

# L'API DOM

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. Introduction à l'API DOM</b>	<b>3</b>
<b>III. Exercice : Appliquez la notion</b>	<b>6</b>
<b>IV. Les sélecteurs</b>	<b>7</b>
<b>V. Exercice : Appliquez la notion</b>	<b>11</b>
<b>VI. Accéder aux attributs</b>	<b>12</b>
<b>VII. Exercice : Appliquez la notion</b>	<b>15</b>
<b>VIII. Modifier le DOM</b>	<b>16</b>
<b>IX. Exercice : Appliquez la notion</b>	<b>19</b>
<b>X. Auto-évaluation</b>	<b>20</b>
A. Exercice final .....	20
B. Exercice : Défi .....	22
<b>Solutions des exercices</b>	<b>27</b>

## I. Contexte

**Durée :** 1 h

**Environnement de travail :** Repl.it

**Pré-requis :** Connaissance du HTML et des bases du JavaScript

### Contexte

Une Interface de Programmation d'Application, dite **API**, est un outil regroupant un ensemble de classes, de propriétés et de méthodes. Elles ont pour but de faciliter le développement d'applications en permettant aux développeurs de ne pas réinventer la roue.

Dans le cadre du développement d'applications ou sites web, l'API DOM permet de faire le lien entre un document web (HTML, CSS) et les scripts (JavaScript) qui manipuleront et modifieront le contenu du document. Le DOM apporte au JavaScript la notion de page web, sans laquelle il n'aurait aucun intérêt.

## II. Introduction à l'API DOM

### Objectifs

- Savoir définir le DOM
- Savoir définir un nœud
- Décrire le DOM

### Mise en situation

Une page web est un document HTML pouvant être affiché dans un navigateur. Le DOM permet de représenter cette page sous la forme d'un ensemble d'objets. On le représente généralement sous la forme d'un **arbre** dans lequel les ramifications sont les éléments HTML et CSS. Nous allons voir ici les éléments qui le composent.

### Définition

Le **DOM** (pour *Document Object Model*) est une interface de programmation pour les documents HTML et XML. Il fournit une page dont les programmes peuvent modifier la structure, son style et son contenu. Cette représentation du document permet de le voir comme un groupe structuré (un arbre) de nœuds et d'objets possédant différentes propriétés et méthodes. Fondamentalement, il relie les pages web aux scripts ou langages de programmation. Il est régi par les normes de la W3C.

Tous les éléments d'un document, c'est-à-dire le document lui-même, les en-têtes, les blocs de contenu `<div>` `</div>`, les paragraphes `<p>` `</p>`, le style `color : #FFFFFF`, le texte et bien d'autres font partie du DOM.

### Exemple

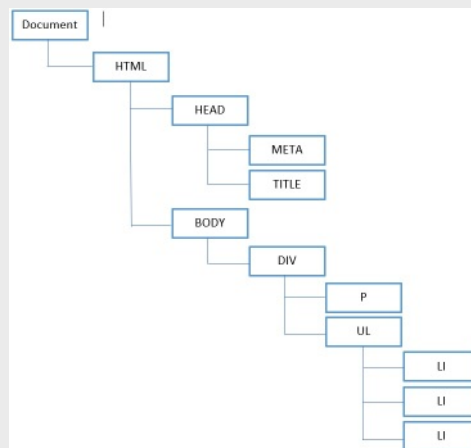
```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Document web</title>
6   </head>
7   <body>
```

```

8   <div>
9   <p>J'appartiens au DOM</p>
10  <ul>
11    <li>Document</li>
12    <li>Object</li>
13    <li>Model</li>
14  </ul>
15 </div>
16 </body>
17 </html>

```

La page web représentée par le code HTML ci-dessus peut être modélisée par le schéma ci-dessous.



### Méthode Comment accéder au DOM ?

Il n'y a rien à installer ou à télécharger pour manipuler le DOM. C'est notre navigateur qui implémente un DOM pour rendre accessibles nos pages web à JavaScript. Nous pouvons donc commencer à utiliser l'API.

Les points d'entrée principaux sont les éléments **document** et **window** :

- **document** représente le contenu de la page web, autrement dit l'arborescence du DOM : nous allons grâce à lui parcourir les éléments, en ajouter ou en supprimer.
- **window** représente la fenêtre du navigateur, dans laquelle est chargé un document.

### Exemple Window

```

1 window.addEventListener('load', () => {
2   window.alert('Bonjour')
3 })

```

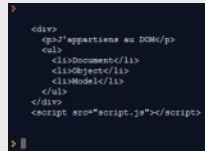
Dans cet exemple, la popup avec le message "Bonjour" sera affichée quand la page web sera entièrement chargée.



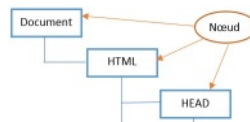
**Exemple Document**

```
1 window.addEventListener('load', () => {
2   console.log(document.body.innerHTML)
3 })
```

Dans cet exemple, nous accédons à notre `<body></body>` grâce à l'élément `document`, puis `body`, puis la propriété `innerHTML` qui permet d'afficher l'arborescence du DOM à partir de l'élément souhaité (ici, `body`).

**Qu'est-ce qu'un nœud ?**

Un **nœud** (ou **node**) est tout simplement un élément du DOM : un texte, une balise HTML, ou un attribut par exemple. Nous pourrions grâce à des méthodes de l'API accéder aux nœuds, les modifier, créer des nœuds frères, parents, enfants...



- Nœud frère : il se situe au même niveau dans l'arborescence.  
`<div><p class="frere"></p><p class="frere"></p></div>`
- Nœud parent : il se situe un niveau au-dessus dans l'arborescence.  
`<div id="parent-de-p"><p></p></div>`
- Nœud enfant : il se situe un niveau en dessous dans l'arborescence.  
`<div><p id="enfant-de-div"></p></div>`

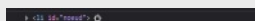
**Exemple Exemple d'accès à un nœud**

```
1 <body>
2   <div>
3     <p>J'appartiens au DOM</p>
4     <ul>
5       <li>Document</li>
6       <li id="noeud">Object</li>
7       <li>Model</li>
8     </ul>
9   </div>
10 </body>
```

Pour sélectionner le nœud portant l'attribut `id="noeud"`, nous utilisons le sélecteur `getElementById()`, que nous verrons en détails dans le prochain chapitre.

```
1 let noeud = document.getElementById('noeud');
2
3 console.log(noeud) // Affiche la balise <li id="noeud">
4 console.log(noeud.parentNode) // Affiche le parent de <li> à savoir <ul>
```

Affichons dans la console notre nœud et son nœud parent.



## Méthode Explication de innerHTML, innerText et textContent

### Qu'est-ce que innerHTML ?

La propriété innerHTML vous permet d'obtenir ou de définir le balisage HTML contenu dans une balise HTML.

Exemple :

```
1 <code>
2 document.getElementById("texte").innerHTML = "<span>Le Paragraphe a changé !</span>";
3 </code>
```

### Qu'est-ce que innerText ?

innerText définit ou renvoie le contenu textuel du nœud spécifié et de tous les nœuds enfants.

Exemple :

```
1 console.log(document.getElementById("monBouton").innerText);
```

### Qu'est-ce que textContent ?

La propriété textContent définit ou renvoie le contenu textuel du nœud spécifié, et de tous ses enfants. Si vous définissez la propriété textContent, tous les nœuds enfants seront supprimés et remplacés par un seul nœud Text contenant la chaîne spécifiée.

Exemple :

```
1 document.getElementById("monBouton").textContent = "Envoyer";;
```

## Syntaxe À retenir

- L'accès aux éléments du DOM est rendu possible grâce à l'élément document. Il s'agit de notre point d'entrée principal pour toutes les manipulations que nous souhaiterons faire sur notre page web : document.body, document.getElementById('id').

## Complément

Définition officielle du DOM<sup>1</sup>

W3C<sup>2</sup>

## III. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



1 [https://developer.mozilla.org/fr/docs/Web/API/Document\\_Object\\_Model/Introduction](https://developer.mozilla.org/fr/docs/Web/API/Document_Object_Model/Introduction)

2 <https://www.w3c.fr/>

3 <https://repl.it/>

## Question

[solution n°1 p.29]

À partir de la page HTML ci-dessous, créez un script JavaScript qui :

- Affiche en console le contenu HTML du body,
- Affiche en console le parent du paragraphe dont l'ID est dom.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width">
6     <title>Document web</title>
7     <link href="style.css" rel="stylesheet" type="text/css" />
8   </head>
9   <body>
10    <h3>Ma page</h3>
11    <p>Bienvenue sur ma page</p>
12    <div id="dom">
13      Fin de la page
14    </div>
15    <script src="script.js"></script>
16  </body>
17 </html>
```

### Indice :

`innerHTML` permet d'obtenir le code HTML d'un élément.

## IV. Les sélecteurs

### Objectifs

- Connaître les sélecteurs disponibles
- Sélectionner des éléments du DOM

### Mise en situation

Comme nous l'avons évoqué précédemment, pour manipuler le DOM, nous aurons besoin de sélectionner ses éléments. Nous pourrions faire appel à des méthodes de l'interface Document. Nous allons voir ici l'ensemble de ces méthodes et leurs subtilités.

#### **document.getElementsByTagName('p')**

Cette méthode renvoie la liste des éléments du DOM portant le nom passé en paramètre. Nous l'utiliserons pour sélectionner directement des balises HTML.

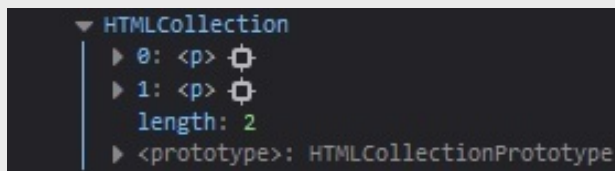
#### **element.item(0).name**

La méthode `item()` renvoie un nœud spécifié dans un objet `NodeList`. Les nœuds sont triés comme ils apparaissent dans le code source, et l'index commence à 0. C'est-à-dire que cette méthode renvoie la valeur de l'attribut `name` dans la balise `input`.

### Exemple

```
1 <body>
2   <p>paragraphe1</p>
3   <p>paragraphe2</p>
4   <span>span1</span>
5   <span>span2</span>
6 </body>

1 let p = document.getElementsByTagName('p')
2 console.log(p)
```



Nous obtenons une `HTMLCollection`, c'est-à-dire un tableau des éléments `<p></p>` présents dans le DOM.

### `document.getElementById('element')`

Cette méthode renvoie l'élément du DOM qui possède l'ID `#element`. Nous l'utiliserons pour sélectionner un élément bien précis.

### Exemple

```
1 <body>
2   <p id="element">paragraphe1</p>
3   <p>paragraphe2</p>
4 </body>

1 let element = document.getElementById('element')
2 console.log(element.innerText)
```

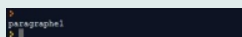


L'élément sélectionné est bien la balise `p` portant l'ID `#element`.

### Fondamental

Un **identifiant** doit être **unique** dans une page web. Si plusieurs éléments possèdent le même ID, la méthode retournera le premier élément trouvé.

```
1 <p id="element">paragraphe1</p>
2 <p id="element">paragraphe2</p>
```



L'élément sélectionné est le premier élément portant l'ID `#element` trouvé.

### `document.getElementsByName('element')`

Cette méthode renvoie la liste des éléments du DOM portant l'attribut `name` passé en paramètre. Il nous sera particulièrement utile pour sélectionner des éléments de formulaire.



**Exemple**

```
1 <body>
2   <form>
3     <input type="lastname">
4   </form>
5 </body>

1 let element = document.getElementsByName('lastname')
2 console.log(element.item(0).name)
```



Nous obtenons une `HTMLCollection` comportant les éléments ayant l'attribut `name="lastname"`.

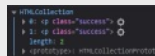
**document.getElementsByClassName('success')**

Cette méthode renvoie la liste des éléments du DOM ayant pour attribut `class="success"`. Nous l'utiliserons pour sélectionner un ensemble d'éléments qui ne sont pas forcément les mêmes balises.

**Exemple**

```
1 <body>
2   <p class="success">success</p>
3   <p class="success">success</p>
4   <p>warning</p>
5   <p class="error">error</p>
6 </body>

1 let success = document.getElementsByClassName('success')
2 console.log(success)
```



Nous obtenons une `HTMLCollection` avec les deux éléments possédant la classe `success`.

**document.querySelector('p.color')**

Cette méthode renvoie le premier élément du DOM correspondant au sélecteur CSS `p.color`.

**Exemple**

```
1 <p class="success">success</p>
2 <p class="color">success</p>
3 <p>warning</p>

1 let success = document.querySelector('p.color')
2 console.log(success)
```



Nous obtenons l'élément `p` ayant la classe `color`.

Il est également possible d'utiliser le sélecteur `document.querySelectorAll()` : ce dernier va retourner cette fois-ci tous les éléments qui correspondent au sélecteur CSS `p.color`.

### Remarque

Les sélecteurs peuvent être enchaînés. Dans un souci de performance, cette technique nous permettra de préciser notre recherche dans le DOM et ainsi d'éviter les recherches trop longues.

### Attention

En termes de performances, nous préférons l'utilisation des sélecteurs `querySelector` et `getElementById`, car, une fois l'élément trouvé, la recherche dans le DOM sera interrompue. En comparaison, `getElementsByClassName` ou `getElementsByTagName` devront parcourir l'ensemble du DOM afin de trouver tous les éléments correspondants. Toutefois, il est préférable d'utiliser `querySelector` car il permet de retourner le premier élément qui correspond à un ou plusieurs sélecteurs CSS spécifiés dans la page HTML. Il s'agit du sélecteur le plus utilisé en langage JavaScript. Mais pour renvoyer toutes les correspondances, il faut plutôt utiliser la méthode `querySelectorAll()`.

D'une manière générale, la manipulation du DOM est lente et risque de ralentir significativement votre application.

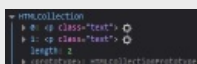
Nous éviterons par exemple les recherches trop génériques sur des éléments qui reviennent souvent, tels que :

- `document.getElementsByTagName('div')`
- `document.getElementsByClassName('maClass')`

### Exemple

```
1 <body>
2   <div id="p1">
3     <p class="text"></p>
4     <p class="text"></p>
5   </div>
6   <div id="p2">
7     <p class="text"></p>
8     <p class="text"></p>
9   </div>
10 </body>

1 let text = document.getElementById('p1').getElementsByClassName('text')
2 console.log(text)
```



Ici, l'enchaînement des sélecteurs `getElementById` et `getElementsByClassName` nous permet de rechercher uniquement les éléments ayant la classe `text` dans l'élément ayant l'ID `#p1`. La recherche dans l'élément `p2` n'a pas lieu.

### Interagir avec le CSS avec `.style`

En JavaScript, vous pouvez parfois vouloir récupérer les styles CSS appliqués à un élément par le biais des feuilles de style. Il existe plusieurs façons de le faire, selon que vous souhaitez récupérer les styles en ligne ou les styles rendus. La propriété `DOM style` est utilisée pour appliquer le CSS à un élément HTML.

**Syntaxe**    **À retenir**

L'utilisation d'un sélecteur se fait à partir de l'élément `document`. Il possède des méthodes que nous pouvons invoquer selon nos besoins :

- `document.getElementById` pour sélectionner un élément par son `id`.
- `document.getElementsByClassName` pour sélectionner des éléments par leur attribut `class`.
- `document.getElementsByName` pour sélectionner des éléments par leur attribut `name`.
- `document.getElementsByTagName` pour sélectionner des éléments par leur nom de balise HTML.
- `document.querySelector` pour sélectionner des éléments par les sélecteurs CSS.

**Complément**

HTMLCollection<sup>1</sup>

## V. Exercice : Appliquez la notion

Vous disposez du code HTML suivant :

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width">
6     <title>Application sélecteurs</title>
7     <link href="style.css" rel="stylesheet" type="text/css" />
8   </head>
9   <body>
10    <p id="titre">Titre de la page</p>
11    <div id="contenu">
12      <span>contenu 1</span>
13      <span>contenu 2</span>
14      <p class="info">info 1</p>
15    </div>
16    <form>
17      <input type="text" name="email">
18    </form>
19    <p class="info">info 2</p>
20    <script src="script.js"></script>
21  </body>
22 </html>
```

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



<sup>1</sup> <https://developer.mozilla.org/fr/docs/Web/API/HTMLCollection>

<sup>2</sup> <https://repl.it/>

## Question

[solution n°2 p.29]

Sélectionnez ces éléments grâce aux méthodes que vous venez de découvrir :

- Le paragraphe ayant pour ID #titre
- Tous les éléments ayant la classe .info
- Le champ d'un formulaire ayant pour attribut name email
- Les balises span contenues dans la div ayant l'ID #contenu
- Les éléments ayant pour sélecteur CSS div#contenu span

## VI. Accéder aux attributs

### Objectifs

- Vérifier qu'un élément possède des attributs ou un attribut en particulier
- Accéder aux attributs d'un élément

### Mise en situation

Pour aller plus loin dans le parcours du DOM, nous allons voir que l'API DOM nous met à disposition des propriétés et méthodes pour accéder aux attributs des éléments sélectionnés. Cela nous permettra, par exemple, de filtrer le résultat de notre recherche.

### Element.attributes

Cette propriété nous permet d'accéder à une liste d'attributs déclarés dans le nœud spécifié. Nous l'utiliserons pour parser l'ensemble des attributs d'une balise HTML.

#### Exemple

```
1 <form>
2   <input type="text" id="lastname" name="lastname">
3 </form>

1 let lastname = document.getElementById('lastname')
2
3 console.log(lastname.attributes)
```

```
type="text"
id="lastname"
name="lastname"
```

`attributes` nous permet d'obtenir tous les attributs du champ du formulaire. Grâce à une boucle, nous pouvons accéder à chacun d'entre eux.

### Element.getAttribute('name')

Cette méthode nous renvoie la valeur de l'attribut passé en paramètre, s'il existe dans le DOM. Nous l'utiliserons pour accéder à la valeur d'un attribut en particulier d'un élément.

**Exemple**

```
1 <form>
2   <input type="text" id="lastname" name="lastname">
3 </form>
```

```
1 let lastname = document.getElementById('lastname')
2 let attributName = lastname.getAttribute('name')
3 console.log(`Valeur de l'attribut name = ${attributName}`) // Affiche la valeur de l'attribut
  name soit lastname
```



`getAttribute` nous permettra de filtrer les résultats d'une recherche dans le DOM. Par exemple, nous pourrions sélectionner toutes les balises `input` d'un formulaire, sauf celles qui possèdent l'attribut `class="info"`.

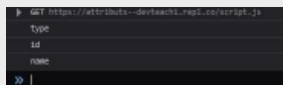
**Element.getAttributeNames()**

Cette méthode nous renvoie une liste des noms d'attributs d'un élément du DOM.

**Exemple**

```
1 <form>
2   <input type="text" id="lastname" name="lastname">
3 </form>
```

```
1 let lastname = document.getElementById('lastname')
2
3 console.log(lastname.getAttributeNames())
```



`getAttributeNames()` nous permettra de vérifier qu'un élément possède bien un attribut, par exemple qu'une balise `input` possède bien l'attribut `type`.

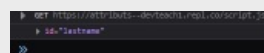
**Element.getAttributeNode('id')**

Cette méthode nous renvoie le nœud d'attribut spécifié en paramètre. Nous l'utiliserons pour accéder à un élément attribut en particulier dans le DOM.

**Exemple**

```
1 <form>
2   <input type="text" id="lastname" name="lastname">
3 </form>
```

```
1 let lastname = document.getElementById('lastname')
2 console.log(lastname.getAttributeNode('id'))
```



Nous obtenons le nœud attribut `id` correspondant à la balise `input` avec l'ID `#lastname`.

## Element.hasAttribute('id')

Cette méthode nous renvoie une valeur booléenne (vrai ou faux) si l'élément possède le nom d'attribut passé en paramètre. Nous l'utiliserons pour filtrer notre résultat de recherche dans le DOM ou bien pour associer des scripts à certains éléments et d'autres non.

### Exemple

```
1 <ul>
2   <li class="item">item 1</li>
3   <li class="item">item 2</li>
4   <li>item 3</li>
5   <li class="item">item 4</li>
6 </ul>

1 let elements = document.getElementsByTagName('li')
2
3 for(let element of elements) {
4   if (element.hasAttribute('class')) {
5     console.log(element.innerText)
6   }
7 }
```

```
0 GET http://localhost:3000/.../script.js
item 1
item 2
item 4
```

Nous obtenons tous les éléments correspondant aux balises `li`, sauf celle qui n'a pas l'attribut `class`.

## Element.hasAttributes()

Cette méthode nous renvoie une valeur booléenne (vrai ou faux) si l'élément possède au moins un attribut ou aucun. Là aussi, nous pourrions l'utiliser pour filtrer nos résultats de recherche dans le DOM et pour associer ou non des scripts à certains éléments.

### Exemple

```
1 <ul>
2   <li id="item">item 1</li>
3   <li>item 2</li>
4 </ul>

1 let elements = document.getElementsByTagName('ul')
2
3 for(let element of elements) {
4   console.log(element.hasAttributes())
5 }
```

```
0 GET http://localhost:3000/.../script.js
false
```

La balise `ul` ne possède aucun attribut : `hasAttributes` nous renvoie donc la valeur `false`.

### Complément

L'un des intérêts de pouvoir parcourir et sélectionner les attributs du DOM est d'en ajouter, de les modifier ou les supprimer. Pour ce faire, nous pourrions utiliser deux méthodes de l'API : `setAttribute('name', 'valeur')` et `removeAttribute('name')`.

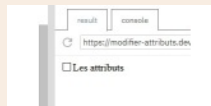
`setAttribute` nous permettra d'ajouter un nouvel attribut ou de modifier un attribut existant en précisant en paramètres le nom de l'attribut, puis sa valeur.

```
1 <input type="checkbox" name="box" id="box">Les attributs
1 let checkbox = document.getElementById('box')
2 checkbox.setAttribute('checked', 'checked')
```



`removeAttribute` nous permettra de supprimer un attribut en précisant en paramètre son nom.

```
1 <input type="checkbox" name="box" id="box" checked="checked">Les attributs
1 let checkbox = document.getElementById('box')
2 checkbox.removeAttribute('checked')
```



### Syntaxe À retenir

L'accès aux attributs se fait à partir d'un élément sélectionné dans le DOM :

- `element.attributes` : propriété permettant d'accéder à la liste d'attributs d'un nœud.
- `element.getAttribute('name')` : méthode permettant d'obtenir la valeur de l'attribut donné pour un nœud.
- `element.getAttributeNames()` : méthode permettant d'obtenir la liste des noms d'attributs d'un nœud.
- `element.getAttributeNode('id')` : méthode permettant d'obtenir un nœud d'attribut donné.
- `element.hasAttribute('id')` : méthode permettant de vérifier si un nœud possède l'attribut donné.

## VII. Exercice : Appliquez la notion

Vous disposez du formulaire HTML suivant :

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width">
6     <title>repl.it</title>
7     <link href="style.css" rel="stylesheet" type="text/css" />
8   </head>
9   <body>
10    <form>
11      <input type="text" name="email" id="email" class="form-email">
12      <input type="checkbox" name="connexion" id="connexion">
13      <label for="connexion">Rester connecté</label>
14      <button>Valider</button>
15    </form>
16    <script src="script.js"></script>
```

```
17 </body>
18 </html>
```

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



## Question

[solution n°3 p.29]

Réalisez les opérations suivantes :

- Affichez l'ensemble des attributs du champ texte
- Affichez la valeur de l'attribut id du champ texte
- Vérifiez que la case à cocher possède l'attribut `checked`. Décochez la case si elle est cochée.
- Affichez le ou les éléments n'ayant aucun attribut.

## VIII. Modifier le DOM

### Objectif

- Modifier une page web

### Mise en situation

Nous savons dorénavant parcourir le DOM et sélectionner des éléments. Il peut être intéressant de connaître les méthodes qui vont nous permettre de le modifier. Concrètement, nous pourrions réaliser une page web entièrement en JavaScript grâce à l'API DOM, mais nous verrons qu'il est préférable de se concentrer sur des choses utiles afin d'éviter des lenteurs à notre application.

### `document.createElement('div')`

Cette méthode nous permet de créer l'élément HTML spécifié en paramètre.

#### Attention

Elle n'ajoute pas l'élément dans le DOM : nous l'utiliserons avec l'une des méthodes que nous verrons ci-dessous.

#### Exemple

```
1 <ul id="liste">
2   <li>item 1</li>
3   <li>item 2</li>
4 </ul>

1 let liste = document.getElementById('liste')
2 let li = document.createElement('li')
```

Nous avons créé ici un troisième élément `li` prêt à être ajouté à notre page web.



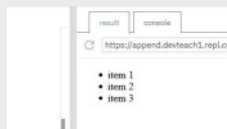
## Element.append()

Cette méthode nous permet d'ajouter des nœuds **après le dernier enfant** de l'élément sur lequel elle est invoquée. Nous l'utiliserons pour ajouter des éléments à notre page après d'autres éléments.

### Exemple

```
1 <ul id="liste">
2   <li>item 1</li>
3   <li>item 2</li>
4 </ul>

1 let liste = document.getElementById('liste')
2
3 // Création de l'élément li
4 let li = document.createElement('li')
5
6 // Ajout d'un texte à notre item de liste
7 li.innerText = 'item 3'
8
9 // Ajout de l'item à la fin de la liste
10 liste.append(li)
```



La méthode `append()` permet d'ajouter des éléments HTML, mais également directement du texte. Ainsi, nous aurions pu remplacer `li.innerText = 'item3'` par `li.append('item 3')`.

### Complément

Il existe une méthode `appendChild()` permettant également d'ajouter après un autre élément de la page. Contrairement à `append()`, elle ne permet pas d'ajouter directement du texte et ne permet pas l'ajout de plusieurs éléments.

## Element.prepend('div')

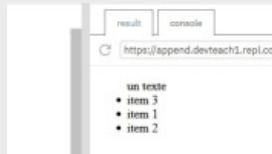
Cette méthode nous permet d'ajouter des nœuds **avant le premier enfant** de l'élément sur lequel elle est invoquée. Nous l'utiliserons pour ajouter des éléments à notre page avant d'autres éléments.

### Exemple

```
1 <ul id="liste">
2   <li>item 1</li>
3   <li>item 2</li>
4 </ul>

1 let liste = document.getElementById('liste')
2
3 // Création de l'élément li
4 let li = document.createElement('li')
5 li.innerText = 'item 3'
6
7 // Ajout de l'item au début de la liste
8 liste.prepend(li)
9
```

```
10 // Ajout d'un texte avant notre liste
11 liste.prepend('un texte')
```



À l'instar de `append()`, nous pouvons ajouter des éléments HTML et du texte à notre page web.

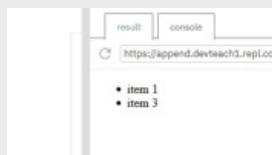
## Element.remove()

Cette méthode nous permet tout simplement de supprimer un nœud du DOM.

### Exemple

```
1 <ul id="liste">
2   <li id="item1">item 1</li>
3   <li id="item2">item 2</li>
4   <li id="item3">item 3</li>
5 </ul>

1 let item2 = document.getElementById('item2')
2 item2.remove()
```



Nous avons supprimé l'élément portant l'ID `#item2` de la liste.

### Complément

Nous pourrions avoir envie de ne pas supprimer définitivement un élément du DOM : soit pour le remettre ensuite, soit pour le déplacer. La méthode `removeChild('child')` pourra alors nous être utile. Elle va supprimer le nœud enfant spécifié et nous le retourner. Nous pourrions donc le stocker pour l'utiliser de nouveau plus tard.

### Attention

En termes de performance, la modification du DOM peut être coûteuse. Elle impose un parcours en profondeur et le déplacement d'objets par rapport à d'autres, ce qui peut être lourd en fonction de la taille de notre application. D'une manière générale, il faut éviter :

- l'utilisation de l'API DOM pour écrire du CSS
- l'écriture complète d'une page en JavaScript
- la création d'une structure trop profonde

**Syntaxe**    **À retenir**

Après avoir sélectionné un élément dans le DOM, nous pouvons le modifier en utilisant plusieurs méthodes de l'API DOM :

- `createElement()` qui permet de créer un élément HTML.
- `append()` qui permet d'ajouter un élément HTML ou texte après un autre élément du DOM.
- `prepend()` qui permet d'ajouter un élément HTML ou texte avant un autre élément du DOM.
- `remove()` qui permet de supprimer un élément du DOM.

**Complément**

`appendChild()`<sup>1</sup>

`removeChild()`<sup>2</sup>

## IX. Exercice : Appliquez la notion

Vous disposez de la page suivante :

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width">
6     <title>repl.it</title>
7     <link href="style.css" rel="stylesheet" type="text/css" />
8   </head>
9   <body>
10    <div id="contenu">
11      <p>1er paragraphe</p>
12      <p>2nd paragraphe</p>
13      <ul>
14        <li>item 1</li>
15        <li>item 2</li>
16        <li>item 3</li>
17      </ul>
18    </div>
19    <script src="script.js"></script>
20  </body>
21 </html>
```

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



1 <https://developer.mozilla.org/fr/docs/Web/API/Node/appendChild>

2 <https://developer.mozilla.org/fr/docs/Web/API/Node/removeChild>

3 <https://repl.it/>

## Question

[solution n°4 p.30]

Modifiez cette page grâce à l'API DOM :

- Ajoutez un titre `h3` au-dessus de la `div` principale
- Supprimez les items de la liste
- Inversez la position des 2 paragraphes

### Indice :

La propriété `firstElementChild` permet de sélectionner le premier enfant d'un élément.

## X. Auto-évaluation

### A. Exercice final

#### Exercice 1

[solution n°5 p.30]

Exercice

Que veut dire API ?

Exercice

DOM est l'acronyme de *Document Object Manager*.

- ☐ Vrai
- ☐ Faux

Exercice

Le DOM est généralement représenté sous la forme...

- ☐ D'un arbre
- ☐ D'une bête à cornes
- ☐ D'un diagramme

Exercice

Quels sélecteurs existent vraiment ?

- ☐ `selectById('id')`
- ☐ `getElementById('id')`
- ☐ `getElementsByTagName('tag')`
- ☐ `getByClassName('class')`

Exercice

Indiquez la ou les affirmation(s) incorrecte(s).

- ☐ `element.attributes` : méthode permettant d'accéder à la liste d'attributs d'un nœud
- ☐ `element.getAttributeNames()` : méthode permettant d'obtenir la liste des noms d'attributs d'un nœud
- ☐ `element.hasAttribute('id')` : méthode permettant de vérifier si un nœud possède l'attribut donné

## Exercice

`append()` permet d'ajouter un ou plusieurs éléments...

- ☐ Après un élément du DOM
- ☐ Avant un élément du DOM

## Exercice

Indiquez l'instruction permettant de sélectionner les `div` ayant la classe `color` dans le DOM.

- ☐ `document.querySelector('div.color')`
- ☐ `document.querySelectorAll('div.color')`

## Exercice

Soit `<div id="content"></div>`. Modifiez l'attribut `id` pour lui donner la valeur `header`.

- ☐ `document.getElementById('content').setAttribute('id', 'footer')`
- ☐ `document.getElementById('content').setAttribute('id', 'header')`
- ☐ `document.getElementById('content').attribute('id', 'header')`
- ☐ `document.getElementById('content').ChangeAttribute('id', 'header')`

## Exercice

Parmi ces déclarations, lesquelles permettent de créer un élément HTML et de l'ajouter dans le DOM ?

- ☐ `let div = document.createElement('div')`
- ☐ `let div = document.createElement('div')`  
`document.body.append(div)`
- ☐ `let parent = document.getElementById('parent')`  
`let child = document.createElement('div')`  
`parent.prepend(child)`
- ☐ `let parent = document.getElementById('parent')`  
`parent.remove()`

## Exercice

En écrivant `element.removeChild(child)`...

- ☐ Nous supprimons `element` du DOM
- ☐ Nous supprimons `child`, enfant de `element` dans le DOM
- ☐ Nous supprimons `child`, parent de `element` dans le DOM
- ☐ Nous pouvons garder en mémoire `child`

## B. Exercice : Défi

On souhaite pouvoir tester différentes couleurs sur un site web. Afin d'éviter de changer le CSS pour chaque couleur, on vous demande d'ajouter une fonctionnalité : la possibilité de changer la couleur des éléments du site en un clic. Pour cela, un écouteur permettant de choisir une couleur a été implémenté sur le badge. Lorsque l'on clique dessus, un `colorpicker` apparaît.

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



### Question

[solution n°6 p.32]

Pour vous aider, un écouteur a été placé sur l'élément badge et la méthode a déjà été implémentée dans le fichier JavaScript.

Une fonction `changeColor()` a aussi été créée. À chaque fois que vous changez la couleur, cette fonction est appelée et la nouvelle couleur est stockée dans la variable `color`.

À partir du code HTML et JavaScript ci-dessous, ajoutez à la fonction `changeColor()` le code qui permette de changer la couleur des éléments suivants :

- Titre,
- Background de la navigation et de la liste,
- Bordure des articles,
- Texte du lien actif,
- Background du badge.

Pour identifier les éléments sur lesquels sont attribuées les couleurs, aidez-vous du fichier CSS.

Pour modifier la couleur du texte des liens actifs, il faut se baser sur la classe active. Cependant, l'élément à modifier est le premier enfant de l'élément portant cette classe.

Pour réaliser cet exercice, vous n'avez pas besoin de modifier le code HTML et CSS.

```
1 /* elements */
2 ul {
3   padding: 0;
4   margin: 0;
5 }
6
7 p {
8   margin: 0;
9 }
10
11 /* header */
12 header {
13   padding-top: 10px;
14   display: flex;
15 }
16
17 header .logo {
18   border-radius: 10px;
19 }
20
21 h1 {
```

1 <https://repl.it/>

```
22 margin: 0;
23 color: #007BFF;
24 }
25
26 header .logo img {
27 width: 100%;
28 height: 100%;
29 }
30
31 #colorpicker {
32   visibility: hidden;
33 }
34
35 header .badge {
36 height: 100%;
37 display: flex;
38 flex-direction: column;
39 justify-content: center;
40 margin-left: calc(100vw - 600px);
41 }
42
43 header .badge .circle {
44 width: 70px;
45 height: 70px;
46 border-radius: 50%;
47 background-color: #007BFF;
48 display: flex;
49 flex-direction: column;
50 justify-content: center;
51 text-align: center;
52 font-size: 2.5em;
53 color: white;
54 }
55
56 /* nav */
57
58 nav {
59 margin-top: 10px;
60 }
61
62 nav ul {
63 padding: 0;
64 display: flex;
65 background: #007BFF;
66 height: 100%;
67 }
68
69 nav ul li {
70 display: flex;
71 flex-direction: column;
72 justify-content: center;
73 border-right: 2px white solid;
74 list-style-type: none;
75 padding-left: 10px;
76 padding-right: 10px;
77 }
78
79 nav ul li a {
```

```
80 color: white;
81 text-decoration: none;
82 }
83
84 nav ul .active {
85 background-color: white;
86 }
87
88 nav ul .active a {
89 color: #007BFF;
90 }
91
92
93 nav ul li:nth-child(1) {
94 border-left: 2px white solid;
95 }
96 }
97
98 /* section */
99 section {
100 margin-top: 10px;
101 display: flex;
102 justify-content: space-between;
103 }
104
105 /* aside */
106
107 aside {
108 width: 16.66%
109 }
110
111 aside ul {
112 width: 100%;
113 height: 100%;
114 background-color: #007BFF;
115 display: flex;
116 flex-direction: column;
117 }
118
119 aside ul li {
120 width: 100%;
121 display: flex;
122 flex-direction: column;
123 justify-content: center;
124 border-bottom: 2px white solid;
125 list-style-type: none;
126 text-align: center;
127 }
128
129 aside ul li a {
130 color: white;
131 text-decoration: none;
132 }
133
134 aside ul .active {
135 background-color: white;
136 }
137
```



```

138 aside ul .active a {
139 color: #007BFF;
140 }
141
142 /* contain-articles */
143 section .contain-articles {
144 width: calc(100vw - 25%);
145 }
146
147 section .contain-articles .small-articles{
148 width: 100%;
149 display: flex;
150 justify-content: space-between;
151 }
152
153 section .contain-articles article{
154 width: 25%;
155 padding: 10px;
156 border: 1px #007BFF solid;
157 text-align: justify;
158 display: flex;
159 flex-direction: column;
160 justify-content: space-between;
161 }
162
163 section .contain-articles .article-large {
164 margin-top: 10px;
165 width: 100%;
166 padding: 10px;
167 border: 1px #007BFF solid;
168 text-align: justify;
169 }
170
171 .shop {
172 padding: 10px;
173 display: flex;
174 justify-content: flex-end;
175 }
176
177

```

```

1 <!doctype html>
2 <html lang="fr">
3   <head>
4     <meta charset="utf-8">
5     <link rel="stylesheet" href="style.css">
6     <title>Exemple</title>
7
8   </head>
9   <body>
10    <header style="height: 100px;">
11      <div style="height: 100px;width: 100px" class="logo">
12        
13      </div>
14      <div style="width: 400px;">
15        <h1>Actu Sport</h1>
16      </div>
17      <div class="badge">

```

```

18     <label for="colorpicker" class="circle">NL</label>
19     <input id="colorpicker" type="color" value="#007BFF">
20 </div>
21 </header>
22 <nav style="height:50px">
23     <ul>
24         <li><a href="#">Accueil</a></li>
25         <li class="active"><a href="#">Football</a></li>
26         <li><a href="#">Rugby</a></li>
27         <li><a href="#">Handball</a></li>
28         <li><a href="#">basket</a></li>
29     </ul>
30 </nav>
31 <section>
32     <aside>
33         <ul>
34             <li style="height: 50px;"><a href="#">France</a></li>
35             <li style="height: 50px;" class="active"><a href="#">Angleterre</a></li>
36             <li style="height: 50px;"><a href="#">Allemagne</a></li>
37             <li style="height: 50px;"><a href="#">Espagne</a></li>
38             <li style="height: 50px;"><a href="#">Italie</a></li>
39         </ul>
40     </aside>
41     <div class="contain-articles">
42         <div class="small-articles">
43             <article>
44                 <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
45                 Praesent eu odio non tortor laoreet ultrices non nec urna. Aenean eget erat
46 pellentesque, euismod nisl ac, convallis lacus.
47                 Mauris ac est ante. Maecenas nec mauris vehicula, lacinia ipsum at, eleifend nibh.
48 Vivamus pharetra,
49                 ex ac bibendum hendrerit, odio ex venenatis mauris, quis tempus ante leo id leo.
50                 Curabitur molestie massa eget lectus dignissim,
51                 sit amet malesuada tellus dapibus. Nulla facilisi. Etiam quis pretium purus. Nullam
52                 vehicula euismod nisl, vel egestas magna euismod ac.
53                 Mauris porttitor eget quam sed viverra. Nullam sollicitudin, odio ut varius
54                 interdu, quam arcu rutrum ligula, vitae suscipit
55                 lacus elit at ligula. Quisque maximus vitae magna sed pulvinar. Quisque non tortor
56                 convallis, hendrerit quam ac, sodales turpis.
57                 Ut tempus imperdiet ligula lobortis pharetra. Nullam id purus vel odio eleifend
58                 hendrerit.
59                 </p>
60             </article>
61             <article>
62                 <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
63                 Praesent eu odio non tortor laoreet ultrices non nec urna. Aenean eget erat
64                 pellentesque, euismod nisl ac, convallis lacus.
65                 Mauris ac est ante. Maecenas nec mauris vehicula, lacinia ipsum at, eleifend nibh.
66 Vivamus pharetra,
67                 ex ac bibendum hendrerit, odio ex venenatis mauris, quis tempus ante leo id leo.
68                 Curabitur molestie massa eget lectus dignissim,
69                 sit amet malesuada tellus dapibus. Nulla facilisi. Etiam quis pretium purus.
70                 Nullam vehicula euismod nisl, vel egestas magna euismod ac.
71                 </p>
72             </article>
73             <article>
74                 <p>Pellentesque mollis iaculis pulvinar. Quisque justo urna, consectetur ut nibh a,
75                 auctor dictum lacus.
76                 Nullam vestibulum efficitur neque. Morbi dictum vulputate nulla sit amet
77                 venenatis. Curabitur accumsan laoreet ullamcorper.
78                 Nulla facilisi. Donec porta id tellus vitae accumsan. Etiam sem diam, fringilla
79                 nec dui sit amet, malesuada laoreet urna.

```

```

67     </p>
68   </article>
69 </div>
70 <article class="article-large">
71   <p>Sed ullamcorper dolor tincidunt, tincidunt eros ut, consectetur neque. Aliquam erat
volutpat. Aliquam dictum ullamcorper eros at lobortis. In elementum, leo eget finibus
sollicitudin, lectus lacus
73   feugiat dui, id rutrum dui lectus id eros. Fusce magna dui, tristique sit amet
fringilla ut, porttitor nec justo.
74   Phasellus faucibus, purus id semper pharetra, eros augue molestie urna, nec
porttitor ipsum lectus at augue.
75   Quisque fermentum eros mollis accumsan pulvinar. Maecenas lacinia nunc et interdum
tincidunt. Curabitur nec
76   quam vel urna sagittis euismod vitae sit amet sapien. Etiam a tempor leo, non
tincidunt eros. Fusce ut felis eu
77   ante cursus aliquam at ut odio. Morbi id fermentum augue. Integer mollis
scelerisque cursus. Sed ut sollicitudin odio,
78   blandit euismod urna. Curabitur vel dui non purus volutpat elementum sed vitae
lectus. Pellentesque gravida nibh
79   eleifend sem pulvinar viverra.</p>
80 </article>
81 </div>
82 </section>
83 <script src="script.js"></script>
84 </body>
85
86 </html>
87
1 const colorpicker = document.getElementById('colorpicker');
2
3 changeColor = (ev) => {
4   const color = ev.target.value;
5   console.log(color)
6   // votre code ici
7 }
8
9 colorpicker.addEventListener('change', changeColor);

```

## Solutions des exercices



**p. 7 Solution n°1**

```
1 let body = document.body;
2 let dom = document.getElementById('dom');
3
4 console.log(body.innerHTML);
5 console.log(dom.parentNode);
```

**p. 12 Solution n°2**

```
1 // Paragraphe avec l'id #titre
2 let p = document.getElementById('titre')
3
4 // Tous les éléments avec la classe .info
5 let infos = document.getElementsByClassName('info')
6
7 // Input email
8 let input = document.getElementsByName('email')
9
10 // Les éléments span de la div avec l'id #contenu
11 let spans = document.getElementById('contenu').getElementsByTagName('span')
12
13 // Les éléments avec le sélecteur CSS div#contenu span
14 let element = document.querySelectorAll('div#contenu span')
15
```

**p. 16 Solution n°3**

```
1 let inputText = document.getElementById('email')
2
3 for(let attribut of inputText.attributes) {
4   console.log(attribut)
5 }
6
7 let inputTextId = inputText.getAttribute('id')
8 console.log(inputTextId)
9
10 let checkbox = document.getElementById('connexion')
11
12 if (checkbox.hasAttribute('checked')) {
13   console.log(checkbox.getAttribute('checked'))
14   checkbox.removeAttribute('checked')
15 }
16
17 let form = document.querySelector('form')
18
19 for(element of form.elements) {
20   if (!element.hasAttributes()) {
21     console.log(`Cet élément n'a pas d'attributs : ${element}`);
22   }
23 }
```

p. 20 Solution n°4

```
1 // Ajout du titre h3.
2 let contenu = document.getElementById('contenu')
3
4 let h3 = document.createElement('h3')
5 h3.append('Titre de la section')
6 document.body.prepend(h3)
7
8 // Suppression des items de la liste.
9 let items = document.getElementsByTagName('li')
10
11 for(item of Array.from(items)) {
12     item.remove()
13 }
14
15 // Inversion des paragraphes.
16 let premierParagraphe = contenu.firstElementChild
17 premierParagraphe.remove()
18
19 contenu.append(premierParagraphe)
```

Exercice p. 20 Solution n°5

Exercice

Que veut dire API ?

Interface Programmation Application

Exercice

DOM est l'acronyme de *Document Object Manager*.

☐ Vrai

☒ Faux

*DOM veut dire Document Object Model.*

Exercice

Le DOM est généralement représenté sous la forme...

☒ D'un arbre

☐ D'une bête à cornes

☐ D'un diagramme

Exercice

Quels sélecteurs existent vraiment ?

☐ selectById('id')

☒ getElementById('id')

☒ getElementsByTagName('tag')

☐ getByClassName('class')

**Exercice**

Indiquez la ou les affirmation(s) incorrecte(s).

- ☒ `element.attributes` : méthode permettant d'accéder à la liste d'attributs d'un nœud  
*Il ne s'agit pas d'une méthode, mais d'une propriété.*
- ☐ `element.getAttributeNames()` : méthode permettant d'obtenir la liste des noms d'attributs d'un nœud
- ☐ `element.hasAttribute('id')` : méthode permettant de vérifier si un nœud possède l'attribut donné

**Exercice**

`append()` permet d'ajouter un ou plusieurs éléments...

- ☒ Après un élément du DOM
- ☐ Avant un élément du DOM

**Exercice**

Indiquez l'instruction permettant de sélectionner les `div` ayant la classe `color` dans le DOM.

- ☐ `document.querySelector('div.color')`
- ☒ `document.querySelectorAll('div.color')`

**Exercice**

Soit `<div id="content"></div>`. Modifiez l'attribut `id` pour lui donner la valeur `header`.

- ☐ `document.getElementById('content').setAttribute('id', 'footer')`
- ☒ `document.getElementById('content').setAttribute('id', 'header')`
- ☐ `document.getElementById('content').attribute('id', 'header')`
- ☐ `document.getElementById('content').ChangeAttribute('id', 'header')`

**Exercice**

Parmi ces déclarations, lesquelles permettent de créer un élément HTML et de l'ajouter dans le DOM ?

- ☐ `let div = document.createElement('div')`
- ☒ `let div = document.createElement('div')`  
`document.body.append(div)`
- ☒ `let parent = document.getElementById('parent')`  
`let child = document.createElement('div')`  
`parent.prepend(child)`
- ☐ `let parent = document.getElementById('parent')`  
`parent.remove()`

**Exercice**

En écrivant `element.removeChild(child)`...

- ☐ Nous supprimons `element` du DOM
- ☒ Nous supprimons `child`, enfant de `element` dans le DOM
- ☐ Nous supprimons `child`, parent de `element` dans le DOM
- ☒ Nous pouvons garder en mémoire `child`

**p. 22 Solution n°6**

```

1 const uls = document.getElementsByTagName('ul');
2 const active = document.getElementsByClassName('active');
3 const articles = document.getElementsByTagName('article');
4 const title = document.getElementsByTagName('h1');
5 const badge = document.getElementsByClassName('circle');
6 const colorpicker = document.getElementById('colorpicker');
7
8 changeColor = (ev) => {
9   const color = ev.target.value;
10  for (let i = 0; i < uls.length; i++) {
11    uls.item(i).style.background= color;
12  }
13
14  for (let i = 0; i < badge.length; i++) {
15    badge.item(i).style.background= color;
16  }
17
18  for (let i = 0; i < articles.length; i++) {
19    articles.item(i).style.borderColor = color;
20  }
21
22  for (let i = 0; i < title.length; i++) {
23    title.item(i).style.color = color;
24  }
25
26  for (let i = 0; i < active.length; i++) {
27    active.item(i).firstChild.style.color = color;
28  }
29 }
30
31 colorpicker.addEventListener('change', changeColor);

```