Table of Contents

[Bab 1: Pengenalan TypeScript 1](#_Toc193573998)

[Penjelasan Materi 1](#_Toc193573999)

[Analogi yang Mudah Dipahami 1](#_Toc193574000)

[Point Penting 2](#_Toc193574001)

[Contoh Kode dan Penjelasan 2](#_Toc193574002)

[Penjelasan Setiap Baris Kode: 2](#_Toc193574003)

[Cara Kerja TypeScript 3](#_Toc193574004)

[Tips dan Trik 3](#_Toc193574005)

[Kesalahan yang Sering Dilakukan Pemula 3](#_Toc193574006)

[Solusi: 4](#_Toc193574007)

# Bab 1: Pengenalan TypeScript

## Penjelasan Materi

TypeScript adalah bahasa pemrograman yang dikembangkan oleh Microsoft sebagai superset dari JavaScript. Ini berarti bahwa semua kode JavaScript yang valid juga merupakan kode TypeScript yang valid. Yang membuat TypeScript istimewa adalah kemampuannya dalam menambahkan fitur tipe statis (static typing) ke dalam JavaScript. Dengan tipe statis, kita dapat menentukan tipe data variabel, parameter fungsi, dan nilai return secara eksplisit, yang membantu mencegah kesalahan sebelum kode dijalankan.

TypeScript juga mendukung fitur-fitur JavaScript modern dan memberikan akses ke fitur-fitur yang mungkin belum tersedia di semua browser. Kode TypeScript dikompilasi menjadi JavaScript, sehingga dapat berjalan di mana saja JavaScript dapat berjalan.

## Analogi yang Mudah Dipahami

Bayangkan Anda sedang membangun rumah. JavaScript seperti membangun rumah tanpa blueprint - Anda bisa mulai membangun langsung, tapi risiko kesalahan lebih tinggi. TypeScript seperti membangun rumah dengan blueprint yang detail - Anda tahu persis di mana setiap bagian harus ditempatkan, ukurannya berapa, dan bagaimana semuanya terhubung.

Contoh lain: JavaScript seperti bermain puzzle tanpa gambar panduan - Anda bisa menyelesaikannya, tapi akan lebih sulit dan mungkin ada kesalahan. TypeScript memberikan Anda gambar panduan (tipe data), sehingga Anda tahu persis bagian mana yang cocok di mana.

## Point Penting

1. **Static Typing**
   * TypeScript memungkinkan pengecekan tipe saat kompilasi
   * Membantu menghindari bug sebelum kode dijalankan
   * Memberikan IntelliSense yang lebih baik di editor kode
2. **Kompatibilitas JavaScript**
   * Semua kode JavaScript adalah kode TypeScript yang valid
   * Dapat menggunakan library JavaScript yang ada
   * Dapat mengadopsi TypeScript secara bertahap
3. **Fitur Modern**
   * Mendukung fitur JavaScript terbaru
   * Memiliki fitur tambahan seperti interface dan enum
   * Dapat dikonfigurasi untuk target JavaScript versi tertentu

## Contoh Kode dan Penjelasan

```typescript // Deklarasi variabel dengan tipe data let nama: string = “Budi”; // Variabel ‘nama’ harus berupa string let umur: number = 25; // Variabel ‘umur’ harus berupa number let aktif: boolean = true; // Variabel ‘aktif’ harus berupa boolean

// Interface untuk mendefinisikan struktur objek interface Pengguna { id: number; // Property ‘id’ wajib ada dan bertipe number nama: string; // Property ‘nama’ wajib ada dan bertipe string email: string; // Property ‘email’ wajib ada dan bertipe string umur?: number; // Property ‘umur’ opsional dan bertipe number }

// Fungsi dengan tipe parameter dan return value function sapaPengguna(pengguna: Pengguna): string { return `Halo ${pengguna.nama}! Email Anda: ${pengguna.email}`; }

// Penggunaan interface dan fungsi const pengguna: Pengguna = { id: 1, nama: “Budi”, email: “budi@email.com” };

console.log(sapaPengguna(pengguna)); // Output: “Halo Budi! Email Anda: budi@email.com” ```

### Penjelasan Setiap Baris Kode:

1. Variabel dengan tipe data eksplisit menunjukkan bagaimana TypeScript memastikan tipe data yang benar
2. Interface `Pengguna` mendefinisikan struktur objek yang wajib diikuti
3. Tanda `?` pada `umur?` menandakan property opsional
4. Fungsi `sapaPengguna` menerima parameter bertipe `Pengguna` dan mengembalikan `string`
5. Objek `pengguna` harus mengikuti struktur interface `Pengguna`

## Cara Kerja TypeScript

1. **Proses Kompilasi**:
   * Kode TypeScript (.ts) ditulis oleh developer
   * TypeScript Compiler (tsc) mengecek tipe dan error
   * Kode dikompilasi menjadi JavaScript (.js)
   * JavaScript dapat dijalankan di browser/Node.js
2. **Type Checking**:
   * Dilakukan saat kompilasi (compile-time)
   * Mencegah error tipe data sebelum runtime
   * Memberikan feedback langsung di editor

## Tips dan Trik

1. **Mulai dengan JavaScript yang Ada**:
   * Ubah file .js menjadi .ts
   * Tambahkan tipe secara bertahap
   * Gunakan `any` untuk tipe yang belum jelas
2. **Manfaatkan Type Inference**:
   * Biarkan TypeScript menebak tipe jika sudah jelas
   * Tambahkan tipe eksplisit untuk parameter fungsi
   * Gunakan tipe eksplisit untuk API contracts
3. **Gunakan Strict Mode**:
   * Aktifkan `strict: true` di tsconfig.json
   * Manfaatkan error checking yang lebih ketat
   * Lebih baik menangani error di awal

## Kesalahan yang Sering Dilakukan Pemula

1. **Mengabaikan Tipe Data**

* // ❌ Buruk: Menggunakan any secara berlebihan  
  function proses(data: any) {  
   return data;  
  }  
    
  // ✅ Baik: Menentukan tipe yang spesifik  
  function proses<T>(data: T): T {  
   return data;  
  }

1. **Tidak Memanfaatkan Interface**

* // ❌ Buruk: Mengulang struktur objek  
  function createUser(nama: string, email: string) { }  
  function updateUser(nama: string, email: string) { }  
    
  // ✅ Baik: Menggunakan interface  
  interface User {  
   nama: string;  
   email: string;  
  }  
  function createUser(user: User) { }  
  function updateUser(user: User) { }

1. **Mengabaikan Null Checks**

* // ❌ Buruk: Tidak menghandle null  
  function getName(user: User) {  
   return user.name.toUpperCase();  
  }  
    
  // ✅ Baik: Menangani kemungkinan null  
  function getName(user: User) {  
   return user?.name?.toUpperCase() ?? "TANPA NAMA";  
  }

### Solusi:

1. Selalu tentukan tipe data yang spesifik
2. Gunakan interface untuk struktur data yang digunakan berulang
3. Aktifkan strict null checks di tsconfig.json
4. Manfaatkan tools linting seperti ESLint
5. Baca error messages TypeScript dengan teliti