Funciones en TypeScript

1 Las funciones son los bloques principales de cualquier programa. En JavaScript, las funciones son la parte más importante ya que JavaScript es un lenguaje de programación funcional. Con funciones, puede implementar/imitar los conceptos de programación orientada a objetos como clases, objetos, polimorfismo y

Las funciones garantizan que el programa sea mantenible, reutilizable y organizado en bloques legibles. Si bien TypeScript proporciona el concepto de clases y módulos, las funciones siguen siendo una parte integral del lenguaje.

Una función con nombre es aquella en la que se declara y llama a una función por su nombre de pila.

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

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

Las funciones también pueden incluir tipos de parámetros y tipos de retorno.

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

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

Una función anónima es aquella que se define como una expresión. Esta expresión se almacena en una variable. Entonces, la función en sí no tiene nombre. Estas funciones se invocan utilizando el nombre de la variable en la que está almacenada la función.

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

greeting(); //Output: Hello TypeScript!
```

Una función anónima también puede incluir tipos de parámetros y tipos de retorno

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

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

Los argumentos son valores pasados a una función. En TypeScript, el compilador espera que una función reciba el número y tipo exacto de argumentos definidos en la firma de la función. Si la función espera tres parámetros, el compilador verifica que el usuario haya pasado valores para los tres parámetros, es decir, busca coincidencias exactas.

```
function Greet(greeting: string, name: string ) : string {
  return greeting + ' ' + name + '!';
}

Greet('Hello','Estudents');//OK, returns "Hello Students!"
Greet('Hi'); // Compiler Error: Expected 2 arguments, but got 1.
Greet('Hi','Bill','Gates'); //Compiler Error: Expected 2 arguments, but got 3.
```

TypeScript tiene una funcionalidad de parámetro opcional. Los parámetros que pueden o no recibir un valor se pueden agregar con un '?' para marcarlos como opcionales.

```
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.
```

Todos los parámetros opcionales deben seguir los parámetros requeridos y deben estar al final.

TypeScript ofrece la opción de agregar valores predeterminados a los parámetros. Entonces, si el usuario no proporciona un valor a un argumento, TypeScript inicializará el parámetro con el valor predeterminado. Los parámetros predeterminados tienen el mismo comportamiento que los parámetros opcionales.

```
function Greet(name: string, greeting: string = "Hello") : string {
  return greeting + ' ' + name + '!';
}

Greet('Ana');//OK, returns "Hello Ana!"
Greet('Ana', 'Hi'); // OK, returns "Hi Ana!".
Greet('Juan Jose'); //OK, returns "Hello Juan !"
```

2 TypeScript - Funciones de flecha

Las notaciones de flecha gruesa se utilizan para funciones anónimas, es decir, para expresiones de funciones. También se les llama funciones lambda en otros idiomas

Usando flecha gruesa =>, eliminamos la necesidad de usar la l apalabra clave function . Los parámetros se pasan entre paréntesis ()y la expresión de la función se incluye entre llaves { }.

En el ejemplo anterior, sumes una función de flecha. (x:number, y:number)denota los tipos de parámetros, :numberespecifica el tipo de retorno. La flecha gruesa =>separa los parámetros de la función y el cuerpo de la función. El lado derecho de =>puede contener una o más declaraciones de código.

```
let sum = (x: number, y: number): number => {
  return x + y;
}

sum(10, 20); //returns 30
```

La siguiente es una función de flecha sin parámetros.

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

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

Además, si el cuerpo de la función consta de una sola declaración, entonces no son necesarias las llaves ni la palabra clave return, como se muestra a continuación.

3 TypeScript: sobrecarga de funciones

TypeScript proporciona el concepto de sobrecarga de funciones. Puede tener varias funciones con el mismo nombre pero diferentes tipos de parámetros y tipos de retorno. Sin embargo, el número de parámetros debe ser el mismo.

```
function add(a:string, b:string):string;

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

function add(a: any, b:any): any {
    return a + b;
}

console.log(add("Hello ", "Steve")) // returns "Hello Steve"
console.log(add(10, 20)); // returns 30
```

No se admite la sobrecarga de funciones con diferente número de parámetros y tipos con el mismo nombre.

```
function display(a:string, b:string):void //Compiler Error: Duplicate function implementation
{
    console.log(a + b);
}

function display(a:number): void //Compiler Error: Duplicate function implementation
{
    console.log(a);
}
```

4

TypeScript - Parámetros Rest

TypeScript introdujo parámetros de descanso para acomodar fácilmente un número n de parámetros.

Cuando no se conoce el número de parámetros que recibirá una función o puede variar, podemos utilizar parámetros de descanso. En JavaScript, esto se logra con la variable "arguments". Sin embargo, con TypeScript, podemos usar el parámetro rest indicado por puntos suspensivos ....

```
1  function Greet(greeting: string, ...names: string[]) {
2    return greeting + " " + names.join(", ") + "!";
3  }
4
5  Greet("Hello", "Steve", "Bill"); // returns "Hello Steve, Bill!"
6
7  Greet("Hello");// returns "Hello !"
```