Exercices : standards Web Components

version

Ce document n'est qu'un premier brouillon en attente d'une relecture complète et de tests en situation réelle. Merci de me signaler toute erreur, incohérence ou manque de clarté.

attention

Les "code de base" donnés ne sont là que pour offrir un point de départ. Vous pourrez parfois y retrouver de mauvaises pratiques et raccourcis. Assurez-vous dès le départ d'identifier de tels problèmes et de les corriger.

Exercice 1 : rappels HTML

Sujet	Revoir les standards de base en lien avec les Web Components.
Objectif	Comprendre l'approche générale "composants" de la plateforme web et disposer d'un environnement de développement familier pour la suite.

Instructions

1. Installez :
   - un IDE ou éditeur web de votre choix (recommandation : VS Code)
   - au moins 2 navigateurs evergreen à jour et utilisant deux moteurs différents
   - Node.js 18 ou supérieur (recommandation : utilisez NVM)
2. Créez un fichier HTML simple.
3. Identifiez 3 éléments HTML standards (ou en voie d'être standardisés) que vous ne connaissiez pas, ou que partiellement.
4. Intégrez ces 3 éléments dans la page HTML, avec au moins une personnalisation via des attributs.
5. (*bonus*) ajoutez un comportement dynamique en JavaScript

   Prenez le temps d'explorer MDN et le standard HTML.

Exercice 2 : Manipuler le DOM et charger des données dynamiques

Sujet	Rappels DOM & AJAX
Objectif	Charger des données depuis une API et les afficher dans le DOM.

Instructions

1. Récupérez les données d'une API publique
   - exemple : https://jsonplaceholder.typicode.com/users)
2. Affichez dynamiquement la liste d'éléments récupérés (e.g. utilisateurs avec leurs noms et emails) sous forme de tableau ou de liste.
3. (*bonus*) Ajoutez une gestion d'erreur (vous pouvez par exemple tester un échec de transaction en passant en mode déconnecté)

Code de base

```
fetch('https://jsonplaceholder.typicode.com/users')
  .then((response) => response.json())
  .then((users) => {
    // Insérez le code pour générer un tableau ici
  });
```

Exercice 3 : Utiliser un template HTML

Sujet	Templates HTML
Objectif	Charger dynamiquement le contenu initial d'un custom element

Instructions

1. Créez un template HTML contenant une carte utilisateur avec un nom et une description.
2. Utilisez JavaScript pour cloner ce template et l'insérer dans la page avec des données différentes pour chaque utilisateur (au moins 3).

Code de base

```
<template id="user-card">
  <div class="card">
    <h3>Nom :</h3>
    <p>Description :</p>
  </div>
</template>
```

Exercice 4 : Isoler les styles avec le Shadow DOM

Sujet	Shadow DOM et CSS
Objectif	Appliquer des styles à un custom element sans impacter son contexte.

Instructions

1. Créez un Custom Element simple avec le contenu de votre choix
2. Créez un shadow DOM attaché à cet élément et stylisez vos éléments.
3. (*bonus*) Ajoutez un bouton permettant de changer un style quand cliqué.

Code de base

```
class UserCard extends HTMLElement {
  constructor() {
    super();
    const shadow = this.attachShadow({ mode: 'open' });
    shadow.innerHTML = `
            <style>
                :host {
                    display: block;
                    background-color: blue;
                    color: white;
                    border: none;
                    padding: 10px;
                }
                h3 {}
                p {}
            </style>
            <h3>Nom : </h3>
            <p>Description : </p>
```

```
        `;
  }
}
customElements.define('styled-button', StyledButton);
```

Exercice 5 : Créer un composant avec des attributs personnalisés

Sujet	Custom Elements & attributs HTML
Objectif	Apprendre à passer des données via des attributs.

Instructions

1. Reprenez votre Custom Element `<user-card>` et donnez lui 3 attributs : `name`, `email` et `role`.
2. Affichez dynamiquement les valeurs associées à ces 3 attributs dans le contenu du custom element.

Code de base

```
class UserCard extends HTMLElement {
  connectedCallback() {
    const name = this.getAttribute('name') || 'Nom inconnu';
    const email = this.getAttribute('email') || 'Email inconnu';
    const role = this.getAttribute('role') || 'Rôle non défini';

    this.innerHTML = `
            <div class="card">
                <h3>${name}</h3>
                <p>Email : ${email}</p>
                <p>Rôle : ${role}</p>
            </div>
        `;
  }
}
customElements.define('user-card', UserCard);
```

Exercice 6 : Apportez de la flexibilité à un template avec des slots

Sujet	Slots et contenu dynamique

Objectif	Maitrisez les slots pour permettre de dynamiser le contenu des web components.

Instructions

1. Reprenez le composant `<user-card>` et permettez de définir tout le contenu de la “card” via un slot.
2. Utilisez des slots nommés pour définir et afficher séparément chaque élément de la carte (`name`,`email`,`role`)
3. Déplacez ces slots nommés et leurs styles vers un template.
4. Définissez un nouveau composant `<user-details>` utilisant ce template.
5. Utilisez le composant `<user-details>` dans `<user-card>` afin que ce dernier ne gère plus que la logique d’affichage des 3 données passées en attribut et les styles associés.
6. (*bonus*) Jouez avec les shadow DOM de vos 2 élements et les styles des slots jusqu’à pouvoir décrire en détail les logiques sous-jascentes.

Exercice 7 (*bonus*) : Intégrer un Web Component dans un framework

Sujet	Compatibilités entre les standards Weh Component et les principales bibliothèques / frameworks web.
Objectif	Identifiez les limites, pré-requis et bonnes pratiques associés à l’utilisation de Web Components avec les principales bibliothèques / frameworks web.

Instructions

1. créez un projet Next.js via la commande suivante : `npx create-next-app@latest`
2. créer un client component faisant appel à `<user-card>` et affichez le

3. intégrez directement `<user-card>` dans un React Server Component
4. (*bonus*) reproduire les mêmes opérations avec Angular et Vue.

Code de base

```
function App() {
  useEffect(() => {
    const card = document.querySelector('user-card');
    card.setAttribute('name', 'Alice');
    card.setAttribute('img', 'alice.jpg');
    card.setAttribute('description', 'Ingénieure Logiciel');
  }, []);

  return <user-card />;
}
```