Table of Contents

# Bab 3: Functions di TypeScript

## Penjelasan Materi

Functions di TypeScript merupakan blok bangunan fundamental yang memperluas kemampuan functions JavaScript dengan sistem tipe yang kuat. TypeScript memungkinkan kita untuk mendefinisikan tipe untuk parameter fungsi dan nilai return, memastikan bahwa fungsi digunakan dengan cara yang benar dan konsisten. TypeScript juga mendukung fitur-fitur modern seperti arrow functions, optional parameters, default parameters, dan function overloading.

Dengan TypeScript, kita bisa mencegah kesalahan umum seperti memanggil fungsi dengan jumlah atau tipe parameter yang salah, atau menggunakan nilai return dengan cara yang tidak sesuai. Ini membuat kode lebih aman dan lebih mudah di-maintain.

## Analogi yang Mudah Dipahami

Bayangkan function seperti mesin di pabrik: - Parameter fungsi seperti bahan baku yang dimasukkan ke mesin - harus sesuai spesifikasi - Tipe return seperti produk yang dihasilkan - kualitasnya sudah ditentukan - Optional parameters seperti bahan tambahan - boleh ada atau tidak - Function overloading seperti satu mesin yang bisa mengolah berbagai jenis bahan baku - Generic functions seperti mesin yang bisa disesuaikan dengan berbagai jenis produk

Atau bayangkan function seperti resep masakan: - Parameter adalah bahan-bahan yang diperlukan - Tipe parameter adalah spesifikasi bahan (mis: 200gr tepung, 2 butir telur) - Return type adalah hasil masakan yang diharapkan - Optional parameters seperti bumbu tambahan yang opsional

## Point Penting

1. **Function Declaration**
   * Cara mendeklarasikan fungsi dengan tipe
   * Arrow functions vs traditional functions
   * Void dan return types
2. **Parameter Types**
   * Required parameters
   * Optional parameters
   * Default parameters
   * Rest parameters
3. **Function Overloading**
   * Multiple function signatures
   * Type guards dalam overloading
   * Implementation signature
4. **Generic Functions**
   * Type parameters
   * Constraints
   * Multiple type parameters
5. **This dalam TypeScript**
   * This parameters
   * This type
   * Arrow functions dan this

## Contoh Kode dan Penjelasan

```typescript // 1. Basic Function dengan Type Annotations function tambah(a: number, b: number): number { return a + b; }

// 2. Arrow Function const kali = (a: number, b: number): number => a \* b;

// 3. Optional Parameters function sapaPengguna(nama: string, gelar?: string): string { if (gelar) { return `Halo ${gelar} ${nama}!`; } return `Halo ${nama}!`; }

// 4. Default Parameters function hitungDiskon(harga: number, persen: number = 10): number { return harga - (harga \* persen / 100); }

// 5. Rest Parameters function jumlahkan(…angka: number[]): number { return angka.reduce((total, n) => total + n, 0); }

// 6. Function Overloading function proses(x: number): number; function proses(x: string): string; function proses(x: number | string): number | string { if (typeof x === “number”) { return x \* 2; } else { return x.toUpperCase(); } }

// 7. Generic Function function identitas(arg: T): T { return arg; }

// 8. Generic dengan Constraints interface HasLength { length: number; }

function logPanjang(arg: T): number { return arg.length; }

// 9. Function sebagai Parameter function operasiMatematika( a: number, b: number, operasi: (x: number, y: number) => number ): number { return operasi(a, b); }

// 10. Praktik Kompleks interface Produk { id: number; nama: string; harga: number; }

interface KeranjangItem extends Produk { jumlah: number; }

class KeranjangBelanja { private items: KeranjangItem[] = [];

tambahProduk(produk: Produk, jumlah: number = 1): void {  
 const existingItem = this.items.find(item => item.id === produk.id);  
  
 if (existingItem) {  
 existingItem.jumlah += jumlah;  
 } else {  
 this.items.push({ ...produk, jumlah });  
 }  
}  
  
hitungTotal(): number {  
 return this.items.reduce(  
 (total, item) => total + (item.harga \* item.jumlah),  
 0  
 );  
}  
  
tampilkanRingkasan(): string {  
 return this.items  
 .map(item => \`\${item.nama} (x\${item.jumlah}): Rp\${item.harga \* item.jumlah}\`)  
 .join("\\n");  
}

}

// Penggunaan const keranjang = new KeranjangBelanja();

keranjang.tambahProduk({ id: 1, nama: “Laptop”, harga: 15000000 });

keranjang.tambahProduk({ id: 2, nama: “Mouse”, harga: 200000 }, 2);

console.log(keranjang.tampilkanRingkasan()); console.log(`Total: Rp${keranjang.hitungTotal()}`); ```

### Penjelasan Setiap Bagian Kode:

1. **Basic Function**: Menunjukkan cara dasar mendefinisikan fungsi dengan tipe
2. **Arrow Function**: Sintaks alternatif yang lebih ringkas
3. **Optional Parameters**: Parameter yang tidak wajib diisi
4. **Default Parameters**: Nilai default jika parameter tidak diisi
5. **Rest Parameters**: Mengumpulkan banyak parameter menjadi array
6. **Function Overloading**: Satu fungsi dengan berbagai signature
7. **Generic Function**: Fungsi yang bisa bekerja dengan berbagai tipe
8. **Generic Constraints**: Membatasi tipe yang bisa digunakan
9. **Function sebagai Parameter**: Callback function dengan tipe
10. **Praktik Kompleks**: Implementasi real-world dengan class dan interface

## Cara Kerja TypeScript Functions

1. **Compile-time Type Checking**:
   * TypeScript memeriksa tipe parameter dan return value
   * Memastikan fungsi dipanggil dengan parameter yang benar
   * Memvalidasi penggunaan nilai return
2. **Type Inference**:
   * TypeScript bisa menebak return type dari fungsi
   * Bekerja dengan konteks tipe parameter
   * Membantu mengurangi penulisan tipe eksplisit
3. **Runtime Behavior**:
   * Setelah kompilasi, fungsi berjalan seperti JavaScript biasa
   * Type information dihapus saat runtime
   * Performance sama dengan JavaScript

## Tips dan Trik

1. **Gunakan Type Inference untuk Return Type**

* // ✅ Baik: TypeScript bisa menebak return type  
  function tambah(a: number, b: number) {  
   return a + b; // TypeScript tahu ini number  
  }

1. **Explicit Return Type untuk API Public**

* // ✅ Baik: Return type eksplisit untuk API  
  export function getData(): Promise<Data> {  
   return fetch('/api/data').then(r => r.json());  
  }

1. **Function Type Aliases**

* // ✅ Baik: Mendefinisikan tipe fungsi yang reusable  
  type OperasiMatematika = (a: number, b: number) => number;  
    
  const tambah: OperasiMatematika = (a, b) => a + b;  
  const kurang: OperasiMatematika = (a, b) => a - b;

## Kesalahan yang Sering Dilakukan Pemula

1. **Mengabaikan Return Type**

* // ❌ Buruk: Return type tidak jelas  
  function proses(data) {  
   if (data > 0) return "Positif";  
   return false;  
  }  
    
  // ✅ Baik: Return type eksplisit  
  function proses(data: number): string | boolean {  
   if (data > 0) return "Positif";  
   return false;  
  }

1. **Callback Hell tanpa Proper Typing**

* // ❌ Buruk: Callback tanpa tipe  
  function getData(callback) {  
   fetch('/api/data')  
   .then(data => callback(data));  
  }  
    
  // ✅ Baik: Callback dengan tipe  
  type DataCallback = (data: ApiResponse) => void;  
    
  function getData(callback: DataCallback) {  
   fetch('/api/data')  
   .then(data => callback(data));  
  }

1. **Overuse of Any**

* // ❌ Buruk: Menggunakan any  
  function proses(data: any): any {  
   return data.map(item => item.value);  
  }  
    
  // ✅ Baik: Menggunakan generics  
  function proses<T extends { value: any }>(data: T[]): any[] {  
   return data.map(item => item.value);  
  }

### Solusi:

1. Selalu definisikan tipe parameter dan return type
2. Gunakan TypeScript strict mode
3. Manfaatkan function type aliases untuk kode yang reusable
4. Hindari penggunaan any
5. Dokumentasikan fungsi dengan JSDoc comments