TypeScript Basics for Automation Testers – Day 8 (Part 2)

Topic: The Return Statement in TypeScript

The **return statement** — this is where most people (including me once!) got completely stuck when learning functions.
You write a function, you call it and suddenly it "returns something"… but what exactly is it returning, and where does it go?

Meaning of Return Statement

The word **"return"** in means *to give back* or *to go back to where you came from.*
In TypeScript, the **return statement** gives back a value from a function and stops the function's execution right there.

So, whenever your function finishes its job, you can tell it,
"Okay, now give me your result back!" — and that's exactly what `return` does.

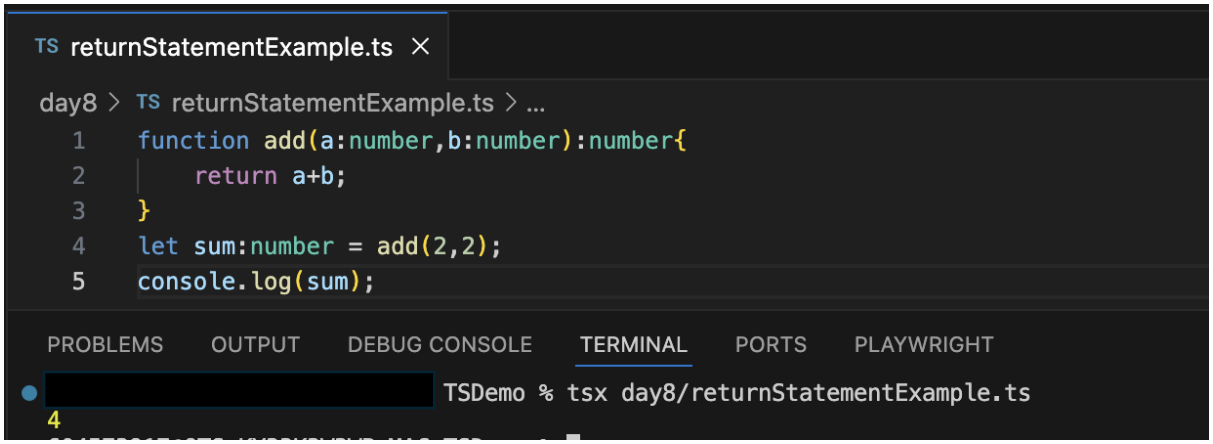
Why We Use Return

We use the `return` statement when we want the function to send a result back to the part of the program that called it.
Without `return`, a function will just perform an action but won't give anything back.

Example:

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

let sum = add(3, 5);
console.log(sum); // 8
```



Here,

- The function **returns** the result of addition.
- The result (`8`) is **sent back** to where the function was called.
- That value gets **stored** inside `sum`.

How "Return" Works

Think of your function as a delivery boy.
When you call the function, you give it an order (some input).
When the delivery boy finishes the job, he comes back and gives you your item (the output).

That "coming back" part is what `return` does.

So when the function hits `return`, it immediately jumps back to the caller, taking the value with it.

Understanding the `return` Statement in TypeScript

The `return` statement is used in a function to send a value back to the caller.
When a function executes a `return` statement, it immediately stops running and outputs the specified value.

Syntax

```
return expression;
```

The **expression** can be any of the following:

- A variable
- A constant value
- A computed expression
- Another function call

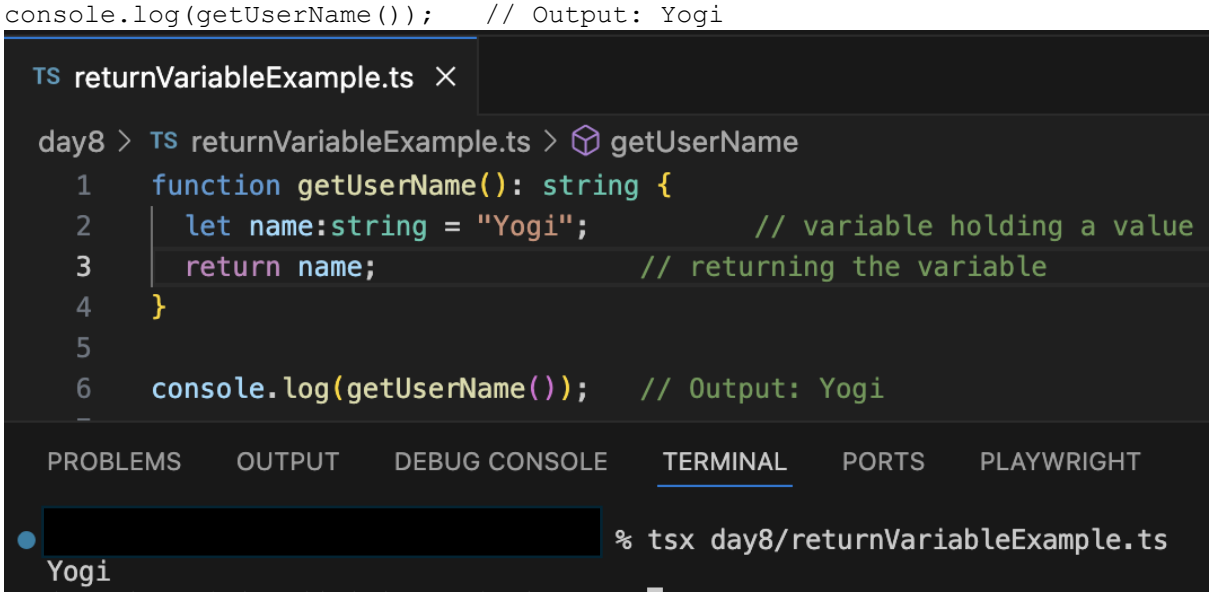
Let’s look at each with clear examples.

Returning a Variable

If a variable holds the value you want to send back, you can directly return that variable.

```
function getUserName(): string {
  let name = "Yogi";          // variable holding a value
  return name;                // returning the variable
}

console.log(getUserName());   // Output: Yogi
```



Explanation:

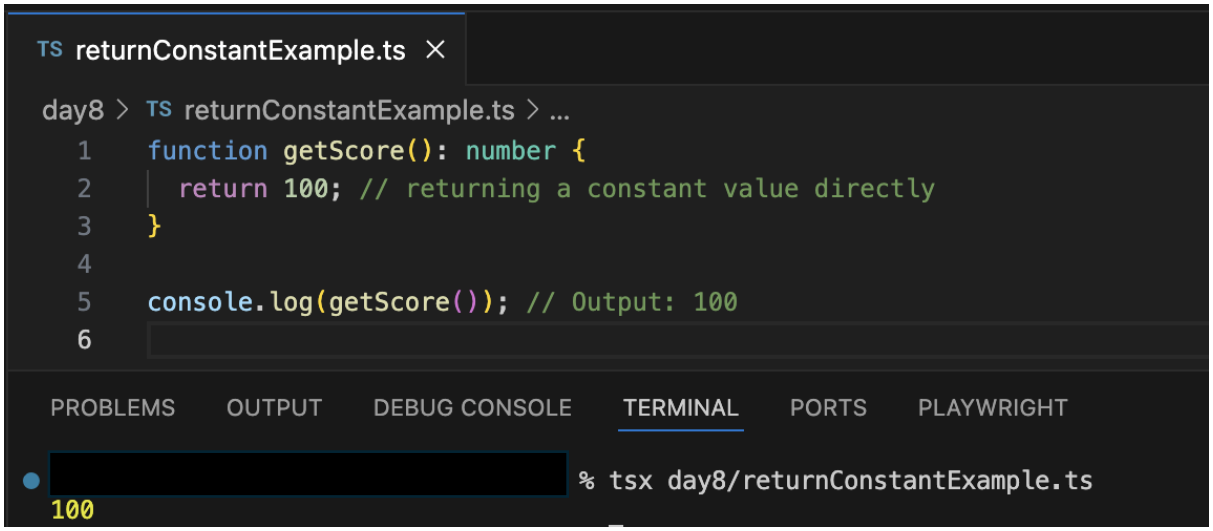
- The variable `name` stores `"Yogi"`.
- `return name;` sends the stored value `"Yogi"` back to the function caller.

Returning a Constant Value

You can also return a fixed or literal value directly, without storing it in a variable.

```
function getScore(): number {
  return 100; // returning a constant value directly
}

console.log(getScore()); // Output: 100
```



Explanation:

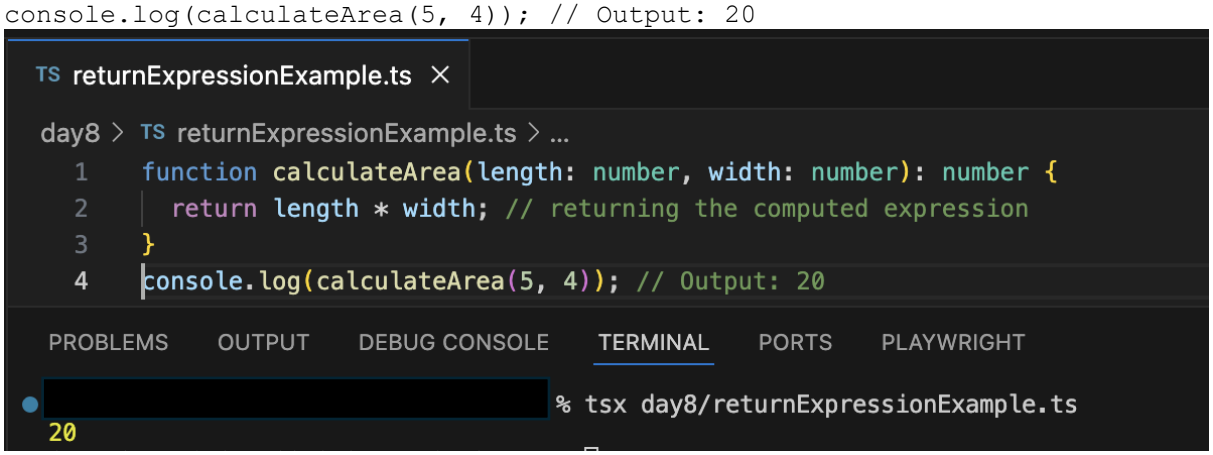
- No variable is needed.
- The function directly returns the constant `100`.

Returning a Computed Expression

A function can return the result of a calculation or an evaluated expression.

```
function calculateArea(length: number, width: number): number {
  return length * width; // returning the computed expression
}

console.log(calculateArea(5, 4)); // Output: 20
```



Explanation:

- The multiplication `length * width` is computed first.
- The result (`20`) is then returned to the caller.

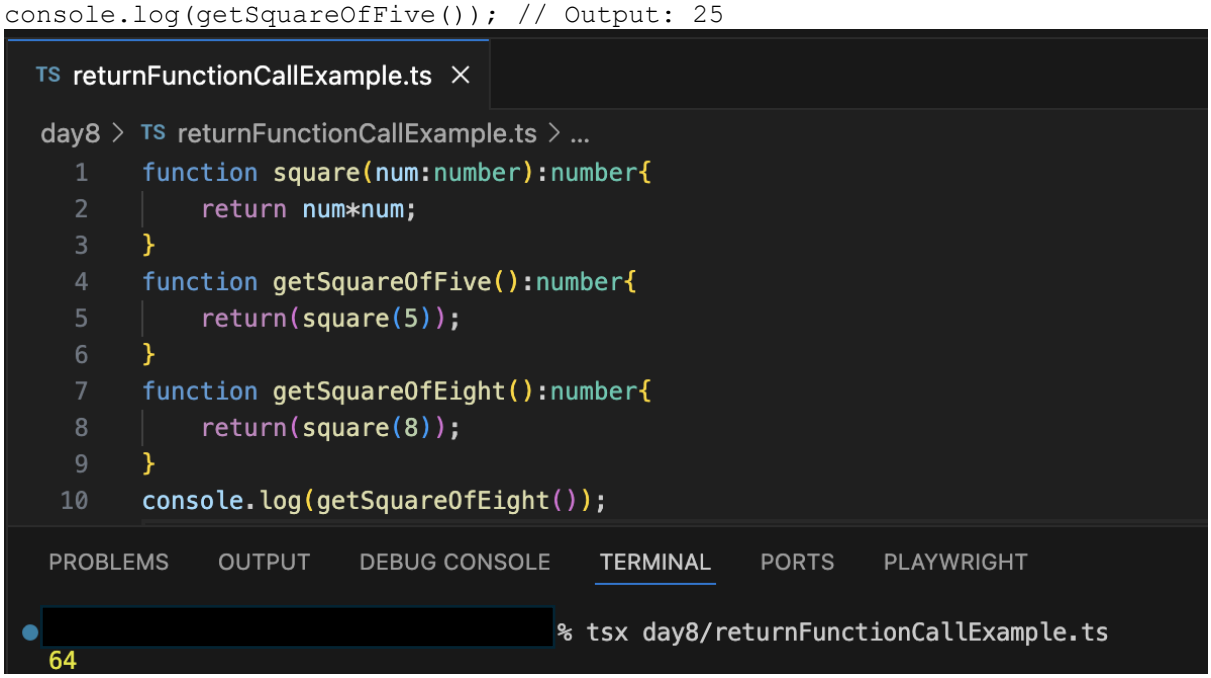
Returning Another Function Call

You can call another function and return its result instead of storing it first.

```
function square(num: number): number {
  return num * num;
}

function getSquareOfFive(): number {
  return square(5); // returning another function's result
}

console.log(getSquareOfFive()); // Output: 25
```



Explanation:

- The inner function `square(5)` returns `25`.
- The outer function `getSquareOfFive()` then returns that same value.

Additional Notes

- A function can have **only one `return` executed** during its call (once it returns, the function stops running).
- If no return statement is specified, the function returns `undefined` by default.
- You can return **any data type** — number, string, boolean, object, array, or even another function.

Type of Return	Description	Example	Returned Value
Variable	Returns a value stored in a variable	`return name;`	`"Yogi"`
Constant	Returns a fixed literal	`return 100;`	`100`
Computed Expression	Returns result of a calculation	`return length * width;`	`20`
Function Call	Returns output of another function	`return square(5);`	`25`

Java vs TypeScript Comparison

Concept	Java	TypeScript
Function definition	`int add(int a, int b) { return a + b; }`	`function add(a: number, b: number): number { return a + b; }`
Type declaration	Declared before variable name	Declared after variable name with `:`
Return type enforcement	Strong (compile-time)	Optional, but recommended for clarity
Return syntax	Same (`return expression;`)	Same (`return expression;`)

Return Type vs Data Type

Till now, we've learned about **data types** like `number`, `string`, and `boolean`.
Now we're talking about **return types** — but don't worry, both are related.

Concept	What It Means	Example
Data type	Defines what type of data a variable holds	`let age: number = 25;`
Return type	Defines what kind of value a function gives back	`function getAge(): number { return 25; }`

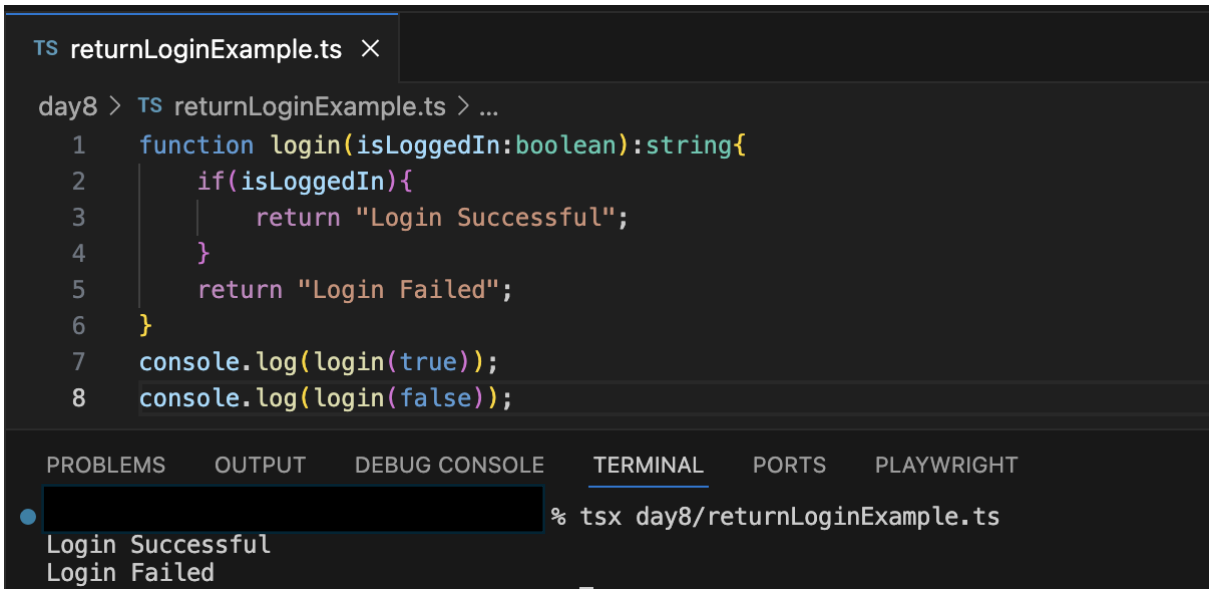
So the **return type** is like a promise that the function makes — "Hey, I'll give back this type of value."

Example: Boolean Parameter, String Return

This is the one that usually confuses most people.

```
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
```



At first glance, you might think:
"Wait, the function takes a **boolean** input but returns a **string**? Is that even allowed?"
Yes, absolutely.

A function's **input type** (parameter) and **output type** (return type) can be completely different.
That's the beauty of return — it only cares about what value it's giving back, not what it received.

Returning Values vs Printing Values

Another common mistake — people confuse `return` and `console.log`.
They are *not the same thing.*

- `console.log` just **prints** the value to the console.
- `return` actually **sends** the value back to the caller.

Example:

```
function greet(): string {
  console.log("Inside the function");
  return "Hello from the function";
}
```

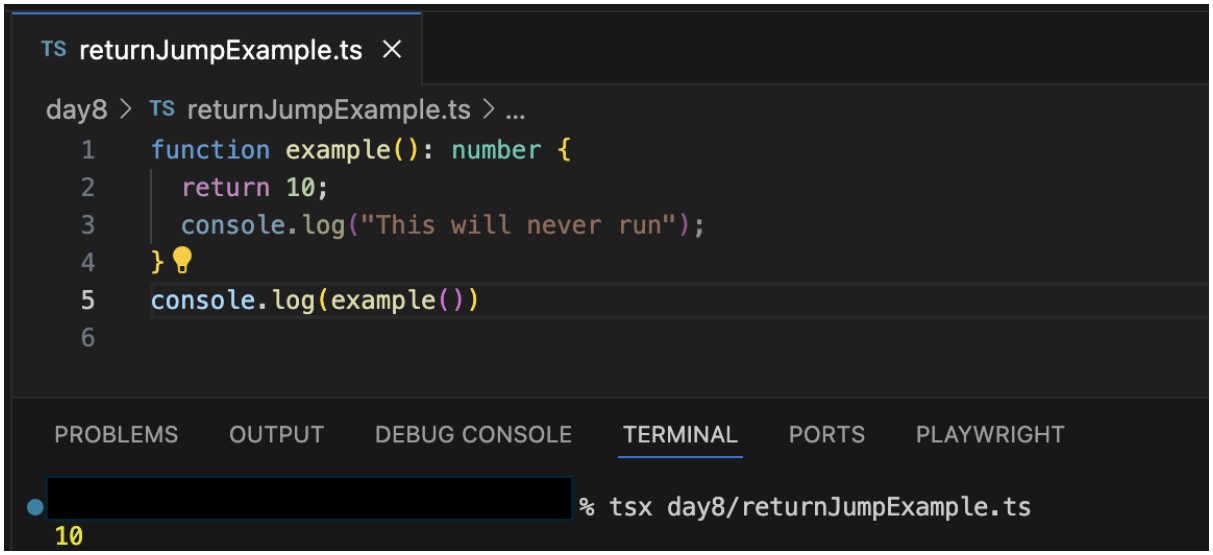
```
let message = greet();
console.log(message);
```

Here,
`console.log("Inside the function")` only prints on the console,
but `return "Hello from the function";` actually sends the message back to the variable `message`.

Why Return is Called a "Jumping" Statement

Because once `return` runs, the control jumps out of the function — nothing after it executes.

```
function example(): number {
  return 10;
  console.log("This will never run");
}
```



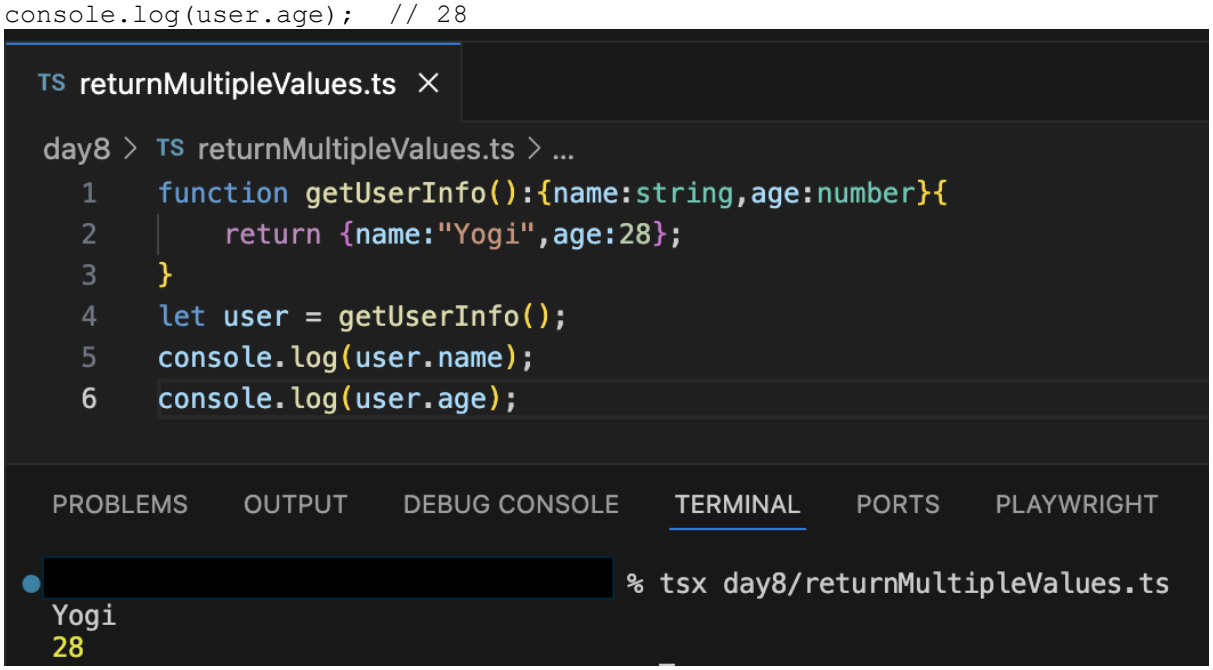
The line after `return` is ignored — the function has already "jumped out."

Returning Multiple Values (Using Object or Array)

You cannot return two separate values directly,
but you can wrap them inside an array or object.

```
function getUserInfo(): { name: string; age: number } {
  return { name: "Yogi", age: 28 };
}

let user = getUserInfo();
console.log(user.name); // Yogi
console.log(user.age);  // 28
```



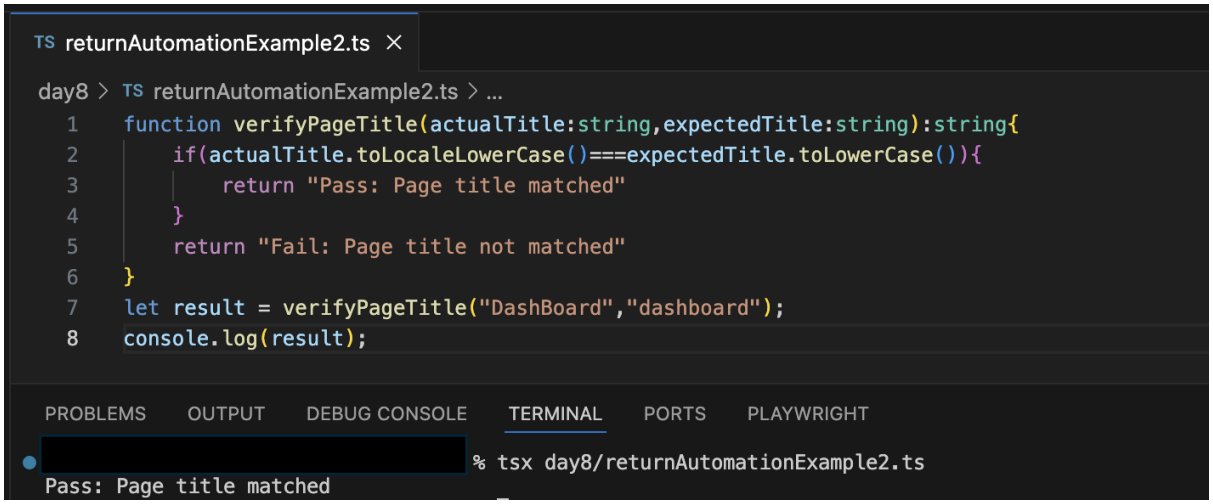
Return in Automation Testing

In automation scripts, we often use `return` to pass results or statuses between functions.

Example:

```
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, `return` helps us reuse the test result later in the script or report it to a logger.

Note

When I first learned `return`, I used to imagine it like this:
If functions were people, `return` would be the one saying —
"Okay, I'm done here… I'm going home with your result!"
And that's exactly what it does.

Questions

1. What does the return statement?
2. Why do we use the return statement in TypeScript?
3. What happens when the return statement is executed inside a function?
4. How is return different from console.log?
5. Can input (parameter) type and return type be different?
6. Why is return considered a jumping statement?
7. How can we return multiple values from a function?
8. How is return used in automation testing?

Answers

1. The return statement means giving a value back to whoever called the function.
2. We use it when we need to reuse the result or pass data between functions.
3. When return executes, the function stops immediately and sends its result to the caller.
4. Return and console.log are different — return sends back the value; console.log just displays it.
5. Yes, input type and return type can be different (like boolean → string).
6. Return is a jumping statement because it ends the function and jumps out instantly.
7. We can return multiple values by wrapping them inside an object or array.
8. In automation testing, return is used to send pass/fail status, validation results, or response data back from a function.
