1. Logical Operators

Logical operators in JavaScript are used to combine or manipulate boolean values (`true` or `false`) to make decisions in code. The primary logical operators are `&&` (AND), `||` (OR), and `!` (NOT).

1.1 Logical AND (`&&`)

The `&&` operator evaluates to `true` only if **both operands** are true. If either operand is false, the result is false.

Truth Table for `&&`

Operand 1	Operand 2	Result
true	true	true
true	false	false
false	true	false
false	false	false

Example

```
console.log(10 > 5 && 20 > 15); // true
console.log(10 > 15 && 20 > 15); // false
```

Short-Circuit Evaluation

The `&&` operator evaluates the left operand first. If it is falsy, the right operand is not evaluated.

```
let x = 0;
console.log(x && someFunction()); // 0
```

Truthy and Falsy Values

Falsy Values:

- `0`, `-0`, `0n`
- `null`
- `undefined`
- `NaN`
- `""`
- `false`

Truthy Values:

- Non-zero numbers: `1`, `-1`, `0.1`
- Non-empty strings: `"abc"`
- Objects `{}`, arrays `[]`, functions `()`

```
console.log(1 && "abc"); // "abc"
console.log(0 && "abc"); // 0
```

1.2 Logical OR (`||`)

The `||` operator evaluates to true if **at least one operand** is true.

Truth Table for `||`

Operand 1	Operand 2	Result
true	true	true
true	false	true
false	true	true
false	false	false

Example

```
console.log(10 > 5 || 20 < 15); // true
console.log(10 < 5 || 20 < 15); // false
```

Short-Circuit Evaluation

```
let y = 42;
console.log(y || someFunction()); // 42
```

Use Case: Default Values

```
let name = userInput || "Guest";
```

1.3 Logical NOT (`!`)

The `!` operator inverts the boolean value.

```
console.log(!true); // false
console.log(!0);    // true
console.log(!!"abc"); // true
```

---

2. Optional Chaining (`?.`)

The optional chaining operator allows safe access to deeply nested object properties.

How It Works

If the value before `?.` is `null` or `undefined`, the entire expression returns `undefined` without throwing an error.

Example

```
const product = {
  category: {
    shoe: {
      brand: "RedTape"
    }
  }
};

console.log(product?.category?.shoe?.brand); // "RedTape"
console.log(product?.category?.shirt?.brand); // undefined
```

Without Optional Chaining

```
console.log(product.category.shirt.brand); // TypeError
```

Use Cases

- Accessing deeply nested properties
- Calling optional methods:

```
const obj = {};
console.log(obj.method?.()); // undefined
```

Notes

- Introduced in ECMAScript 2020
- Works with properties, methods, and array indexes:
  - `obj?.prop`
  - `obj?.method()`
  - `arr?.[index]`

---

3. Literals

Literals are fixed values written directly in the code.

3.1 Primitive Literals

- Single values literals.
- Cannot change (Immutable)

```
  let a = 10. // a allocates a memory where 10 is stored.
      a = 100. // but when we reassign with '100' then it will again allocate
a new memory and a have new memory address.
      a = 300.  // Here same thing will happen and new memory will allocate
and and a have new address again.
 clg(a) // 300 then how it is immutable (explanation is above)?
```

- **Number**: `10`, `0.1`
- **BigInt**: `1234567890123456789 0n`
- **String**: `'hello'`, `"world"`
- **Boolean**: `true`, `false`
- **Null**: `null`
- **Undefined**: `undefined`
- **Symbol**: `Symbol('id')`
- **NaN**

3.2 Non-Primitive Literals

- multi values literals.
- Can change (mutable)

```
const arr = ["html", "sql"];
arr[1] = "css";
console.log(arr); // ["html", "css"]
```

- Here arr[1] = "css" will go to the same memory location and update the value only. Do not change the memory allocation.
- Here the value of const variable changes because the
  memory address do not change only value is being changed and this is possible.

```
const obj = { name: "Avinash" };
obj.name = "Ranjan";
console.log(obj); // { name: "Ranjan" }
```

---

4. Strings in JavaScript

```
- It is primitive bty it is non-primitive in javascript.
- It is a collection of characters.
- Strings are primitive, but act like objects due to
  method support.
```

4.1 Single-Line Strings

```javascript
const user = "Avinash";
const name = 'Ranjan';
console.log(user, name);
```

4.2 Multi-Line Strings

```javascript
let name = `I am Avinash
Ranjan. I am a software developer.`;
console.log(name);
```

String Interpolation

```javascript
console.log(`${name} You are great!`);
```

> This is called ***Template literals*** and this process is called String interpolation.

Properties and Methods

```
const str = "Hello, World!";
console.log(str.length); // 13
console.log(str.toUpperCase());
console.log(str.slice(0, 5));
console.log(str.indexOf("World"));
console.log(str.replace("World", "JavaScript"));
```

Strings are Immutable

```
let str = "HTML";
str[0] = "X";
console.log(str); // "HTML"
```

---

5. Additional Notes

5.1 BigInt

- BigInt always accepts whole number only.

```
const bigNum = 1234567890123456789 0n;
console.log(bigNum + 1n);
```
