Les structures de contrôle conditionnelles en JavaScript

> Les structures de contrôle conditionnelles en JavaScript permettent de prendre des décisions et d'exécuter différents blocs de code en fonction de conditions.
> Elles sont essentielles pour rendre vos programmes dynamiques et interactifs.

1. Les structures conditionnelles principales

1.1. if

> La structure `if` est la plus basique et permet d'exécuter un bloc de code si une condition est vérifiée.

Exemple :

```javascript
const age = 20;
if (age ⩾ 18) {
  console.log("Vous êtes majeur.");
}

const bool = true;
if (bool) {
  console.log("C'est vrai !");
} else {
  console.log("C'est faux !");
}
```

1.2. if...else

> La structure `if...else` permet d'exécuter un bloc de code si la condition est vérifiée, et un autre bloc si elle n'est pas vérifiée.

Exemple :

```javascript
const age = 16;
if (age ⩾ 18) {
  console.log("Vous êtes majeur.");
} else {
  console.log("Vous êtes mineur.");
}
```

1.3. if / else if / else

> Cet ensemble de structures est utilisé pour vérifier des conditions logiques et exécuter différents blocs de code en fonction des résultats.

Exemple :

```javascript
const temperature = 30;

if (temperature > 35) {
  console.log("Il fait très chaud !");
} else if (temperature > 20) {
  console.log("Le temps est agréable.");
} else {
  console.log("Il fait frais.");
}
```

1.4. switch

> La structure `switch` est particulièrement utile pour évaluer une expression et exécuter un cas parmi plusieurs, selon la correspondance.

Exemple :

```javascript
const météo = "pluie";

switch (météo) {
  case "soleil":
    console.log("Sortez vos lunettes de soleil !");
    break;
  case "pluie":
    console.log("N'oubliez pas votre parapluie !");
    break;
  case "neige":
    console.log("Les gants et l'écharpe sont requis !");
    break;
  default:
    console.log("Préparez-vous à tout !");
}
```

1.5. L'opérateur conditionnel (ternaire)

> L'opérateur ternaire est une manière concise d'écrire des conditions simples. Il est idéal pour des affectations ou des résultats rapides.

Exemple :

```javascript
const age = 18;

const message = (age ⩾ 18) ? "Accès autorisé" : "Accès refusé";
console.log(message); // "Accès autorisé"
```

2. Gestion des erreurs avec try...catch

2.1. try...catch

> Utilisé pour gérer les erreurs et éviter qu'un programme ne plante en cas de problème.
>
> Le bloc `try` contient le code à tester, tandis que le bloc `catch` intercepte et gère les erreurs si elles se produisent.

Exemple :

```javascript
try {
  const data = JSON.parse("{ nom: Alice }"); // Syntaxe JSON incorrecte
  console.log(data);
} catch (erreur) {
  console.log("Une erreur est survenue :", erreur.message);
}
```

2.2. finally

> Utilisé avec `try...catch`, le bloc `finally` s'exécute **toujours**, qu'il y ait une erreur ou non.
>
> Il est utile pour des **actions finales** comme nettoyer des ressources.

Exemple :

```javascript
try {
  const resultat = 10 / 2;
  console.log("Résultat :", resultat);
} catch (erreur) {
  console.log("Erreur :", erreur.message);
} finally {
  console.log("Opération terminée.");
}
```

3. À RETENIR

> - **if :** exécute un bloc de code si la condition est vraie.
> - **if...else :** exécute un bloc de code si la condition est vraie, sinon un autre bloc.
> - **If / else if / else :** idéal pour évaluer des conditions différentes séquentiellement.
> - **Switch :** simple à utiliser pour des cas multiples, évitant l'enchaînement de nombreux `if`.
> - **Opérateur ternaire :** pour des conditions concises sur une seule ligne.
> - **Gestion des erreurs :** le bloc `try...catch` permet d'attraper les erreurs et de les gérer proprement, tandis que `finally` garantit l'exécution d'un morceau de code à la fin.
>
> Ces **structures conditionnelles** sont indispensables pour **contrôler la logique** de vos programmes et leur donner la capacité de répondre à **diverses situations**.