TypeScript Basics for Automation Testers – Day 7

Topic: **Looping Statements in TypeScript**

What Are Loops in Programming?

In programming, we often need to perform the same action multiple times.
For example, checking multiple checkboxes, verifying a list of API responses or printing a set of numbers.

Instead of writing the same code again and again, we use **loops**.
A loop executes a block of code **repeatedly** until a given **condition** becomes **false**.

In simple words,

A loop repeats a set of instructions **as long as** a condition is true.

We already know what a condition is—it always returns a **boolean** (`true` or `false`).
So loops depend heavily on conditions.

Why Do We Need Loops in Automation Testing?

In automation, you often:

- Click multiple elements in a list.
- Validate repeated API responses.
- Retry actions until an expected state appears.

Loops make these tasks simple and readable.

Types of Loops in TypeScript

TypeScript supports three main types of loops:

1. **while loop**
2. **do…while loop**
3. **for loop**

Let’s understand each one clearly.

1.While Loop

The **while** loop repeatedly executes a block of code **as long as** the given condition evaluates to `true`.

Syntax

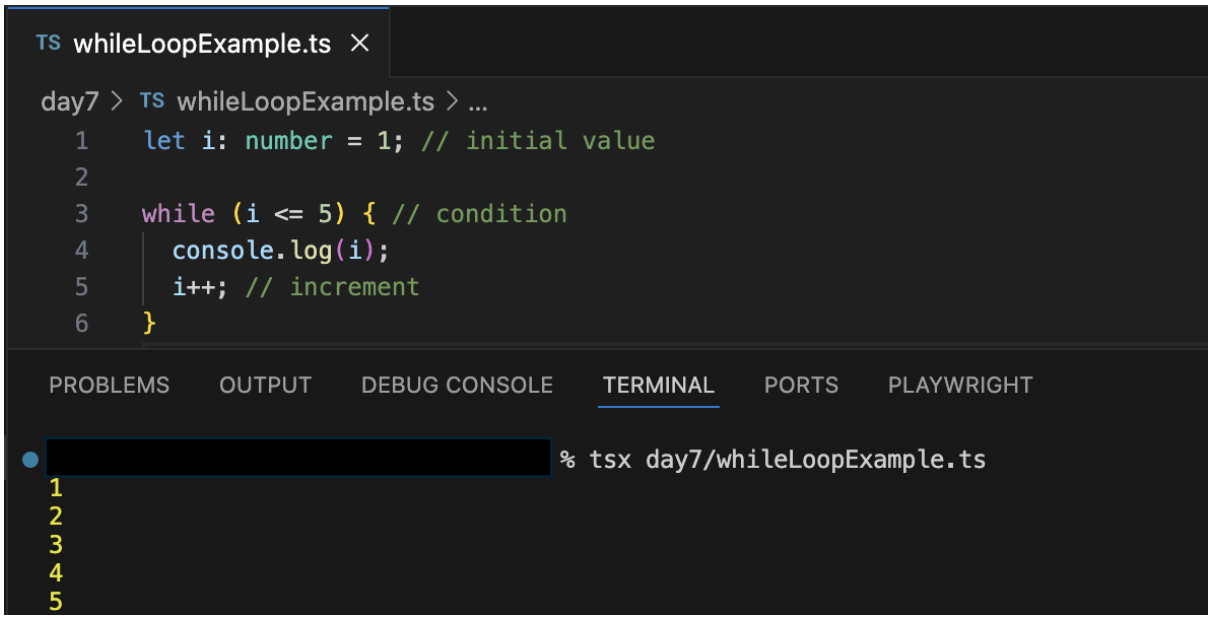
```
while (condition) {
  // code to execute
}
```

How It Works

- The condition is checked **first**.
- If it's `true`, the code inside runs.
- After execution, it checks again.
- The loop stops when the condition becomes `false`.

So, it's a **"check first, then run"** type of loop.

Example 1: Basic While Loop




```
let i: number = 1; // initial value

while (i <= 5) { // condition
  console.log(i);
  i++; // increment
}
```

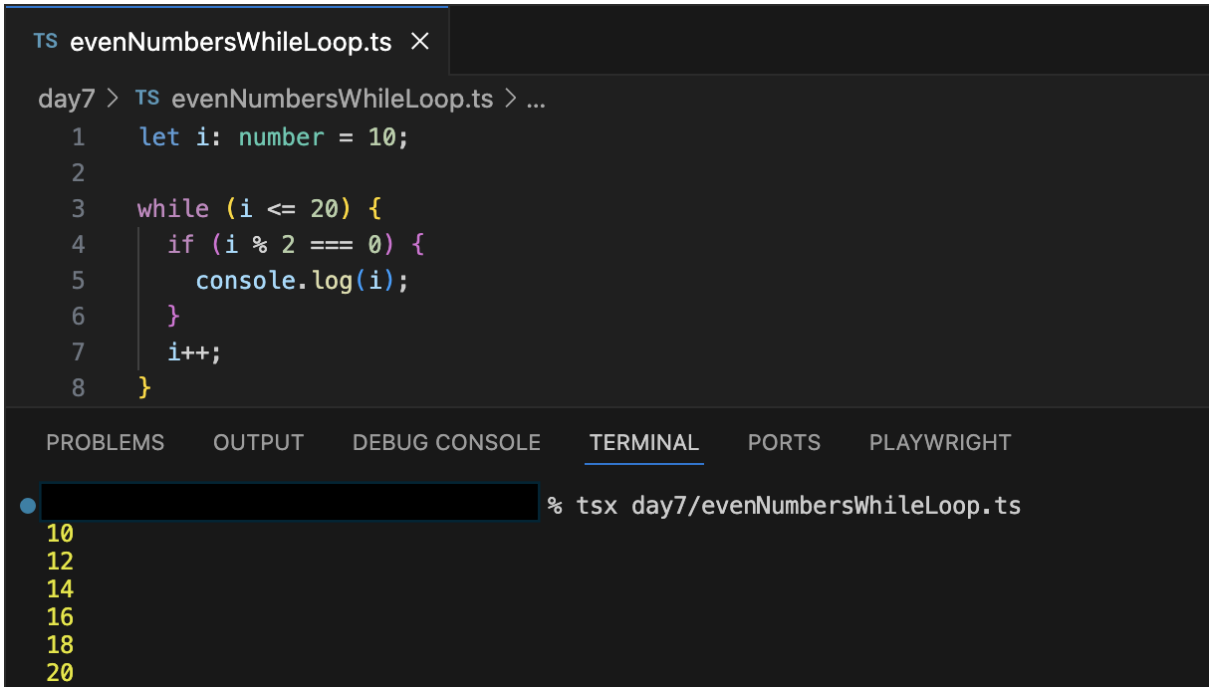
Explanation (Step-by-Step):

1. Start with `i = 1`.
2. Check if `i <= 5` → true → print `1`.
3. Increment `i` → `i = 2`.
4. Again check `i <= 5` → still true.
5. Continue until `i` becomes 6.
6. When `i = 6`, condition fails → loop stops.

This prints:

```
1
2
3
4
5
```

Example 2: While Loop with Condition Inside




```
let i: number = 10;

while (i <= 20) {
  if (i % 2 === 0) {
    console.log(i);
  }
  i++;
}
```

Explanation:
This prints all even numbers between 10 and 20.
It checks the main loop condition (`i <= 20`) first, then the inner condition (`i % 2 === 0`).

When to Use While Loop

Use a **while** loop when:

- You **don't know** how many times you need to run the code.
- You just want to continue **until** a certain condition becomes false.

Automation Example:
Keep checking if a button becomes enabled:

```
while (!isButtonEnabled()) {
  console.log("Waiting for button to enable...");
}
```

2.Do–While Loop

The **do–while** loop runs the code block **once first**, and **then checks** the condition.

Even if the condition is false at the beginning, the loop will execute once.

Syntax

```
do {
  // code to execute
} while (condition);
```

Example 1: Basic Do–While Loop

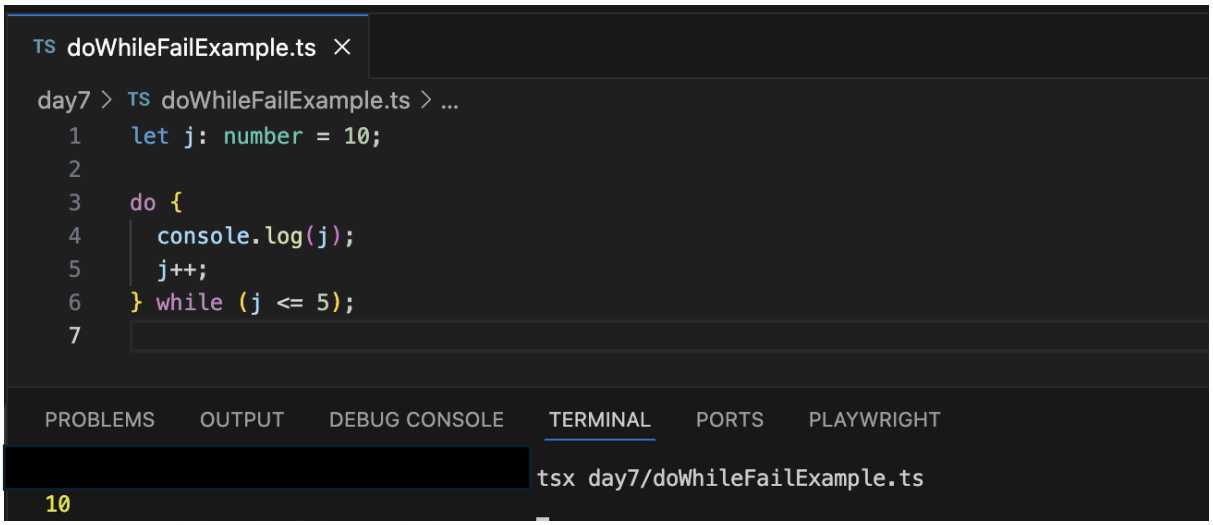
```
let j: number = 1;

do {
  console.log(j);
  j++;
} while (j <= 5);
```

Output:

```
1
2
3
4
5
```

Example 2: Condition Fails Initially




```
let j: number = 10;

do {
  console.log(j);
  j++;
} while (j <= 5);
```

Output:

```
10
```

Explanation:
Even though the condition `j <= 5` is false right from the start,
the loop executes once because the condition is checked **after** the execution.

When to Use Do–While Loop

Use **do–while** when you need the code to execute **at least once**, even if the condition fails.

Automation Example:
You might want to click a "Retry" button at least once, even before checking if a retry is needed.

```
do {
  clickRetryButton();
} while (!isResponseSuccessful());
```

3.For Loop

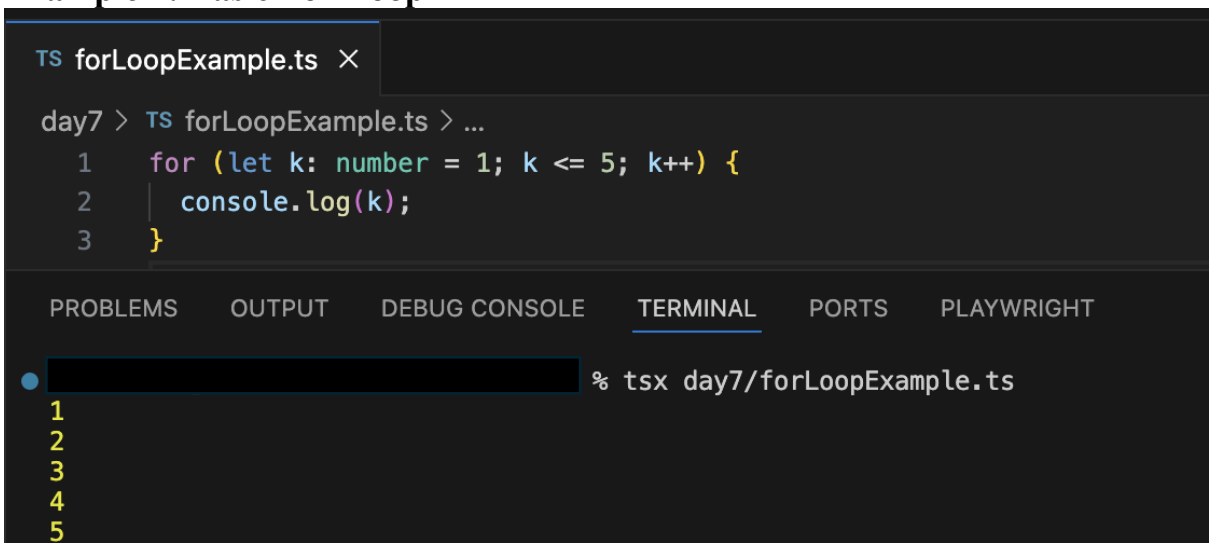
The **for** loop is used when the number of iterations is **known** beforehand.

It combines **initialization**, **condition** and **increment/decrement** in one line.

Syntax

```
for (initialization; condition; increment/decrement) {
  // code to execute
}
```

Example 1: Basic For Loop

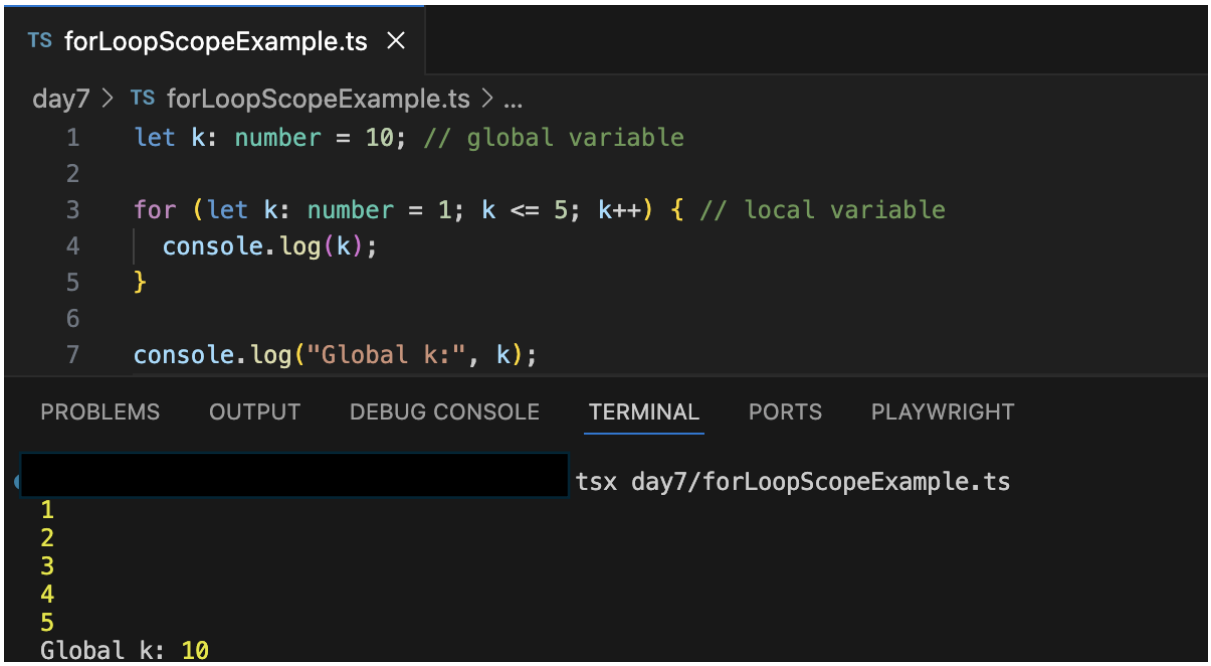



```
for (let k: number = 1; k <= 5; k++) {
  console.log(k);
}
```

Execution Flow:

1. `let k = 1` → initialize.
2. Check `k <= 5` → true → print `k`.
3. `k++` → increment.
4. Re-check → loop continues until condition fails.

Example 2: Understanding Local vs Global Variables




```
let k: number = 10; // global variable

for (let k: number = 1; k <= 5; k++) { // local variable
  console.log(k);
}

console.log("Global k:", k);
```

Output:

```
1
2
3
4
5
Global k: 10
```

Explanation:
The `k` inside the loop is **local** and different from the **global** `k`.
If we remove the `let` inside the loop, the same global variable will be reused.

When to Use For Loop

Use a **for** loop when:

- You know the exact number of times you want to execute code.
- You're iterating over fixed lists or counters.

Automation Example:
Looping through all checkboxes on a page:

```
for (let i = 0; i < allCheckboxes.length; i++) {
  await allCheckboxes[i].click();
}
```

Java vs TypeScript – Loop Comparison

Feature	Java	TypeScript
Syntax	`for (int i=0; i<5; i++)`	`for (let i=0; i<5; i++)`
Variable Declaration	Uses primitive types like `int`, `float`	Uses `number`, `string` etc.
Scoping	`int` variables are function-scoped	`let` variables are block-scoped
Console Output	`System.out.println()`	`console.log()`
Loop Types	while, do-while, for, for-each	while, do-while, for, for-of, for-in

---

Questions

1. What is a loop and why do we use it in programming?
2. How does a while loop work?
3. What is the key difference between while and do–while loops?
4. When should we use a for loop?
5. Explain the concept of local vs global variables in for loops.
6. Write a TypeScript example to print all even numbers between 1 and 10 using while loop.
7. How can you use a loop in automation testing scenarios?
8. What happens if the condition in a while loop is always true?

Answers

1. A loop allows us to execute a block of code repeatedly until a condition becomes false. It's used to avoid code repetition.
2. The while loop checks the condition first and executes only if it's true.
3. The do–while loop executes once even if the condition is false, while the while loop may not run at all.
4. The for loop is used when the number of iterations is known.
5. Local variables are declared inside a loop block and visible only there; global ones exist outside and are visible everywhere.
6.
```
let i = 1;
while (i <= 10) {
  if (i % 2 === 0) console.log(i);
  i++;
}
```

7. Loops can automate repetitive steps like clicking multiple buttons or validating multiple records.
8. If the condition is always true, the loop becomes **infinite** and never stops.
