Technologies du Web

JavaScript (Ne pas confondre avec le langage Java)
Langage de programmation interprété par le navigateur. Ce code est donc visible et copiable par l'internaute.
Il ne doit donc pas contenir de mots de passe ni de « secrets » de fabrication !!!

HTML produit des pages statiques, figées.
Le JavaScript permet de modifier dynamiquement une page en fonction d'évènements liés au clavier, à la souris, à l'horloge système, …
Les technologies complétant JavaScript sont par exemple :

- JSON, XML : Façon d'organiser des données pour les échanger avec des serveurs.
- Ajax / Fetch : pour accéder à un serveur de façon asynchrone
- Promise / Callback : Pour créer des fonctions asynchrones
- WebSocket / SocketIO / SSE : pour accéder à un serveur de façon asynchrone avec mode PUSH
- NodeJS : Le JavaScript côté serveur

Depuis 1997, JavaScript suit la normalisation ECMAScript qui enrichit les fonctionnalités chaque année.
Grace au JavaScript, on peut dynamiquement (et sans rechargement) modifier l'apparence de la page web, en fonction **d'événements** (déplacement souris, clic, redimensionnement fenêtre, …).

Un JavaScript peut effectuer des calculs, des analyses de chaines de caractères, des opérations répétitives (boucles), des tests et des affichages.

Le JavaScript sait également utiliser le **DOM** (*Document Object Model*) qui permet de « voir » la page html comme un ensemble d'objet, avec leurs méthodes et leurs propriétés.

Et JQuery ??
C'est une bibliothèque extrêmement populaire qui a beaucoup amélioré les fonctionnalités du JavaScript dans les années troubles ou tout n'était pas encore bien fini. Il offre une écriture plus concise du code.

Par exemple, pour sélectionner l'ensemble des balises de classe 'toto', on écrit :

En JQuery :	$('.toto')
En JavaScript :	document.querySelectorAll('.toto') ;

MAIS : JavaScript s'est beaucoup bonifié ces dernières années, ce qui rend JQuery non nécessaire. Ce n'est que l'avis d'un prof… Beaucoup de développeurs continuent de l'utiliser par habitude.

Sommaire

1 Où placer le JavaScript ?

Un JavaScript peut être :

- intégré n'importe où dans la page html en insérant une balise `<script> </script>`
- ou placé dans un fichier à part (Ex : monScript.js). Dans ce cas, on ajoute le lien avec le script en ajoutant le code :

```
<script src="monScript.js"></script>
```

  - dans le <head> (s'il s'agit de bibliothèques)
  - ou à la fin du <body> si le script touche à la structure du HTML

2 Premier exemple : *getElementById / querySelector*

TESTEZ : Voici un script exécuté au chargement de la page :

```
<!-- Zone HTML -->
      <p id="demo" > Bonjour<br>Nous sommes le : </p>

<!-- Zone JS -->
<script>
 let ladate = new Date();
 let msg = ladate.getDate()+"/"+(ladate.getMonth()+1)+"/"+ladate.getFullYear();

 document.getElementById("demo").innerHTML += msg;
</script>
```

Bonjour
Nous sommes le : 29/11/2021

Explications de la zone <script> :

Les 2 premières lignes servent à créer une variable contenant « Nous sommes le : 29/11/2021 » avec bien sûr la date du jour …

- La date du jour utilise un **objet** *Date* du JavaScript. La date obtenue n'est pas au format JJ/MM/AA.
- On remarque qu'une variable est déclarée avec « let » mais n'est pas typée.

```
let ladate = new Date();
```

- Pour la mettre au format « français » on extrait de l'objet ***Date*** le jour, le mois et l'année et on « colle » tout ça par concaténation (+) dans une variable :

```
let msg = ladate.getDate() + "/" + (ladate.getMonth()+1) + "/" + ladate.getFullYear();
```

- Le signe « + » permet de concaténer les chaînes de caractères

3e ligne : Accès au DOM : Pour ajouter du texte dans la balise HTML du document, on utilise le code :

```
document.getElementById("demo").innerHTML += msg;
```

- **`document.getElementById`**`("demo")` : dans le document, on recherche la balise dont l'attribut « id » est « demo ». Ici ce sera la balise : `<p id="demo"> Bonjour </p>`
- A partir de là, on atteint la propriété « .innerHTML » de la balise, et on modifie sa valeur en y ajoutant (+=) le message contenu dans la variable « msg ».
- InnerHTML représente la zone « data » d'une balise (<p>, , …) Dans notre exemple : Bonjour
- Au départ, il y a « Bonjour » dans cette zone. Le texte « Nous sommes le : 29/11/2021 » est ajouté (+=) par le script.
- C'est de la programmation OBJET : objet « document », méthode « getElementById », propriété « innerHTML »

A SAVOIR : En HTML5/JS ECMAScript, on peut remplacer :

```
document.getElementById("demo").innerHTML += msg ;
```

par :

```
document.querySelector("#demo").innerHTML += msg ;
```

ou même :

```
demo.innerHTML += msg;
```

Remarque : une balise possédant un attribut « id » est directement un objet accessible dans le DOM.

3 *CALLBACK* : Déclencher un script sur un évènement : *addEventListener*

Différentes façons de déclencher sur évènement (exemple de l'évènement *click*) :

Méthode HTML : ~~<*element onclick="myScript()"*>~~ OBSOLETE !!!

Méthode « DOM » (DOM 0): ~~*objet*.onclick = function(){*myScript*};~~ OBSOLETE !!!

(DOM 2): *Objet*.addEventListener("click", *myScript*);

Exemple : TESTEZ ce script exécuté sur un évènement de la souris : quand on clique sur le bouton, le message change :

```
<p id="demo" > Bonjour </p>
<button id="maDiv"> ALLEZ! </button>

<script>
        maDiv.addEventListener('click',mon_callback);

        function mon_callback()
        {
            document.querySelector("#demo").innerHTML="Au revoir";
        }
</script>
```

Explications : Ici, on utilise la méthode « *addEventListener* » qui permet d'associer à l'objet « maDiv » une réaction à l'évènement « click » : en cas de « click », on exécute la fonction « `mon_callback`».
On dit d'une fonction qu'elle est de type « CALLBACK » si elle est appelée par un gestionnaire d'événement.
Le gestionnaire d'évènements est invoqué par la méthode « addEventListener ».

« maDiv » est un objet créé automatiquement grâce à l'attribut « id » de la balise <button>. Pour jouer la sécurité (certains navigateurs n'ont pas encore cette fonction) on peut choisir de faire un vrai *getElementById* ou un *querySelector* :

```
let maDiv = document.querySelector("#maDiv");
```

ou

```
let maDiv = document.getElementById("maDiv");
```

IMPORTANT :
Une fonction CALLBACK se déclare avec le mot **function,** comme pour une fonction classique.
Cependant, il n'est pas possible de transmettre des arguments entre parenthèses. D'ailleurs, on ne met pas les parenthèses lorsqu'on indique son nom dans « addEventListener ».

Par contre, on peut indiquer dans les parenthèses de la fonction callback le nom d'une variable qui contiendra tous le contexte du gestionnaire d'évènement. Cette fonctionnalité sera vue plus tard.

4 Exploiter un objet du DOM

Avec ***getElementById*** ou ***querySelector***, on obtient 1 objet qui représente 1 balise ciblée.
On peut ensuite modifier les propriétés de cette balise comme on veut, par JavaScript.
On vient de le faire en touchant à la propriété *.innerHTML*.
Allons plus loin.

4.1 Sélection multiple et tableau d'objets : querySelectorAll

Saisissez le code suivant :

```
<!DOCTYPE html>
<html>
  <head>
     <style>
          div { width:100px; height:50px; background-color:grey; margin:10px;}
     </style>
  </head>

  <body>
     <div> poire </div>
     <div> orange </div>
     <div class='ici'> pomme </div>
     <div> banane </div>
     <button id='go'>OK</button>
  </body>

  <script>
     go.addEventListener('click', action);

     function action()
     {
          let divIci = document.querySelector(".ici");
          let listeDiv = document.querySelectorAll("div");

          divIci.style.backgroundColor = 'blue';

          console.log('divIci:', divIci);
          console.log('ListeDiv:',listeDiv);
     }

  </script>
</html>
```

Explications :

- Déjà, on constate qu'il y a 3 langages dans ce code : HTML (orange), CSS(vert), JavaScript(bleu) … C'est pour l'exemple et la page est assez courte ! Dans la pratique, on créera 3 fichiers distincts comme on a appris.
- Pour mettre l'événement « click » sur le bouton, on utilise l'objet « go » auto généré par le navigateur à partir de l' « id = go » de la balise <button>
- Pour la avec un attribut « class » : On utilise **querySelector** pour l'atteindre. Le **point « . »** indique que l'on cherche un attribut de type « class ». C'est la même notation que pour le CSS.
- Avec **querySelectorAll**, on recherche TOUTES les balises de type .
- Et avec les **console.log**, on affiche les 2 objets créés. On va regarder dedans ! Pour cela activer le mode « **Développeur** » de votre navigateur en appuyant **sur F12**

TRAVAIL :

1. Observez dans la console les 2 objets affichés :
   Le premier (divICI) représente un objet simple du DOM : on peut y trouver ses propriétés de style, et son contenu (innerHTHM)
   Le deuxième (listeDiv) est un TABLEAU d'objets du DOM
2. TESTEZ : Pour modifier 1 des éléments du TABLEAU, il faut préciser l'index de l'objet dans le tableau :

```
listeDiv[0].style.backgroundColor = 'red';
```

   Le symbole [0] indique qu'on veut utiliser le 1er élément du tableau (la numérotation commence à 0).

4.2 Parcourir un tableau d'objets

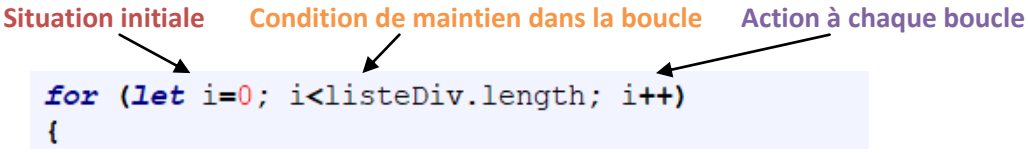
Dans l'exemple précédent, on a modifié 1 élément du tableau.
On va maintenant modifier TOUS les éléments du tableau, même si on ne sait pas forcément combien il y en a.
Pour ça on utilise une BOUCLE pour parcourir TOUS les éléments du tableau jusqu'à la fin.
En JavaScript, il existe la boucle WHILE et la boucle FOR. Les 2 peuvent s'utiliser avec le même résultat.

Par exemple testez avec la boucle FOR :

```
for (let i=0; i<listeDiv.length; i++)
{
    listeDiv[i].style.backgroundColor = 'red';
}
```

Explications :

- La boucle FOR demande 3 informations :



  Donc ici :
  - Situation initiale : créer une variable i initialisée à 0
  - Condition de maintien dans la boucle : tant que « i » plus petit que le nombre d'éléments du tableau
  - Action à chaque boucle : incrémentation de « i »
    L'action concerne généralement la variable de boucle « i ». Pour les autres actions, on utilise les accolades { }
- Dans les crochets [ ] on retrouve l'index du tableau, la variable « i » qui varie de 0 à 3 dans notre exemple, puisque la boucle va s'interrompre quand i = 4 (listeDiv.length donne 4 puisqu'il y a 4 div dans la zone HTML)

Travail : Par JavaScript on voudrait changer la couleur des « div ». On va mettre des couleurs différentes dans chaque div. Les couleurs seront dans un tableau que vous allez créer et remplir :

```
let couleur = ['blue', '#ff2300', 'pink', 'lightgrey', 'green'];
```

Modifiez le code pour que la couleur sélectionnée dans le tableau change pour chaque div.

5 Exploiter un objet « événement » *CALLBACK*

Lorsqu'un évènement est déclenché, il est possible d'en extraire des informations utiles pour le script, par exemple l'objet cible (*target*) qui est à l'origine de l'évènement (*e*) : `e.target`

Ce peut être aussi une information concernant l'évènement lui-même.

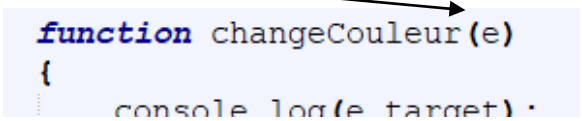
TESTEZ : On vous donne le fichier « callback.html »
Encore une fois, on a « mélangé » du HTML, CSS, JavaScript…

Vous êtes en mesure d'analyser le code :

- Ligne 21 : déclaration d'une liste (tableau) d'objets ayant l'attribut *class='rond'*
- Ligne 22 : déclaration d'une liste de couleur (tableau)
- Lignes 25-28 : chaque objet de *class='rond'* est associé à l'événement « *mouseover* » et exécutera la fonction *callback* nommée **changeCouleur**

C'est maintenant qu'il y a nouveauté : La façon de déclarer la fonction : on y ajoute un argument :

Objet « e » **auto généré** par le gestionnaire d'événement au moment du « *mouseover* »



Lorsqu'une fonction est appelée par *callback*, elle reçoit un objet qui contient le « contexte » de l'appel fonction.
Dans cet objet, on trouve par exemple le nom de l'événement, le nom de la balise concernée (la *target*) , …

A TESTER : Il est intéressant de voir le contenu de l'objet événement : `console.log(e) ;`
Que voit-on dans la zone « *target* » de l'objet « e » ?
Testez la page : normalement la couleur change à chaque passage souris.
Mais au bout d'un moment, ça ne fonctionne plus : On dépasse la capacité du tableau de couleur.
Pour remédier, on va utiliser :

6 Le test IF( )

La fonction *if* du JavaScript fonctionne de façon très classique (comme en C/C++/PHP).

Dans l'exemple précédent, la variable « coul » est l'index du tableau de couleurs.
Ajoutez au bon endroit un test :
si « coul » est plus grand que le nombre de couleurs du tableau, on remet « coul = 0 »

Ecrivez cette solution !

7 Divers…

7.1 Evénement « chargement » *onload (ancien mais toujours utilisé…)*

```
window.onload = function(){
      document.body.style.color = "blue";
}
```

A chaque rechargement de page le script met en texte bleu tout ce que contient le <body>
La *méthode* onload existe pour d'autres objets, les images par exemple.

7.2 Des POPUPs :

Les fonctions *alert(), confirm(), prompt()* servent à ouvrit des fenêtres popup pour un message ou une saisie.

7.3 Synthèse :

Une petite synthèse JavaScript :

- Vous notez que JavaScript est un langage de programmation qui manipule des OBJET.
  Un OBJET est une « boite » qui contient des données et des fonctionnalités pour traiter ces données. On appelle cela des *attributs* et des *méthodes*.
- JavaScript considère que votre page HTML est une collection d'OBJETS (DOM : *Document Object Model*). La page est un objet, les balises sont des objets, les évènements sont des objets.
- L'objet « document » représente l'ensemble de votre page html.
- Les méthodes « querySelector() » et querySelectorAll() » permettent d'accéder aux autres objets du document. Entre les ( ) on indique le « sélecteur CSS».

  Liste des sélecteurs css :
  https://developer.mozilla.org/fr/docs/Web/CSS/S%C3%A9lecteurs_CSS
  Discussion comparée entre JavaScript et JQuery :
  https://www.alsacreations.com/article/lire/1445-dom-queryselector-queryselectorall-selectors-api.html

  Sélecteurs CSS courants : "nom_balise" ".nom_classeCSS" "#attribut_ID"

- La méthode « querySelectorAll() » fournit une **collection d'objets** qui s'apparente à un « tableau ». On utilise généralement une boucle (for, while, …) pour parcourir tous les objets, ou un *iterator*.
- L'objet « évènement » est généré automatiquement et contient toutes sortes d'informations utiles.

Le navigateur possède un mode « débug » qui permet de « voir » les erreurs dans le JavaScript et d'afficher le contenu d'un objet (ou d'une variable simple)
Pour afficher la fenêtre de « debug », sur FireFox : clic-droit → Examiner l'élément → Console
Pour afficher une variable dans la Console : console.log(*nom_de_la_variable*)
Exemple : console.log(e) ;