**Function**

**Type Annotations With Function**

Type annotations in TypeScript are used to explicitly specify the data types for function parameters and return values. These annotations help both developers and the TypeScript compiler understand and enforce the expected types of values that functions should accept as arguments and return as results. Type annotations for functions are typically added using colons (**:**) within the function's parameter list and after the parameter list to denote the return type.

Here's a simple example illustrating type annotations with functions:

// Function to calculate the area of a rectangle

function calculateRectangleArea(long: number, width: number): number {

return long \* width;

}

// Calling the function with valid arguments

const long = 5;

const width = 3;

const area = calculateRectangleArea(long, width);

console.log(`The area of the rectangle is: ${area}`);

In this revised example:

1. We define a function calculateRectangleArea that takes two parameters: lengthRectAngle and widthOfRectAngle . We use type annotations to specify that both parameters should be of type number. Additionally, we specify that the function should return a value of type number by using : number after the parameter list.
2. We calculate the area of a rectangle by multiplying the lengthRectAngle and widthOfRectAngle parameters and returning the result.
3. We call the calculateRectangleArea function with valid arguments, lengthRectAngle and widthOfRectAngle , which are both numbers.
4. We store the result of the function call in a variable named area and log the calculated area to the console.

**Optional And Default Parameter**

In TypeScript, you can make function parameters optional or provide default values to parameters, allowing for more flexible and convenient function usage. This feature is especially useful when you have functions that can accept a varying number of arguments or when you want to provide default values in case the caller does not supply a particular argument.

Let's break down optional and default parameters:

1. **Optional Parameters:**
   * An optional parameter is one that may or may not be provided when calling a function.
   * To define an optional parameter, you use a ? symbol after the parameter name in the function's parameter list.
   * The TypeScript compiler allows you to call the function with or without providing a value for the optional parameter.

Example:

function greetOptional(name: string, age?: number) {

if (age !== undefined) {

console.log(`Hello, ${name}! You are ${age} years old.`);

} else {

console.log(`Hello, ${name}!`);

}

}

greetOptional("Alice"); // Output: "Hello, Alice!"

greetOptional("Bob", 30); // Output: "Hello, Bob! You are 30 years old."

1. **Default Parameters:**
   * A default parameter is a parameter that automatically takes a default value if no argument is provided during the function call.
   * To set a default value for a parameter, you use the = operator in the parameter list.

Example:

// Default Parameters:

function greetDefault(name: string, age: number = 25) {

console.log(`Hello, ${name}! You are ${age} years old.`);

}

greetDefault("Alice"); // Output: "Hello, Alice! You are 25 years old."

greetDefault("Bob", 30); // Output: "Hello, Bob! You are 30 years old."

In the examples above, the age parameter is optional in the first case, and it defaults to 25 in the second case if not provided.

Using optional and default parameters in your functions gives you greater flexibility when designing your function interfaces, allowing callers to provide only the information they have while ensuring a smooth execution of your code. This feature is valuable for creating functions that can adapt to various scenarios and requirements.

**Function Rest Parameter**

In TypeScript, as well as in JavaScript, a rest parameter allows a function to accept an indefinite number of arguments as an array. This means that you can pass any number of arguments to the function, and they will be collected into an array, which can be accessed within the function. Rest parameters are specified using the spread operator **...** followed by the parameter name in the function's parameter list.

// Function with a rest parameter and type annotations

function addAll(...nums: number[]): number {

let result = 0;

for (let i = 0; i < nums.length; i++) {

result += nums[i];

}

return result;

}

// Calling the function with various numeric arguments

console.log(addAll(10, 20, 30, 100, 10.5, +true)); // Output: 170.5

In this code:

* The addAll function is defined with a rest parameter ...nums, which captures a variable number of numeric arguments and stores them in an array named nums. The type annotation number[] specifies that the elements in the nums array should be of type number.
* Inside the function, a for loop is used to iterate through the nums array and calculate the sum of all the numbers provided as arguments.

pro tip **+true** in JavaScript is a unary plus operator (**+**) applied to the boolean **true**, resulting in the numeric value **1**.

**Arrow Function**

An arrow function in TypeScript is a concise way to define functions using a more compact syntax compared to traditional function expressions. Arrow functions are also sometimes called "fat arrow" functions due to the use of the **=>** syntax. They were introduced in ECMAScript 6 (ES6) and have since become a popular way to write functions in JavaScript and TypeScript.

const addWithArrow = (num1: number, num2: number): number => num1 + num2;

console.log(addWithArrow(10, 20)); // Output: 30

* The addWithArrow variable is assigned an arrow function that performs the same addition operation.

**Anonymous Function**

An anonymous function in TypeScript (and in JavaScript) is a function that doesn't have a name. Instead of declaring a named function, you define the function directly as an expression. Anonymous functions are often used in various scenarios, including passing functions as arguments to other functions or creating functions on the fly. They are particularly useful when you need a small, one-off function without polluting the global scope with a function name.

const add = function (num1: number, num2: number): number {

return num1 + num2;

};

console.log(add(10, 20)); // Output: 30

* The add variable is assigned an anonymous function that adds two numbers, num1 and num2.

**void & never**

In TypeScript, void and never are two distinct types that represent different concepts:

1. **void Type:**
   * The void type is used to indicate the absence of a value. It represents functions that do not return any value or return undefined.
   * It's often used to annotate functions that perform actions or side effects but don't produce a meaningful result that needs to be assigned or captured.

Example:

// void

function greet(name: string): void {

console.log(`Hello, ${name}!`);

}

function noReturnValue(): void {

// This function doesn't return a value (implicitly returns undefined).

}

In the greet function, it logs a message but doesn't return anything, so its return type is void.

1. **never Type:**
   * The never type is used to represent values that never occur. It's typically associated with functions that never return, either because they throw exceptions, enter infinite loops, or have unreachable code.

Example:

// never

function throwError(message: string): never {

throw new Error(message);

}

function infiniteLoop(): never {

while (true) {

// This function never exits (infinite loop).

}

}

In the throwError function, it throws an error and never returns normally. In the infiniteLoop function, it enters an infinite loop and never exits, resulting in a never return type.