ScopeTreasure Courses

JAVASCRIPT COURSES · LESSON 3.4 · CODE SCOPE SERIES

Operators

Doing the maths and making comparisons: arithmetic, assignment, the === golden rule, logical operators, and decisions built from combos.

Introduction

Variables store values — but real programs **do things** with values: add a bill, check a discount, decide who passes. The symbols that do this work are called **operators**. You already know some from school maths, but JavaScript has a few of its own, and one golden rule that will save you from a very famous bug.

In this lesson you will do arithmetic, meet the assignment family, compare values on a scale, learn why `===` is the only equals you should trust, and combine everything into one real discount decision.

(Roman Urdu: Operators woh nishan hain jo hisaab karte hain aur faislay banate hain — aur === sab se bara usool hai.)

Key points

- **Arithmetic:** `+` add, `-` subtract, `*` multiply, `/` divide, `%` **remainder**, `**` power. And `++` adds exactly one: after `cups++`, the value of `cups` is one bigger.
- `%` **gives what is left over:** `10 % 3` is `1` — ten cups in groups of three make three full groups, and **1 cup is left**.

  (Roman Urdu: % hamesha bacha hua hissa deta hai — 10 ko 3 ke groups mein baanto, 1 bach jayega.)
- **Assignment:** `=` **STORES, it does not compare.** `price = 250` puts `250` in the box. The shortcut family does maths and stores in one step: `price += 50` means `price = price + 50`. Same pattern for `-=`, `*=`, `/=`.
- **Comparison answers true or false:** `>` bigger, `<` smaller, `>=` bigger or equal, `<=` smaller or equal — like checking a value on a scale.
- **The loose trap:** `5 == "5"` is `true`. The double equals `==` **quietly converts the type** before comparing, so a number and a string can look "equal".

  (Roman Urdu: == chupke se type badal deta hai — is liye 5 aur "5" usay barabar lagte hain.)
- **The golden rule: always use `===`.** `5 === "5"` is `false` — strict equals checks the **value and the type**. Use `===` for equal and `!==` for not equal, always.
- **Logical operators combine decisions:** `&&` means ALL must pass (switches in series — one off, all off), `||` means ANY one is enough (switches in parallel), `!` flips a boolean: `!true` is `false`.
- **Combos make real decisions:** real checks mix everything — a bill, a member status, a coupon — joined with `&&` and `||` into one `true` or `false`.

Code examples

Example 1 — Arithmetic and the remainder

```
let cups = 10;

console.log(cups + 5);
console.log(cups - 4);
console.log(cups * 3);
console.log(cups / 4);
console.log(cups % 3);
console.log(2 ** 3);

cups++;
console.log(cups);
```

What it does: One variable, every arithmetic operator. Watch `/` give a decimal answer, `%` give only the leftover from grouping, `**` raise 2 to the power of 3, and `++` quietly add one to `cups` before the last print.

What you will see: seven lines in the Console: `15`, `6`, `30`, `2.5`, `1`, `8`, and finally `11`.

Example 2 — Loose vs strict: why === wins

```
console.log(5 == "5");
console.log(5 === "5");
console.log(5 != "5");
console.log(5 !== "5");
console.log(250 >= 250);
```

What it does: The same two values, compared four ways. Loose `==` converts `"5"` to a number and says equal; strict `===` checks the type and says not equal. The `!=` and `!==` pair mirrors it. The last line checks greater-or-equal on the same number.

What you will see: five lines in the Console: `true`, `false`, `false`, `true`, `true`. Memorise the pattern: whenever types might differ, only `===` and `!==` tell the truth.

Example 3 — The combo: a real discount decision

```
let bill = 1200;
let isMember = true;
let hasCoupon = false;

let bigBill = bill >= 1000;
let discount = bigBill && (isMember || hasCoupon);

console.log(bigBill);
console.log(discount);

if (discount) {
  console.log("DISCOUNT EARNED");
} else {
  console.log("No discount");
}
```

What it does: A real shop rule built from many operators at once: the bill must be at least 1000 (`>=`), **and** the customer must be a member **or** have a coupon (`||` inside the brackets). Both parts join with `&&`. The result drives an `if` — the full if/else lesson is next, 3.5.

What you will see: three lines in the Console: `true`, `true`, and `DISCOUNT EARNED` — because the bill is big enough and the customer is a member, even without a coupon.

Quick recap

- Arithmetic does maths (`%` = remainder), assignment stores (`=` and the `+=` family).
- **Golden rule: always `===` and `!==`** — `==` converts types quietly and lies to you.
- `&&` = all must pass, `||` = any one is enough, `!` = flip. Combos build real decisions.

Common mistakes

1. **Using `==` instead of `===`** — `5 == "5"` returns `true` and hides type bugs until they explode later. Fix: make `===` and `!==` your only equals, always.
2. **Using `=` when you mean to compare** — `if (price = 250)` stores a value instead of checking one, and the condition is almost always true. Fix: inside a condition, compare with `===`.
3. **Reading `%` as percent or divide** — `10 % 3` is not a percentage; it is the remainder, `1`. Fix: read `%` as "what is left after making full groups".

Practice

1. Start with `let bill = 0;` and use `+=` three times to add items of 250, 150, and 300. Print the bill after each step and check the final total is `700`.
2. Write a combo decision of your own: a student passes if `marks >= 40` **and** (`attendance >= 75` **or** `hasMedicalNote` is true). Test it with values that print `true`, then change one value so it prints `false`.

Practice quiz

1. Code checks 10 % 3 === 3 and fails — what is the fix?

- Change === 3 to === 1, because the remainder of 10 / 3 is 1
- Change % to /
- Change 10 to 9
- Change === to =

Answer: Change === 3 to === 1, because the remainder of 10 / 3 is 1

2. if (marks = 90) always runs — what is the fix?

- Change = to ===, because = stores a value instead of comparing
- Change if to while
- Add quotes around 90
- Change marks to Marks

Answer: Change = to ===, because = stores a value instead of comparing

3. What does this print?

```
let x = 5;
x++;
console.log(x);
```

- 6
- 5
- 51
- Error

Answer: 6

4. What does this print?

```
console.log(10 % 3);
```

- 1

- 3
- 0
- 3.33

Answer: 1

5. What does this print?

```
const a = 2;
const b = 3;
console.log(a * b);
```

- 6
- 5
- 23
- Error

Answer: 6

6. What does this print?

```
console.log(5 > 3);
```

- true
- false
- 5
- 3

Answer: true

7. What does this print?

```
console.log(2 ** 3);
```

- 8
- 6
- 5
- 9

Answer: 8
