1. Optional Chaining (?)

// - Allows you to safely access properties of an object without causing errors if the object is null or undefined.

// Example:

const user = { profile: { name: "John" } };

console.log(user.profile?.name); // "John"

console.log(user.address?.street); // undefined

Optional Chaining ?. with addEventListener Example

document

.getElementById("scrollCntnt01")

?.addEventListener("click", function () {

// Select the main content element

const mainContent = document.getElementById("mainContent01");

if (mainContent) {

// Scroll the main content into view

mainContent.scrollIntoView({ behavior: "smooth" });

}

});

2. Nullish Coalescing (??)

// - Returns the right-hand side operand when the left-hand side is null or undefined. It does not return the right-hand side for other falsy values like 0, false, or "".

// Example:

let value = null;

console.log(value ?? "Default Value"); // "Default Value"

value = 0;

console.log(value ?? "Default Value"); // 0

3. Spread Operator (...)

// - Allows you to unpack elements from arrays or objects or to merge them.

// Example:

const arr1 = [1, 2, 3];

const arr2 = [...arr1, 4, 5];

console.log(arr2); // [1, 2, 3, 4, 5]

// Using the Spread Operator with objects

const obj1 = { name: "Alice", age: 25 };

const obj2 = { ...obj1, city: "New York" };

console.log(obj2); // { name: "Alice", age: 25, city: "New York" }

4. Rest Operator (...)

// The Rest Operator is used to collect arguments in functions or extract part of an array or object.

// Using the Rest Operator to collect arguments in a function

function sum(...numbers) {

return numbers.reduce((acc, num) => acc + num, 0);

}

console.log(sum(1, 2, 3, 4)); // 10

// Using the Rest Operator to extract part of an array

const [first, ...rest] = [1, 2, 3, 4];

console.log(first); // 1

console.log(rest); // [2, 3, 4]

5. Destructuring Assignment

// - Allows you to unpack values from arrays or properties from objects into variables.

// Example:

const person = { name: "John", age: 30 };

const { name, age } = person;

console.log(name, age); // "John" 30

6. Ternary Operator (? :)

// - A shorthand for an if-else statement.

// Example:

const age = 18;

const message = age >= 18 ? "Adult" : "Minor";

console.log(message); // "Adult"

7. Logical AND (&&) and OR (||)

// - AND (&&): Returns the second operand if the first one is truthy.

// - OR (||): Returns the first operand if it is truthy, otherwise returns the second operand.

// Example:

const isActive = true;

const status = isActive && "Active"; // "Active"

console.log(status);

const name = "";

const displayName = name || "Guest"; // "Guest"

console.log(displayName);

8. Template Literals (` `)

// - Allows for string interpolation and multi-line strings.

// Example:

const name = "John";

const greeting = `Hello, ${name}!`;

console.log(greeting); // "Hello, John!"

9. Function Arrow (=>)

// - A shorthand for writing anonymous functions, especially useful for inline callbacks.

// Example:

const add = (a, b) => a + b;

console.log(add(2, 3)); // 5

10. ?? vs ||

The difference between Nullish Coalescing (??) and the Logical OR (||) operator lies in how they handle falsy values:

1. Nullish Coalescing (??)

The Nullish Coalescing (??) operator only considers null or undefined as falsy values. It returns the right-hand operand only if the left-hand operand is either null or undefined.

Other falsy values like 0, false, "" (empty string), NaN are not considered falsy by ??.

let value1 = null*;*

console.log(value1 ?? "Default Value")*; // "Default Value"*

let value2 = 0*;*

console.log(value2 ?? "Default Value")*; // 0 (doesn't coalesce because 0 is not null or undefined)*

let value3 = ""*;*

console.log(value3 ?? "Default Value")*; // "" (doesn't coalesce because empty string is not null or undefined)*

2. Logical OR (||)

The Logical OR (||) operator returns the right-hand operand if the left-hand operand is falsy. It considers any falsy value (null, undefined, 0, false, "", NaN) as a reason to return the right-hand operand.

let value1 = null*;*

console.log(value1 || "Default Value")*; // "Default Value"*

let value2 = 0*;*

console.log(value2 || "Default Value")*; // "Default Value" (because 0 is falsy)*

let value3 = ""*;*

console.log(value3 || "Default Value")*; // "Default Value" (because empty string is falsy)*

Key Differences:

?? only considers null or undefined as falsy.

|| considers all falsy values (e.g., 0, false, "", NaN, null, undefined).

Use Case:

Use ?? when you only want to fall back to the default value for null or

undefined, and not for other falsy values like 0 or "".

Use || when you want to handle all falsy values in the same way and provide a default value for anything that is falsy.