Javascript 01

Romain Jalabert - 13.02.2023 - IPSSI

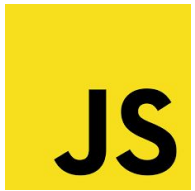
JS - Quid ?

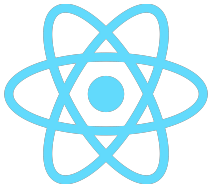
JavaScript permet la gestion des interactions entre l'utilisateur et le navigateur ainsi que l’évolution dynamique du contenu.

Javascript créé en **1995** par **Brandon Eich** est un langage qui va être interprété directement coté navigateur par un moteur / interpréteur JS. Chaque grand navigateur a son propre JS engine (V8 pour Chrome par exemple)

JS possède de nombreux framework front aux avantages divers. Notamment **React, Vue, Svelt, Next, Phaser JS (jeux vidéos), React Native (pour mobile)**

JS est aussi disponible pour gérer l’aspect back-end grâce à **node JS** et **Express** entre autres.





La syntaxe

Les variables en JS s'écrivent **var** et respectent généralement une écriture **camelCase** dans leur nom.

```
var coolNumber = 8
```

Les différents types de variables :

- Number
- String
- Boolean
- Array
- Object

Afin d'inspecter une variable (visualiser son contenu) on utilise **console.log(var)**

Pour visualiser le type de la variable on utilise **typeof.**

ex: **console.log(typeof 'hey')** va donner **string** dans la console

La portée des variables (ou Scope)

var possède ses limites lorsque l'on écrit du JS, particulièrement parce que l'on peut changer le contenu de var et réécrire celle-ci, mais aussi parce que sa portée est globale ou fonctionnelle.

On va donc souvent préférer **const** et **let** dont la portée est celle d'un bloc (entre {}) mais qui déclarées hors d'un bloc (ou d'une fonction) acquièrent une portée globale.

La différence entre **const** et **let** tient du fait que **let peut être réécrite**, contrairement à const dont la valeur est constante.

On préférera la plupart du temps utiliser const à part si l'on est sûr de devoir réécrire la valeur de la variable auquel cas **let** sera adéquate.

Les opérateurs

> Supérieur à …

< Inférieur à …

>= Supérieur ou égal à …

<= Inférieur ou égal à …

= : opérateur d'affectation (pour changer la valeur d'une variable, d'un élément de tableau…)

== : opérateur de comparaison souple (ne compare que la valeur, pas le type)

=== : opérateur de comparaison stricte (compare type & valeur)

Les opérateurs de comparaison **&& (ET)** et **|| (OU)** suivent la même logique que pour php

Afin de **concaténer** deux chaînes de caractères on utilise **+**

L'opérateur **%** dit 'modulo' va nous permettre de retourner le reste d'une division

Les boucles

WHILE qui va exécuter les instructions tant que notre condition est valable

```
var x = 20;

while ( x <= 20 ) {
    console.log(x);
}
```

DO … WHILE qui va exécuter une première fois les instructions puis jusqu'à ce que notre condition soit vérifiée

```
do {
    console.log(x);
} while ( x <= 20 );
```

Les boucles

FOR qui va exécuter répéter les instructions pour chaque nouvelle valeur de ma variable de contrôle i

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

FOR … IN qui va exécuter les instructions pour chaque propriété dans l'objet

```
var objets = {
    nom : 'table',
    couleur : 'bleu',
};

for (var key in objets) {
    console.log(key)          // nom, couleur
    console.log(objets[key])  // table, bleu
}
```

Les boucles

FOR … OF qui pour chaque élément d'une entité itérable exécute les instructions définies dans la boucle.

```
var planets = [
    { name : "mars", size: 18900 },
    { name : "venus", size: 4500 },
    { name : "jupiter", size: 1290 }
];


for (var planet of planets) {
    console.log('La planète ' + planet.name + ' à peu près' + planet.size + ' de km de diamètre')
};
```

Les conditions

IF va exécuter les instructions si la condition est respectée.
Avec **IF … ELSE** on va pouvoir définir des instructions lorsque notre condition n'est pas respectée

```
var open = false;

if (open) {
    console.log("c'est ouvert !");
} else {
    console.log("fermé !");
}
```

ELSE IF va nous permettre de rajouter une condition supplémentaire

```
var number = 23;

if (number < 10) {
    console.log("Moins de 10 !");
} else if (number > 25){
    console.log("Plus de 25 !");
} else {
    console.log("Ni l'un ni l'autre !")
}
```

Les conditions

SWITCH permet de spécifier des instructions selon la valeur d'une variable

```
var food = 'fondue';

switch (food) {
  case 'fondue':
    console.log('Normale ou Savoyarde ?');
    break;
  case 'pizza':
    console.log('Margarita ?');
    break;
  case 'sushi':
    console.log('Quelle sauce ?');
    break;
  default:
    console.log('Désolé mais nous ne trouvons pas !');
}
```

Exercices

1) Afficher dans la console les chiffres de 1 à 500
2) Afficher dans cette liste seulement les nombres pairs
3) Afficher une erreur dans la console pour les impairs (console.error)

JS - Les fonctions

Sans paramètres :

```
function hello() {
        console.log("hi there !");
}

hello();
```

Avec paramètres :

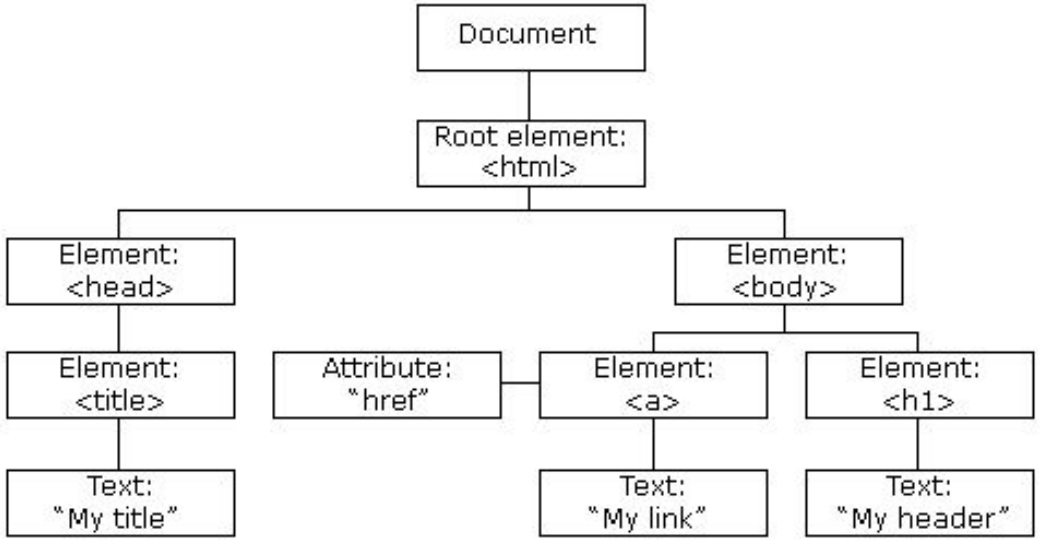
```
function getMessage(name) {
    return 'Are you ' + name + ' ?';
}

getMessage('Joe');
```

Au moment d'appeler la fonction on lui passe un **argument** (ici ‘Joe’)

DOM : Document Object Model

Le HTML **Document Object Model** est une notion fondamentale en JS et représente une arborescence des **objets** qui constituent une page. C'est le navigateur qui crée le DOM lors du chargement d'une page. JS va venir se saisir de ces éléments pour rendre les pages dynamiques. On parle alors de manipulation du DOM.



Récupérer un élément HTML

En passant par l'**id**

```
var form = document.getElementById('form');
```

En passant par **querySelector**

```
var test = document.querySelector('.test');
```

Avec **querySelectorAll**

```
var elements = document.querySelector('#unid, .intro');
//elements vaut NodeList [ <p.intro>, <div#unid> ]

var introduction = elements[0]; // <p.intro>
var unid = elements[1]; // <div#unid>
```

Créer un élément HTML

```
var element = document.createElement('div');
```

Ajouter du contenu texte et HTML

```
var item = document.createElement('li');

var list = document.querySelector('ul');

list.appendChild(item);
```

Modifier le contenu d’un élément HTML

Avec **textContent**

```
element.textContent = 'Hi !';
```

Avec **innerHTML**

```
element.innerHTML = '<h1>Hi !</h1>';
```

Les Objets

Ils vont nous servir à stocker de l'information : strings, fonctions, boolean, tableaux, autres objets …

Les objets prennent la forme { clé1 : valeur1, clé2 : valeur2 }

```
var object = {
    name : 'obj',
    content: [
        'strings',
        'stuffs',
        '',
      ],
    add : function() {
        //
    }
}
```

Les événements

Créer un écouteur d'événement avec **addEventListener**

```
element.addEventListener(
    eventType,
    handler
  )
```

L'écouteur nous permettre de déclencher des instructions via une fonction (aussi appelée ***callback***) lorsqu'un événement précis se déclenche.

```
var button = document.getElementById('button');

var add = function() {
    // DO SOMETHING
};

button.addEventListener('click', add);
```

Exercices

1) Créer une fonction qui écrit Hello ! dans une pop-up (avec alert()).
2) Écrire une fonction qui affiche un argument dans une pop-up.
3) Si le message est trop long (20 caractères), afficher un message d'erreur à la place.
4) Créer un carré de couleur dont la couleur change lorsque l’on clique dessus. Il doit passer de vert à bleu (et de bleu à vert quand on clique à nouveau)
5) Créer un bouton + qui incrémente une valeur jusqu’à 20 ou elle s'arrête.

Exercice Articles

À l’aide des données du tableau d’objets dans data.js, afficher la liste des articles sur notre index en utilisant quasi-exclusivement du JS (createElement, les sélecteurs etc).

Quelques persos cools !

Alfred

Housekeeper

Joe

Taxidriver

Akira

Cyborg

Elektra

Sniper

Travaux Pratiques

On va faire un **éditeur de code** en JS. On veut pouvoir rentrer du HTML dans une zone spécifique du code et que celui-ci soit traduit lorsque l'on appuie sur une touche (enter ce serait chouette !). Astuce : utiliser **innerHTML**. La zone de code est un **textarea.**

1) On crée notre index.html avec les éléments html qui conviennent
2) On va ensuite créer une page app.js que l'on importe dans notre index.html
3) Enfin on va pouvoir commencer à déclarer les variables adéquates (se saisir de nos éléments HTML) et créer une fonction avec les instructions nécessaires à notre éditeur.

Les événements JS avec target

JS peut réagir à des **événements** qui se produisent en HTML.

Un event HTML peut être lié à une action du navigateur (page qui finit de charger) ou à celle d'un utilisateur (clique sur un bouton, input qui change …)

On peut ainsi exécuter du code lorsque ces événements sont détectés. *Par exemple*, lorsque l'on utilise un eventListener sur un élément HTML il est possible de récupérer des informations sur l'événement en question (si l'on clique, on peut récupérer la cible du clic ainsi que ses informations avec target).

```
const play = document.querySelector('.btn-play');

play.addEventListener('click', (e) => {console.log(e.target)})

// Retourne <div class="btn-play">
```

Travaux Pratiques

Classique mais efficace, on va faire une **to-do liste** en JS. Il y a beaucoup de ressources sur Internet donc évitez le copier coller intégral l'idée étant de comprendre ce que l'on fait ! On veut pouvoir ajouter une todo dans un premier temps. On verra pour la suppression, le filtrage et le check par la suite.

Si vous vous sentez d'aller plus loin c'est top !

On va procéder par étape :

1) Créer notre fichier html, importer dans celui-ci app.js ainsi que index.css
2) Créer nos premiers éléments HTML (l'input, output etc)
3) Styliser un minimum nos éléments
4) Créer les sélecteurs dans notre app.js

Ressources

W3schools : https://www.w3schools.com/js/default.asp

MDN : https://developer.mozilla.org/en-US/docs/Web/JavaScript

JS “Cheat Sheet” : https://htmlcheatsheet.com/js/

Stack Overflow pour voir si quelqu’un à déjà rencontré votre problème ! +

Pour les animations : animate.css (https://animate.style/)