ci-dessous :

# Les Opérateurs Arithmétiques

Pour commencer notre apprentissage des opérateurs, nous allons pouvoir commencer par les opérateurs arithmétiques (ceux utilisés en mathématiques). En Javascript, les opérateurs arithmétiques comprennent :

- L'addition avec l'opérateur `+`
- La soustraction avec l'opérateur `-`
- La multiplication avec l'opérateur `*`
- La division avec l'opérateur avec l'opérateur `/`
- Le modulo avec l'opérateur `%`
- La puissance avec l'opérateur `**`

A ces opérateurs s'ajoutent les opérateurs d'incrémentation `(++)` et de décrémentation `(--)`, qui se présentent de la sorte :

```
main.js > ...
1 let myValue = 100;
2 console.log(myValue); // 100
3 myValue--;
4 console.log(myValue); // 99
5 myValue++;
6 console.log(myValue); // 100
7 |
```

Des opérateurs de comparaison sont également disponibles sous la forme de `>`, `<`, `>=`, `<=`, `!=`, `==` et `====`. Ces opérateurs permettent de tester les rapports de supériorité, d'infériorité, d'égalité ou de différence de deux valeurs comme dans l'exemple ci-dessous :

```
let nbA = 10, nbB = 25;

console.log(nbA != nbB); // true
console.log(nbA == nbB); // false
console.log(nbA >= nbB); // false
console.log(nbA < nbB); // true
```

# Les Chaines de Caractères

La manipulation du texte est un composant essentiel de la programmation. Lorsque l'on travaille avec des « **strings** », il nous est possible de les concaténer (les assembler en phrases) via l'utilisation de l'opérateur `+` vu précédemment.

Il est possible de concaténer des variables d'autres types qu'une string à cette phrase, via ce que l'on appelle le « **type conversion** ». Ce processus a lieu automatiquement, et nous permet de facilement adjoindre des nombres à notre chaîne de caractère par exemple.

Depuis ES6, il existe également ce que l'on appelle les « **template literals** » qui permettent une syntaxe plus aisée des phrases composées de plusieurs variables :

```
main.js > ...
1 let prenom = "Sarah";
2 let nom = "MARTIN";
3 let age = 25;
4
5 let maPhrase = prenom + " " + nom + " a bientôt " + age + " ans !"; // Pre ES6
6 let maPhrase2 = `${prenom} ${nom} a bientôt ${age} ans !`; // Post ES6
```

Pour réaliser cette syntaxe, on se sert de « **backticks** » (Alt Gr + 7) pour créer une string dans laquelle on peut injecter nos variables sous la forme de ``${variableName}``. Les template literals permettent également la création de chaînes de caractères multi-lignes sans avoir à utiliser l'ancienne méthode (qui fonctionnait via l'ajout d'un caractère de retour à la ligne => `\n`),

*@Utopios Consulting*

# **Les Structures de Contrôle**

# Les Structures Conditionnelles

Dans un programme, il est très fréquent que l'on ait besoin d'exécuter une partie du code seulement lors d'un choix d'utilisateur ou si une variable possède une certaine valeur. Pour ce faire, on a recourt à des structures conditionnelles.

Pour réaliser de telles structures, il nous faut utiliser des blocs de type **IF / ELSE IF / ELSE**

- **if (condition) {...}** qui permet de spécifier qui faire au programme en cas de condition validée
- **else if (condition) {...}** qui permettent de donner des actions différentes en cas de condition différentes de la première mais également validée
- **else {...}** permettant au programme de faire une série d'instruction si aucune des conditions préalable n'est remplie

La condition se trouvant dans les deux premières structures doit renvoyer une valeur de type booléenne (Vrai ou Faux). Il est ainsi possible d'avoir autant de structure **ELSE IF** que l'on veut, mais on ne peut avoir qu'une seule structure **IF**. De plus, la structure **ELSE** est optionnelle (mais est chaudement recommandée dans la plupart des cas).

On peut également réaliser une structure conditionnelle en se servant d'une structure de type **SWITCH**

- **switch (variable) {...}** sert à lancer la construction d'une structure conditionnelle. La variable sera celle évaluée par les **cases**
- **case <condition>:** Permet de spécifier quelles instructions faire en cas de réalisation de la condition
- **break;** Permet de stopper le bloc de type **case** pour passer à un autre bloc **case** ou sortir du bloc **switch**
- **default:** Permet de réaliser une série d'instruction par défaut si aucun des blocs **case** n'a de condition vraie

# Particularités du Javascript

En utilisant le Javascript, il faut faire attention à certains de ces aspects :

- **Type Coercion** : En Javascript, le langage va chercher à transformer les valeurs en des types différents en fonction de ce que l'on demande. Par exemple, l'addition d'une string et d'un number cause la concaténation de deux strings, alors que la soustractions de deux string cause la soustraction de deux nombres. On obtient donc des fois des résultats improbables et il faut faire très attention à ce que l'on fait de nos variables
- **Truthy / Falsy Values** : En JS, il existe des valeurs qui sont estimées par défaut à Faux ou à Vrai. Par exemple, undefined, un objet vide le nombre 0 ou une string vide sera considérée par le Javascript comme une valeur fausse. A contrario, une variable contenant un chiffre, même négatif, sera considérée comme vraie.
- **L'Egalité == / ===** : Ces deux opérateurs servent à vérifier l'égalité entre deux variables, mais possèdent une subtilité. L'opérateur == permet de vérifier l'égalité des variables en prenant en compte le type coercion, alors que l'opérateur === vérifie également l'égalité du type de la variable (Par exemple, en usant de l'opérateur ===, la chaîne de caractère 2 ne sera pas égale au chiffre 2)
- **Le Mode Strict** : En ajoutant la chaîne de caractère « use strict » en tout début d'un script Javascript, on peut activer une série de vérification dans le but d'obtenir un code plus sécurisé. Ce mode nous empêche de faire des erreurs et permet de voir des erreurs plus facilement dans la console au lieu de stopper l'exécution du programme.

# La Logique Booléenne

La logique booléenne est la thématique de la programmation permettant de combiner plusieurs conditions lors de l'évaluation d'un booléen. On peut ainsi créer des condition du type « Sarah possède une carte de bibliothèque ET n'a pas de livre en retard », « John est végétarien OU végétalien ET il a au dessus de 21 ans ».

La logique booléenne s'attaque ainsi aux mot-clés ET, OU et XOR (OU exclusif) pour créer ce que l'on appelle des tables de vérités. En Javascript, l'utilisation de la logique booléenne se fait via l'utilisation des opérateurs **&&**, **||** et **!**:

- **ET (&&)** : Si les deux expression sont vraies, alors la combinaison sera vraie
- **OU (||)** : Si l'une des deux expression est vraie, alors la combinaison sera vraie
- **NOT (!)** : Si l'expression est vraie, elle deviendra fausse, et vis-versa

| Table de vérité de ET |   |        |
|-----------------------|---|--------|
| a                     | b | a ET b |
| 0                     | 0 | 0      |
| 0                     | 1 | 0      |
| 1                     | 0 | 0      |
| 1                     | 1 | 1      |

| Table de vérité de OU |   |        |
|-----------------------|---|--------|
| a                     | b | a OU b |
| 0                     | 0 | 0      |
| 0                     | 1 | 1      |
| 1                     | 0 | 1      |
| 1                     | 1 | 1      |

```
// Logical AND operator
true && true; // true
true && false; // false
false && true; // false
false && false; // false

// Logical OR operator
true || true; // true
true || false; // true
false || true; // true
false || false; // false
```

# Les Structures Itératives

En plus de pouvoir faire choisir notre utilisateur dans le but de lui proposer une section de notre programme ou l'autre via les structures conditionnelles, il nous faut pouvoir lui proposer une répétition afin que le programme ne nous demande pas à nous développeur de coder plusieurs fois la même chose. C'est l'objectif des structures itératives. Il en existe de plusieurs types, chacune correspondant à son équivalent en algorithmie classique :

- **for (...){...}** : Cette structure permet de répéter les instructions un certain nombre de fois déterminé à l'avance. Par exemple, on peut répéter 10 fois la même série d'instruction avec une boucle for se servant d'une variable (que l'on nomme par convention « **i** » qui sera incrémentée à chaque fin du bloc d'instruction et servant de bloquant à la répétition (si l'on dépasse la valeur fixée, alors la boucle se brise) :

```
for (let i = 0; i < 10; i++) {  
    console.log("Je me répète");  
}
```

- **while (condition){...}** : Via l'utilisation de cette structure itérative, on peut répéter potentiellement à l'infini une série d'instruction (Attention donc aux boucles infinies provenant d'un soucis de logique métier). Pour ces boucles, une condition sera évaluée à chaque fois que l'on atteint le début du bloc while et la fin du bloc ramènera au début. Si la condition est vraie, alors le bloc sera exécuté, sinon, il sera ignoré et le programme passera à la suite :

```
const isLost = true;  
  
// Attention aux boucles infinies de ce type !  
while (isLost) {  
    console.log("Je suis perdu pour toujours !");  
}
```

*@Utopios Consulting*

# **Les Structures de Données**

# Les Tableaux

@Utopios Consulting

Les tableaux servent en programmation à stocker plusieurs valeurs en une seule variable (Une variable classique pourrait s'apparenter à une boite, et un tableau à une commode possédant plusieurs tiroirs). Via l'utilisation d'un tableau, on peut créer aisément des structures de données plus ou moins complexes. Il est intéressant de savoir qu'un tableau peut également contenir un tableau.

Pour créer un tableau, on peut le faire lors de sa déclaration en lui affectant plusieurs valeurs de la sorte :

```
let monTableau = ["A", "B", "C"];

console.log(monTableau);
```

Pour parcourir le tableau, il faut passer par une notation entre crochets et se servir de l'index de l'élément que l'on cherche. Les tableaux commencent avec un index de valeur 0, ce qui fait que si l'on veut obtenir le second élément, il nous fait procéder de la sorte :

```
let monTableau = ["A", "B", "C"];

console.log(monTableau); // ["A", "B", "C"]
console.log(monTableau[1]); // B
```

Pour obtenir la taille de notre tableau (savoir combien d'élément il contient), on peut se servir de sa propriété **.length** de la sorte :

```
console.log(monTableau[1]); // B
console.log(monTableau.length); // 3
```

# Les méthodes de bases d'un Array

Pour manipuler un tableau, il faut généralement avoir recours à ses méthodes (des fonctionnalités liées à sa classe). Pour se faire, nous avons besoin d'utiliser une syntaxe du type **nomVariable.nomMethode(params)**. Plusieurs méthodes existent, dont voici une liste non exhaustive :

- **.push(<valeur>)** : Sert à ajouter un élément à la fin du tableau
- **.unshift(<valeur>)** : Sert à ajouter un élément au début du tableau
- **.pop()** : Sert à retirer le dernier élément du tableau
- **.pop(<valeur>)** : Sert à retirer la première valeur trouvée à partir de la fin du tableau
- **.shift()** : Sert à retirer le premier élément du tableau
- **.shift(<valeur>)** : Sert à retirer la première valeur trouvée à partir du début du tableau
- **.indexOf(<valeur>)** : Permet d'obtenir l'index d'une valeur dans le tableau
- **.includes(<valeur>)** : Permet de vérifier que le tableau contient une valeur

# Les Objets

@Utopios Consulting

Un objet, à son niveau le plus basique de compréhension, n'est ni plus ni moins qu'un contenant permettant d'avoir une accumulation de valeurs que l'on peut nommer. Il fonctionne de la même façon qu'un tableau à cela près que la syntaxe de son affectation et ses méthodes diffèrent.

```
let monObjet = {  
    prenom: "John",  
    nom: 'MARTIN',  
    age: 28,  
    estDiplome: true,  
    chiens: [  
        "Albert",  
        "Bernie",  
        "Chloée"  
    ]  
}
```

Pour naviguer dans un objet, il existe deux notations possibles :

- La notation des crochets : Tout comme pour les tableaux, on place entre crochets ce que l'on veut atteindre (ici le nom de la propriété et non son index)
- La notation des points : On place le nom de la propriété après la nom de l'objet

Ainsi, les deux notation suivantes amènent au même résultat :

```
console.log(monObjet.age); // 28  
console.log(monObjet['age']); // 28
```

*@Utopios Consulting*

# **Les Fonctions**