JavaScript continue Statement

(Skipping Iterations + Filtering Logic)

Subtitle: Learn how to skip specific loop iterations without stopping the loop and use it for clean filtering and control flow.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

🧠 Introduction

In loops, not every iteration is useful.

Sometimes you need to:

- ignore invalid data
- skip unwanted values
- continue processing remaining items

Stopping the loop (like break) is too extreme.

Instead, JavaScript provides:

continue → skip current iteration, move to next

⚙️ Step 1: What is continue?

continue means:

Skip the current loop cycle and immediately move to the next iteration.

Syntax:

```
for (let i = 0; i < 5; i++) {
  if (i === 2) {
    continue;
  }
  console.log(i);
}
```

🔄 Step 2: Execution Flow

Start loop → check condition → if continue → skip remaining code → next iteration

Output:

0

1

3

4

👉 2 is skipped, not printed

🧠 Step 3: Why continue Exists

Without continue:

- you need nested if conditions
- code becomes messy

With continue:

- cleaner logic
- early skip behavior
- better readability

🌍 Step 4: Real-World Use Cases

1. Skipping Invalid Data

```
let users = ["Ali", "", "Sara", null, "Ahmed"];


for (let i = 0; i < users.length; i++) {
  if (!users[i]) {
    continue;
  }
  console.log(users[i]);
}
```

2. Filtering Even Numbers

```
for (let i = 1; i <= 10; i++) {
  if (i % 2 !== 0) {
    continue;
  }
  console.log(i);
}
```

3. Skipping Blocked Users

```
let users = [
  { name: "Ali", blocked: false },
  { name: "Sara", blocked: true },
  { name: "Ahmed", blocked: false }
];

for (let user of users) {
  if (user.blocked) {
    continue;
  }
  console.log(user.name);
}
```

🧠 Step 5: Mental Model

Loop iteration → check condition → skip → move next → repeat

⚙️ Step 6: continue vs break

Feature	continue	break
Action	skip iteration	stop loop
Effect	partial control	full control
Loop	continues	ends

⚠️ Step 7: Common Mistakes

❌ 1. Confusing continue with break

- continue → skip one step
- break → stop entire loop

❌ 2. Using continue unnecessarily

Sometimes if-condition is better.

❌ 3. Infinite loop risk in while

```
while (i < 5) {
 if (i === 2) continue;
 i++;
}
```

👉 i++ skipped → infinite loop risk

🧪 Step 8: Mini Exercises

Exercise 1

Print numbers 1–20 except multiples of 3.

Exercise 2

Skip empty strings in array.

Exercise 3

Print only active users.

🧠 Step 9: Mini Quiz

1. What does continue do?
2. Difference between continue and break?
3. Can continue cause infinite loops?
4. Where is continue used in real systems?

🚀 Step 10: Thinking Upgrade

If you understand continue:

- you can control data flow inside loops
- you can build filtering logic efficiently
- you reduce unnecessary nested conditions

👉 This is essential for clean data processing systems.

📌 Step 11: Summary

- continue skips current iteration
- loop continues normally after skip
- used for filtering and cleanup logic
- different from break (which stops loop)
- improves readability and control flow