Henallux – DA/IA2 – Technologies web

Atelier 5

le D.O.M.

Sommaire

[1. Le DOM et l’arborescence HTML 2](#_Toc179190128)

[2. Modifier les éléments 11](#_Toc179190129)

[3. modifier l’arborescence HTML 15](#_Toc179190130)

[4. un exercice récapitulatif 19](#_Toc179190131)

# Le DOM et l’arborescence HTML

Objectifs

* Rappels sur l’arborescence HTML (cf cours de 1re)
* Découvrir la manière dont elle est représentée en JavaScript
* Explorer l’arborescence d’un document HTML via Javascript

La structure d’arborescence (qu’on appelle souvent de manière imprécise « arbre ») est utilisée dans de nombreux domaines informatiques. En particulier, c’est ce genre de structure qu’on retrouve dans un document HTML. L’objectif de cet exercice est de rappeler les concepts-clefs de l’arborescence HTML, un sujet déjà abordé dans le cours de HTML/CSS.

## Étape 1 : un document comme point de départ

Créez un fichier HTML dans lequel vous copierez le code suivant.

<html>

<head>

<meta charset="UTF-8"/>

<title>Exemple en HTML</title>

</head>

<body>

<p>Une <strong><a href="https://fr.wikipedia.org/wiki/Arbre\_enraciné">arborescence</a></strong> possède :</p>

<ul>

<li>Une <em>racine</em> et</li>

<li>des <em>feuilles</em></li>

</ul>

</body>

</html>

Observez le contenu du fichier et notez que la structure des balises peut être présentée sous la forme d’une arborescence, à savoir la suivante :

html

head

meta

title

body

p

ul

li

em

li

strong

em

a

Si vous n’êtes pas certain de bien connaître la signification des mots suivants, revoyez vos cours précédents et/ou consultez Internet :

* racine (root),
* feuilles (leaves),
* fils (child),
* parent (parent),
* descendant (descendant),
* ancêtre (ancestor),
* frère/sœur (sibling),
* nœud (node)

## Étape 2 : à la découverte du DOM

Lorsque le navigateur reçoit un fichier HTML, il analyse la structure de celui-ci et l’affiche.

Pour pouvoir réaliser du DHTML (= dynamic HTML… vous ne savez plus ce que c’est ? voir les slides du module 1), les scripts JavaScript doivent être capables d’accéder aux divers éléments HTML qui constituent la page.

Le protocole (encore un mot que vous devriez être capables de définir…) utilisé pour permettre à JavaScript de communiquer avec le navigateur au sujet du contenu HTML est le **DOM** (ou **D**ocument **O**bject **M**odel).

Tout en analysant le code HTML de la page à afficher, le navigateur définit toute une série d’objets JavaScript qui correspondent aux différents éléments de son arborescence. La structure de ces objets (attributs, méthodes, hiérarchie…) est définie selon les standards du DOM.

Ouvrez le document créé ci-dessus et, dans la console JavaScript, évaluez la variable

document

La console vous indique que c’est un objet de type HTMLDocument. Cliquez sur le triangle pour voir son contenu.

Cet objet possède un bon paquet de propriétés… Entre autres, vous pouvez apercevoir…

* body, qui représente l’élément HTML correspondant au <body> de la page ;
* head, qui représente l’élément HTML correspondant à la section <head> ;
* images (une HTMLCollection[0]), qui est une sorte de tableau reprenant toutes les images (<img>) du document ; ici, il n’y en a aucune, d’où la taille de 0 ;
* links (une HTMLCollection[1]), qui est une sorte de tableau reprenant tous les éléments HTML de type lien (<a>) ; ici, il n’y en a qu’un seul, d’où la taille de 1 ;
* title, qui reprend le « titre d’onglet » <title> du document HTML.

L’objet JavaScript document représente le document tout entier. À partir de lui, on peut accéder (en lecture et/ou en écriture) à tous les éléments HTML affichés.

Testez par exemple les commandes suivantes une par une et observez leur effet.

document.title = "Ça change le titre";

console.log(document.charset);

alert (document.links[0].href);

## Étape 3 : l’arborescence via le DOM

Dans le contenu de l’objet document, repérez la propriété children. Observez que celle-ci est une HTMLCollection[1], c’est-à-dire un « tableau » de taille 1.

Évaluez

document.children[0]

dans la console pour voir quel est ce « fils » numéro 0.

La console devrait vous répondre en affichant une balise HTML. En fait, le « fils » en question est un objet JavaScript qui, « en interne », représente cette balise et son contenu ou, plus exactement, le nœud correspondant dans l’arborescence HTML.

Toujours dans la console, affectez cet objet à la variable racine. Cet objet possède lui-même une propriété appelée children. Évaluez-la en entrant

racine.children

dans la console.

Notez que les deux fils de racine correspondent aux deux balises principales qu’on retrouve normalement dans chaque document HTML. On peut également accéder à ces deux objets directement à partir de document via les propriétés document.head et document.body. Vous pouvez vous assurer que c’est bien le cas en évaluant les égalités suivantes.

document.head == document.children[0].children[0]

document.body == document.children[0].children[1]

## Étape 4 : « we need to go deeper! » (référence obscure ?)

Poursuivez votre descente dans l’arborescence HTML en évaluant

document.body.children

L’arborescence suivante devrait vous aider à vous y retrouver.

html

head

meta

title

body

p

ul

li

em

li

strong

em

a

Pouvez-vous déterminer à quel élément HTML les expressions suivantes correspondent ? (Vous pouvez vérifier vos réponses en les évaluant).

document.body.children[1].children[0]

document.body.children[0].children[0].children[0]

document.body.children[1].children[1].children[0]

Si l’évaluation de ces lignes ne vous permet pas d’identifier l’élément avec précision, vous pouvez ajouter .innerHTML pour afficher le contenu HTML des éléments en question sous forme de châine de caractères.

document.body.children[1].children[0].innerHTML

document.body.children[0].children[0].children[0].innerHTML

document.body.children[1].children[1].children[0].innerHTML

## Étape 5 : Quelques nœuds de plus…

Jusqu’ici, vous avez pu vous rendre compte que…

* le document HTML suit une structure d’arborescence où chaque nœud est un élément HTML ;
* chaque élément HTML est représenté par un objet JavaScript ;
* chacun de ces objets JavaScript (qu’on appelle des « nœuds HTML ») possède une propriété appelée children et citant les nœuds-enfants.

À côté de la propriété children, il en existe une autre qui lui est similaire… mais légèrement différente. Celle-ci s’appelle childNodes.

Recommençons au début en évaluant

document.childNodes

et en observant que son contenu semble identique à celui de document.children. Aucune différence à ce niveau-ci, ce qu’on peut vérifier en évaluant

document.childNodes[0] == document.children[0]

Que se passe-t-il un niveau plus bas ? Comparez

racine.children

et

racine.childNodes

Cette fois-ci, il y a une différence… les deux n’ont pas le même nombre de composantes. Pouvez-vous l’expliquer ?

(Si vous n’y parvenez pas, aucun inquiétude… continuez l’atelier ; on en reparle plus bas !)

## Étape 6 : children et childNodes

La différence entre ces deux propriétés est peut-être plus claire quand on s’intéresse à un élément HTML « plus profond ». Considérons par exemple l’élément HTML qui correspond au premier élément de la liste <ul>, à savoir la partie

<li>Une <em>racine</em> et</li>

du document HTML.

En utilisant uniquement children, définissez la variable JavaScript li1 pour qu’elle contienne l’objet associé à cette partie du document HTML. Vous pouvez vérifier que vous y êtes parvenu en évaluant

li1.innerHTML

ce qui devrait donner la chaîne de caractères

"Une <em>racine</em> et"

Maintenant, évaluez li1.children et li1.childNodes et comparez les résultats.

* Quels sont les éléments qui apparaissent en plus dans li1.childNodes ?
* À quelle(s) partie(s) du document HTML correspondent-ils ?

Vous devriez avoir remarqué que la propriété children se contente de citer les balises enfants, alors que childNodes cite non seulement ces balises enfants mais également les bouts de texte qui se trouvent entre elles.

Quand on cherche à cibler un élément HTML, cette différence est importante, car les indices des éléments changent… Ainsi, pour li1, l’élément <em>racine</em> est le « child » numéro 0 mais le « childNode » numéro 1.

Pour vérifier que vous avez bien compris la différence, comparez les deux propriétés pour l’élément associé à la balise <ul>. Pour cibler cet élément, vous pouvez repartir de document ou de document.body, mais vous pouvez également utiliser la ligne ci-dessous.

let ul = li1.parentNode;

La propriété parentNode est une autre propriété partagée par tous les nœuds HTML. Celle-ci pointe tout naturellement vers le nœud parent. Et document, possède-t-il un parentNode ?

Maintenant que vous avez ciblé la balise <ul>, comparez

ul.children

et

ul.childNodes

## Étape 7 : et si on automatisait tout ça ?

Pour mieux visualiser l’arborescence HTML, vous allez définir une fonction qui affichera des informations sur un nœud donné puis qui fera de même pour chacun de ses « children » et chacun de ses « childNodes ».

Dans un premier temps, définissez une fonction affiche(elem) qui affichera dans la console les informations suivantes

ELEMENT : ***nomÉlément*** (***xx*** children et ***xx*** childNodes)

Notez que vous pouvez récupérer le nom d’un élément HTML via sa propriété nodeName et que vous pouvez déterminer le nombre de « children » et de « childNodes » comme s’il s’agissait de tableaux (même s’il s’agit plus précisément dans ce cas-ci de « HTMLCollections », ces dernières fonctionnent grosso modo comme des tableaux).

Ainsi, l’appel

affiche(ul)

devrait afficher

ELEMENT : UL (2 children et 5 childNodes)

Ajoutez quelques lignes à votre fonction pour qu’elle cite ensuite les noms de chacun de ses « childNodes ». Ainsi, le même appel que ci-dessus devrait désormais afficher

ELEMENT : UL (2 children et 5 childNodes)

Childnodes : #text LI #text LI #text

Complétez votre fonction en ajoutant des appels récursifs : après avoir affiché ces informations pour l’élément passé en argument, la fonction devra se rappeler elle-même sur chacun des « children » (chacun des éléments-fils).

Cette fois-ci, l’appel devrait produire le résultat suivant.

ELEMENT : UL (2 children et 5 childNodes)

Childnodes : #text LI #text LI #text

ELEMENT : LI (1 children et 3 childNodes)

Childnodes : #text EM #text

ELEMENT : EM (0 children et 1 childNodes)

Childnodes : #text

ELEMENT : LI (1 children et 2 childNodes)

Childnodes : #text EM

ELEMENT : EM (0 children et 1 childNodes)

Childnodes : #text

Cette fonction pourra vous être utile dans les exercices suivants… gardez son code de côté ! (Avez-vous utilisé une instruction for adaptée à ce problème ?)

## Étape 8 : getElement…

Le DOM offre d’autres méthodes plus directes pour cibler un élément HTML dans une page HTML. Vous avez déjà eu l’occasion d’en utiliser une au cours des exercices précédents.

document.getElementById("identificateur")

retourne (l’objet qui décrit) l’élément HTML dont l’identificateur est donné.

À côté de cette méthode, il en existe deux autres fort similaires :

document.getElementsByTagName("h1")

document.getElementsByClassName("important")

le première permet de rechercher des éléments HTML en fonction de leur balise (« tag ») et la seconde, en fonction de leur classe.

Notez une différence importante dans le nom de ces deux méthodes : le « s » après Eléments (quand on vous disait, en 1re, que les « s » dans les identificateurs étaient importants !).

Pour un **identificateur** donné, il ne doit exister qu’un seul élément HTML. Mais le document peut tout à fait comporter plusieurs éléments correspondant à la même **balise** (h1 dans l’exemple) ou ayant la même **classe**.

Ces deux nouvelles méthodes renvoient donc une collection (ce qui revient plus ou moins à un tableau) d’objets plutôt qu’un seul objet.

Dans la console, évaluez

document.getElementsByTagName("li")

et observez que cela renvoie une collection comportant deux objets (correspondant aux deux balises <li>).

Pouvez-vous trouver le code à écrire pour changer la couleur des deux textes en italique ? Conseil : utilisez une boucle for-of, et souvenez-vous que, pour modifier la couleur d’un élément HTML, vous pouvez utiliser

elem.style.color = "blue";

## Étape 9 : le retour des sélecteurs CSS

Finalement, il est également possible de cibler des éléments HTML en utilisant un sélecteur CSS (une bonne occasion de revoir ce sujet ?). Voici deux exemples d’utilisation.

document.querySelector("#menu")

document.querySelectorAll("a.exterieur")

La première expression renvoie le nœud HTML du premier élément HTML qui correspond au sélecteur donné (dans l’exemple, vu qu’on utilise un identificateur, il ne devrait y en n’avoir qu’un de toutes façons). La seconde expression renvoie la collection de tous les nœuds HTML correspondant au sélecteur (ici, tous les liens <a> de classe « exterieur »).

Pouvez-vous déterminer ce que les expressions suivantes vont retourner ?

document.querySelectorAll("ul em")

document.querySelectorAll("li:first-of-type")

Notez que, contrairement à getElementById qui n’est utilisable que sur l’objet document, les quatre autres méthodes (getElementsByTagName, getElementsByClassName, querySelector et querySelectorAll), elles, sont utilisables sur n’importe quel nœud HTML (on limite alors la recherche à l’intérieur de ce nœud HTML) !

Les expressions suivantes sont donc correctes.

ul.getElementsByTagName("em")

li1.querySelectorAll("em")

Vous aurez l’occasion d’utiliser toutes ces méthodes dans les exercices qui suivent.

## Étape 10 : en guise de résumé

Voici les manières de cibler un élément HTML introduites dans les étapes précédentes.

1. *Aller directement sur l’élément*  
   C’est possible pour les balises <head> et <body> auxquelles on peut accéder vis document.head et document.body.
2. *En utilisant les collections*  
   L’objet document possède des propriétés qui regroupent divers éléments de même type. C’est par exemple le cas pour…  
   document.images : liste (« tableau ») de toutes les images <img>  
   document.links : liste (« tableau ») de tous les liens <a>  
   document.forms : liste (« tableau ») de tous les formulaires <form>  
   Par exemple, on peut accéder directement à la 4e image via document.images[3]
3. *En voyageant dans l’arborescence*  
   Une fois qu’on a ciblé un élément, on peut se déplacer « pas à pas » grâce aux propriétés suivantes.  
   children : liste (« tableau ») de toutes les balises enfants  
   firstElementChild, lastElementChild : première/dernière balise enfant  
   childNodes : liste (« tableau ») de tous les nœuds enfants (y compris les textes)  
   firstChild, lastChild : premier/dernier nœud enfant (y compris les textes)  
   parentNode : élément parent  
   previousSibling, nextSibling : frère précédent/suivant (y compris les textes)  
   previousElementSibling, nextElementSibling : balise frère précédent/suivant
4. *En effectuant une recherche*  
   Divers types de critères sont possibles, en fonction de la méthode utilisée.  
   getElementById(id) : selon un identificateur (renvoie 1 nœud HTML, utilisable uniquement sur document)  
   getElementsByTagName(tag) : selon une balise (renvoie une collection)  
   getElementsByClassName(class) : selon une classe (renvoie une collection)  
   querySelector(sel) : selon un sélecteur CSS (renvoie le 1er nœud correspondant)  
   querySelectorAll(sel) : selon un sélecteur CSS (renvoie une collection)

# Modifier les éléments

Objectifs

* Utiliser le DOM pour modifier le contenu de la page HTML
* Utiliser le DOM pour modifier le style d’affichage d’éléments HTML

Savoir cibler un nœud HTML du document est intéressant… mais une fois le nœud ciblé, que peut-on en faire ? Grâce au DOM, on peut accéder à toutes les propriétés qui sont définies dans le code HTML (attributs) et CSS (style).

## Étape 1 : le contenu d’un noeud HTML

Le document HTML qui servira de base à cet exercice est décrit ci-dessous. Il s’agit d’une version légèrement modifiée de celui de l’exercice 1.

<html>

<head>

<meta charset="UTF-8"/>

<title>Exemple en HTML</title>

<style>

.important {color: red}

.tresImportant {background-color: yellow}

</style>

</head>

<body>

<p>Une définition <span class="important">importante</span>.</p>

<p>Une <strong><a href="https://fr.wikipedia.org/wiki/Arbre\_enraciné">arborescence</a></strong> possède :</p>

<ul>

<li id="li1">Une <em data-col="red">racine</em> et</li>

<li>des <em data-col="blue" data-ang="leaf">feuilles</em></li>

</ul>

</body>

</html>

Utilisez l’identificateur li1 pour placer dans la variable li1 l’objet DOM représentant la première balise <li> puis évaluez et comparez les deux expressions suivantes.

li1.textContent

li1.innerHTML

Vous avez sans doute remarqué la différence entre les deux réponses ?

Cette différence existe non seulement quand on tente de lire le contenu d’un nœud HTML mais aussi quand on veut remplacer/définir ce contenu. Entrez les lignes suivantes une par une dans la console et examinez leur effet !

let contenu = "du <strong>gras</strong> et de <em>l’italique</em>";

li1.innerHTML = contenu;

li1.textContent = contenu;

Évaluez ensuite

li1.innerHTML

pour voir ce qui s’est passé lors de l’affectation li1.textContent = …

Certains caractères de la chaîne contenu ont été automatiquement traduits en entités HTML (pour rappel, une entité HTML est une série de symboles commençant généralement par & et se terminant par ; et représentant un caractère qui possède une signification particulière en HTML).

*Résumé.* On peut accéder au contenu d’un nœud HTML via les propriétés innerHTML et textContent. La première permet de manipuler un contenu qui peut contenir des balises HTML ; la seconde s’intéresse au texte affiché.

## Étape 2 : les attributs HTML

Outre le contenu d’un nœud HTML (c’est-à-dire ce qui se trouve entre la balise ouvrante et la balise fermante), le DOM permet également d’accéder aux attributs de la balise.

Considérez la balise <a> correspondant au lien vers Wikipédia. La destination du lien est contenue dans l’attribut href. Le DOM associe la plupart des attributs avec une propriété de même nom.

Après avoir défini la variable lien et lui avoir donné comme valeur le nœud HTML correspondant à l’unique balise <a>, entrez les lignes suivantes une à une dans la console et observez leur effet / la valeur affichée.

lien.href

lien.href = "http://www.google.com";

Cliquez sur le lien pour vérifier le changement.

On peut également ajouter un attribut de la même manière.

lien.target = "\_blank";

*Note.* Une autre manière de lire/modifier la valeur d’un attribut consiste à utiliser les méthodes getAttribute et setAttribute (plus d’infos sur <https://developer.mozilla.org/en/docs/Web/API/Element/getAttribute> et <https://developer.mozilla.org/en/docs/Web/API/Element/setAttribute>).

## Étape 3 : les attributs de données HTML5

On peut également utiliser le code HTML pour ajouter des attributs non standards à une balise. Ces attributs non standards pourraient par exemple être utilisés pour stocker des informations utiles au code JavaScript.

HTML5 introduit la notion de « data attributes » pour mettre un peu d’ordre et standardiser la manière dont ces attributs supplémentaires sont codés. Un « data attribute » est tout simplement un attribut dont le nom commence par « data- »

Dans le DOM, ces « data attributes » sont rassemblés dans la propriété dataset sous la forme d’un objet (ou tableau associatif) dont les clefs sont les mots ajoutés après le tiret.

Évaluez l’expression suivante et interprétez le résultat en vous basant sur le code HTML donné plus haut.

let em2 = li1.nextElementSibling.firstElementChild;

em2.dataset

Quelle syntaxe utiliser pour obtenir la valeur « blue » (au moins deux options s’offrent à vous ; en cas de doute, revoyez les slides sur les objets en JavaScript) ?

Quelle instruction JavaScript faut-il utiliser pour changer la donnée « ang » en « leaves » (au pluriel, au lieu de « leaf ») ? Vérifiez que votre commande a bien un effet en examinant l’élément HTML via l’inspecteur Firefox (clic droit sur « feuilles » puis Inspect).

*Résumé.* On peut accéder aux attributs HTML d’un nœud en utilisant leur nom comme propriété. Pour les « data attributes » associés à des attributs dont le nom commence par « data- », on peut passer par la propriété dataset.

## Étape 4 : le style CSS

Parmi les attributs HTML, « style » joue un rôle particulier vu qu’il se charge de toutes les informations de style « inline ». Le DOM offre des facilités pour accéder aux propriétés CSS qui peuvent être décrites dans l’attribut style.

Ces propriétés CSS se retrouvent sous la forme de propriétés d’objets JavaScript portant quasiment le même nom. Quelques transformations de noms doivent toutefois être prises en compte. Généralement, il s’agit juste de supprimer les tirets et d’utiliser le style camelCase.

Entrez les lignes suivantes comme exemple de modifications de style.

lien.style.backgroundColor = "lemonchiffon";

lien.style.fontStyle = "italic";

lien.style.fontSize = "300%";

lien.style.padding = "10px";

lien.style.border = "1px solid olive";

lien.style.borderRadius = "10px";

À titre d’exercice, définissez une fonction changeCol(elem) qui reçoit un nœud HTML et modifie sa couleur d’écriture (propriété « color »). La nouvelle couleur sera celle qui se trouve dans le **data attribute** col de l’élément (observez le code relatif aux deux balises <em> pour voir un exemple).

Ensuite, placez dans une nouvelle variable ems la collection des nœuds HTML relatifs à des balises <em> (vous avez le choix entre deux méthodes pour faire cela : recherche par balise ou recherche via un sélecteur CSS) puis utilisez une boule for-of pour appliquer la fonction changeCol à chacun de ces éléments.

Si tout se passe bien, le mot « racine » devrait devenir rouge et le mot « feuille », bleu.

*Note importante.* Il vaut mieux n’utiliser la propriété « style » qu’en écriture (pour modifier le style), pas en lecture (pour connaître le style actuel d’un élément). En effet, il s’agit ici du style « inline » de l’élément. Si l’élément possède des caractéristiques de style à cause d’une classe ou d’une règle CSS, celles-ci ne seront pas reprises lors de la lecture de la propriété « style ».

## Étape 5 : la classe et les règles CSS

La ou les classes d’un élément HTML sont citées, en HTML, dans l’attribut « class ». Le DOM, quant à lui, les reprend dans la propriété classList (et pas dans une propriété qui serait appelée « class » !!).

Relisez les deux définitions de style CSS du fichier HTML puis observez l’effet des commandes suivantes, que vous rentrerez une à une.

let span = document.querySelector("span");

span.classList

span.classList.add("tresImportant"); // ajouter une classe

span.classList.toggle("important"); // modifier on/off

span.classList.contains("important") // renvoie un booléen

span.classList.remove("tresImportant"); // enlever une classe

Ces méthodes permettent de donner ou de retirer une classe à un élément HTML. Par contre, il est impossible en JavaScript de modifier directement les définitions des classes (ni les autres règles CSS).

*Résumé.* On peut modifier l’apparence d’un élément HTML soit en ciblant la propriété style soit en lui ajoutant/retirant des classes via la propriété classList. JavaScript ne permet cependant pas de modifier directement les règles CSS.

# modifier l’arborescence HTML

Objectifs

* Utiliser le DOM pour modifier la structure de l’arborescence HTML
* Utiliser le DOM pour créer et ajouter des éléments HTML

Dans les exercices précédents, vous avez vu comment cibler un élément HTML, c’est-à-dire un nœud de l’arborescence et comment modifier son contenu, ses attributs, son style d’affichage et ses classes.

Le DOM permet également de modifier directement l’arborescence HTML en ajoutant, supprimant ou déplaçant des nœuds.

## Étape 1 : le document de départ

Comme dans les exercices précédents, commencez par créer un document HTML avec le contenu suivant, examinez le code et sa structure, puis lisez le document sous Firefox et ouvrez la console.

<html>

<head>

<meta charset="UTF-8"/>

<title>Exemple en HTML</title>

</head>

<body>

<h1>Section 1</h1>

<div id="section1">

<p id="s1p1">La section qui commence par ce paragraphe parle de choses intéressantes.</p>

<p id="s1p2">Nous voici au 2<sup>e</sup> paragraphe et on ne connaît toujours pas le sujet.</p>

<p id="s1p3">Et ce n’est pas le paragraphe numéro 3 qui en dira plus !</p>

<p id="s1p4">Ce quatrième paragraphe est inutile !</p>

</div>

<h1>Section 2</h1>

<div id="section2">

<p id="s2p1">Un seul paragraphe dans la 2<sup>e</sup> section.</p>

</div>

</body>

</html>

## Étape 2 : supprimer un nœud HTML

Lorsqu’on désire déplacer ou supprimer un nœud HTML, il faut donner cet ordre à son père dans l’arborescence.

Dans un premier temps, placez dans la variable cible le nœud HTML correspondant au 4e paragraphe de la 1re section (utilisez son identificateur, « s1p4 »).

À partir de cible, définissez la variable s1 qui devrait contenir le nœud correspondant à la 1re section (le <div> dont l’identificateur est « section1 », mais vous pouvez l’atteindre plus facilement en partant de cible).

Pour supprimer le 4e paragraphe, entrez l’instruction suivante.

s1.removeChild(cible);

Pour supprimer un élément HTML, il faut donc envoyer à son parent un message removeChild en passant le fils à supprimer comme argument.

Notez que l’élément HTML supprimé existe toujours… mais il n’est tout simplement plus rattaché à l’arborescence HTML et donc, plus affiché. Vous pouvez vous en convaincre en évaluant la variable cible puis en entrant l’instruction suivante.

s1.appendChild(cible);

## Étape 3 : déplacer un nœud HTML

Modifiez la valeur de la variable cible pour que, désormais, elle se réfère au nœud HTML correspondant au paragraphe de la section 2. Pour changer, pourquoi ne pas utiliser querySelector et cibler le premier élément HTML répondant au critère : être un paragraphe dans le div d’identificateur « section2 ».

Définissez également la variable suivante.

let p2 = document.getElementById("s1p2");

Dans la suite, on va aborder plusieurs méthodes permettant de déplacer un élément HTML (dans l’exemple, on déplacera le paragraphe correspondant à la variable cible).

La première méthode a déjà été utilisée dans l’étape précédente. Elle consiste à simplement ajouter l’élément à l’intérieur d’un élément parent, en tant que son dernier fils.

s1.appendChild(cible);

Notez que le paragraphe cible a bien été déplacé de sorte qu’il est désormais le dernier fils de s1.

Deux autres méthodes permettent de déplacer plus finement la cible. Le message est encore envoyé au (nouveau) parent de la cible mais on ajoute un argument qui désigne l’un des fils actuels de ce parent. La cible est alors placée soit avant ce fil, soit à la place de ce fils.

Pour placer avant p2 :

s1.insertBefore(cible,p2);

Pour remplacer p2 :

s1.replaceChild(cible,p2);

## Étape 4 : créer un nouveau nœud

Finalement, pour ajouter un nouvel élément HTML dans un document, il faut tout d’abord créer le nœud qui lui correspondra puis, ensuite, le placer dans l’arborescence à l’endroit désiré.

On peut créer un nouvel élément HTML en utilisant la méthode suivante, à laquelle on fournit le nom de la balise correspondant au nœud créé.

let nouveauPara = document.createElement("p");

nouveauPara.innerHTML = "Voici le nouveau paragraphe !";

Une fois le nouveau nœud créé, il faut le « remplir » (voir la seconde instruction ci-dessus) puis le placer au bon endroit. Pour placer le nœud, vous pouvez employer l’une des trois méthodes présentées dans l’étape précédente. Par exemple :

s1.appendChild(nouveauPara);

## Étape 5 : HTMLCollections – une mise en garde

Vous avez peut-être remarqué que les fonctions qui permettent de recueillir une liste d’éléments HTML renvoient un objet appelé une « HTMLCollection ». C’est par exemple le cas pour la définition suivante.

let paras = document.getElementsByTagName("p");

Une HTMLCollection représente un groupe d’éléments HTML et fonctionne « plus ou moins » comme un tableau. Le but de cette étape est de mettre en évidence les différences importantes et de voir comment éviter qu’elles ne causent des problèmes.

Pour voir le contenu de paras, définissez la fonction suivante, qui est censée afficher les identificateurs de chacun des éléments d’une HTMLCollection.

function afficheID (col) {

for (let elem of col) console.log (elem.id);

}

Exécutez cette fonction sur paras. Vous devriez voir apparaître une série d’identificateurs de paragraphes correspondant à ceux qui sont déclarés dans le document HTML (et éventuellement une ligne blanche pour le paragraphe ajouté à l’étape précédente et qui ne comporte pas d’identificateur).

Lisez la boucle suivante, tentez d’en deviner l’effet, puis exécutez-la et observez le résultat…

let s2 = document.getElementById("section2");

for (let para of paras) s2.appendChild(para);

Il y a de grandes chances pour que le résultat ne corresponde pas à ce que vous avez prévu. Certains paragraphes ont été déplacés, mais pas tous… Exécutez à nouveau la fonction afficheID sur paras pour voir ce que la collection contient maintenant.

Comprenez-vous ce qui s’est passé ? Voyez-vous d’où vient le problème ?

En fait, dans une HTMLCollection, les éléments sont toujours cités dans l’ordre dans lesquels ils apparaissent dans le document. Lorsqu’une boucle for-of se contente d’observer les éléments sans les déplacer, cela ne pose pas de problème (c’est le cas de la boucle for-of de la fonction afficheID). Par contre, quand la boucle déplace les éléments dans le document, des problèmes surviennent…

Ici, le 1er élément de la collection était un paragraphe de la section 1. Lorsqu’il a été déplacé dans la section 2, l’ordre des éléments dans la collection a été modifiée : le paragraphe déplacé, plutôt que d’être cité en 1re position, s’est retrouvé plus loin et l’élément qui occupait la 2e position s’est retrouvé en tête de la collection. Malgré tout ça, la boucle for-of s’est poursuivie et s’est occupée de l’élément qui se trouvait alors en 2e position. Ainsi, l’élément qui était, au départ, en 2e position et qui s’est retrouvé propulsé en 1re position n’a pas été traité et ne sera pas traité par la suite. Cela explique pourquoi certains paragraphes n’ont pas été déplacés par la boucle for-of.

Comment éviter ce problème ?

Il faut transformer la HTMLCollection en un tableau (où l’ordre des éléments reste fixe) avant d’exécuter la boucle for-of. Voici une manière de faire (certains éléments de la syntaxe utilisée n’ont pas encore été vus et seront abordés plus tard).

let tabParas = Array.of(...paras);

Évaluez tabParas et constatez qu’il s’agit bien d’un tableau « standard ». Exécutez ensuite la boucle sur cette nouvelle variable.

for (let para of tabParas) s2.appendChild(para);

Maintenant, tous les paragraphes se retrouvent bien dans la section 2.

## Étape 6 : numérotation en guise d’application

Écrivez un script JavaScript qui aura pour effet de numéroter le texte de chacun des paragraphes, c’est-à-dire d’ajouter « 1) » au début du 1er paragraphe, « 2) » au début du 2e paragraphe, et ainsi de suite.

Pouvez-vous réaliser cette boucle directement sur la HTMLCollection ou devez-vous la transformer tout d’abord en un tableau standard ?

Écrivez ensuite une boucle qui va déplacer chacun des paragraphes vers la section 1 de sorte qu’ils s’y retrouvent dans l’ordre inverse (vous pourrez vérifier que votre code atteint bien cet objectif grâce aux numéros que vous venez d’ajouter).

# un exercice récapitulatif

Histoire de faire le point sur tout ce qui a été vu au cours des trois exercices précédents, voici un premier exercice récapitulatif.

## Étape 1 : document HTML

Cet exercice se base sur le fichier HTML suivant.

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8"/>

</head>

<body>

<p>Un tiramisu pour 8 personnes :</p>

<ul>

<li data-key="oeufs">3 gros oeufs</li>

<li data-key="sucre">100g de sucre</li>

<li data-key="sucre vanillé">1 sachet de sucre <strong>vanillé</strong></li>

<li id="masca" data-key="mascarpone">250g de mascarpone</li>

<li data-key="biscuits">24 <i>biscuits</i> à la cuillère</li>

<li data-key="café">1/2 litre de café noir <strong>non sucré</strong></li>

<li data-key="cacao">30g de poudre de cacao <strong>amer</strong></li>

</ul>

</body>

</html>

Avant toute chose, définissez

let b = document.body;

dans la console afin de faciliter l’accès aux éléments HTML de la page.

## Étape 2

On désire obtenir le texte du paragraphe, à savoir « Un tiramisu pour 8 personnes : » (par exemple pour le stocker dans une variable). Parmi les expressions suivantes, quelles sont celles qui donneront la réponse attendue ? Pourquoi les autres réponses ne sont-elles pas correctes ?

* b.children[0].innerHTML
* b.children[1].innerHTML
* b.childNodes[0].innerHTML
* b.childNodes[1].innerHTML
* b.firstChild.innerHTML
* b.firstElementChild.innerHTML
* b.getElementsByTagName("p").innerHTML
* b.querySelector("p").innerHTML

## Étape 3

On désire obtenir le texte de la ligne concernant le sucre, à savoir « 100g de sucre ». Comment l’obtenir à partir de la variable b en utilisant

(a) children,

(b) childNodes,

(c) firstChild et nextSibling,

(d) firstElementChild et nextElementSibling ?

## Étape 4

Si on définit

let masca = document.getElementById("masca");

comment obtenir, à partir de la variable masca, le contenu en gras de la ligne correspondant au sachet de sucre (à savoir « vanillé ») ?

Que pensez-vous de l’expression suivante ? Comment la compléter pour obtenir le résultat désiré ?

masca.parentNode.getElementsByTagName

## Étape 5

Déterminez les valeurs des expressions suivantes (pour vérifier vos réponses, entrez-les simplement dans la console !)

b.children[1].childNodes[9].children[0].innerHTML

masca.parentNode.lastElementChild.lastElementChild.innerHTML

b.getElementsByTagName("strong")[1].parentNode.innerHTML

b.getElementsByTagName("strong")[1].parentNode.textContent

## Étape 6

Écrivez des scripts JavaScript dont les effets seront les suivants :

1. Modifier la couleur d’écriture de chacun des éléments en gras (balise <strong>).
2. Ajouter en fin de liste l’ingrédient manquant, à savoir : « un petit verre de marsala sec ». Associez-lui aussi un data attribute « key » valant « marsala ».
3. Préfixer chacun des éléments de la liste d’un label contenant le mot-clef de l’ingrédient entre crochets (et suivi d’un espace). Ces labels seront contenus dans des balises <span> qui posséderont la classe « label ». Notez que le mot-clef d’un ingrédient peut être récupéré dans le data attribute « key ».
4. Cibler tous les <span> ajoutés au point 2 en utilisant leur classe et modifier leur style d’écriture pour qu’ils soient en italique.
5. Cibler tous les éléments de la liste qui comportent une partie en gras et de les déplacer vers le début de la liste.  
   *Note : utiliser un sélecteur pour cibler toutes les balises <strong> qui se trouvent dans un <li> ; à partir des éléments ciblés, trouvez comment obtenir la balise <li> qui doit être déplacée. Une autre option consiste à utiliser un sélecteur CSS plus complexe pour cibler directement les <li> qui contiennent un <strong>.*

## Étape 7

Ouvrez un nouvel onglet et effectuez une recherche sur Google (sur le sujet que vous voulez, mais arrangez-vous pour qu’il y ait plusieurs résultats).

Observez que, pour chaque résultat, Google donne un titre.

Écrivez un script JavaScript qui affiche dans la console les textes des titres en question (bien sûr, le script doit fonctionner pour toutes les pages de résultats Google). Utilisez l’inspecteur de Firefox pour en apprendre plus sur la structure de la page.