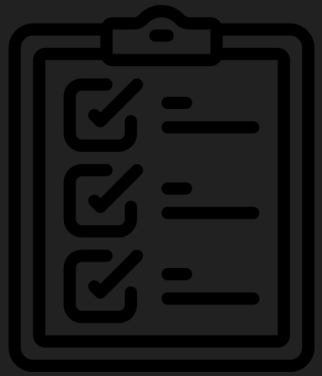
Introduction au langage JavaScript

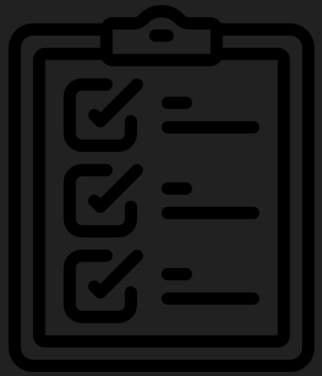
Maîtrisez le langage du web interactif

Plan du cours



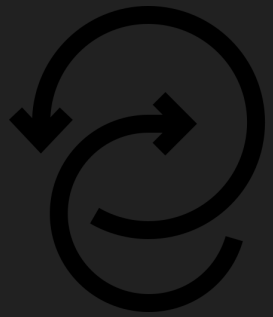
- Les objectifs
- Le JavaScript
- Histoire et évolution
- Avantages
- Outils de développement
- La balise script
- Variables
- Commentaires
- Type de données
- Opérateurs arithmétiques
- Opérateurs de comparaison

Plan du cours



- Opérateurs logiques
- Structures conditionnelles
- Structures iteratives
- Les tableaux
- Les fonctions
- Les objets prédéfinis
- La POO
- Fonctions anonymes et fléchées
- JavaScript DOM
- Gestion des événements

Les structures itératives



Les structures itératives

En JavaScript, il existe plusieurs structures itératives (boucles) qui permettent de répéter des instructions. Voici les trois structures itératives les plus couramment utilisées :

1. for
2. while
3. do while

Les structures itératives

Boucle for : La boucle for permet d'exécuter un bloc de code un certain nombre de fois en spécifiant une condition d'arrêt. Elle est souvent utilisée lorsque le nombre d'itérations est connu à l'avance.

```
for (initialisation; condition; expression d'itération) {
    // Code à exécuter
}
```

Les structures itératives

Exemple:

```
for (let i = 0; i < 5; i++) {
    console.log(i);
}
```

Dans cet exemple, la boucle for exécute le bloc de code cinq fois, en affichant les valeurs de i de 0 à 4.

Les structures itératives

La boucle "while" répète l'exécution d'un bloc de code tant qu'une condition spécifiée est vraie. Elle est particulièrement utile lorsque vous ne connaissez pas à l'avance le nombre exact d'itérations nécessaires.

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

Les structures itératives

Exemple:

```
let i = 0;
while (i < 5) {
 console.log(i);
 i++;
}
```

Dans cet exemple, la boucle while exécute le bloc de code tant que la valeur de i est inférieure à 5, en affichant les valeurs de i de 0 à 4.

Les structures itératives

Boucle do-while : La boucle do-while est similaire à la boucle while, mais elle exécute le bloc de code au moins une fois, puis répète l'exécution tant que la condition est vraie.

```
do {
    // Code à exécuter
} while (condition);
```

Les structures itératives

```
let i = 0;
do {
 // Bloc de code à répéter
 console.log(i);
 i++;
} while (i < 5);
```

Dans cet exemple, la boucle do-while exécute le bloc de code une première fois, puis le répète tant que la valeur de i est inférieure à 5, en affichant les valeurs de i de 0 à 4.

Les structures itératives

Il est important de noter que chaque boucle peut être contrôlée avec des instructions "break" pour sortir prématurément de la boucle, ou "continue" pour passer à l'itération suivante sans exécuter le reste du bloc de code.

Les structures itératives

```
for (var i = 1; i <= 10; i++) {
    console.log(i);
    if (i === 5) {
      break; // Sort de la boucle si i est égal à 5
    }
}
```

Les structures itératives

```
for (var i = 1; i <= 5; i++) {
    if (i === 3) {
      continue;
      // Passe à l'itération suivante si i est égal à 3
    }
    console.log(i);
}
```

Les tableaux

Les tableaux

Les tableaux sont des structures de données utilisées pour stocker et organiser des collections d'éléments. Ils peuvent contenir des valeurs de différents types, tels que des nombres, des chaînes de caractères, des objets, etc. Les tableaux sont flexibles et peuvent être modifiés dynamiquement en ajoutant, supprimant ou modifiant des éléments.

Les tableaux

Déclaration d'un tableau :

```
let tableau = []; // syntaxe littérale
let tableau = new Array(); // syntaxe avec le constructeur Array()
```

Les tableaux

Ajout d’un élément à un tableau:

```
tableau.push(1); // Ajoute l'élément 1 à la fin du tableau
tableau.unshift(2); // Ajoute l'élément 2 au début du tableau
```

Les tableaux

Suppression d'éléments d'un tableau:

```
tableau.pop(); // Supprime le dernier élément du tableau
tableau.shift(); // Supprime le premier élément du tableau
delete tableau[2]; // Supprime l'élément à l'index 2 du tableau
```

Les tableaux

Accès aux éléments d'un tableau:

```
console.log(tableau[0]); // Affiche le premier élément du tableau
```

Les tableaux

Autres opérations sur les tableaux:

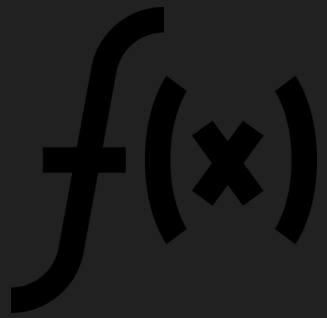
```
tableau.length; // Renvoie le nombre d'éléments dans le tableau
tableau.indexOf(element); // Renvoie l'index de la première occurrence de l'élément
tableau.slice(1, 4); // Renvoie un nouveau tableau avec les éléments de l'indice 1 à l'indice 3
tableau.sort(); // Trie les éléments du tableau (par ordre lexicographique pour les chaînes de caractères)
tableau.fill(0); // Tous les éléments du tableau sont remplacés par la valeur 0
tableau.fill(0, 1, 4); // les éléments de l'indice 1 à l'indice 3 sont remplacés par la valeur 0
```

Les tableaux

Autres opérations sur les tableaux:

```
tableau.join(','); // Renvoie une chaîne de caractères représentant tous les éléments du tableau séparés par ','
tableau.reverse(); //  Les éléments du tableau sont inversés
tableau.splice(2, 2); // Supprime 2 éléments à partir de l'indice 2
tableau.splice(2, 0, 'a', 'b'); // Ajoute 'a' et 'b' à partir de l'indice 2
tableau.toString(); // convertit un tableau en une chaîne de caractères
tableau.concat([4, 5, 6]); // Renvoie un nouveau tableau résultant de la concaténation du tableau existant avec [4, 5, 6]
tableau.map(fonction); // crée un nouveau tableau avec les résultats de l'appel d'une fonction fournie sur chaque élément du tableau appelant.
```

Les fonctions



Les fonctions

les fonctions sont des blocs de code réutilisables qui peuvent être appelés et exécutés à partir d'autres parties de votre programme. Les fonctions sont utilisées pour encapsuler une logique spécifique, effectuer des calculs, modifier des valeurs, ou exécuter des actions spécifiques.
avantages:
Réutilisabilité, Modularité, Abstraction, Organisation du code, Réduction de la duplication de code, Facilitation du débogage.

Les fonctions

Pour déclarer une fonction en JavaScript, vous pouvez utiliser la syntaxe suivante

```
function nomDeLaFonction() {
   // Bloc de code à exécuter
   // Instructions et logique de la fonction
}
```

Les fonctions

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

Dans cet exemple, nous avons déclaré une fonction nommée direBonjour qui se contente d'afficher le message "Bonjour !" dans la console.

Les fonctions

Pour appeler une fonction en JavaScript, vous devez utiliser le nom de la fonction suivi de parenthèses, éventuellement avec des arguments si la fonction en accepte.

on prend l’exemple ci-dessus avec la fonction direBonjour

```
direBonjour();
```

exécutera le code définit dans la fonction direBonjour

Les fonctions

Une fonction avec paramètres est une fonction qui accepte des valeurs (arguments) lors de son appel. Les paramètres permettent de passer des données spécifiques à une fonction, ce qui lui permet de les utiliser et de les manipuler à l'intérieur de son bloc de code.

```
function nomDeLaFonction(parametre1, parametre2) {
   // Bloc de code à exécuter
   // Utilisation des paramètres dans les instructions de la fonction
}
```

Les fonctions

```
function direBonjourPrenom(prenom) {
    console.log("Bonjour ", prenom);
}
```

Dans cet exemple, nous avons déclaré une fonction direBonjourPrenom qui accepte un paramètre : prenom. À l'intérieur de la fonction, nous utilisons ce paramètre pour afficher un message personnalisé.

Les fonctions

Pour appeler la fonction direBonjourPrenom, vous devez utiliser le nom de la fonction suivi de parenthèses en mettant le prenom de la personne à dire bonjour entre les parenthèses.

```
direBonjourPrenom("Alin");
```

```
affiche Bonjour Alin dans la console.
```

Les fonctions

Une fonction avec retour de valeur en JavaScript est une fonction qui renvoie une valeur spécifique après avoir effectué des opérations ou des calculs. L'instruction return est utilisée pour renvoyer la valeur souhaitée à partir de la fonction.
Les fonctions avec retour de valeur sont utiles lorsque vous avez besoin de récupérer et d'utiliser le résultat d'une opération spécifique. Vous pouvez utiliser la valeur renvoyée par la fonction dans d'autres parties de votre programme pour effectuer des manipulations supplémentaires, des comparaisons, des assignations, etc.
une fonction ne peut avoir qu’un seul return.

Les fonctions

```
function nomDeLaFonction(parametre1, parametre2, ...) {
    // Bloc de code à exécuter
    // Utilisation des paramètres dans les instructions de la
    fonction
   return resultat; // Valeur à renvoyer
}
```

Les fonctions

```
function addition(a, b) {
    let resultat = a + b;
    return resultat;
}
let somme = addition(3, 5);
console.log(somme); // Affiche 8
```

Les fonctions

En JavaScript, il est possible de définir des valeurs par défaut pour les paramètres d'une fonction. Cela signifie que si aucun argument n'est passé lors de l'appel de la fonction pour un paramètre donné, sa valeur par défaut sera utilisée à la place. Voici comment définir des paramètres par défaut dans une fonction :

```
function nomDeLaFonction(param = valeurParDefaut) {
    // Bloc de code à exécuter
    // Utilisation des paramètres dans les instructions de la fonction
}
```

Les fonctions

```
function direBonjourPrenom(prenom = "Anonyme") {
   console.log("Bonjour, " + nom + " !");
}
direBonjourPrenom(); // Affiche "Bonjour, Anonyme !"
direBonjourPrenom("Michel"); // Affiche "Bonjour, Michel !"
```

Liens utils

- https://developer.mozilla.org/en-US/docs/Web/JavaScript
- https://www.w3schools.com/js/default.asp
- https://devdocs.io/javascript/