Programmation Web HEIG-VD

JavaScript

Créer des pages web dynamiques et interactives.

JavaScript est un langage orienté objet c'est à dire que ce langage dispose d'objets. Un objet est une entité qui possède des propriétés et peut être manipulée avec des méthodes.
Une méthode est une fonction spécifique à un objet.
La POO (Programmation Orientée Objet) est une manière d'écrire des programmes en utilisant des objets avec leurs propriétés et leurs méthodes !

JavaScript est riche en objets prédéfinis (ou natifs ; fournit par l'environnement de développement).

Fonction

Sorte de sous-programme. Une fonction est composée d'une suite d'instructions qui forment le corps de la fonction. Il est parfois possible de passer des valeurs à une fonction et une fonction peut éventuellement retourner (ou renvoyer) une valeur.

En JavaScript il existe de nombreuses fonctions qui permettent de réduire la quantité de code à faire. C'est ainsi judicieux d'en faire parfois en début de programme pour simplifier par la suite le code. [Par exemple pour aller chercher des types d'éléments dans notre code HTML].

- Il existe plusieurs manière d'écrire une fonction. [Fonction comme en Java ou fonction fléchées].
- Une fonction est une procédure JavaScript, un ensemble d'instructions effectuant une tâche ou calculant une valeur.
- En JavaScript, les fonctions sont des objets de première classe.

Une fonction qui ne renvoie pas de valeur retourne undefined.

En résumé pour une fonction il faut coder :

```
Le nom de la fonction.
Une liste d'arguments à passer à la fonction, entre parenthèses et séparés par des virgules.
Les instructions JavaScript définissant la fonction, entre accolades, { }.
```

Utilité des principaux algorithmes en programmation :

```
Faire des comparaisons entre différents arguments
Effectuer des petits calculs récurrents de manière automatisée
Afficher certains éléments selon certaines conditions
```

Fonction récursive

Une fonction qui s'appelle elle-même est appelée une fonction récursive. D'une certaine manière, une récursion est semblable à une boucle. Une récursion et une boucle exécutent le même code plusieurs fois et s'appuient sur une condition

Tableau

Beaucoup des Data utilisées/traitées sur le web sont sous formes de tableau. Cela permet de les regrouper et les traiter plus efficacement.

Javascript propose une structure de données permettant de stocker l'ensemble de ces données dans une "variable commune" : le tableau.

Objets

JavaScript est conçu autour d'un paradigme simple, basé sur les objets [qui peut être comparé aux objets du monde réel].

Un objet est un ensemble de propriétés.

```
Une tasse est un objet. Elle a plusieurs propriétés comme une couleur et un poids. Ce poids associé à cette couleur rep
```

Une propriété est une association entre un nom (aussi appelé clé) et une valeur.

```
La couleur de notre tasse est associée à la valeur rose.
```

Les propriétés d'un objet sont des variables tout ce qu'il y a de plus classiques, exception faite qu'elles sont attachées à des objets. Les propriétés d'un objet représentent ses caractéristiques et on peut y accéder avec une notation utilisant le point « . »

La valeur d'une propriété peut être une fonction, auquel cas la propriété peut être appelée « méthode ».

```
Si pour définir la couleur de la tasse on dit qu'on peut utiliser seulement du rouge/bleu/vert/noir/blanc on peut faire
```

DOM

###Introduction
Manipuler le DOM permet de récupérer des éléments d'une page et de modifier leur contenu, leurs classes, leurs styles.

- Pouvoir capter les interactions des utilisateurs (clic de souris, remplissage d'un formulaire) et de pouvoir réagir à ces événements.
- Communiquer avec un site web pour récupérer et lui envoyer des données
- Créer du code asynchrone en JS pour paralléliser des requêtes HTTP et délayer l'exécution d'une fonction JS.
  ###Qu'est-ce que le DOM ?
  Document Object Model. C'est une interface de programmation qui est une représentation du HTML et d'une page web qui peremt d'accéder auy éléments de cette page web et de les modifier avec le langage JS.

Le DOM est comme un arbre. Chaque élément peut avoir zéro ou plusieurs enfants, qui peuvent eux-même avoir un ou plusieurs enfants et ainsi de suite.

Dans le DOM il y a toujours un élément racine, le point de départ du document : la balise HTML, qui a comme enfant le head et le body.
###Á quoi sert le DOM ?
Avec une interface de programmation qui permet de parcourir le DOM, on peut interagir avec lui.

Le DOM est ainsi une représentation d'une page HTML et permet d'accéder aux différents éléments de la page grâce au langage JS.

Accéder aux éléments du DOM.

Chaque élément du DOM est un objet JS avec ces propriétés et ces fonctions pour le manipuler.

Avant de pouvoir manipuler le DOM, il faut savoir retrouver les éléments de la page.

Le document est un objet auquel on a accès dans notre code JS. C'est le point de départ du DOM et représente la page entière.

Document.getElementById()
Méthode pour retrouver un élément. C'est la seule qui permet de retrouver facilement un élément précis.

```
<p id="my-anchor"> My content </p>
const myAnchor = document.getElementById('my-anchor')
```

Document.getElementsByClassName)

Cette méthode est identique à la précédente mais sa recherche se fera sur la class des éléments et retournera la liste des éléments correspondent.

Document.getElementsByTagName()

Avec cette méthode, la recherche est pour tous les éléments avec un nom de balise précis. On récupère ainsi toute la liste des éléments correspondants.

Document.querySelector()

Cette méthode est plus complexe et plus puissante. Elle permet de faire une recherche complexe dans le DOM, en mélangeant plusieurs procédés. Ce sélecteur permet de cibler certains éléments.

> Possible de coder pour chercher dans l'élément ayant un certain id, les éléments de types
>
> qui ont pour classe article, afin de récuérer le lien a qui est un enfant direct.

Cette méthode ne renvoie pas une liste des résultats, mais le premier élément qui correspond à la recherche.

Les recherches depuis un élément

Il n'y a pas qu'avec le document qu'on peut faire des recherches. On peut aussi passer par les éléments.
Chaque élément est un objet JS (avec ces propriétés et ces fonctions).
Il existe des propriétés et des fonctions pour parcourir les enfants et le parent de chaque élément.

- element.children : propriété qui retourne la liste des enfants de l'élément
- element.parentElement : propriété qui retourne l'élément parent de celui-ci
- element.nextElementSibling/element.previousElementSibling : ces propriétés permettent de naviguer vers l'élément suivant/précédent de même niveau que notre élément.

Plusieurs fonctions peuvent être utilisés pour récupérer de la matière dans le DOM et les enfants et parents d'un élément particulier peuvent être parcouru.

Modifier le DOM

Une fois qu'on a accéder aux différents éléments du DOM on peut les Modifier

Modifier le contenu d'un éléments

Il existe des propriétés pour modifier directement le contenu de l'élément.

- innerHTML [il faut rentrer du texte représentant du contenu html avec

  ]

- text.content [demande un simple texte qui ne sera pas interprété comme étant du html donc si on mettait des balises paragraphes elles seraient interprétées comme du texte]

On peut ainsi : sélectionner un élément dans notre DOM et venir lui intégrer du texte (sous forme de liste ou autre par exemple)

Modifier les classes

classList : propriété qui permet de modifier directement la liste de classe d'un élément.

La propriété classList fournit une série de fonction qui permet de modifier une liste de classe (car modifier ça veut pouvoir dire ajouter, supprimer, changer ou autre)

Quelques exemples de fonctions qui sont fournis avec la propriété classList :

- add : ajoute la ou les classes spécifiées
- remove : supprime la ou les classes spécifiées
- contains : vérifie si la classe spécifiée est contenue par cet élément
- replace : remplace l'ancienne classe par la nouvelle classe

Changer les styles d'un éléments

style : propriété qui permet de récupérer et modifier les différents styles d'un élément. C'est un objet qui a une propriété pour chaque style existant. Par exemple pour modifier la couleur d'un arrière plan :
element.style.bcakgroundColor = '#000';

Modifier les attributs

setAttribute : fonction pour définir ou remplacer les attributs d'un élément. Elle prend en paramètre le nom de l'attribut et la valeur et ne retourne rien.
Exemple avec elt qui fait référence à un élément de type input :

```
elt.setAttribute("type", "password");   // Change le type de l'input en un type password
elt.setAttribute("name", "my-password");    // Change le nom de l'input en my-password
elt.getAttribute("name");               // Retourne my-password
```

`

Créez de nouveaux éléments

document.createElement : fonction qui permet de créer un nouvel élément du type spécifié qu'on pourra ensuite insérer dans le DOM. Cette fonction prend en paramètre le nom de la balise de notre élément et nous renvoie l'élément nouvellement crée.

Un élément qui est créé avec cette fonction ne fait pas encore partie du document, donc il ne va pas apparaître sur la page. Pour le voir il faudra d'abord l'ajouter en tant qu'enfant à un élément.

Ajoutez des enfants

Plusieurs façons différentes d'ajouter un élément dans la page, mais la plus connue est appendChild.

appendChild : fonction qui permet d'ajouter un élément à la liste des enfants du parents depuis lequel est appelé la fonction. Cette fonction prend en paramètre l'élément à ajouter en tant qu'enfant. L'élément depuis lequel est appelé cette fonction devient donc le parent de notre élément.

Supprimez et remplacez des éléments

- removeChild : fonction pour supprimer un élément. Prend en paramètre l'élément à supprimer du parent et retourne cet élément.

- replaceChild : fonction qui permet de remplacer un élément. Prend en paramètres le nouvel élément ainsi que l'élément à remplacer et retourne ce dernier.

Écouter des événements

L'écoute des événements permet de savoir lorsqu'un utilisateur clic sur un bouton et de réagir à ce clic.

Qu'est-ce qu'un événement ?

Un événement est une réaction à une action émise par l'utilisateur (comme le clic sur un bouton ou la saisie d'un texte dans un formulaire).

> En JavaScript un événement est représenté par un nom (click, mousemove) et une fonction qui s'appelle callback. Par défaut, l'événement est propagé donc si on indique pas à l'événement qu'on le traite, il sera transmis à l'élément parent et ainsi de suite jusqu'é l'élément racine.

La fonction callback est la fonction que nous allons préciser. Elle sera ainsi appelé à chaque fois que l'action que l'on desire suivre (le click) sera exécutée (l'utilisateur a bel et bien cliké).

> La fonction callback qu'on spécifie est ainsi appelé à chaque fois que l'utilisateur cliquera sur l'élément. duquel on désire suivre le clic.

Réagir à un clic sur un élément

Pour pouvoir réagir à l'événement, il faut écouter cet événement.

addEventListener() : fonction qui permet d'écouter un événement. (Click sur un élément). Cette fonction permet d'écouter tout type d'événements.
Elle prend en paramètre le nom de l'événement à écouter et la fonction à appeler dès que l'événement est exécuté.

Oneclick : exemple. Si on veut réagir au clic sur un lien. Il faut commencer par récupérer l'élément qui correspond au lien. Puis, appeler la méthode element.addEventListener('click', 'onClick').

Oneclick est une fonction qu'on va définir et qui sera appelée à chaque fois que l'utilisateur cliquera sur le lien. On peut donc choisir ce qu'on fera : récupérer des informations depuis un serveur, afficher un message ou autre. Le comportement par défaut de l'élément actionné sera tout de même exécuté. Donc même si dans notre fonction

callback on vient afficher un message, le navigateur va ouvrir le lien. De la même manière, si l'utilisateur clic sur un bouton de validation de formulaire, celui-ci sera envoyé. Il est possible de désactiver ce comportement par défaut !

callback : fonction appelée lorsque l'utilisateur clique sur le lien. Elle prend en paramètre le contenu de l'événement qui vient de se produire et met à disposition des fonctions et propriétés. Cet objet contient une fonction preventDefault.

preventDefault : lorsqu'on appelle cette fonction dans notre callback, on demande au gestionnaire des événements de ne pas exécuter le comportement par défaut de notre élément (en l'occurrence la redirection vers une autre page pour un lien)
Si on exécute cette fonction pendant un événement onsubmit sur un formulaire empêchera le formulaire de s'envoyer au serveur par exemple.

stopPropagation() : c'est une fonction de l'objet que notre fonction reçoit en paramètre. Il permet d'empêcher la propagation de l'événement vers son parent. Exemple : si on a un élément pour lequel on veut afficher un message lorsqu'on clique dessus. Mais à l'intérieur de cet événement, il y a un autre élément qui doit nous afficher un autre message lorsqu'on clique dessus. Par défaut, quand on clique sur l'élément intérieur, le message va s'afficher, puis notre élément parent va lui aussi recevoir l'événement du clic et changer le message. Donc pour cela, il faut stopper la propagation de l'événement.

Récupérer des données utilisateurs avec les événements

Les données liées aux événements sont par exemple la position de la souris ou le texte saisi dans un formulaire.

Comprendre ce que sont les données liées à un événement

Lorsqu'on reçoit un événement, la fonction callback reçoit un paramètre contenant les informations sur l'événement. Ces informations sont reçues sous la forme d'un objet et qui dépend du type d'événement reçu.

Chaque événement implémente l'objet Event. Ce qui veut dire que chaque événement a au minimum les mêmes fonctions et propriétés que l'objet Event. Ce qui comprend entre autre :

- preventDefaut() : qui empêche l'exécution du comportement par défaut de l'élément quand il reçoit l'événement.
- stopPropagation() : qui empêche la propagation de l'événement vers d'autres éléments.
- Et d'autres propriétés en fonction du type d'événement.

Détecter le mouvement de la souris

Pour pouvoir détecter le mouvement de la souris, il faut écouter l'événement mousemove. Cet événement fournit un objet de type MouseEvent.

MouseEvent : dès que la souris bouge, notre fonction callback sera appelé avec un paramètre de type MouseEvent, qui contient les données sur le mouvement de la souris.

> L'objet MouseEvent peremt de récupérer entre autre : clientX/clientY qui sont les positions de la souris dans les coordonnées locales (contenu du DOM)
> offsetX/offsetY qui sont la position de la souris par rapport à l'élément sur lequel on écoute l'événement.
> pageX/pageY ; position de la souris par rapport au document entier.
> screenX/screenY : position de la souris par rapport à la fenêtre du navigateur.
> MovementX/movementY : position de la souris par rapport à la position de la souris lors du dernier événement mousmove,

Lire le contenu d'un champ texte

Lorsqu'un utilisateur rempli un champ dans un formulaire il est possible de lire le contenu du champ texte modifié par ce dernier.

Il existe une liste des événements pour pouvoir trouver celui qui peut correspondre à ce que l'on souhaite faire. Ici, c'est détecter les changements dans notre champ texte.

change : événement qui fonctionne avec les éléments de type input, select et textarea. Cet événement est déclenché lorsque le champ perd le focus, c'est-à-dire lorsque l'utilisateur passe à autre chose en cliquant ailleurs et qu'il a fini sa saisie pour ce champ. Cet événement fonctionnne aussi pour les case à cocher (checkbox) et les cases à choix unique (radio).
Il permet donc de détecter que le texte saisi dans le champ a changé.

Pour récupérer la valeur du champ, une fois qu'il a été modifié, il suffit d'accéder à la valeur de l'élément cible. event.target.value. Ici, target correspond à l'élément sur lequel s'est produit l'événement. Donc le champ input. Et ce type d'élément contient une propriété value qui permet de récupérer ou définir la valeur du champ.

input : événement qui fonctionne comme change sauf qu'il est déclenché dès que le contenu du champ est modifié, même si l'utilisateur n'a pas encore fini de saisir ce qu'il souhaite.

Formulaire

> Comportement par défaut (transmettre au backend les données rentrées par l'utilisateur).
> Donc utilisation de preventDefault() pour annuler ce comportement.

```
document.getElementById('notes'). appendChild(p);
```

Pour ajouter en dernier enfant de la liste de notes le p (qui contient ici le texte récupéré du formulaire).

Fetch : pouvoir récupérer de la Data

- Faire des requêtes réseaux. (Une adresse relative est faisable).
- Le Fetch est une requête non-bloquante (donc il attend pas d'avoir les ressources avant de continuer le code) et asynchrone (si on fait deux requêtes on ne peut pas garantir que la première arrive en première et que la deuxième arrive en second).

> Utiliser then comme *lorsque tu auras fini de faire ce que tu dois faire* . Utilisation du principe de promesse.
> Permet d'utiliser la data lorsqu'elle est récupérée.

```
fetch ('hello.txt')
.then(reponse => reponse.text()
.then(txt => console.log(txt)));
```

> Le tuyau (http) n'accèpte que du texte. Lorsqu'on récupère de la data sur un serveur ; sérialisation.
> Puis passage dans le tuyau.
> Et enfin désérialisation pour retomber sur la Data.

> Utilisation du langage de XML. (Qui est faisable en DOM)

Sérialisation : passage en txt.