Comparison Operators in JavaScript

Comparison operators are used to compare values in JavaScript. They return a Boolean value (`true` or `false`) based on the comparison.

---

1. Equality and Inequality Operators

1.1 `==` (Loose Equality)

- Compares two values for equality, performing type conversion if necessary.

Example:

```
console.log(5 == "5"); // true (type conversion happens)
console.log(5 == 5);   // true
console.log(5 == 6);   // false
```

1.2 `===` (Strict Equality)

- Compares two values for equality without type conversion (strict comparison).

Example:

```
console.log(5 === "5"); // false (no type conversion)
console.log(5 === 5);   // true
console.log(5 === 6);   // false
```

1.3 `!=` (Loose Inequality)

- Compares two values for inequality, performing type conversion if necessary.

Example:

```
console.log(5 != "5"); // false (type conversion happens)
console.log(5 != 6);   // true
```

1.4 `!==` (Strict Inequality)

- Compares two values for inequality without type conversion.

Example:

```
console.log(5 !== "5"); // true (no type conversion)
console.log(5 !== 5);   // false
console.log(5 !== 6);   // true
```

---

2. Relational Operators

2.1 `>` (Greater Than)

- Checks if the left value is greater than the right value.

Example:

```
console.log(5 > 3);  // true
console.log(3 > 5);  // false
console.log(5 > 5);  // false
```

2.2 `<` (Less Than)

- Checks if the left value is less than the right value.

Example:

```
console.log(5 < 3);  // false
console.log(3 < 5);  // true
console.log(5 < 5);  // false
```

2.3 `>=` (Greater Than or Equal To)

- Checks if the left value is greater than or equal to the right value.

Example:

```
console.log(5 >= 3);  // true
console.log(5 >= 5);  // true
console.log(3 >= 5);  // false
```

2.4 `<=` (Less Than or Equal To)

- Checks if the left value is less than or equal to the right value.

Example:

```
console.log(5 <= 3);  // false
console.log(5 <= 5);  // true
console.log(3 <= 5);  // true
```

---

3. Special Operators

3.1 `typeof`

- Returns the type of a value.

Example:

```
console.log(typeof 5);       // "number"
console.log(typeof "Hello"); // "string"
console.log(typeof true);    // "boolean"
```

3.2 `instanceof`

- Checks if an object is an instance of a specific class or constructor.

Example:

```
let date = new Date();
console.log(date instanceof Date);   // true
console.log(date instanceof Object); // true
```

Truth Table for Comparison Operators

Operator	Description	Example	Result
`==`	Loose equality	`5 == "5"`	`true`
`===`	Strict equality	`5 === "5"`	`false`
`!=`	Loose inequality	`5 != "5"`	`false`
`!==`	Strict inequality	`5 !== "5"`	`true`
`>`	Greater than	`5 > 3`	`true`
`<`	Less than	`3 < 5`	`true`
`>=`	Greater than or equal to	`5 >= 5`	`true`

Operator	Description	Example	Result
`<=`	Less than or equal to	`3 <= 5`	`true`
`typeof`	Returns data type	`typeof 5`	`"number"`
`instanceof`	Checks instance of constructor	`[] instanceof Array`	`true`

By mastering these comparison operators, you can write robust and efficient conditional logic in JavaScript.

Multiple Condition Checks in JavaScript

When you need to evaluate multiple conditions in a single statement, you can use **logical operators**. These operators help you combine or modify conditions.

Logical Operators

1. `&&` **(AND)**

   - Returns `true` only if **all** conditions are true.
   - If any condition is false, the whole expression is false.

   Syntax:

   ```
   condition1 && condition2
   ```

   Example:

   ```
   let age = 25;
   let hasLicense = true;

   if (age >= 18 && hasLicense) {
       console.log("You are allowed to drive.");
   } else {
       console.log("You are not allowed to drive.");
   }
   ```

   Flow:

   - Check `condition1`. If `false`, stop and return `false`.
   - If `condition1` is `true`, check `condition2`.
   - Return `true` only if both are `true`.

---

2. `||` **(OR)**

   - Returns `true` if **at least one** condition is true.

- Returns `false` only if **all** conditions are false.

Syntax:

```
condition1 || condition2
```

Example:

```javascript
let hasCar = false;
let hasBike = true;

if (hasCar || hasBike) {
    console.log("You have a vehicle.");
} else {
    console.log("You don't have a vehicle.");
}
```

Flow:

- Check `condition1`. If `true`, stop and return `true`.
- If `condition1` is `false`, check `condition2`.
- Return `false` only if both are `false`.

---

3. `!` **(NOT)**

- Reverses the truth value of a condition.
- If the condition is `true`, `!` makes it `false` (and vice versa).

Syntax:

```
!condition
```

Example:

```javascript
let isRaining = true;

if (!isRaining) {
    console.log("You can go outside without an umbrella.");
} else {
    console.log("Take an umbrella with you.");
}
```

Combining Multiple Conditions

You can combine multiple conditions with logical operators to handle complex scenarios.

Example 1: Using `&&` and `||`

```
let age = 20;
let hasID = true;
let isStudent = false;

if ((age >= 18 && hasID) || isStudent) {
    console.log("You qualify for the discount.");
} else {
    console.log("You don't qualify for the discount.");
}
```

Flow:

1. Check `(age >= 18 && hasID)`:
   - If `age` is `>= 18` **and** `hasID` is `true`, return `true`.
2. If the first condition is `false`, check `isStudent`:
   - If `isStudent` is `true`, return `true`.
3. If neither is `true`, the `else` block executes.

Example 2: Nested Conditions

```
let temperature = 25;
let weather = "sunny";

if (temperature > 20) {
    if (weather === "sunny") {
        console.log("It's a great day for a walk.");
    } else {
        console.log("It's warm, but not sunny.");
    }
} else {
    console.log("It's too cold outside.");
}
```

Example 3: Avoiding Nested Conditions with Logical Operators

The previous example can be simplified:

```
let temperature = 25;
let weather = "sunny";

if (temperature > 20 && weather === "sunny") {
    console.log("It's a great day for a walk.");
} else if (temperature > 20) {
    console.log("It's warm, but not sunny.");
} else {
    console.log("It's too cold outside.");
}
```

Short-Circuiting in Logical Operators

1. `&&` Short-Circuiting

- If the first condition is `false`, the rest are not checked.

Example:

```
let loggedIn = false;

loggedIn && console.log("Welcome back!"); // Nothing is printed
```

2. `||` Short-Circuiting

- If the first condition is `true`, the rest are not checked.

Example:

```
let userName = "";

let displayName = userName || "Guest";
console.log(displayName); // "Guest"
```

Real-World Example

Example: Checking Login

```
let username = "admin";
let password = "1234";

if ((username === "admin" && password === "1234") || username === "superuser") {
    console.log("Login successful!");
} else {
    console.log("Invalid credentials.");
}
```

Flow:

1. If `username` is `"admin"` **and** `password` is `"1234"`, grant access.
2. If `username` is `"superuser"`, grant access.
3. Otherwise, deny access.

Mastering logical operators and condition combinations allows you to handle complex scenarios efficiently in your JavaScript code!