1 - Introduction à JavaScript

Objectif : Présenter aux participants ce qu'est JavaScript et son rôle dans le développement web, ainsi qu'installer et configurer l'environnement de développement.

1.1 Qu'est-ce que JavaScript ?

- Langage de programmation dynamique, léger et interprété.
- Fonctionne côté client, permettant des interactions dynamiques avec l'utilisateur sans recharger la page.
- Possibilité de fonctionner côté serveur avec des technologies comme Node.js.
- Créé par Brendan Eich en 1995 et standardisé sous le nom d'ECMAScript.

1.2 Rôle de JavaScript dans le développement web

- Manipulation du Document Object Model (DOM) pour modifier les éléments HTML et CSS.
- Validation des formulaires et collecte d'informations utilisateur.
- Création d'animations et d'effets visuels.
- Communication avec des serveurs pour récupérer ou envoyer des données (Ajax, Fetch API, etc.).
- Développement d'applications web en temps réel et d'applications mobiles avec des frameworks tels que React, Angular et Vue.js.

1.3 Installation et configuration de l'environnement de développement

- Navigateurs web modernes : supportent les dernières fonctionnalités de JavaScript (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari).
- Éditeur de texte pour la programmation : Visual Studio Code, Sublime Text, Atom, Notepad++.
- Console du navigateur : outil pour tester des scripts JavaScript et déboguer le code (F12 ou Ctrl+Shift+I pour ouvrir la console dans la plupart des navigateurs).
- Activation de JavaScript dans le navigateur : vérifier que JavaScript est activé pour permettre l'exécution des scripts.

1.4 Premier programme JavaScript

- Création d'un fichier HTML et ajout d'un script JavaScript à l'intérieur.
- Syntaxe de base pour insérer un script JavaScript dans un fichier HTML :

```
<html>
<head>
  <title>Introduction à JavaScript</title>
</head>
<body>
  <h1>Hello, JavaScript!</h1>
  <script> console.log("Hello, world!"); </script>
</body>
</html>`
```

- Enregistrement du fichier et ouverture dans un navigateur web pour voir le résultat.
- Vérification de la console du navigateur pour afficher le message "Hello, world!".

---

2 - Déclaration et utilisation des variables

Objectif : Apprendre aux participants à déclarer et utiliser des variables en JavaScript, ainsi qu'à connaître le type des variables.

2.1 Types de variables en JavaScript

- Var : Ancienne méthode pour déclarer des variables. Portée fonctionnelle et globale.
- Let : Introduit avec ES6 (ECMAScript 2015). Portée au niveau du bloc, évite les problèmes de portée globale.
- Const : Introduit avec ES6. Portée au niveau du bloc, valeur immuable (ne peut pas être modifiée après l'affectation initiale).

2.2 Déclaration et affectation des variables

- Syntaxe pour déclarer et affecter des variables en JavaScript :

```
var nom = "John";
let age = 30;
const PI = 3.14159;
```

2.3 Opérations de base sur les variables

- Addition, soustraction, multiplication et division :

```
let a = 10;
let b = 20;

let addition = a + b;
let soustraction = a - b;
let multiplication = a * b;
let division = a / b;
```

2.4 Savoir le type d'une variable

- Utilisation de l'opérateur `typeof` pour connaître le type d'une variable :

```
let nombre = 42;
let texte = "Hello, world!";
let estVrai = true;

console.log(typeof nombre); // "number"
console.log(typeof texte); // "string"
console.log(typeof estVrai);// "boolean"
```

- Attention a NaN : Not a Number :

```
let test = NaN;
// NaN = Not a Number
// Ca arrive quand on fait des trucs interdits sur des chiffres
//(ou ce qu'on croit être des chiffres)
if(typeof test === "number") {

    if(isNaN(test)) {
        console.log("C'est un nombre mais c'est pas un nombre : Nan")
    } else {
        console.log("c'est un nombre")

    }
} else {
    console.log("ce n'est pas un nombre")
}
```

2.5 Exemple pratique

- Créer un programme qui demande à l'utilisateur son nom et son âge, puis affiche un message avec ces informations :

```
<!DOCTYPE html>
<html>
<head>
  <title>Variables et types en JavaScript</title>
</head>
<body>
  <script>
    let prenom = prompt("Quel est votre prénom ?");
    let age = prompt("Quel est votre âge ?");

    alert("Bonjour, " + prenom + ". Vous avez " + age + " ans.");
  </script>
</body>
</html>
```

Bonus : Vérifier que l'âge est bien un nombre.

Les tableaux en JavaScript

Objectif : Apprendre aux participants à créer et manipuler des tableaux numériques, associatifs et à deux dimensions en JavaScript.

3.1 Les tableaux numériques

- Création d'un tableau numérique en JavaScript :

```
let fruits = ["pomme", "banane", "cerise", "orange"];
```

- Accès aux éléments d'un tableau :

```
console.log(fruits[0]); // "pomme"
console.log(fruits[2]); // "cerise"
```

- Modification d'un élément du tableau :

```
fruits[1] = "fraise";
console.log(fruits[1]); // "fraise"
```

- Ajout et suppression d'éléments :

```
fruits.push("kiwi"); // ajoute "kiwi" à la fin du tableau
fruits.unshift("mangue"); // ajoute "mangue" au début du tableau
fruits.pop(); // supprime le dernier élément du tableau
fruits.shift(); // supprime le premier élément du tableau
```

3.2 Les tableaux associatifs (objets)

- Syntaxe pour créer un objet :

```
let objet = {
  propriete1: valeur1,
  propriete2: valeur2
};
```

- Exemple d'un objet représentant une personne :

```
let etudiant = {
  prenom: "Jean",
  nom: "Dupont",
  age: 25,
  note: 15.5
};
```

- Accès et modification des propriétés d'un objet :

```
console.log(etudiant.prenom); // "Jean"
console.log(etudiant["nom"]); // "Dupont"

etudiant.age = 26;
etudiant["note"] = 16;
```

3.3 Les tableaux à deux dimensions

- Création d'un tableau à deux dimensions en JavaScript :

```
let matrice = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
```

- Accès et modification des éléments d'un tableau à deux dimensions :

```
console.log(matrice[0][1]); // 2
console.log(matrice[2][0]); // 7

matrice[1][1] = 10;
```

3.4 Exemple pratique

- Créer un programme qui demande à l'utilisateur d'entrer les noms des étudiants et leurs notes, puis affiche les informations sous forme d'un tableau :

```
<!DOCTYPE html>
<html>
<head>
  <title>Tableaux en JavaScript</title>
</head>
<body>
  <script>
    let etudiants = [];
    let n = parseInt(prompt("Combien d'étudiants voulez-vous ajouter ?"));

    for (let i = 0; i < n; i++) {
      let nom = prompt("Entrez le nom de l'étudiant " + (i + 1) + " :");
      let note = parseFloat(prompt("Entrez la note de l'étudiant " + (i + 1) + " :"));
      etudiants.push({nom: nom, note: note});
    }

    console.log("Liste des étudiants et de leurs notes :");
    for (let etudiant of etudiants) {
      console.log(etudiant.nom + " : " + etudiant.note);
    }
  </script>
</body>
</html>
```

Opérations sur les chaînes de caractères

Objectif : Apprendre aux participants à manipuler les chaînes de caractères en JavaScript, notamment la concaténation et les méthodes couramment utilisées.

4.1 Concaténation de chaînes de caractères

- Utilisation de l'opérateur `+` pour concaténer des chaînes de caractères :

```
let prenom = "John";
let nom = "Doe";
let nomComplet = prenom + " " + nom;
console.log(nomComplet); // "John Doe"
```

- Concaténation avec les littéraux de gabarit (template literals) :

```
let nomComplet2 = `${prenom} ${nom}`; // backticks -> altGr + 7
console.log(nomComplet2); // "John Doe
```

4.2 Méthodes et propriétés utiles pour les chaînes de caractères

- `length` : propriété qui retourne la longueur d'une chaîne de caractères

```
let texte = "Hello, world!";
console.log(texte.length); // 13
```

- `toUpperCase()` et `toLowerCase()` : méthodes pour convertir une chaîne en majuscules ou en minuscules

```
let texteMaj = texte.toUpperCase();
let texteMin = texte.toLowerCase();
console.log(texteMaj); // "HELLO, WORLD!"
console.log(texteMin); // "hello, world!"
```

- `indexOf()` : méthode pour rechercher la première occurrence d'une sous-chaîne dans une chaîne de caractères

```
let texte = "Hello, world!";
let position = texte.indexOf("world");
console.log(position); // 7
```

- `substring()` : méthode pour extraire une sous-chaîne à partir d'une chaîne de caractères

```
let extrait = texte.substring(0, 5);
console.log(extrait); // "Hello"
console.log(texte); // "Hello world!"
```

- `split()` : méthode pour diviser une chaîne de caractères en un tableau de sous-chaînes

```
let mots = texte.split(" ");
console.log(mots); // ["Hello,", "world!"]

let mots = texte.split("");

console.log(mots); // ["H","e","l", "l", "o", " ", "w", "o", "r", "l", "d", "!"]


let other = "Ceci est une phrase super longue."

other.split("phrase") // ["Ceci est une ", " super longue."]
```

4.3 Exemple pratique

- Créer un programme qui demande à l'utilisateur d'entrer une phrase, puis affiche le nombre de mots et le nombre de caractères de la phrase :

```
<!DOCTYPE html>
<html>
<head>
  <title>Manipulation des chaînes de caractères en JavaScript</title>
</head>
<body>
  <script>
    let phrase = prompt("Entrez une phrase :");
    let mots = phrase.split(" ");
    let nbMots = mots.length;
    let nbCaracteresWithoutSpace = phrase.replace(/ /g, "").length;

    alert(`Votre phrase contient ${nbMots} mots et ${nbCaracteresWithoutSpace}
caractères (sans les espaces).`);
  </script>
</body>
</html>
```

Opérations arithmétiques et modification de variables

Objectif : Apprendre aux participants à effectuer des opérations arithmétiques et à modifier des variables avec des opérateurs d'incrémentation et de décrémentation.

5.1 Opérations arithmétiques de base

- Addition, soustraction, multiplication, division et modulo :

```
let a = 10;
let b = 3;

console.log(a + b); // 13
console.log(a - b); // 7
console.log(a * b); // 30
console.log(a / b); // 3.3333333333333335
console.log(a % b); // 1
```

5.2 Opérateurs d'incrémentation et de décrémentation

- Incrémentation : ajout de 1 à la valeur d'une variable

```
let x = 5;
x++; // équivalent à x = x + 1
console.log(x); // 6
x-- //
```

5.3 Opérateurs d'assignation combinés

- Combinaison d'une opération arithmétique avec une assignation

```
let z = 20;
z += 5; // équivalent à z = z + 5
console.log(z); // 25

z *= 2; // équivalent à z = z * 2
console.log(z); // 50
```

5.4 Exemple pratique

- Créer un programme qui demande à l'utilisateur d'entrer deux nombres, puis affiche le résultat de toutes les opérations arithmétiques de base, sous forme d'objet :

```
let result = {
    addition : x,
    soustraction :y,
...
}
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Opérations arithmétiques en JavaScript</title>
</head>
<body>
  <script>
    let num1 = parseFloat(prompt("Entrez un nombre :"));
    let num2 = parseFloat(prompt("Entrez un autre nombre :"));

    let addition = num1 + num2;
    let soustraction = num1 - num2;
    let multiplication = num1 * num2;
    let division = num1 / num2;
    let modulo = num1 % num2;

    alert(`Résultats des opérations :
      - Addition : ${addition}
      - Soustraction : ${soustraction}
      - Multiplication : ${multiplication}
      - Division : ${division}
      - Modulo : ${modulo}`);
  </script>
</body>
</html>
```

Déclaration et utilisation des fonctions

Objectif : Apprendre aux participants à déclarer et utiliser des fonctions en JavaScript, afin d'organiser et de réutiliser le code plus efficacement.

6.1 Déclaration de fonctions

- Syntaxe pour déclarer une fonction en JavaScript :

```
function nomDeLaFonction(parametre1, parametre2) {
  // Instructions à exécuter
}
```

- Exemple d'une fonction simple :

```
function afficherBonjour() {
  console.log("Bonjour !");
}
```

6.2 Appel d'une fonction

- Syntaxe pour appeler une fonction en JavaScript :

```
nomDeLaFonction(arguments);
```

- Exemple d'appel de la fonction `afficherBonjour` :

```
afficherBonjour(); // Affiche "Bonjour !"
```

6.3 Fonctions avec paramètres

- Exemple d'une fonction avec des paramètres :

```
function afficherNomComplet(prenom, nom) {
  console.log(prenom + " " + nom);
}
```

- Appel de la fonction avec des arguments :

```
afficherNomComplet("John", "Doe"); // Affiche "John Doe"
```

6.4 Fonctions avec valeur de retour

- Exemple d'une fonction qui retourne une valeur :

```
function additionner(a, b) {
  return a + b;
}
```

- Utilisation de la valeur retournée par la fonction :

```
let somme = additionner(10, 20);
console.log(somme); // 30
```

6.5 Exemple pratique

- Créer un programme qui utilise des fonctions pour convertir une température entre Celsius et Fahrenheit :

```
<!DOCTYPE html>
<html>
<head>
  <title>Fonctions en JavaScript</title>
</head>
<body>
  <script>
    function celsiusToFahrenheit(celsius) {
      return (celsius * 9) / 5 + 32;
    }

    function fahrenheitToCelsius(fahrenheit) {
      return ((fahrenheit - 32) * 5) / 9;
    }

    let tempC = parseFloat(prompt("Entrez une température en degrés Celsius :"));
    let tempF = celsiusToFahrenheit(tempC);
    alert(`${tempC} °C équivaut à ${tempF} °F.`);

    let tempF2 = parseFloat(prompt("Entrez une température en degrés Fahrenheit :"));
    let tempC2 = fahrenheitToCelsius(tempF2);
    alert(`${tempF2} °F équivaut à ${tempC2} °C.`);
  </script>
</body>
</html>
```

7 - Conditions et boucles en JavaScript

Objectif : Apprendre aux participants à utiliser des structures de contrôle pour exécuter du code en fonction de conditions et répéter des instructions avec des boucles.

7.1 Conditions

- Syntaxe de la structure `if...else` :

```
if (condition) {
  // Instructions à exécuter si la condition est vraie
} else {
  // Instructions à exécuter si la condition est fausse
}
```

- Exemple d'utilisation de `if...else` :

```
let age = 18;

if (age >= 18) {
  console.log("Vous êtes majeur.");
} else {
  console.log("Vous êtes mineur.");
}
```

7.2 Boucle `for`

- Syntaxe de la boucle `for` :

```
for (initialisation; condition; miseAJour) {
  // Instructions à exécuter
}
```

- Exemple d'utilisation de la boucle `for` pour afficher les nombres de 1 à 10 :

```
for (let i = 1; i <= 10; i++) {
  console.log(i);
}
```

7.3 Boucle `while`

- Syntaxe de la boucle `while` :

```
while (condition) {
  // Instructions à exécuter
}
```

- Exemple d'utilisation de la boucle `while` pour afficher les nombres de 1 à 10 :

```
let j = 1;

while (j <= 10) {
  console.log(j);
  j++;
}
```

7.4 Exemple pratique

- Créer un programme qui demande à l'utilisateur d'entrer un nombre, puis affiche la table de multiplication de ce nombre :

```
<!DOCTYPE html>
<html>
<head>
  <title>Conditions et boucles en JavaScript</title>
</head>
<body>
  <script>
    let nombre = parseInt(prompt("Entrez un nombre pour afficher sa table de
multiplication :"));

    console.log(`Table de multiplication de ${nombre} :`);

    for (let i = 1; i <= 10; i++) {
      console.log(`${nombre} x ${i} = ${nombre * i}`);
    }
  </script>
</body>
</html>
```

8 - Gestion des événements en JavaScript

Objectif : Apprendre aux participants à utiliser les événements pour exécuter du code en réponse à des actions de l'utilisateur.

8.1 Introduction aux événements

- Les événements sont des actions qui se produisent lors de l'interaction avec la page Web, par exemple : clics de souris, appuis sur des touches, chargement de la page, etc.
- JavaScript peut être utilisé pour détecter ces événements et exécuter du code en réponse à ces actions.

8.2 Ajout d'un gestionnaire d'événements

- Syntaxe pour ajouter un gestionnaire d'événements à un élément HTML :

```
<element evenement="nomDeLaFonction()">
```

- Exemple d'utilisation d'un gestionnaire d'événements pour un bouton :

```
<button onclick="alert('Bouton cliqué !')">Cliquez ici</button>
```

9.3 Utilisation de la méthode `addEventListener`

- Syntaxe pour ajouter un gestionnaire d'événements à un élément en utilisant `addEventListener` :

```
element.addEventListener("evenement", nomDeLaFonction);
```

- Exemple d'utilisation de `addEventListener` pour un bouton :

```
<button id="monBouton">Cliquez ici</button>

<script>
  document.getElementById("monBouton").addEventListener("click", function() {
    alert("Bouton cliqué !");
  });
</script>
```

9.4 Exemple pratique

- Créer un programme qui change la couleur du texte d'un paragraphe lorsqu'on clique sur un bouton :

```html
<!DOCTYPE html>
<html>
<head>
  <title>Gestion des événements en JavaScript</title>
  <style>
    .couleur1 { color: red; }
    .couleur2 { color: blue; }
  </style>
</head>
<body>
  <p id="monParagraphe">Ceci est un exemple de texte.</p>
  <button id="monBouton">Changer la couleur</button>

  <script>
    let bouton = document.getElementById("monBouton");
    let paragraphe = document.getElementById("monParagraphe");

    bouton.addEventListener("click", function() {
      if (paragraphe.classList.contains("couleur1")) {
        paragraphe.classList.remove("couleur1");
        paragraphe.classList.add("couleur2");
      } else {
        paragraphe.classList.remove("couleur2");
        paragraphe.classList.add("couleur1");
      }
    });
  </script>
</body>
</html>
```

9 - Manipulation du DOM avec JavaScript

Objectif : Apprendre aux participants à manipuler le Document Object Model (DOM) en utilisant JavaScript pour ajouter, modifier ou supprimer des éléments et attributs HTML.

10.1 Sélection d'éléments

- Sélection d'un élément par son ID :

```javascript
let element = document.getElementById("monElement");
```

- Sélection d'éléments par leur classe :

```javascript
let elements = document.getElementsByClassName("maClasse");
```

- Sélection d'éléments par leur nom de balise :

```
let elements = document.getElementsByTagName("nomDeLaBalise");
```

10.2 Modification de contenu

- Modifier le contenu d'un élément :

```
element.innerHTML = "Nouveau contenu";
```

- Modifier la valeur d'un attribut :

```
element.setAttribute("attribut", "nouvelleValeur");
```

10.3 Ajout et suppression d'éléments

- Créer un nouvel élément :

```
let nouvelElement = document.createElement("nomDeLaBalise");
```

- Ajouter un nouvel élément en tant qu'enfant d'un élément existant :

```
element.appendChild(nouvelElement);
```

- Supprimer un élément enfant :

```
element.removeChild(enfantASupprimer);
```

10.4 Exemple pratique

- Créer un programme qui permet à l'utilisateur d'ajouter et de supprimer des éléments de liste :

```html
<!DOCTYPE html>
<html>
<head>
  <title>Manipulation du DOM avec JavaScript</title>
</head>
<body>
  <input id="elementInput" type="text" placeholder="Élément de liste">
  <button id="ajouter">Ajouter</button>
  <button id="supprimer">Supprimer le dernier élément</button>
  <ul id="maListe"></ul>

  <script>
    let ajouterBtn = document.getElementById("ajouter");
    let supprimerBtn = document.getElementById("supprimer");
    let elementInput = document.getElementById("elementInput");
    let liste = document.getElementById("maListe");

    ajouterBtn.addEventListener("click", function() {
      let nouvelElement = document.createElement("li");
      nouvelElement.textContent = elementInput.value;
      liste.appendChild(nouvelElement);
      elementInput.value = "";
    });

    supprimerBtn.addEventListener("click", function() {
      if (liste.lastChild) {
        liste.removeChild(liste.lastChild);
      }
    });
  </script>
</body>
</html>
```

Utilisation des requêtes AJAX en Javascript

Objectif : Apprendre aux participants à utiliser les requêtes AJAX pour communiquer avec des serveurs et échanger des données sans recharger la page.

11.1 Introduction à AJAX

- AJAX (Asynchronous JavaScript and XML) est une technique qui permet de communiquer avec un serveur et d'échanger des données de manière asynchrone, sans recharger la page.
- Les données échangées peuvent être sous différents formats, tels que XML, JSON, HTML, etc.

- XMLHttpRequest et Fetch API sont les principales méthodes pour effectuer des requêtes AJAX en JavaScript.

11.2 XMLHttpRequest

- Syntaxe pour créer une requête AJAX avec XMLHttpRequest :

```
let xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    // Traiter les données reçues
  }
};
xhttp.open("GET", "url", true);
xhttp.send();
```

11.3 Fetch API

- Syntaxe pour créer une requête AJAX avec Fetch API :

```
fetch("url")
  .then(response => response.text())
  .then(data => {
    // Traiter les données reçues
  });
```

11.4 Exemple pratique

- Créer un programme qui récupère des données JSON à partir d'une API et les affiche :

```html
<!DOCTYPE html>
<html>
<head>
  <title>Requêtes AJAX en JavaScript</title>
</head>
<body>
  <button id="charger">Charger les données</button>
  <div id="resultat"></div>

  <script>
    let chargerBtn = document.getElementById("charger");
    let resultatDiv = document.getElementById("resultat");

    chargerBtn.addEventListener("click", function() {
      fetch("https://jsonplaceholder.typicode.com/users")
        .then(response => response.json())
        .then(data => {
          let html = "<ul>";
          for (let utilisateur of data) {
            html += `<li>${utilisateur.name} (${utilisateur.email})</li>`;
          }
          html += "</ul>";
          resultatDiv.innerHTML = html;
        })
        .catch(error => {
          console.error("Erreur :", error);
        });
    });
  </script>
</body>
</html>
```

11 - Introduction aux Promesses et Async/Await en JavaScript

Objectif : Apprendre aux participants à utiliser les Promesses et Async/Await pour simplifier la gestion des opérations asynchrones en JavaScript.

12.1 Promesses

- Les Promesses sont des objets qui représentent le résultat d'une opération asynchrone.
- Une Promesse peut être dans l'un des trois états suivants : en attente (pending), résolue (fulfilled) ou rejetée (rejected).
- Syntaxe pour créer une Promesse :

```
let promesse = new Promise((resolve, reject) => {
  // Effectuer une opération asynchrone
  // Si l'opération réussit, appeler resolve() avec le résultat
  // Sinon, appeler reject() avec la raison de l'échec
});
```

- Utilisation des méthodes `then()` et `catch()` pour gérer les Promesses :

```
promesse
  .then(resultat => {
    // Traiter le résultat
  })
  .catch(erreur => {
    // Gérer l'erreur
  });
```

12.2 Async/Await

- Les mots-clés `async` et `await` permettent de simplifier la gestion des Promesses en utilisant un style de code plus proche des opérations synchrones.
- Syntaxe pour définir une fonction asynchrone :

```
async function maFonction() {
  // Utiliser await pour attendre le résultat d'une Promesse
}
```

- Exemple d'utilisation de `await` pour attendre le résultat d'une Promesse :

```
async function recupererDonnees() {
  try {
    let response = await fetch("url");
    let data = await response.json();
    // Traiter les données
  } catch (erreur) {
    // Gérer l'erreur
  }
}
```

12.3 Exemple pratique

- Modifier l'exemple de la partie 11 pour utiliser Async/Await :

```html
<!DOCTYPE html>
<html>
<head>
  <title>Async/Await en JavaScript</title>
</head>
<body>
  <button id="charger">Charger les données</button>
  <div id="resultat"></div>

  <script>
    let chargerBtn = document.getElementById("charger");
    let resultatDiv = document.getElementById("resultat");

    chargerBtn.addEventListener("click", async function() {
      try {
        let response = await fetch("https://jsonplaceholder.typicode.com/users");
        let data = await response.json();
        let html = "<ul>";
        for (let utilisateur of data) {
          html += `<li>${utilisateur.name} (${utilisateur.email})</li>`;
        }
        html += "</ul>";
        resultatDiv.innerHTML = html;
      } catch (error) {
        console.error("Erreur :", error);
      }
    });
  </script>
</body>
</html>
```

12 - Introduction aux modules JavaScript

Objectif : Apprendre aux participants à organiser et structurer leur code en utilisant des modules JavaScript pour améliorer la modularité et la réutilisabilité du code.

13.1 Concepts de base des modules

- Les modules permettent de séparer le code en plusieurs fichiers pour faciliter l'organisation, la maintenance et la réutilisation du code.
- Un module est un fichier JavaScript qui peut exporter des variables, des fonctions ou des classes pour être utilisées dans d'autres modules.
- Les modules sont chargés et exécutés de manière asynchrone et isolée, ce qui évite les conflits de noms et les problèmes de portée.

13.2 Exportation et importation de membres

- Syntaxe pour exporter un membre depuis un module :

```
export const maVariable = "valeur";
export function maFonction() {}
export class MaClasse {}
```

- Syntaxe pour importer un ou plusieurs membres depuis un autre module :

```
import { maVariable, maFonction, MaClasse } from "./monModule.js";
```

- Syntaxe pour importer un module entier :

```
import * as monModule from "./monModule.js";
```

13.3 Exportation par défaut

- Un module peut avoir un export par défaut, qui est généralement utilisé lorsqu'un module ne contient qu'une seule chose à exporter :

```
export default function maFonction() {}
```

- Syntaxe pour importer un export par défaut :

```
import maFonction from "./monModule.js";
```

13.4 Exemple pratique

- Créer un module `calculs.js` qui exporte des fonctions pour effectuer des opérations arithmétiques simples :

```js
// calculs.js
export function addition(a, b) {
  return a + b;
}

export function soustraction(a, b) {
  return a - b;
}

export function multiplication(a, b) {
  return a * b;
}

export function division(a, b) {
  return a / b;
}
```

```html
<!DOCTYPE html>
<html>
<head>
  <title>Modules JavaScript</title>
</head>
<body>
  <script type="module">
    import { addition, soustraction, multiplication, division } from "./calculs.js";

    console.log("Addition:", addition(4, 2));
    console.log("Soustraction:", soustraction(4, 2));
    console.log("Multiplication:", multiplication(4, 2));
    console.log("Division:", division(4, 2));
  </script>
</body>
</html>
```