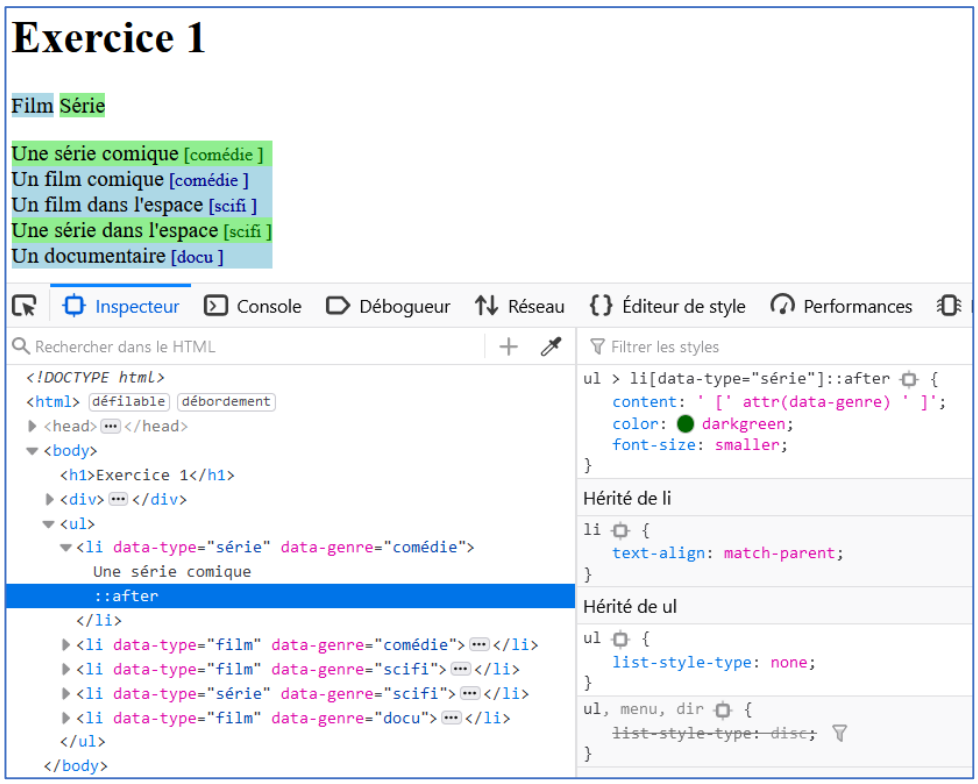
IUT GON – MMI – S5 – Dev Front TD 2 - Personnalisation

Exercice 1 – Attributs personnalisés et CSS

Considérons le document exo1.html. Il contient une liste non-ordonnée dont les éléments ont des attributs personnalisés `data-type` et `data-genre`. Sans modifier le code HTML, complétez le code CSS en utilisant ces attributs pour :

- Styliser la couleur de fond des éléments `li` et `span` selon la valeur de l'attribut `data-type` (fond vert clair si série, fond bleu clair si film).
- Ajouter comme dernier enfant de chaque `li` la valeur de `data-genre` qui lui est associée, encadrée de crochets [ ], avec le pseudo-élément `::after`. La couleur du texte doit être vert foncé si `data-type` est une série et bleu foncé pour un film. La taille du texte est plus petite que le texte principal des éléments de liste.

Vous devez obtenir le rendu suivant.

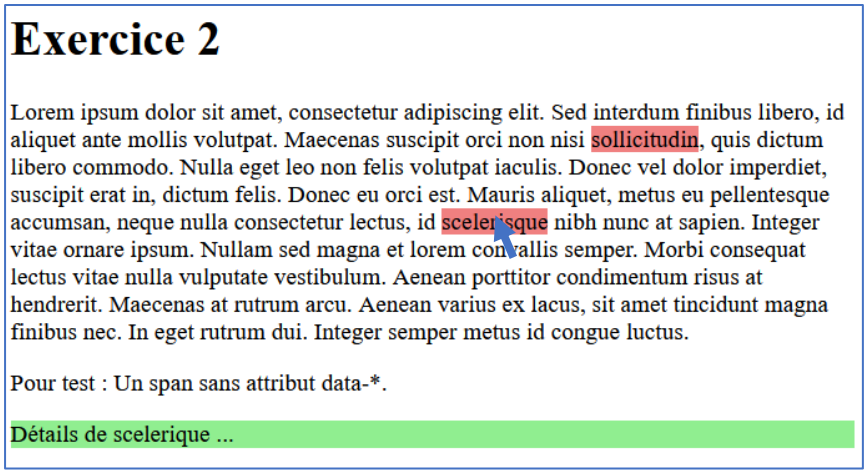


Exercice 2 – Attributs personnalisés et JS

Considérons le document exo2.html. Sans modifier le code HTML, complétez le code JS pour que pour tout élément `span` avec un attribut `data-desc`, sur réception de l'événement :

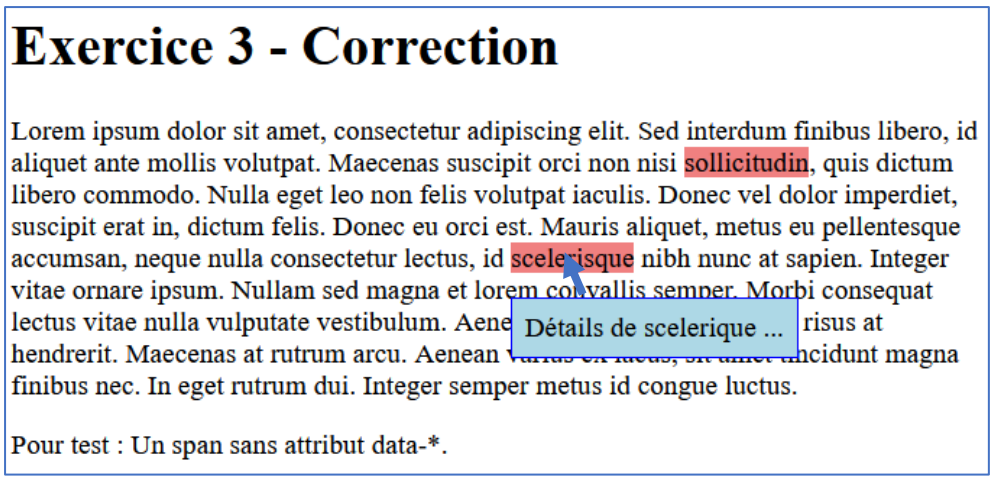
- `mouseover` : le contenu de la `div` d'`id` `desc` corresponde à la valeur de l'attribut `data-desc` du `span` (voir rendu ci-dessous)
- `mouseout` : le contenu de la `div` soit vidé.

Complétez le code CSS pour obtenir le rendu suivant, avec un pointeur pour le curseur de la souris sur `mouseover` pour les éléments `span` possédant un attribut `data-desc`.

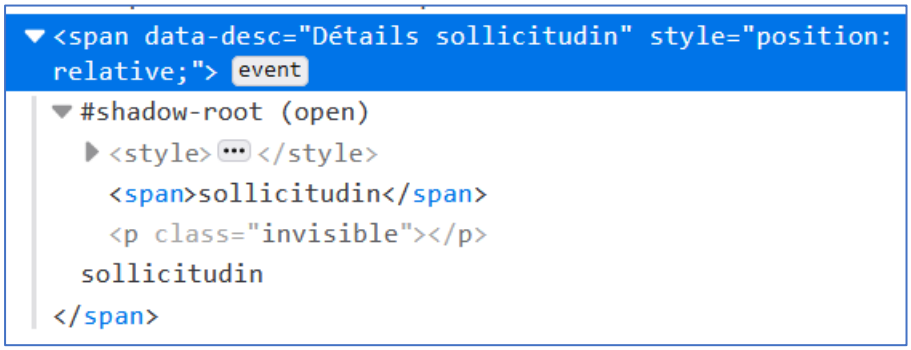


Exercice 3 – Attributs personnalisé, DOM caché

L'objectif est de modifier l'exercice 2 pour qu'une info-bulle apparaisse à côté du `span` avec la valeur de l'attribut `data-desc` au lieu de l'afficher plus bas dans la page comme contenu de la `div`, voir le rendu ci-dessous.



a) Pour définir l'info-bulle, vous devez attacher un arbre *shadow* à tout élément `span` avec attribut `data-desc`, pour obtenir les deux structures suivantes, exprimant les cas info-bulle non-rendue et rendue, respectivement :



```
▼ <span data-desc="Détails sollicitudin" style="position:
  relative;"> event
  ▼ #shadow-root (open)
    ▶ <style> ··· </style>
      <span>sollicitudin</span>
      <p class="">Détails sollicitudin</p>
    sollicitudin
  </span>
```

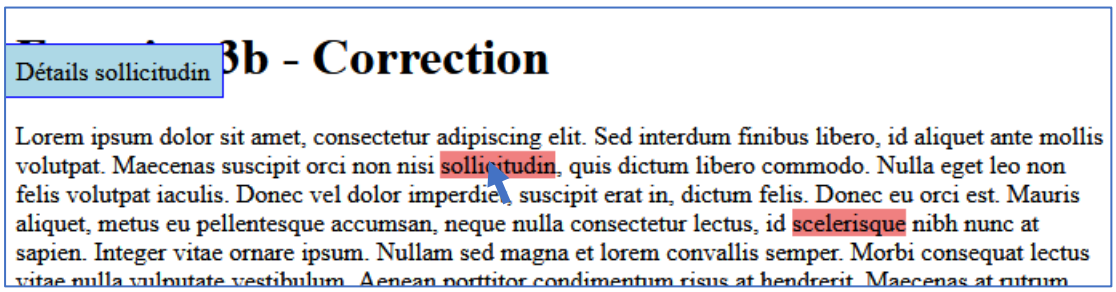
Détails :

- Un élément `span` décrit le contenu initial (puisqu'il n'est plus rendu mais accessible selon le principe du DOM caché).
- L'élément `p` représente l'info-bulle, avec une classe invisible pour préciser le cas non-rendu. Utilisez la méthode `toggle()` de la liste de classes pour passer d'un cas à l'autre. L'élément `style` (interne au *shadow* DOM) décrit le style de l'info-bulle. Utilisez la propriété `position` pour sortir l'élément `p` du flux. Le style de l'info-bulle sera donc cachée. Précisez celle de son parent (`span`) pour pouvoir positionner `p` relativement, avec un style en ligne pour plus d'importance (cascade), en JS. Notez que cela n'empêchera pas d'écraser le style de l'élément `p` en CSS puisqu'il n'est pas caché.
- Remarque : Le style de l'élément `span` caché hérite par défaut de celui du `span` hôte.

b) Le modèle d'info-bulle ci-dessus comporte deux inconvénients majeurs pour le style :

- Le style de l'hôte du DOM caché (l'élément `span`) peut être écrasé en CSS, en particulier sa position relative dont dépend celle de l'info-bulle (élément `p` caché). Ajoutez la règle CSS suivante dans le style interne principal et observez le problème dans le navigateur (voir rendu ci-dessous) :

```
span[data-desc] { position: static !important; }
```

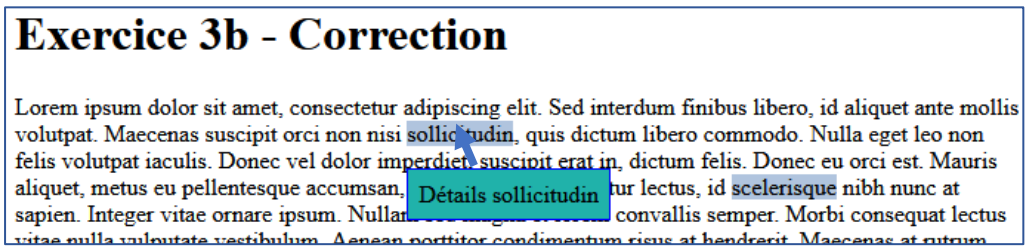


- Le style de l'info-bulle ne peut pas être modifiée via le CSS principal (non-caché), par exemple les couleurs.

Proposez une solution à ces problèmes, sans modifier la structure du DOM principal et du DOM caché :

- En complétant la position de l'élément `span` hôte (style en ligne en JS) avec la règle CSS `!important`, d'après l'importance des sélecteurs CSS, aucun autre sélecteur (style interne, externe, en ligne) ne pourra écraser la valeur de la propriété.
- En utilisant l'attribut `part` pour pouvoir styliser l'info-bulle (élément `p`) dans le CSS principal avec le pseudo-élément `::part()` (voir ci-dessous pour un rendu). Comme précédemment, les propriétés CSS importantes (`position`, `display`, `top`, `left`, …) ne devront pas être modifiables via le CSS principal. Ajoutez les règles `position: static !important;` et

```
display: inline-block !important;
```
au CSS interne principal pour l'info-bulle afin de le vérifier.

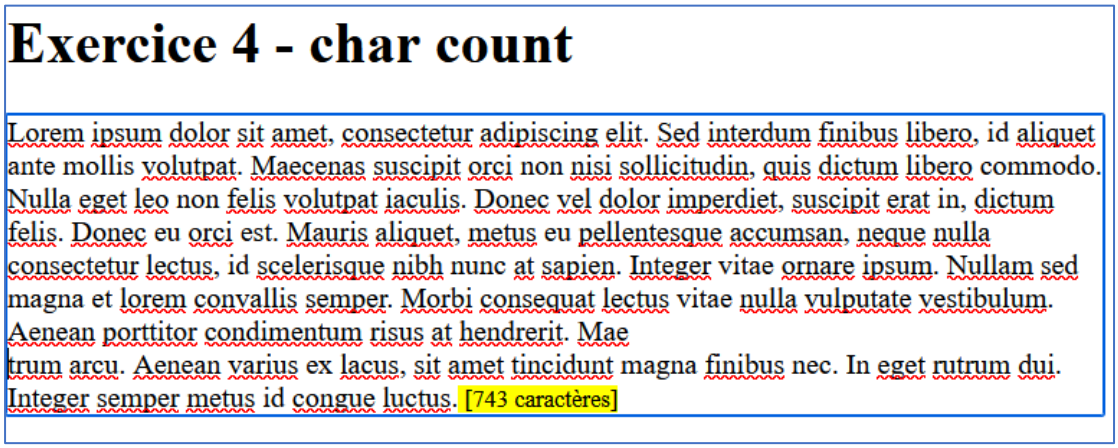


Exercice 4 – Eléments personnalisés intégrés

a) L'objectif consiste à créer un élément personnalisé intégré nommé `char-count`

- héritant de l'élément `p`
- dont le contenu est éditable, voir attribut `contenteditable`
- possédant un attribut personnalisé `data-nb-char` ayant pour valeur de nombre de caractères du paragraphe, mis à jour automatiquement en JS lorsque le contenu change (mise à jour du texte par l'utilisateur) capté via l'événement `input`
- le nombre de caractère sera généré en CSS après le paragraphe, avec le pseudo-élément `::after`

Le rendu est le suivant. Respectez le style (bordure du paragraphe, couleur de fond et taille de police du contenu généré avec le pseudo-élément).



b) Testez votre code, vous pouvez remarquer qu'il y a un problème lorsque tous les caractères sont supprimés, au moins avec Firefox, il reste un caractère qui est en fait un élément `br`. Pour simplifier, nous considérerons que tous les éléments br qui se suivent et qui sont situés en fin de paragraphe ne seront pas comptés pour le déterminer le nombre de caractères. Complétez votre code en conséquence. Vous devez pour cela programmer un algorithme parcourant le contenu du paragraphe depuis son dernier enfant, si cet enfant est un élément HTML `br`, alors il ne faut retirer un caractère (passage à la ligne) au compteur, puis passer à l'avant-dernier enfant et répéter l'opération, tant que l'enfant considéré en un élément `br`.

Exercice 5 – Eléments personnalisés autonomes

Recopiez l'exercice 4, vers exo5.html et modifiez ce dernier pour remplacer l'élément p par un élément autonome <`char-count`> ayant les mêmes fonctionnalités.

Exercice 6 – Liste déroulante

L'objectif est de transformer une liste non-ordonnée, avec sous-listes non-ordonnées, pour que toute sous-liste puisse être affichée/masquée lorsque l'utilisateur clique sur le contenu de l'élément de liste associé. Nous définissons pour cela un élément personnalisé intégré nommé `expanding-list`, comme dans l'exemple :
https://mdn.github.io/web-components-examples/expanding-list-web-component/
Par contre, contrairement à cet exemple, nous n'ajouterons pas de nouvel élément HTML en JS pour simplifier le processus de stylisation (un `span` est ajouté dans l'exemple pour simplifier le `click` sur le contenu d'un élément `li`).

En partant de cet exemple, modifiez-le pour avoir les même fonctionnalités mais sans ajouter de nouvel élément en JS. Vous allez devoir utiliser la pseudo-classe :scope pour sélectionner les enfants (et les descendants) d'un élément avec `querySelectorAll()` par exemple : https://developer.mozilla.org/fr/docs/Web/CSS/:scope
Notez également que l'on peut stopper la propagation d'un événement au moyen de la méthode `stopPropagation()` ou `stopImmediatePropagation()`, par exemple pour éviter que le `click` ne se propage sur les éléments de liste enfants :
https://developer.mozilla.org/en-US/docs/Web/API/Event/stopPropagation
https://developer.mozilla.org/en-US/docs/Web/API/Event/stopImmediatePropagation

Exercice 7 – Elements personnalisés et template

Dans exo7.html, reproduire le principe de l'exercice 4 en utilisant un modèle (élément `template`) pour définir la structure et le style de `char-count` :

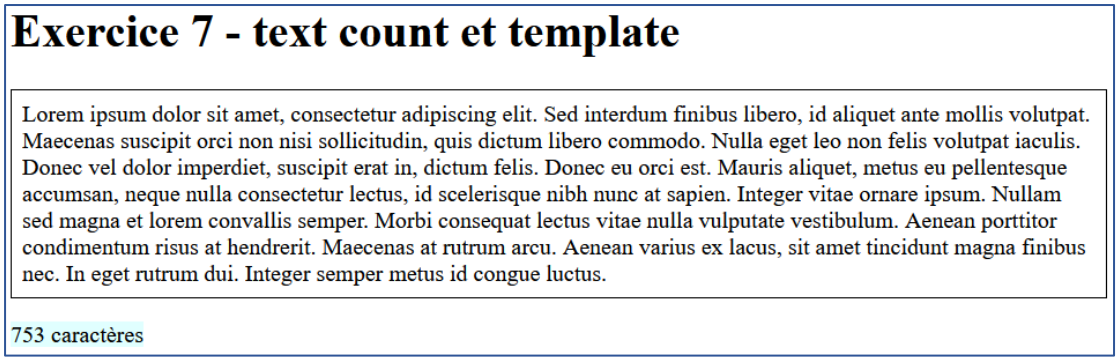
```
<template id="char-count">
   <style type="text/css">
      /* style par défaut cache pour p et output */
   </style>
   <p>Saisissez du texte</p><output>18 caractères</output>
</template>
```

Le paragraphe va contenir le texte pour lequel on compte le nombre de caractères, le nombre de caractère sera contenu dans l'élément `output`.

Voici comment récupérer le modèle en JS dans la classe `CharCount` et comment le
```
const template = document.getElementById("char-count");
const = self.attachShadow({mode: 'open'});
const frag = template.content.cloneNode(true);
shadow.appendChild(frag);
```

Il faut ensuite recopier le contenu du paragraphe éditable dans celui du DOM caché. Il sera donc en double (dans le DOM caché et dans le DOM principal).

Le rendu est légèrement différent de celui de l'exercice 4, voir ci-dessous, ainsi que la présentation du DOM caché. Le style ne doit pas être modifié dans le CSS principal.




```
<template id="char-count">
  #document-fragment
    <style type="text/css">…</style>
    <p>Saisissez du texte</p>
    <output>18 caractères</output>
  </template>
<char-count nb-char="753"> custom…
  #shadow-root (open)
    <style type="text/css">…</style>
    <p contenteditable="true"> event
      Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed interdum
      finibus libero, id aliquet ante mollis volutpat. Maecenas suscipit orci
      non nisi sollicitudin, quis dictum libero commodo. Nulla eget leo non
      felis volutpat iaculis. Donec vel dolor imperdiet, suscipit erat in,
      dictum felis. Donec eu orci est. Mauris aliquet, metus eu pellentesque
      accumsan, neque nulla consectetur lectus, id scelerisque nibh nunc at
      sapien. Integer vitae ornare ipsum. Nullam sed magna et lorem convallis
      semper. Morbi consequat lectus vitae nulla vulputate vestibulum. Aenean
      porttitor condimentum risus at hendrerit. Maecenas at rutrum arcu.
      Aenean varius ex lacus, sit amet tincidunt magna finibus nec. In eget
      rutrum dui. Integer semper metus id congue luctus.
    </p>
    <output>753 caractères</output>
  <p contenteditable="">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed interdum
```

Exercice 8 – Eléments personnalisés, templates, et slots

Pour éviter de recopier d'avoir 2 fois le même contenu (voir rendu ci-dessus), celui contenu dans le DOM principal, et celui recopié depuis ce DOM principal vers le DOM caché, on utilise le principe des slots :
https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_templates_and_slots#creating_a_template_with_some_slots
https://developer.mozilla.org/en-US/docs/Web/API/HTMLSlotElement

Cela va permettre de définir des slots dans le modèle, de le préciser dans le DOM principal, et cela sera remplacé automatiquement au moment où le modèle est appliqué. Les slots permettent aussi d'utiliser d'autres éléments que celui prévu dans le modèle. Dans le rendu ci-dessous on peut observer que nous utilisons un span plutôt qu'un p dans le DOM principal, et que dans le DOM caché, ce span n'est qu'un référence vers

le span du DOM principal.

Notez que le style ne doit toujours pas être modifié dans le CSS principal.

Notez également qu'en CSS, pour la partie cachée, on peut accéder à un élément qui a été « slotté » avec le pseudo-élément `::slotted()` :