1 Intro

Javascript permet de rendre les pages Web dynamiques en modifiant leur contenu (html, css) en fonction de certaines actions de l'utilisateur (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>` , et peuvent être écrits dans un fichier externe avec la balise `<script src="fichier.js"></script>`

Il est préférable d'écrire les scripts à la fin du document html (avant la fermeture de body) pour que l'ensemble de la page soit chargée avant l'exécution des scripts.

La console de Javascript est accessible avec l'inspecteur de Firefox.
Pour écrire dans la console : `console.log("du texte")` .

Les scripts Javascript sont exécutés par le navigateur du client, sans interaction avec le serveur.

2 Quelques éléments du langage

Pour se forcer à écrire un code correct et rigoureux, 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` .

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

Les tableaux existent en Javascript avec la même syntaxe que les listes Python.
Exple : `let monTableau = ["André", "Marie", "Sylvie"];`
`monTableau[1]` renvoie `Marie`
On ajoute un élément au tableau avec la méthode `push` .
`monTableau.push("Luc");`
La taille du tableau est obtenue avec l'attribut `length` : `monTableau.length;`

Les fonctions sont introduits avec le mot clef `function` .
Exple :

```
function maFonction(arg1, arg2){
  console.log(arg1 + arg2);
  return arg1 + arg2;
}
```

3 Structures de contrôle

On retrouve les structures de contrôles classiques. Voir les exemples suivants.

```
function test(a){
  if (a > 0){
    return "positif";
  }else if(a == 0){
    return "nul";
  }else{
    return "negatif";
  }
}
```

```
let maxi = 25;
for (let i = 0; i <= maxi; i++){
  console.log(i);
}
```

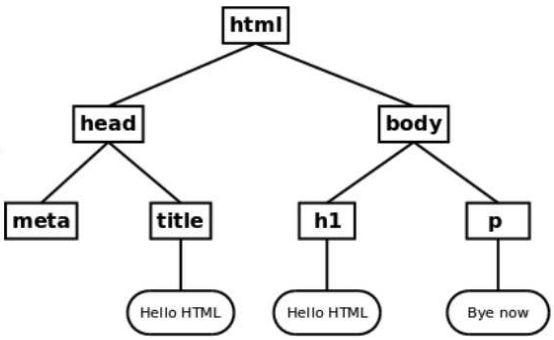
```
let n = 0;
while (n < 12){
  console.log(n);
  n++;
}
```

4 Modifier le DOM en Javascript

4.1 arbre DOM

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

```
<!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>
```



4.2 Accéder aux nœuds du DOM

Javascript permet de modifier les nœuds de l'arbre DOM.

L'objet `document` est le document manipulé.
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) :

- `document.documentElement` : racine est du document, correspond à l'élément `html`.
- `document.body` : correspond à l'élément `body`.
- `document.getElementById("identifiant")` : correspond à l'élément dont l'identifiant est `identifiant`.
- `document.querySelector("selecteurCss")` : correspond au **premier** élément défini par le sélecteur CSS `selecteurCss`.
- `document.querySelectorAll("selecteurCss")` : renvoie un **tableau listant tous les éléments** définis par le sélecteur CSS `selecteurCss`.

Les nœuds du DOM peuvent être de type **texte** ou **élément html**.
Les objets de type **Node** regroupent indistinctement ces deux types de nœuds.
Les objet de type **Element** regroupent seulement les nœuds de type élément html (c'est principalement eux que l'on souhaite manipuler en général).

4.3 Modifier le CSS d'un élément

`document.getElementById("identifiant").style.color="blue";` : redéfinit la propriété `color` du CSS.

`document.getElementById("identifiant").style.backgroundColor="red";` : redéfinit la propriété `background-color` du CSS. Bien noter la **transformation du tiret** en notation **camelCase** !

4.4 Modifier un nœud du DOM

On commence par "récupérer" le nœud d'intérêt :
`let monElement=document.getElementById("identifiant");` : on peut maintenant travailler sur l'objet`monElement`.

- `monElement.children` : renvoie un tableau listant tous les éléments enfants du nœud.
- `monElement.getAttribute(nomAttribut)` : récupère la valeur d'un attribut .
- `monElement.setAttribute(nomAttribut, nouvelleValeur)` : modifie un attribut.
- `monElement.classList.add("nouvelleClasse")` : ajoute une classe à l'élément.
- `monElement.textContent="nouveau texte"` : modifie le texte de l'élément.

4.5 Créer une nouvelle branche du DOM

Observer l'exemple suivant ajoutant un nouveau paraphe dans le document html :

`let nouveauParag=document.createElement("p");` : créer un nouvel élément de type paragraphe.
`let nouveauTexte=document.createTextNode("joli contenu");` : créer un nouveau nœud du type texte (avec son contenu).
`nouveauParag.appendChild(nouveauText);` : ajoute le nœud texte comme un fils de l'élément paragraphe.
`let maDiv=document.getElementById("identifiant");` : récupère l'objet maDiv où l'on veut ajouter le paragraphe créé.
`maDiv.appendChild(nouveauParag);` : ajoute le paragraphe en tant que fils de maDiv.

Autre exemple pour ajouter un lien hypertexte avec ses attributs :

```
let monA = document.createElement("a");
monA.setAttribute("href", "http://example.com");
monA.setAttribute("title", "Exemple");
let monText = document.createTextNode("Le site example.com");
monA.appendChild(monText);
maDiv.appendChild(monA);
```

Il est aussi possible (entre plein d'autres choses !) de supprimer un nœud ou de remplacer un nœud par un autre :

```
elementPere.removeChild(elementFils);
elementPere.replaceChild(nouveauFils, ancienFils);
```

5 Programmation événementielle

Le principe de la programmation événementielle est de **lancer l'exécution d'une fonction lorsqu'un certain événement est déclenché** au niveau de l'application. Pour une application Web, les événements sont par exemple des clics, ou survols de souris, etc.

5.1 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 (on général, le code se réduit à l'appel d'une fonction).
Les gestionnaires d'événement 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 :

```
<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.getElementById("clac").textContent="Merci !";
}

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

5.2 Méthode 2 : créer ses propres gestionnaires d'événements

Il est possible d'ajouter un "capteur d'événement" (**event listener**) à tout élément du DOM. Il est alors nécessaire de définir la fonction à exécuter lorsque l'événement écouté est **capté**. C'est une fonction **callback**.

La syntaxe est la suivante :

```
monElement.addEventListener("click", fonctionCallback);

function fonctionCallback(event){
  console.log("on a cliqué sur monElement !");
  console.log(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.

5.2.1 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 ...

5.2.2 Complément

- `event.preventDefault()` : empêche l'action par défaut de l'événement (ex : activation d'un lien)
- `event.stopPropagation()` : arrête la propagation de l'événement à travers les différentes "couches" html.
- `monElement.removeEventListener("click", fonctionCallback)` : retire le capteur d'événement sur monElement.