Les opérateurs en JavaScript

> Les opérateurs sont des symboles ou des mots-clés permettant de réaliser des opérations sur des valeurs ou variables.
>
> Ils sont essentiels pour manipuler des données et construire des programmes dynamiques.

1. Les familles d'opérateurs

1.1. Les opérateurs arithmétiques

> Utilisés pour effectuer des calculs mathématiques sur des nombres.

Exemple :

```javascript
const a = 10;
const b = 5;

const addition = a + b; // 15
const soustraction = a - b; // 5
const multiplication = a * b; // 50
const division = a / b; // 2
const modulo = a % b; // 0 (reste de la division)
const puissance = a ** b; // 100000 (10^5)
```

1.2. Les opérateurs d'assignation

> Servent à assigner une valeur à une variable.

Exemple :

```javascript
let x = 10;
x += 5; // équivaut à x = x + 5 (résultat : 15)
x *= 2; // équivaut à x = x * 2 (résultat : 30)
x -= 10; // équivaut à x = x - 10 (résultat : 20)
x /= 4; // équivaut à x = x / 4 (résultat : 5)
x %= 2; // équivaut à x = x % 2 (résultat : 1)
x **= 3; // équivaut à x = x ** 3 (résultat : 1)
```

1.3. Les opérateurs de comparaison

> Comparent deux valeurs et renvoient un résultat booléen (`true` ou `false`).

Exemple :

```javascript
const a = 10;
const b = 5;

console.log(a > b);  // true
console.log(a < b);  // false
console.log(a ⩾ b); // true
console.log(a ⩽ b); // false
console.log(a == b); // false (égalité abstraite)
console.log(a === b); // false (égalité stricte)
console.log(a ≠ b); // true (différence abstraite)
console.log(a !== b); // true (différence stricte)
console.log(a == "10"); // true (égalité abstraite, conversion de type)
console.log(a === "10"); // false (égalité stricte, pas de conversion de type)
console.log(a ≠ "5"); // true (différence abstraite, conversion de type)
console.log(a !== "5"); // true (différence stricte, pas de conversion de type)
```

1.4. Les opérateurs logiques

> Permettent de combiner plusieurs conditions logiques.

Exemple :

```javascript
const a = true;
const b = false;

console.log(a && b); // false (ET logique)
console.log(a || b); // true (OU logique)
console.log(!a);     // false (NON logique)

//  Opérateur de nullish coalescing (??) ou coalescence nulle
// Renvoie la valeur de droite si la valeur de gauche est null ou undefined
const valeur = null;
const resultat = valeur ?? "Valeur par défaut";
console.log(resultat); // "Valeur par défaut"

const valeur2 = 0;
const resultat2 = valeur2 ?? "Valeur par défaut"; // 0 (car 0 n'est pas null ou undefined)
```

1.5. Les opérateurs de chaîne (concaténation)

> Spécifiques pour manipuler les chaînes de caractères.

Exemple :

```javascript
const texte1 = "Bonjour";
const texte2 = "tout le monde";

const result = texte1 + " " + texte2; // "Bonjour tout le monde"
```

1.6. Les opérateurs conditionnels (ternaire)

> Permettent d'écrire des conditions courtes.

Exemple :

```javascript
const age = 18;
const estMajeur = (age ⩾ 18) ? "Oui" : "Non"; // "Oui"
```

2. Les types d'opérateurs

2.1. Préfixés et post-fixés

> Appliqués avant ou après une variable pour incrémenter ou décrémenter sa valeur.

Exemple :

```javascript
let a = 5;

console.log(++a); // 6 (préfixé : incremente avant l'utilisation)
console.log(a++); // 6 (post-fixé : incremente après l'utilisation)
console.log(a);   // 7
```

2.2. Décalage de bits (bitwise shift)

> Utilisés pour déplacer les bits (0 ou 1) d'un nombre vers la gauche ou la droite.
> Chaque bit d'un nombre est décalé d'une position vers la gauche, et un 0 est ajouté à droite.

Exemple :

```javascript
const a = 5; // 0101 en binaire
const b = a << 1; // Décalage à gauche : 1010 (10 en décimal)
const c = a >> 1; // Décalage à droite : 0010 (2 en décimal)
```

2.3. Instance et type

> Pour vérifier la relation ou le type des objets.

Exemple :

```javascript
console.log(5 instanceof Number); // false (les nombres ne sont pas des objets)
console.log(typeof 5); // "number"
```

3. Opérateurs spéciaux (... et ?.)

L'operateur `...` (spread/rest, opérateur de décomposition/reste) et `?.` (optional chaining) sont des opérateurs spéciaux en JavaScript.
Utilisés pour la décomposition d'objets et la protection contre les erreurs.
Le premier permet de décomposer un objet ou un tableau en ses éléments individuels

Le second permet d'accéder à une propriété d'un objet sans provoquer d'erreur si l'objet est `null` ou `undefined`.

Exemple :

```javascript
const tableau = [1, 2, 3, 4, 5, 6];
const [a, b, ...rest] = tableau; // a = 1, b = 2, rest = [3, 4, 5, 6]

const objet = { nom: "Alice", age: 25, disponible: true };
const { nom, ...autres } = objet; // nom = "Alice", autres = { age: 25, disponible: true }

const utilisateur2 = { nom: "Bob" };
const nomUtilisateur2 = utilisateur2?.nom; // "Bob" (pas d'erreur)

const utilisateur = null;
const nomUtilisateur2 = utilisateur.nom; // TypeError: Cannot read property 'nom' of null
const nomUtilisateur = utilisateur?.nom; // undefined (pas d'erreur)
```

4. À RETENIR

- Familles principales :
  - Arithmétiques : addition (`+`), soustraction (`-`), etc.
  - Assignation : `=` et ses variations comme `+=`, `*=`.
  - Comparaison : `===`, `!==`, `<`, `>`.
  - Logiques : `&&` (ET), `||` (OU), `!` (NON).
  - Chaînes : concaténation avec `+`.
  - Conditionnels : opérateur ternaire (`condition ? valeur1 : valeur2`).
- Types importants :
  - Préfixés/post-fixés : `++`, `--`.
  - Décalage de bits : `<<`, `>>`.
  - Instance et type : `instanceof`, `typeof`.
- Spéciaux :
  - `...` (spread/rest)
  - `?.` (optional chaining).