R3.12 - 2024/2025

TD 1 : DOM - Opérations élémentaires

Exercice 1 : Petites opérations entre <a>mis

a) Créez une page HTML vierge. Ajoutez dans le corps de la page un élément `span` contenant le mot "Javascript". Dans un élément `<script>` que vous placerez après l'élément `span`, écrire le code JS pour ajouter " c'est très bien" à la suite de "Javascript". Vous avez bien entendu le droit d'attribuer un identifiant à l'élément `<span>`.
b) Placez à présent l'élément `<script>` avant l'élément `<span>` et testez. Comment expliquer ce résultat ?
c) Ajoutez dans votre page 2 liens (élément `<a>`) vers deux sites différents. Puis écrire le code JS qui "pourrit" les liens de sorte à ce que le premier envoie vers la cible du second et inversement.
d) Ajoutez à présent d'autres liens et généralisez la question précédente de sorte à permuter aléatoirement les liens et leur cible. Pour ce, on répétera l'opération suivante autant de fois qu'il y a de liens dans la page : Prendre deux liens au hasard et échanger leur cible. Au besoin, on se renseignera sur les méthodes de la classe `Math` et en particulier `random()` et `floor()`.
e) Ajoutez le code JS qui supprimera tous les liens de votre page.

Exercice 2 : Full JS

Créer exclusivement à l'aide d'instructions JS le corps de page suivant :

```
<body>

    <section style="background-color: darkcyan;width:300px;padding:3%">

        <div style="background-color:darkseagreen;width:100px;padding:2%">
            <p>Lorem ipsum dolor</p>
        </div>

        <div style="background-color:burlywood;width:100px;padding:2%">
            <p>sit amet consectetur</p>
        </div>

    </section>

</body>
```

Exercice 3 : Le DOM c'est carré

Dans cet exercice vous allez créer dynamiquement des éléments HTML. Pour ce, l'interface DOM définit la méthode `createElement` pour l'objet `document` qui attend en paramètre le nom de l'élément à créer. Par exemple `document.createElement("img")` retourne un noeud de la classe `HTMLImageElement` (soit un élément *<img>* en HTML).

a) Créez une page HTML vide à l'exception d'un élément `<script>`. Ecrire une fonction `randomColor` qui retourne une couleur RGB choisie aléatoirement. La fonction retournera cette couleur sous la forme

d'une chaîne respectant le format `"rgb(x, y, z)"`, où `x, y, z` sont 3 valeurs choisies au hasard entre 0 et 255. On remarquera que ce format est valide en CSS... Testez votre fonction en l'utilisant pour choisir aléatoirement la couleur de fond de votre page.

b) On souhaite à présent créer un élément représentant un carré de couleur sur la page. Pour ce on veut utiliser un élément HTML `div` dont on contrôlera l'apparence via sa propriété de style. Ecrire une fonction `createSquare` qui retourne un objet `HTMLDivElement` (soit un élément * * en HTML) tel que :
   — sa couleur est définie aléatoirement
   — son padding est de 3 pixels
   — sa propriété de style `display` vaut `"inline-block"` (important pour la suite)
   La dimension du carré sera passée en paramètre de votre fonction sous la forme d'une chaîne qui comprend l'unité (par exemple "24px"). Un second paramètre permettra de transmettre la valeur du padding, là encore sous la forme d'une chaîne incluant l'unité. Testez là en ajoutant un petit carré dans votre page.

c) Créer une nouvelle fonction `multiSquares` qui attend en paramètre un nombre et qui insère dans la page autant de carrés obtenus grâce à votre fonction `createSquare`. Testez en ajoutant 100 carrés colorés dans votre page. Inspectez l'arborescence DOM obtenue via les outils de développement pour voir l'arborescence que vous avez générée.

d) Au lieu d'avoir des carrés les uns à la suite des autres, on souhaite les imbriquer les uns dans les autres, quitte à moduler leur taille si besoin. Créer une fonction `imbricateSquares` qui prend en paramètre un nombre et qui ajoute dans la page autant de carrés imbriqués les uns dans les autres. Note : pour moduler la taille des cubes, renseignez-vous si besoin sur la valeur `"auto"` pour les propriétés de style `width` et `height`. Testez votre fonction en ajoutant 100 carrés imbriqués dans votre page. Comparez l'arborescence DOM obtenue vis à vis de la question précédente.

e) Créer une fonction `changeColor` qui dont l'appel aura pour effet de changer (au hasard) la couleur de tous les carrés imbriqués (ou non d'ailleurs). Testez votre fonction en insérant des carrés, puis en changeant leur couleur.

f) Connaissez vous la méthode `setInterval` de l'objet `window` ? Renseignez-vous sur la question, puis faites en sorte que la couleur des carrés change aléatoirement toutes les 200 millisecondes.

g) Imaginez une solution pour aboutir au même résultat mais sans la fonction `changeColor`. Trouvé ? Pouvez-vous l'écrire sous la forme d'une fonction anonyme maintenant ?

Exercice 4 : Killer Node

Pour cet exercice, on rappelle que tout noeud possède une propriété `nodeType` qui nous renseigne sur le type du noeud (HTML element, text...). Chaque type de noeud correspond à une valeur constante définie dans la classe `Node`. En particulier la constante `Node.TEXT_NODE` (qui vaut 3 par convention) indique un noeud texte.

a) Ecrivez une fonction qui attend un noeud et qui retourne un tableau contenant tous les noeuds texte présents dans le sous arbre enraciné en ce noeud. Notez bien qu'on ne peut pas répondre à cette question sans réaliser une foncton récursive. L'idée est de regarder le type du noeud courant ; si c'est un noeud texte on l'ajoute au tableau ; sinon il faut répéter l'opération sur ses enfants (s'il en a).

b) Ecrivez une fonction `removeTextNodes` qui supprime tout le contenu textuel d'un document. Pour tester à grande échelle votre code, vous pourrez récupérer le source HTML d'un "vrai" site (un site d'information par exemple, ou de l'université) et inclure votre code dans un élément `<script>` en fin de page.

c) En suivant une démarche analogue, écrivez une fonction `removeImgNodes`