Examen de Programmation Web - Copie d'Étudiant

Licence Informatique - 3ème année

Nom : Martin Sophie **Date :** 21 mars 2025

N° Étudiant : 20230142

Partie 1 : Concepts fondamentaux

Question 1

Expliquez la différence entre les méthodes HTTP GET et POST, et donnez un exemple d'utilisation appropriée pour chacune.

> La méthode GET est utilisée pour demander des données depuis un serveur. Les paramètres sont inclus dans l'URL sous forme de paires clé-valeur après un point d'interrogation. Cette méthode est idéale pour les requêtes qui ne modifient pas de données, comme la recherche. Par exemple, quand on fait une recherche sur Google, les termes recherchés sont visibles dans l'URL.
>
> La méthode POST est utilisée pour envoyer des données au serveur pour traitement. Les données sont incluses dans le corps de la requête HTTP et non dans l'URL. Cette méthode est appropriée pour les formulaires contenant des informations sensibles comme les mots de passe, ou pour les opérations qui modifient des données sur le serveur. Par exemple, quand on se connecte à un site web ou qu'on publie un commentaire.

Question 2

Qu'est-ce que le DOM (Document Object Model) ? Expliquez comment JavaScript interagit avec le DOM pour modifier dynamiquement une page web.

Le DOM (Document Object Model) est une représentation en mémoire de la structure d'une page web, organisée comme un arbre d'objets. Il représente chaque élément HTML comme un nœud dans cet arbre, avec des propriétés et des méthodes.

JavaScript interagit avec le DOM en permettant:

- De sélectionner des éléments spécifiques grâce à des méthodes comme `getElementById()`, `querySelector()`, etc.
- De modifier le contenu des éléments via des propriétés comme `innerHTML` ou `textContent`
- De manipuler les attributs des éléments avec `setAttribute()`
- D'ajouter ou supprimer des éléments avec `createElement()`, `appendChild()`, `removeChild()`
- De réagir aux événements utilisateur comme les clics ou la saisie de texte

Par exemple, pour ajouter dynamiquement un paragraphe à la page, on utiliserait:

```
const newParagraph = document.createElement('p');
newParagraph.textContent = 'Voici un nouveau paragraphe';
document.body.appendChild(newParagraph);
```

Question 3

Expliquez le concept de "responsive design" et décrivez trois techniques ou outils CSS que vous pouvez utiliser pour rendre un site web responsive.

Le responsive design est une approche de conception web qui vise à offrir une expérience de consultation optimale sur différents appareils (ordinateurs, tablettes, smartphones) en adaptant automatiquement l'affichage à la taille de l'écran et à l'orientation de l'appareil.

Trois techniques CSS pour créer un site responsive:

1. **Media Queries**: Permettent d'appliquer différents styles CSS selon les caractéristiques du périphérique comme la largeur d'écran.

```
@media (max-width: 768px) {
    /* Styles pour écrans de moins de 768px */
```

```
}
```

2. **Grilles flexibles (Flexbox)**: Permet de créer des mises en page flexibles qui s'adaptent automatiquement à l'espace disponible.

```
.container {
    display: flex;
    flex-wrap: wrap;
}
```

3. **Unités relatives**: Utilisation d'unités comme %, em, rem, vh, vw plutôt que des valeurs fixes en pixels.

```
.element {
    width: 80%; /* Pourcentage de la largeur du parent */
    font-size: 1.2rem; /* Relatif à la taille de police de bas
}
```

Partie 2 : HTML/CSS

Question 4

Observez le code HTML suivant et identifiez au moins quatre erreurs ou mauvaises pratiques. Proposez les corrections nécessaires.

Erreurs et corrections:

1. Utilisation de `<header>` au lieu de `<head>` pour l'en-tête du document.

```
<head> au lieu de <header>
```

2. Balise `<title>` mal fermée (manque /).

```
<title>Ma page</title>
```

3. Propriété CSS incorrecte pour la police et point-virgule manquant.

```
    font-size: 12px;
    color: red;
```

4. Balise `<p>` non fermée.

```
<p>Ceci est un autre paragraphe.</p>
```

5. Balise `<img>` non fermée et attribut alt manquant.

```
<img src="image.jpg" alt="Description de l'image">
```

6. Structure de `<table>` incomplète, il manque la fermeture de la balise.

```
</table>
```

Code corrigé:

```
<!DOCTYPE html>
<html>
<head>
  <title>Ma page</title>
  <style>
    p {
      font-size: 12px;
      color: red;
    }
  </style>
</head>
<body>
  <h1>Bienvenue sur ma page!</h1>
  <p>Ceci est un paragraphe.</p>
  <p>Ceci est un autre paragraphe.</p>
  <img src="image.jpg" alt="Description de l'image">
  <table>
    <tr><td>Cellule 1</td><td>Cellule 2</td></tr>
  </table>
</body>
</html>
```

Question 5

Écrivez le code HTML et CSS pour créer un formulaire de contact contenant les éléments suivants :

```
<!-- HTML -->
<div class="contact-form-container">
  <form id="contactForm">
    <div class="form-group">
      <label for="name">Nom</label>
      <input type="text" id="name" name="name" placeholder="Votre
    </div>

    <div class="form-group">
      <label for="email">Email*</label>
      <input type="email" id="email" name="email" placeholder="Vot
    </div>

    <div class="form-group">
      <label for="phone">Téléphone</label>
      <input type="tel" id="phone" name="phone" placeholder="Votre
    </div>

    <div class="form-group">
      <label for="subject">Sujet</label>
      <select id="subject" name="subject">
        <option value="support">Support technique</option>
        <option value="info">Demande d'information</option>
        <option value="other">Autre</option>
      </select>
    </div>

    <div class="form-group">
      <label for="message">Message</label>
      <textarea id="message" name="message" rows="5" placeholder="
    </div>

    <button type="submit">Envoyer</button>
  </form>
```

```
</div>

<!-- CSS -->
<style>
  .contact-form-container {
    max-width: 600px;
    margin: 0 auto;
    padding: 20px;
  }

  .form-group {
    margin-bottom: 20px;
  }

  label {
    display: block;
    margin-bottom: 5px;
    font-weight: bold;
  }

  input, select, textarea {
    width: 100%;
    padding: 10px;
    border: 1px solid #ddd;
    border-radius: 4px;
    font-size: 16px;
  }

  textarea {
    resize: vertical;
  }

  button {
    background-color: #4CAF50;
    color: white;
    padding: 12px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
```

```
    width: 100%;
  }

  button:hover {
    background-color: #45a049;
  }
</style>
```

Partie 3 : JavaScript

Question 6

Analysez le code JavaScript suivant et expliquez ce qu'il fait. Identifiez également tout problème potentiel et proposez des améliorations.

Analyse du code:

Ce code utilise XMLHttpRequest pour récupérer des données depuis une API. Lorsque l'utilisateur clique sur un bouton avec l'ID "fetchButton", la fonction fetchData() est appelée, qui envoie une requête GET à l'URL "https://api.example.com/data". Quand la réponse est reçue (readyState 4 et status 200), le code analyse les données JSON, puis crée un élément div pour chaque objet dans le tableau résultant, définit leur contenu sur la propriété "name" de l'objet et les ajoute au body de la page.

Problèmes et améliorations:

1. **Gestion des erreurs manquante**: Il n'y a pas de gestion d'erreur pour l'analyse JSON ou si la requête échoue.

   ```
   xhr.onerror = function() {
     console.error('Une erreur est survenue');
   };
   ```

2. **Utilisation de var**: Il est préférable d'utiliser let ou const au lieu de var pour éviter les problèmes de portée.

   ```
   const xhr = new XMLHttpRequest();
   ```

3. **Pas de vérification si fetchButton existe**: Si l'élément n'existe pas, une erreur sera générée.

```
const fetchButton = document.getElementById('fetchButton');
if (fetchButton) {
  fetchButton.addEventListener('click', fetchData);
}
```

4. **Méthode moderne**: Fetch API est plus moderne et plus simple à utiliser que XMLHttpRequest.

```
function fetchData() {
  fetch('https://api.example.com/data')
    .then(response => {
      if (!response.ok) {
        throw new Error('Erreur réseau');
      }
      return response.json();
    })
    .then(data => {
      data.forEach(item => {
        const div = document.createElement('div');
        div.textContent = item.name;
        document.body.appendChild(div);
      });
    })
    .catch(error => {
      console.error('Problème avec la requête fetch:', error);
    });
}
```

Question 7

Écrivez une fonction JavaScript qui prend un tableau de nombres en entrée et retourne un nouveau tableau contenant uniquement les nombres pairs, triés par ordre croissant.

```
function getEvenNumbersSorted(numbers) {
  // Vérifier si l'entrée est un tableau
  if (!Array.isArray(numbers)) {
    throw new Error('L\'argument doit être un tableau');
  }

  // Filtrer pour ne garder que les nombres pairs
  const evenNumbers = numbers.filter(num => {
    // Vérifier que c'est bien un nombre
    if (typeof num !== 'number') {
      return false;
    }
    // Garder seulement les nombres pairs
    return num % 2 === 0;
  });

  // Trier les nombres pairs par ordre croissant
  return evenNumbers.sort((a, b) => a - b);
}

// Exemple d'utilisation:
// const result = getEvenNumbersSorted([5, 2, 9, 8, 1, 6, 3, -4]);
// console.log(result); // [-4, 2, 6, 8]
```

Question 8

Créez une classe JavaScript `TodoList` avec les méthodes demandées.

```
class TodoList {
  constructor() {
    this.tasks = [];
    this.nextId = 1; // Pour générer des IDs uniques
  }

  addTask(taskData) {
```

```
    if (!taskData.title) {
      throw new Error('Le titre de la tâche est obligatoire');
    }

    const task = {
      id: this.nextId++,
      title: taskData.title,
      description: taskData.description || '',
      completed: false,
      createdAt: new Date()
    };

    this.tasks.push(task);
    return task.id; // Retourner l'ID de la nouvelle tâche
  }

  removeTask(id) {
    const initialLength = this.tasks.length;
    this.tasks = this.tasks.filter(task => task.id !== id);

    // Retourner true si une tâche a été supprimée
    return this.tasks.length !== initialLength;
  }

  markAsCompleted(id) {
    const task = this.tasks.find(task => task.id === id);
    if (!task) {
      return false; // Tâche non trouvée
    }

    task.completed = true;
    return true;
  }

  listTasks() {
    // Retourner une copie du tableau pour éviter les modification
    return [...this.tasks];
  }

  listCompletedTasks() {
```

```
        return this.tasks.filter(task => task.completed);
    }
}
```

Fin de l'examen