TypeScript Basics for Automation Testers – Day 8

Topic: Jumping Statements in TypeScript

What are Jumping Statements?

jumping means "to skip or move suddenly from one point to another."
In programming, **jumping statements** allow the program to *jump out of* or *skip* certain parts of code — instead of following the usual top-to-bottom flow.

These are mainly used to **control the loop flow or function behaviour**.

Types of Jumping Statements

1. **break**
2. **continue**
3. **return**

1.Break Statement

"Break" means **to stop**.
In TypeScript, the **`break` statement** immediately stops (or breaks) the loop when a condition is met.

Why and Where to Use

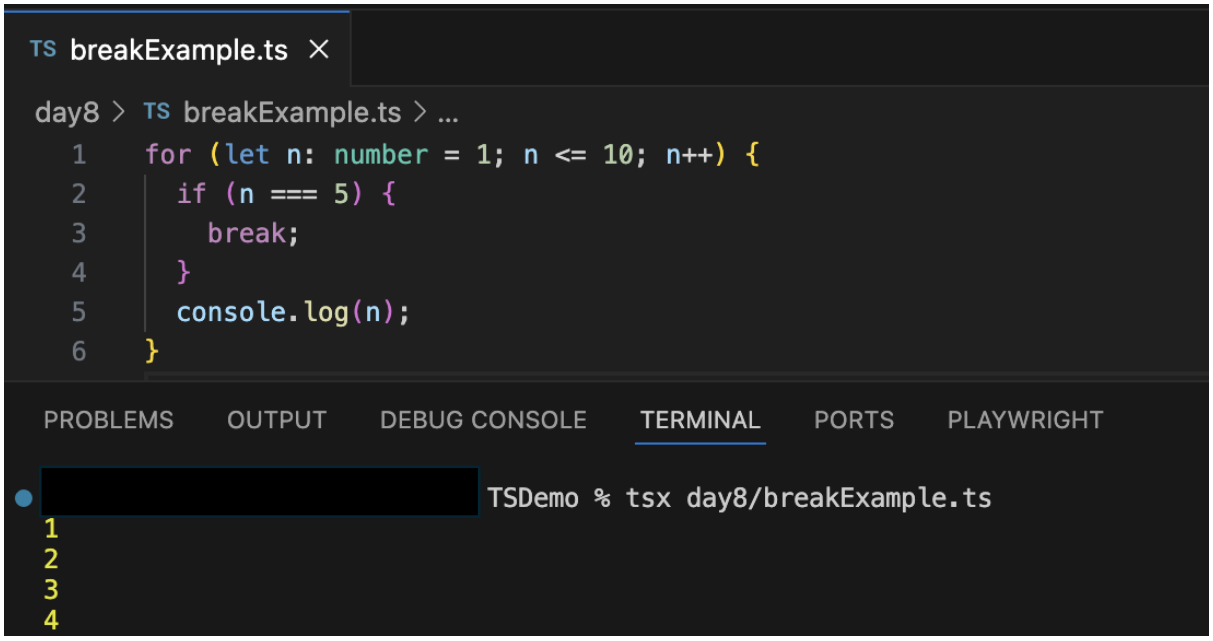
Used inside **loops** or **switch statements** when you want to **stop execution** as soon as a particular condition becomes true.

Syntax

```
for (initialization; condition; inc/dec) {
  if (condition) {
    break;
  }
  // code
}
```

Example 1: (basic loop)

```
for (let n: number = 1; n <= 10; n++) {
  if (n === 5) {
    break;
  }
  console.log(n);
}
```



Explanation:
Here, the loop starts from 1 and runs till 10, but when `n` becomes 5, the `break` statement stops the loop immediately.
Output → 1, 2, 3, 4

Comparison Reminder

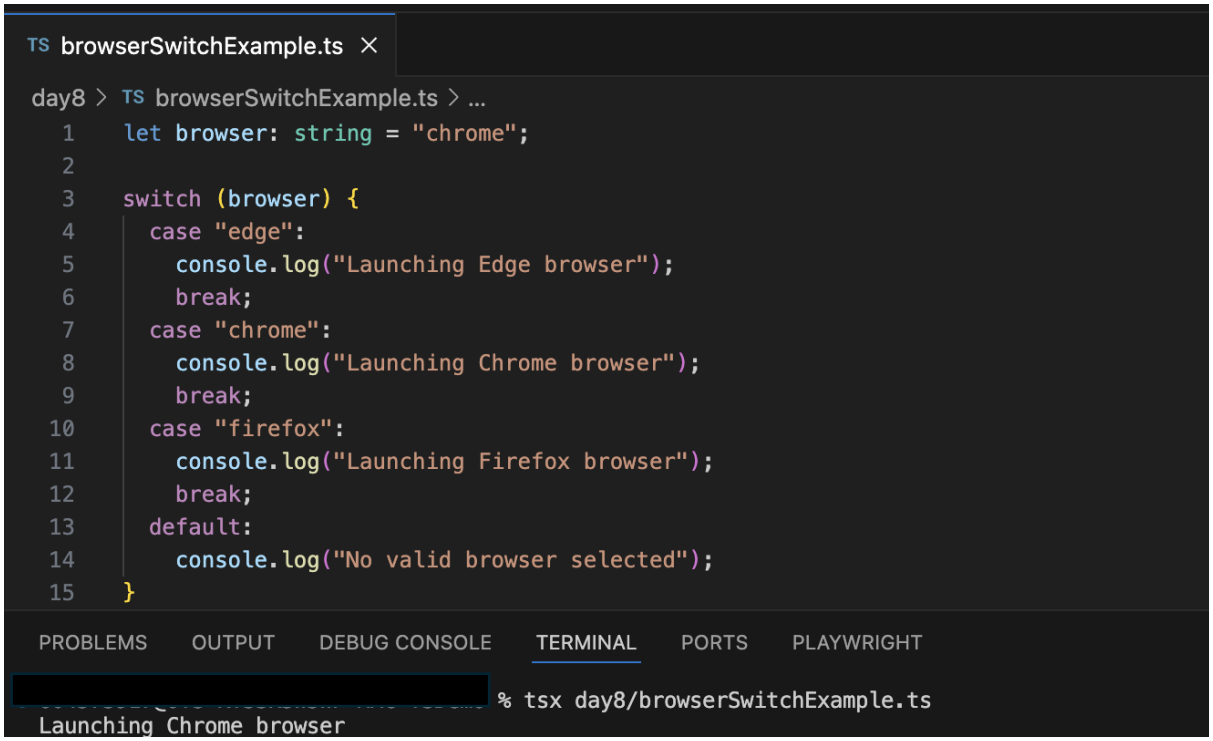
- `==` checks only **value** → `5 == "5"` true
- `===` checks **value + data type** → `5 === "5"` false

Example 2 (Automation Scenario):

Imagine you are automating **browser selection** in a `switch` statement:

```
let browser: string = "chrome";

switch (browser) {
  case "edge":
    console.log("Launching Edge browser");
    break;
  case "chrome":
    console.log("Launching Chrome browser");
    break;
  case "firefox":
    console.log("Launching Firefox browser");
    break;
  default:
    console.log("No valid browser selected");
}
```



Explanation:
Once a matching case is found, the `break` prevents executing the rest of the cases.

2.Continue Statement

"Continue" means **skip** the current iteration and move to the next one.

Why and Where to Use

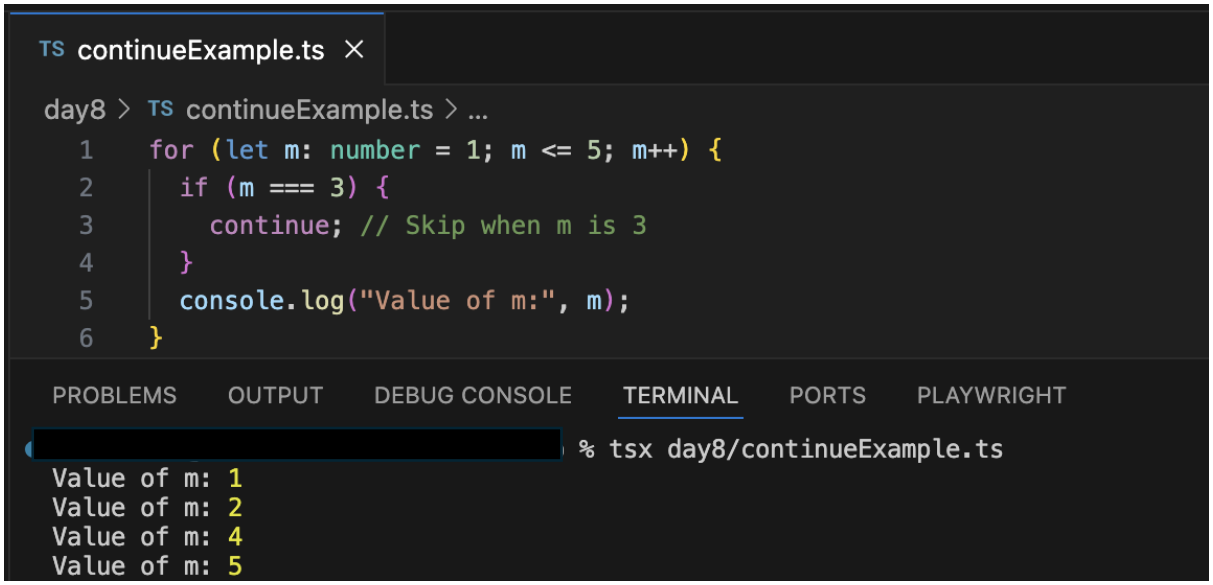
Used inside **loops** when you want to **skip some steps** but still continue the rest of the loop.

Syntax

```
for (initialization; condition; inc/dec) {
  if (condition) {
    continue;
  }
  // code
}
```

Example 1: (basic loop)

```
for (let m: number = 1; m <= 5; m++) {
  if (m === 3) {
    continue; // Skip when m is 3
  }
  console.log("Value of m:", m);
}
```



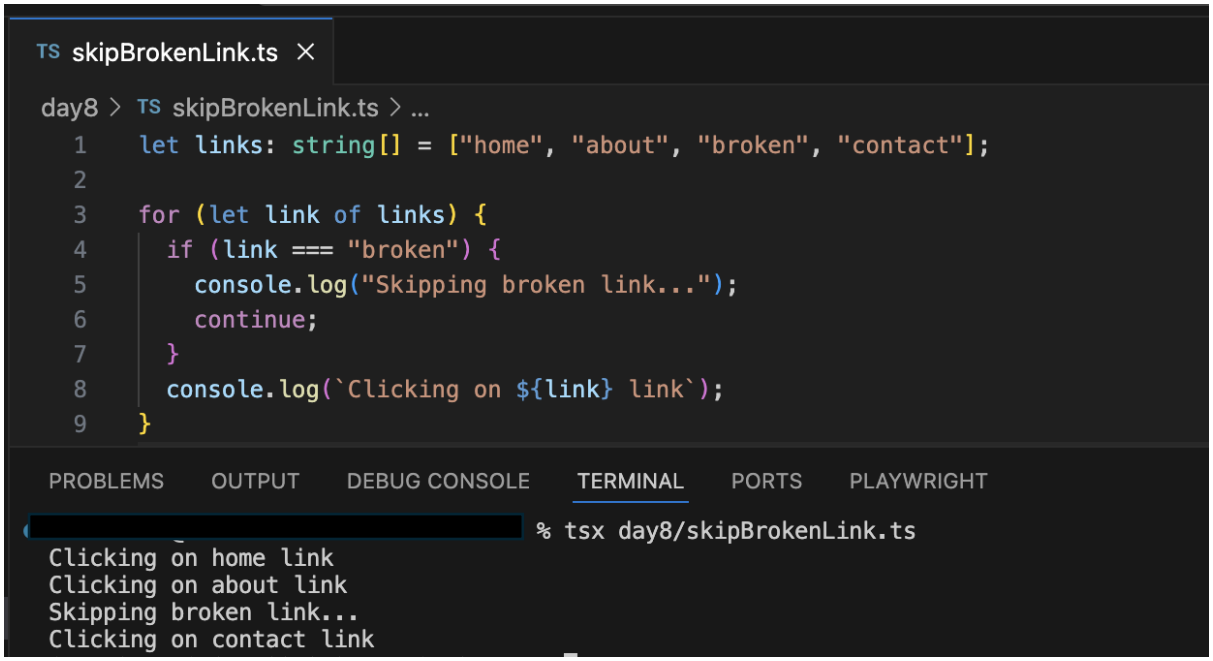
Explanation:
When `m` is 3, that iteration is skipped, so output will be:
→ 1, 2, 4, 5

Example 2 (Automation Scenario):

Imagine you're automating **clicking all links on a page**, but some links are **broken** (status = 404).
You can skip those broken links using `continue`.

```
let links: string[] = ["home", "about", "broken", "contact"];

for (let link of links) {
  if (link === "broken") {
    console.log("Skipping broken link...");
    continue;
  }
  console.log(`Clicking on ${link} link`);
}
```



3.Return Statement

In English, “return” means **to give back** or **go back**.
In TypeScript, the `return` statement **sends a value back** from a function or **stops the execution** of a function.

Why and Where to Use

Used **inside functions** when:

- You want to send a computed result back to the caller.
- You want to stop function execution before it ends naturally.

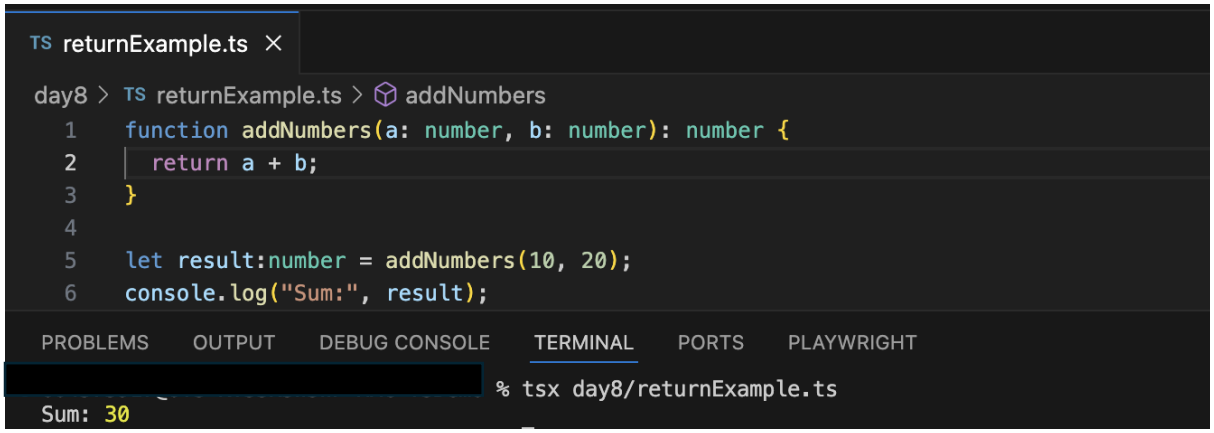
Syntax

```
function functionName(): datatype {
  // code
  return value;
}
```

Example 1:

```
function addNumbers(a: number, b: number): number {
  return a + b;
}

let result = addNumbers(10, 20);
console.log("Sum:", result);
```



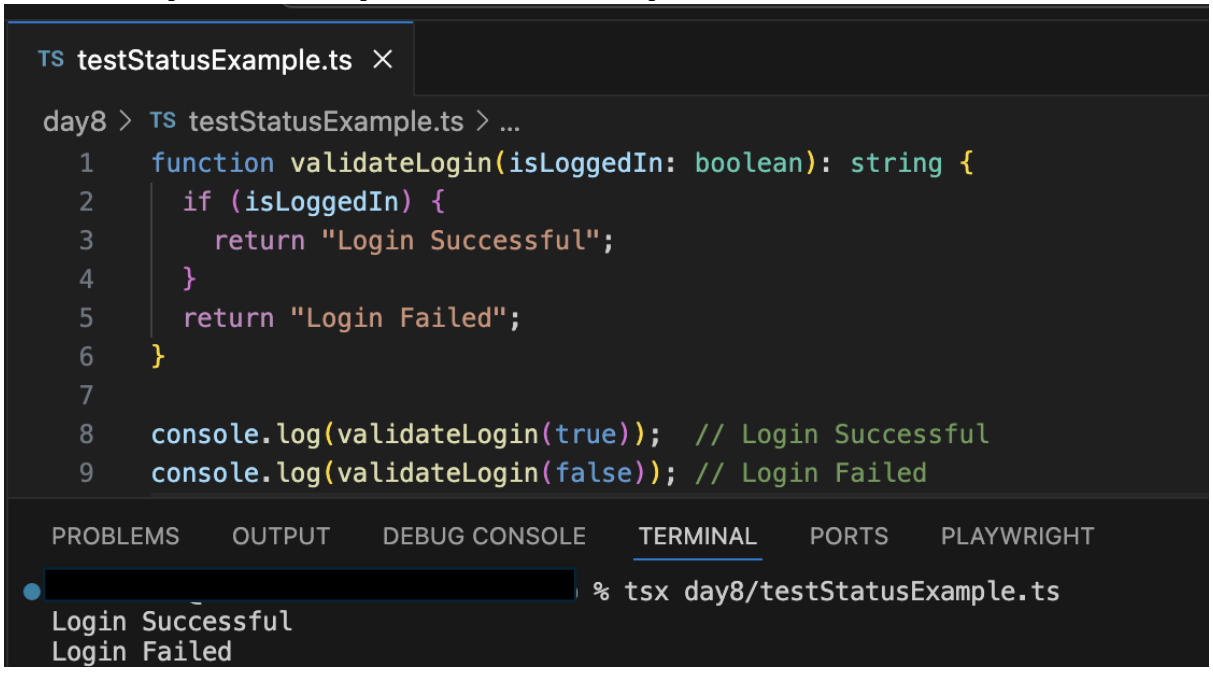
Explanation:
The `return` keyword sends the result `30` back to the caller.

Example 2 (Automation Scenario):

In automation testing, we can use `return` to **return the test status** (like pass or fail) from a function.

```
function validateLogin(isLoggedIn: boolean): string {
  if (isLoggedIn) {
    return "Login Successful";
  }
  return "Login Failed";
}

console.log(validateLogin(true));  // Login Successful
console.log(validateLogin(false)); // Login Failed
```



Let's look closely at this example again:

```
function validateLogin(isLoggedIn: boolean): string {
  if (isLoggedIn) {
    return "Login Successful";
  }
  return "Login Failed";
}

console.log(validateLogin(true));  // Login Successful
console.log(validateLogin(false)); // Login Failed
```

Understanding the Example

If you see clearly, this looks a little confusing at first.
The function `validateLogin()` has a **parameter of type boolean**, but its **return type is string**.
So, how can a function that accepts a boolean value return a string message? Let's break it down.

- The **parameter** `isLoggedIn` only tells us *what kind of input* the function expects — in this case, `true` or `false`.
- The **return type** (`string`) defines *what kind of output* the function will send back once its job is done.

Inside the function, based on the condition, we are returning one of the two string messages:

- `"Login Successful"` if the boolean `isLoggedIn` is true
- `"Login Failed"` if the boolean is false

So, input and output can be of **different types** — there's no rule saying they must match.

Datatype vs Return Type

Till now, we have been talking about **data types** — number, string, boolean, etc.
But when it comes to functions, we start hearing about **return types**.
Let's make the difference clear:

Concept	Meaning	Example
Datatype	The kind of data a variable can hold	`let age: number = 25;`
Return type	The kind of data a function gives back when it completes	`function getAge(): number { return 25; }`

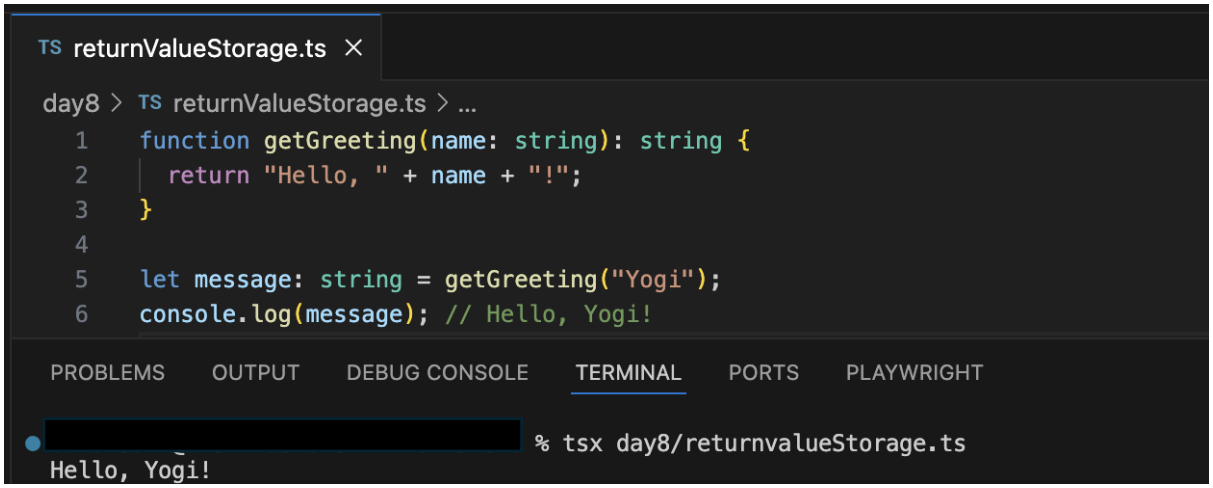
So, the datatype tells us what a variable can *store*, and the return type tells us what a function can *send back*.

How to Store a Function's Returned Value

When we reuse a function, the result that comes from the function can be stored inside a variable.
This is how we reuse outputs easily across the program.

```
function getGreeting(name: string): string {
  return "Hello, " + name + "!";
}

let message: string = getGreeting("Yogi");
console.log(message); // Hello, Yogi!
```



Here,

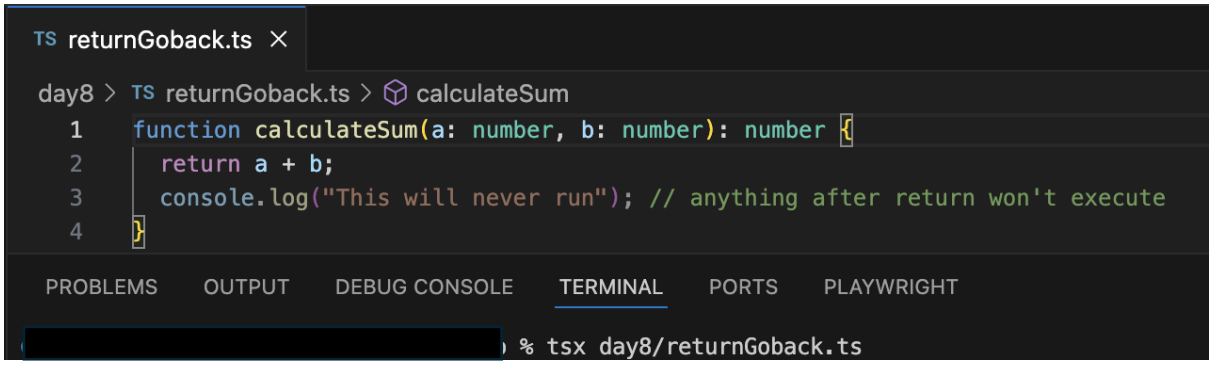
- `getGreeting()` returns a string.
- We stored that return value inside the variable `message`, which is also of type `string`.
- Now we can reuse `message` anywhere in our script.

Why "Return" Means "Go Back"

Think of the `return` statement as a way of *sending the result back to the place where the function was called*.
Once `return` executes, the control **goes back** to the caller — the function immediately stops running and gives its output.

Example:

```
function calculateSum(a: number, b: number): number {
  return a + b;
  console.log("This will never run"); // anything after return won't
execute
}
```



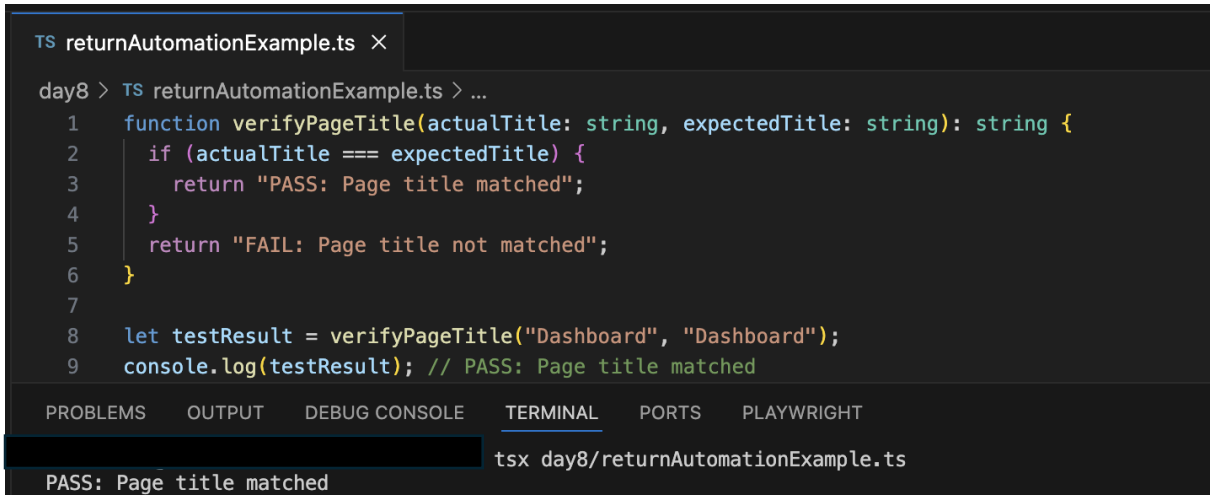
Here, once `return a + b;` runs, the function is done — it "goes back" with the sum to whoever called it.
That's why we call it a **jumping statement** too: it jumps out of the function immediately.

Real-world Automation Example

In automation testing, you might use a function to check the title of a web page and return the test result:

```
function verifyPageTitle(actualTitle: string, expectedTitle: string):
string {
  if (actualTitle === expectedTitle) {
    return "PASS: Page title matched";
  }
  return "FAIL: Page title not matched";
}

let testResult = verifyPageTitle("Dashboard", "Dashboard");
console.log(testResult); // PASS: Page title matched
```



Here, `verifyPageTitle()` takes two strings and returns a message string.
The return type is string because that's what the function is giving back.
The values we pass in are also strings, but that's not mandatory — input and output types can differ based on our need.

Summary Table

Jump Statement	Action	Commonly Used In
break	Stops the loop or switch immediately	Loops, Switch
continue	Skips current iteration and moves to next	Loops
return	Ends function and sends value back	Functions

Questions (8 Total)

1. What is the purpose of jumping statements in TypeScript?
2. What are the three main types of jumping statements?
3. What does the `break` statement do inside a loop?
4. What is the key difference between `break` and `continue`?
5. In which scenarios is `continue` most commonly used?
6. What does the `return` statement do in a function?
7. Can `return` be used outside of a function? Why or why not?
8. What is the difference between == and === in TypeScript?

Answers

1. Jumping statements change the normal flow of program execution by skipping or stopping parts of code.
2. The three jumping statements are **break**, **continue**, and **return**.
3. The `break` statement immediately exits the loop when a condition is met.
4. `break` stops the entire loop; `continue` skips only the current iteration.
5. `continue` is used when we want to skip specific cases (like invalid data or broken links) but still continue looping.
6. The `return` statement ends the function and optionally sends a value back to the caller.
7. No, `return` cannot be used outside a function — it's meant only for function control flow.
8. == compares only values; === compares both values and data types (strict equality).
