Complément Web – CM1 : Les bases du JS

Objectifs du module

- Connaître les concepts de base du JavaScript
- Utiliser le JavaScript pour dynamiser et manipuler vos pages web
- Créer des formulaires riches

Evaluation

Un TD noté de 4h en fin de module où vous ferez usage de tout ce que vous avez vu pendant le cours. Supports, TDs précédents, et ressources internet seront autorisés.

I/ Une introduction

A) Qu'est-ce que le JavaScript ?

Le JavaScript a été créé initialement en 1995 pour rendre le contenu sur internet plus « vivant » en rajoutant notamment des interactions côté client (navigateur).

Les programmes dans ce langage sont appelés des scripts, et peuvent être écrits directement dans une page HTML, et sera exécuté automatiquement au chargement de la page. Ces scripts sont fournis et exécutés en texte brut **; il n'y a donc pas de nécessité de compiler/préparer son code JavaScript pour l'exécuter**. Tout script écrit en JavaScript est **sensible à la case**.

Au jour d'aujourd'hui, la portée du JavaScript a grandement évolué au-delà d'un simple langage de scripts pour navigateur, et est considéré comme un langage de programmation mature et polyvalent. En effet, le JavaScript ne s'exécute pas seulement dans un navigateur, mais peut également être utilisé sur un serveur, ou plus généralement, n'importe quel appareil disposant d'un programme spécial dénommé « un moteur JavaScript ». Dû à cette explosion en popularité, le JavaScript est gouverné par une spécification appelé « **ECMAScript** » utilisé par tous les grands moteurs JavaScript existants.

Un moteur JavaScript est compliqué par sa nature, mais dans les grandes lignes, il fonctionne ainsi :

1) Le moteur lit un script et parse son contenu

2) Il compile le code contenu dans le script à la volée en langage machine

3) Il exécute le code

Aujourd'hui, ces moteurs sont tellement avancés et optimisés que ces étapes sont réalisés en parallèle, ce qui rend ce processus virtuellement instantané (selon la complexité du script bien entendu)

B) Les possibilités du JavaScript dans un navigateur

Les capacités du JavaScript vont grandement dépendre de l'environnement dans lequel il s'exécute ; le JavaScript dans un navigateur n'aura pas accès à toutes les fonctionnalités possible avec un serveur NodeJS (moteur de JavaScript côté serveur) et vice versa. Ici, dans ce cours, nous allons principalement nous intéresser au JavaScript et ses possibilités dans un contexte web. Dans un navigateur, le JavaScript permet par exemple :

- De rajouter du contenu HTML à la volée dans la page, de changer du contenu existant, ou encore de changer le style de certains éléments
- Réagir à des actions utilisateurs : un clic sur la page, bouger la souris, scroller sur la page, appuyer sur une touche...
- Envoyer des requêtes à un serveur distant sans devoir recharger la page (Requêtes AJAX)
- Stocker des données côté client

Le JavaScript dans un contexte navigateur est **limité** pour la sécurité des utilisateurs. En effet, le JavaScript est téléchargé et exécuté automatiquement lors de la visite d'une page web, et ne pas limiter les possibilités du langage serait une porte ouverte sur les machines de tous les utilisateurs qui visitent une page web. Ainsi, dans un contexte navigateur, le JavaScript ne peut pas :

- Accéder au filesystem de l'utilisateur (lire/écrire les fichiers présents sur la machine de l'utilisateur). *NB : Il est tout de même possible de bénéficier d'un accès limité à un fichier via un input de fichiers HTML par exemple, mais ceci se fait du coup après une action spécifique d'un utilisateur*.
- Accéder aux fonctions bas niveau de votre système d'exploitation (pour lire les informations sur le hardware de la machine par exemple).
- Accéder librement à certains périphériques de l'utilisateur, comme son micro ou sa webcam. Pour cela, l'utilisateur doit explicitement donner l'autorisation avant.
- Chaque fenêtre ou onglet du navigateur sont isolés et ne peuvent se connaître/interagir entre-deux. Ceci empêche le site https://jevaistehacker.com/ de lire les mails d'un utilisateur qui aurait par hasard un onglet https://mail.google.com/ ouvert à côté. *NB : Il existe une petite exception à cette règle, mais il s'agit d'une situation très spécifique qui reste confinée au même site sur un même domaine*.
- Le JavaScript permet de communiquer avec un serveur distant via des requêtes AJAX (détaillés plus tard dans ce cours). Celles-ci fonctionnent très bien lorsqu'elles vont sur le même domaine, mais sont vite handicapées lorsqu'elles tentent de communiquer avec un serveur sur un autre domaine, sauf si cet autre domaine autorise explicitement les requêtes provenant de serveurs tiers. Ceci empêche le site https://jevaistehacker.com/ d'envoyer des requêtes vers https://facebook.com/ au nom de l'utilisateur en utilisant sa session.

C) En bref...

Le JavaScript est un langage complet et puissant qui a réussi à évoluer à partir d'un langage très simple initialement qui permettait de rajouter des interactions simples dans une page web. Il est devenu aujourd'hui un pilier de référence polyvalent dans le développement web, permettant de créer de véritables applications, et a même transcendé le navigateur pour être utilisé dans d'autres contextes. Dans le domaine du développement web d'aujourd'hui, c'est un langage quasiment essentiel.

Dans un navigateur, le JavaScript permet d'interagir pleinement avec sa page en html/css de façon assez simple, et est omniprésent dans tous les navigateurs modernes sur tous les supports (Windows, Mac, Linux, Android, iOS...).

Dû en grande partie à son historique, il y a généralement plein de façons différentes de faire les mêmes actions en JavaScript. Les principes restent les mêmes, mais il se peut, surtout dans des codebases plus anciennes, de trouver des vieilles façons de faire qui ne sont plus trop d'actualité. Ce cours va principalement se focaliser sur du javascript dit « moderne », mais ne devrait pas trop nuire à une compréhension de l'ancienne façon de faire.

Enfin, le JavaScript possède des fonctionnalités que l'on pourrait considérer « orienté objet », mais on parle plutôt de de « prototyping-based Object Oriented Programming language ». Les spécificités des objets en JavaScript seront détaillés plus tard.

II/ Mise en œuvre

A) La balise <script>

Un script JavaScript peut être inséré presque n'importe où dans une page HTML en utilisant la balise <script> dans votre code HTML. Par exemple :

```
<!DOCTYPE html>
<html>
  <body>
    <p>Avant le script</p>
    <script>
        alert("Hello World !") ;
    </script>
  </body>
</html>
```

Ces balises peuvent être insérées dans le <head> (ils seront exécutés avant que la page ne s'affiche) </head> ou dans le <body> (ils seront exécutés séquentiellement dans la page). Il est également possible de créer des scripts avec une source externe :

```
<script src="/chemin/vers/script.js"></script>
```

Cette source externe peut également être une URL complète vers un autre domaine. Vous n'êtes pas limités du nombre de balises script que vous pouvez insérer dans la page. Celles-ci seront toujours parsées et exécutées an moment où elles sont déclarées.

A noter que la méthode privilégiée est d'inclure son script dans un fichier .js externe. En plus de clairement découper votre code entre HTML/JS, le fichier .js sera téléchargé séparément de votre fichier HTML, et pourra ainsi bénéficier d'être mis en cache séparément de votre fichier HTML, ce qui aura pour avantage de réduire la bande passante et améliorer la rapidité de chargement de votre page.

B) Le JavaScript « moderne »

Le JavaScript a grandement évolué sans gros problèmes de compatibilité, l'idée principale était d'ajouter des fonctionnalités sans changer les fonctions et mécanismes existant. Le bénéfice de cette approche était de ne pas casses du code existant au fur et à mesure des mises à jour des spécifications, et par extension, des

navigateurs web. Le gros inconvénient par contre est que n'importe quelle erreur ou imperfection dans la spécification de JS sont désormais coincés dans le langage à jamais.

En 2009, une nouvelle version d'ECMAScript baptisée ECMAScript5 (ES5) est apparue, et a introduit une série de nouvelles fonctions tout en modifiant quelques fonctions existantes. Afin de préserver la fonctionnalité des vieux sites, ces fonctions sont désactivées par défaut ; il faut les activer spécifiquement avec une directive spéciale "use strict".

Dans ce cours, nous allons spécifiquement utiliser que du JavaScript dit moderne, donc il convient d'utiliser le use strict dans tous vos scripts.

Pour utiliser le use strict, il convient simplement de mettre "use strict" en haut de vos scripts

```
"use strict";

// Votre code ici
```

Attention, cette déclaration doit être mise impérativement en tête de fichier pour qu'elle soit prise en compte.

C) Les fonctions de base du JavaScript

1) Les variables

Le JavaScript est un langage non typé. Cela signifie qu'une variable déclarée peut prendre n'importe quelle valeur à n'importe quel moment. Une variable peut devenir un string, un number, ou même encore une fonction.

Cela implique un lot d'avantages, mais surtout d'inconvénients. Une variable peut évoluer de multiples façons dans un programme complexe, et ne pas avoir la sécurité du typage peut entraîner un lot d'erreurs inattendues. En contrepartie, la liberté d'affectation peut s'avérer pratique dans certaines situations, mais toujours au détriment de la lisibilité de votre code.

Le mot clef « let »

Une variable se déclare avec le mot clef « let ». Exemple d'utilisation

```
let mesage; // Déclaration de la variable. Sa valeur initiale sera "undefined"
message = "Hello world !"; // Affectation du string "Hello world !" à la variable message
alert(message); // Utilisation de la fonction intégrée "alert" du navigateur pour afficher le contenu de la variable "message"
```

Une variable let ne peut être utilisée avant qu'elle ne soit déclarée, et elle sera confinée au block dans laquelle elle est déclarée. Par exemple :

```
alert(variable1); // Cette ligne va crasher car variable1 n'est pas encore déclarée
let variable1 = "test";
```

ou encore :

```
let userPasswordInput = prompt(); // Demande à l'utilisateur de saisir un texte
if (userPasswordInput === "password") {
  let outputMessage = "Votre mot de passe est nul";
}
outputMessage = "Votre mot de passe n'est pas le mot de passe le plus commun";
alert(outputMessage);
```

L'exemple ci-dessus va également crasher car la let « outputMessage » est déclarée dans un if, et ne pourra qu'être utilisé à l'intérieur de celui-ci. Si on voulait corriger cela, il faudrait déclarer la variable « outputMessage » juste avant le if.

NB : Il existe également le mot clef « var » pour déclarer une variable. Celle-ci est plus ancienne que le « let » et vient avec tout un lot de règles qu'il faut connaître. Ainsi, utiliser le mot clef « var » sans connaître les effets secondaires potentiels peut avoir des répercussions sur votre script. En JavaScript moderne, il est déconseillé d'utiliser le « var » vu que ses règles plus obscures sont une nuisance à la compréhension du code en général ; à l'inverse du « let » qui fonctionne de façon beaucoup plus intuitive et logique.

L'utilisation de const

Une « const » est une variable alternative très similaire au fonctionnement du let (au niveau de sa portée, donc uniquement utilisable après qu'elle ne soit déclarée, et uniquement confinée au bloc où il est créé).

Une const à l'inverse d'une let ne pourra être réaffectée par une autre valeur par la suite. Par exemple :

```
const myBirthday = "11/08/1993"; // Déclaration de la constante
myBirthday = "01/01/2020"; // <- Le script va crasher car une constante ne peut pas être réaffectée.
```

Il convient d'utiliser ce mot clef pour toutes les données qui n'auront jamais vocation à changer. Cela permet de les protéger et de s'assurer qu'elles ne seront pas réaffectée au long de l'exécution de votre script.

Nommage des variables et quelques conventions…

Le nommage des variables est assez libre, mais sont tout de même soumises à quelques règles :

- Un nom de variable doit commenter par une lettre (majuscule ou minuscule) ou un « _ »
- Un nom de variable peut comporter des lettres, chiffres, et le caractère « _ » (donc pas d'espaces)
- Les noms de variables doivent être différent des mots réservés (if, else, let…)

Ces règles laissent un certain nombre de libertés dans le nommage des variables. Cependant, les codebases en JavaScript respectent pour la plupart des conventions assez strictes. Bien que vous puissiez théoriquement nommer vos variables comme vous le souhaitez, il convient généralement de respecter ces quelques règles :

- Une variable (let) doit avoir un nom en camelCase (commencer par une minuscule, et les « espaces » sont remplacés par une lettre majuscule). Par exemple : « let listeClients »
- Une const qui a vocation à être une valeur fixe et déclarée de façon permanente dans votre code est généralement écrite entièrement en majuscules, et les espaces sont remplacés par des « _ ». Par exemple : « const COLOR_ORANGE = "#FF7F00"; »

- Une const qui est le résultat d'un calcul, ou une assignation spécifique variable doit être écrite avec les mêmes règles que le let (en camelCase). Par exemple : « const pageLoadTime = /*valeur calculée du temps de chargement de la page*/; »

Les tableaux

Le JavaScript peut utiliser des tableaux, et dispose d'une série de fonctions pratiques pour les manipuler. Un tableau s'initialise de la façon suivante :

```
let array = []; // Initialisation d'un tableau vide
let array2 = [40, 23, 42]; // Initialisation d'un tableau avec 3 entrées
let array3 = [128, "Ceci est un string", null]; // Initialisation d'un tableau avec des
valeurs différentes
```

Une fois un tableau initialisé, il existe des méthodes pour insérer des données, supprimer des entrées spécifiques, etc. Ces méthodes peuvent être trouvées facilement via des recherches sur les divers sites de documentation, et ne seront pas listées ici pour éviter de saturer les informations.

2) Les commentaires

Comme dans d'autres langages, il est possible d'écrire des commentaires dans un code JavaScript, ou un tronçon de code qui ne sera pas interprété comme du code, et permet donc d'annoter des lignes avec des messages pour aider à la compréhension d'un script, ou d'indiquer des informations supplémentaires.

Ajoutez des commentaires en mettant « // » avant le commentaire. Tout ce qui suit dans la même ligne est un commentaire (comme en C++) Exemple :

```
alert("javascript"); // ouverture d'une fenêtre d'alerte
```

Les commentaires au style C « /* ... */ » sur plusieurs lignes sont utilisés aussi, mais il faut veiller à ne pas imbriquer ces commentaires.

3) Les fonctions

Une fonction peut être définie comme un sous-programme que vous pouvez définir librement. Elles sont particulièrement utiles pour effectuer une action similaire à différents endroits d'un script. Les fonctions sont les briques de construction d'un programme, et permettent à un bout de code en particulier d'être appelé autant de fois que nécessaire sans répétition.

Il existe des fonctions incluses dans le langage (comme « alert("message") »), mais il est bien entendu possible de créer les nôtres.

Déclaration d'une fonction

Dans son état le plus simple, un fonction est déclarée ainsi :

```
function sayHello() {
  alert("Hello!");
}
```

Une fonction peut également être assignée à une variable :

```
const sayHello = function () {
  alert("Hello!");
};
```

Une fois déclarée, une fonction peut être appelée n'importe où elle est disponible avec simplement son nom

```
sayHello();
```

L'exécution de cette fonction affichera une « alert » avec le message « Hello! »

Paramètres d'une fonction

Une fonction peut également disposer de paramètres qui permettent de moduler son fonctionnement :

```
function showMessage(author, text) {
  alert("Message from '" + from + "' : " + text);
}
```

Cette fonction peut être appelée en spécifiant un auteur et un texte, et peuvent être utilisées à l'intérieur de la fonction. On peut appeler une telle fonction avec ses paramètres comme ceci :

```
showMessage("Jean-Michel", "Le JavaScript c'est trop bien!");
```

L'exécution de cette fonction affichera une « alert » avec le message « Message from 'Jean-Michel' : Le JavaScript c'est trop bien! »

Il est également possible de spécifier des valeurs par défaut pour les paramètres d'une fonction :

```
function showMessage(author, text = "Default message") {
  alert("Message from '" + from + "' : " + text);
}
```

Ici, si on appelle la fonction ainsi:

```
showMessage("Jean-Michel");
```

L'exécution de cette fonction affichera une « alert » avec le message « Message from 'Jean-Michel' : Default message ».

Valeurs de retour

Une fonction est également capable de retourner une valeur. Dans son état le plus simple, voici un exemple d'application :

```
function sum(a, b) {
  return a + b;
}
alert(sum(4, 7));
```

Ce script affichera « 11 » dans une modale d'alerte.

D) Le DOM

Le DOM est une abréviation pour « Document Object Model », et c'est par le biais de cet objet qu'il est possible d'interagir avec la page HTML via votre code JavaScript.

Le point d'entrée du DOM pour vos scripts est l'objet « document ». C'est lui qui va vous permettre d'accéder par la suite à tous vos nœuds HTML.

1) Les sélecteurs d'éléments du DOM

Afin d'interagir avec le DOM, il convient de pouvoir effectuer des sélections sur les éléments que l'on souhaite interagir avec/modifier. Les fonctions principales dont vous avez besoin sont les suivantes :

document.querySelector(selector)

Cette fonction est la plus importante et polyvalente. Vous l'utilisez en spécifiant en paramètre de cette fonction un string comportant un sélecteur, comme vous avez déjà l'habitude de faire en CSS. Il retournera le premier élément correspondant Par exemple :

```
<html>
  <body>
    <p>First paragraph</p>
    <p id="p2">Second Paragraph with a specific ID</p>
    <ul>
      <li>First list item</li>
      <li class="odd">Second list item with a custom class</li>
      <li>Third list item with <span>nested</span> elements</li>
      <li class="odd">
        Fourth item with <span>nested</span> elements and a custom class
      </li>
    </ul>
  </body>
  <script>
    document.querySelector("p"); // Retournera le premier élément « p » dans l'HTML ci-
dessus
    document.querySelector("li.odd"); // Retournera le premier élément « li » avec la
classe« odd »
    document.querySelector("#p2"); // Retournera l'élément avec l'ID « p2 »
  </script>
</html>
```

Notez que si aucun élément ne match dans l'HTML de la page, document.querySelector retournera la valeur « null ».

Cette fonction permet donc sélectionner n'importe quel élément unique en fonction d'un sélecteur précis à définir.

Notez cependant que cette fonction peut retourner un seul et unique élément, même si il y a plusieurs éléments qui matchent la query, seul le premier sera retourné.

Par exemple, avec l'HTML de l'exemple précédent, il y a deux éléments avec la classe « odd ». Mais l'exécution de la commande « document.querySelector(".odd") » retournera que le premier élément des deux. Si il y a besoin de faire une sélection multiple, il faut procéder à la fonction suivante :

document.querySelectorAll(selector)

Cette fonction a le même fonctionnement que le document.querySelector, mais retourne quoi qu'il arrive un tableau avec les résultats (qui peut du coup être vide).

Ainsi, en reprenant l'exemple de la section précédente, une exécution de la commande « document.querySelector(".odd") » retournera un tableau avec les deux éléments HTML qui disposent de la class « odd ».

Cette fonction, à l'inverse de la précédente, permet de récupérer plusieurs éléments, et donc d'itérer dessus en fonction du besoin.

Autres fonctions que vous pourrez rencontrer…

Il est possible que vous rencontriez d'autres méthodes (plus anciennes) pour sélectionner des éléments dans le DOM. Ces fonctions sont par exemple

- document.getElementById(id)
- document.getEementByName(name)
- document.getElementByClassName(className)

Ces fonctions ont le mérite d'être clair de par leur nom de méthode, mais sont rapidement plus limitées que les méthodes du querySelector vu précédemment, dans le sens qu'il n'est pas possible de faire des sélecteurs spécifiques de façon simple et concise. Il est de ce fait conseillé d'utiliser les querySelectors à la place de ces anciennes méthodes.

2) Fonctions de base sur les éléments HTML

Dans la section précédente, il a été vu comment effectuer une sélection sur un élément dans notre DOM. Maintenant, nous allons voir quelques fonctions simples pour interagir avec les éléments en question que l'on récupère.

En supposant l'HTML suivant :

```
<!-- HTML -->
<html>
  <body>
    <h1>First Header</h1>
    <p>First paragraph</p>
  </body>
</html>
```

Vous pouvez stocker l'élément h1 dans une const comme ceci :

```
// JS
const h1 = document.querySelector("h1");
```

Et enfin, vous pouvez ensuite exécuter une série de petites fonctions utiles pour faire des interactions simples avec. Voici une petite liste de fonctions simples et utiles que vous pouvez utiliser dès maintenant :

innerHTML

```
h1.innerHTML = "Toto";
```

Cette commande remplace l'entièreté du contenu de l'élément HTML par le string spécifié. A noter que vous pouvez mettre des strings simple, ou même des balises HTML complètes. L'exemple suivant par exemple est valide :

```
h1.innerHTML = "First Header <i>with new markup inside</i>";
```

Vous pouvez donc complètement créer des éléments et des balises qui n'étaient pas du tout présent dans votre HTML initialement.

classList

Une autre fonction très pratique est de pouvoir accéder à l'attribut « class » de votre node HTML. L'objet classList vous met à disposition une petite API simple à utiliser pour interagir avec les classes de votre élément :

```
h1.classList; // Retourne un tableau avec les classes dont dispose l'élément HTML
h1.classList.add(newClass); // Rajoute une classe à l'élément
h1.classList.remove(classToRemove); // Retire une classe à l'élément
h1.classList.toggle(classToToggle); // Rajoute la classe si elle n'existe pas, la supprime
sinon
```

Ces fonctions devraient fournir toutes les fonctionnalités dont vous avez besoin pour manipuler des classes

E) Gérer un évènement « click »

La section précédente a surtout parlé des fonctions de base du langage. Mais pour le moment, sauf afficher des messages de base, nous nous sommes que très peu intéressés aux façons d'interagir avec le contenu de notre page HTML via des actions utilisateur.

Le JavaScript utilise ce qu'on appelle des évènements. Ceux-ci permettent d'intercepter ce que l'utilisateur fait sur la page, et de réagir en fonction des besoins. Ces évènements incluent le click, un appui sur une touche du clavier, ou encore un scroll.

Ces évènements seront approfondis en détail dans un TD futur. Pour le moment, nous allons simplement apprendre à créer et gérer un évènement « click » simple :

```
function sayHello() {
  alert("Hello!");
}
document.querySelector("button").addEventListener("click", sayHello);
```

Dans l'exemple ci-dessus, nous déclarons une fonction « sayHello ». Ensuite, après avoir sélectionné un élément « button » dans notre HT ML (qu'on suppose existe), nous utilisons la fonction « addEventListener ». Le premier argument spécifie l'évènement qu'on veut écouter (ici, le click), et le deuxième argument précise la fonction que l'on souhaite exécuter une fois que l'évènement est capturé.

III/ Notes complémentaires de fin

N'hésitez surtout pas à utiliser la console du navigateur en appuyant sur F12. Cette ressource indique non seulement les éventuelles erreurs que vous introduisez dans vos scripts, mais elle permet également de tester des fonctions à la volée. Habituez-vous d'emblée à utiliser les outils fantastiques disponibles dans les navigateurs pour débugger vos applications.

N'hésitez pas à faire usage de la fonction « console.log() » dans vos scripts lors de vos développements. Cette fonction permet d'insérer un contenu dans la console du navigateur, et peut s'avérer très pratique pour tester le contenu d'une variable, ou vérifier qu'un bout de code s'exécute.

Ce cours est fortement basé sur la ressource libre est open source https://javascript.info/ qui est une documentation fantastique, détaillée, et très bien expliquée de tout l'univers du JavaScript. Ce support que vous lisez est suffisant comme introduction, mais n'hésitez pas à vous intéresser à cette ressource pour aller plus loin !

Par ailleurs, la communauté JavaScript est énorme, et quasiment toutes les réponses aux questions « basiques » sont trouvables avec une recherche Google bien placée. Prenez le réflexe d'effectuer vos recherches en anglais ; les résultats seront beaucoup plus nombreux, récents, et pertinents !