Corrections : Exercices Web Components

> 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.

Correction Exercice 1 : Rappels HTML

Objectif

Revoir les standards HTML liés aux Web Components et explorer des éléments méconnus ou récents.

Solution

1. **Créer un fichier HTML** avec des éléments récents ou peu connus :

```html
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Exercice 1 : Rappels HTML</title>
  </head>
  <body>
    <!-- Utilisation de nouveaux éléments HTML -->
    <dialog open>Bienvenue dans un composant <code>dialog</code> !</dialog>
    <details>
      <summary>Résumé interactif</summary>
      Contenu caché révélé lorsque vous cliquez.
    </details>
    <progress value="50" max="100">50%</progress>
  </body>
</html>
```

2. **Explication des balises** :
   - `<dialog>` : Crée une boîte de dialogue native, affichée ou masquée.
   - `<details>` : Permet d'afficher un contenu caché après interaction.
   - `<progress>` : Barre de progression affichant un pourcentage.
3. **Ajout d'un comportement dynamique** avec JavaScript :

```javascript
const progress = document.querySelector('progress');
let value = 50;

setInterval(() => {
  value = (value + 10) % 100;
  progress.value = value;
}, 1000);
```

Correction Exercice 2 : Manipuler le DOM et charger des données dynamiques

Solution complète

Afficher une liste d'utilisateurs en récupérant les données via une API.

1. **HTML de base** :

```html
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Exercice 2 : DOM et AJAX</title>
  </head>
  <body>
    <ul id="user-list"></ul>
    <script src="app.js"></script>
  </body>
</html>
```

2. **JavaScript pour charger les données** :

```javascript
const userList = document.getElementById('user-list');

fetch('https://jsonplaceholder.typicode.com/users')
  .then((response) => {
    if (!response.ok) throw new Error('Erreur de chargement');
    return response.json();
  })
  .then((users) => {
    users.forEach((user) => {
      const li = document.createElement('li');
      li.textContent = `${user.name} - ${user.email}`;
      userList.appendChild(li);
    });
  })
  .catch((error) => console.error('Erreur :', error));
```

Correction Exercice 3 : Utiliser un template HTML

Solution

1. **HTML avec un template** :

```html
<template id="user-card-template">
  <div class="card">
    <h3></h3>
    <p></p>
  </div>
</template>
<div id="cards-container"></div>
```

2. **JavaScript pour cloner et insérer des templates** :

```javascript
const template = document.getElementById('user-card-template');
const container = document.getElementById('cards-container');

const users = [
  { name: 'Alice', description: 'Développeuse' },
  { name: 'Bob', description: 'Designer' },
  { name: 'Charlie', description: 'Chef de projet' },
];

users.forEach((user) => {
  const clone = template.content.cloneNode(true);
  clone.querySelector('h3').textContent = user.name;
  clone.querySelector('p').textContent = user.description;
  container.appendChild(clone);
});
```

Correction Exercice 4 : Isoler les styles avec le Shadow DOM

Solution

1. **Custom Element avec styles encapsulés** :

```javascript
class StyledCard extends HTMLElement {
  constructor() {
    super();
    const shadow = this.attachShadow({ mode: 'open' });
    shadow.innerHTML = `
            <style>
                .card {
                    border: 1px solid #ddd;
```

```
                    border-radius: 5px;
                    padding: 10px;
                    background-color: #f9f9f9;
                }
            </style>
            <div class="card">
                <h3>Carte stylée</h3>
                <p>Contenu encapsulé.</p>
            </div>
        `;
    }
}

customElements.define('styled-card', StyledCard);
```

2. **HTML pour utiliser le composant** :

```
<styled-card></styled-card>
```

Correction Exercice 5 : Créer un composant avec des attributs personnalisés

Solution

1. **JavaScript pour les attributs dynamiques** :

```
class UserCard extends HTMLElement {
  static get observedAttributes() {
    return ['name', 'email', 'role'];
  }

  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
  }

  connectedCallback() {
    this.update();
  }

  attributeChangedCallback() {
    this.update();
  }

  update() {
    const name = this.getAttribute('name') || 'Nom inconnu';
    const email = this.getAttribute('email') || 'Email inconnu';
```

```
        const role = this.getAttribute('role') || 'Rôle non défini';

        this.shadowRoot.innerHTML = `
                <div class="card">
                    <h3>${name}</h3>
                    <p>Email : ${email}</p>
                    <p>Rôle : ${role}</p>
                </div>
            `;
      }
    }

    customElements.define('user-card', UserCard);
```

2. **HTML avec un exemple** :

```
<user-card name="Alice" email="alice@example.com" role="Développeuse"></user-car
```

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

Solution

1. **Créer un composant `<user-details>` avec des slots nommés :**

```
class UserDetails extends HTMLElement {https://stenciljs.com/docs/angular
    super();
    const shadow = this.attachShadow({ mode: 'open' });
    shadow.innerHTML = `
            <style>
                .details {
                    font-family: Arial, sans-serif;
                    border: 1px solid #ddd;
                    padding: 10px;
                    border-radius: 5px;
                }
                ::slotted(h3) {
                    color: #333;
                    margin: 0;
                }
                ::slotted(p) {
                    color: #666;
                }
            </style>
            <div class="details">
                <slot name="name"></slot>
```

```
                <slot name="email"></slot>
                <slot name="role"></slot>
            </div>
        `;
  }
}

customElements.define('user-details', UserDetails);
```

2. **Utiliser ce composant dans `<user-card>` :**

```
class UserCard extends HTMLElement {
connectedCallback() {
    this.innerHTML = `
            <user-details>
                <h3 slot="name">${this.getAttribute('name') || 'Nom inconnu'}</h
                <p slot="email">${this.getAttribute('email') || 'Email inconnu'}
                <p slot="role">${this.getAttribute('role') || 'Rôle non défini'}
            </user-details>
        `;
}
}

customElements.define('user-card', UserCard);
```

3. **HTML pour tester :**

```
<user-card name="Alice" email="alice@example.com" role="Développeuse"></user-car
```