## TypeScript - functions

➔ Functions are the fundamental building block of any applications that are used to perform specific tasks.

➔ We can use functions to implement/mimic the concepts of object-oriented programming like classes, objects, polymorphism, and abstraction.

➔ While TypeScript provides the concept of classes and modules, functions still are an integral part of the language.

### Advantage of function

- Code reusability
- Less coding
- Easy to debug

Functions are divided into two types:

1. Named function
2. Anonymous functions

**Named function**

➔ When we declare and call a function by its given name, then the function is known as a named function.

```JavaScript
function display() {
  console.log("Hello TypeScript!");
}

display(); //Output: Hello TypeScript
```

Functions can also include parameter types and return type.

```JavaScript
function Sum(x: number, y: number) : number {
  return x + y;
}

Sum(2,3); // returns 5
```

**Anonymous function**

- ➔ A function without a name is known as an anonymous function.
- ➔ An anonymous function is one which is defined as an expression.
- ➔ This expression is stored in a variable.
- ➔ So, the function itself does not have a name. These functions are invoked using the variable name that the function is stored in.

```JavaScript
let greeting = function() {
  console.log("Hello TypeScript!");
};
greeting(); //Output: Hello TypeScript!
```

An anonymous function can also include parameter types and return type.

```JavaScript
let Sum = function(x: number, y: number) : number {
  return x + y;
}
Sum(2,3); // returns 5
```

## Function Parameters

- Parameters are values or arguments passed to a function.
- In TypeScript, the compiler expects a function to receive the exact number and type of arguments as defined in the function signature.
- If the function expects three parameters, the compiler checks that the user has passed values for all three parameters i.e. it checks for exact matches.

JavaScript

```
function Greet(greeting: string, name: string ) : string {
    return greeting + ' ' + name + '!';
}
Greet('Hello','Steve');//OK, returns "Hello Steve!"
Greet('Hi'); // Compiler Error: Expected 2 arguments, but got 1.
Greet('Hi','Bill','Gates'); //Compiler Error: Expected 2 arguments, but got 3.
```

- This is unlike JavaScript, where it is acceptable to pass less arguments than what the function expects. The parameters that don't receive a value from the user are considered as `undefined`.

**Optional Parameters**

- TypeScript has optional parameter functionality. The parameters that may or may not receive a value can be appended with a '?' to mark them as optional.

**Note** - All optional parameters must follow required parameters and should be at the end.

JavaScript

```
function Greet(greeting: string, name?: string ) : string {
    return greeting + ' ' + name + '!';
}
Greet('Hello','Steve');//OK, returns "Hello Steve!"
Greet('Hi'); // OK, returns "Hi undefined!".
Greet('Hi','Bill','Gates'); //Compiler Error: Expected 2 arguments, but got 3.
```

In the above example, the second parameter `name` is marked as optional with a question mark appended at the end. Hence, the function `Greet()` accepts either 1 or 2 parameters and returns a greeting string. If we do not specify the second parameter then its value will be `undefined`.

### Default Parameters

TypeScript provides the option to add default values to parameters. So, if the user does not provide a value to an argument, TypeScript will initialize the parameter with the default value. Default parameters have the same behavior as optional parameters

JavaScript

```
function Greet(name: string, greeting: string = "Hello") : string {
    return greeting + ' ' + name + '!';
}
Greet('Steve');//OK, returns "Hello Steve!"
Greet('Steve', 'Hi'); // OK, returns "Hi Steve!".
Greet('Bill'); //OK, returns "Hello Bill!"
```

Fat arrow notations are used for anonymous functions i.e for function expressions. They are also called lambda functions in other languages.

***(param1, param2, ..., paramN) => { expression }***

Using fat arrow `=>`, we dropped the need to use the `function` keyword. Parameters are passed in the parenthesis `()`, and the function expression is enclosed within the curly brackets `{ }`.

arrow function without parameters -

JavaScript

```
let Print = () => console.log("Hello TypeScript");
Print(); //Output: Hello TypeScript
```

arrow function with parameters -

JavaScript

```
let sum = (x: number, y: number): number => {
    return x + y;
}
sum(10, 20); //returns 30
```