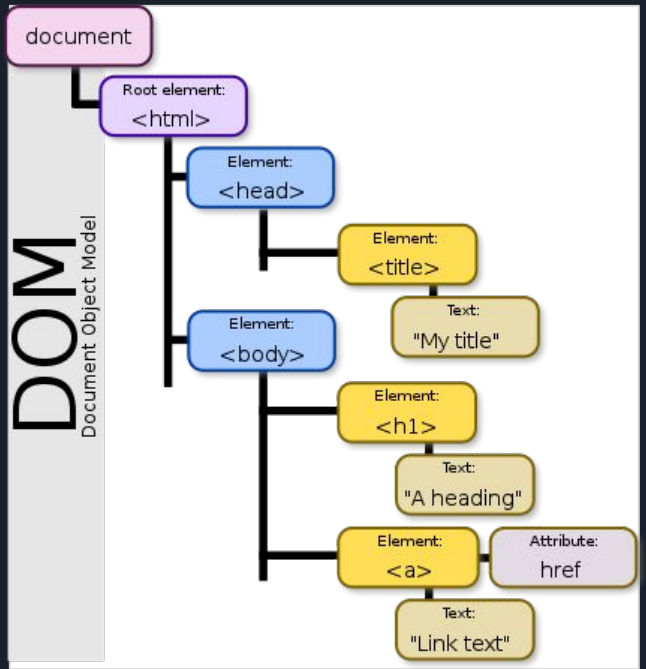
Javascript

Javascript et le DOM

Le DOM

- DOM: Document Object Model
- Le Document Object Model pour une page HTML, c'est un objet qui représente sous forme d'arbre la structure hiérarchique des balises HTML de la page au sein du navigateur



Le DOM

- Rappel de vocabulaire HTML:
  - Considérant deux éléments B et C contenus dans un élément A:
    - A est le parent de B et C
    - B et C sont des enfants de A
  - Considérant un élément C contenu dans un élément B, lui même contenu dans un élément A:
    - A est le parent de B, B est l'enfant de A
    - B est le parent de C, C est l'enfant de B
    - C est un descendant de A

```
<div id="A">
    <div id="B"></div>
    <div id="C"></div>
</div>
```

```
<div id="A">
    <div id="B">
        <div id="C"></div>
    </div>
</div>
```

Le DOM : accéder aux éléments

- L'objet `document` est un objet global accessible dans le code javascript exécuté par le navigateur. Il représente le DOM.
- Plusieurs méthodes de cet objet permettent de récupérer des éléments HTML de la page:
  - `document.getElementById(id)`
    - Prend en paramètre l'identifiant de l'élément à récupérer
    - Retourne l'élément si il est trouvé dans le document, null sinon
    - Exemple: `document.getElementById("mainTitle");`
    - Documentation
  - `document.getElementsByClassName(classNames)`
    - Prend en paramètre la/les classes des éléments à récupérer
    - Retourne une collection des éléments possédant la classe spécifiée
    - Exemple: `document.getElementsByClassName("redButton bigFont");`
    - Documentation

Le DOM : accéder aux éléments

- `document.getElementsByTagName(tagName)`
  - Prend en paramètre le type (p, div, span…) des éléments à récupérer
  - Retourne une collection des éléments du type spécifié
  - Exemple: `document.getElementsByTagName("p");`
  - Documentation
- `document.querySelector(selectors)`
  - Prend en paramètre des sélecteurs CSS
  - Retourne le premier élément correspondant ou null
  - Exemple: `document.querySelector("div.bigFont");`
  - Documentation
- `document.querySelectorAll(selectors)`
  - Prend en paramètre des sélecteurs CSS
  - Retourne la liste des éléments correspondant
  - Documentation

- Note: `getElementsByClassName`, `getElementsByTagName`, `querySelector`, et `querySelectorAll` peuvent être également appelées sur n'importe quel élément et effectueront la recherche sur les descendants de cet élément

Le DOM : Element

- Documentation
- Propriétés:
  - `className`: classe(s) de l'élément (lecture et écriture)
    - `element.className = "bigFont redButton";`
  - `style`: style de l'élément (lecture et écriture)
    - `element.style.color = "red";`
    - `element.style.backgroundColor = "blue";`
  - `classList`: liste des classe (lecture seule) (Documentation)
    - `element.classList.add("class1");`
    - `element.classList.remove("class1");`
  - `textContent`: contenu textuel (lecture et écriture)
    - `element.textContent = "My text content !";`

Le DOM : Element

- Méthodes:
  - `setAttribute`: modifier un attribut de l'élément
    - `element.setAttribute("type", "button");`
  - `getAttribute`: récupérer la valeur d'un attribut de l'élément
    - `element.getAttribute("name");`
  - `removeAttribute`: supprimer un attribut de l'élément
    - `element.removeAttribute("name");`

Le DOM : Créer un nouvel élément

1. Créer l'élément avec la méthode `document.createElement`:
   - Prend en paramètre le type d'élément à créer
   - Retourne le nouvel élément
   - Exemple: `document.createElement("input");`
   - Documentation
2. Modifier les attributs de l'élément avec les méthodes vues précédemment
3. Insérer l'élément dans le DOM:
   - `appendChild`: Peut être appelé par le corps de la page, ou n'importe quel autre élément, qui sera alors le parent du nouvel élément. Le nouvel élément sera le dernier enfant.
     - Prend en paramètre l'élément à insérer
     - Exemples:
       - `document.body.appendChild(newElement);`
       - `otherElement.appendChild(newElement);`
     - Documentation
   - `insertBefore` (Documentation)

Le DOM : Créer un nouvel élément, exemple

```
let form = document.createElement("form");

let inputName = document.createElement("input");
inputName.setAttribute('type', "text");

let inputSubmit = document.createElement("input");
inputSubmit.setAttribute('type', "button");
inputSubmit.setAttribute('value', "Envoyer");

form.appendChild(inputName);
form.appendChild(inputSubmit);

document.body.appendChild(form);
```

Le DOM : Autres fonctions

- Supprimer un élément:
  - `removeChild`
    - Prend en paramètre l'élément à supprimer
    - Retourne l'élément supprimé
    - Exemple: `element.removeChild(elementToRemove);`
    - Documentation
- Remplacer un élément:
  - `replaceChild`
    - Prend en premier paramètre le nouvel élément et en second l'élément à remplacer
    - Retourne l'élément remplacé
    - Exemple: `element.replaceChild(newElement, elementToReplace);`
    - Documentation

Le DOM : Les évènements

- "Écouter" un évènement sur un élément avec `addEventListener()`
  - Prend deux paramètres:
    - le nom de l'évènement à traiter, sous forme de chaîne de caractère
    - la fonction a exécuter lorsque cet évènement survient
  - Exemple: `element.addEventListener('click', event => { console.warn(event); });`
  - Documentation
  - Un même élément peut avoir plusieurs listeners pour un même évènement
  - Certaines éléments on déjà des listeners branchés sur un évènement (par exemple, les liens et les boutons des formulaires). On peut prévenir le comportement par défaut en utilisant `event.preventDefault()`

Exercices

- Lors de la suite d'exercices, vous veillerez à écrire du code qui fonctionne, mais également à le rendre le plus élégant possible, vous veillerez donc à:
  - Nommer vos variables avec des noms qui ont du sens
  - Découper éventuellement le code en fonctions (dont le nom aura également du sens)
- Comment identifier du code qui aurait besoin d'être placé dans une fonction ?
  - Vous voyez un ensemble d'instructions qui est répété à plusieurs endroits dans votre programme
  - Vous avez une fonction qui commence à être longue (critère un peu subjectif) et peu lisible, vous pouvez donc peut être découper son traitement en plusieurs sous-fonctions

Exercice 1 : Agir sur des éléments existants

1. Dans votre répertoire de travail, créez un fichier ex1.html avec le contenu suivant:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Hello !</title>
        <link rel="stylesheet" type="text/css" href="ex1.css">
    </head>
    <body>
        <h1> Hello ! </h1>
        <button id="button" >Click me</button>
        <img id="myImg" src="ex1.jpg" >
        <script src="ex1.js"></script>
    </body>
</html>
```

Exercice 1 : Agir sur des éléments existants

2. Créez un fichier ex1.css avec le contenu suivant:

```
#myImg {
    visibility: hidden;
}
```

3. Ajoutez une image de votre choix, au format .jpg, et nommez-la ex1.jpg
4. Créez un fichier ex1.js
5. Ouvrez le fichier ex1.html dans votre navigateur. L'image ne doit pas apparaître.
6. Modifiez le fichier ex1.js pour que, lorsque vous cliquez sur le bouton, l'image apparaisse si elle est invisible et disparaisse si elle est visible.

- Aide:
  - Vous aurez besoin de lier l'évènement "click" au bouton
  - Vous aurez besoin de jouer sur la propriété "style" de l'élément img
  - L'attribut css "visibility" peut prendre deux valeurs: "hidden" ou "visible"

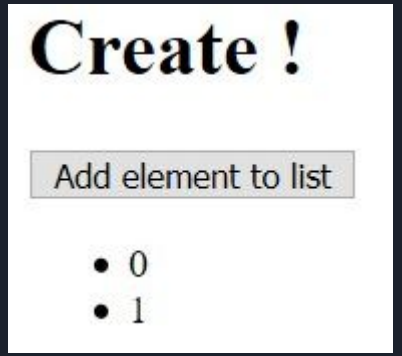
Exercice 2 : Créer des éléments

1. Dans votre répertoire de travail, créez un fichier ex2.html avec le contenu suivant:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Create !</title>
    </head>
    <body>
        <h1>Create !</h1>
        <button id="button" >Add element to list</button>
        <ul id="list">
        </ul>
        <script src="ex2.js"></script>
    </body>
</html>
```

Exercice 2 : Créer des éléments

2. Dans votre répertoire de travail, créez un fichier ex2.js
3. Lorsque vous ouvrez votre page ex2.html dans votre navigateur, faites en sorte que, lorsque vous cliquez sur le bouton, un nouvel élément soit ajouté à la liste. Le contenu sera défini par un compteur:



- Aide:
  - Vous aurez besoin de créer des éléments (`document.createElement`)
  - Vous aurez besoin d'ajouter des enfants à des éléments (`element.appendChild`)
  - Vous aurez besoin de modifier le texte des éléments (`element.textContent`)

Exercice 3.1 : Créer une liste

1. Dans votre répertoire de travail, créez un fichier ex3.html avec le contenu suivant:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>User List</title>
    </head>
    <body>
        <h1>User List</h1>
        <script src="ex3.js"></script>
    </body>
</html>
```

Exercice 3.1 : Créer une liste

2. Dans votre répertoire de travail, créez un fichier ex3.js avec le contenu suivant:

```
const userlist = [
    { id: 1, prenom: "Damien", age: 40, role: "utilisateur" },
    { id: 2, prenom: "Camille", age: 29, role: "administrateur" },
    { id: 3, prenom: "Marie", age: 35, role: "utilisateur" },
    { id: 4, prenom: "Roger", age: 60, role: "utilisateur" },
];
```

3. Lorsque vous ouvrez votre page ex3.html dans votre navigateur, faites en sorte d'obtenir le résultat suivant:

Exercice 3.2 : Affichage conditionnel

- Faites en sorte d'obtenir le résultat suivant:

- Instructions:
  - Si le rôle d'un utilisateur est "administrateur", il s'affiche en rouge, sinon en bleu

Exercice 3.3 : Autres évènements

- Faites en sorte d'obtenir le résultat suivant:

- Instructions:
  - Lorsque la souris passe sur une ligne, la couleur de fond de la ligne passe au gris
  - Lorsque la souris sort d'une ligne, la couleur de fond de la ligne repasse au blanc
- Aide:
  - Vous aurez besoin des évènements "mouseenter" et "mouseleave"

Exercice 3.4 : Action supprimer

- Faites en sorte d'obtenir le résultat suivant:

- Instructions:
  - Lorsque vous cliquez sur le bouton "Supprimer" d'un utilisateur, il doit disparaître de la liste
- Aide:
  - Vous aurez besoin de la fonction `removeChild`
  - L'élément HTML qui représente un utilisateur doit avoir un identifiant qui permet de le récupérer pour ensuite le supprimer quand on clique sur son bouton "Supprimer"

Exercice 3.5 : Formulaire d'ajout

- Ajoutez le formulaire suivant avant la liste des utilisateurs (vous pouvez le faire en HTML):

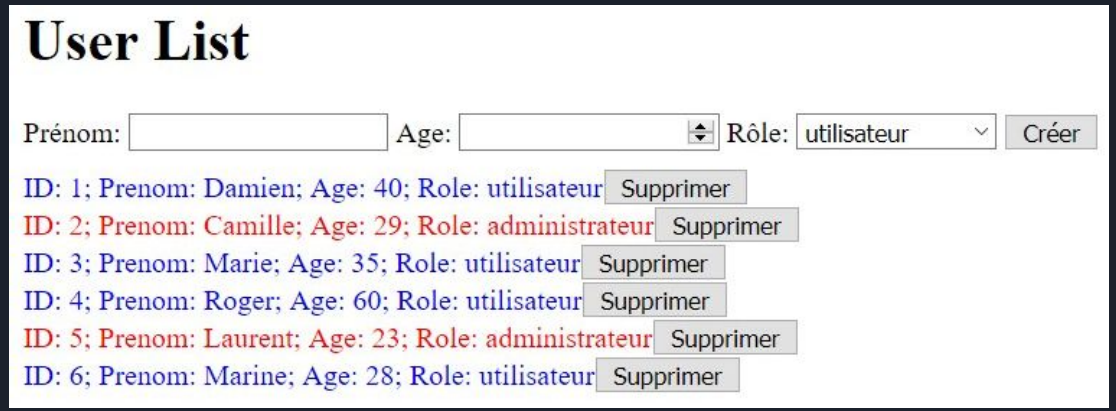
- Instructions et contraintes:
  - Lorsque vous cliquez sur le bouton "Créer":
    - le nouvel utilisateur doit apparaître dans la liste
    - les champs du formulaire seront remis à zéro
  - Notez qu'il n'y a pas de champ pour l'identifiant. L'identifiant sera généré automatiquement à la création de l'utilisateur en utilisant l'identifiant le plus élevé des utilisateurs déjà existant + 1

(aide sur la slide suivante)

Exercice 3.5 : Formulaire d'ajout

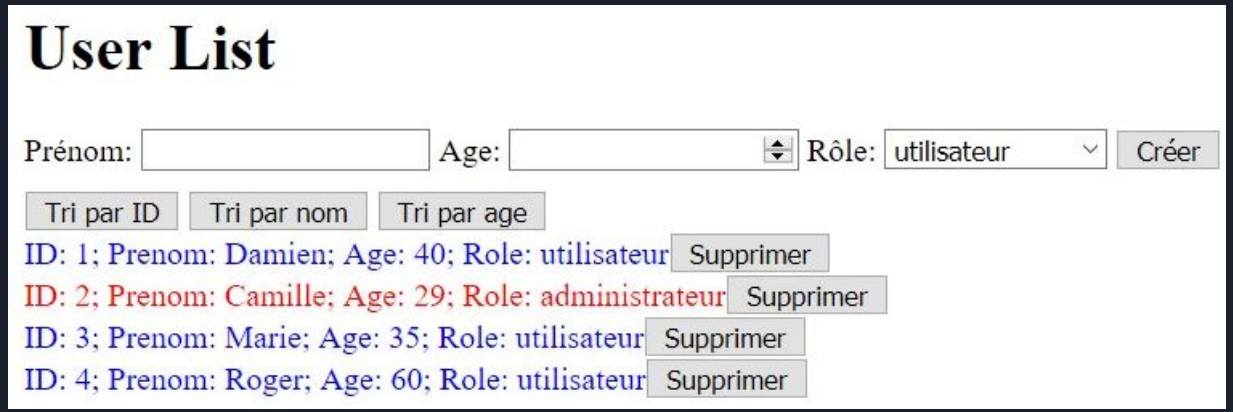
- Aide:
  - Pour accéder au contenu d'un élément input ou select dans le javascript, vous pouvez utiliser l'attribut `value` de l'élément une fois que vous l'avez récupéré
  - Si vous utilisez un élément <form>, vous aurez besoin d'utiliser l'instruction `event.preventDefault();` dans la fonction qui traite l'évènement de clic du bouton "Créer"

Exemple d'état suite à l'ajout de deux utilisateurs:



Exercice 3.6 : Fonctions de tri

- Ajoutez 3 boutons avant la liste des utilisateurs (vous pouvez le faire en HTML):



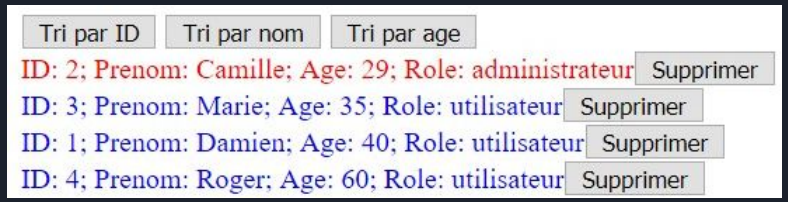
- Instructions:
  - Lorsque vous cliquez sur le bouton "Tri par ID", les utilisateurs sont triés par ordre d'ID croissant
  - Lorsque vous cliquez sur le bouton "Tri par nom", les utilisateurs sont triés par ordre alphabétique du prénom
  - Lorsque vous cliquez sur le bouton "Tri par age", les utilisateurs sont triés par ordre d'âge croissant

(aide sur la slide suivante)

Exercice 3.6 : Fonctions de tri

- Aide:
  - Pour trier la liste des utilisateurs:
    - Par ID croissant: `userlist.sort((a,b) => a.id - b.id);`
    - Par prénom: `userlist.sort((a,b) => a.prenom.localeCompare(b.prenom));`
    - Par âge croissant: `userlist.sort((a,b) => a.age - b.age);`
  - Conseil: si ce n'est pas déjà fait, utilisez un élément qui contient tous les utilisateurs
  - Pour supprimer tout le contenu (texte et éléments) d'un élément, utilisez `votreElement.innerHTML = "";`

Affichage après un clic sur "Tri par nom"



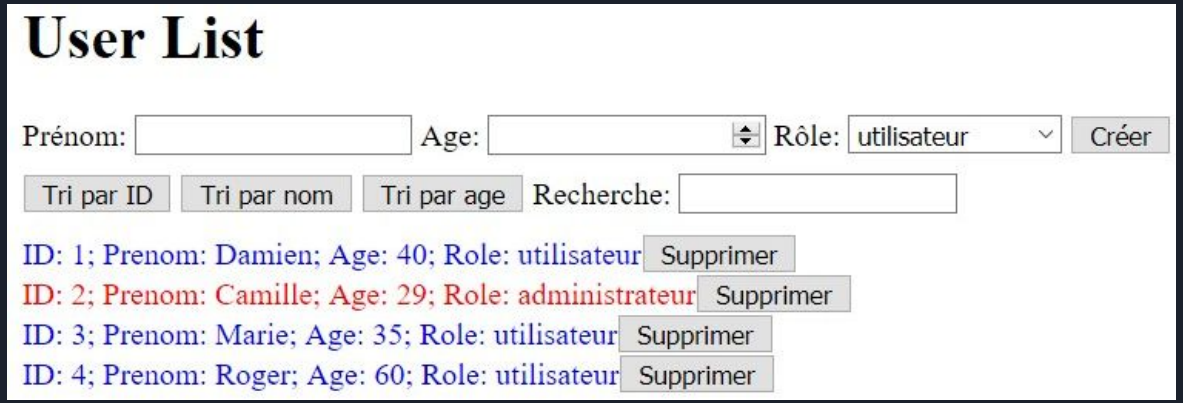
Affichage après un clic sur "Tri par age"

Exercice 3.6 : Fonctions de tri, bonus

- Si vous avez de l’avance sur le reste du groupe, faites en sorte que:
  - Si vous cliquez sur "Tri par age" alors que la liste est déjà triée par âge croissant, alors faite le tri par ordre d’âge décroissant
  - Si vous cliquez sur "Tri par ID" alors que la liste est déjà triée par ID croissant, alors faite le tri par ordre d’ID décroissant
  - Si vous cliquez sur "Tri par nom" alors que la liste est déjà triée par ordre alphanumérique croissant, alors faite le tri par ordre alphanumérique décroissant

Exercice 3.7 : Fonction de recherche

- Ajoutez un champ de recherche avant la liste des utilisateurs (vous pouvez le faire en HTML):



- Instructions:
  - Lorsque le contenu du champ de recherche change, une recherche est effectuée pour chercher une correspondance avec l'ID, le prénom, l'âge ou le rôle

(aide sur la slide suivante)

Exercice 3.7 : Fonction de recherche

- Aide:
  - Vous aurez besoin de traiter l'évènement "input" du champ de recherche
  - Pour effectuer une recherche sur la liste des utilisateurs en fonction d'une variable `searchValue`:

```
let newUserList = userlist.filter(
        user => user.age == searchValue
        || user.prenom.includes(searchValue)
        || user.role.includes(searchValue)
        || user.id == searchValue
        );
```

Affichage après une recherche avec "mi"

Affichage après une recherche avec "60"

Exercice 3.8 : Persistance des données

- En utilisant le localStorage du navigateur, faites en sorte de sauvegarder la liste des utilisateurs pour qu'elle ne soit pas remise à l'état initial à chaque rafraichissement de la page

Aide:

- Vous pouvez utiliser JSON.stringify pour transformer un objet javascript en chaine de caractère (ici, cela peut vous permettre de transformer la userlist sous forme de chaîne de caractère pour pouvoir la stocker dans le localStorage)
- Vous pouvez utiliser JSON.parse pour transformer une chaine de caractère en objet javascript (ici, cela peut vous permettre de récupérer la userlist stockée sous forme de chaîne de caractère dans le localStorage pour en faire un tableau javascript)

Exercice 3.9 : Fonction d’édition

- Proposez une fonctionnalité permettant d’éditer (modifier) un utilisateur (son ID ne doit pas être modifiable)