TypeScript Guide : Functions And Its Parameters

EP: 05

Ariba M.
@frontendcharm

Ariba M.
@frontendcharm

Keep Swiping...

• TypeScript Functions

Functions in TypeScript are blocks of code that can be called to perform specific tasks. They allow the organization of reusable and maintainable code.

demo.ts

```
//Syntax:
function functionName(param1: type, param2: type):
returnType {
    // function body
}


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

console.log(add(5, 3)); // Output: 8
```

• Function Types

Function types are a way to specify the types of the parameters and return value of a function.

A function type has two parts: parameters and return type.

demo.ts

```
//Syntax:
let myFunction: (param1: type, param2: type) ⇒
returnType;


// Example:
let add: (a: number, b: number) ⇒ number;

add = (x, y) ⇒ {
    return x + y;
};
console.log(add(5, 3)); // Output: 8
```

• Optional Parameters

Optional parameters allow functions to be called with varying numbers of arguments. Optional parameters are denoted by a question mark ? after the parameter name.

demo.ts

```typescript
//Syntax:
function functionName(param1: type, param2?: type):
returnType {
    // function body
}
```

demo.ts

```typescript
// Example:
function greet(name: string, greeting?: string): string {
    if (greeting) {
        return `${greeting}, ${name}!`;
    } else {
        return `Hello, ${name}!`;
    }
}
console.log(greet("Alice"));
// Output: Hello, Alice!

console.log(greet("Alice", "Good morning"));
// Output: Good morning, Alice!
```

• Default Parameters

Default parameters allow parameters to have a default value if no value or undefined is passed. Similar to JavaScript, you can use default parameters in TypeScript with the same syntax:

demo.ts

```
//Syntax:
function functionName(param1: type, param2: type = defaultValue):
returnType {
    // function body
}
```

demo.ts

```
// Example:
function greet(name: string, greeting: string = "Hello"): string {
    return `${greeting}, ${name}!`;
}
console.log(greet("Alice"));
// Output: Hello, Alice!

console.log(greet("Alice", "Good morning"));
// Output: Good morning, Alice!
```

• Rest Parameters

A rest parameter allows a function to accept zero or more arguments of the specified type.

To declare a rest parameter, you prefix the parameter name with three dots and use the array type as the type annotation

demo.ts

```
//Syntax:
function functionName(...params: type[]): returnType {
    // function body
}

// Example:
function getTotal(...numbers: number[]): number {
    let total = 0;
    numbers.forEach((num) => total += num);
    return total;
}

console.log(getTotal()); // 0
console.log(getTotal(10, 20)); // 30
console.log(getTotal(10, 20, 30)); // 60
```

Ariba M.
@frontendcharm

Keep Swiping...

• Function Overloading

Function overloading allows the creation of multiple function signatures for a single function, enabling different type combinations of parameters. It allows you to establish the relationship between the parameter types and result types of a function.

demo.ts

```typescript
//Syntax:
function functionName(param1: type): returnType;
function functionName(param1: type, param2: type): returnType;
function functionName(param1: any, param2?: any): any {
    // function body
}

// Example:
function add(a: number, b: number): number;
function add(a: string, b: string): string;
function add(a: any, b: any): any {
   return a + b;
}

console.log(add(5, 3)); // Output: 8
console.log(add("Hello, ", "world!")); // Output: Hello, world!
```

• Arrow Functions

Arrow functions in TypeScript provide a concise syntax for writing functions and come with features like lexical `this` binding, which can simplify working with functions, especially in object-oriented programming and callback scenarios.

demo.ts

```typescript
//Syntax:
(param1: Type1, param2: Type2): ReturnType => {
    // function body
}

// single-line functions
(param1: Type1, param2: Type2): ReturnType => expression;

// Example:
const add = (a: number, b: number): number => {
    return a + b;
};

console.log(add(5, 3)); // Output: 8
```