

TypeScript for Automation Testers – Day 9

Topic: Functions in TypeScript (Part 1)

What is a Function?

Let's start simple — think of a **function** as a *small machine that performs a specific task whenever you ask it to.*

A function is a block of code designed to perform a particular operation.

You write the function once and **reuse it whenever needed**, instead of rewriting the same logic again and again.

Why, Where, and When Do We Use Functions in TypeScript?

- **Why:** To avoid repeating the same code multiple times.
- **Where:** Wherever you need to perform a repeated logic or operation (for example, login validation, clicking elements, or performing API calls).
- **When:** Whenever you notice a repeated pattern or operation in your program, convert that logic into a reusable function.

How Can We Reuse a Function?

We reuse a function by **calling it**.

When we call a function, the control jumps to that function's block, executes it and returns back once it's done.

Calling a Function in Java vs TypeScript

Let's understand this with both languages because the way we **call** functions slightly differs.

Language	How We Define	How We Call
Java	Functions are written inside a class and called methods .	You create an object of that class and call the method using that object. Example: <code>obj.login();</code>
TypeScript	Functions can exist independently (outside class) or inside class .	You can call them directly by name (if outside a class) or using an object (if inside a class).

Example – Function in TypeScript

Let's say we want to add two numbers.

```
function add(a: number, b: number): number {  
    return a + b;  
}  
  
console.log(add(5, 10)); // Output: 15
```



```
ts functionAddNumbers.ts ×  
day9 > ts functionAddNumbers.ts > ...  
1   function add(a: number, b: number): number {  
2     return a + b;  
3   }  
4  
5   console.log(add(5, 10)); // Output: 15  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS PLAYWRIGHT  
● TSDemo % tsx day9/functionAddNumbers.ts  
15
```

Here,

- `function` is the keyword.
- `add` is the function name.
- `number` is a return type of the function `add`
- `a` and `b` are parameters.
- `return` gives back the result.
- `console.log(add(5,10))` is where we **call** the function.

Real-Life Example in Automation Testing

Web Automation:

Imagine you often click on a login button across different test cases.

Instead of repeating `driver.findElement(...).click()` every time, you can write a function like:

```
function clickLoginButton() {  
    console.log("Clicked the login button");  
}  
  
// Reuse the function whenever needed  
clickLoginButton();
```

The screenshot shows a code editor with a terminal window at the bottom. The terminal window has a blue header bar with the text "TSDemo % tsx day9/functionLoginAction.ts". Below the header, the terminal output shows the message "Clicked the login button". The terminal tab is highlighted with a blue underline.

```
TS functionLoginAction.ts ×  
day9 > TS functionLoginAction.ts > ...  
1   function clickLoginButton() {  
2     |   console.log("Clicked the login button");  
3   }  
4  
5   // Reuse the function whenever needed  
6   clickLoginButton();  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS PLAYWRIGHT  
● TSDemo % tsx day9/functionLoginAction.ts  
Clicked the login button
```

API Automation:

If you frequently send a GET request, you can create one function to handle it:

```
function getAPIResponse(url: string): void {  
  console.log(`Sending GET request to ${url}`);  
}  
  
// Call function wherever needed  
getAPIResponse("https://api.testserver.com/users");
```

The screenshot shows a code editor with a terminal window at the bottom. The terminal window has a blue header bar with the text "TSDemo % tsx day9/functionApiGetCall.ts". Below the header, the terminal output shows the message "Sending get request to https://api.testserver.com/users". The terminal tab is highlighted with a blue underline.

```
TS functionApiGetCall.ts ×  
day9 > TS functionApiGetCall.ts > ...  
1   function getAPIResponse(url: string): void {  
2     |   console.log(`Sending GET request to ${url}`);  
3   }  
4  
5   // Call function wherever needed  
6   getAPIResponse("https://api.testserver.com/users");  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS PLAYWRIGHT  
● TSDemo % tsx day9/functionApiGetCall.ts  
Sending get request to https://api.testserver.com/users
```

Function vs Method — Are They the Same?

They sound similar but are not exactly the same.

Term	Where It Exists	How It's Used
Function	Standalone block of code (can exist outside classes).	Can be called directly using its name.
Method	A function inside a class or object .	Must be called using the class/object name.

Example:

```
// Function
function greet() {
    console.log("Hello from a function!");
}
greet(); // Function call

// Method inside a class
class Greeter {
    sayHello() {
        console.log("Hello from a method!");
    }
}

let obj = new Greeter();
obj.sayHello(); // Method call
```

The screenshot shows a code editor with a dark theme. A file named `functionVsMethod.ts` is open, containing the provided TypeScript code. The code defines a function `greet` and a class `Greeter` with a method `sayHello`. Both methods log a message to the console. The code editor has a terminal tab at the bottom where the command `tsx day9/functionVsMethod.ts` is run, resulting in the output "Hello from a function!" followed by "Hello from a method!".

```
TS functionVsMethod.ts ×

day9 > TS functionVsMethod.ts > ...
1  // Function
2  function greet() {
3      console.log("Hello from a function!");
4  }
5  greet(); // Function call
6
7  // Method inside a class
8  class Greeter {
9      sayHello() {
10         console.log("Hello from a method!");
11     }
12 }
13
14 let obj = new Greeter();
15 obj.sayHello(); // Method call

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    PLAYWRIGHT
● [redacted]: TSDemo % tsx day9/functionVsMethod.ts
Hello from a function!
Hello from a method!
```

Parameters vs Arguments

These two words are often confusing, but here's the simple difference:

- **Parameters:** The names you define in the function definition.
- **Arguments:** The actual values you pass when calling the function.

Example:

```
function multiply(x: number, y: number) { // x, y are PARAMETERS
    console.log(x * y);
}
```

```
multiply(3, 4); // 3, 4 are ARGUMENTS
```

The screenshot shows a terminal window with the following content:

```
TS parametersVsArguments.ts ×

day9 > TS parametersVsArguments.ts > ...
1   function multiply(x: number, y: number) { // x, y are PARAMETERS
2     console.log(x * y);
3   }
4
5   multiply(3, 4); // 3, 4 are ARGUMENTS
```

Below the code, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and PLAYWRIGHT. The TERMINAL tab is selected. At the bottom, it shows the command: TSDemo % tsx day9/parametersVsArguments.ts and the number 12.

Types of Functions in TypeScript

Now that you know what a function is, let's look at the three basic types in TS.

1. Named Function

A function with a name that you can call directly.

```
function greetUser() {
    console.log("Welcome, Tester!");
}
greetUser();
```

2. Anonymous Function (Nameless Function)

A function that doesn't have a name, often stored inside a variable.

```
let greet = function() {
    console.log("Hello from an anonymous function!");
};
greet();
```

3. Arrow Function (Lambda Function)

A compact way to write functions, introduced in ES6.

```
let greet = () => console.log("Hello from an arrow function!");
greet();
```

Java vs TypeScript – Function Comparison

Concept	Java	TypeScript
Function Declaration	Must be inside a class	Can be standalone or inside a class
Function Keyword	void / public static etc.	function keyword
Function Overloading	Supported	Supported (with signature matching)
Arrow Function	Not available	Available
Function Reuse	Through object reference	Direct or through object reference

Questions

1. What is a function in TypeScript?
 2. Why do we use functions in programming?
 3. How is a function called in Java vs TypeScript?
 4. What is the difference between a function and a method?
 5. What are parameters and arguments in a function?
 6. What are the three types of functions in TypeScript?
 7. What is an arrow (lambda) function used for?
 8. Give a real-world example of using a function in test automation.
-

Answers

1. A function is a reusable block of code that performs a specific task when called.
 2. Functions reduce code duplication and improve reusability.
 3. In Java, functions are called through class objects; in TypeScript, they can be called directly or through an object.
 4. A function is standalone, while a method exists inside a class or object.
 5. Parameters are placeholders in function definition; arguments are actual values passed during function call.
 6. Named function, Anonymous function, and Arrow function.
 7. Arrow functions provide a shorter syntax for defining functions and preserve `this` context.
 8. Example: Creating a reusable `login()` function to automate login actions across multiple test cases.
-