Loop Patterns

(Engineering Thinking with Real Data Logic)

Subtitle: Learn how professionals use loops not just for repetition but for solving structured problems like filtering, searching, and accumulating data.

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

🧠 Introduction

So far, you've learned loops as syntax:

- for loop
- while loop
- nested loops
- break / continue

Now the real shift happens:

Loops are not about repetition — they are about **patterns of problem-solving**.

Most beginner code fails not because they don't know loops, but because they don't know **what pattern to apply**.

⚙️ Step 1: What are Loop Patterns?

Loop patterns are **reusable thinking models** used to solve common data problems.

Instead of writing random loops, you choose a pattern:

- Filter Pattern
- Accumulate Pattern
- Search Pattern

🔍 Step 2: Search Pattern (Find Something)

Problem:

Find a specific value in data.

Example:

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

```
let target = "Sara";

let found = false;



for (let i = 0; i < users.length; i++) {

  if (users[i] === target) {

    found = true;

    break;

  }

}



console.log(found);
```

Logic:

- check each item
- stop when found
- avoid unnecessary work

Real Use Cases:

- login verification
- product search
- database lookup

➕ Step 3: Accumulate Pattern (Build Result)

Problem:

Combine values into one result.

Example (Sum):

```
let numbers = [10, 20, 30];



let sum = 0;
```

```
for (let i = 0; i < numbers.length; i++) {
  sum += numbers[i];
}

console.log(sum);
```

Logic:

- start with initial value
- keep adding results
- final output is accumulated value

Real Use Cases:

- total price calculation
- analytics (views, clicks)
- scoring systems

🧹 Step 4: Filter Pattern (Select Data)

Problem:

Extract only useful data.

Example:

```
let numbers = [1, 2, 3, 4, 5];

let evens = [];

for (let i = 0; i < numbers.length; i++) {
  if (numbers[i] % 2 === 0) {
    evens.push(numbers[i]);
  }
}
```

console.log(evens);

Logic:

- check condition
- keep only valid items
- ignore rest

Real Use Cases:

- filtering products
- active users list
- search results

🧠 Step 5: Mental Model of Patterns

Loop → Decision → Action → Result

Each pattern changes only the "Action" part.

⚙️ Step 6: Pattern Comparison

Pattern	Purpose	Output
Search	find item	boolean / item
Accumulate	combine data	single value
Filter	select items	array

🌍 Step 7: Real System Example

E-commerce System

let products = [

{ name: "Shoes", price: 100 },

{ name: "Bag", price: 50 },

```
  { name: "Watch", price: 200 }
];

// Filter
let cheap = [];

for (let i = 0; i < products.length; i++) {
  if (products[i].price < 100) {
    cheap.push(products[i]);
  }
}
```

Same system uses all patterns:

- search product
- filter products
- accumulate total cart

⚠️ Step 8: Common Mistakes

❌ 1. Writing random loops

No clear pattern = messy logic

❌ 2. Mixing patterns

Search + filter + accumulate in same loop unnecessarily

❌ 3. Forgetting early exit in search

wastes performance

❌ 4. Not initializing accumulator properly

leads to wrong results

🧪 Step 9: Mini Exercises

Exercise 1

Find maximum number in array (accumulate pattern).

Exercise 2

Filter users above age 18.

Exercise 3

Search for product in list.

🧠 Step 10: Mini Quiz

1. What is a loop pattern?
2. Difference between filter and search?
3. What is accumulator used for?
4. Why are patterns better than random loops?

🚀 Step 11: Thinking Upgrade

If you understand loop patterns:

- you stop writing basic loops
- you start solving structured problems
- you think like backend systems

👉 This is transition from beginner → engineer mindset.

📌 Step 12: Summary

- Loop patterns = structured ways of using loops
- Search = find item
- Filter = select items
- Accumulate = build result
- Used in all real applications
- Core foundation of data processing