Funciones en TS

Antonio Espín Herranz

Funciones

- Definición
- Con / Sin parámetros
- Con parámetros opcionales
- Con parámetros por defecto
- Resto de parámetros
- Parámetros con valores null
- Funciones void
- Sobrecarga de funciones
- Funciones flecha
- Funciones autoejecutables

Definición

function functionName(parameter1: **type**, parameter2: **type**): **returnType** {

    // Function body

    // You can use parameter1, parameter2, etc. inside the function

    return valueOfTypeReturnType; // Optional return statement

}

Con / Sin parámetros

- Las funciones pueden tener o no parámetros.
- No son obligatorios
- Se pueden inicializar (parámetros por defecto)
- O tener un número indeterminado de parámetros: rest

Con parámetros opcionales

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

- Podemos tener varios parámetros opcionales pero, siempre van a la derecha y a la izquierda los obligatorios

Con parámetros por defecto

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

// También es posible esta operación
function multiply(a: number, b: number = 2 * a): number {
    return a * b;
}
```

Resto de parámetros

```
function functionName(param1: type, ...restParams: type[]): returnType {
    // Function body
}
```

Parámetros con valores null

```
function printLength(input: string | null): void {
    if (input !== null) {
        console.log(input.length);
    } else {
        console.log("Input is null");
    }
}

printLength("hello"); // Output: 5
printLength(null);
```

Reciben el nombre de tipos **guard**

Parámetros con valores null:

```
function printLength(input: string | null = ""): void {
    console.log(input.length);
}

printLength("hello"); // Output: 5
printLength(null);    // Output: 0
```

Ejemplo

```
function procesarValor(valor: string | number): void {
    if (typeof valor === "string") {
        // El tipo es string
        console.log(valor.toUpperCase());
    } else {
        // El tipo es un number
        console.log(valor.toFixed(2));
    }
}
```

Funciones void

- Las funciones void indican que la función no devuelve nada.

```
function functionName(param1: type1, param2: type2): void {
   // Function body
}
```

Sobrecarga de funciones

```
function functionName(param1: type1): returnType1;
function functionName(param1: type1, param2: type2): returnType2;
function functionName(param1: type1, param2: type2, param3: type3): returnType3;

function functionName(param1: any, param2?: any, param3?: any): any {
    // Function body
}
```

Ejemplo

```
function combinar(a: string, b: string): string;
function combinar(a: number, b: number): number;
function combinar(a: any, b: any): any {
   return a + b;
}

console.log(combinar(2, 3)); // 5
console.log(combinar("Hola, ", "Antonio")); // "Hola, Antonio"
```

Funciones flecha

```
// Con parámetros:
const funcionFlecha = (param1: tipo, param2: tipo): tipoDeRetorno => {
    // cuerpo de la función
};

// Sin parámetros:
const saludar = (): string => {
    return "Hola!";
};
```

Las funciones flecha son anónimas.

Mantienen el contexto de this, a diferencia de las funciones normales.

Pueden ser usadas como funciones de callback para simplificar el código.

Funciones autoejecutables

- Se conocen como funciones IIFE Inmediately Invoked Function Expressions, se ejecutan en el momento que se definen.

```
(function() {
   console.log("Soy una función autoejecutable");
})();
```

- Pueden ser de tipo flecha:

```
(() => {
   console.log("Soy una IIFE con función flecha");
})();
```

Funciones autoejecutables

- Sirven para encapsular variables y evitar conflictos a nivel global;

```
(function() {
    const secreto = "Este es mi ámbito privado";
    console.log(secreto);
})();
```

- // Aquí, intentar acceder a "secreto" lanzaría un error porque está fuera de alcance.