Function Overloading in TypeScript

What is Function Overloading?

Function Overloading allows us to define **multiple versions of a function with the same name**, but with **different parameter types or return types**.

In **TypeScript**, we can achieve:

- **Function Overloading**
- **Method Overloading**

Method Overloading is part of Object-Oriented Programming (OOP) and is a key concept in **Polymorphism**.

Java vs TypeScript Overloading

Feature	Java	TypeScript
Overloading Type	Only **Method Overloading**	**Function & Method Overloading**
Requirement	Class is required	Class is **not required**

Steps to Achieve Function Overloading in TypeScript

Step 1: Define Function Signatures

A **function signature** means a function **without a body or implementation**. It just defines the name, parameters, and return type.

```
function getInfo(id: number): string;      // First signature
function getInfo(name: string): string;    // Second signature
```

Both functions have the same name (`getInfo`) but different parameter types (`number` and `string`).

Step 2: Implement the Function

In TypeScript, even if you have multiple signatures, you **write only one actual function implementation**.

Example Implementation:

```
function getInfo(param: number | string): string {
  if (typeof param === "number") {
    return `User ID is ${param}`;
  } else {
    return `Username is ${param}`;
  }
}
```

🔍 **Explanation:**

- The `param` is declared as a **union type** (`number | string`) to accept both.
- We use the **`typeof` operator** to check the type at runtime:
  - If it's a number → return a message with ID.
  - If it's a string → return a message with username.

⚠️ Even though there are multiple signatures, we always write **one implementation** function.

Step 3: Call the Function

Depending on the argument you pass, the correct signature is selected.

```
console.log(getInfo(8));         // Output: User ID is 8
console.log(getInfo("Yogesh"));  // Output: Username is Yogesh
```

📄 Full Code Example

```
// Function Signatures
function getInfo(id: number): string;
function getInfo(name: string): string;

// Implementation
function getInfo(param: number | string): string {
  if (typeof param === "number") {
    return `User ID is ${param}`;
  } else {
    return `Username is ${param}`;
  }
}

// Function Calls
console.log(getInfo(8));          // Output: User ID is 8
console.log(getInfo("Yogesh"));   // Output: Username is Yogesh
```

Important Rules for Function Overloading

1. **Function name** must be **exactly the same**.
2. You can change the **type** or the **number of parameters**.
3. Even the **return type** can be different in each signature.
4. But — **only one implementation** is allowed.
