Programmation Web « Client Riche »
M413 - TD n°1 (séance 1 et 2)

1 Objectifs

Ce TD illustre la partie du cours sur la programmation web et en particulier les concepts liés à la conception d'application riches coté client, c'est-à-dire dans votre navigateur.

Les principaux concepts abordés seront :

- La prise en main des environnements de développement.
- Les principes de base du langage JavaScript.
- La découverte du DOM.

Vos « **livrables** » qui seront à déposer dans **la boîte de dépôt ad-hoc sur le LMS Moodle d'UCA avant la date spécifiée**, seront basés sur l'arborescence fournie et constitués **a minima** des **dossiers assets/, css/, js/** et des **fichiers** suivants :

- Les réponses à chacune des questions posées devrons être rédigées au format « Markdown » dans le fichier « **README.md** » fournie.
- Des fichiers index.html, js/td{x}.js et css/td{x}.css, et plus si nécessaire…

https://fr.wikipedia.org/wiki/Markdown

2 Documentation

Pour ce TD, mais également pour les suivants, sauf mention contraire dans l'énoncé ou si un lien vous est fourni, vous ne devez pas aller chercher des réponses sur internet !

Seuls les sites suivants sont autorisés :

- Le cours sur : https://lms.univ-cotedazur.fr/
- Le site de Mozilla : https://developer.mozilla.org/fr/
- Les sites du W3C : pour la validation HTML5 ou du CSS
  Mais aussi pour la documentation sur le DOM.

3 Prise en main de l'environnement

Pour réaliser ce TD, vous aurez également besoin d'un éditeur de texte pour écrire le code de vos pages. Vous pouvez par exemple utiliser un des logiciels **Brackets**, **Notepad**++, **Eclipse**, etc.

Pensez à sauvegarder régulièrement votre travail et à commenter votre code.

N'hésitez pas à sauvegarder les différentes versions, évolutions d'un même exercice de manière à pouvoir facilement réviser ou le réutiliser par la suite…

4 Les Outils pour parcourir le Document Object Model

De nos jours, des outils de débogage évolués sont intégrés aux différents navigateurs (cf. cours). Ces outils vous seront très utiles pour explorer l'arbre DOM de votre page, vérifier les propriétés CSS ou encore connaitre les erreurs de votre code JavaScript !

5 Rappel de cours

Lors de ce TD, l'ensemble des pages web écrites devra (sauf mention contraire) être constitué de pages XHTML5 valide ne contenant pas de mise en forme (squelette index.html fournis).

La mise en forme sera dans un ou plusieurs fichiers CSS valides
(squelette css/td{x}.css fournis).

De même, le code JavaScript sera également dans des fichiers séparés
(squelette js/td{x}.js fournis).

JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives mais aussi pour les serveurs. C'est un langage orienté objet à prototype, c'est-à-dire que les bases du langage et ses principales interfaces sont fournies par des objets qui ne sont pas des instances de classes, mais qui sont chacun équipés de constructeurs permettant de créer leurs propriétés, et notamment une propriété de prototypage qui permet d'en créer des objets héritiers personnalisés. En outre, les fonctions sont des objets de première classe.

Le langage a été créé en 1995 par Brendan Eich (Brendan Eich étant membre du conseil d'administration de la fondation Mozilla à cette époque) pour le compte de Netscape Communications Corporation.

5.1 Introduction

JavaScript est un langage de programmation à part entière, permettant de réaliser des applications complexes dès que l'on a acquis une connaissance suffisante du langage et de ses diverses possibilités. Deux restrictions qu'il convient toutefois de souligner :

- JavaScript ne dispose d'aucune fonctionnalité graphique.
- Pour des raisons de sécurité, JavaScript ne peut ni lire ni écrire un fichier.

JavaScript est un langage **non-typé**. Cela signifie qu'il n'est pas forcément utile de déclarer les variables qui vont être utilisées et encore moins d'indiquer leur type. En fait, une variable est implicitement déclarée dès son apparition et typée en conséquence.

```
<script>
/*<![CDATA[*/
... Insérez ici votre code JavaScript ...
/*]]>*/
</script>
```

Un programme JavaScript s'intègre directement dans votre page HTML entre deux balises **<script>** et **</script>**. Si vous souhaité avoir du code XHTML valide, il est préférable d'ajouter les balises CDATA comme présenté ci-dessous :

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

Vous pouvez, et cela est vivement recommandé si vous souhaitez partager des fonctions JavaScript entre plusieurs pages HTML, utiliser un fichier externe pour stocker l'ensemble de vos fonctions JavaScript :

JavaScript peut intervenir dans un document HTML de deux façons :

- Il s'exécute au chargement de la page. Le programme JavaScript a pour objet d'écrire dans le document du code HTML pouvant s'adapter dynamiquement à plusieurs facteurs tels le type de configuration matérielle et/ou logicielle du client (navigateur, plugins...), le contenu éventuel de certains cookies, la date, l'heure du moment présent, etc.
- Il s'agit de scripts qui ne s'exécuteront non pas au moment du chargement de la page, mais une fois celle-ci chargée, lorsque l'utilisateur va interagir avec elle au moyen des différents objets qu'elle contient (liens, boutons, champs de texte...), mais aussi par des actions sur l'environnement (déplacement de la fenêtre, activation d'une autre fenêtre, déplacement de la souris, frappe au clavier...). Ces réactions seront donc provoquées par des événements qui se seront produits après la fin de chargement de la page.

5.2 Introduction au Document Object Model

Document Object Model a pour acronyme DOM. C'est une API pour du (X)HTML et XML valide. En suivant le DOM, les programmeurs peuvent construire des documents, naviguer dedans, ajouter, modifier ou effacer des éléments à ces documents. En tant que recommandation du W3C, l'objectif du DOM est de fournir une interface de programmation standard pour être utilisée par tous (applications, OS). Suivant le DOM, un document a une structure logique d'arbre

(ou de forêt). Le nom DOM provient de l'approche retenue par le W3C : une approche proche des modèles objets (propriétés, méthodes, description).

Le DOM a pour but d'uniformiser la manipulation des documents "web", notamment par les scripts. Cependant, afin de rester indépendant, le DOM est écrit en OMG (Object Managment Group) IDL (interface definition language) tout en proposant les liens pour une transcription dans les différentes implémentations (i.e. langage de programmation) comme ECMAscript (une sorte de standardisation de javascript). Chaque niveau de DOM correspond à des itérations successives, enrichissant au fur et à mesure le DOM.

La traduction d'un DOM dans les faits n'est pas immédiate (consulter le DOM).

La différence principale entre DOM (Core) et DOM HTML (application du DOM à l'HTML) est l'existence de méthodes conçue pour les scripts.Il y a en plus la spécialisation des classes :

- **HTMLDocument** hérite de l'interface document.
- **HTMLElement** hérite de l'interface élément (qui hérite de l'élément **node**).

https://developer.mozilla.org/fr/docs/Web/API/Node

D'un point de vue DOM, la différence entre HTML et XHTML est l'aspect sensible à la casse. En particulier les noms des balises sont en minuscule (pour **getElementsByTagName**()).

Le niveau 3 appliqué à l'HTML est défini sur les pages DOM Level 3 du W3C.

https://developer.mozilla.org/fr/docs/Web/API/Element/getElementsByTagName

6 Exercice 1 : Commençons avec l'objet document

Nous allons commencer par écrire une simple page HTML qui devra contenir 1 balise <**h1**>, 2 balises <**h2**> et 3 balises <**h3**>. Chacune de ces balises contiendra un court texte facilement identifiable et différent. Vous aurez donc au moins 6 textes différents. Libre à vous d'ajouter d'autres textes et d'autre balise à votre page. Le titre de votre page (défini par la balise <**title**> de la section <**head**>) devra être « TD1 - Exercice 1 ».

6.1 La propriété « document.title »

Ajoutez un id= « **title** » à votre <**h1**>.

Écrivez une méthode **defineHeading1()**, qui au chargement de la page, recherche dans celle-ci la balise ayant l'id « **title** » et change le titre de la page avec le contenu de la balise.

- Quel sera l'évènement qui déclenchera l'appelle de votre fonction ?
- Quelle méthode avez-vous utilisée pour récupérer l'objet représentant votre balise <**h1**> ?
- Quelle propriété de l'objet représentant votre balise <**h1**> avez-vous utilisée pour récupérer le texte de celui-ci ?

https://developer.mozilla.org/fr/docs/Web/HTML/Element/meta

Maintenant on souhaite faire la même chose, mais en récupérant le texte de la première balise **<h2>** du document.

- Quelle(s) méthode(s) avez-vous utilisée pour récupérer l'objet représentant la première balise **<h2>** ?

Attention, cette fois, il n'y a pas d'id dans la balise. Ecrivez une fonction **defineHeading2()** qui effectue cela.

Maintenant modifiez votre code pour prendre non pas le première mais la dernière balise **<h2>**. Attention, le code devra fonctionner quel que soit le nombre de balises **<h2>**.

S'il n'y a aucune balise **<h2>**, le titre de la page sera votre nom et prénom. Vous ferez cela dans la méthode **defineHeading3()**.

- Comment faire pour connaitre le nombre de balise **<h2>** du document ?

https://developer.mozilla.org/fr/docs/Web/API/HTMLCollection

Modifiez votre page en ajoutant le **<h1>**, le 2eme **<h2>**et le premier **<h3>** avec la classe CSS « **firstOrLast** ».

Ecrivez une fonction **defineHeading4()** qui sélectionne le premier élément de cette classe comme titre pour le document, si le nombre d'éléments de cette classe est paire. Si le nombre est impair, on utilisera le dernier.

S'il n'y en a aucun, le titre de la page sera votre nom et prénom.

- Quelle méthode avez-vous utilisée pour récupérer les objets de votre classe ?
- Comment avez-vous déterminé si un nombre est pair ?

6.2 Les propriétés innerHTML, innerText et textContent

Ajoutez à votre page une balise div contenant une balise p contenant elle-même un petit texte (de 3 mots minimum) dont une partie est entourée par une balise **.**

Ajoutez à votre page un deuxième ensemble de balise identique au précédent mais dont seul le texte change.

Ecrivez une fonction **swapInnerHTML()** qui échange le contenu des deux balises **<p>**.

On pourra faire l'hypothèse qu'il n'y a que nos 2 balises **<p>** dans toute la page.

- Quelles différences existe-t-il entre les 3 propriétés innerHTML, innerText et textContent ?

https://developer.mozilla.org/fr/docs/Web/API/Element/innertHTML

https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/innerText

https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent

6.3 La propriété document.lastModified

Une des choses importantes pour une page web est de savoir quand celle-ci a été mise à jour et par qui.

Si on peut retrouver ces informations facilement aux travers des métadonnées associées à un fichier, cela n'est pas forcément facile à faire pour tout le monde.

Voyons comment faire cela facilement avec une petite fonction JavaScript.

Ajoutez à votre page les balises **<meta />** pour l'auteur, la description et les mots clés.

Ajoutez une balise ** ** vide à la fin de votre page. Celle-ci aura l'id « **update-date** ».

Ecrivez une fonction **dateAlter**(), qui automatiquement, au chargement de la page, ajoute un texte du type « Dernière modification : le vendredi 18 janvier 2021 par Nom Prénom » dans la div (en lieu et place du texte existant, s'il y en a un).

La date sera récupérée via une propriété de l'objet document. L'auteur lui sera identifié à l'aide de la balise **<meta />**.

Pour cela vous aurez besoin d'un objet Date. Voici une rapide présentation (mais partielle) de celui-ci extraite du site de Mozilla.org :

```
Constructeurs :
new Date();
new Date( milliseconds);
new Date( dateString);
new Date( year, month, day [, hour, minute, second, millisecond ]);
Méthodes :
Date.prototype.getDate()
Renvoie le jour du mois (entre 1 et 31) pour la date donnée, selon le temps local.
Date.prototype.getDay()
Renvoie le jour de la semaine (entre 0 et 6) pour la date donnée, selon le temps local.
Date.prototype.getFullYear()
Renvoie l'année (sans chiffre implicite, 1999 sera renvoyé et pas 99 par exemple) pour la date donnée, selon le temps local.
Date.prototype.getHours()
Renvoie l'heure (entre 0 et 23) pour la date donnée, selon le temps local.
Date.prototype.getMilliseconds()
```

```
Renvoie les millisecondes (entre 0 et 999) pour la date donnée, selon le temps local.

Date.prototype.getMinutes()

Renvoie les minutes (entre 0 et 59) pour la date donnée, selon le temps local.

Date.prototype.getMonth()

Renvoie le mois (entre 0 et 11) pour la date donnée, selon le temps local.

Date.prototype.getSeconds()

Renvoie les secondes (entre 0 et 59) pour la date donnée, selon le temps local.

Date.prototype.getTime()

Renvoie la valeur numérique de la date donnée, exprimée en nombre de millisecondes écoulées depuis le premier janvier 1970, 00:00:00 UTC (pour les temps antérieurs, ce sont des valeurs négatives qui seront renvoyées).
```

Lorsque plusieurs personnes éditent un même fichier, il est possible d'avoir plusieurs auteurs et donc plusieurs balise **<meta />** avec l'attribut **name="author".**

https://html.spec.whatwg.org/#meta-author

- Comment modifier votre code pour qu'il permette de sélectionner le 1er auteur de la liste ?
- Même question avec le dernier auteur de la liste.

7 Exercice 2 : l'objet Date

Comme nous avons déjà commencé à utiliser l'objet Date, continuons un peu à étudier son fonctionnement.

Ajoutez une balise paragraphe dans votre page. Celle-ci contiendra le texte suivant : « il reste xxx jours avant le 19 juillet 202x » (202x étant année en cours). Ajoutez une fonction **getNbDays()** qui calcule le nombre de jour restant quand on clique sur la balise paragraphe et qui remplace les **xxx** par la valeur. On pensera également à supprimer le « s » de jours quand cela sera nécessaire.

- Comment obtenez-vous le nombre de jours ?
- Comment faites-vous la mise à jour du texte ?

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date/now

7.1 setInterval() et setTimeout()

Ajoutez une nouvelle balise paragraphe avec l'id « **clock** » à votre page.

Dès le chargement de votre page, cette balise devra contenir l'heure au **format « hh:mm:ss »**. Pour cela vous créerez deux fonctions **updateClock1()** et **updateClock2()**. La première devra utiliser la méthode **setInterval()**, la deuxième la méthode **setTimeout()**.

Pour cela vous aurez besoin d'un peu de documentation.

Voici une rapide présentation (mais partielle) des ces deux fonctions :

window.setInterval()

Appelle une fonction de manière répétée, avec un certain délai fixé entre chaque appel.

```
intervalID = window.setInterval( fonction, delai[, param1, param2, ...]);
intervalID = window.setInterval( code, delai);
```

où

intervalID est un ID unique d'intervalle qui peut être passé à window.clearInterval() fonction est la fonction qui doit être appelée de manière répétée.

code, dans la syntaxe alternative, est une chaîne représentant le code à exécuter de manière répétée.

delai *est le nombre de millisecondes (millièmes de seconde) que setInterval() doit attendre avant chaque appel de fonction.*

https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval

window.setTimeout()

Exécute un morceau de code ou une fonction après un délai déterminé.

```
timeoutID = window.setTimeout( fnct, delay[, param1, param2, ...]);
timeoutID = window.setTimeout( code, delay);
```

où

timeoutID est l'identificateur numerique du timeout, qui peut être utilisé avec window.clearTimeout.

fnct est la fonction que vous désirez exécuter après delai millisecondes.

code est, dans la syntaxe alternative, une chaîne contenant le code à exécuter après delai millisecondes. (L'utilisation de cette syntaxe n'est pas recommandée pour les mêmes raisons que l'utilisation d'eval()).

delay est le nombre de millisecondes (millièmes de seconde) après lequel la fonction doit être appelée. Le délai réel peut s'avérer plus long (cf. documentation.

https://developer.mozilla.org/fr/docs/Web/API/WindowOrWorkerGlobalScope/setTimeout

- Laquelle des deux méthodes de l'objet **window** avez-vous utilisé ? Pourquoi ?

Une fois que votre horloge fonctionne avec les deux méthodes, ajoutez une balise div avec l'id « **graphic-clock** ». Cette balise contiendra le texte et les balise **<img />** que vous souhaitez.

Vous devez maintenant écrire une fonction **updateGraphicClock()** qui affiche de manière graphique une horloge dans cette balise. Pour cela vous utiliserez les images du dossier assets/images/ fourni.

8 Exercice 3 : HTML, CSS et JavaScript (obligatoire)

8.1 Champ Texte et Couleur d'arrière-plan

Ajoutez un champ texte de saisie (**input** avec **type="text"** dans un formulaire) avec un fond de couleur blanc.

On rappelle que la mise en forme doit être gérée par des instructions CSS qui seront, de préférence, contenues dans un fichier différent du fichier de la page HTML.

Pour cela préparer trois classes de styles CSS comme dans l'exemple ci-dessous :

```
.white {
      background-color: rgb(255,255,255);
}
.green {
      background-color: rgb(150,255,150);
}
.red {
      background-color: rgb(255,150,150);
}
```

Faites-en sorte que la zone de texte devienne rouge si le texte entré n'est pas un nombre et si l'utilisateur tape un nombre, alors le fond doit devenir vert.

Attention, si la zone de texte est vide, elle doit être de couleur blanche.

- Quel évènement avez-vous utilisé ?
- Comment avez-vous fait changer la couleur du champ texte ?

8.2 Menu déroulant

Rappel :

La propriété CSS display, peut prendre (entre autre) une des valeurs suivantes :

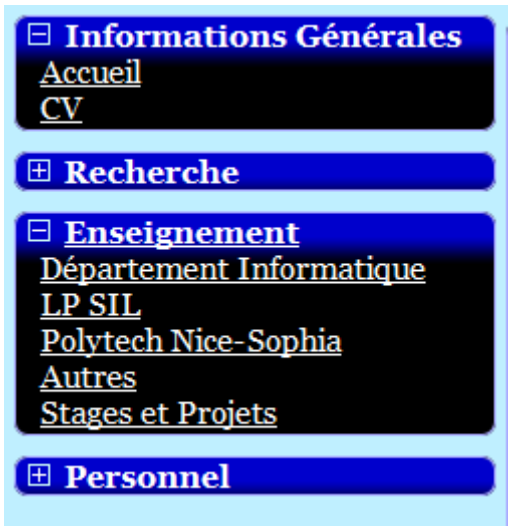
none : le bloc ne sera pas affiché.

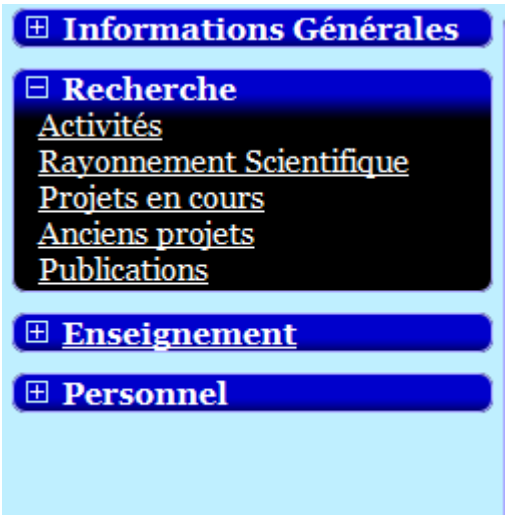
inline : le bloc sera considéré comme étant une seule ligne.

block : spécifie un bloc.

A l'aide de cette propriété et d'une petite fonction en JavaScript, réalisez un petit menu déroulant dans le même genre que celui des images ci-dessous. Ce menu sera contenu dans une balise **<aside>**. Pensez bien à ajouter **les icones « plus » et « moins »** à votre menu.

Attention, les icones devront être ajoutées depuis le Javascript et non dans le HTML.





9 Exercice 4 : parcours de l'arbre DOM

Pour finir ce TD, nous allons écrire une fonction **« search() »** qui se déclenchera au clic sur un bouton. Elle ira lire le contenu d'un champ texte et surlignera dans la page tous les mots correspondant au texte de ce champ texte.

Commencez par ajouter un paragraphe qui contiendra le champ texte et le bouton.

Votre fonction recherche devra, si c'est sa première utilisation, faire une sauvegarde de la page (on va dire seulement du corps de celle-ci) dans une variable. Si ce n'est pas la première utilisation, elle restaurera la page des modifications apportées préalablement.

Ensuite vous allez devoir parcourir le DOM pour recherche le texte souhaité dans toute le page et le remplacer par une balise ****.

Attention, il ne faut remplacer que le texte et pas les balises HTML. Notre nouvelle balise **** sera de la classe CSS « **select** » et contiendra le texte recherché.

La classe CSS « **select** » devra mettre l'arrière-plan de la balise en **jaune**.

Pour faire cet exercice, vous aurez besoin d'utiliser les méthodes de gestion des nœuds, ainsi que les propriétés « **childNodes** » et « **nodeType** » :

https://developer.mozilla.org/fr/docs/Web/API/Node/childNodes

https://developer.mozilla.org/fr/docs/Web/API/Node/nodeType

https://developer.mozilla.org/fr/docs/Web/API/NodeList

Une fois que cela fonctionne, ajoutez un deuxième champ texte à votre page. A chaque lettre saisie, il appellera une fonction « **interactiveSearch()** » dont le but est similaire à la fonction « **search()** » précédente.