1) Sujet — 2) Explications — 3) Exemples

©jeremy

Javascript

JS Strings

- at() — *Renvoi le premier charactère du text* — ex →
```
let text = "Mon text";
let character = text.at(0);
```
- charAt() — *Renvoi le premier charactère d'une chaîne de charactère* — ex →
```
let text = "HELLO WORLD";
let letter = text.charAt(0);
```
- trim() — *Supprime les espaces* — ex →
```
let text = "      Hello World!      ";
let result = text.trim();
```
- lenght — *Renvoi la longeur de la chaîne de charactère* — ex→
```
let text = "Hello World!";
let length = text.length;
```
- toLowerCase() — *Converti tous les charactères de la chaîne en minuscule* — ex→
```
let text = "Hello World!";
let result = text.toLowerCase();
```
- toUpperCase() — *Converti tous les charactères de la chaîne en Majuscule* — ex→
```
let text = "Hello World!";
let result = text.toUpperCase();
```
- replace() — *Remplace la chaîne de la chaîne de charactère* — ex→
```
let text = "Visit Microsoft!";
let result = text.replace("
Microsoft", "W3Schools");
```
- slice() — *Découpe les 5 première position d'une chaîne de charactère* — ex→
```
let text = "Hello world!";
let result = text.slice(0, 5);
```
- split() — *Divise les mots d'une chaîne de charactère* — ex→
```
let text = "How are you doing
today?";
const myArray = text.split(" ");
```
- constructor — *Récupère le constructeur d'une chaîne de charactère* — ex→
```
let message = "Hello World!";
let text = message.constructor;
```
- match() — *Une recherche de "ain" à l'aide d'une chaîne de charactère* — ex→
```
let text = "The rain in SPAIN stays
mainly in the plain";
text.match("ain");
```
- repeat() — *Créer des copies d'un texte* — ex→
```
let text = "Hello world!";
let result = text.repeat(2);
```

JS Numbers

- Float — *Décimaux* — ex→
```
let decimal = 2.5;
```
- Integer — *Entier* — ex→
```
let entier = 8;
```
- BigInt — *Nombres entiers de grande taille* — ex→
```
let grandNombre =
123456789012345678901234567890
1234567890n;
let somme = grandNombre +
10n; // BigInt
```
- Hexadécimal — *Hexadécimal : Préfixé par 0x.* — ex→
```
let hex = 0xFF;
```
- Binaire — *Binaire : Préfixé par 0b* — ex→
```
let bin = 0b1010;
```
- Octal — *Octal : Préfixé par 0o.* — ex→
```
let oct = 0o77;
```

JS Arrays

Raisonnement
- Déductif
- Inductif
- Analogique

JS Dates

Raisonnement
- Déductif
- Inductif
- Analogique

JS Operators

Arithmetic

- \+ — *Addition* — ex→ `x + y`
- * — *Multiplication* — ex→ `x*y`
- / — *Créer des copies d'un texte* — ex→ `x/y`
- ** — *Exponentiation* — ex→ `x**y`
- \- — *Soustraction* — ex→ `x-y`
- % — *Modulo* — ex→
```
let x = 10;
x %= 3; // x devient 1 (car 10 ÷ 3
donne un reste de 1)
```
- ++ — *pré-Incrémentation* — ex→ `++i`
- ++ — *post-Incrémentation* — ex→ `i++`
- -- — *pré-Décrémentation* — ex→ `--i`
- -- — *post-Décrémentation* — ex→ `i--`

Assignment

- = — *égalité* — ex→ `x=y`
- += — *Addition avec affectation* — ex→
```
let x = 10;
x += 5; // x devient 15
```
- -= — *Soustraction avec affectation* — ex→
```
let x = 10;
x -= 3; // x devient 7
```
- *= — *multiplication avec affectation* — ex→
```
let x = 4;
x *= 3; // x devient 12
```
- /= — *division avec affectation* — ex→
```
let x = 20;
x /= 4; // x devient 5
```
- %= — *modulo avec affectation* — ex→
```
let x = 10;
x %= 3; // x devient 1 (car 10 ÷ 3
donne un reste de 1)
```
- **= — *d'exponentiation avec affectation* — ex→
```
let x = 2;
x **= 3; // x devient 8 (2^3 = 8)
```

Logical Assignment Operators

- &&= — *ET logique avec affectation* — ex→ `x &&= y // x = x && y`
- ||= — *OU logique avec affectation* — ex→ `x ||= y // x = x || y`
- ??= — *Nullish coalescing avec affectation* — ex→ `x ??= y // x = x ?? y`

Comparison

- == — *Vérifie si deux valeurs sont égales en ignorant le type.* — ex→
```
5 == '5'; // true (convertit le type
implicitement)
```
- === — *Vérifie si deux valeurs sont égales et ont le même type.* — ex→
```
5 === '5'; // false (types différents)
5 === 5;  // true
```
- != — *Vérifie si deux valeurs sont différentes en ignorant le type.* — ex→
```
5 != '5'; // false (convertit le type
implicitement)
```
- !== — *Vérifie si deux valeurs sont différentes ou ont des types différents.* — ex→
```
5 !== '5'; // true (types différents)
5 !== 5;  // false
```
- > — *Vérifie si une valeur est strictement supérieure à une autre.* — ex→
```
10 > 5; // true
5 > 10; // false
```
- < — *Vérifie si une valeur est strictement inférieure à une autre.* — ex→
```
5 < 10; // true
10 < 5; // false
```
- >= — *Vérifie si une valeur est supérieure ou égale à une autre.* — ex→
```
10 >= 5; // true
5 >= 5;  // true
4 >= 5;  // false
```
- <= — *Vérifie si une valeur est inférieure ou égale à une autre.* — ex→
```
5 <= 10; // true
5 <= 5;  // true
10 <= 5; // false
```
- ? — *Permet d'évaluer une condition et de retourner une des deux valeurs possibles en fonction du résultat (condition vraie ou fausse). condition ? valeur_si_vrai : valeur_si_faux;* — ex→
```
let age = 18;
let message = age >= 18 ? "
Adulte" : "Mineur";
console.log(message); // "Adulte"
```

Logical

- && — *Logical AND - "ET logique"*
Cet opérateur renvoie true si toutes les conditions à gauche et à droite sont vraies. — ex→
```
console.log(true && true);  // true (
les deux sont vraies)
console.log(true && false); // false (
une condition est fausse)
console.log(false && true); // false (
la première est fausse)
console.log(0 && "Hello");  // 0 (0
est falsy)
```
- || — *Logical OR - "OU logique"*
Cet opérateur renvoie true si au moins une des deux conditions est vraie. — ex→
```
console.log(true || false); // true (
au moins une est vraie)
console.log(false || true); // true (
au moins une est vraie)
console.log(false || false); // false (
aucune n'est vraie)
console.log(0 || "Hello");  // "Hello" (
0 est falsy, "Hello" est truthy)
```
- ! — *Logical NOT - "NON logique"*
Cet opérateur inverse la valeur logique d' une expression. Si une valeur est truthy, elle devient falsy et vice versa. — ex→
```
let age = 18;
let message = age >= 18 ? "
Adulte" : "Mineur";
console.log(message); // "Adulte"
```