Functions

In TypeScript, a function is a block of code that performs a specific task, and it can be invoked or called from other parts of your program. Functions in TypeScript can be typed, meaning you can specify the types of the function's parameters and its return type. This helps catch errors at compile time and makes your code more predictable and easier to understand.

Here is a basic example of a function in TypeScript:

// Function declaration with typed parameters and return type

function greet(name: string): string {

return `Hello, ${name}!`;

}

// Calling the function

let message: string = greet("Alice");

console.log(message); // Output: Hello, Alice!

**Key Points:**

1. **Function Declaration**: function greet(name: string): string declares a function named greet that takes a parameter name of type string and returns a string.
2. **Return Type**: The return type of the function is specified after the parameter list, separated by a colon. In this case, the function returns a string.
3. **Function Call**: greet("Alice") calls the function with the argument "Alice", and the returned value is assigned to the variable message.

**Example with Optional and Default Parameters:**

You can also define functions with optional and default parameters.

// Function with an optional parameter

function greet(name: string, greeting?: string): string {

if (greeting) {

return `${greeting}, ${name}!`;

} else {

return `Hello, ${name}!`;

}

}

let message1: string = greet("Alice");

console.log(message1); // Output: Hello, Alice!

let message2: string = greet("Bob", "Hi");

console.log(message2); // Output: Hi, Bob!

In this example:

* **Optional Parameter**: greeting?: string indicates that the greeting parameter is optional. If it is not provided, the function will use the default greeting "Hello".

// Function with a default parameter

function greet(name: string, greeting: string = "Hello"): string {

return `${greeting}, ${name}!`;

}

let message3: string = greet("Alice");

console.log(message3); // Output: Hello, Alice!

let message4: string = greet("Bob", "Hi");

console.log(message4); // Output: Hi, Bob!

In this example:

* **Default Parameter**: greeting: string = "Hello" sets a default value of "Hello" for the greeting parameter. If the caller does not provide this argument, the default value will be used.

**Example with Rest Parameters:**

You can also use rest parameters to handle functions with an arbitrary number of arguments.

// Function with rest parameters

function sum(...numbers: number[]): number {

return numbers.reduce((total, num) => total + num, 0);

}

let result: number = sum(1, 2, 3, 4, 5);

console.log(result); // Output: 15

In this example:

* **Rest Parameters**: ...numbers: number[] allows the function to accept any number of number arguments, which are available as an array within the function.
* **Array Method**: The reduce method is used to sum up all the numbers in the array.

Returning function values

In TypeScript, you can define functions that return values of specific types. This can be specified using the : type syntax after the parameter list. Below are several examples demonstrating how to return values from functions, including simple values, objects, and other functions.

### Simple Value Return

A function that returns a simple value such as a string or number.

function add(a: number, b: number): number {

return a + b;

}

let result: number = add(5, 3);

console.log(result); // Output: 8

### Returning an Object

A function can return an object with a specified type.

interface Person {

name: string;

age: number;

}

function createPerson(name: string, age: number): Person {

return { name, age };

}

let person: Person = createPerson("Alice", 30);

console.log(person); // Output: { name: 'Alice', age: 30 }

### Returning a Function

A function can return another function. This is useful for creating higher-order functions.

function multiplier(factor: number): (x: number) => number {

return (x: number) => x \* factor;

}

let double = multiplier(2);

console.log(double(5)); // Output: 10

**Asynchronous Function (Promise Return Type)**

A function that performs asynchronous operations and returns a Promise.

function fetchData(url: string): Promise<string> {

return new Promise((resolve, reject) => {

setTimeout(() => {

// Simulate an asynchronous operation

resolve(`Data from ${url}`);

}, 2000);

});

}

fetchData("https://example.com").then((data) => {

console.log(data); // Output: Data from https://example.com

});

### Generic Function

A function can use generics to return values of different types based on the provided arguments.

function identity<T>(arg: T): T {

return arg;

}

let str = identity<string>("Hello");

let num = identity<number>(42);

console.log(str); // Output: Hello

console.log(num); // Output: 42

### Summary

TypeScript provides robust type-checking capabilities for function return values, ensuring that the values returned from functions adhere to specified types. This helps in catching errors early and makes the code more predictable and easier to maintain.