Le funzioni in TypeScript

Docente
Mario Perna
prof.perna.mario@darzo.net

A.S.
2025/2026
Materia
TEPSIT (Laboratorio)

Introduzione

Le funzioni rappresentano un pilastro fondamentale in molti linguaggi di programmazione.

In TypeScript servono ad organizzare il codice e renderlo riutilizzabile.

TypeScript fornisce un sistema di tipizzazione forte per garantire la sicurezza dei tipi nelle funzioni.

In TS possono essere utilizzate:

- funzioni esplicite,
- espressioni di funzioni,
- arrow function
- funzioni generiche.

Definizione di una funzione

- Le funzioni sono blocchi di codice riutilizzabili che possiedono un compito specifico.
- In TypeScript, le funzioni possono essere dichiarate e utilizzate seguendo regole ben definite.

Dichiarazione di una funzione 'esplicita'

```
1  //Dichiarazione di una funzione 'esplicita'
2  function saluta(nome: string): void {
3      console.log(`Ciao, ${nome}!`);
4  }
5
6  let nomeUtente: string = "Harry Potter";
7  saluta(nomeUtente); // Chiamata alla funzione
```

Definizione di una funzione

```
1   //Dichiarazione di una funzione 'esplicita'
2   function saluta(nome: string): void {
3       console.log(`Ciao, ${nome}!`);
4   }
```

- **function** è la parola chiave per dichiarare una funzione.
- **saluta** è l'identificatore della funzione.
- **(nome: string)** è la lista dei parametri formali con i loro tipi.
- **void** indica che la funzione non restituisce un valore.
- **${nome}** in JS e TS è nota come "**template strings**".
  Questa notazione permette di incorporare variabili o espressioni all'interno di una stringa in modo più leggibile e conveniente.

Dichiarazione di funzioni come espressioni

```
1   //Assegnazione di una funzione a una variabile
2   let saluta = function(nome: string): void {
3       console.log(`Ciao, ${nome}!`);
4   };
```

- Le funzioni sono blocchi di codice riutilizzabili che possiedono un compito specifico.
- In TypeScript, le funzioni possono essere dichiarate e utilizzate seguendo regole ben definite.

Le "Arrow Function" (Lambda)

```
1   //Assegnazione di un'Arrow Function
2   let saluta = (nome: string): void => {
3       console.log(`Ciao, ${nome}!`);
4   };
```

- **(nome: string)** è la lista dei parametri formali con i loro tipi.
- **=>** in questo contesto rappresenta la notazione delle Arrow Function.
- **void** indica che la funzione non restituisce un valore.
- **${nome}** in JS e TS è una "**template strings**".

Restituzione di valori

```
1   //Funzione per la somma di due numeri
2   function somma(a: number, b: number): number{
3       return a+b;
4   }
5   //Chiamata della funzione
6   let risultato = somma(10, 13); // Risultato conterrà 23
```

La funzione restituirà la somma dei valori passati come parametri di riferimento della funzione.

Restituzione di valori

```
5    //Chiamata della funzione
6    let risultato = somma(10, '13'); //ERRORE!
```

Qualora venissero passati valori non coerenti con i tipi di dato dichiarati nei parametri formali, il compilatore segnalerebbe l'errore.

Funzioni con parametri opzionali e predefiniti

```
1   //Funzione per il calcolo dell'area di un rettangolo
2   function calcolaArea(base: number, altezza: number=5): number{
3       return base*altezza;
4   }
5   let area1: number = calcolaArea(10);    //Area1 conterrà 50
6   let area2: number = calcolaArea(10,8);  //Area2 conterrà 80
```

Come si può notare al parametro **altezza** è stato assegnato il valore **5** di default.

In fase di chiamata della funzione **se non viene passato il secondo parametro, verrà considerato il valore 5**.

Overload delle funzioni

Come in Java anche in TS è possibile effettuare l'overload delle funzioni (metodi in Java).

```
1  //Overload della funzione 'stampa()'
2  function stampa(dato: string): void;
3  function stampa(dato: number): void;
4  function stampa(dato: any): void {
5      console.log(dato);
6  }
7
8  stampa("Ciao!"); // Stampa "Ciao!"
9  stampa(42); // Stampa 42
```

Funzioni Generiche

```
1   function lunghezzaArray<T>(array: T[]): number{
2       return array.length;
3   }
4   let numeri: number[] = [1, 2, 3, 4, 5];
5   let lunghezza: number = lunghezzaArray(numeri); // lunghezza conterrà 5
```

Il codice è un esempio di una funzione generica di TypeScript, che è una funzione che può lavorare con diversi tipi di dati in modo flessibile.

In questo caso, la funzione **lunghezzaArray** accetta un array di qualsiasi tipo **(T)** e restituisce la sua lunghezza come un valore di tipo number.

NOTA: T è utilizzato nei tipi generici ed è arbitrario. Può essere sostituito con qualsiasi altro identificatore valido. La notazione è comunemente utilizzata come convenzione per rappresentare un tipo generico.

Type "any" nelle funzioni

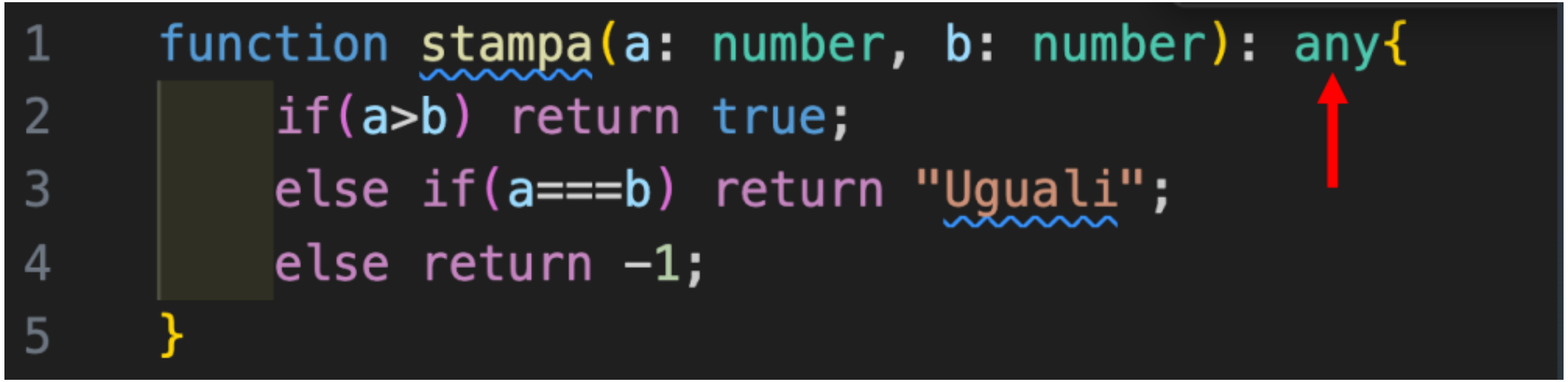
Consideriamo la seguente funzione:

```
1    function stampa(a: number, b: number): boolean{
2        if(a>b) return true;
3        else if(a===b) return "Uguali";
4        else return -1;
5    }
```

Come si può dedurre nelle righe 3 e 4 c'è un errore nel return, ciò avviene poiché la funzione restituisce solo tipi boolean.

Type "any" nelle funzioni

Con il tipo **any** il problema è stato risolto!



In TypeScript, il tipo **any** è un tipo speciale che rappresenta un valore di qualsiasi tipo.

Quando una funzione è dichiarata con questo tipo, il compilatore TypeScript ignorerà la verifica dei tipi restituiti per quella funzione. Ciò consente di restituire qualsiasi tipo di valore da una funzione di ricevere errori di tipo durante la compilazione.