Mastering Loops in TypeScript

1. Problem Statement

A city library needs a **Transaction Processor** that can handle hundreds of book check-ins, check-outs, and returns every day. The system must:

- Iterate through lists of transactions.
- Stop processing when a priority book arrives.
- Skip cancelled transactions.
- Tally total transactions per day.
- Process at least one pending return even if the queue is short.
- Summarize inventory and active visitors.

Manually handling this in spreadsheets is too slow and error-prone. You need to use loops in TypeScript to automate these tasks efficiently.

---

2. Learning Objectives

By the end of this tutorial, you will be able to:

- Use `for`, `while`, and `do…while` loops for definite and indefinite iteration.
- Apply `break` to exit loops early.
- Apply `continue` to skip to the next iteration.
- Use `for…in` to iterate object keys.
- Use `for…of` to iterate array and string values.
- Avoid infinite loops and common pitfalls.

---

3. Concept Introduction with Analogy

Imagine you start your day at the library's **Conveyor Belt Station**. This belt brings in little bins-each bin is a "transaction packet" that tells you what happened with a book (checkout, return, cancellation, or a high-priority request). Here's how your shift unfolds:

1. **Inspecting Each Bin (Loops)**
   The belt never stops unless you tell it to. You pick up each bin one by one, inspect its label, and decide what to do. This is your loop-processing each transaction in order.

2. **Skipping Cancellations (`continue`)**
   Some bins are marked **"Cancelled"**-no action needed. You glance at them, set them aside, and wave the belt on. In code, you use `continue` to skip the rest of the steps for that iteration and move to the next bin.

3. **Handling Priority Requests (`break`)**
   Every so often, a bin arrives with a **"PRIORITY"** sticker-this is an urgent reservation. You press the big red button and

stop the belt so you can process that first. In code, you use `break` to exit the loop immediately and deal with high-priority work.

4. **Ensuring at Least One Return (`do…while`)**
   Even if the day's returns aren't queued, policy says you must check for a **return** at least once (in case a librarian added one while you were busy). You reach out, inspect a bin, and then keep checking until you've processed every return. That first guaranteed check is like a `do…while` -run at least once, then repeat if the condition holds.

5. **Keeping Daily Tallies (Loop Counters)**
   At a corner of your workspace is a clipboard. Every time you handle a checkout, return, or priority, you tick a box on that clipboard. Your loop counters in code do the same-incrementing counts for each transaction type so you can report totals at day's end.

6. **Reviewing the Inventory Ledger (`for…in`)**
   Between belt cycles, you glance at the library's **Inventory Ledger**, a book listing each title and its current stock. You flip through each entry-this is akin to a `for…in` loop over an object's keys, checking "The Hobbit," then "1984," then "TypeScript Guide," and so on.

7. **Greeting Visitors (`for…of`)**
   At the end of your shift, you have a list of visitors waiting at the front desk. You call out each name and welcome them: "Alice, your book is ready. Bob, enjoy your reading." That's a `for…of` loop-iterating directly over an array of visitor names and performing an action for each.

---

4. Technical Deep Dive: Loop Types in TypeScript

Loop Type	Use Case	Syntax
for	Known iterations (e.g., array or fixed count)	for (init; condition; step) { … }
while	Unknown iterations until a condition becomes false	while (condition) { … }
do…while	Run block at least once, then repeat if condition true	do { … } while (condition);
for_in	Iterate enumerable property keys of an object	for (let key in obj) { … }
for_of	Iterate values of iterable (array, string, etc.)	for (let item of iterable) { … }

Control Statements:

- **`break`** : Exit current loop immediately.
- **`continue`** : Skip the rest of this iteration, continue with next.

---

5. Step-by-Step Code Walkthrough

5.1 Data Modeling

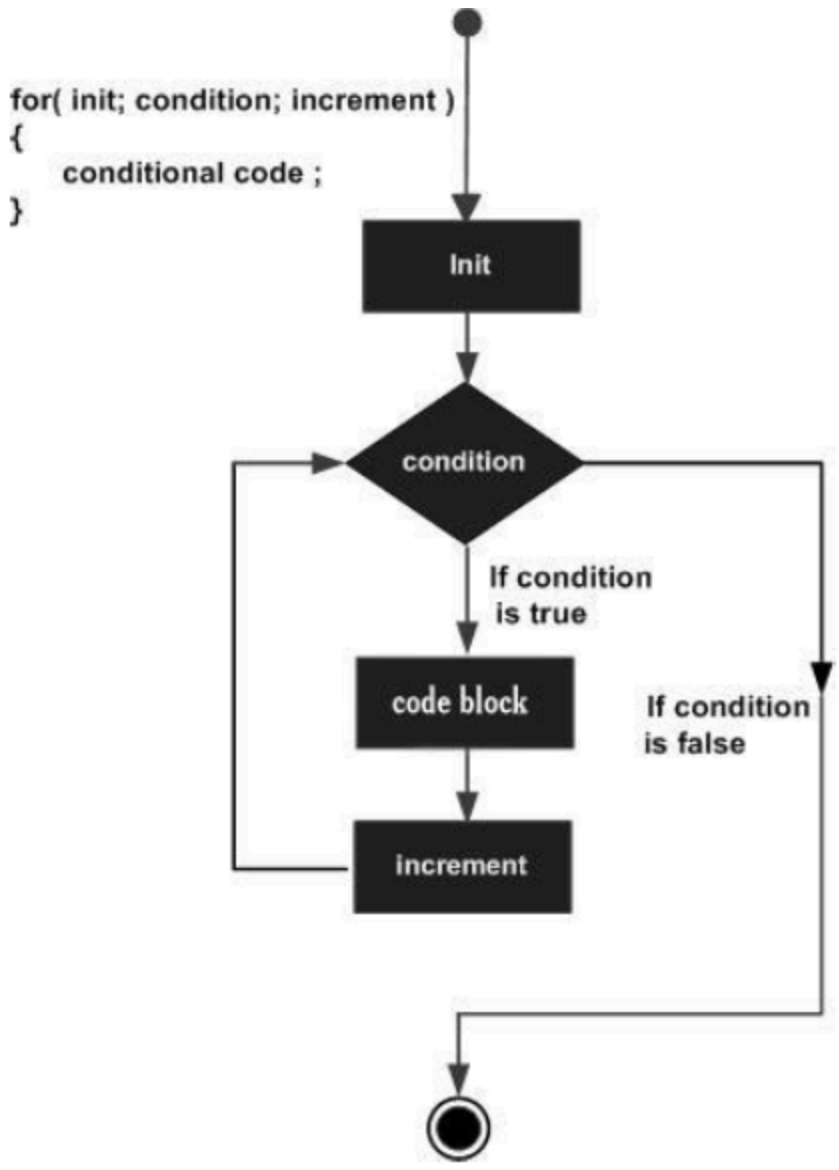
```ts
type Transaction = {
  id: number;
  type: "checkout" | "return" | "cancelled" | "priority";
};

const transactions: Transaction[] = [
  { id: 1, type: "checkout" },
  { id: 2, type: "cancelled" },
  { id: 3, type: "return" },
```

```
  { id: 4, type: "priority" },
  { id: 5, type: "checkout" }
];

const inventory: { [title: string]: number } = {
  "The Hobbit": 3,
  "1984": 5,
  "TypeScript Guide": 2
};

const visitors: string[] = ["Alice", "Bob", "Carol"];
```

5.2 `for` Loop: Process a Fixed List

```
let totalProcessed = 0;
for (let i = 0; i < transactions.length; i++) {
  const tx = transactions[i];
  if (tx.type === "cancelled") {
    continue; // skip cancelled
  }
  if (tx.type === "priority") {
    console.log("Priority transaction encountered, stopping.");
    break;    // stop on priority
  }
  console.log(`Processing transaction ${tx.id}: ${tx.type}`);
  totalProcessed++;
}
console.log(`Total processed: ${totalProcessed}`);
```

Flowchart

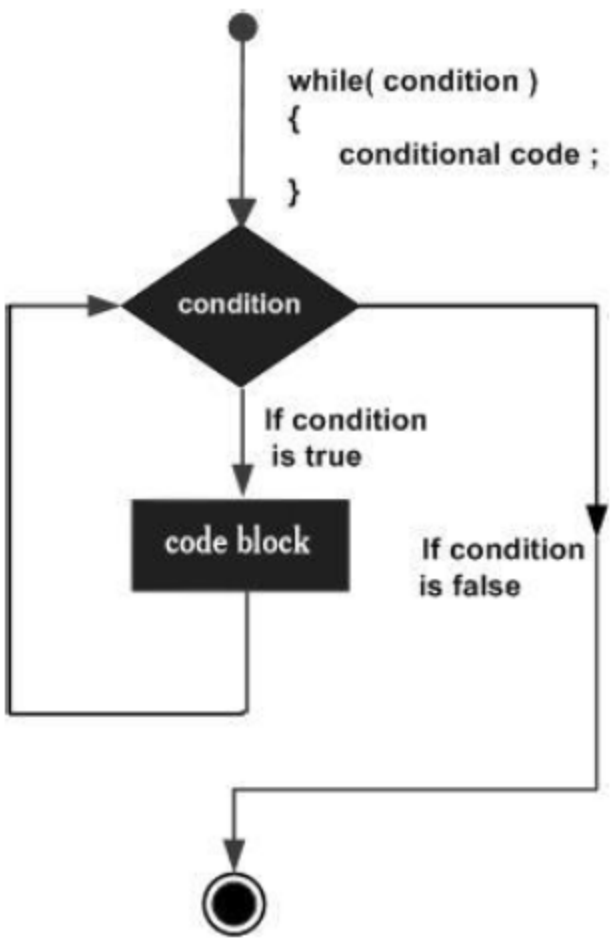


5.3 `while` Loop: Process Until Empty

```
let queue = [...transactions];
let processedWhile = 0;
while (queue.length > 0) {
  const tx = queue.shift()!;
  if (tx.type === "cancelled") continue;
  console.log(`While loop processed: ${tx.id}`);
  processedWhile++;
}
```

Flow Diagram

The flow diagram of the while loop looks as follows

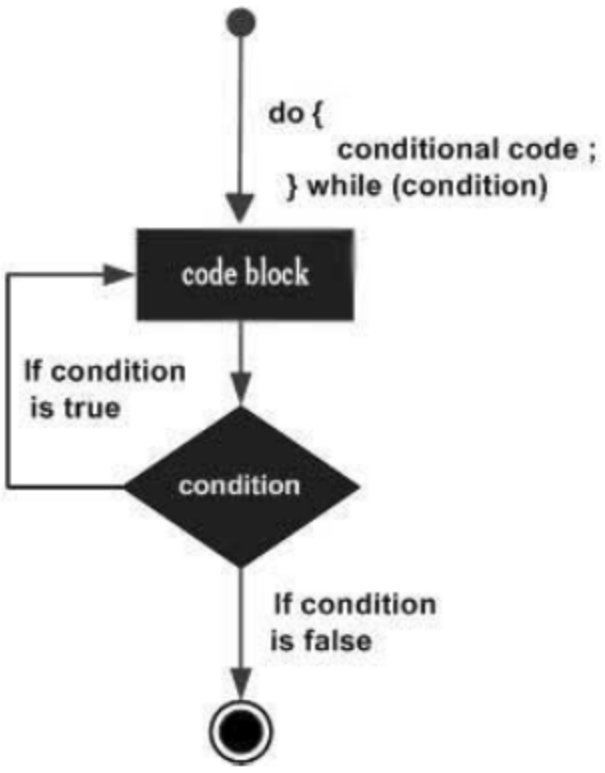


5.4 `do...while` Loop: Always Run Once

```
let pendingReturns = 0;
let idx = 0;
do {
  const tx = transactions[idx];
  if (tx.type === "return") {
    console.log(`Handling return transaction ${tx.id}`);
    pendingReturns++;
  }
  idx++;
} while (idx < transactions.length);
console.log(`Pending returns: ${pendingReturns}`);
```

Flowchart

The flowchart of the do...while loop looks as follows



---

6. Interactive Challenge / Mini-Project

Your Turn!

1. Add a counter for each transaction type (`checkout` , `return` , `priority` , `cancelled`) using a **`for`** loop and an object.
2. Use a **`while(true)`** infinite loop with a **`break`** condition when a new priority transaction arrives.
3. Modify the **`do…while`** loop to handle a dynamic queue (an array you can `push` new returns into).
4. Use **`for…in`** to reset all inventory counts to zero.
5. Display visitor names in reverse order using a **`for`** or **`for…of`** loop.

---

7. Common Pitfalls & Best Practices

- **Off-by-one errors** in `for` loops: ensure correct start and end conditions.
- **Infinite loops**: always include a `break` or update condition.
- **`for…in` on arrays**: returns string keys-use only for objects.
- **`for…of` on objects**: not supported-use `for…in` or `Object.entries` .
- **Clear loop counters**: avoid reusing loop variables across loops.

---

8. Quick Recap & Key Takeaways

- Use **`for`** for fixed or array-based iteration.
- Use **`while`** when the iteration count is unknown.
- Use **`do…while`** to guarantee at least one execution.
- Use **`break`** and **`continue`** to control loop flow.
- Use **`for…in`** for object keys, **`for…of`** for iterable values.

9. Optional: Programmer's Workflow Checklist

- Choose the right loop type for the task.
- Declare loop variables with proper types.
- Avoid infinite loops-ensure exit conditions.
- Use `break` and `continue` sparingly for clarity.
- Leverage `for…in` for objects and `for…of` for arrays/strings.
- Test each loop with edge cases (empty arrays, single items).