Table of Contents

# Bab 2: Tipe Data Dasar di TypeScript

## Penjelasan Materi

Tipe data dalam TypeScript adalah fondasi penting yang membedakannya dari JavaScript. TypeScript menyediakan sistem tipe statis yang kuat, memungkinkan kita untuk menentukan tipe data secara eksplisit untuk variabel, parameter fungsi, dan nilai return. Sistem tipe ini membantu mencegah bug dan membuat kode lebih mudah dipahami serta di-maintain.

TypeScript memiliki beberapa tipe data dasar yang sering digunakan: number, string, boolean, array, tuple, enum, any, void, null, dan undefined. Setiap tipe data memiliki karakteristik dan use case tersendiri yang penting untuk dipahami oleh developer TypeScript.

## Analogi yang Mudah Dipahami

Bayangkan tipe data seperti wadah atau kotak penyimpanan. Setiap wadah dirancang khusus untuk jenis barang tertentu: - Number seperti kotak khusus angka - hanya bisa menyimpan angka - String seperti kotak surat - hanya bisa menyimpan teks - Boolean seperti saklar lampu - hanya bisa on atau off - Array seperti rak buku - bisa menyimpan banyak item sejenis - Tuple seperti kotak makan dengan sekat - setiap sekat memiliki fungsi khusus - Enum seperti seragam sekolah - pilihan terbatas tapi jelas

## Point Penting

1. **Number**
   * Mencakup semua jenis angka (integer dan floating-point)
   * Mendukung operasi matematika
   * Bisa dalam bentuk desimal, biner, oktal, atau heksadesimal
2. **String**
   * Untuk teks dan karakter
   * Mendukung template literals
   * Memiliki banyak method bawaan untuk manipulasi string
3. **Boolean**
   * Hanya memiliki nilai true atau false
   * Digunakan untuk logika kondisional
   * Hasil dari operasi perbandingan
4. **Array**
   * Kumpulan nilai dengan tipe yang sama
   * Bisa dideklarasikan dengan dua cara
   * Mendukung berbagai operasi array
5. **Tuple**
   * Array dengan jumlah dan tipe elemen tetap
   * Setiap posisi memiliki tipe spesifik
   * Berguna untuk data yang berstruktur tetap

## Contoh Kode dan Penjelasan

```typescript // 1. Number let decimal: number = 6; // Bilangan desimal let hex: number = 0xf00d; // Bilangan heksadesimal let binary: number = 0b1010; // Bilangan biner let octal: number = 0o744; // Bilangan oktal let float: number = 3.14; // Bilangan pecahan

// 2. String let nama: string = “Budi”; // String dengan double quotes let pesan: string = ‘Halo’; // String dengan single quotes let template: string = `Halo ${nama}!`; // Template literal dengan interpolasi

// 3. Boolean let selesai: boolean = false; // Nilai boolean false let aktif: boolean = true; // Nilai boolean true

// 4. Array let angka: number[] = [1, 2, 3]; // Array of numbers, sintaks pertama let buah: Array = [“apel”, “jeruk”, “mangga”]; // Array of strings, sintaks kedua

// 5. Tuple let koordinat: [number, number] = [123.456, -78.90]; // Tuple dengan dua number let pengguna: [string, number] = [“budi”, 25]; // Tuple string dan number

// 6. Enum enum StatusPesanan { MENUNGGU = “MENUNGGU”, DIPROSES = “DIPROSES”, SELESAI = “SELESAI” }

// 7. Any dan Unknown let tidakPasti: any = 4; // Bisa diubah ke tipe apapun let tidakTahu: unknown = “hello”; // Lebih aman dari any

// 8. Void dan Never function log(pesan: string): void { // Fungsi yang tidak mengembalikan nilai console.log(pesan); }

function error(pesan: string): never { // Fungsi yang tidak pernah selesai throw new Error(pesan); }

// Contoh penggunaan kompleks interface Produk { id: number; nama: string; harga: number; stok: number; kategori: string[]; status: StatusPesanan; }

function hitungTotal(produk: Produk, jumlah: number): number { return produk.harga \* jumlah; }

const laptop: Produk = { id: 1, nama: “Laptop Gaming”, harga: 15000000, stok: 10, kategori: [“elektronik”, “komputer”], status: StatusPesanan.MENUNGGU };

console.log(hitungTotal(laptop, 2)); // Output: 30000000 ```

### Penjelasan Setiap Bagian Kode:

1. **Number**: Menunjukkan berbagai cara mendeklarasikan angka
2. **String**: Demonstrasi tiga cara penulisan string
3. **Boolean**: Penggunaan nilai true/false
4. **Array**: Dua cara mendeklarasikan array
5. **Tuple**: Array dengan jumlah dan tipe tetap
6. **Enum**: Kumpulan konstanta bernama
7. **Any/Unknown**: Tipe data fleksibel
8. **Void/Never**: Tipe return value khusus

## Cara Kerja TypeScript

1. **Type Checking**:
   * TypeScript melakukan pengecekan tipe saat kompilasi
   * Mencegah operasi yang tidak valid antar tipe
   * Memberikan error jika tipe tidak sesuai
2. **Type Inference**:
   * TypeScript bisa menebak tipe data otomatis
   * Bekerja berdasarkan nilai inisialisasi
   * Membantu mengurangi penulisan tipe eksplisit
3. **Type Compatibility**:
   * Mengecek apakah tipe data kompatibel
   * Memastikan operasi antar tipe aman
   * Mencegah error runtime

## Tips dan Trik

1. **Gunakan Type Inference**

* // ✅ Baik: Biarkan TypeScript menebak tipe  
  let angka = 42; // TypeScript tahu ini number  
  let teks = "Hello"; // TypeScript tahu ini string

1. **Hindari Any**

* // ❌ Buruk: Menggunakan any  
  let data: any = getData();  
    
  // ✅ Baik: Gunakan tipe spesifik atau unknown  
  let data: DataType = getData();

1. **Manfaatkan Literal Types**

* // ✅ Baik: Menggunakan literal types  
  type Arah = "UTARA" | "SELATAN" | "TIMUR" | "BARAT";  
  let tujuan: Arah = "UTARA"; // Hanya bisa salah satu dari 4 nilai

## Kesalahan yang Sering Dilakukan Pemula

1. **Mengabaikan Null/Undefined**

* // ❌ Buruk: Tidak menghandle null  
  function getNama(user: { nama: string }) {  
   return user.nama;  
  }  
    
  // ✅ Baik: Handle null/undefined  
  function getNama(user: { nama?: string }) {  
   return user.nama ?? "Tanpa Nama";  
  }

1. **Salah Menggunakan Array vs Tuple**

* // ❌ Buruk: Menggunakan array untuk data terstruktur  
  let user: string[] = ["Budi", "25"];  
    
  // ✅ Baik: Menggunakan tuple untuk data terstruktur  
  let user: [string, number] = ["Budi", 25];

1. **Terlalu Banyak Any**

* // ❌ Buruk: Menggunakan any untuk semua  
  function proses(data: any): any {  
   return data;  
  }  
    
  // ✅ Baik: Menggunakan generics  
  function proses<T>(data: T): T {  
   return data;  
  }

### Solusi:

1. Selalu tentukan tipe data yang spesifik
2. Gunakan strict mode di tsconfig.json
3. Manfaatkan type inference TypeScript
4. Hindari penggunaan any kecuali benar-benar diperlukan
5. Selalu handle kasus null dan undefined