

TP 002 : Prise en main du JavaScript

Dans un premier temps téléchargez l'archive du site, qui est la correction du TP001. J'ai réorganisé les fichiers en les rangeant dans des dossiers.

L'exercice préliminaire consiste à corriger les erreurs concernant les chemins dans chaque fichier (6 en tout).

Créez maintenant un fichier javascript (script.js) que vous rangerez dans un dossier ad-hoc.

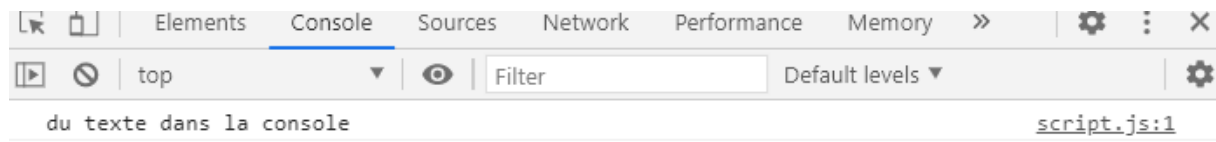
Exercice 1 : La console

Dans votre fichier javascript, écrivez l'instruction suivante (une instruction se finit toujours par un point-virgule) :

```
console.log("du texte dans la console");
```

Ensuite, dans votre fichier html, ajoutez dans le head l'élément **script** pour charger votre fichier javascript.

Enfin, dans le navigateur, ouvrez votre fichier html et ouvrez la console (CTRL+MAJ+J) ; vous devriez observer ceci :



La console présente deux intérêts :

- Quand vos algorithmes n'aboutissent pas au résultat attendu, vous pouvez afficher vos variables à des étapes intermédiaires pour essayer d'identifier où ça pêche.
- C'est à cet endroit que s'afficheront les erreurs générées par votre code

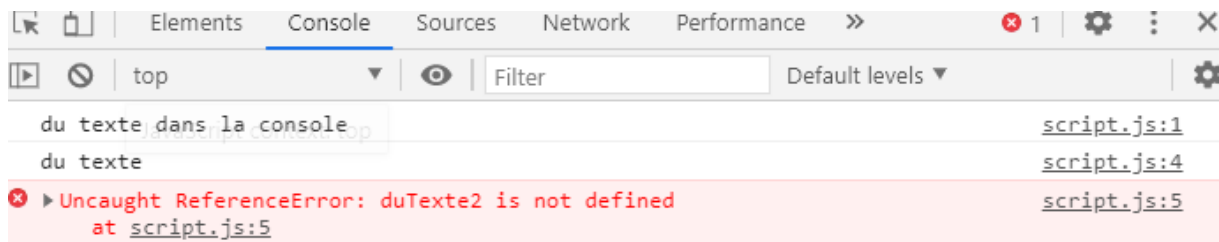
A présent, dans le fichier javascript, **déclarez** une variable qui s'appellera *duTexte*. **Affectez**-lui la chaîne de caractère « du texte ». **Accéder** à cette variable dans la console.

```
var duTexte;  
duTexte= "du texte";  
console.log(duTexte);
```



Accédez maintenant la variable *duTexte2* :

```
console.log(duTexte2);
```



La console nous informe de notre première erreur. Elle nous indique le fichier javascript et, dans celui-ci, la ligne qui a généré cette erreur. Enfin l'information principale (en rouge) concerne la nature de l'erreur. Ici, elle nous indique que notre variable *duTexte2* n'existe pas ; en effet, on ne l'a pas déclarée.

Sans modifier les lignes de code déjà écrites, rajoutez les lignes de code nécessaires pour **déclarer** cette variable et lui **affecter** la chaîne de caractère « cela résout mon erreur ».

Enfin, une dernière opération : **la concaténation**. On a vu comment afficher du texte et comment afficher une variable, maintenant nous allons afficher les deux en même temps :

```
console.log("ma variable vaut : "+duTexte+(4+5));
```

du texte dans la console	script.js:1
du texte	script.js:4
cela résout mon erreur	script.js:6
ma variable vaut : du texte9	script.js:7

La concaténation consiste à enchaîner plusieurs informations. Ici, du texte, puis une variable et enfin une addition. La concaténation est une opération dont au moins l'une des opérandes doit être une chaîne de caractère. C'est pourquoi 4+5 donne 9.

La concaténation ne se fait pas seulement pour l'affichage console, elle peut servir pour affecter une valeur à une variable.

Vous savez maintenant déclarer une variable, lui affecter une valeur et l'afficher dans la console en lui concaténant du texte. Vous savez également lire les informations fournies par un message d'erreur.

Exercice 2 : Premier accès au DOM, parcours d'éléments

Dans votre fichier javascript, placez votre code précédent en commentaire (*/* votre code */*).

Dans l'exercice précédent, `console` était un objet, log une de ses fonctions et entre parenthèses les paramètres de la fonction.

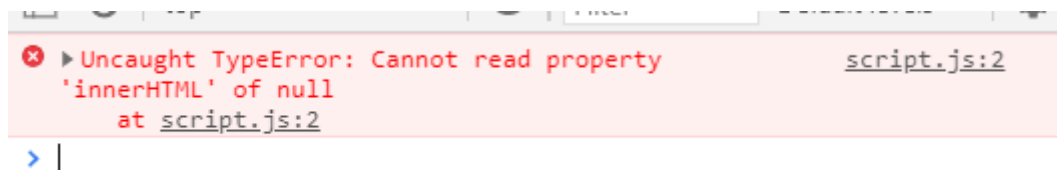
Sachant cela, appelez la fonction `getElementById()` de l'objet `document`, avec comme paramètre `"footer"`. Affectez le résultat de cette fonction à une variable nommée `monFooter`.

Ceci a pour effet de récupérer l'élément HTML qui possède l'identifiant `"footer"`. Dans le javascript, il prend la forme d'un objet `Element` qui, comme `console` et `document`, possède un ensemble de fonctions ou de propriétés.

Pour afficher le contenu de l'élément récupéré, on utilise la propriété `innerHTML` :

```
console.log(monFooter.innerHTML);
```

Cette fois, pas de parenthèse, ce n'est pas une fonction. Ceci aurait dû afficher « Le pied de page » dans la console ; à la place on a une erreur.



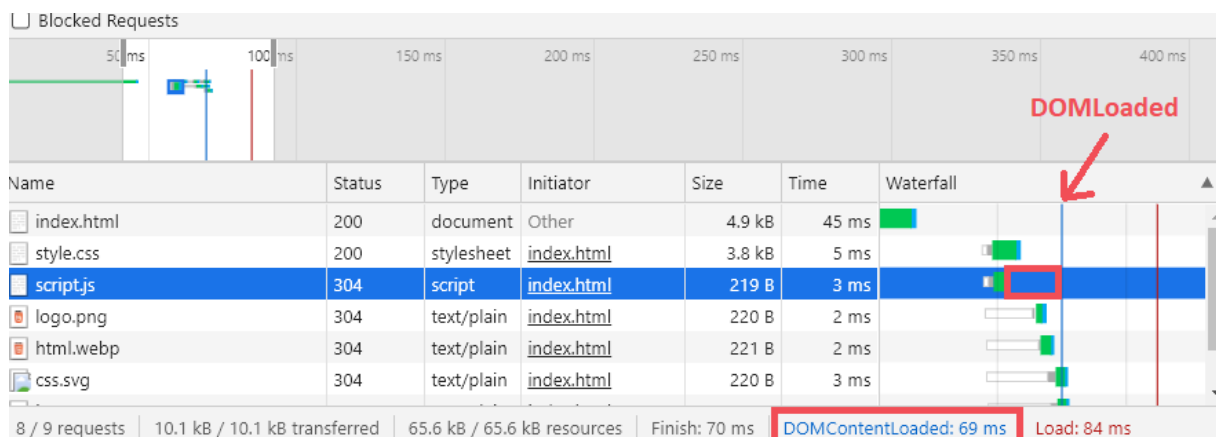
La console nous informe qu'on essaye d'**accéder** à la **propriété** `innerHTML` d'un **objet null**.

Cela veut dire que `monFooter` (l'**objet** pour lequel on essaie d'**accéder** à la **propriété** `innerHTML`) vaut **null**.

Ce qui signifie que la variable existe bien mais qu'il n'y a rien dedans.

S'il n'y a rien dedans, on comprend que notre accès au DOM avec la **méthode** `document.getElementById`, n'a pas fonctionné. Deux raisons à cela :

- Soit l'identifiant fourni en paramètre n'existe pas (ce qui n'est pas notre cas)
- Soit l'élément n'existe pas encore lorsque l'instruction est appelée. On se rappelle que le navigateur lit de haut en bas les documents. Or, on a placé notre script dans le `head`, avant que le footer n'existe, les instructions javascript sont donc appelées trop tôt.



On peut facilement s'en rendre en affichant l'onglet network à la place de la console (Actualisez la page).

La barre bleu indique que le DOM est chargé. Cela se produit bien après qu'on ait chargé et lu le fichier javascript.

Proposez et appliquez une solution simple à ce problème.

Pour la suite on va utiliser la méthode `querySelector` de l'objet `document`. Cette méthode prend en paramètre un sélecteur css.

Avec cette méthode affichez le contenu de la boîte pub dans la console.

```

Le pied de page      script.js:2
pub                  script.js:3
> |

```

La méthode `querySelector` ne récupère qu'un seul élément.

Remplacez le sélecteur précédent par le sélecteur `"td"`. Cela devrait afficher 12 dans la console (c'est le premier td).

Si on a besoin de récupérer plusieurs éléments on utilise la méthode `querySelectorAll()`.

Dans l'instruction précédente remplacez `querySelector` par `querySelectorAll`.

```

Le pied de page      script.js:2
undefined            script.js:3
>

```

L'affichage devient alors `undefined`. Avec `querySelector`, nous récupérons un objet `Element` qui possédait la propriété `innerHTML`. Avec `querySelectorAll`, nous récupérons un objet `NodeList`, qui est une liste d'`Element` et qui ne possède donc pas de propriété `innerHTML`.

Remplacez donc `innerHTML` par `length`.

Cette propriété affiche le nombre d'éléments dans la liste.

Que faire pour connaître la liste des propriétés d'un objet ?

Pour récupérer un élément de la liste il faut utiliser les crochets avec comme paramètre l'index de l'élément dans la liste. L'index va de 0 à length-1.

Affichez le contenu du 3^{ème} élément de la liste (cela doit afficher 85).

Mettez le code précédent en commentaire.

Dans la deuxième section du document HTML, ajoutez un attribut class qui vaudra "card" dans chacune des div.

Avec la méthode querySelectorAll, récupérez ces divs grâce à leur classe et **affectez** les à une variable nommé mesCards.

Dans la console affichez la taille de mesCards, pour vous assurer que cela a fonctionné.

taille mesCards : 3

script.js:2

La boucle for va vous permettre de parcourir les éléments d'une liste.

```
for(var i = 0 ; i<5 ; i++) {
    console.log(i);
}
```

Ces instructions permettent d'afficher dans la console 0, 1, 2, 3, 4.

- var i = 0 : initialisation
- i<5 : condition d'arrêt. La boucle s'arrête dès que i n'est plus inférieur à 5
- i++ : pas d'incréméntation. A chaque tour de boucle i est incrémenter de 1.

Appliquez ceci à votre liste de cards. La condition d'arrêt se fait par rapport à la taille de la liste.

L'affichage de la console devrait donc donner :

0

script.js:4

1

script.js:4

2

script.js:4

Maintenant, on veut afficher la source de l'image contenue dans nos card.

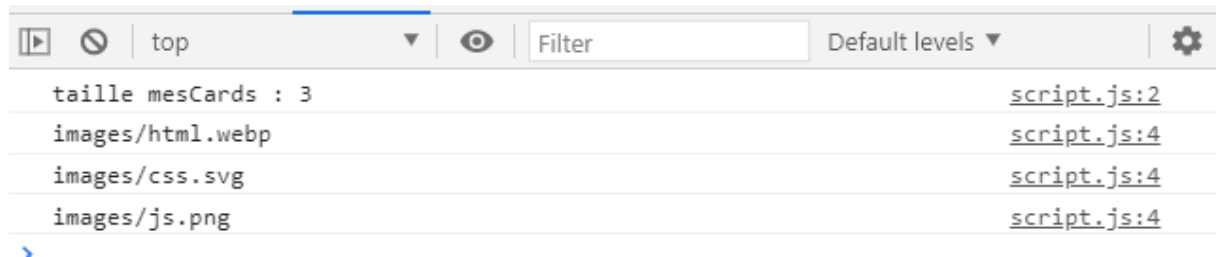
```
<div class="card" style="--color:orange;" ></div>
<div class="card" style="--color:blue;"> </div>
<div class="card" style="--color:yellow;"> </div>
```

Pour cela il faut d'abord récupérer l'élément voulu dans la liste (avec les crochets).

Le paramètre des crochets va évoluer grâce à la boucle et ainsi parcourir chacun des éléments de la liste.

Pour chacun de ces éléments on va récupérer l'img. Il existe plusieurs méthodes. L'une d'entre elle consiste à utiliser la propriété `firstChild` qui va renvoyer le premier élément enfant, ici l'img.

Sur l'élément récupéré, vous utiliserez la méthode `getAttribute`.



taille mesCards : 3	script.js:2
images/html.webp	script.js:4
images/css.svg	script.js:4
images/js.png	script.js:4

Vous savez donc maintenant accéder à un ou plusieurs éléments dans le DOM et utiliser des méthodes ou accéder à des propriétés de ceux-ci. Grâce aux boucles vous avez vu comment parcourir les éléments et comment appliquer plusieurs fois la même opération en une seule instruction. Vous avez également commencé à effleurer la navigation à l'intérieur du DOM en accédant à l'élément fils ; il existe d'autres méthodes de navigation à voir dans le cours ou sur la doc en fonction des besoins.

Exercice 3 : Fonctions, événements et callback

Avant de passer à un cas plus pratique, il vous reste à découvrir le plus important du javascript : la gestion des événements.

Vous allez continuer à travailler avec la boucle précédente. Pour le moment le résultat s'affiche à l'ouverture de la page.

Commencez par placer vos instructions à l'intérieur d'une fonction nommée *maFonction*.

```
function maFonction( ){  
    les instructions ;  
}
```

Plus rien ne devrait s'afficher dans la console.

Il existe plusieurs méthodes pour gérer les événements :

- *Ajouter des listeners aux éléments dans le JavaScript*
- *Rajouter un attribut dans le HTML*
- *Changer la propriété des éléments dans le JavaScript*

Vous allez tester chacune de ces méthodes.

Commencez par ajouter un attribut onclick dans l'élément h1 de votre document HTML.

Cet attribut prendra pour valeur un appel à la fonction *maFonction*.

Quand vous cliquez sur le titre vous devriez voir dans la console l'affichage précédent. Les attributs d'événements prennent en valeur du javascript, de la même manière que l'attribut style prend du css.

La liste de ces attributs est disponible sur la documentation :

https://www.w3schools.com/tags/ref_eventattributes.asp

Dans le fichier javascript créez une nouvelle fonction nommée *maFonction2* qui fera un affichage console de la chaîne de caractère « ma fonction 2 ».

Avec la méthode `querySelector`, récupérez l'élément h1, et affectez à sa propriété onclick la fonction *maFonction2*. *./!\ Attention si vous mettez des parenthèses cela revient à faire un appel de la fonction.*

Lorsque vous cliquez sur le titre vous devriez avoir l'affichage suivant dans la console

ma fonction 2	script.js:10
>	

L'affichage précédent a disparu : la propriété onclick modifie l'attribut onclick de l'HTML.

Enfin, vous allez voir les listeners.

Un listener est une liste de callback associés à un type d'événement.

```
document.querySelector("h1").addEventListener("click",  
maFonction);
```

Cette instruction va ajouter *maFonction* comme callback du click sur le h1.

Ajoutez également *maFonction2* en callback du même élément et du même événement.

Au clique sur le titre vous devriez avoir l'affichage suivant dans la console :

ma fonction 2	<u>script.js:10</u>
taille mesCards : 3	<u>script.js:3</u>
images/html.webp	<u>script.js:5</u>
images/css.svg	<u>script.js:5</u>
images/js.png	<u>script.js:5</u>
ma fonction 2	<u>script.js:10</u>

Le premier affichage est celui du onclick. Puis chaque callback de la liste des listeners est appelé.

Enfin, remplacez l'élément script de votre HTML dans le head.

Vous devriez avoir des erreurs qui apparaissent, comme vu précédemment.

Créez une fonction nommée *start*.

Dans cette fonction mettez chacune des instructions qui provoquent une erreur.

A présent, sur l'objet window, ajouter la fonction *start* comme callback de l'événement "DOMContentLoaded".

Ainsi ces instructions ne seront exécutées qu'une fois que le DOM est chargé. Cliquez sur le titre et vérifiez que l'affichage console est bien le même qu'avant.

Exercice 4 : Changer de thème, modifier le CSS en javascript

Je vous ai ajouté un bouton dans le document HTML. Vous allez faire en sorte qu'au clique sur celui-ci le thème du site change.

Ce bouton va être de type toggle, c'est-à-dire qu'il va alterner entre deux états.

Cela signifie que dans le javascript il vous faudra une variable de type booléen (true ou false).

Créez une variable nommée « sombre » et qui vaut true (parce que le thème originel du site est sombre).

Créez une fonction qui se nomme « theme » et qui ne prend pas de paramètre.

A l'aide d'un attribut HTML, faites-en sorte qu'au clique sur le bouton toggle la fonction *theme* soit appelée.

Puisqu'on fait une fonction de type toggle, la première étape de la fonction consiste à changer la valeur de la variable sombre : si elle vaut true alors elle devient false, sinon elle devient true.

Ce qui se traduit en javascript avec un bloc de condition :

```
if(sombre)
    sombre=false;
else
    sombre=true;
```

Une version plus concise consiste à utiliser l'opérateur « ! ». Le point d'exclamation inverse la valeur d'un booléen :

```
sombre=!sombre;
```

Maintenant, rajoutez, après l'instruction sur sombre, un nouveau bloc de condition. La condition se fera sur sombre.

- Si sombre vaut true : grâce à la méthode `querySelector` de l'objet `document`, récupérez le `body`. Sur l'élément récupérez accédez à la propriété `style` puis à la propriété `backgroundColor` de `style` et affectez-lui la chaîne de caractère « `gray` ».
- Sinon faites la même chose mais affectez-lui la chaîne de caractère « `white` ».

Vous devriez constater qu'à chaque clique sur le bouton, le fond du body alterne de couleur.

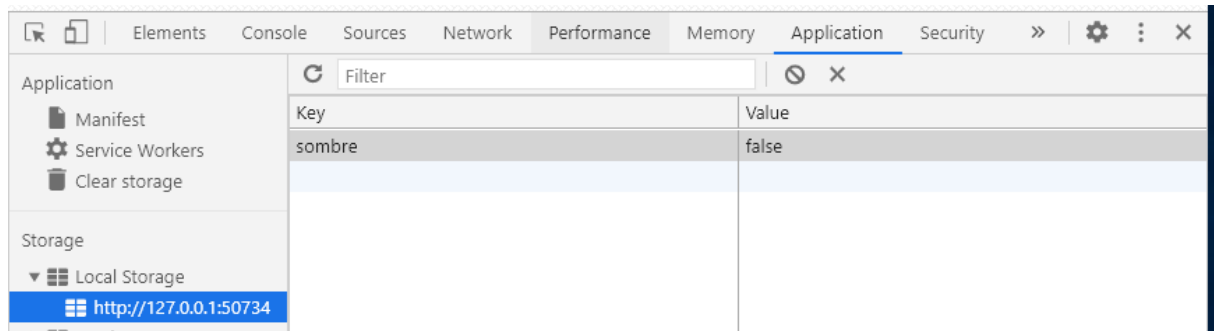
Si vous actualisez la page, le thème revient toujours sur celui de base.

Vous allez faire en sorte que cet état soit sauvegardé.

Dans la fonction *theme*, à la fin, rajoutez l'instruction suivante : sur l'objet `localStorage`, appelez la méthode `setItem`. Passez en premier paramètre la chaîne de caractère « sombre » et en deuxième paramètre la variable `sombre`.

Le premier paramètre est la clef, qui permettra de retrouver cette valeur avec la méthode `getItem`.

La variable `sombre` est donc sauvegardée et mise à jour à chaque appel de *theme*. Vous pouvez le constater en ouvrant l'onglet application et en cliquant sur le bouton *toggle*.



Si vous actualisez la page le thème revient toujours au thème de base. Il vous faut faire deux choses :

- Charger la valeur stocker dans `localStorage` et la mettre dans la variable `sombre`.
- Ajouter la fonction *theme* comme callback de l'événement `DOMContentLoaded` de l'objet `window`

/!\ La fonction `getItem` retourne une chaîne de caractère et il vous faut un booléen. Cherchez comment convertir une chaîne de caractère en booléen. (Indice : l'opérateur `==` retourne un booléen)

A présent, si vous avez correctement fait les choses, la couleur devrait changer à chaque actualisation de la page. C'est parce que la méthode *theme* est appelée au chargement de la page et c'est une fonction *toggle*. Une solution simple serait de stocker la valeur inverse de la variable `sombre`.

Mettez en place cette solution.

Enfin, nous allons voir comment modifier des variables CSS pour modifier simplement le style de plusieurs éléments.

Notons que si le changement de thème entraine le changement de beaucoup de propriétés, il peut être préférable de créer un deuxième fichier de style et d'importer l'un ou l'autre ; ce qui n'est pas notre cas, nous allons seulement modifier la couleur du fond de page et des boites.

Dans le fichier CSS, j'ai créé deux variables `--colorBoite` et `--colorFond`, ce sont ces valeurs que vous allez faire évoluer.

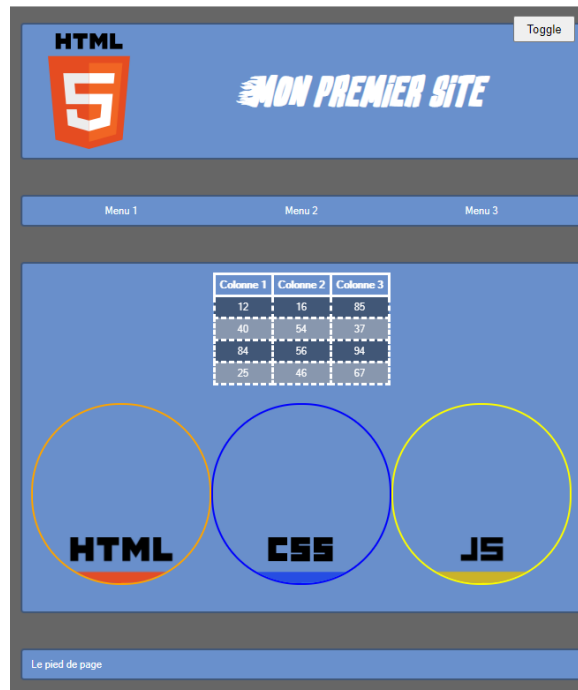
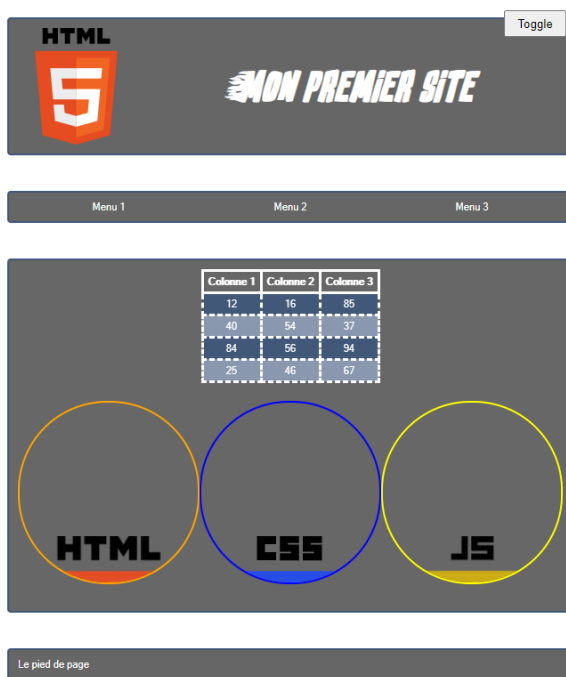
Premièrement, créez une variable dans votre fichier javascript qui s'appelle `root` et qui prend pour valeur la propriété `documentElement` de l'objet `document`.

Créez deux variables : `blue` et `gray` et affectez leur respectivement `"#6990CC"` et `"#666666"`.

Dans la fonction `theme`, dans le bloc de condition, retirez les deux instructions qui changent la couleur de fond.

A l'aide de la méthode `setProperty` de la propriété `style` de l'objet `root`, modifiez la valeur de `--colorBoite` et `--colorFond`

- Pour le thème sombre vous affecterez aux boites la variable `blue` et au fond la variable `gray`
- Pour le thème clair vous affecterez aux boites la variable `gray` et au fond la chaîne de caractère « `white` ».



Exercice 5 : Tooltip

Dans cet exercice, vous allez afficher un tooltip au survol de certains éléments. Il va falloir ajouter un callback sur l'événement de survol de chaque élément concerné, créer dynamiquement une div pour afficher le tooltip, la positionner sur le curseur et faire disparaître le tooltip quand on ne survole plus l'élément.

Pour commencer, vous allez ajouter un attribut `data-tooltip` aux éléments qui bénéficieront de cette fonctionnalité. Dans notre cas, ce sera nos 3 cards. La valeur de cet attribut sera « `html` », « `css` » et « `js` » selon la card.

Déclarez une variable `mesTooltips`.

Créez une fonction `initTooltip`.

Ajoutez cette fonction aux callback de l'événement `DOMContentLoaded` de l'objet `window`.

A partir de là on travaillera dans cette fonction.

Grâce à `querySelectorAll`, la variable `mesTooltips` se verra affecter **tous** les éléments **ayant** un **attribut `data-tooltip`** (trouvez le sélecteur adapté).

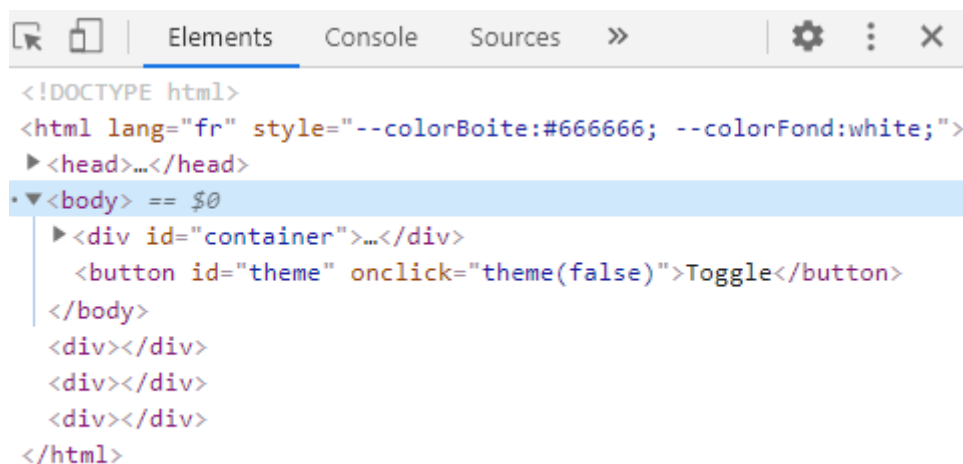
Ecrivez à présent une boucle qui parcourt les éléments de `mesTooltips`.

Dans cette boucle, déclarez une variable nommée `tooltip` avec le mot-clef **`let`** plutôt que **`var`**.

Proposez une explication à l'utilisation du mot-clef `let` plutôt que `var`.

Créez un élément div grâce à la méthode `createElement` de l'objet `document` et affectez-le à la variable `tooltip`.

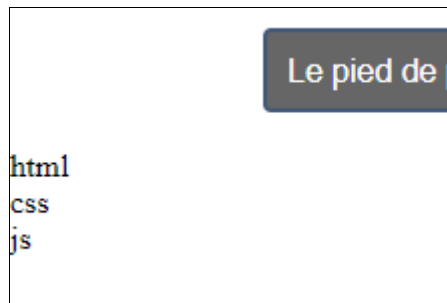
Ajoutez cet élément à votre page grâce à la méthode `appendChild`. Vous pouvez utiliser la variable `root` déclarée plus tôt.



```
<!DOCTYPE html>
<html lang="fr" style="--colorBoite:#666666; --colorFond:white;">
  <head>...</head>
  <body> == $0
    <div id="container">...</div>
    <button id="theme" onclick="theme(false)">Toggle</button>
  </body>
  <div></div>
  <div></div>
  <div></div>
</html>
```

Si vous regardez les éléments de votre page, vous devriez voir 3 divs créées. Il y a 3 cards, donc il y a eu 3 tours de boucle.

A présent, affectez au contenu de ces divs (*innerHTML*) la valeur de l'attribut *data-tooltip* ; grâce à la méthode *getAttribute*.



Pour chaque élément de *mesTooltips*, ajoutez un callback sur l'événement *mousemove*. Ce callback sera une fonction anonyme avec un paramètre.

```
mesTooltips[i].addEventListener("mousemove", function(e){  
    desInstructions;  
});
```

La fonction est déclarée à la volée et est dite anonyme parce qu'elle n'a pas de nom. Le paramètre "e" sera rempli automatiquement par le navigateur et correspond à l'événement ; on pourra donc accéder aux propriétés de "e" pour avoir des informations sur l'événement, telle que la position de la souris dans la page.

Dans cette fonction, pour tester, ajoutez une instruction qui affiche dans la console la coordonnée Y de la souris. Observez le résultat en bougeant la souris au-dessus des cards.

Une fois testée, retirez cette instruction.

Dans cette fonction modifiez la propriété de style **position**, de la variable *tooltip* et affectez-lui la valeur **fixed**. Puis, modifiez les propriétés de style **left** et **top** pour leur affecter la position de la souris. La valeur doit être concaténée à "px" pour être prise en compte.

Le tooltip devrait alors se déplacer avec votre souris au-dessus de la card associée et ne plus bouger une fois que la souris ne survole plus la card.

Ajoutez un callback sur l'événement ad-hoc pour que la propriété **display** du tooltip vaille « none » quand la souris quitte la card.

Faites de même pour l'événement **mouseenter** ; cette fois, pour que le tooltip soit affiché.

Enfin, dernière étape, faites-en sorte que le tooltip soit masqué à l'ouverture de la page.

De cette manière si vous voulez ajouter un tooltip à un élément il n'y a plus qu'à créer un attribut *data-tooltip* à l'élément souhaité.

Ajoutez un attribut *data-tooltip* au titre pour tester.

