

La manipulation du DOM

Ajouter un élément dans une page via JavaScript

Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

<script>
    let div = document.createElement('div');
</script>
</body>
</html>
```

Tout d'abord il faut savoir que beaucoup de nos commandes commenceront par “document” (car ça affecte le document en cours), ensuite, pour la création d'un élément, on utilise la méthode ‘createElement(“typeDelementAcréer”)’ et on passe en paramètre le type d' élément que l'on veut créer (dans notre exemple, une “div”). Je stocke l' élément dans une variable “div”

Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <script>
        let div = document.createElement('div');
        document.body.append(div);

    </script>
</body>
</html>
```

Attention, dans l'étape précédente, nous juste créé l'~~élément~~ mais nous ne l'avions pas inséré dans la page. Pour ce faire nous devons écrire "document" (j'avais prévenu qu'on allait souvent l'utiliser), ensuite 'body' ~~car~~ nous voulons l'insérer dans le body du document, puis 'append(élémentAinsérer)' qui prend en paramètre l'élément à insérer.

Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

<script>
    let div = document.createElement('div');
    document.body.append(div);
    ...
</script>
</body>
</html>
```



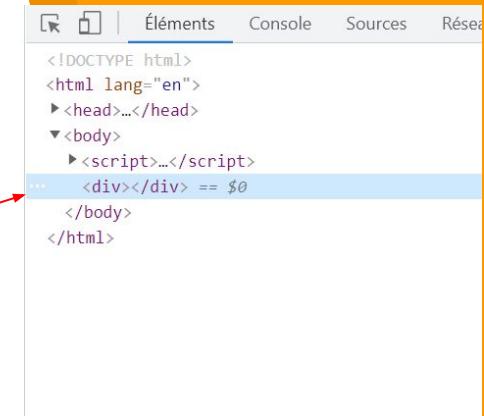
“Append” signifie “à la suite” c'est pour cela que notre “div” se met sous notre balise javascript (elle se met à la suite des éléments déjà présents dans “body”). Si j'insère une autre “div” à l'élément “body”, elle ira sous la balise “div” déjà créée, et ainsi de suite.... Pour l'instant rien ne s'affiche dans la page car notre “div” est vide mais on peut la voir dans notre console.

Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

<script>
    let div = document.createElement('div');
    document.body.appendChild(div);

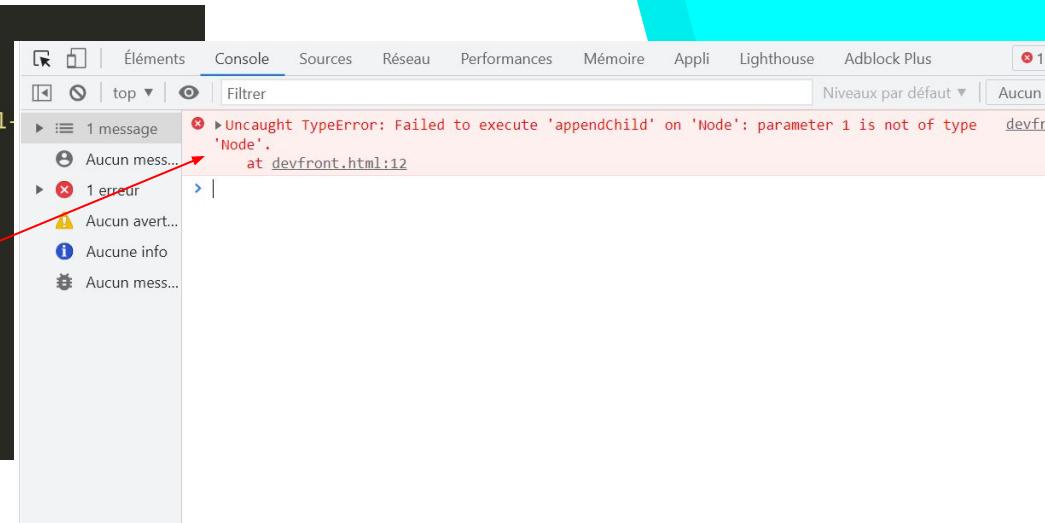
</script>
</body>
</html>
```



Il existe une autre méthode qui ressemble beaucoup à “append”, c'est “appendChild(“élémentAintégrer”)”.

Javascript - insérer un élément

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-
6     <title>Document</title>
7 </head>
8 <body>
9
0 <script>
1     Let div = document.createElement('div');
2     document.body.appendChild('coucou');
3
4 </script>
5 </body>
6 </html>
```



Les 2 différences entre ces méthodes sont :

- 1/ “appendChild” ne peut pas insérer de texte....

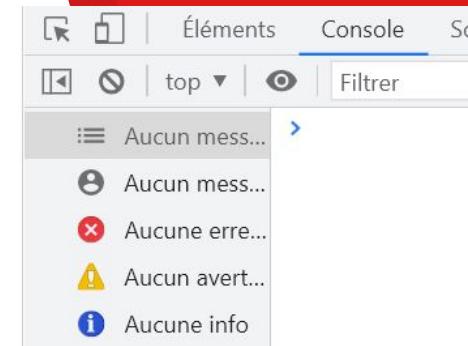
Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <script>
        let div = document.createElement('div');
        document.body.append('coucou');

    </script>
</body>
</html>
```

A red arrow points from the word "coucou" in the code to the same word displayed in the browser's DOM preview.



Les 2 différences entre ces méthodes sont :

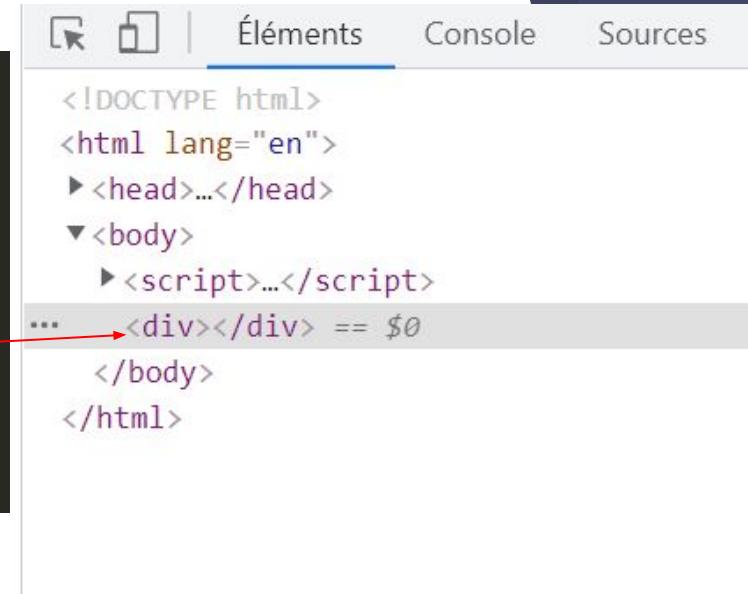
1/ “appendChild” ne peut pas insérer de texte.... Alors que “append” oui !

Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <script>
        let div = document.createElement('div');
        let div2 = document.createElement('div');
        document.body.appendChild(div, div2);

    </script>
</body>
</html>
```



Les 2 différences entre ces méthodes sont :

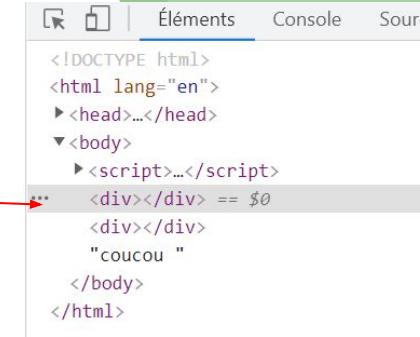
2/ “appendChild” ne peut pas insérer plusieurs éléments à la fois....

Javascript - insérer un élément

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

<script>
  let div = document.createElement('div');
  let div2 = document.createElement('div');
  document.body.append(div, div2, 'coucou');
</script>
</body>
</html>
```

coucou



Les 2 différences entre ces méthodes sont :

2/ “appendChild” ne peut pas insérer plusieurs éléments à la fois.... Alors que “append” oui !

Ajouter un texte dans une page via JavaScript

Javascript - insérer un texte

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>

    <script>
      let div = document.createElement('div');
      div.innerText = "Texte dans la div !";
      document.body.append(div);

    </script>
  </body>
</html>
```

Texte dans la div !

Éléments Console Sources Réseau Performance

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <script>...</script>
    ...<div>Texte dans la div !</div> == $0
  </body>
</html>
```

Pour mettre du texte dans notre “div”, avant son insertion, j’utilise la propriété “.innerText” à laquelle j’attribue une valeur (ma phrase)

Javascript - insérer un texte

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <script>
        let div = document.createElement('div');
        div.textContent = "Texte dans la div !";
        document.body.append(div);

    </script>
</body>
</html>
```

Texte dans la div !

The screenshot shows the browser's developer tools with the 'Éléments' (Elements) tab selected. The DOM tree is displayed, starting with the <!DOCTYPE html> declaration, followed by the <html> element, then the <head> and <body> elements. Inside the <body> element, there is a <script> block and a <div> element. The <div> element has the text content 'Texte dans la div !'. A red arrow points from the highlighted line of code in the left panel to this <div> element in the DOM tree.

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <script>...</script>
    ...   <div>Texte dans la div !</div> == $0
    </body>
  </html>
```

Il y a une seconde propriété qui permet l'ajout de texte, c'est “.textContent”. Cependant, il y a une légère différence entre ces 2 propriétés....

Javascript - insérer un texte

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span>Salut</span>
    </div>

    <script>
        // let div = document.createElement('div');
        // div.textContent = "Texte dans la div !";
        // document.body.append(div);

    </script>
</body>
</html>
```

Coucou Salut

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span style="display:none">Salut</span>
    </div>

    <script>
        // let div = document.createElement('div');
        // div.textContent = "Texte dans la div !";
        // document.body.append(div);

    </script>
</body>
</html>
```

Coucou

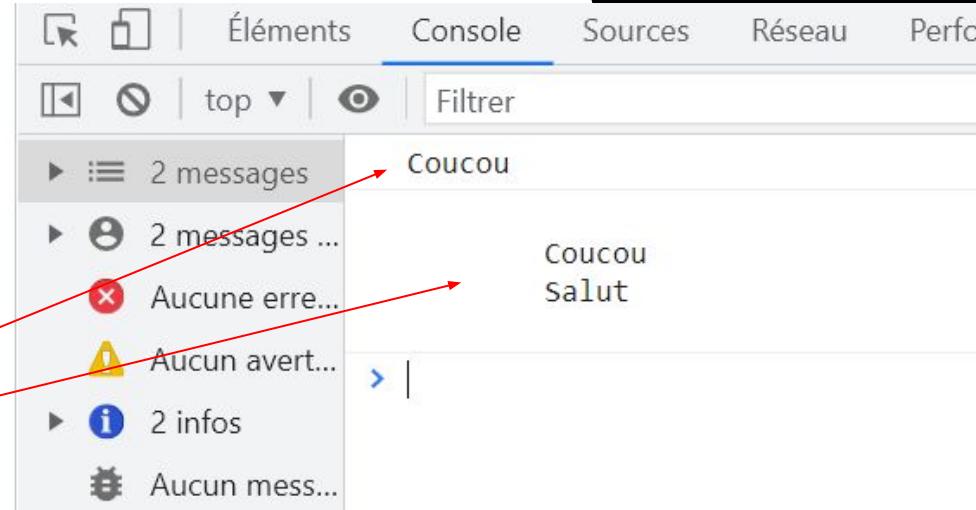
Pour mieux comprendre, je créé une “div” avec à l’intérieur 2 “span”. Dans chaque “span” j’écris un mot. La première “Coucou” et la deuxième “Salut”. Puis je mets un “display:none;” à la seconde “span”. Seul “Coucou” reste visible

Javascript - insérer un texte

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span style="display:none;">Salut</span>
    </div>

    <script>
        // let div = document.createElement('div');
        // div.textContent = "Texte dans la div !";
        // document.body.append(div);

        console.log(document.querySelector('div').innerText);
        console.log(document.querySelector('div').textContent);
    </script>
</body>
</html>
```



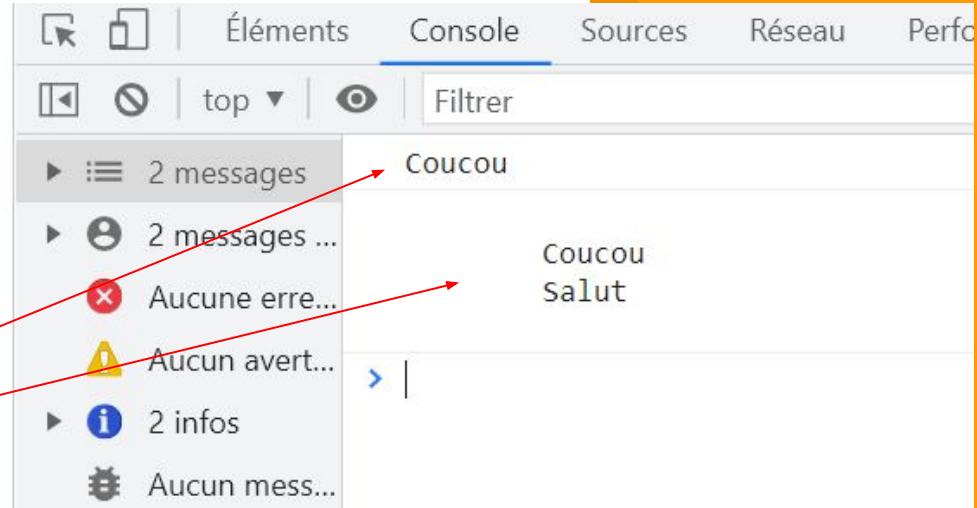
Deux choses à voir avant l'explication, la méthode “querySelector(“élémentAsélectionner”)” sert à sélectionner un élément de la page. A part le “body” et le “head”, il faudra forcément utiliser cette méthode pour sélectionner un élément. Cela fonctionne comme le CSS (querySelector('#monId .maClass')). Attention, cette méthode fonctionne si l’élément est unique que la page.

Javascript - insérer un texte

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span style="display:none;">Salut</span>
    </div>

    <script>
        // let div = document.createElement('div');
        // div.textContent = "Texte dans la div !";
        // document.body.append(div);

        console.log(document.querySelector('div').innerText);
        console.log(document.querySelector('div').textContent);
    </script>
</body>
</html>
```



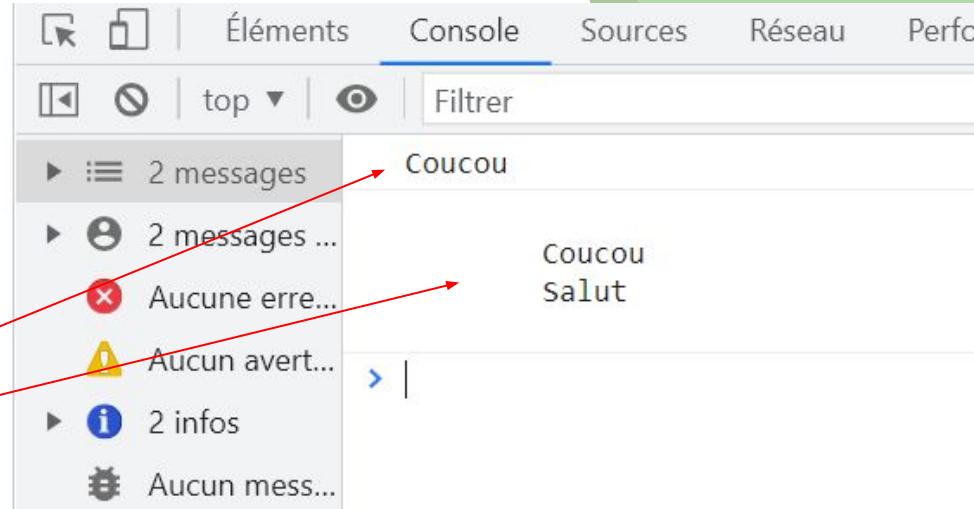
La deuxième chose c'est que, comme beaucoup d'autres propriétés, si “.textContent” ou “.innerText” n'ont pas d'assignation (Ex: `innerText = “un truc”`), au lieu d'ajouter un texte, ces propriétés vont récupérer un texte.

Javascript - insérer un texte

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span style="display:none;">Salut</span>
    </div>

    <script>
        // let div = document.createElement('div');
        // div.textContent = "Texte dans la div !";
        // document.body.append(div);

        console.log(document.querySelector('div').innerText);
        console.log(document.querySelector('div').textContent);
    </script>
</body>
</html>
```



Donc on peut voir que “.innerText” va récupérer le texte visible par l'utilisateur alors que “.textContent” va récupérer tout, même le texte caché par le style. De plus, il respecte l'indentation et la structure du HTML.

Javascript - insérer un texte

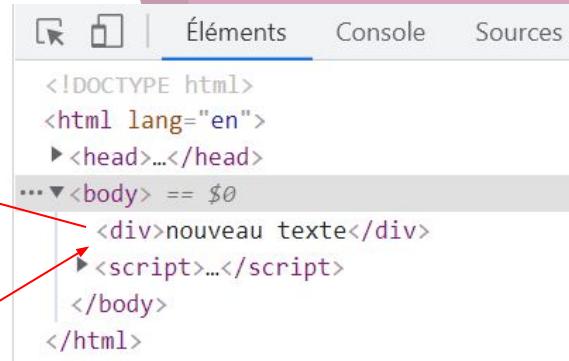
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=
      <title>Document</title>
    </head>
  <body>
    <div>
      <span>Coucou</span>
      <span style="display:none;">Salut</span>
    </div>

    <script>
      // let div = document.createElement('div');
      // div.textContent = "Texte dans la div !";
      // document.body.append(div);

      document.querySelector('div').innerText = "nouveau texte";

    </script>
  </body>
</html>
```

nouveau texte



The screenshot shows the browser's developer tools with the "Elements" tab selected. The DOM tree is displayed, starting with the <!DOCTYPE html> declaration, followed by the <html> element with lang="en". Inside the <html> element is the <head> and <body> sections. The <body> section contains a <div> element with the text "nouveau texte". Below the <div> is a <script> element. A red arrow points from the text "nouveau texte" in the code editor to the "nouveau texte" text node in the DOM tree. Another red arrow points from the "nouveau texte" text node in the DOM tree back to the corresponding line in the script editor.

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  ... <body> == $0
    <div>nouveau texte</div>
    <script>...</script>
  </body>
</html>
```

Bien entendu, si j'attribue une nouvelle valeur à la propriété “innerText” (ou “textContent”), il remplace le texte (et même le code) précédent.

Ajouter du HTML dans une page via JavaScript

Javascript - insérer du HTML

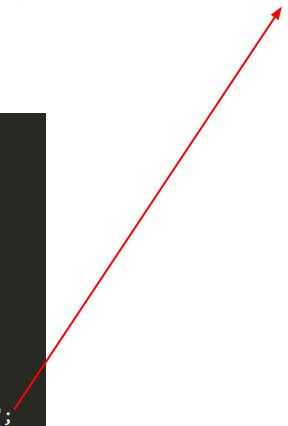
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span style="display:none;">Salut</span>
    </div>

    <script>
        // let div = document.createElement('div');
        // div.textContent = "Texte dans la div !";
        // document.body.append(div);

        document.querySelector('div').innerText = "<strong>nouveau texte</strong>";

    </script>
</body>
</html>
```

nouveau texte



Maintenant, si je souhaite insérer ce même texte en gras, je serais tenté de mettre des balises “``”....mais “`innerText`” (et “`textContent`”) ne gère pas les balises HTML.

Javascript - insérer du HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div>
      <span>Coucou</span>
      <span style="display:none;">Salut</span>
    </div>
    <script>
      document.querySelector('div').innerHTML = "<strong>nouveau texte</strong>";
    </script>
  </body>
</html>
```

nouveau texte



Pour l'insertion de balises HTML, on peut utiliser la propriété “innerHTML”
ATTENTION ! Mal utilisée, cette propriété peut rendre vulnérable votre site et permettre à un hacker d'insérer un code malveillant.

Javascript - insérer du HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div></div>

  <script>

    let strong = document.createElement('strong');
    strong.innerText = "nouveau texte"
    document.querySelector('div').append(strong);

  </script>
</body>
</html>
```

nouveau texte

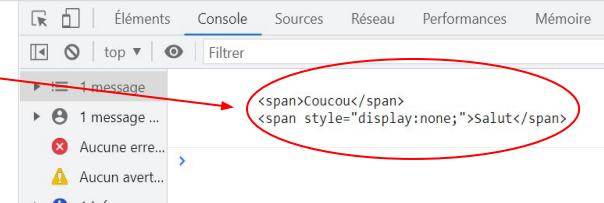
Une façon plus “safe” d’insérer du HTML dans une page est de créer un élément HTML en JS, puis d’insérer du contenu directement à l’intérieur. Enfin, insérer directement la balise HTML créée avec le contenu dans la “div”

Javascript - insérer du HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <span>Coucou</span>
        <span style="display:none;">Salut</span>
    </div>

    <script>
        console.log(document.querySelector('div').innerHTML)
    </script>
</body>
</html>
```

Coucou



De la même façon que “innerText” ou “textContent”, si j’utilise “innerHTML” seul, cette méthode ira me récupérer le code situé dans un élément souhaité.

Supprimer du HTML dans une page via JavaScript

Javascript - supprimer du HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div class="coucou">coucou</div>
    <div class="belette">belette</div>
    <div class="salut">salut</div>

    <script>
      const divBelette = document.querySelector('.belette');
      divBelette.remove();

    </script>
  </body>
</html>
```

coucou
salut



```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <div class="coucou">coucou</div>
    ...
    <div class="salut">salut</div> == $0
    <script>...</script>
  </body>
</html>
```

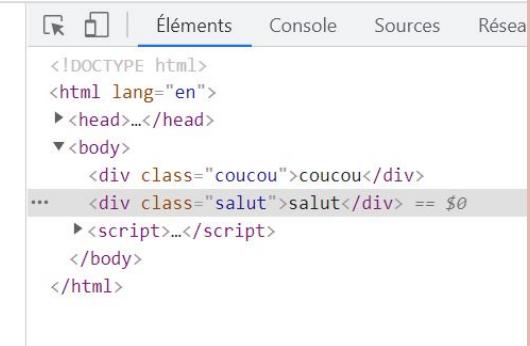
La propriété “remove()” permet de détruire un élément HTML.
Dans l'exemple ci-dessus, j'aurais pu écrire pour aller plus vite :
document.querySelector('.belette').remove();

Javascript - supprimer du HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div class="coucou">coucou</div>
      <div class="belette">belette</div>
      <div class="salut">salut</div>
    </div>

    <script>
      const container = document.querySelector('#container');
      const divBelette = document.querySelector('.belette');
      container.removeChild(divBelette);
    </script>
  </body>
</html>
```

coucou
salut



```
!DOCTYPE html
<html lang="en">
  <head>...</head>
  <body>
    <div class="coucou">coucou</div>
    ... <div class="salut">salut</div> == $0
    <script>...</script>
  </body>
</html>
```

Une autre méthode peut également détruire un élément HTML, c'est la méthode “removeChild(“élémentEnfantAdétruire”)”. Il faut l'utiliser directement sur l'élément parent et passer en paramètre, l'enfant à détruire. Autre syntaxe :
`document.querySelector('#container').removeChild(document.querySelector('.belette'));`

Manipuler les attributs via JavaScript

Javascript - manipuler les attributs

The screenshot shows a browser developer tools window. On the left, the DOM tree displays the following HTML code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
    </div>

    <script>
      const divCoucou = document.querySelector('.coucou');
      console.log(divCoucou.getAttribute('id'))
    </script>
  </body>
</html>
```

A red arrow points from the line `const divCoucou = document.querySelector('.coucou');` to the `coucou` element in the DOM tree. Another red arrow points from the line `console.log(divCoucou.getAttribute('id'))` to the `test` message in the console.

The DOM tree shows a `div` element with the `coucou` class, which contains the text "coucou". The console panel on the right shows the output of the script: "1 message" and "test".

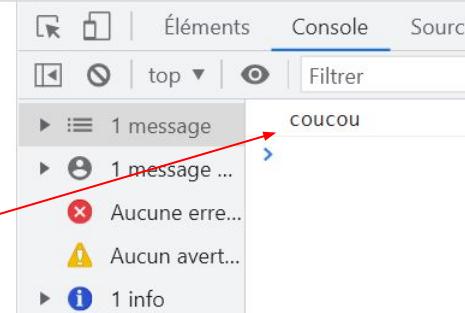
Pour lire un attribut, on utilise la méthode “`getAttribute('attributArécupérer')`” avec en paramètre l’attribut à lire. Dans l’exemple ci-dessus, l’attribut “`id`” de la “`divCoucou`”, est bien égal à “`test`”. Autre syntaxe possible : `document.querySelector('.coucou').getAttribute('id')`

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
    </div>

    <script>
      const divCoucou = document.querySelector('.coucou');
      console.log(divCoucou.getAttribute('class'))
    </script>
  </body>
</html>
```

coucou



Si je change d'attribut, nous récupérons bien sa valeur.

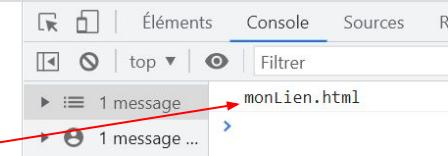
Autre syntaxe possible :

```
document.querySelector('.coucou').getAttribute('class')
```

Javascript - manipuler les attributs

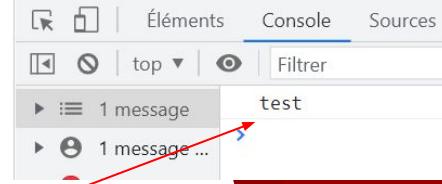
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a href="monLien.html">Lien</a>
    </div>
    <script>
      console.log(document.querySelector('#container a').getAttribute('href'));
    </script>
  </body>
</html>
```

coucou
[Lien](#)



Dernier exemple sur un lien en sélectionnant son attribut “href”

Javascript - manipuler les attributs



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="container">
        <div id="test" data-test="hjhjh" class="coucou">coucou</div>
        <a class="lien" href="monLien.html">Lien</a>
    </div>

    <script>
        const divCoucou = document.querySelector('.coucou');
        console.log(divCoucou.id);
    </script>
</body>
</html>
```

The screenshot shows a browser's developer tools open to the 'Console' tab. It displays two messages: 'coucou' and 'Lien'. Below the messages, the 'test' filter is applied, which is highlighted with a red arrow pointing from the explanatory text below.

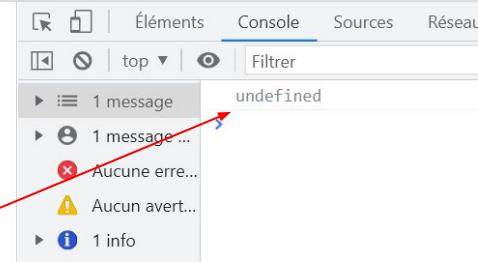
Il existe une syntaxe plus simple, c'est de mettre directement un point, puis l'attribut que vous cherchez à sélectionner (".id" pour l'id). Autre syntaxe : `document.querySelector('.coucou').id`

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien" href="monLien.html">Lien</a>
    </div>

    <script>
      const divCoucou = document.querySelector('.coucou');
      console.log(divCoucou.class);
    </script>
  </body>
</html>
```

coucou
[Lien](#)



Malheureusement, cette syntaxe ne fonctionne pas sur tous les attributs...

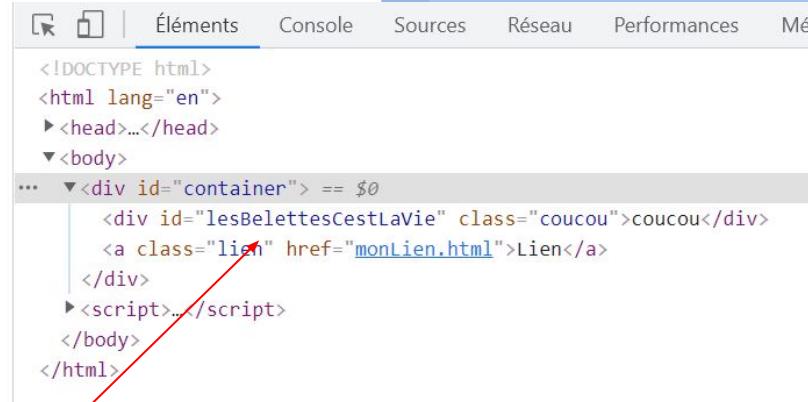
Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
  <title>Document</title>
</head>
<body>
  <div id="container">
    <div id="test" class="coucou">coucou</div>
    <a class="lien" href="monLien.html">Lien</a>
  </div>

  <script>
    document.querySelector('.coucou').setAttribute('id', 'lesBelettesCestLaVie');

  </script>
</body>
</html>
```

coucou
Lien



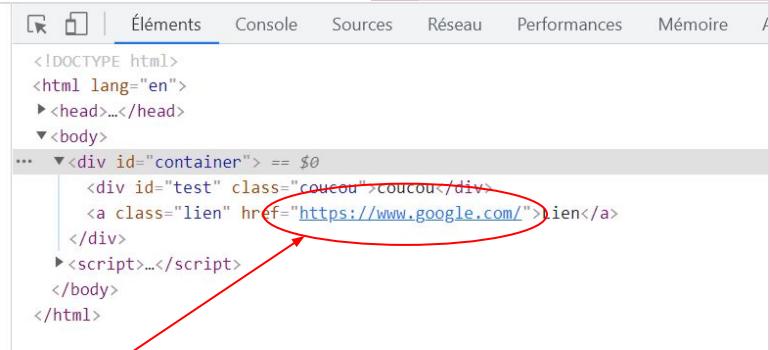
Pour modifier la valeur d'un attribut, il faut utiliser une autre méthode “`setAttribute('attributAchanger', 'nouvelleValeur')`” . Si l'attribut n'existe pas, il l'aurait créé.

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, i
        <title>Document</title>
    </head>
    <body>
        <div id="container">
            <div id="test" class="coucou">coucou</div>
            <a class="lien" href="monLien.html">Lien</a>
        </div>

        <script>
            document.querySelector('a').setAttribute('href', 'https://www.google.com/');
        </script>
    </body>
</html>
```

coucou
Lien



Dans cet exemple, je modifie la destination du lien.

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('.coucou').removeAttribute('id')
    </script>
  </body>
</html>
```

coucou
[Lien](#)



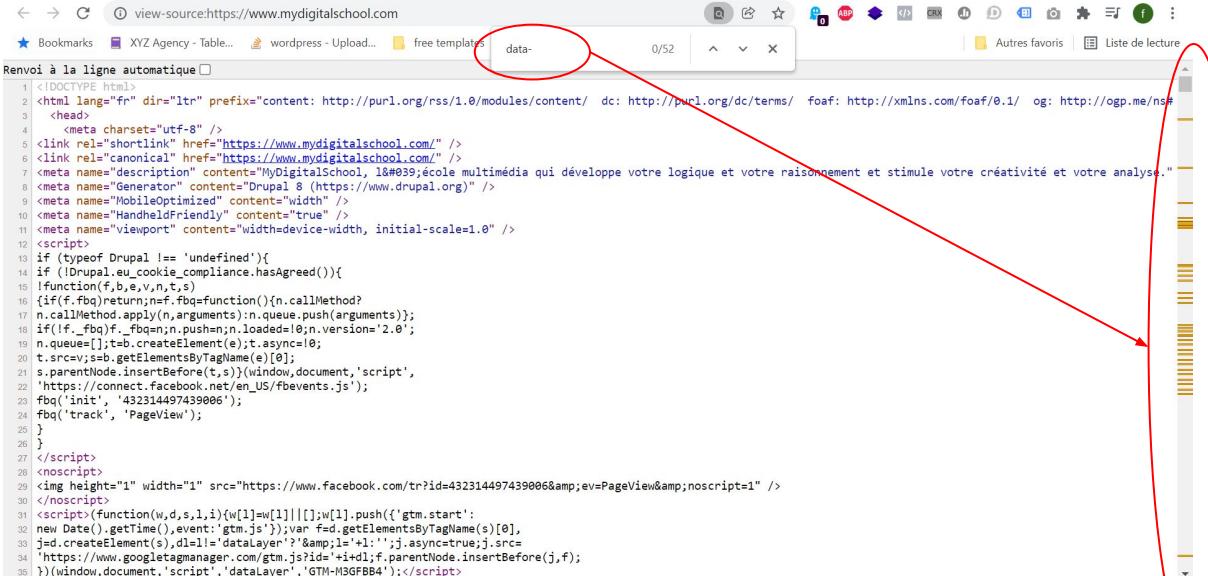
Bien sûr, si il est possible de récupérer, modifier un attribut, il est aussi possible de le supprimer avec la méthode "removeAttribute('élémentAdétruire')"

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a data-test="test" data-quantite="2" class="lien" href="monLien.html">Lien</a>
    </div>
    | 
</body>
</html>
```

Il y a aussi la possibilité d'ajouter plus d'attributs à n'importe quels éléments (notamment quand le site commence à être complexe). Pour ce faire, il existe les “data-attributes”. Ce sont des attributs comme les autres, à part qu'ils commencent tous par “data-”.
Syntaxe “data-nomQueVousVoulez = “votre valeur” ”

Javascript - manipuler les attributs



```
<!DOCTYPE html>
<html lang="fr" dir="ltr" prefix="content: http://purl.org/rss/1.0/modules/content/ dc: http://purl.org/dc/terms/ foaf: http://xmlns.com/foaf/0.1/ og: http://ogp.me/ns#>
<head>
<meta charset="utf-8" />
<link rel="shortlink" href="https://www.mydigitalschool.com/" />
<link rel="canonical" href="https://www.mydigitalschool.com/" />
<meta name="description" content="MyDigitalSchool, l'école multimédia qui développe votre logique et votre raisonnement et stimule votre créativité et votre analyse." />
<meta name="Generator" content="Drupal 8 (https://www.drupal.org)" />
<meta name="MobileOptimized" content="width" />
<meta name="HandheldFriendly" content="true" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<script>
if (typeof Drupal !== 'undefined'){
if (!Drupal.eu_cookie_compliance.hasAgreed()){
(function(f,b,e,v,n,t,s){
if(f.fbq) return; n=f.fbq=nfbq=function(){n.callMethod?
n.callMethod.apply(n,arguments):n.queue.push(arguments)};
if(!f._fbq)f._fbq=n;n.push=n.loaded=0;n.version='2.0';
n.queue=[];t=b.createElement(e);t.async=0;
t.src=s;b.getElementsByTagName(e)[0];
s.parentNode.insertBefore(t,s)}(window,document,'script',
'https://connect.facebook.net/en_US/fbevents.js');
fbq('init', '432314497439006');
fbq('track', 'PageView');
}
}
</script>
<noscript>

</noscript>
<script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
new Date(),event:'gtm.js'});var f=d.getElementsByTagName(s)[0],
j=d.createElement(s),dl=l['dataLayer']?'&'+l['dataLayer']:';j.async=true;j.src=
['https://www.googletagmanager.com/gtm.js?id='+i+dl,f.parentNode.insertBefore(j,f);
})(window,document,'script','dataLayer','GTM-M3GFBB4');
```

Dans cet exemple, les attributs "data" sont très utiles car ils permettent de mettre des informations récupérables en JavaScript

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.
  <title>Document</title>
</head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a data-test="test" data-quantite="2" class="lien" href="monLien.html">Lien</a>
    </div>

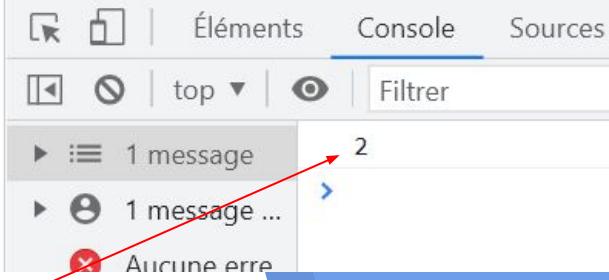
    <script>
      console.log(document.querySelector('a').dataset)
    </script>
  </body>
</html>
```

coucou
[Lien](#)



Pour pouvoir accéder aux attributs “data”, il existe la propriété “dataset” qui n'est ni plus, ni moins un objet recensant tous les attributs “data” d'un élément choisi.

Javascript - manipuler les attributs

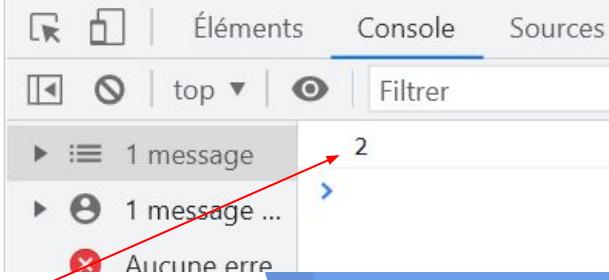


```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a data-test="test" data-quantite="2" class="lien" href="monLien.html">Lien</a>
    </div>

    <script>
      console.log(document.querySelector('a').dataset.quantite)
    </script>
  </body>
</html>
```

The code shows an HTML document with a container div containing a test div and a link. The link has a data-quantite attribute set to 2. A script logs the value of this attribute to the console.

coucou
Lien



Après "dataset", il me suffit de taper le nom de l'attribut "data" pour le récupérer.

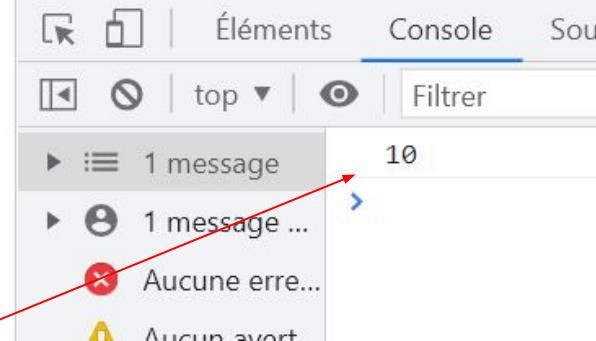
Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
      <title>Document</title>
    </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a data-test="test" data-quantite="2" class="lien" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('a').dataset.quantite = 10
      console.log(document.querySelector('a').dataset.quantite)

    </script>
  </body>
</html>
```

coucou
[Lien](#)



Comme toutes propriétés, je peux lui assigner une nouvelle valeur....

Javascript - manipuler les attributs

The screenshot shows a browser window with developer tools open. The 'Elements' tab is selected, displaying the DOM tree. The page structure is as follows:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a data-test="test" data-quantite="2" class="lien" href="monLien.html" data-belette="animal trop souvent oublé">Lien</a>
    </div>

    <script>
      document.querySelector('a').dataset.belette = "animal trop souvent oublé"
    </script>
  </body>
</html>
```

A red oval highlights the line of code in the script block: `document.querySelector('a').dataset.belette = "animal trop souvent oublé"`. A red arrow points from this oval to the corresponding attribute value in the DOM tree: `data-belette="animal trop souvent oublé"`.

Je peux lui également créer un nouvel attribut (comme on crée une nouvelle clé dans un tableau et une nouvelle valeur)

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, ini
      <title>Document</title>
    </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a data-test="test" data-quantite="2" class="lien" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('a').removeAttribute('data-quantite')

    </script>
  </body>
</html>
```

coucou
[Lien](#)



Et pour supprimer un “data-attribut”, il suffit d’utiliser la méthode
“removeAttribute(nomDuData-attributAdétruire)”

Javascript - manipuler les attributs

The screenshot shows a browser's developer tools. On the left, the DOM tree is displayed with several nodes highlighted in red. A red arrow points from the highlighted 'class' attribute in the DOM tree to the corresponding line of code in the code editor on the right. The code editor contains the following script:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('a').setAttribute('class', 'maClass');
    </script>
  </body>
</html>
```

Maintenant, voyons comment manipuler les “class” (car oui, c'est un peu spécial...). Dans l'exemple ci-dessus, on voit que mon élément “a” possède 2 “class”. Si je veux en ajouter une, je serais tenté d'utiliser la méthode “`setAttribute()`”. Malheureusement, cette méthode va juste supprimer les “class” existantes et les remplacer par l'autre...

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('a').setAttribute('class', 'maClass');
    </script>
  </body>
</html>
```

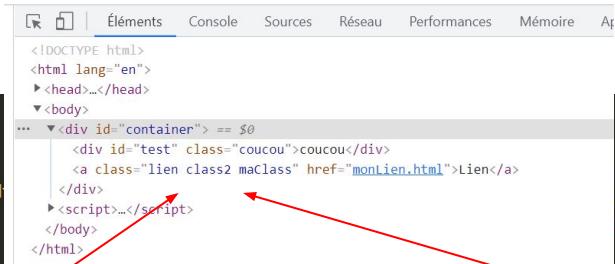
```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    ... <div id="container"> == $0
      <div id="test" class="coucou">coucou</div>
      <a class="maClass" href="monLien.html">Lien</a>
    </div>
    <script> document.querySelector('a').setAttribute('class', 'maClass'); </script>
  </body>
</html>
```

Alors, vous allez me dire “oui mais quand il n'y a qu'une ‘class’, c'est possible...” Et c'est pas faux mais souvent il y a plusieurs “class” dans cet attribut....

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div id="container">
    <div id="test" class="coucou">coucou</div>
    <a class="lien class2" href="monLien.html">Lien</a>
  </div>

  <script>
    const classes = document.querySelector('a').getAttribute('class');
    document.querySelector('a').setAttribute('class', classes+' maClass');
  </script>
</body>
</html>
```



```
<body>
  <div id="container">
    <div id="test" class="coucou">coucou</div>
    <a class="lien class2" href="monLien.html">Lien</a>
  </div>

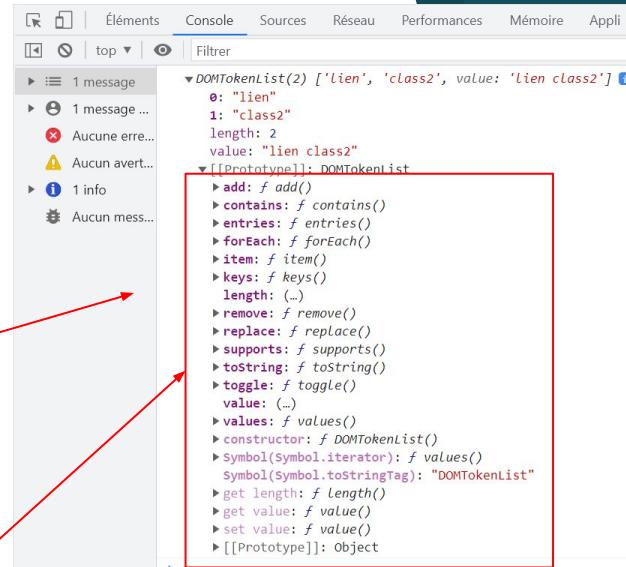
  <script>
    document.querySelector('a').setAttribute('class', 'lien class2 maClass');
  </script>
</body>
</html>
```

D'autres pourraient me dire “et si j'ajoute les “class” existantes plus ma nouvelle “class” dans la méthode “setAttribute()”.... Oui, ça pourrait passer....au début.....

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        console.log(document.querySelector('a').classList);
    </script>
</body>
</html>
```



La propriété “classList” est là pour ça !
Elle possède de nombreuses méthodes plus simples et très utiles !

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-sca
    <title>Document</title>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        document.querySelector('a').classList.add('maClass');
    </script>
</body>
</html>
```

coucou
Lien



The screenshot shows the browser's developer tools with the "Éléments" (Elements) tab selected. The DOM tree is visible, starting with the <html> element. Inside the <body> element, there is a <div> with the id "container". Within "container", there is a <div> with the id "test" and class "coucou", containing the text "coucou". Next to it is an <a> tag with the class "lien class2" and href "monLien.html", labeled "Lien". A script block is present at the bottom of the body, containing the line "document.querySelector('a').classList.add('maClass');". This line is highlighted with a red arrow pointing from the explanatory text below.

La méthode “add(“classAajouter”) ajoute simplement une “class” en plus (et ne détruit pas les “class” existantes)

Javascript - manipuler les attributs

The screenshot shows a browser's developer tools with the "Éléments" (Elements) tab selected. The left panel displays the HTML code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('a').classList.remove('class2');
    </script>
  </body>
</html>
```

The right panel shows the DOM tree under the "body" node. A red arrow points from the "class2" class in the code to the "class2" class in the DOM tree. Another red arrow points from the "remove('class2')" method in the script to the "class2" class in the DOM tree.

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <div id="container"> == $0
      <div id="test" class="coucou">coucou</div>
      <a class="lien" href="monLien.html">Lien</a>
    </div>
    <script> document.querySelector('a').classList.remove('class2'); </script>
  </body>
</html>
```

La méthode “remove(“classAdétaire”) supprime simplement une “class” (sans détruire les autres “class” existantes)

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
      a.classToggle{
        color: red;
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien class2" href="nonLien.html">Lien</a>
    </div>

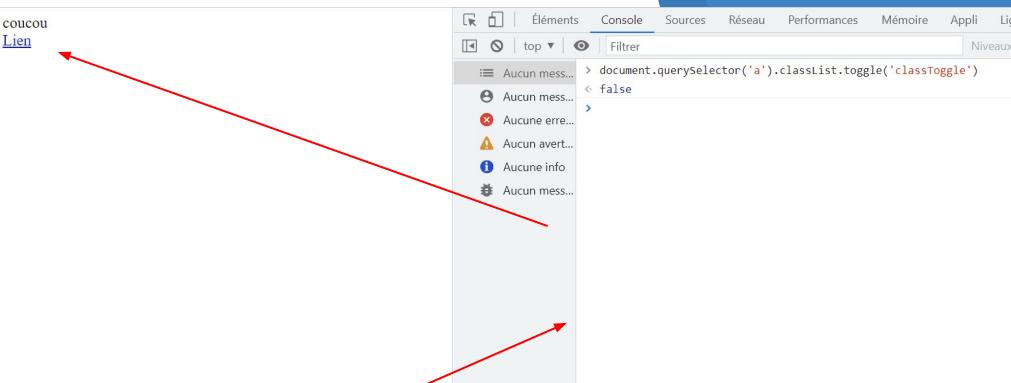
    <script>
      document.querySelector('a').classList.toggle('classToggle');
    </script>
  </body>
</html>
```

La méthode “`toggle(“classAjouterOuSupprimer”)`” est très utile ! Elle permet d’ajouter une “`class`” si elle n’existe pas OU de la supprimer si elle existe. Pour pouvoir voir son effet, j’ai ajouté du CSS qui dit ”Si la balise ‘`a`’ a la ‘`class`’ ‘`toggleClass`’ , alors le lien s’affichera en rouge, sinon, couleur de base....”

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        a.classToggle{
            color: red;
        }
    </style>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        document.querySelector('a').classList.toggle('classToggle');
    </script>
</body>
</html>
```



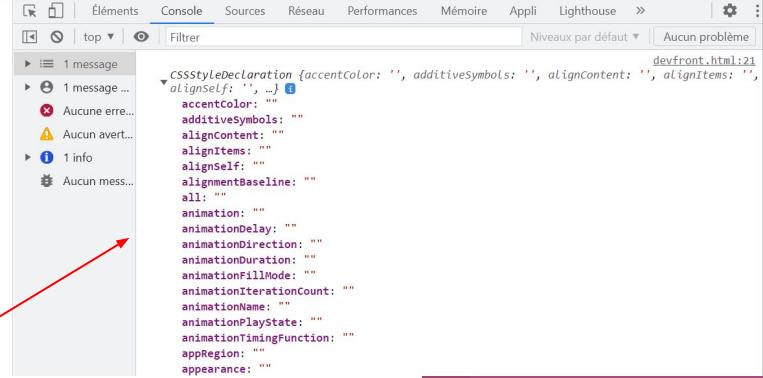
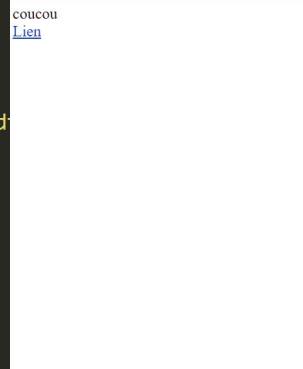
On voit bien que lorsque que j'exécute la méthode “toggle()”, le JS ajoute puis supprime la “class” “toggleClass” car le lien change de couleur.

Manipuler le style via JavaScript

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        a.classToggle{
            color: red;
        }
    </style>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        console.log(document.querySelector('a').style);
    </script>
</body>
</html>
```

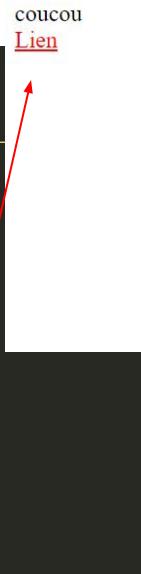


De même que JS nous met à disposition une propriété pour manipuler les “class”, il nous met également une propriété pour le style.....
La propriété “style” vous donne accès à toutes les propriétés CSS que vous connaissez.

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
    <title>Document</title>
    <style>
        .classToggle{
            color: red;
        }
    </style>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        document.querySelector('a').style.color = "red";
    </script>
</body>
</html>
```



```
<!DOCTYPE html>
...<html lang="en"> == $0
▶ <head>...</head>
▼ <body>
    ▼<div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html" style="color: red;">Lien</a>
    </div>
    <script> document.querySelector('a').style.color = "red"; </script>
</body>
</html>
```

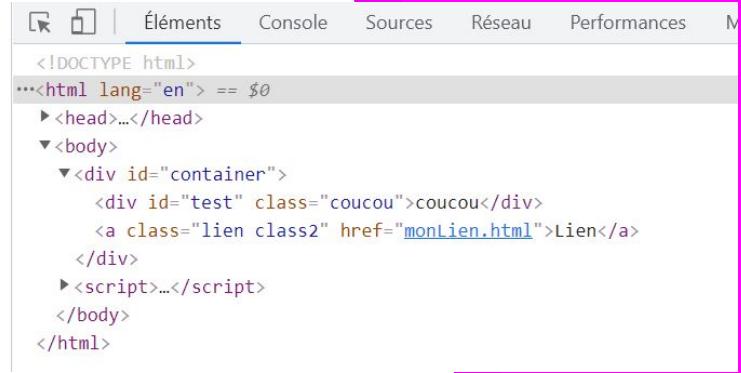
Il suffit simplement d'ajouter la propriété CSS que vous souhaitez modifier et lui assigner une nouvelle valeur. Exemple avec “color”. On peut voir dans la console qui lui a ajouté une balise “style”.

Javascript - manipuler les attributs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, i
<title>Document</title>
<style>
    .classToggle{
        color: red;
    }
</style>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
        document.querySelector('a').style.color = "red";
        document.querySelector('a').style.text-decoration = "none";
    </script>
</body>
</html>
```

coucou
[Lien](#)



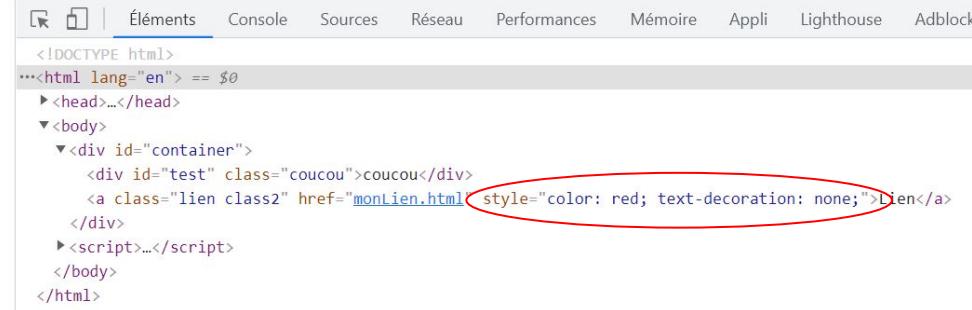
Si je veux lui modifier une valeur CSS qui s'écrive avec le symbole “-”, cela ne fonctionnera pas. Exemple “text-decoration”

Javascript - manipuler les attributs



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
      .classToggle{
        color: red;
      }
    </style>
  </head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien class2" href="monLien.html">Lien</a>
    </div>

    <script>
      document.querySelector('a').style.color = "red";
      document.querySelector('a').style.textDecoration = "none";
    </script>
  </body>
</html>
```



```
<!DOCTYPE html>
<html lang="en"> == $0
  <head>...</head>
  <body>
    <div id="container">
      <div id="test" class="coucou">coucou</div>
      <a class="lien class2" href="monLien.html" style="color: red; text-decoration: none;">Lien</a>
    </div>
    <script>...</script>
  </body>
</html>
```

Pour faire passer des propriétés CSS qui contiennent des “-”, vous devrez supprimer l’écrire en “chameau” (camel case). “text-decoration” deviendra “textDecoration”. “margin-top” deviendra “marginTop”. “padding-left” deviendra “paddingLeft”....

Javascript - manipuler les attributs

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7   <style>
8     .classToggle{
9       color: red;
10    }
11   </style>
12 </head>
13 <body>
14   <div id="container">
15     <div id="test" class="coucou">coucou</div>
16     <a class="lien class2" href="monLien.html" style="color: white; text-decoration: none; background: black; padding: 10px 15px; display: block; width: 30px;">Lien</a>
17   </div>
18
19   <script>
20     document.querySelector('a').style.cssText = "color: white; text-decoration: none; background: black; padding: 10px 15px; display: block; width: 30px;";
21   </script>
22 </body>
23 </html>
```

coucou
Lien



Si vous avez beaucoup de propriétés à modifier, écrire pour chaque propriété “document.querySelector('a').style.proprieteCSS = valeur” peut être loooonnnnnng ! Vous pouvez alors utiliser la propriété “cssText” et lui assigner comme valeur toutes les propriétés / valeurs que vous voulez (comme dans un attribut “style” classique)

Javascript - manipuler les attributs

The screenshot shows a browser window with developer tools open. On the left, the page content displays "coucou" in a white font on a black background, followed by a black rectangular button labeled "Lien". On the right, the browser's element inspector shows the DOM structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
    a.classToggle{
        color: red;
    }
</style>
</head>
<body>
    <div id="container">
        <div id="test" class="coucou">coucou</div>
        <a class="lien class2" href="monLien.html" style="color: white; text-decoration: none; background: black; padding: 10px 15px; display: block; width: 30px;">Lien</a>
    </div>
</body>
</html>
```

A red arrow points from the "style" attribute of the anchor tag in the DOM tree to the corresponding line of code in the script block of the browser screenshot.

Ou alors, on peut lui ajouter un attribut “style” avec la méthode
“`setAttribute(“style”, “styleAajouter...”)`”

querySelector **&** **querySelectorAll**

Javascript - querySelector & querySelectorAll

```
document.querySelector('monElementASélectionner')
document.querySelector('monID')
document.querySelector('maClass')
document.querySelector('#id .maClass p')
```

Nous avons vu plus tôt que la méthode “querySelector(“element”)” permet de sélectionner un élément (par sa “class”, son “ID”, son type d’élément HTML,...) présent dans la page.

Javascript - querySelector & querySelectorAll

```
document.getElementById('JeDoisForcementMettreUnID')
document.getElementsByClassName('jeDoisForcementMettreUneClass')
document.getElementsByTagName('jeDoisForcementMettreUnElementHTML')
```

Avant la méthode “querySelector(“element”)", nous avions une méthode pour sélectionner soit une “class”, soit un “ID”, soit un élément HTML. La méthode “querySelector()" est nouvelle par rapport aux autres.

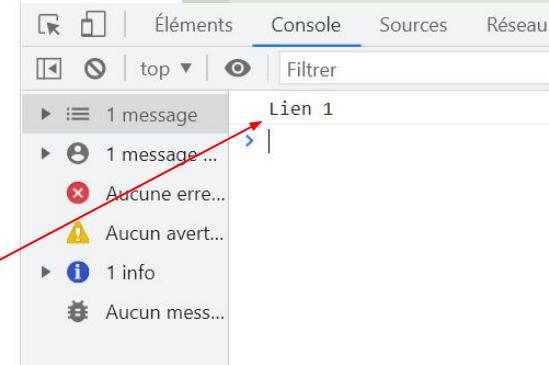
Javascript - querySelector & querySelectorAll

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <a href="#" class="unLien">Lien 1</a>
    <a href="#" class="unLien">Lien 2</a>
    <a href="#" class="unLien">Lien 3</a>
    <a href="#" class="unLien">Lien 4</a>
    <a href="#" class="unLien">Lien 5</a>

    <script>
        console.log(document.querySelector('a.unLien').innerText);
    </script>
</body>
</html>
```

[Lien 1](#) [Lien 2](#) [Lien 3](#) [Lien 4](#) [Lien 5](#)



Le problème avec la méthode “querySelector()” est que si je souhaite sélectionner plusieurs éléments.....bah je peux pas. Cette méthode s’arrête au premier lien. Alors comment faire pour afficher tous les textes présents dans chaque lien ?

Javascript - querySelector & querySelectorAll

The screenshot shows a browser's developer tools open to the 'Console' tab. A red arrow points from the explanatory text below to the 'script' block in the code editor on the left, and another red arrow points from the 'script' block to the NodeList object in the console output on the right.

Lien 1 Lien 2 Lien 3 Lien 4 Lien 5

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <a href="#" class="unLien">Lien 1</a>
    <a href="#" class="unLien">Lien 2</a>
    <a href="#" class="unLien">Lien 3</a>
    <a href="#" class="unLien">Lien 4</a>
    <a href="#" class="unLien">Lien 5</a>

    <script>
        console.log(document.querySelectorAll('a.unLien'));
    </script>
</body>
</html>
```

Console output:

```
1 message
1 message ...
Aucune erreur...
Aucun avertissement...
1 info
Aucun message...

▼ NodeList(5) [a.unLien, a.unLien, a.unLien, a.unLien, a.unLien]
▶ 0: a.unLien
▶ 1: a.unLien
▶ 2: a.unLien
▶ 3: a.unLien
▶ 4: a.unLien
length: 5
[[Prototype]]: NodeList
```

La solution est la méthode “querySelectorAll(‘groupeDelements’)” car cette méthode va générer un tableau de tous ces éléments

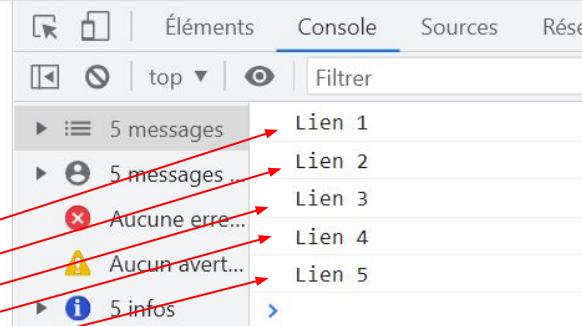
Javascript - querySelector & querySelectorAll

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <a href="#" class="unLien">Lien 1</a>
    <a href="#" class="unLien">Lien 2</a>
    <a href="#" class="unLien">Lien 3</a>
    <a href="#" class="unLien">Lien 4</a>
    <a href="#" class="unLien">Lien 5</a>

<script>
    console.log(document.querySelectorAll('a.unLien')[0].innerText);
    console.log(document.querySelectorAll('a.unLien')[1].innerText);
    console.log(document.querySelectorAll('a.unLien')[2].innerText);
    console.log(document.querySelectorAll('a.unLien')[3].innerText);
    console.log(document.querySelectorAll('a.unLien')[4].innerText);
</script>
</body>
</html>
```

Lien 1 Lien 2 Lien 3 Lien 4 Lien 5



Comme n'importe quel tableau, je peux mettre un index pour sélectionner l'index de l'élément que je souhaite sélectionner pour ensuite récupérer le texte. Bon, là c'est marrant, j'ai 5 éléments, mais si j'en avais 300, je vais pas les faire un par un. Par hasard, on connaît pas moyen lus rapide de parcourir un tableau ??????????

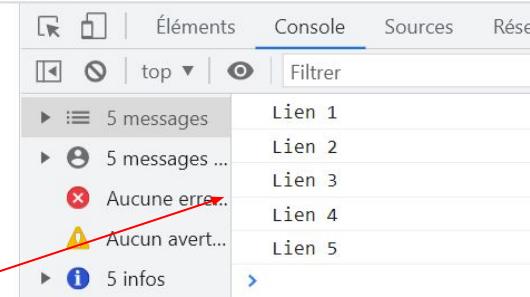
Javascript - querySelector & querySelectorAll

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>

    <a href="#" class="unLien">Lien 1</a>
    <a href="#" class="unLien">Lien 2</a>
    <a href="#" class="unLien">Lien 3</a>
    <a href="#" class="unLien">Lien 4</a>
    <a href="#" class="unLien">Lien 5</a>

  <script>
    const tableauDeLiens = document.querySelectorAll('a.unLien');
    tableauDeLiens.forEach(function(element, index) {
      console.log(element.innerText);
    });
  </script>
</body>
</html>
```

Lien 1 Lien 2 Lien 3 Lien 4 Lien 5



La boucle forEach !!! (ou autre du moment que c'est une boucle)

Javascript - querySelector & querySelectorAll

```
//Je stocke tous mes éléments dans un tableau "tableauDeLiens"  
const tableauDeLiens = document.querySelectorAll('a.unLien');  
//Je parcours ce tableau  
tableauDeLiens.forEach( function(element, index) {  
    // "element" est égal à la ligne du tableau en cours (  
    document.querySelectorAll('a.unLien')[indexDeLaLigne])  
    //Avec la propriété "innerText", on va récupérer le texte présent dans le lien ligne par ligne  
    console.log(element.innerText);  
});
```

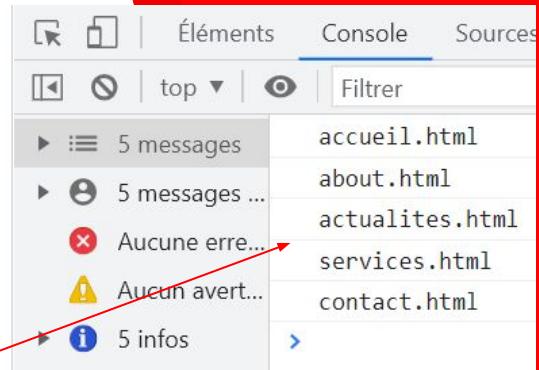
Explications

Javascript - querySelector & querySelectorAll

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, ini
<title>Document</title>
</head>
<body>
    <a href="accueil.html" class="unLien">Lien 1</a>
    <a href="about.html" class="unLien">Lien 2</a>
    <a href="actualites.html" class="unLien">Lien 3</a>
    <a href="services.html" class="unLien">Lien 4</a>
    <a href="contact.html" class="unLien">Lien 5</a>

    <script>
        const tableauDeLiens = document.querySelectorAll('a.unLien');
        tableauDeLiens.forEach( element, index) {
            console.log(element.getAttribute('href'));
        };
    </script>
</body>
</html>
```

Lien 1 Lien 2 Lien 3 Lien 4 Lien 5



Autre exemple où on utilise la méthode “getAttribute()” pour sélectionner le texte de l'attribut “href”

Javascript - querySelector & querySelectorAll

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <a href="accueil.html" class="unLien">Lien 1</a>
    <a href="about.html" class="unLien">Lien 2</a>
    <a href="actualites.html" class="unLien">Lien 3</a>
    <a href="services.html" class="unLien">Lien 4</a>
    <a href="contact.html" class="unLien">Lien 5</a>

    <script>
        const tableauDeLiens = document.querySelectorAll('a.unLien');
        tableauDeLiens.forEach( element, index) {
            element.setAttribute('href', element.innerText)
        });
    </script>

</body>
</html>
```

[Lien 1](#) [Lien 2](#) [Lien 3](#) [Lien 4](#) [Lien 5](#)

The screenshot shows the browser's developer tools with the "Éléments" (Elements) tab selected. The DOM tree is displayed, starting with the <html> tag. Inside the <body> tag, there are five anchor elements (<a>) each with a href attribute set to "Lien 1" through "Lien 5" respectively, and a class attribute set to "unLien". A script block is also present in the body.

```
<!DOCTYPE html>
<html lang="en">
    <head>...</head>
    <body>
        <a href="Lien_1" class="unLien">Lien 1</a>
        <a href="Lien_2" class="unLien">Lien 2</a>
        <a href="Lien_3" class="unLien">Lien 3</a>
        <a href="Lien_4" class="unLien">Lien 4</a>
        <a href="Lien_5" class="unLien">Lien 5</a>
        <script>...</script>
    </body>
</html>
```

Dans l'exemple ci-dessus, je récupère le texte cliquable de chaque lien et je l'insère dans chaque attribut “href”

Les écouteurs d'événements

Javascript - EventListener

```
Element_a_ecouter.addEventListener('action', function(e){  
    //Code à exécuter quand action à lieu  
})
```

Javascript - EventListener

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div class="lien">Coucou</div>

    <script>
      const div = document.querySelector('div.lien');
      div.addEventListener('click', function(e){
        alert(div.innerText);
      })
    </script>
  </body>
</html>
```

Coucou

Dans cet exemple, on sélectionne l'élément HTML “div” que l'on stocke dans une variable “div”. Ensuite, on utilise la méthode d'écoute d'événements “addEventListener('action', 'function callback')” avec comme action “click” (donc quand je clique sur la “div”) et une fonction anonyme qui sera exécutée au moment où la “div” sera cliquée (donc lancer une “alert” avec comme message le texte présent dans la “div”)

Javascript - EventListener

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div class="lien">Coucou</div>

    <script>
      const div = document.querySelector('div.lien');
      div.addEventListener('mouseover', function(e){
        alert(div.innerText);
      })
    </script>
  
```

Console

L'événement “mouseover” est l'équivalent de “hover” en CSS

Javascript - EventListener

```
<option value="Wallis and Futuna">Wallis and Futuna</option>
<option value="Western Sahara">Western Sahara</option>
<option value="Yemen">Yemen</option>
<option value="Zambia">Zambia</option>
<option value="Zimbabwe">Zimbabwe</option>

</select>

<script>
    document.querySelector('select').addEventListener('change', function(e){
        alert(document.querySelector('select').value);
    })
</script>

</body>
</html>
```

Afghanistan ▾

L'événement “change” détecte le moindre changement sur un élément...

Vous avez la liste des événements :

<https://developer.mozilla.org/fr/docs/Web/Events>

Javascript - EventListener

```
<script>
    //Attend que la partie HTML soit chargé pour être exécuté
    document.addEventListener('DOMContentLoaded', function(){
        //Tout le code ici sera exécuté quand le reste de la page sera chargée

        document.querySelector('select').addEventListener('change', function(e){
            alert(document.querySelector('select').value);
        })

    })
</script>
```

D'ailleurs, très important, tout le code JS doit être à l'intérieur d'un écouteur déclenché par l'événement “DOMContentLoaded”. Cela évite les erreurs de chargement. Imaginons que le JS charge un écouteur d'événement sur un élément HTML qui n'est pas encore chargé.... Vous aurez une erreur !

Javascript - EventListener

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>

    <a href="#" class="lien">Lien 1</a>
    <a href="#" class="lien">Lien 2</a>
    <a href="#" class="lien">Lien 3</a>
    <a href="#" class="lien">Lien 4</a>
    <a href="#" class="lien">Lien 5</a>

  </body>
</html>

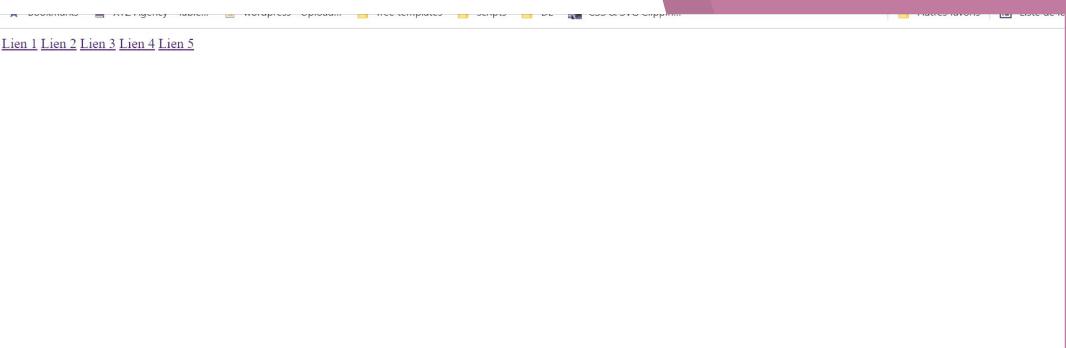
<script>
  document.addEventListener('DOMContentLoaded', function(){

    document.querySelector('a.lien').addEventListener('change', function(e){
      e.preventDefault();
      alert(document.querySelector('a.lien').innerText);
    })
  })
</script>
```

On a vu comment écouter un élément HTML, mais comment écouter plusieurs éléments HTML ????
Dans l'exemple ci-dessus, rien ne fonctionnera.... :(

Pour info, la méthode “preventDefault()” annule le comportement par défaut d'un élément. (ex: un lien cliqué nous envoie vers une autre page, avec cette méthode, il se passera rien)

Javascript - EventListener



```
document.addEventListener('DOMContentLoaded', fu  
  
    //Je mets mon groupe d'élément dans un tableau  
    const a = document.querySelectorAll('.lien');  
    //Je parcours mon tableau grâce à ma boucle forEach  
    a.forEach( function(element, index) {  
        //Sur chaque élément, je mets un écouteur d'événement au "click"  
        element.addEventListener('click',function(e){  
            //Je bloque le comportement par défaut du lien  
            e.preventDefault();  
            //Je lance une "alert" avec comme texte le texte cliquable.  
            alert(this.innerText);  
        })  
    });  
})  
</script>
```

Tadaaaa ! Et là, vous vous dîtes..... "PU****, c'est quoi ce "this" encore..."

Javascript - EventListener

```
<a href="#" class="lien">Lien 1</a>
<a href="#" class="lien">Lien 2</a>
<a href="#" class="lien">Lien 3</a>
<a href="#" class="lien">Lien 4</a>
<a href="#" class="lien">Lien 5</a>
```

Je suis obligé d'utiliser ‘this’, car il détermine quel élément parmi tous ceux qui sont sur “écoute” a été cliqué. Il permet de sélectionner précisément l’élément qui a reçu le “click” et donc d’en extraire le texte cliquable (l’ancre)