Javascript

La Logique du Web

Antoine Dieudonné

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

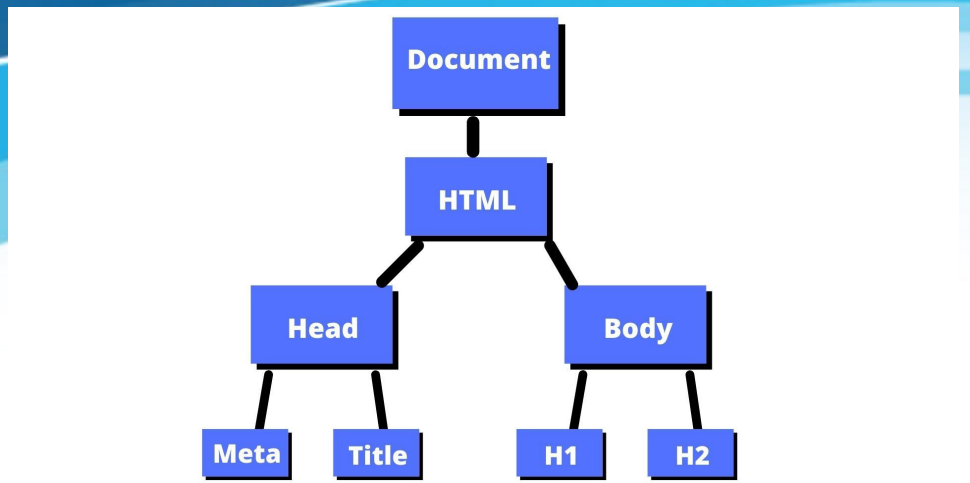
Introduction au Javascript

Qu'est-ce que le Javascript ?

Alors que l'HTML créé 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 intéractivité 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 **** 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 créé 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.



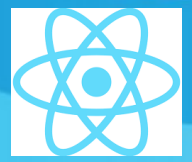
A quoi ça sert ?

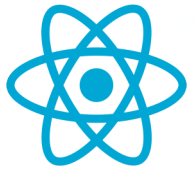
Le Javascript est un langage utilisé dans plusieurs domaine, et qui peut être couple à ce que l'on appelle des « **framework** » dans le but d'élargir ses possibilités rapidement. Ces frameworks ont été développées par des pionniers du JS, et permettent de faire sortir le Javascript de son but principal – 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 langage les plus populaire de part sa facilité d'apprentissage et de part ses possible. En apprenant le Javascript, vous pourrez, à terme, développer plusieurs projets dans des environnements divers et variés.

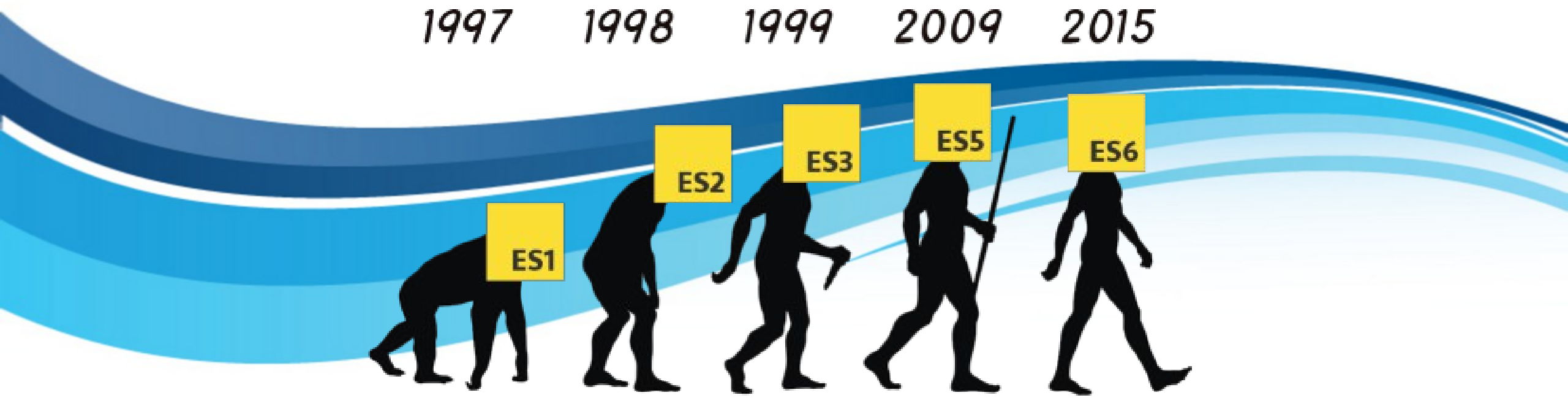


Un peu d'Histoire...

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 chaine 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)

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 boite 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 défini par la façon dont on défini 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

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 trops 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 connaitre 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 > ...

```
let myValue = 100;
console.log(myValue); // 100
myValue--;
console.log(myValue); // 99
myValue++;
console.log(myValue); // 100
```

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 chaine de caractère par exemple.

Depuis ES6, il existe également ce que l'on appelle les « **template litterals** » 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 litterals permettent également la création de chaines 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**),

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 tranformer 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 chaine de caractère 2 ne sera pas égale au chiffre 2)
- **Le Mode Strict :** En ajoutant la chaine 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 structure 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 atteins 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 !");
}
```

Les Structures de Données

Les Tableaux

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 structure 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.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).** Plussieurs 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

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
```

Les Fonctions

Qu'est-ce qu'une fonction ?

Une fonction sert à éviter la répétition du code. En effet, via l'utilisation de ces fonctions, on peut tout simplement écrire que l'on veut utiliser la fonction XXX pour réaliser toute une série d'instruction au lieu de devoir récrire plusieurs fois nos instructions. Pour cela, il nous suffit de déclarer une fonction et de la remplir de nos instructions, puis de l'appeler.

Pour créer une fonction, il nous suffit d'utiliser le mot-clé « function » suivit de notre nom de fonction. Puis viennent entre parenthèse les potentiels paramètres de notre fonction (une fonction n'est pas obligée d'avoir des paramètres) et les instructions de la fonction placées entre deux accolades. Par exemple :

```
function maPremiereFonction () {
    console.log("Je suis dans la fonction");
}

maPremiereFonction();
```

L'appel d'une fonction se fait de son côté simplement par l'écriture du nom de notre fonction et de l'ajout de parenthèses (remplies ou non en fonction des paramètres que l'on veut passer à la fonction).

Les fonction paramétrées

Notre fonction n'a pas uniquement pour but de répéter toujours la même série d'instruction, mais peut créer des variantes de ces instructions en fonction de paramètres. Pour ajouter un paramètre, il suffit de l'écrire entre les parenthèses lors de la déclaration de notre fonction. Son utilisation au sein de la fonction n'est pas obligatoire, mais elle est grandement souhaitée (Sinon, pourquoi donc avoir voulu ajouter un paramètre en premier lieu ?).

On peut par exemple faire une fonction servant à donner le caractère majeur ou non d'une personne via passage de son année de naissance en paramètre de la sorte :

```
function testMajority (birthYear) {
    if ((2022 - birthYear) >= 18) {
        console.log("La personne est majeure !");
    }
    else {
        console.log("La personne n'est pas majeure...");
    }
}

testMajority(1987);
testMajority(2009);
```

Retourner une valeur

De plus, les fonctions ont généralement pour but de retourner une valeur. Ce genre de fonction est très souvent utilisé dans le but de remplir des variables lors de leur déclaration afin de leur affecter une valeur immédiatement. Par exemple, notre fonction précédente pourrait, en plus de dire si la personne est majeur ou non, retourner l'âge de la personne, que l'on pourrait ainsi stocker dans une variable dédiée. Pour retourner une valeur, il faut que la fonction possède en son sein une instruction commençant par le mot-clé « **return** ». Lors du retour d'une valeur par une fonction, cette fonction se stoppe, et tout le code situé après le retour ne sera pas exécuté, il faut donc faire attention à où se trouve ce fameux retour.

```
function testMajority (birthYear) {
    let age = (2022 - birthYear);

    if (age >= 18) {
        console.log("La personne est majeure !");
    }
    else {
        console.log("La personne n'est pas majeure...");
    }

    return age;
}

const firstPersonAge = testMajority(1987);
console.log(firstPersonAge); // 35
const secondPersonAge = testMajority(2009);
console.log(secondPersonAge); // 13
```

Function Expression vs Declaration

Il est possible de créer des fonction de deux manières principales. La méthode que nous avons vu jusque maintenant amène au processus de « function declaration ». A côté de cela existe la mécanique des « fonction expressions », qui se présentent de la sorte :

```
const testMajority = function (birthYear) {
    let age = (2022 - birthYear);

    if (age >= 18) {
        console.log("La personne est majeure !");
    }
    else {
        console.log("La personne n'est pas majeure...");
    }

    return age;
}

const firstPersonAge = testMajority(1987);
console.log(firstPersonAge); // 35
const secondPersonAge = testMajority(2009);
console.log(secondPersonAge); // 13
```

Ce processus est très utilisé lorsqu'il est combiné à l'utilisation de fonction dites « Arrow Function » (fonctions fléchées). Par exemple, une fonction fléchée permet de raccourcir facilement notre code en déclarant en une ligne une petite fonction réalisant un calcul que l'on souhaiterai répéter dans notre code.

Les fonctions fléchées

Une fonction fléchée est une autre façon de définir une fonction en programmation. Dans certains langages de programmations, ces fonctions sont appelées « **fonction lamba** ». En Javascript, on parle communément de « **fat arrow function** » de par leur utilisation du signe égale afin de tracer une flèche. Les fonctions fléchées se présentent de la sorte :

```
const calcAge = (birthYear) => 2022 - birthYear;

let age1 = calcAge(1984);
console.log(age1); // 38
```

On voit ici que le retour est ce qui se trouvera à droite de la flèche. Il est cependant possible de faire des fonction réalisant d'autres instructions avant de réaliser un retour. Pour ce faire, il nous fait altérer légèrement la syntaxe de notre fonction de sorte à obtenir :

```
const testMajority = (birthYear) => {
    let age = (2022 - birthYear);

    if (age >= 18) {
        console.log("La personne est majeure !");
    }
    else {
        console.log("La personne n'est pas majeure...");
    }

    return age;
}
```

Le DOM

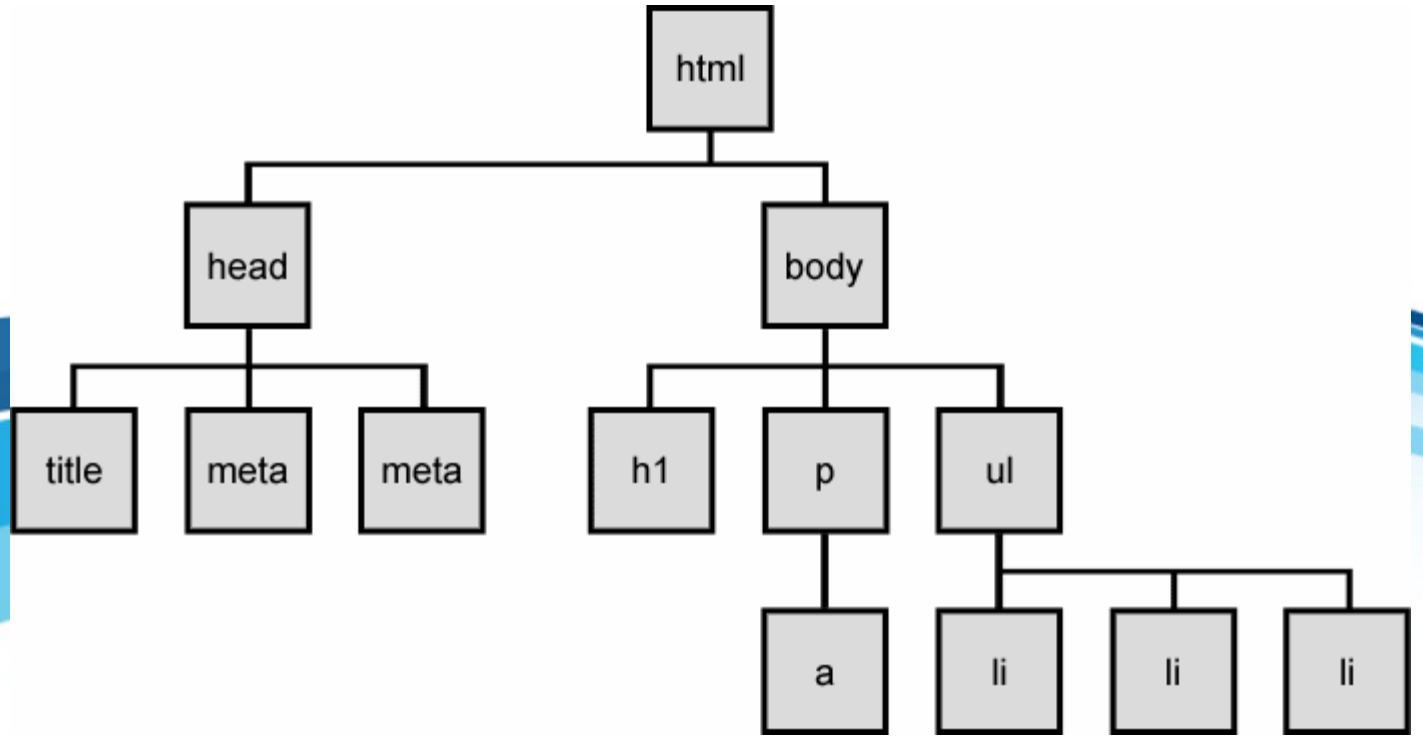
Le Document Object Model

Dans une page internet, il existe ce que l'on appelle le DOM (Document Object Model). Ce DOM est en réalité une hierarchie sous forme d'arbre qui comprend l'ensemble de notre HTML de façon à obtenir les liens entre les parents et les enfants. Commençant au niveau du super-parent Document, on descend ensuite à chaque fois qu'on rencontre une série de balises dans la structure de notre HTML, pour atteindre petit à petit les enfants et leurs contenu potentiel :

Pour le Javascript, cette structure est très avantageuse, car elle correspond à ce que l'on pourrait trouver dans des objets.

Pour atteindre un élément, on peut via le Javascript et l'API du Web utiliser les méthodes de la classe « **document** », telles que :

- **.getElementById()**
- **.getElementsByClassName()**
- **.querySelector()**
- **.querySelectorAll()**



Sélectionner et Manipuler l'HTML

Si l'on veut sélectionner l'HTML, il y a plusieurs méthodes, comme l'utilisation de **.getElementById()** ou de **.getElementsByClassName()**

```
let monHeader = document.getElementById("my-heading");
let mesListItems = document.getElementsByClassName("list-group-items");
```

Malgré tout, la solution la plus simple est sans doute d'utiliser la méthode du .**querySelector()** (et sa version multiple **.querySelectorAll()**). Pour ce faire, il suffit d'avoir recourt à la même façon de sélectionner les éléments que lorsque l'on fait du CSS. C'est-à-dire que la sélection d'une Id et la sélection d'éléments par leur classe commune via des syntaxes de ce type :

```
let monHeader2 = document.querySelector("#my-heading");
let mesListItems2 = document.querySelectorAll(".list-group-items");
```

Une fois les éléments HTML sélectionnés, il est assez facile d'en extraire leur contenu ou de le modifier. Pour ce faire, on passe par les propriétés de l'élément, comme on le ferait avec un objet classique :

```
console.log(monHeader.textContent);
monHeader.textContent = "Mon nouveau titre de site web !";
```

Les Events Listeners

Notre page HTML est capable de créer des évènements que le Javascript peut écouter. Par exemple, le clic d'un bouton, ou le survol d'un input, l'appui sur une touche du clavier ou la modification d'une valeur de <textarea>, tout ceci correspond à un évènement. En écoutant l'évènement que l'on souhaite suivre, on peut ensuit réagir à son déclenchement.

Pour créer un Event Listener, il n'y a rien de plus simple. Il suffit de sélectionner un élément HTML et d'exécuter sur lui la méthode **.addEventListener()** Dans cette méthode, il va falloir passer en premier le type de l'évènement sous la forme d'une **string,** puis une fonction qui sera exécutée lors de l'évènement (Baptisée l'Event Handler). Il est fréquent que l'on passe des fonction fléchées dans cette partie, de sorte à obtenir à la fin ce genre d'instruction :

```
let myButton = document.querySelector("#my-button");

myButton.addEventListener('click', () => {
    console.log(document.querySelector("#mon-input").textContent);
});
```

Manipuler le Style

Il est possible que l'on veuille modifier le style de notre page HTML via le Javascript, par exemple pour changer la couleur de fond d'une image ou d'une pour réagir à des changements effectués par l'utilisateur de notre site web. Pour ce faire, on peut utiliser tout d'abord la sélection d'un élément par les méthodes vues précédemment. Puis, il nous faut accéder à sa propriété **style**. Enfin, il nous est possible de modifier chaque propriété CSS via une syntaxe différente. En CSS, nous aurions par exemple eu une syntaxe de ce genre :

```css
#my-heading {
    background-color: red;
}
```

Cependant, en Javascript, les espaces dans les noms des propriétés CSS se traduisent en notation **camel case** de la sorte :

```javascript
document.querySelector("#my-heading").style.backgroundColor = "blue";
```

Si l'on veut modifier plusieurs styles à la fois, une syntaxe différente est possible. On peut passer dans la propriété **style.cssText** une string de la sorte :

```javascript
document.querySelector("body").style.cssText =
    `
    background-color: red;
    color: white;
    `
```

Manipuler les Classes

En plus de pouvoir modifier le style de nos éléments, le Javascript peut également facilement modifier les classes que portent nos différentes balises. En effet, chaque élément du DOM possède une propriété nommée **classList** qui contient l'ensemble des classes dont dispose l'élément HTML. Cet ensemble peut ensuite aisément se voir retirer ou ajouter des nouveaux éléments via les méthodes ci-dessous :

- **classList.remove(<valeur>) :** Cette méthode va tout simplement retirer la classe de l'élément HTML si elle se trouve parmi la liste des classes.
- **classList.add(<valeur>) :** Cette méthode va ajouter quant à elle une classe à l'élément HTML, lui permettant par exemple de subit ainsi des modification par le CSS.

```
const closeButton = document.querySelector("#close-btn");
const openButton = document.querySelector("#open-btn");

closeButton.addEventListener('click', () => {
    document.querySelector("#my-window").classList.add("hidden");
});

openButton.addEventListener('click', () => {
    document.querySelector("#my-window").classList.remove("hidden");
});
```

Gérer les touches du Clavier

Si l'on veut pouvoir proposer une interactivité entre notre site web et le clavier de l'utilisateur, il nous est possible de le faire via trois évènement que l'on peut retrouver au niveau du document lui-même :

- **keydown** : Lorsque l'utilisateur appuie sur une touche de son clavier
- **keypress**: Lorsque l'utilisateur laisse son doigt sur une touche de son clavier
- **keyup** : Lorsque l'utilisateur relâche la touche de son clavier

```
document.addEventListener('keydown', () => {
    console.log("Une touche a été pressée !");
});
```

Une fois l'évènement écouté via l'ajout d'un « event listener » au niveau du document via l'instruction ci-dessus. Cet évènement sera généré pour n'importe laquelle des touches du clavier. Il nous est cependant possible de modifier la chose de façon à récupérer l'évènement et de ne sélectionner que les touches qui nous intéressent. Pour cela, il faut ajouter un paramètre à l'event listener (le paramètre sera ici automatiquement l'évènement). Un évènement de ce type possède une propriété **.key**, qu'il nous est possible d'utiliser dans le cadre d'une condition pour obtenir une fonction plus aboutie de la sorte :

```
document.addEventListener('keydown', (event) => {
    if (event.key === "Escape") {
        document.querySelector("#my-window").classList.add("hidden");
    }
});
```

jQuery

Qu'est-ce que jQuery ?

Alors que le Javascript devenait de plus en plus populaire, de nombreuses personnes commençaient à penser que la syntaxe du langage et la manipulation du DOM était incroyablement verbeuse et complexe. Un simple ajout d'event listeners sur toute une famille de boutons pouvait rapidement se traduire en une série d'instructions de la sorte :

```
for (let i = 0; i < document.querySelectorAll("button").length; i++) {
    document.querySelectorAll("button")[i].addEventListener('click', function() {
        document.querySelector("h1").style.color = "red";
    });
}
```

Le créateur de jQuery, John Resig, proposa alors une alternative via sa librairie. Grâce à l'utilisation de jQuery, on peut désormais réaliser la même chose que dans l'exemple ci-dessus, mais en beaucoup moins de temps, et beaucoup plus facilement :

```
$("button").click(function() {
    $("h1").css("color", "red");
});
```

Installer jQuery

Tout comme lors de l'ajout de Bootstrap pour améliorer la rapidité d'écriture du CSS de notre site, il va nous falloir télécharger et installer la librairie jQuery pour altérer notre Javascript. Pour ce faire, il suffit de se rendre sur le site de jQuery.

Une fois sur place, on peut encore une fois voir que l'option de CDN est disponible. Dans le cadre d'un développement facilité et rapide, nous nous servirons de cette option. Pour récupérer jQuery via le Content Delivery Network, il suffit de copier coller l'un des liens proposés sur la page officielle en bas de notre page HTML. Par exemple, en se servant de celui de Google :

```
</body>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script src="./main.js"></script>
</html>
```

Il faut bien faire ici attention à placer le CDN de jQuery AVANT celui de notre script personnel sous peine de coder sur un script non fonctionnel au bout du compte.

Sélectionner le DOM avec jQuery

Pour sélectionner le DOM avec jQuery, il n'y a en réalité rien de plus simple. Il suffit d'utiliser la syntaxe de type **$(<selection>).** Par exemple, la sélection du body se ferait via **$("body")** alors que la sélection de tous les éléments portant la classe « list-group-item » se ferait avec l'instruction **$("list-group-item").** On remarque ici donc qu'il n'y a aucune différence pour jQuery entre la sélection unique et la sélection multiple. Il faut donc bien faire attention à ce que l'on fait et ce que l'on sélectionne. Cette syntaxe permet aisément de réduire notre code, comme le monde les équivalences ci-dessous :

```
let buttons = document.getElementsByClassName("btn");
let buttons2 = $("btn");
```

Par contre, il est intéressant de noter que tout comme le CSS, il est possible de créer des sélections basées sur des hierarchies, comme par exemple **$(ul.list-group li.list-groupitem)** pour sélectionner tous les éléments de type <li> portant la classe « list-group-item » contenus dans un élément de type <ul> portant la classe de « list-group ».

```
let myListGroupItems = $("ul.list-group.list-group-flush li.list-group-item");
```

Manipuler le CSS avec jQuery

Pour modifier le CSS avec jQuery, il suffit d'utiliser la méthode **.css()** sur notre sélection, puis de lui passer en premier paramètre la propriété CSS et en second sa valeur. Par exemple, changer la couleur de texte du body se ferait de la sorte :

```
$("body").css("color", "red");
```

Pour obtenir la valeur d'une propriété CSS, il suffit d'ommettre le second paramètre. Par exemple, pour obtenir la valeur RGB de la couleur de texte de notre NavBar, on pourrait imaginer opter pour cette instruction :

```
console.log($(".navbar").css("color"));
```

Cependant, malgré qu'il soit possible de modifier le CSS depuis le Javascript, il vaut mieux l'éviter. Si l'on souhaite modifier le design de notre site de façon interactive, alors il vaut mieux se servir de classes que l'on ajoute ou retire à nos balises pour les altérer. Ces classes seront ainsi gérées dans le CSS, et notre Javascript ne fera que de les rendre ou non actives via l'utilisation des méthodes **.addClass(<classe>)** et **.removeClass(<classe>)** Ainsi, nous respectons le principe du « **Separation of Concerns** ». L'ajout ou le retrait de plusieurs classes se fait en les séparant par un espace. Pour savoir si l'élément possède la classe, on peut utiliser la méthode **.hasClass(<c**

```
$("h1").addClass("big-title md-6");

$("h1").removeClass("big-title");
```

Modifier les éléments avec jQuery

Si l'on souhaite modifier le texte contenu dans un élément HTML, on peut le faire via l'utilisation de la méthode **.text(<texte>)**. Attention à ne pas modifier trop d'élément d'un coup. On peut également modifier l'HTML avec la méthode **.html(<html>)**.

```
$("button").text("Click Me !");

$("li#myli01").html(`<a href="./indexedDB.html">Retour à la page d'accueil</a>`);
```

Pour manipuler les attributs des éléments, il existe également une méthode avec jQuery : **.attr(<attribut>, <valeur>)**. Avec seulement le nom de l'attribut, on peut récupérer sa valeur, comme lorsque l'on utilise la méthode **.css()**. Pour faire la modification de la destination d'un lien, on peut par exemple utiliser cette instruction :

```
$("a#my-website-link").attr("href", "./index.html");
```

Si l'on veut ajouter ou retirer des éléments dans notre DOM, on peut se servir des méthodes **.before(<element>)** et **.after(<element>)** pour les placer en dehors de l'élément sélectionné, avant ou après lui. Une variante sous la forme de **.prepend(<element>)** et **.append(<element>)** existe également, mais celle-ci va ajouter l'élément dans l'élément actuellement sélectionné, juste après sa balise ouvrante / avant sa balise fermante. Pour supprimer un élément du DOM, il existe également la méthode **.remove()**

```
$("button").prepend(`<i class="bi bi-eye></i>`);
```

Modifier les éléments avec jQuery

Si l'on souhaite modifier le texte contenu dans un élément HTML, on peut le faire via l'utilisation de la méthode **.text(<texte>)**. Attention à ne pas modifier trop d'élément d'un coup. On peut également modifier l'HTML avec la méthode **.html(<html>)**.

```
$("button").text("Click Me !");

$("li#myli01").html(`<a href="./indexedDB.html">Retour à la page d'accueil</a>`);
```

Pour manipuler les attributs des éléments, il existe également une méthode avec jQuery : **.attr(<attribut>, <valeur>)**. Avec seulement le nom de l'attribut, on peut récupérer sa valeur, comme lorsque l'on utilise la méthode **.css()**. Pour faire la modification de la destination d'un lien, on peut par exemple utiliser cette instruction :

```
$("a#my-website-link").attr("href", "./index.html");
```

Events Listeners avec jQuery

Pour créer de l'interactivité, on aura rapidement besoin de créer des event listeners pour par exemple réagir à la pression d'un bouton de notre site. Dans le cas d'un clic sur un bouton, on peut se servir de la méthode **.click(<event handler>)** :

```
$("button").click(() => {
    console.log("Un bouton a été pressé !");
});
```

De même, si l'on veut réagir à l'appui sur une touche dans tout le document, on peut avoir recourt à la sélection du document et à la méthode **.keypress(<event handler>) :**

```
$(document).keypress(function (e) {
    console.log(e.key);
});
```

Pour d'autres évènements, il existe également une syntaxe plus globale. Lorsque l'on utilise jQuery, il suffit de sélectionner les éléments qui nous intéressent et de se servir de la méthode **.on(<event>, <event handler>)** pour réagir à l'évènement levé et exécuter automatiquement la fonction que l'on désire.

```
$("h1").on("mouseover", () => {
    console.log("Ca chatouille !");
});
```

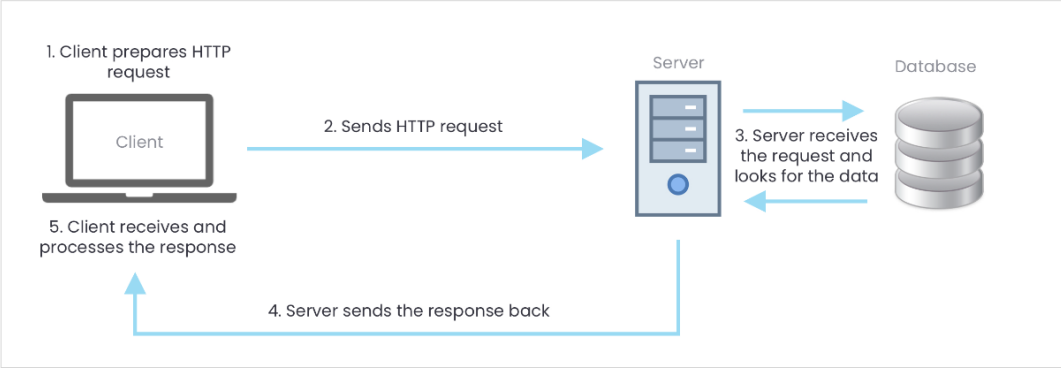
Introduction Ajax

AJAX-Introduction

AJAX signifie JavaScript asynchrone et XML. AJAX utilise l'objet XMLHttpRequest pour effectuer des requêtes HTTP afin d'envoyer et de recevoir des données à partir de sources externes.

Tout ce qui se passe en arrière-plan et une fois la page chargée, cela ajoute beaucoup d'interactivité à la page Web sans avoir à la recharger, ce qui est excellent pour l'expérience utilisateur.

L'image ci-dessous montre comment fonctionne AJAX.



Imaginons un champ de recherche pour comprendre comment on peut utiliser AJAX pour faire la recherche sans recharger la page.

1. Un événement se produit dans le client (l'utilisateur tape quelque chose dans la zone de recherche)
2. Le navigateur envoie une requête HTTP au serveur avec le terme de recherche
3. Le serveur recherche les données qui correspondent à ce terme de recherche dans la base de données.
4. Le serveur renvoie les résultats au client. (Normalement en JSON)
5. La réponse est reçue et analysée pour devenir un objet Javascript.
6. Le contenu de la page est modifié pour afficher les résultats de la recherche.

Maintenant que nous savons comment fonctionne AJAX, nous sommes prêts à faire notre première demande.

Introduction Ajax

- L'une des tâches fondamentales d'une application frontale est de communiquer avec les serveurs via le protocole HTTP.
- JavaScript peut envoyer des requêtes réseau au serveur et charger de nouvelles informations chaque fois que nécessaire sans recharger la page.
- Le terme pour ce type d'opération est Asynchronous JavaScript And XML (AJAX), qui utilise l'objet XMLHttpRequest pour communiquer avec les serveurs.
- Il peut envoyer et recevoir des informations dans différents formats, notamment des fichiers JSON, XML, HTML et texte.
- Ici, nous verrons comment faire une requête réseau pour obtenir des informations du serveur de manière asynchrone avec l'API fetch() et Axios et apprendre comment ils peuvent être utilisés pour effectuer différentes opérations.

Fetch

- L'API Fetch fournit une interface JavaScript pour accéder et manipuler des parties du pipeline HTTP, telles que les requêtes et les réponses.

- Il fournit également une méthode globale fetch() qui fournit un moyen simple et logique de récupérer des ressources de manière asynchrone sur le réseau.

- La méthode fetch() prend un argument obligatoire - le chemin d'accès à la ressource que vous souhaitez récupérer - et renvoie une Promise qui se résout avec un objet de la classe intégrée Response dès que le serveur répond avec des en-têtes.

- Le code ci-dessous illustre une requête de récupération très basique dans laquelle nous récupérons un fichier JSON sur le réseau.

javascript

```javascript
fetch('examples/example.json')
  .then((response) => {
    // Do stuff with the response
  })
  .catch((error) => {
    console.log('Looks like there was a problem: \n', error);
  });
```

Fetch

Une fois qu'une réponse est récupérée, l'objet renvoyé contient les propriétés suivantes :

- **response.body** : un getter simple exposant un ReadableStream du contenu du corps.
- **response.bodyUsed** : stocke un booléen qui déclare si le corps a déjà été utilisé dans une réponse.
- **response.headers** : l'objet d'en-tête associé à la réponse.
- **response.ok** : un booléen indiquant si la réponse a réussi ou non.
- **response.redirected** : Indique si la réponse est le résultat d'une redirection ou non.
- **response.status** : code d'état de la réponse.
- **response.statusText** : le message d'état correspondant au code d'état.
- **response.type** : le type de la réponse.
- **response.url** : l'URL de la réponse.

Fetch

Il existe un certain nombre de méthodes disponibles pour définir le contenu du corps dans différents formats :

- **response.json():** Parse the response as JSON
- **response.text():** Read the response and return as text
- **response.formData():** Return the response as FormData object
- **response.blob():** Return the response as Blob
- **response.arrayBuffer():** Return the response as ArrayBuffer

Axios

Axios est une bibliothèque Javascript utilisée pour effectuer des requêtes http à partir de node.js ou XMLHttpRequests à partir du navigateur, et elle prend en charge l'API Promise native de JS ES6.

- **Certaines fonctionnalités essentielles d'Axios, selon la documentation, sont :**

  - **Il peut être utilisé pour intercepter les requêtes et les réponses http.**
  - **Il transforme automatiquement les données de requête et de réponse.**
  - **Il permet une protection côté client contre XSRF.**
  - **Il a un support intégré pour la progression du téléchargement.**
  - **Il a la capacité d'annuler les demandes.**

Axios

Axios n'est pas une API JavaScript native, nous devrons donc importer manuellement dans notre projet. Pour commencer, nous devrons inclure les commandes suivantes :

Using cdn

```html
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

Using npm

```node
npm install axios
```

Using bower

```bash
bower install axios
```

Axios

javascript

```javascript
axios.get('examples/example.json')
  .then((response) => {
    // handle success
    console.log(response);
  })
  .catch((error) => {
    // handle error
    console.log(error);
  })
```

Axios

Axios fournit également plus de fonctions pour effectuer d'autres requêtes réseau, correspondant aux verbes HTTP que vous souhaitez exécuter, tels que :

- axios.request(config)
- axios.get(url[, config])
- axios.delete(url[, config])
- axios.head(url[, config])
- axios.options(url[, config])
- axios.post(url[, data[, config]])
- axios.put(url[, data[, config]])
- axios.patch(url[, data[, config]])

Vous pouvez consulter la configuration complète de la demande dans la documentation officielle.

Axios

La réponse d'une requête contient les informations suivantes :

- **response.data :** La réponse fournie par le serveur.
- **response.status :** Le code d'état HTTP de la réponse du serveur.
- **response.statusText :** Message d'état HTTP de la réponse du serveur.
- **response.headers :** Les en-têtes avec lesquels le serveur a répondu.
- **response.config :** La configuration qui a été fournie à axios pour la requête.
- **response.request :** La requête qui a généré cette réponse.

JSON data

Il existe un processus en deux étapes lors de la gestion des données JSON avec fetch(). Tout d'abord, nous devons faire la demande réelle, puis nous appelons la méthode .json() sur la réponse.

javascript

```javascript
fetch("examples/example.json") // first step
  .then(response => response.json()) // second step
  .then(data => {
    console.log(data)
  })
  .catch(error => console.error(error))
```

JSON data

Axios transforme automatiquement les données JSON une fois la demande résolue, nous n'avons donc pas grand-chose à faire ici.

javascript

```javascript
axios
  .get("examples/example.json")
  .then(response => {
    console.log(response.data)
  })
  .catch(error => {
    console.log(error)
  })
```