Prérequis

Kit de survie : Apprenez en 10 minutes l'API du DOM.

DOM 1%

kit de survie

💥En action :https://codepen.io/dupontcodepen/pen/qBJrZxr

🆒Sélectionnez tous les paragraphes

```
1. const articles = document.querySelectorAll("p");
```

Comptez le nombre de paragraphes

```
1. const articles = document.querySelectorAll("p");
2. console.log(articles.length)
```

Une boucle sur la sélection des paragraphes

```
1. const articles = document.querySelectorAll("p");
2. for (let article of articles) {
3.         console.log(article.innerText.split(" ").length); // count words
4. }
```

Ajouter une classe à tous les paragraphes

```
1. const articles = document.querySelectorAll("p");
2. for (let article of articles) {
3.     article.classList.add("second");
4. }
```

Ajoutez un événement click (🪒 voir amélioration par délégation)

```
1. const articles = document.querySelectorAll("p");
2. for (let para of articles) {
3.   para.addEventListener("click", () => {
4.     para.classList.add("selected");
5.   });
6. }
```

🥷Somme des valeurs contenu dans l'attribut data-code

```
1. let value = 0;
2. for (let para of articles) {
3.   value+= parseInt(para.dataset.code) // parseInt transforme la valeur
   en entier !
4. }
5. console.log(value)
```

Affichez le résultat dans un div.

```
1. let value = 0;
2. for (let para of articles) {
3.   value+= parseInt(para.dataset.code) // parseInt transforme la valeur
   en entier !
4. }
5. document.querySelector("div").innerText = value
```

🥇La délégation

```
1. const func = function (event) {
2.   const targetElement = event.target; // p or em
3.   if (targetElement.closest("p")) {
4.     let para = targetElement.closest("p");
5.     para.classList.toggle("selected");
6.   }
7. };
8.
9. const parent = document.querySelector(".container");
10. parent.addEventListener("click", func, false);
```

🪛Mise en place du TD

Vous pouvez dans la suite du TD utiliser codepen.io ou de préférence l'éditeur **VScode.**

Nous allons mettre en place le TD avec l'éditeur VScode.

Visual Studio code

Créez un répertoire 📁dom[1] .

Dans un terminal, lancez les deux commandes suivantes pour créer et se déplacer dans le répertoire dom.

C:\Users\DD>**mkdir dom**

C:\Users\DD>**cd dom**

Lancez ensuite votre éditeur VisualStudio.

C:\Users\DD\dom>**code .**[2]

Commençons par quelques rappels sur le fonctionnement de JS et son interaction avec la structure du DOM.

Rappels

Nous allons rappeler que le code sur le DOM doit être lancé après que la structure soit montée en mémoire.

Dans VScode créez un fichier index.html

index.html

[1] code en ligne de commande.
[2] notez le point .

```
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.   <script src="index.js"></script>
8. </head>
9. <body>
10.
11. </body>
12. </html>
```

En lig.7 faites un lien vers un fichier js.

Dans VScode créez un fichier index.js et ajoutez le code suivant

index.js

```
1. document.body.insertAdjacentHTML("afterbegin", `<h1> Module
   </h1>`);
```

Testez le code en lançant un serveur (extension : live-serveur[3]).

😈L'affichage de la console indiquera

index.js:1 Uncaught TypeError: Cannot read properties of null (reading 'appendChild')

at index.js:1:15

[3] Vérifiez que vous avez installé l'extension : liveServeur.

index.html:41 Live reload enabled.

🆘Mais d'où vient notre problème ?

Analyse : Nous aurions pu écrire des informations dans la console, faire des calculs pour montrer que JS répond.

Mais la structure de la page **elle ne répond pas ?**

En fait, la structure n'est pas montée en mémoire. Ainsi l'objet body est introuvable. Sa valeur est *null*.

Ce qui explique le message d'erreur "*Cannot read properties of null*"

Il est donc indispensable avant de manipuler la structure qu'elle soit disponible.

Nous allons donc découvrir l'intérêt du mot clef **defer,** attribut de script.

Correction

🆒L'erreur observée dans le code précédent est classique. Voici deux corrections possibles pour assurer la présence de la structure avant toute manipulation.

Solution 1

On garantit que la structure est montée en mémoire avant l'exécution du code .

Modifiez le code du fichier index.html.

index.html

1. <!DOCTYPE html>

```
2. <html lang="en">
3.
4. <head>
5.   <meta charset="UTF-8">
6.   <meta name="viewport" content="width=device-width,
     initial-scale=1.0">
7.   <title>Document</title>
8. </head>
9. <body>
10.
11.  <script src="index.js"></script>
12. </body>
13. </html>
```

Lig. 11 : On corrige le code en plaçant l'appel au script avant la balise fermante de </body>.

Solution 2 : Defer

Une autre solution consiste à ajouter l'attribut **defer** directement dans la balise <script> placée dans la balise <head>.

Modifiez le code du fichier index.html.

index.html

```
1. <!DOCTYPE html>
2. <html lang="en">
3.
4. <head>
```

```
5.    <meta charset="UTF-8">
6.    <meta name="viewport" content="width=device-width,
      initial-scale=1.0">
7.    <title>Document</title>
8.    <script src="index.js" defer></script>
9.  </head>
10. <body>
11.
12. </body>
13. </html>
```

🪛Testez votre code en lançant le serveur de l'extension live-server.

Maintenant que nous avons monté la structure, revenons aux méthodes de base du DOM qui permettent sa manipulation.

Objectifs

Nous allons apprendre à manipuler les méthodes suivantes :

1. getElementsByTagName
2. querySelectorAll
3. textContent
4. innerHTML

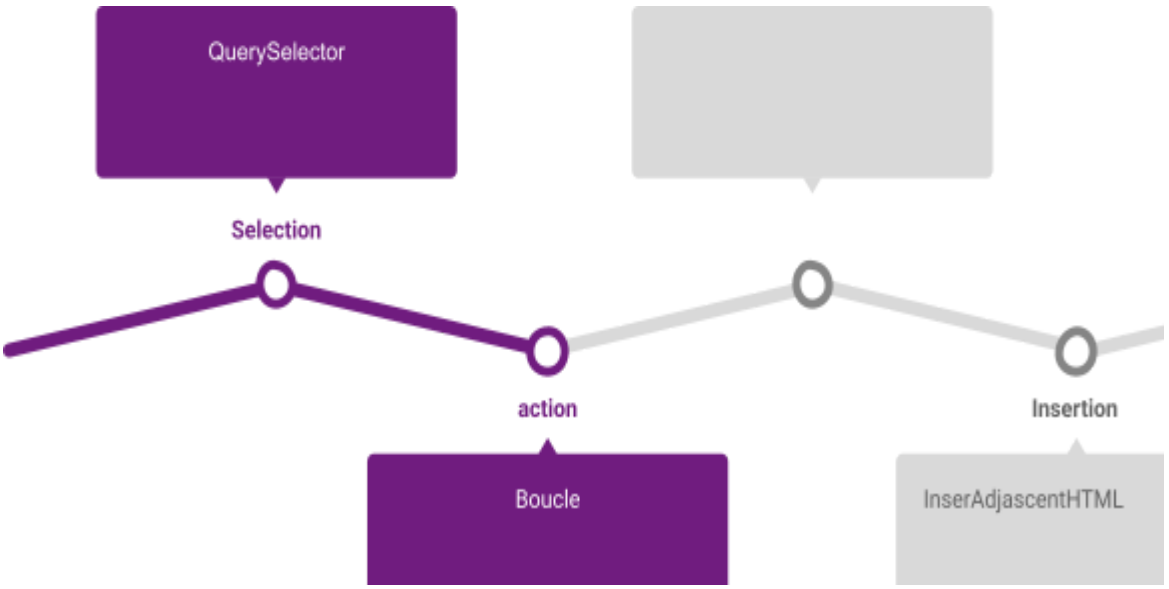
Exercice

🪛Donnez un code pour modifier le contenu des deux paragraphes.

code Avant	Code Après
<!DOCTYPE html> <html> <head> <meta charset="utf-8"> <meta name="viewport" content="width=device-width"> <title>code</title> </head> <body> item 1 item 2 </body> </html>	<!DOCTYPE html> <html> <head> <meta charset="utf-8"> <meta name="viewport" content="width=device-width"> <title>code</title> </head> <body> **Modified HTML file** **Modified HTML file** </body> </html>

🆘 La première réflexion est d'imaginer un algorithme simple pour résoudre le problème.

Généralement, on sélectionne un ensemble d'éléments pour agir dessus !



Ainsi, si l'on veut modifier des paragraphes, il faut :

1️⃣ Sélectionner le(s) paragraphe(s)

2⃣Modifier leur contenu

Compléter le code suivant où :

1⃣Les lig. 1-3 correspondent à la sélection

2⃣Les lignes 7-8 correspondent à la modification.

code

```
1. const paras = document.--?--('p');
2. const p1 = paras[--?--],
3.       p2 = paras[--?--];
4.
5. console.log(` text of  ${paras[0].id} is
   ${p1.textContent} `);
6.
7. p1.--?-- = "Modified HTML file";
8. p2.--?-- = "<i>Modified</i> HTML file";
```

Lig.1 : On recherche tous les paragraphes.

Lig1-3 : On définit deux références p1 et p2 en accédant à la HTMLCollection contenant les éléments <p> trouvés, dans l'ordre dans lequel ils apparaissent dans le sous-arbre.

Lig. 7-8 Vous modifiez le contenu par du contenu en HTML.

Notez bien la différence de résultat à obtenir entre les lig.7 et lig. 8.

🆘Compléter le code directement ici.

🪛Pouvez vous écrire une seconde écriture pour la lig. 1 et 7 --?--

Nous allons examiner des méthodes pour ajouter des éléments dans le DOM.

Objectif:

Nous allons apprendre à manipuler les méthodes suivantes.

1. createElement
2. appendChild
3. insertBefore
4. childNodes

Création d'un élément

Soit le code suivant :

```
<body>
    <div id="content">
        <p id="p1">item 1</p>
        <p id="p2">item 2</p>
    </div>
</body>
```

🪛Créez deux paragraphes avec le texte indiqué.

```
const p_1 = document.--?--('p');
const p_2 = document.--?--('p');
p_1.--?-- = "I was created dynamically!";
p_2.--?-- = "I was also created dynamically!”;
```

Compléter les --?-- directement sur ce code

👹Comprenez-vous pourquoi les paragraphes n'apparaissent pas encore.

Nous allons tenter de comprendre la différence entre la mémoire JS et le DOM

Objectif:

Nous allons comprendre qu'un élément créé dans votre code JS doit être ajouté au DOM.

Soit le html de base

```
<div id="content">
      <p id="p1">item 1</p>
      <p id="p2">item 2</p>
</div>
```

🪛Remplacez dans code suivant les --?-- pour insérer :

1️⃣ l'élément *p_1* avant le premier enfant du nœud *id="content"* *et*

2️⃣ l'élément *p_2* après le dernier enfant.

```
1. const p_1 = document.createElement('p');
2. const p_2 = document.createElement('p');
3. p_1.textContent = "I was created dynamically!";
4. p_2.textContent = "I was also created dynamically!";
5.
6. const parent = document.--?--('content');
7. const firstChild = parent.--?--[0];
```

```
8. parent.--?--(p_1, firstChild);
9. parent.--?--(p_2);
```

Voici le html que l'on doit obtenir :

```
<div id="content"><p>I was created dynamically!</p>
    <p id="p1">item 1</p>
    <p id="p2">item 2</p>
<p>I was also created dynamically!</p></div>
```

🆘insertion

Nous allons maintenant découvrir comment manipuler un ensemble de nœuds sélectionnés.

Objectif

Nous allons apprendre à manipuler les opérateurs suivant

- L'opérateur for of sur un itérateur
- La propriété classList pour ajouter/retirer une classe à un élément

🥷Prérequis

Regexp : test

🆒Dans la suite, nous donnerons les expressions régulières à manipuler.

Nous allons aborder l'itération sur une liste.

Itération sur un ensemble.

Soit le code suivant :

```
1. <div id="content">
2.    <p id="p1">urgent : le dom</p>
3.    <p id="p2">item 2</p>
4. <div>
5. <p>test urgent </p>
```

Créez une classe CSS *highlight* dans l'onglet de votre éditeur (codepen, visualStudio ...).

style.css

```
.highlight { background: #ff0; font-style: italic; }
```

Complétez la fonction **highlightParas** qui recherche les paragraphes contenant un mot passé en paramètre.

highlightParas.js

```
1. function highlightParas(containing) {
2.
3.      if(typeof containing === 'string')
4.      containing = new RegExp(`\\b${containing}\\b`, 'i');
5.      const paras = document.--?--('p');
6.
7.      for(let p of --?--) {
8.           if(containing.test(p.textContent))
```

```
9.            p.--?--('highlight');
10.     }
11.   }
12.
13.   highlightParas('urgent');
```

lig.3 : Teste que l'argument est bien un string

lig.4 : Création d'une expression régulière[4]

lig.8 : Teste si le texte de p contient "containing" !

lig.9 : Ajout d'une classe "highlight" au paragraphe contenant "containing"

lig.13 : Lance l'application avec la recherche du mot "urgent"

🪛Nous devrions mettre en avant deux paragraphes contenant le mot "urgent" avec la fonction **`highlightParas('urgent')`**

urgent : le dom

item 2

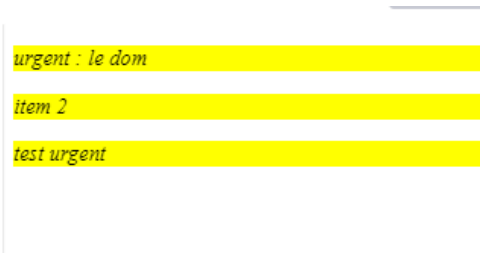
test urgent

🪛Mettez en avant uniquement les paragraphes contenant le nom "item"

Il vous suffit de lancer : **highlightParas('item');**

Voici l'affichage !

[4] regEXP : https://dupontnodejs.blogspot.com/p/es6-regexp.html



👹Ce n'est pas le résultat attendu. En effet le paragraphe "test urgent" ne contient pas le mot "item" alors qu'il est en jaune.

Nous devons mettre en place une fonction **removeParaHighlights()** qui permet avant de lancer une recherche d'annuler les recherches précédentes.

Nous allons découvrir une formidable API : classList.

Objectif

Manipuler les classes d'un élément devient avec classList un jeu d'enfant. Voyons comment !

Nous allons étudier la propriété classList et ses méthodes.

🪛Complétez la fonction **`removeParaHighlights`** qui retire à tous les paragraphes la classe *highligth*.

```
1. function removeParaHighlights() {
2.
3.    const highlightedParas =
   document.--?--('p.highlight');
4.    for(let p of highlightParas) {
5.       p.--?--('highlight');
6.  }
7. }
```

Testez le bon fonctionnement avec le code suivant :

```
highlightParas('urgent');
removeParaHighlights();
highlightParas('item');
```

Nous allons utiliser le template string pour nous faciliter nos manipulations.

template string

🪛Mettez la classe en paramètre de la fonction écrite précédemment.

🪛Créez une nouvelle fonction `removeParasClass`.

removeParasClass.js

```
1. function removeParasClass(removedClass) {
2.  let c = removedClass;
3.  const paras = document.querySelectorAll( `p.${c}` );
4.  for(let p of paras) {
5.    p.--?--(c);
6.  }
7. }
```

Notez lig.3 notez le ` après la) obtenu sur la touche *alt gr 7*

🪛Testez le bon fonctionnement avec le code suivant :

```
highlightParas('urgent');
removeParasClass('highlight');
```

```
3. highlightParas('item');
```

Nous allons découvrir l'intérêt de manipuler les attributs d'un élément. Il permet de mémoriser dans le DOM des données.

Objectifs :

Nous allons revoir les sélecteurs et étudier les attributs [data-]

attributs[5] `data-*`

🥷HTML permet d'associer des données directement dans un élément HTML à l'aide des attributs data-*. Nous pouvons stocker et manipuler de l'information directement.

Dans la figure suivante, nous voyons que le prix d'un article **data-price="3.99"** est mémorisé directement dans le HTML.

```
<td>tomate</td>
<td>
    <input type="number" data-price="3.99" class="quantity"
      value="0" min="0" max="99" maxlength="2">
</td>
<td>3.99</td>
<td>
    <output name="item_total" class="item_total">0.00</output
</td>
```

Output

Produit	Qty	Prix	Total
tomate	0	3.99	0.00
poireau	0	2	0.00
Total			0.00

Il sera récupéré en ainsi :

🥷`itemPrice =parseFloat`[6]`(qtyFields[i].`**`dataset.price`**`);`

[5] https://duponttd.blogspot.com/2016/11/attribut-data.html

[6] Nous récupérons une valeur de type String. **ParseFloat** transforme le String en Float.

Revenons à notre exemple fil-rouge.

Considérons le HTML suivant

```
<button data-action="highlight" data-containing="urgent">
```

Nous avons déclaré un bouton avec deux attributs dataSet, chacun représentant un attribut de données :

1 **`data-action="highlight"`** mémorise le type de l'action.

2 **`data-containing="urgent"`** mémorise le mot à rechercher dans le reste de votre page.

La balise `<button>` contient en "cachette"[7] deux informations utiles à notre programme . Notez que ces attributs sont largement utilisés par les sélecteurs d'attributs .

Objectifs :

🪛Etudiez les attributs [data-] et dataset et comprendre comment récupérer les informations pour la mise en place des fonctions de callback des événements.

Nous allons découvrir comment ces deux attributs de données vont servir à écrire notre code.

🖊️Voici l'expression de nos besoin :

"Si on clique sur le bouton les paragraphes qui contiennent le mot urgent sont mis en lumière."

[7] Elles ne sont pas visibles à l'écran.

Commençons par écrire le code avec les boutons d'interaction.

Soit le HTML

```
<button data-action="highlight" data-containing="urgent">
 URGENT
</button>
<button data-action="removeHighlights">
 Remove
</button>
<div id="content">
   <p id="p1">urgent : le dom</p>
   <p id="p2">item 2</p>
  <div>
  <p>test urgent </p>
```

🪛 Sélectionnez l'ensemble des éléments contenant l'attribut `data-action="highlight"`.

🆘Pour cela, il faut trouver le sélecteur qui permet de récupérer des éléments avec un attribut data-action.

```
const highlightActions=document.querySelectorAll('--?--');
```

🪛Affichez pour le premier élément l'ensemble des attributs avec :

highlightActions[0].dataset, vous devriez avoir le résultat suivant :

[object DOMStringMap[8]] {

action: "highlight",

containing: "urgent"

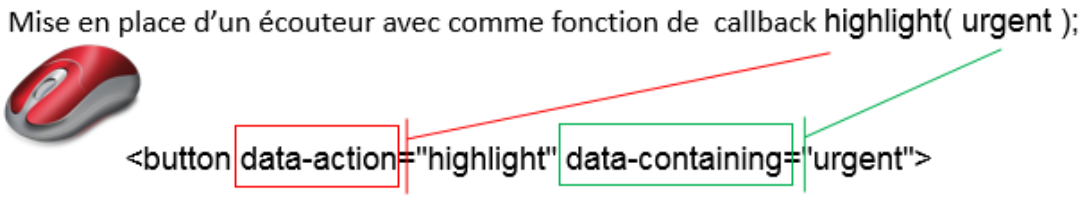
[8] https://developer.mozilla.org/en-US/docs/Web/API/DOMStringMap

}

Et maintenant les interactions. Nous allons passer à la mise en place d'un événement sur un bouton.

event

Nous rappelons nos besoins : *"Si on clique sur le bouton les paragraphes qui contiennent le mot urgent sont mis en lumière."*



Reprenons le code où l'on déclare deux fonctions de callBack.

code

```
function highlightParas(containing) {
    if (typeof containing === 'string')
        containing = new RegExp(`\\b${containing}\\b`, 'i');
    const paras = document.getElementsByTagName('p');
    for (let p of paras) {
        if (containing.test(p.textContent))
            p.classList.add('highlight');
    }
}

function removeParaHighlights() {
    const paras = document.querySelectorAll('p.highlight');
    for (let p of paras) {
        p.classList.remove('highlight');
```

```
15.        }
16.    }
17.
18.    const highlightActions =
   document.querySelectorAll('[data-action="highlight"]');
```

Complétez le code d'un événement sur le **click** ajoutant la classe **highlight.**

```
1. for(let a[9] of highlightActions) {
2.  a.addEventListener(' --?-- ', evt => {
3.  evt.preventDefault();
4.  highlightParas( --?-- );
5.  });
6. }
```

Lig.2 : Ajoutez l'événement 'click'

Lig. 3 : On annule tous les comportements de base.

Lig. 4 : Utilisez **data-containing** qui indique quel mot rechercher.

Ajoutez au code un bouton "clear".

```
<button data-action="removeHighlights">
```

Complétez le code pour ajouter un événement sur le **click** pour supprimer la classe **highlight**.

[9] a est un bouton qui à une data-action égale à highlight.

```
1. const removeHighlightActions =
2.  document.querySelectorAll('[data-action="--?--"]');
3. for(let a[10] of removeHighlightActions) {
4.   a.addEventListener('click', evt => {
5.   evt.preventDefault();
6.   --?--
7.  });
```

🆘Lig. 2 : On recherche les éléments `data-action="removeHighlights"`.

🆘Lig. 6 : On appelle la fonction `removeParaHighlights()`

⏩ En Action.

Remarque !

Sur l'exemple on se rend compte que tous les éléments peuvent être associés à une action, pas uniquement les boutons d'action.

Pour le vérifier, cliquez sur le bouton mais également sur le titre.

🪲Recherche d'un bug !

Contexte : nous avons réécrit le code précédent avec `Array.from`

⏩ (lien).

Array.from permet d'exprimer le "callback" directement :

```
1. Array.from(document.querySelectorAll('[data-action="highlight"]'),
   (el) =>
2.   el.addEventListener("click", (evt) => {
3.     evt.preventDefault();
```

[10] a est un bouton qui à une data-action égale à removeHighlights.

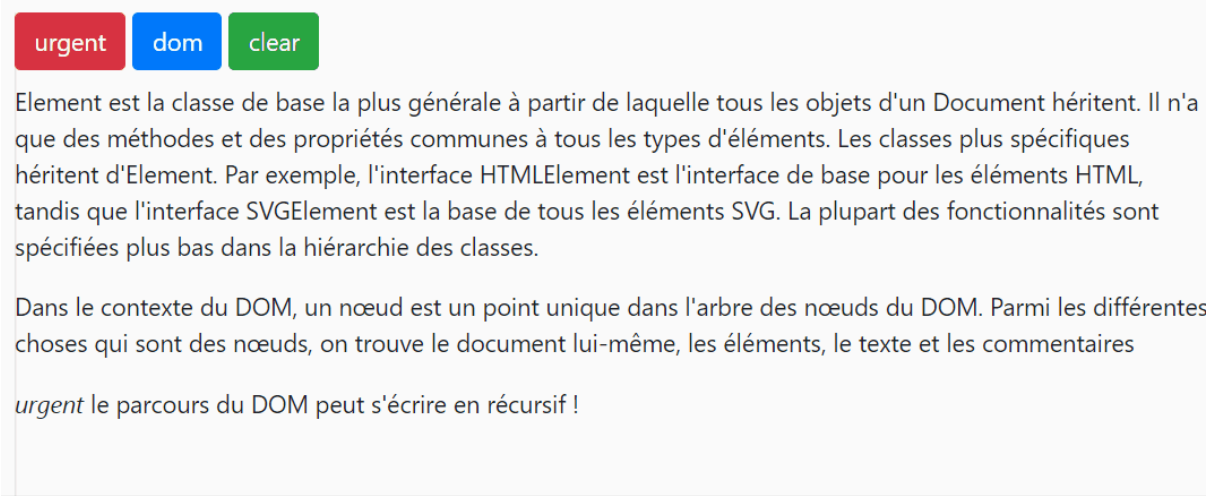
```
4.      highlightParas(el.dataset.containing);
5.    })
6. );
```

Le code précédent est équivalent à

```
1. const highlightActions =
   document.querySelectorAll('[data-action="urgent"]');
2.
3. for (let el of highlightActions) {
4.   el.addEventListener("click", (evt) => {
5.     evt.preventDefault();
6.     highlightParas(el.dataset.containing);
7.   });
8. }
```

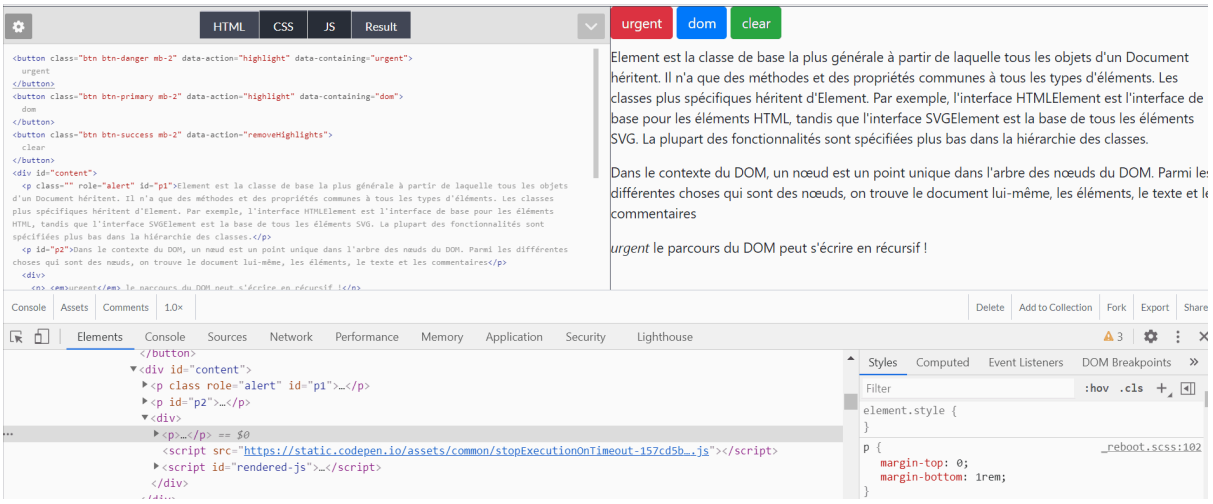
Lig. 1 : Array.from est génial, cela transforme la `NodeList` statique en un tableau et permet d'enchaîner par une méthode map qui est équivalente à la boucle sur l'ensemble des éléments.

Mais, il semble qu'il y ait un bug sur le bouton clear !



🥷Votre mission si vous …[11] est de corriger mon code !

Aide : pensez à inspecter pour comprendre le problème de la fonction remove !



[11] C'est dans les films "Mission impossible", … si vous l'acceptez

🥇Objectif : concevoir une application !

Voici son fonctionnement en ligne : https://dupontdenis.github.io/DOM-TD/

Nous voudrions donc réaliser cette page de recherche.

Recherchez find >

Dans le contexte du DOM, un nœud est un point unique dans l'arbre des nœuds du DOM. Parmi les différentes choses qui sont des nœuds, on trouve le document lui-même, les éléments, le texte et les commentaires

Ajoutez le nom de votre choix

le parcours du DOM peut s'écrire en récursif !

⏩Code : https://codepen.io/dupontcodepen/full/zYBJNpX

Commencez par lire les commentaires : ⏩ Help