Prérequis 

Kit de survie

 DOM 1%

🪛Mise en place

Vous pouvez dans la suite du TD utiliser codepen.io ou VS code.

Visual Studio code

Créez un répertoire 📁dom[1]

C:\Users\DD>mkdir dom

C:\Users\DD>cd dom

Lancer 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.

---

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

Dans VScode créez un fichier index.html

index.html

```
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 s

Dans VScode créez un fichier index.js

index.js

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

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

L'affichage de la console indiquera

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

at index.js:1:15

index.html:41 Live reload enabled.

Mais d'où vient notre problème ? Nous aurions pu écrire des informations dans la console, faire des calculs, mais la structure de la page **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*"

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

Correction

🆒L'erreur observée dans le code précédent est classique. Voici deux corrections possibles.

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.

Revenons aux méthodes de base du DOM.

Objectifs

Nous allons apprendre à manipuler les méthodes suivantes :

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

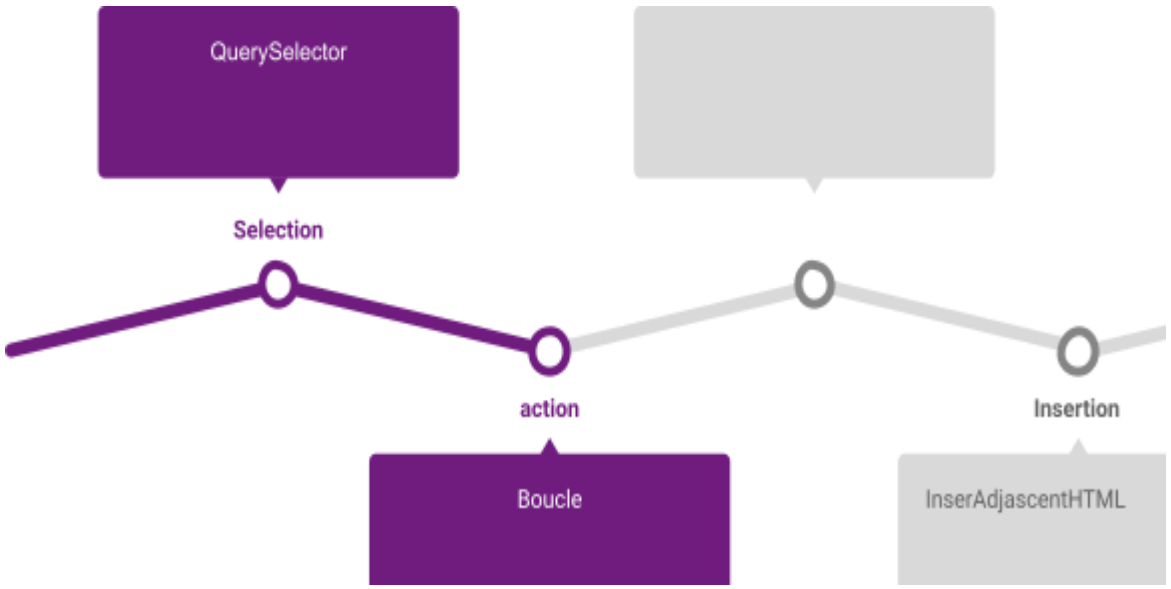
Exercice

🪛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 :

1. Sélectionner le(s) paragraphe(s)
2. Modifier leur contenu

Compléter le code suivant où :

1. Les lig. 1-3 correspondent à la sélection
2. Les 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 :

1. 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[3]

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

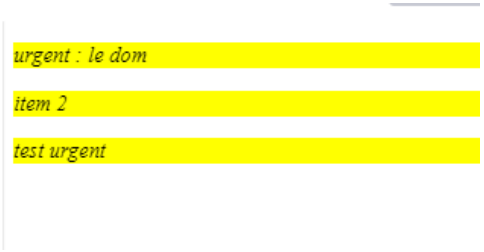
test urgent

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

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

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

Voici l'affichage !



👹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 :

```
1. highlightParas('urgent');
2. removeParaHighlights();
3. 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 :

```
1. highlightParas('urgent');
2. 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[4] 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.

[4] https://duponttd.blogspot.com/2016/11/attribut-data.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`[5]`(qtyFields[i].dataset.price);`

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 :

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

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

La balise `<button>` contient en "cachette"[6] 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.

[5] Nous recuperons une valeur de type String. ParseFloat transforme le String en Float.
[6] Elles ne sont pas visibles à l'écran.

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."

Commençons par écrire le code  avec les buttons 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 :

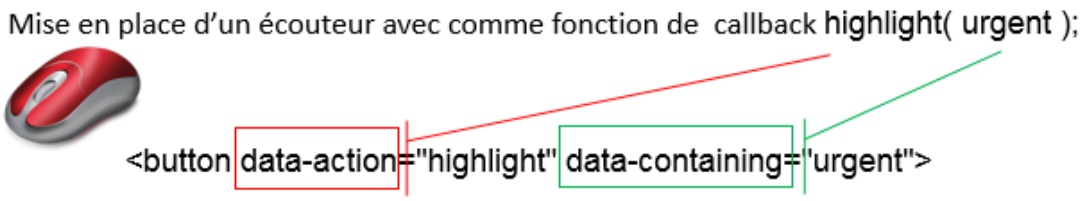
highlightActions[0].dataset, vous devriez avoir le résultat suivant :

[object DOMStringMap[7]] {
  action: "highlight",
  containing: "urgent"
}

Nous allons passer à la mise en place de notre événement sur le boutton.

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 suivant :

code

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

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

```
6.          if (containing.test(p.textContent))
7.              p.classList.add('highlight');
8.      }
9. }
10.
11.   function removeParaHighlights() {
12.       const paras = document.querySelectorAll('p.highlight');
13.       for (let p of paras) {
14.           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 of highlightActions) {
2.  a.addEventListener(' --?-- ', evt => {
3.  evt.preventDefault();
4.  highlightParas( --?-- );
5.  });
6. }
```

 Lig.2 : Ajoutez l'évenement 'click'

En lig. 3 : On annule tous les comportements de base.

 Lig. 4 : Il nous faut utiliser data-set. En particulier 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**.

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

🆘Lig. 2 : On recherche les éléments avec `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();
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 !

urgent dom clear

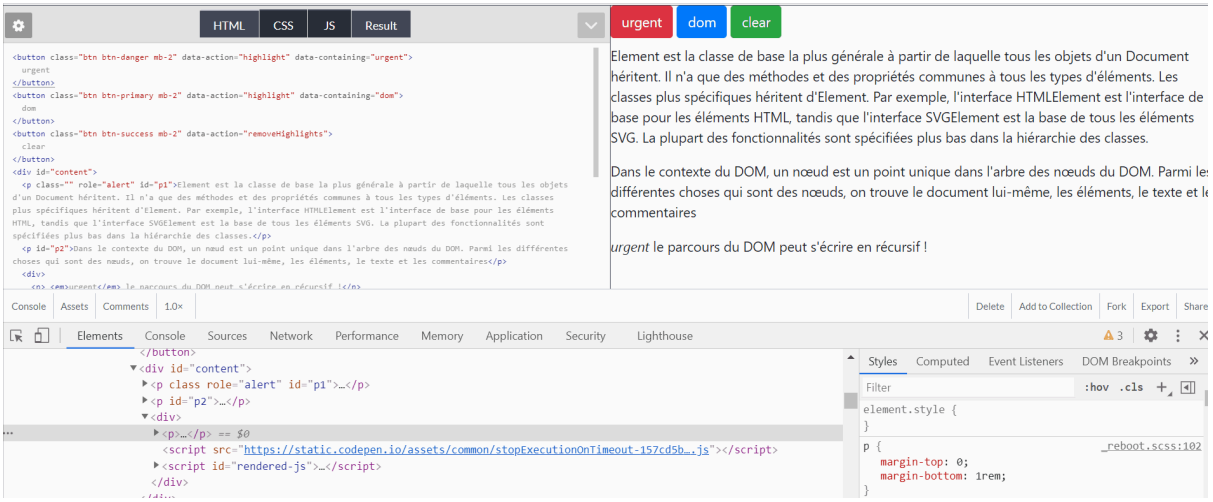
Element est la classe de base la plus générale à partir de laquelle tous les objets d'un Document héritent. Il n'a que des méthodes et des propriétés communes à tous les types d'éléments. Les classes plus spécifiques héritent d'Element. Par exemple, l'interface HTMLElement est l'interface de base pour les éléments HTML, tandis que l'interface SVGElement est la base de tous les éléments SVG. La plupart des fonctionnalités sont spécifiées plus bas dans la hiérarchie des classes.

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

urgent le parcours du DOM peut s'écrire en récursif !

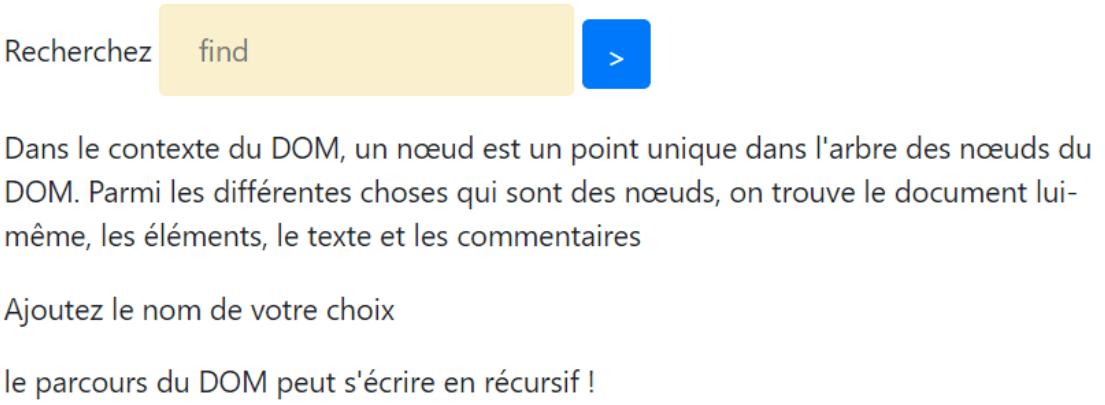
🥷Votre mission si vous … est de corriger mon code !

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



🥇Objectif : concevoir une application !

Nous voudrions réaliser cette page de recherche.



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

Commencez par lire les commentaires : ⏩ Help

👹Suppléments

EX : création d'éléments

On désire à partir d'un objet (récupéré sur une base) afficher un tableau !

```
var PERSON = [
  {name: "dupont", age:52, country: "Tanzania"},
  {name: "tintin", age:"infinie", country: "World"},

];
```

name	age	country
dupont	52	Tanzania
tintin	infinie	World

Voici la structure à créer dynamiquement

Completer le code :

// data est l'objet un tableau d'objet.

function buildTable(data) {

Création de l'objet table

var table = document.--?--("table");

// récupération des entêtes

var fields = Object.keys(data[0]);

// fields = ["name", "age", "country"]

// création de la ligne tr qui contiendra les th

var headRow = document.--?-- ("tr");

// faire un foreach sur fields pour créer le th

fields.--?--(function(field) {

var headCell = document.--?--("th");

// mettre le texte de headCell égale à fields

```
      headCell.--?-- = field;
// ajouter headCell à headRow
      headRow.--?--(headCell);
    });

// ajouter headRow à table
    table.--?--(headRow);

// ajouter une ligne du tableau par objet de data
// faire un forEach sur data
    data.--?--(function(object) {

// créer un tr
      var row = document.--?--("tr");

// pour chaque champ (name,age,country)
      fields.--?--(function(field) {

// création d'une case td
        var cell = document.--?--("td");

// ajouter le texte de l'objet à la case   { name: "dupont", age:52, country:
"Tanzania" },
        cell.--?-- = object[field];

// cas des nombres !
        if (typeof object[field] == "number")
```

```
            cell.style.cssText =
"background-color:black;color:white;text-align:right;"
// ajout à la row la cell
        --?--.appendChild(--?--);
      });

// ajout de la row au tableau
      table.--?--(row);
    });

//retourne le tableau !
    return --?--;
  }
  document.body.appendChild(buildTable(PERSON));
```

EX : Création d'éléments

On aimerait écrire une fonction :

```
document.body.appendChild(
   elt("div", {id: "test", class: "niv"}, "hello",
     elt("p", {class: "picturepanel"}, "para1"),
     elt("div", {class: ""},
       elt("p", {class: "picturepanel"}, "para2"),
       elt("p", {class: "picturepanel"}, "para3"))));
```

```
<div id="test" class="niv">
  "hello"
  <p class="picturepanel">para1</p>
  <div class>
    <p class="picturepanel">para2</p>
    <p class="picturepanel">para3</p>
  </div>
</div>
```

```
function elt(name, attributes) {
   let node = document.createElement(name);

   if (attributes) {
```

Lire : http://duponttd.blogspot.fr/2015/11/enumerale-en-action.html

Faire une boucle sur attributes et mettre chaque attribut à node
node.setAttribute

for...

```
    }
```

Arguments permet de récupérer la liste de tous les arguments d'une fonctions

```
var panel = elt("p", {id: "test", class: "picturepanel"}, "hello", newdiv);
```

"hello" est le second arguments de type "string", newdiv et de type node

```
    for (var i = 2; i < arguments.length; i++) {
      Récupérer l'argument dans la var child
      if (typeof child == "string")
          Créer un texte node et l'affecter dans child
     Ajouter child à node
    }
    return node;
}

//Création d'un paragraphe de texte new et de classe picturepanel
var newP = elt("p", {class: "picturepanel"}, "new");

//Création d'un div de texte hello et d'identifiant "test" et de classe "picturepanel" contenant le noeud newdiv
var panel = elt("div", {id: "test", class: "picturepanel"}, "hello", newP);

//ajout de panel à body
document.body.appendChild(panel);
```

Mais on pourra écrire !

```
document.body.appendChild(
        elt(“div", {id: "test", class: "picturepanel"}, "hello",
            elt("p", {class: "picturepanel"}, "new"));
```

rappels : querySelectorAll

Syntaxe

```
elements = document.querySelectorAll(selecteurs[8]);
```

Exemples :

```
let divs = document.querySelectorAll("div")
```

Remarques :

`divs` n'est pas un tableau mais une liste node.

Boucle classique	new[9]

for(i=0;paras.length>i;i++) {
paras[i].classList.add('highlight');
}
	for(let p of paras) {
	p.classList.add('highlight');
	}

[8] http://dupontcss.blogspot.com/p/selecteurs.html

[9] Pour les vieux navigateurs, l'emploi des boucles `for of` est à remplacer par une boucle classique.

Distantiel

🆘 https://dupontdistanciel.blogspot.com/2020/11/td-dom.html

Quit de survie

Fonctions de base à tester dans la console de http://dupontcours.free.fr/JavaScript/DOM-javascript/dom.html

🆒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 selection des paragraphes

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

1️⃣Ajouter 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", () => {
```

```
    para.classList.add("selected");
  });
}
```

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

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

Affichez le résultat dans un div.

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

🥇La délégation

```
const fun = function (event) {
  const targetElement = event.target; // p or em

  if (targetElement.closest("p")) {
    let para = targetElement.closest("p");
    para.classList.toggle("selected");
  }
};

const parent = document.querySelector(".container");
parent.addEventListener("click", fun, false);
```

https://codepen.io/dupontcodepen/pen/qBJrZxr