III - Initiation à javascript

a) Commandes de base en javascript

Javascript est un langage de programmation possédant ses spécificités (permettant l'interaction sur une page Web), mais il possède des fonctionnalités communes avec d'autres langages comme python.
C'est un langage qui s'exécute côté client (même si maintenant on trouve des scripts côté serveur) : il est hébergé sur le même serveur que le site d'un page web consultée mais une fois chargé, c'est le navigateur (le client) qui l'exécute.

Il faut tout d'abord comprendre que les blocs en javascript sont **délimités par des accolades**.

- **Les variables**

  En javascript, on peut déclarer des variables à la volée (sans mot-clé), comme en python, mais c'est à éviter car elles seraient comme globales.

  Il faut donc utiliser les **mots-clés let** (pour une variable dont la valeur peut changer) et **const** (pour une constante) pour les définir de sorte qu'elles aient une action que nous sommes en mesure de contrôler. Le mot-clé **var** existe aussi et est assez proche de **let** (même si son champs d'action est un peu différent).

- **Les tuples et listes**

  Les tuples n'existent pas en javascript. Les listes sont vues comme des tableaux (du type Array) et on peut également avoir des tableaux de tableaux (ou tableaux à plusieurs dimensions).
  Comme pour les listes, l'élément d'index i d'un tableau nommé tab est obtenu par tab[i] et le premier index est 0.
  La longueur d'un tableau est obtenu par la méthode **length** sous la forme **tab.length**.

- **Les boucles :**

En python	En javascript

`for i in range(n):`
instructions
	`for(let i=0;i<n;i++){`
	instructions
	`}`

`while condition :`
instructions
	`while(condition){`
	instructions
	`}`

  Pour parcourir les éléments d'un tableau ou d'une chaine de caractères, en python, on écrit :
  `for elt in iterable:`

  En javascript, il n'y a pas d'équivalent. La première solution consiste à travailler avec les index (pour les types Array et chaînes de caractères) :

  ```
  for(let i=0;i<iterable.length;i++){
  ```

  Pour ceux qui désireraient aller plus loin dans le javascript, je signale qu'il existe bien une syntaxe similaire pour un tableau (nommé tab) avec la méthode forEach : `tab.forEach(function(elt){...})`

  Il existe bien une instruction `for(let elt in monObjet){...}` mais elle est réservée pour des types particuliers appelés objets (qui peuvent être vus comme des dictionnaires).

- **Les instructions conditionnelles :**

En python	En javascript

`if condition :`
instructions1
`else :`
instructions2
	`if (condition) {`
	instructions1
	`} else {`
	instructions2
	`}`

- **Les fonctions :**

En python	En javascript

`def maFonction(arguments):`
instructions
`return ...`
	`function maFonction(arguments){`
	instructions
	`return ...`
	`}`

On trouve de très nombreux tutoriels ou explications pour programmer en javascript. Pour bien débuter :

- MDN Firefox :
  https://developer.mozilla.org/fr/docs/Apprendre/Commencer_avec_le_web/Les_bases_JavaScript
  Pour aller plus loin :
  https://developer.mozilla.org/fr/docs/Web/JavaScript/Une_r%C3%A9introduction_%C3%A0_JavaScript

- OpenClassroom : https://openclassrooms.com/fr/courses/2984401-apprenez-a-coder-avec-javascript
  Pour aller un peu plus loin :
  https://openclassrooms.com/fr/courses/1916641-dynamisez-vos-sites-web-avec-javascript

b) Mise en place d'un fichier javascript

1. **HTML et CSS**
   On dispose d'un fichier index.html qui utilise une feuille de style nommée style.css.
   Ce fichier html est un résumé de ce qu'il faut retenir en python, avant l'année de première et au cours de cette année.
   Sa lecture n'est pas aisée car tout est sur une même page.
   Nous allons utiliser un langage de programmation (javascript) pour rendre la page dynamique.
   Dans un premier temps, il faut charger le fichier javascript (dans le même esprit que le fichier style.css). Ajouter dans l'entête de la page html la ligne : `<script src="script.js"></script>` où script.js est le nom du fichier javascript.

   Comme pour le style de la page, le javascript pourrait être intégré à la page html, mais c'est une mauvaise pratique.

2. **Déclarer et construire le fichier js**
   Notre fichier js vient d'être déclaré, il faut alors créer un nouveau fichier (avec Visual Studio Code ou Notepad++) et le nommer script.js (l'enregistrer au même niveau que le fichier index.html).
   Remarque : lors de l'enregistrement, il faut bien sélectionner le type js pour que le fichier soit au bon format. L'éditeur adaptera alors la coloration du code.

   On peut effectuer un premier test pour vérifier qu'il est bien pris en compte :

   écrire `console.log("test")` : ceci écrira alors le mot "test" dans la console du navigateur. Ouvrir dans le navigateur le fichier index.html puis la console qui se trouve dans les outils de développement (ou Ctrl+Maj+I sous Firefox, F12 sous chrome).

   Le mot "test" est bien écrit (on peut aussi écrire `alert("test")` qui ouvrira un popup avec le mot écrit – ceci interrompt alors la lecture du code de la page, je ne l'utilise que rarement, seulement quand j'ai une boucle infinie et que je ne trouve pas l'erreur car cela empêche le navigateur de planter...).

   On peut aussi demander d'afficher des éléments HTML :

   `console.log(document.head)` ou `console.log(document.body)`.

   Ceci n'affiche pas des éléments textuels présents sur la page mais les différents objets qui sont construits grâce aux balises.

   On se rend tout de même compte que la seconde instruction renvoie null alors que le corps de la page existe bien. En effet, le code js est exécuté avant de compléter la page.

   > Remarque : Dans le cas présent, déclarer le fichier script.js à la fin du fichier html solutionnerait le problème de l'affichage de null. Mais c'est une solution à éviter car on pourrait se retrouver avec des erreurs dans le cas d'une page demandant le chargement d'un fichier externe un peu lourd.
   > La bonne méthode consiste à ne lire le fichier js qu'une fois la page html complètement chargée (voir paragraphe suivant).

3. **La bonne pratique**
   On doit attendre que toute la page soit affichée pour que le script s'exécute. Il existe une instruction nommé `window.onload` qui signifie justement « après le chargement de la page ». On lui affectera une fonction qui sera ainsi appelée dès que la page aura été chargée.

   Modifier le fichier script.js :

```
function initialisation(){
   console.log("test")
   console.log(document.body)
}
window.onload = initialisation
```

   Regarder dans la console : l'élément body n'est plus égal à null.

Débuggage : très important en javascript
Dans tout le travail qui va suivre, il faudra regarder les messages d'erreur qui seront écrits dans cette console. Une erreur en javascript est toujours décrite dans la console avec comme précision importante le numéro de la ligne qui pose problème.
On pourra se servir de messages avec console.log pour identifier l'erreur.
ATTENTION : javascript est sensible à la casse (comme python). Remplacer une minuscule par une majuscule dans un nom de variable cause une erreur.

c) Actions sur la page

1. **Premières actions pour comprendre le principe**

   Effacer les deux lignes avec `console.log`.

   La page HTML dispose de boutons. Nous allons montrer comment il est possible de modifier la page par une action simple sur l'un des boutons.

   L'idée est qu'au clic sur un bouton, son id s'affiche dans la balise <nav>.

   (a) Ajouter du texte dans une balise

   Dans le fichier script.js, créer la fonction `ajoutNav` :

```
function ajouterNav(elt) {
   document.querySelector('#ajoutNav').innerHTML = elt.id
}
```

   - Cette fonction `ajouterNav` admet un elt HTML comme argument (ce sera un bouton, mais l'argument permettra de l'appliquer à plusieurs boutons) ;
   - `document.querySelector('#ajoutNav')` est l'élément HTML d'id ajoutNav (# indique que c'est un id, on aurait mis . pour une classe et rien pour une balise).
     Comme ajoutNav est un id, on aurait pu utiliser `document.getElementById('ajoutNav')` qui aurait eu le même effet.
   - `innerHTML` est une méthode dont la valeur égale au contenu HTML de l'élément d'id ajoutNav ;
   - `elt.id` est l'id de l'élément placé en argument.

   Au final cette fonction ajoute au span d'id ajoutNav (présent dans la balise <nav>) le texte égal à l'id d'un élément de la page (ce sera un bouton ici mais ce pourrait être n'importe quel élément HTML).

   (b) Action sur un bouton

   Dans ce premier exemple on va agir au clic sur le premier bouton (celui sur lequel est écrit "Variables" et dont l'id vaut "btnVariables"). Quand on cliquera sur ce bouton, le texte "btnVariables" (c'est-à-dire l'id du bouton) sera écrit dans le span d'id ajoutNav.
   Cela pourrait être fait de 3 façons :

   - dans la page HTML, en ajoutant l'attribut **onclick** dans la création du bouton avec pour valeur `"ajoutNav(this)"` où `this` jouera le rôle du bouton sur lequel est cette action.
     Voilà ce que cela donnerait :
     `<button id="btnVariables" value="variables" onclick="ajoutNav(this)">Variables</button>`
     Cette méthode est cependant à éviter car l'objectif est de ne pas mettre de javascript dans la page HTML.
   - Dans la fonction initialisation du fichier script.js :
     `document.querySelector('#btnVariables').onclick = function() { ajoutNav(this)}`
     Cette ligne donne une propriété onclick à l'élément d'id btnVariables. Cette propriété est une fonction (dite fonction anonyme dans ce cas) qui appelle la fonction ajoutNav avec pour argument `this` c'est-à-dire l'élément sur lequel se situe l'action (ici le bouton).
   - Dans la fonction initialisation du fichier script.js :

```
document.querySelector("#btnVariables").addEventListener("onclick", function() {
  ajoutNav(this)
}
```

     C'est assez proche du précédent exemple, mais ce `addEventListener` permet d'ajouter plusieurs actions à un onclick (pratique pour des projets assez conséquents).

   Dans les deux derniers cas, `document.querySelector("#btnVariables")` pourrait être remplacé par `document.getElementById("btnVariables")` car `"btnVariables"` est un id.

   Modifier la fonction initialisation du fichier script.js pour qu'un cliquant sur le premier bouton apparaisse « btnVariables » dans la page (à côté ou en-dessous des boutons). *Utiliser la deuxième méthode.*

   Dans la fonction `ajouterNav`, remplacer `elt.id` par `elt.value`. Expliquer ce nouvel affichage.

(c) Action sur tous les boutons

Avant la fonction initialisation, on va créer la liste (de type Array en javascript) des id des boutons :

```
const btns=["btnVariables", "btnBoucles", ...]
```

J'utilise `const` ici car ce tableau ne sera pas modifié (sinon j'aurais utilisé let).

La ligne `document.querySelection...` de la fonction initialisation sera intégrée dans une boucle permettant de donner l'action à tous les boutons du bandeau de navigation.

L'action devait être effectuée sur chacun des boutons et leur liste étant connue, on peut utiliser une boucle pour (voir sa syntaxe dans la partie a)).

```
for (let i=0;i<btns.length;i++){
   ...
}
```

Cette fois-ci, ce ne sera pas `"#btnVariables"` mais `"#"+btns[i]` pour que l'action soit sur chaque bouton.

Que se passe-t-il quand on clique sur chaque bouton de la balise <nav> ?

(d) Deux autres événements sur les boutons

Dans le code précédent, remplacer `onclick` par `onmouseover`.

Quel changement cela produit-il ?

Créer la fonction suivante :

```
function enleverNav() {
   document.querySelector('#ajoutNav').innerHTML = ''
}
```

Cette fonction sans paramètre servira à vider le contenu HTML du span d'id "ajoutNav".

En s'inspirant de ce qui a été fait avec `onmouseover`, ajouter sur chaque bouton la méthode `onmouseout` qui appellera la fonction `enleverNav`.

Ainsi au survol du bouton, son nom s'affiche, mais quand la souris n'est plus au-dessus, le nom disparait.

Recréer l'événement `onclick` sur chaque bouton (donc toujours dans la boucle) et y mettre :
`this.style.color = '#0000EE'`.

Que se passe-t-il ?

2. **Pour aller plus loin**

Ajouter les deux fonctions suivantes dans le fichier javascript :

```
function PageVide() {
   //On vide toutes les sous-partie et les articles parents
   for (let i = 0; i < btns.length; i++) {
      let nomPartie = document.getElementById(btns[i]).value
      document.getElementById(nomPartie).style.display = "none"
      let leParent = document.getElementById(nomPartie).parentElement
      if (leParent.style.display !== "none") {
         leParent.style.display = "none"
      }
   }
}
function affichage(titre) {
   //titre est le nom de l'id de la sous-partie à afficher
   document.getElementById(titre).style.display = "block"
   let articleParent = document.getElementById(titre).parentElement
   articleParent.style.display = "block"
}
```

Au début de la fonction initialisation, appeler la fonction `PageVide`.

Au clic sur chacun des boutons, on appelle la fonction `PageVide` puis la fonction affichage avec pour argument `this.value` (regarder sur chaque bouton ce à quoi correspond `value`).

Remarque : dans ces fonctions, j'ai utilisé `document.getElementById(...)` mais j'aurais pu utiliser comme auparavant `document.querySelector('#...)`.