I – Intro

JavaScript permet de rendre les pages Web dynamiques en modifiant leur contenu (html, css) en fonction de certaines actions de l'utilisateur générant des événements (clic, scroll, survol...).

Remarque : JavaScript, conçu en 1995, est aujourd'hui une implémentation de la norme **ECMAScript**.

> Les scripts sont inclus dans un élément `<script>`. C'est une bonne pratique de les écrire dans un fichier externe, lié au fichier HTML (dans l'en-tête par exemple) avec la balise `<script src="fichier.js" defer></script>`.

L'attribut `defer` (facultatif en réalité) permet d'attendre que le navigateur ait entièrement chargé le code HTML et construit le DOM avant d'exécuter le script.

Les scripts JavaScript sont exécutés par le navigateur du client, sans interaction avec le serveur.
Il est possible de visualiser et d'interagir directement avec la console de l'interpréteur JavaScript du navigateur en activant les outils de développement (touche F12 en général).

II – Quelques éléments du langage

Détail préliminaire :
Pour se forcer à écrire un code rigoureux qui respecte strictement les normes ECMAScript, on commencera toujours les scripts par l'instruction `"use strict";`.

> Chaque instruction se termine par un **point-virgule.**
> **Les accolades** définissent des **blocs d'instructions**.
> Les variables doivent être déclarées avec le mot clef `let` (si la variable peut changer) ou `const` (si la variable est constante) (ou même `var` pour une variable globale).
> Les fonctions sont introduits avec le mot clef `function`.

Exemple :

```
function maFonction(arg1, arg2){ // l'indentation est facultative
    console.log(arg1 + arg2);    // console.log = équivalent du print de Python
    return arg1 + arg2;
}
```

III – Structures de contrôle

On retrouve les structures de contrôles classiques (voir exemples à suivre) :

```
function test(a){
    if (a > 0){           // structure conditionnelle : if ... else if ... else
        return "positif";
    } else if (a === 0){  // le "triple égal" teste l'égalité de valeur et de type
        return "nul";
    } else{
        return "négatif";
    }
}
```

```javascript
let n = 0;
while (n <= 12){       // Boucle while
    console.log(n);
    n++;               // incrémentation automatique, équivalent à n = n + 1
}
```

```javascript
for (let n = 0; i <= 12; n++){   // Construction d'une boucle for
    console.log(n);
}
/* let n = 0 : instruction réalisée initialement au début de la boucle
   n++       : instruction réalisée à chaque fin d'itération de boucle
   n <= 12   : teste la poursuite de la boucle
*/
```

IV – Types de données composés

L'équivalent des dictionnaires de Python sont des **Objets** en JavaScript.
Exemple : `let monObjet = {"nom" : "Durand", "prenom" : "Louis", "age" : 45};`

Les tableaux existent en JavaScript avec la même syntaxe que les listes Python.
Exemple : `let monTableau = ["André", "Marie", "Sylvie"];`
`monTableau[1]` vaut `"Marie"`.

On ajoute un élément au tableau avec la méthode `push`.
`monTableau.push("Luc");`

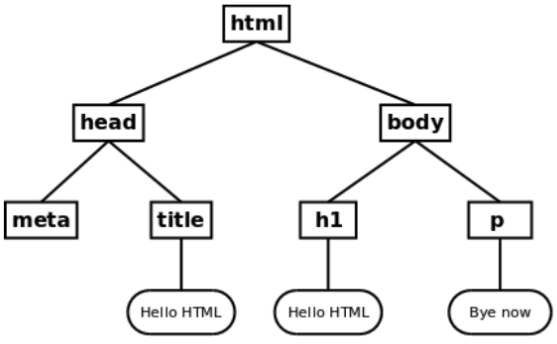
La taille du tableau est obtenue avec la **propriété** `length` : `monTableau.length;`

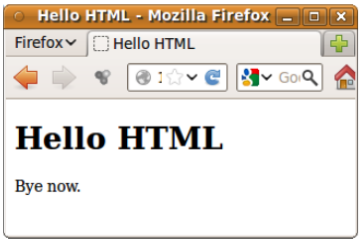
V – Modifier le DOM en JavaScript

1. Arbre DOM

Le **DOM** (Document Object Model) qui est la représentation du document html sous forme d'**arbre** faisant apparaître les différents **éléments** de la page.
Le navigateur construit le DOM à partir du code html et affiche la vue résultante :

```html
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Hello HTML</title>
</head>
<body>
  <h1>Hello HTML</h1>
  <p>Bye now.</p>
</body>
</html>
```





2. Accéder aux nœuds du DOM

L'API DOM (Application Programming Interface) implémentée par le navigateur permet à JavaScript d'accéder et de modifier les nœuds de l'arbre DOM.

L'objet `document` identifie le document manipulé (la page html).
À partir de cet objet on peut atteindre tous les nœuds du DOM (soit par leur nom propre, soit par leur identifiant, soit par leur sélecteur CSS). On pourra se limiter à utiliser les sélecteurs CSS selon la syntaxe suivante :

```
document.querySelector("selecteurCSS");
```

3. Modifier le CSS d'un élément à travers la propriété `style`

Exemple : modification des couleurs en JavaScript :

```
document.querySelector(".important").style.color = "red";
/* définit la propriété <color> des éléments de classe <important> */
document.querySelector("#super").style.backgroundColor = "blue";
/* définit la propriété <background-color> de l'élément dont l'id vaut <super>.
Bien noter la transformation du tiret en notation camelCase ! */
```

4. Modifier un nœud du DOM

On commence par créer une variable ciblant le nœud d'intérêt :

```
let monElement = document.querySelector("selecteurCss");
```

On peut alors travailler sur l'objet `monElement`.

Exemples :

- `monElement.getAttribute(nomAttribut)` : récupère la valeur d'un attribut.
- `monElement.setAttribute(nomAttribut, nouvelleValeur)` : modifie un attribut.
- `monElement.textContent = "nouveau texte"` : modifie le texte de l'élément.
- `monElement.innerHTML = "code html"` : modifie le code html à l'intérieur de l'élément.

VI – Programmation événementielle

> Le principe de la programmation événementielle est de **lancer l'exécution d'une fonction lorsqu'un certain événement est déclenché**.

Pour une application Web, les événements sont par exemple des clics, ou survols de souris, etc.

1. Méthode 1 : utiliser des événements prédéfinis

Il est possible d'ajouter des attributs de gestion d'événement aux éléments HTML et d'y associer un code JavaScript (en général, le code se réduit à l'appel d'une fonction).
Les gestionnaires d'événements principaux sont `onclick` (réaction à clic de souris), ou `onmouseover` (réaction à un déplacement de la souris). Il en existe d'autres comme `onblur` ou `onfocus`...

Exemple (dans le fichier HTML) :

```
<button id="clac" onclick="change_nom()">Cliquez-moi !</button>
<input type="text" onfocus="change_couleur(this)">
```

Accompagné du code JavaScript suivant :

```
function change_nom(){
    document.querySelector("#clac").textContent = "Merci !";
}

function change_couleur(element){
    element.style.background = "red";
}
```

2. Méthode 2 : créer ses propres gestionnaires d'événements

Il est possible d'ajouter un « gestionnaire d'événement » (**event listener**) à tout élément du DOM. Il est alors nécessaire de définir la fonction (dite fonction callback) à exécuter lorsque l'événement écouté est **capté**.

La syntaxe est la suivante :

```
/* ajout d'un gestionnaire d'événement de click sur l'élément monElement
qui exécute la fonction action lorsque le click a lieu */
monElement.addEventListener("click", action);

function action(event){   // objet Event reçu automatiquement en paramètre
    console.log("on a cliqué sur monElement !");
    console.log(event);  // affiche les propriétés de l'objet Event
}
```

Lorsqu'un événement est capté, la fonction callback reçoit automatiquement un objet **Event** en paramètre. Cet objet contient des propriétés sur l'événement capté (par exemple coordonnées de la souris lors d'un clic, ou la touche du clavier).
L'objet Event a aussi deux attributs importants :

- currentTarget : le nœud du DOM qui a déclenché la fonction callback.
- target : le nœud du DOM qui a reçu l'événement en premier.

3. Types d'événements écoutables

- événements liés à la souris : mouseenter, mouseleave, click, mousedown ...
- événements liés au clavier : keyup, keydown, keypress.
- événements globaux : load, unload, resize, scroll ...
- événements liés aux formulaires : focus, change, blur, submit ...