Functions are the building blocks of readable, maintainable, and reusable code. A function is a set of statements to perform a specific task. Functions organize the program into logical blocks of code. Once defined, functions may be called to access code. This makes the code reusable. Moreover, functions make it easy to read and maintain the program’s code.

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

|  |  |
| --- | --- |
| **Sr.No** | **Funtions & Description** |
| 1. | [Defining a Function](https://www.tutorialspoint.com/typescript/typescript_defining_function.htm)  A function definition specifies what and how a specific task would be done. |
| 2. | [Calling a Function](https://www.tutorialspoint.com/typescript/typescript_calling_function.htm)  A function must be called so as to execute it. |
| 3. | [Returning Functions](https://www.tutorialspoint.com/typescript/typescript_returning_function.htm)  Functions may also return value along with control, back to the caller. |
| 4. | [Parameterized Function](https://www.tutorialspoint.com/typescript/typescript_parameterized_function.htm)  Parameters are a mechanism to pass values to functions. |

**Optional Parameters**

Optional parameters can be used when arguments need not be compulsorily passed for a function’s execution. A parameter can be marked optional by appending a question mark to its name. The optional parameter should be set as the last argument in a function. The syntax to declare a function with optional parameter is as given below −

function function\_name (param1[:type], param2[:type], param3[:type])

**Example: Optional Parameters**

function disp\_details(id:number,name:string,mail\_id?:string) {

console.log("ID:", id);

console.log("Name",name);

if(mail\_id!=undefined)

console.log("Email Id",mail\_id);

}

disp\_details(123,"John");

disp\_details(111,"mary","mary@xyz.com");

* The above example declares a parameterized function. Here, the third parameter, i.e., mail\_id is an optional parameter.
* If an optional parameter is not passed a value during the function call, the parameter’s value is set to undefined.
* The function prints the value of mail\_id only if the argument is passed a value.

On compiling, it will generate following JavaScript code −

//Generated by typescript 1.8.10

function disp\_details(id, name, mail\_id) {

console.log("ID:", id);

console.log("Name", name);

if (mail\_id != undefined)

console.log("Email Id", mail\_id);

}

disp\_details(123, "John");

disp\_details(111, "mary", "mary@xyz.com");

The above code will produce the following output −

ID:123

Name John

ID: 111

Name mary

Email Id mary@xyz.com

**Rest Parameters**

Rest parameters are similar to variable arguments in Java. Rest parameters don’t restrict the number of values that you can pass to a function. However, the values passed must all be of the same type. In other words, rest parameters act as placeholders for multiple arguments of the same type.

To declare a rest parameter, the parameter name is prefixed with three periods. Any nonrest parameter should come before the rest parameter.

**Example: Rest Parameters**

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

var i;

var sum:number = 0;

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

sum = sum + nums[i];

}

console.log("sum of the numbers",sum)

}

addNumbers(1,2,3)

addNumbers(10,10,10,10,10)

* The function addNumbers() declaration, accepts a rest parameter *nums*. The rest parameter’s data type must be set to an array. Moreover, a function can have at the most one rest parameter.
* The function is invoked twice, by passing three and six values, respectively.
* The for loop iterates through the argument list, passed to the function and calculates their sum.

On compiling, it will generate following JavaScript code −

function addNumbers() {

var nums = [];

for (var \_i = 0; \_i < arguments.length; \_i++) {

nums[\_i - 0] = arguments[\_i];

}

var i;

var sum = 0;

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

sum = sum + nums[i];

}

console.log("sum of the numbers", sum);

}

addNumbers(1, 2, 3);

addNumbers(10, 10, 10, 10, 10);

The output of the above code is as follows −

sum of numbers 6

sum of numbers 50

**Default Parameters**

Function parameters can also be assigned values by default. However, such parameters can also be explicitly passed values.

**Syntax**

function function\_name(param1[:type],param2[:type] = default\_value) {

}

**Note** − A parameter cannot be declared optional and default at the same time.

**Example: Default parameters**

function calculate\_discount(price:number,rate:number = 0.50) {

var discount = price \* rate;

console.log("Discount Amount: ",discount);

}

calculate\_discount(1000)

calculate\_discount(1000,0.30)

On compiling, it will generate following JavaScript code −

//Generated by typescript 1.8.10

function calculate\_discount(price, rate) {

if (rate === void 0) { rate = 0.50; }

var discount = price \* rate;

console.log("Discount Amount: ", discount);

}

calculate\_discount(1000);

calculate\_discount(1000, 0.30);

Its output is as follows −

Discount amount : 500

Discount amount : 300

* The example declares the function, *calculate\_discount*. The function has two parameters - price and rate.
* The value of the parameter *rate* is set to *0.50* by default.
* The program invokes the function, passing to it only the value of the parameter price. Here, the value of *rate* is *0.50* (default)
* The same function is invoked, but with two arguments. The default value of *rate* is overwritten and is set to the value explicitly passed.

**Anonymous Function**

Functions that are not bound to an identifier (function name) are called as **anonymous functions**. These functions are dynamically declared at runtime. Anonymous functions can accept inputs and return outputs, just as standard functions do. An anonymous function is usually not accessible after its initial creation.

Variables can be assigned an anonymous function. Such an expression is called a function expression.

**Syntax**

var res = function( [arguments] ) { ... }

**Example ─ A Simple Anonymous function**

var msg = function() {

return "hello world";

}

console.log(msg())

On compiling, it will generate the same code in JavaScript.

It will produce the following output −

hello world