Exercices JavaScript

Calculer la somme de deux nombres:

- **Exercice:** Demander à l'utilisateur de saisir deux nombres et afficher leur somme.

```
let num1 = parseInt(prompt("Entrez le premier nombre :"));
let num2 = parseInt(prompt("Entrez le deuxième nombre :"));
let somme = num1 + num2;
alert("La somme est : " + somme);
```

Tableau et boucle:

- **Exercice:** Créer un tableau de fruits et afficher chaque fruit dans une liste à puces.

```
let fruits = ["Pomme", "Banane", "Orange"];
let liste = "<ul>";
for (let i = 0; i < fruits.length; i++) {
   liste += "<li>" + fruits[i] + "</li>";
}
liste += "</ul>";
document.write(liste);
```

Fonction pour vérifier si un nombre est pair:

- **Exercice:** Écrire une fonction qui prend un nombre en entrée et retourne true si le nombre est pair, false sinon.

```
function estPair(nombre) {
   return nombre % 2 === 0;
}
let nombre = 10;
if (estPair(nombre)) {
   console.log(nombre + " est pair");
} else {
   console.log(nombre + " est impair");
}
```

Objet et méthode:

- **Exercice:** Créer un objet représentant une personne avec les propriétés nom, âge et une méthode pour afficher ses informations.

```
let personne = {
   nom: "John Doe",
   age: 30,
   afficherInfos: function() {
      console.log("Nom: " + this.nom);
      console.log("Âge: " + this.age);
   }
};
personne.afficherInfos();
```

Asynchrone avec Promise:

- **Exercice:** Simuler une requête API avec une Promise qui résout après 2 secondes et affiche un message de succès.

```
function fetchData() {
   return new Promise((resolve, reject) => {
      setTimeout(() => {
         resolve("Données récupérées avec succès !");
      }, 2000);
   });
}

fetchData()
```

```
.then(data => {
  console.log(data);
})
.catch(error => {
  console.error(error);
});
```

Manipulation du DOM : Création d'éléments dynamiques

- **Exercice:** Créer un formulaire permettant à l'utilisateur de saisir un texte. À chaque saisie, ajouter le texte saisi dans une liste non ordonnée à la page.

```
const input = document.getElementById('myInput');
const list = document.getElementById('myList');

input.addEventListener('input', () => {
  const newListItem = document.createElement('li');
  newListItem.textContent = input.value;
  list.appendChild(newListItem);
  input.value = ''; // Effacer le contenu de l'input
});
```

Fonction récursive : Calcul de la factorielle

- **Exercice:** Écrire une fonction récursive pour calculer la factorielle d'un nombre.

```
function factorielle(n) {
  if (n === 0) {
    return 1;
  } else {
    return n * factorielle(n - 1);
  }
}
console.log(factorielle(5));
```

Création d'un objet "Livre"

- **Exercice:** Créer un constructeur pour créer des objets "Livre" avec les propriétés titre, auteur et année de publication. Ajouter une méthode pour afficher les informations du livre.

```
function Livre(titre, auteur, annee) {
  this.titre = titre;
  this.auteur = auteur;
  this.annee = annee;
  this.afficherInfos = function() {
    console.log(`Titre: ${this.titre}, Auteur: ${this.auteur}, Année: ${this.annee}`);
  };
}
const monLivre = new Livre("Le Petit Prince", "Antoine de Saint-Exupéry", 1943);
monLivre.afficherInfos();
```

Asynchrone avec async/await : Requête API

- **Exercice:** Utiliser async/await pour faire une requête à une API (par exemple, l'API de l'heure actuelle) et afficher le résultat dans un élément HTML.

```
async function fetchTime() {
  const response = await fetch('https://api.timezonedb.com/v2/get-time-
zone?key=YOUR_API_KEY&format=json&by=zone&zone=Europe/Paris');
  const data = await response.json();
  document.getElementById('currentTime').textContent = data.formatted;
}
fetchTime();
```

Programmation orientée objet : Héritage

- **Exercice:** Créer une classe Animal avec les propriétés nom et age. Créer ensuite des classes Chat et Chien héritant de Animal et ajoutant des propriétés spécifiques (par exemple, race pour le chien).

```
class Animal {
```

```
    constructor(nom, age) {
      this.nom = nom;
      this.age = age;
    }
}

class Chien extends Animal {
    constructor(nom, age, race) {
      super(nom, age);
      this.race = race;
    }
}

const monChien = new Chien("Médor", 3, "Berger Allemand");
console.log(monChien);
```

Créer une fonction qui mémorise ses appels

- **Exercice:** Créer une fonction qui, à chaque appel, ajoute 1 à une valeur interne et retourne cette valeur.

```
function compteur() {
    let count = 0;
    return function() {
      return count++;
    };
}

const monCompteur = compteur();
console.log(monCompteur());
console.log(monCompteur()); // Affiche 1
```

Créer un compteur incrémental utilisant des objets et des closures (version ES6+)

```
function createCounter() {
  let count = 0;

  return {
    increment() {
      return ++count;
    }
  };
}

const counter1 = createCounter();
const counter2 = createCounter();

console.log(counter1.increment()); // Affiche 1
console.log(counter1.increment()); // Affiche 2
console.log(counter2.increment()); // Affiche 1 (compteur indépendant)
```

Exemple avec une classe :

```
class Counter {
  constructor() {
    this.count = 0;
  }

  increment() {
    return ++this.count;
  }
```

```
}

const counter = new Counter();
console.log(counter.increment());
```

Exemple 1 : Créer une classe Personne

```
class Personne {
  constructor(nom, age) {
    this.nom = nom;
    this.age = age;
  }

  saluer() {
    console.log(`Bonjour, je m'appelle ${this.nom} et j'ai ${this.age}
 ans.`);
  }
}

// Créer une instance de la classe
const personne1 = new Personne("Alice", 30);
personne1.saluer(); // Affiche : Bonjour, je m'appelle Alice et j'ai 30 ans.
```

Héritage et polymorphisme avec des classes Animal et Chat

```
class Animal {
  constructor(nom) {
    this.nom = nom;
  }

  faireDuBruit() {
    console.log("L'animal fait du bruit.");
  }
}

class Chat extends Animal {
  faireDuBruit() {
    console.log("Miaou !");
  }
}
const chat = new Chat("Félix");
chat.faireDuBruit(); // Affiche : Miaou !
```

Créer une classe CompteBancaire avec des getters et setters

```
class CompteBancaire {
  constructor(titulaire, solde) {
    this.titulaire = titulaire;
    this.solde = solde;
  }

  get solde() {
    return this._solde;
  }

  set solde(nouveauSolde) {
    if (nouveauSolde < 0) {
```

```
      console.error("Le solde ne peut pas être négatif.");
    } else {
      this._solde = nouveauSolde;
    }
  }
  retirer(montant) {
    if (montant <= this.solde) {
      this.solde -= montant;
      console.log(`Retrait de ${montant}€ effectué. Nouveau solde : ${this.solde}€`);
    } else {
      console.error("Solde insuffisant.");
    }
  }
}
const compte = new CompteBancaire("Bob", 1000);
compte.retirer(200); // Retrait de 200€ effectué. Nouveau solde : 800€
compte.solde = -50; // Affiche un message d'erreur car le solde ne peut pas être négatif
```

Créer une classe CompteBancaire avec des getters et setters

```
class Rectangle {
  constructor(longueur, largeur) {
    this.longueur = longueur;
    this.largeur = largeur;
  }

  calculerSurface() {
    return this.longueur * this.largeur; [1]

  }
}
// Création d'un rectangle
const monRectangle = new Rectangle(5, 3);

// Calcul de la surface
const surface = monRectangle.calculerSurface();

// Affichage du résultat
console.log("La surface du rectangle est de", surface, "unités carrées.");
```

Exemple d'utilisation plus avancée:

```
class Rectangle {
  // ... (même code que précédemment)

  // Méthode pour afficher les dimensions

  afficherDimensions() {
    console.log(`Longueur : ${this.longueur}, Largeur : ${this.largeur}`);
  }
}

const rectangle1 = new Rectangle(10, 7);
rectangle1.afficherDimensions(); // Affiche : Longueur : 10, Largeur : 7
console.log("Surface :", rectangle1.calculerSurface());
```