Aubrée Romain **Sotoca**
10/12/2025

Groupe 1

Atelier 12 TP 1 Partie 2

1)

a)

Regardez cette page :

```
<html>
<body>
  <div>Users:</div>
  <ul>
    <li>John</li>
    <li>Pete</li>
  </ul>
</body>
</html>
```

Pour chacun des éléments suivants, donnez au moins un moyen d'y accéder :

- Le noeud `<div>` du DOM ?
- Le noeud `<ul>` du DOM ?
- Le deuxième `<li>` (avec Pete) ?

On utilise *"document.body.firstElementChild"*

On utilise *"document.body.lastElementChild"*

On utilise *"document.body.lastElementChild.lastElementChild"*

b)

Si `element` – est un nœud élément arbitraire du DOM ...

- Est-il vrai que `elem.lastChild.nextSibling` est toujours `null` ?
- Est-il vrai que `elem.children[0].previousSibling` est toujours `null` ?

Oui, car *"lastChild"* désigne le dernier enfant, et vue que le dernier n'a pas de suivant il est forcément *"null"*.

Non, du moins ça dépend, si il y a un *“child”* d’un autre nœud il prendra celui-là, sinon, si c’est le premier *“child”* utilisé, il prendra *“null”*.

c)

Écrivez le code pour colorer toutes les cellules du tableau diagonal en rouge.

Vous devrez obtenir toutes les diagonales `<td>` de la `<table>` et les colorer en utilisant le code :

```
// td doit être la référence à la cellule du tableau
td.style.backgroundColor = 'red';
```

Le résultat devrait être :

1:1	2:1	3:1	4:1	5:1
1:2	2:2	3:2	4:2	5:2
1:3	2:3	3:3	4:3	5:3
1:4	2:4	3:4	4:4	5:4
1:5	2:5	3:5	4:5	5:5

```
<!DOCTYPE html>
<body>
  <script>
    let table = document.createElement('table');
    table.border = "1";
    document.body.append(table);

    let ligne = 0;
    while (ligne < 5) {
      let tr = table.insertRow();

      let cellule = 0;
      while (cellule < 5) {
        tr.insertCell().innerText = "X";
        cellule++;
      }
      ligne++;
    }
    let i = 0;
    while (i < table.rows.length) {
      table.rows[i].cells[i].style.backgroundColor = 'red';
      i++;
    }
  </script>
</body>
```

2)

Voici le document avec le tableau et formulaire

Comment trouver ?...

1. Le tableau avec `id="age-table"`.
2. Tous les éléments `label` dans ce tableau (il devrait y en avoir 3).
3. Le premier `td` dans ce tableau (avec le mot "Age").
4. Le `form` avec `name="search"`.
5. Le premier `input` dans ce formulaire.
6. Le dernier `input` dans ce formulaire.

a)

- *"let table = document.getElementById('age-table');"*
- *"let labels = table.querySelectorAll('label');"*
- *"let td = table.querySelector('td');"*
- *"let form = document.querySelector('form[name="search"]');"*
- *"let firstInput = form.querySelector('input');"*
- *"let inputs = form.querySelectorAll('input');*

  let lastInput = inputs[inputs.length - 1];"

3)

a)

Le script affiche 1.

Le navigateur est en train de lire le <body>.

La balise <script> est le dernier élément inséré.

Donc, document.body.lastChild correspond à la balise <script> elle-même.

La propriété *“nodeType”* pour une balise élément (comme *“<script>”*) est 1.

b)

Cela va afficher **“BODY”**.

c)

- “document” est une instance de la classe HTMLDocument. C'est le point d'entrée de la page.
- Il se trouve en haut de l’arbre logique.
- Il hérite de “Node”.
- C'est un nœud du DOM. C'est pour ça qu'il possède des propriétés comme *“.firstChild”*, *“.lastChild”* ou *”.childNodes”*.

  Ce n’est pas un *“Element”* ou *“HTMLElement”*.

  Le document est le conteneur, il n'est pas lui-même une balise.

  Les balises comme *“<body>”* ou *“ ”* héritent de *“Element”*. Le document, lui, hérite directement de *“Node”* via *“Document”*.

4)

a)

Écrivez le code pour sélectionner l'élément avec l'attribut `data-widget-name` dans le document et pour lire sa valeur.

```
<!DOCTYPE html>
<html>
<body>

  <div data-widget-name="menu">Choose the genre</div>

  <script>
    /* your code */
  </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

  <div data-widget-name="menu">Choose the genre</div>
  <script>
    let element = document.querySelector('[data-widget-name]');
    let valeur1 = element.dataset.widgetName;
    let valeur2 = element.getAttribute('data-widget-name');
    alert(valeur1);
  </script>

</body>
</html>
```

b)

Mettez tous les liens externes en orange en modifiant leur propriété `style`.

Un lien est externe si :

- Son `href` contient `://`
- Mais ne commence pas par `http://internal.com`.

Exemple :

```html
<a name="list">the list</a>
<ul>
  <li><a href="http://google.com">http://google.com</a></li>
  <li><a href="/tutorial">/tutorial.html</a></li>
  <li><a href="local/path">local/path</a></li>
  <li><a href="ftp://ftp.com/my.zip">ftp://ftp.com/my.zip</a></li>
  <li><a href="http://nodejs.org">http://nodejs.org</a></li>
  <li><a href="http://internal.com/test">http://internal.com/test</a></li>
</ul>

<script>
  // setting style for a single link
  let link = document.querySelector('a');
  link.style.color = 'orange';
</script>
```

Le résultat devrait être :

The list:

- http://google.com
- /tutorial.html
- local/path
- ftp://ftp.com/my.zip
- http://nodejs.org
- http://internal.com/test

```html
<html>
<style>
  body { font-family: sans-serif; }
  ul { list-style-type: disc; }
</style>
<body>
  <a name="list">the list</a>
  <ul>
    <li><a href="http://google.com">http://google.com</a></li>
    <li><a href="/tutorial">/tutorial.html</a></li>
    <li><a href="local/path">local/path</a></li>
    <li><a href="ftp://ftp.com/my.zip">ftp://ftp.com/my.zip</a></li>
    <li><a href="http://nodejs.org">http://nodejs.org</a></li>
    <li><a href="http://internal.com/test">http://internal.com/test</a></li>
  </ul>
  <script>
    let links = document.querySelectorAll('a');
    for (let link of links) {
      let href = link.getAttribute('href');
      if (!href) continue;
      if (href.includes('://') && !href.startsWith('http://internal.com')) {

        link.style.color = 'orange';
      }
    }
  </script>

</body>
</html>
```

the list

- http://google.com
- /tutorial.html
- local/path
- ftp://ftp.com/my.zip
- http://nodejs.org
- http://internal.com/test

5)

a)

Nous avons un élément DOM vide `elem` et une chaîne de caractères `text`.

Lesquelles de ces 3 commandes feront exactement la même chose ?

1. `elem.append(document.createTextNode(text))`
2. `elem.innerHTML = text`
3. `elem.textContent = text`

Le 1 et le 3 car :

Avec 1 et 3 : On verra écrit : *"Salut"* (avec les *"<>"*).

Avec 2 : On verra écrit : *"Salut"* (en gras, sans les *"<>"*).

b)

Créez une fonction `clear(elem)` qui supprime tout de l'élément.

```
<ol id="elem">
  <li>Hello</li>
  <li>World</li>
</ol>

<script>
  function clear(elem) { /* votre code */ }

  clear(elem); // efface la liste
</script>
```

```html
<!DOCTYPE html>
<html>
<body>

<ol id="elem">
  <li>Hello</li>
  <li>World</li>
</ol>

<script>
  function clear(elem) {
    elem.innerHTML = '';
  }
  clear(document.getElementById('elem'));
</script>

</body>
</html>
```

c)

Dans l'exemple ci-dessous, l'appel `table.remove()` supprime le tableau du document.

mais si vous l'exécutez, vous pouvez voir que le texte `"aaa"` est toujours visible.

Pourquoi cela se produit-il ?

```html
<table id="table">
  aaa
  <tr>
    <td>Test</td>
  </tr>
</table>

<script>
  alert(table); // la table, comme il se doit

  table.remove();
  // pourquoi y a-t-il encore "aaa" dans le document ?
</script>
```

"aaa" est à l'extérieur de la table donc il n'est pas affecté.

d)

Écrivez une interface pour créer une liste à partir des entrées utilisateur.

Pour chaque élément de la liste :

1. Interrogez un utilisateur sur son contenu en utilisant `prompt`.
2. Créez le `<li>` avec et ajoutez-le à `<ul>`.
3. Continuez jusqu'à ce que l'utilisateur annule l'entrée (en appuyant sur la touche `Esc` ou une entrée vide).

Tous les éléments doivent être créés dynamiquement.

Si un utilisateur tape des balises HTML, elles doivent être traitées comme un texte.

```
<!DOCTYPE html>
<html>
<body>

<script>
  const ul = document.createElement('ul');
  document.body.appendChild(ul);

  while (true) {
    let data = prompt("Entrez un élément :");

    if (data === null || data === "") {
      break;
    }

    let li = document.createElement('li');
    li.textContent = data;

    ul.appendChild(li);
  }
</script>

</body>
</html>
```

e)

Écrivez une fonction `createTree` qui crée une liste imbriquée `ul/li` à partir de l'objet imbriqué.

Par exemple :

```
let data = {
  "Fish": {
    "trout": {},
    "salmon": {}
  },

  "Tree": {
    "Huge": {
      "sequoia": {},
      "oak": {}
    },
    "Flowering": {
      "apple tree": {},
      "magnolia": {}
    }
  }
};
```

La syntaxe :

```
let container = document.getElementById('container');
createTree(container, data); // crée l'arbre dans le conteneur
```

```
<!DOCTYPE html>
<html>
<body>
<div id="container"></div>
<script>
  let data = {
    "Fish": {
      "trout": {},
      "salmon": {}
    },
    "Tree": {
      "Huge": {
        "sequoia": {},
        "oak": {}
      },
      "Flowering": {
        "apple tree": {},
        "magnolia": {}
      }
    }
  };
  function createTree(container, obj) {
    if (Object.keys(obj).length === 0) return;
    let ul = document.createElement('ul');
    for (let key in obj) {
      let li = document.createElement('li');
      li.textContent = key;
      createTree(li, obj[key]);
      ul.appendChild(li);
    }
    container.appendChild(ul);
  }
  let container = document.getElementById('container');
  createTree(container, data);
</script>
</body>
</html>
```

f)

Il y a un arbre organisé comme un `ul/li` imbriqué.

Écrivez le code qui ajoute à chaque `<li>` le nombre de ses descendants. Sautez les feuilles (nœuds sans enfants).

Le resultat :

- Animals [9]
  - Mammals [4]
    - Cows
    - Donkeys
    - Dogs
    - Tigers
  - Other [3]
    - Snakes
    - Birds
    - Lizards
- Fishes [5]
  - Aquarium [2]
    - Guppy
    - Angelfish
  - Sea [1]
    - Sea trout

```
<!DOCTYPE html>
<html>
<body>
<div id="container"></div>
<script>
  let lis = document.querySelectorAll('li');

for (let li of lis) {
  let count = li.querySelectorAll('li').length;

  if (count > 0) {
    li.firstChild.textContent += ' [' + count + ']';
  }
}
</script>
</body>
</html>
```

g)

Écrivez une fonction `createCalendar(elem, year, month)`.

L'appel doit créer un calendrier pour l'année/le mois donné et le mettre dans `elem`.

Le calendrier doit être un tableau, où une semaine est un `<tr>` et un jour est un `<td>`. Le dessus du tableau doit être un `<th>` avec les noms des jours de la semaine : le premier jour doit être le lundi, et ainsi de suite jusqu'au dimanche.

Par exemple, `createCalendar(cal, 2012, 9)` devrait générer dans l'élément `cal` le calendrier suivant :

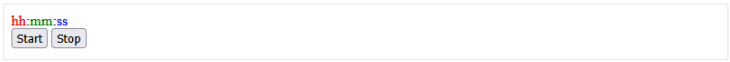
MO	TU	WE	TH	FR	SA	SU
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

P.S. Pour cette tâche, il suffit de générer le calendrier, il ne doit pas encore être cliquable.

```
<!DOCTYPE html>
<html>
<body>
<div id="container"></div>
<script>
  function createCalendar(elem, year, month) {
    let mon = month - 1;
    let d = new Date(year, mon);

    let table = '<table><tr><th>MO</th><th>TU</th><th>WE</th><th>TH</th><th>FR</th><th>SA</th><th>SU</th></tr><tr>';

    let day = d.getDay();
    if (day === 0) day = 7;
    for (let i = 1; i < day; i++) {
      table += '<td></td>';
    }

    while (d.getMonth() === mon) {
      table += '<td>' + d.getDate() + '</td>';

      if (d.getDay() === 0) {
        table += '</tr><tr>';
      }
      d.setDate(d.getDate() + 1);
    }

    table += '</tr></table>';

    elem.innerHTML = table;
  }
</script>
</body>
</html>
```

h)

Créez une horloge colorée comme ici :



Utilisez HTML/CSS pour le style, JavaScript ne met à jour que le temps dans les éléments.

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .hh { color: red; }
    .mm { color: green; }
    .ss { color: blue; }
  </style>
</head>
<body>

  <div>
    <span id="h" class="hh">hh</span>:<span id="m" class="mm">mm</span>:<span id="s" class="ss">ss</span>
  </div>

  <button onclick="start()">Start</button>
  <button onclick="stop()">Stop</button>

  <script>
    let timer;

    function format(num) {
      return num < 10 ? '0' + num : num;
    }

    function update() {
      let d = new Date();
      document.getElementById('h').textContent = format(d.getHours());
      document.getElementById('m').textContent = format(d.getMinutes());
      document.getElementById('s').textContent = format(d.getSeconds());
    }

    function start() {
      if (!timer) {
        timer = setInterval(update, 1000);

        function stop() {
          clearInterval(timer);
          timer = null;
        }
      </script>

    </body>
    </html>
```

i)

Écrivez le code pour insérer `<li>2</li><li>3</li>` entre deux `<li>` ici :

```
<ul id="ul">
  <li id="one">1</li>
  <li id="two">4</li>
</ul>
```

```
<!DOCTYPE html>
<html>
<body>

<ul id="ul">
  <li id="one">1</li>
  <li id="two">4</li>
</ul>

<script>
  let one = document.getElementById('one');

  one.insertAdjacentHTML('afterend', '<li>2</li><li>3</li>');
</script>
</body>
</html>
```

i)

Il y a un tableau :

```
<table>
<thead>
  <tr>
    <th>Name</th><th>Surname</th><th>Age</th>
  </tr>
</thead>
<tbody>
  <tr>
    <td>John</td><td>Smith</td><td>10</td>
  </tr>
  <tr>
    <td>Pete</td><td>Brown</td><td>15</td>
  </tr>
  <tr>
    <td>Ann</td><td>Lee</td><td>5</td>
  </tr>
  <tr>
    <td>...</td><td>...</td><td>...</td>
  </tr>
</tbody>
</table>
```

Il peut y avoir plus de lignes.

Écrivez le code pour le trier par la colonne `"name"`.

```
<!DOCTYPE html>
<html>
<body>
  <table id="grid">···
  </table>
  <script>
    let table = document.getElementById('grid');
    let tbody = table.querySelector('tbody');
    let rowsArray = Array.from(tbody.rows);
    rowsArray.sort((rowA, rowB) => {
      let nameA = rowA.cells[0].textContent;
      let nameB = rowB.cells[0].textContent;

      return nameA.localeCompare(nameB);
    });
    tbody.append(...rowsArray);
  </script>
</body>
</html>
```

6)

a)

Write a function `showNotification(options)` that creates a notification: `<div class="notification">` with the given content. The notification should automatically disappear after 1.5 seconds.

The options are:

```
// shows an element with the text "Hello" near the right-top of the window
showNotification({
  top: 10, // 10px from the top of the window (by default 0px)
  right: 10, // 10px from the right edge of the window (by default 0px)
  html: "Hello!", // the HTML of notification
  className: "welcome" // an additional class for the div (optional)
});
```

```
<!DOCTYPE html>
<html>
<body>

  <script>
    function showNotification({top = 0, right = 0, className, html}) {
      let div = document.createElement('div');
      div.className = "notification";
      if (className) div.classList.add(className);
      div.style.top = top + 'px';
      div.style.right = right + 'px';
      div.innerHTML = html;
      document.body.append(div);
      setTimeout(() => div.remove(), 1500);
    }

    showNotification({
      top: 10,
      right: 10,
      html: "Hello!",
      className: "welcome"
    });
  </script>

</body>
</html>
```

b)

Écrivez le code qui renvoie la largeur d'une barre de défilement standard.

Pour Windows, il varie généralement entre `12px` et `20px`. Si le navigateur ne lui réserve pas d'espace (la barre de défilement est à moitié translucide sur le texte, cela arrive également), alors il peut s'agir de `0px`.

P.S. Le code devrait fonctionner pour tout document HTML, ne dépend pas de son contenu.

```
<!DOCTYPE html>
<html>
<body>

  <script>
    function getScrollbarWidth() {
      const outer = document.createElement('div');
      outer.style.visibility = 'hidden';
      outer.style.width = '100px';
      outer.style.position = 'absolute';
      outer.style.top = '-9999px';
      outer.style.overflow = 'scroll';
      document.body.appendChild(outer);
      const inner = document.createElement('div');
      inner.style.width = '100%';
      outer.appendChild(inner);
      const scrollbarWidth = outer.offsetWidth - inner.offsetWidth;
      outer.parentNode.removeChild(outer);

      return scrollbarWidth;
    }

    console.log("Largeur de la scrollbar : " + getScrollbarWidth() + "px");
  </script>
</body>
</html>
```

c)

La propriété `elem.scrollTop` est la taille de la partie déroulante à partir du haut. Comment obtenir la taille du défilement inférieur (appelons-le `scrollBottom`) ?

Écrivez le code qui fonctionne pour un `element` arbitraire.

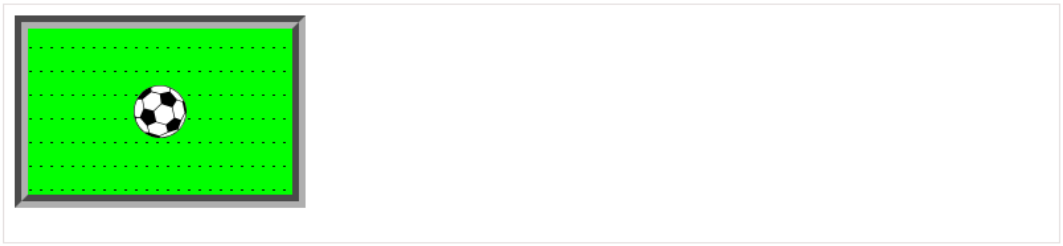
P.S. Veuillez vérifier votre code: s'il n'y a pas de défilement ou que l'élément est entièrement défilé vers le bas, alors il devrait retourner `0`.

```
<!DOCTYPE html>
<html>
<body>

  <script>
    function getScrollBottom(elem) {
      return elem.scrollHeight - elem.scrollTop - elem.clientHeight;
    }
  </script>

</body>
</html>
```

d)

Quelles sont les coordonnées du centre de terrain ?

Calculez et utilisez-les pour placer la balle au centre du champ vert :



- L'élément doit être déplacé par JavaScript, pas CSS.
- Le code doit fonctionner avec n'importe quelle taille de boule (`10`, `20`, `30` pixels) et n'importe quelle taille de champ, ne pas être lié aux valeurs données.

P.S. Bien sûr, le centrage pourrait être effectué avec CSS, mais ici, nous voulons exactement JavaScript. De plus, nous rencontrerons d'autres sujets et des situations plus complexes lorsque JavaScript doit être utilisé. Ici, nous faisons un "échauffement".

```
window.onload = function() {
  let field = document.getElementById('field');
  let ball = document.getElementById('ball');
  let centerX = Math.round(field.clientWidth / 2 - ball.offsetWidth / 2);
  let centerY = Math.round(field.clientHeight / 2 - ball.offsetHeight / 2);
  ball.style.left = centerX + 'px';
  ball.style.top = centerY + 'px';
  console.log(`La balle est placée à : Left: ${centerX}px, Top: ${centerY}px`);
}
```

e)

1) *"getComputedStyle"* renvoie du texte avec l'unité, alors que *"clientWidth"* renvoie un nombre pur.
2) *"clientWidth"* inclut le padding et retire la barre de défilement, alors que *"getComputedStyle"* donne la largeur CSS brute.
3) *"getComputedStyle"* donne des décimales précises, alors que *"clientWidth"* arrondit toujours à l'entier.

7)

a)

Dans l'iframe ci-dessous, vous pouvez voir un document avec le "champ" vert.

Utilisez JavaScript pour trouver les coordonnées de la fenêtre des coins pointés par des flèches.

Il y a une petite fonctionnalité implémentée dans le document pour plus de commodité. Un clic à n'importe quel endroit montre les coordonnées là-bas.

```
<script>
  document.onclick = function(e) {
    document.getElementById('coords').innerHTML = e.clientX + ':' + e.clientY;
  };
  let field = document.getElementById('field');
  let rect = field.getBoundingClientRect();
  let coin1 = [rect.left, rect.top];
  let coin2 = [rect.right, rect.bottom];
  let coin3 = [rect.left + field.clientLeft, rect.top + field.clientTop];
  let coin4 = [
    rect.left + field.clientLeft + field.clientWidth,
    rect.top + field.clientTop + field.clientHeight
  ];
  console.log("1. Extérieur Haut-Gauche : " + coin1.join(':'));
  console.log("2. Extérieur Bas-Droite  : " + coin2.join(':'));
  console.log("3. Intérieur Haut-Gauche : " + coin3.join(':'));
  console.log("4. Intérieur Bas-Droite  : " + coin4.join(':'));
</script>
```

b)

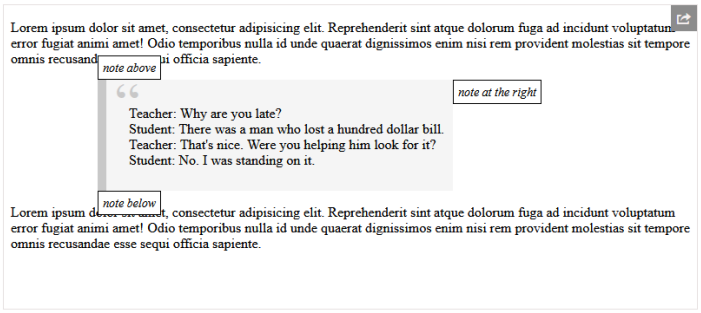
Créez une fonction `positionAt(anchor, position, elem)` qui positionne `elem`, en fonction de `position` près de l'élément `anchor`.

La `position` doit être une chaîne de caractères avec l'une des 3 valeurs :

- `"top"` – position `elem` juste au dessus de `anchor`
- `"right"` – position `elem` immédiatement à droite de `anchor`
- `"bottom"` – position `elem` juste en dessous `anchor`

Il est utilisé à l'intérieur de la fonction `showNote(anchor, position, html)`, fournie dans le code source de la tâche, qui crée un élément "note" avec `html` donné et l'affiche à la `position` donnée près de `anchor`.

Voici la démo des notes :




```
<!DOCTYPE html>
<html>
<body>

        <script>
          function positionAt(anchor, position, elem) {
            let anchorCoords = anchor.getBoundingClientRect();
            elem.style.position = 'absolute';
            let top = 0;
            let left = 0;
            switch (position) {
              case "top":
                top = anchorCoords.top - elem.offsetHeight;
                left = anchorCoords.left;
                break;

              case "right":
                top = anchorCoords.top;
                left = anchorCoords.right;
                break;
              case "bottom":
                top = anchorCoords.bottom;
                left = anchorCoords.left;
                break;
            }
            elem.style.top = (top + window.scrollY) + "px";
            elem.style.left = (left + window.scrollX) + "px";
          }
                </script>

</body>
</html>
```

Conclusion

Approfondissement sur le JavaScript.