let's cover the different data types and variable declarations in TypeScript:

1. **Primitive Data Types**:
   TypeScript supports the following primitive data types:
   - `number`: Represents numeric values, including integers and floating-point numbers.
   - `string`: Represents textual data, enclosed in single (') or double ("") quotes.
   - `boolean`: Represents a logical value, either `true` or `false`.
   - `null`: Represents the absence of a value.
   - `undefined`: Represents a variable that has been declared but not assigned a value.
   - `symbol`: Represents a unique identifier, often used for object property keys.
   - `bigint`: Represents arbitrary precision integers.

2. **Variable Declarations**:
   TypeScript provides three ways to declare variables:
   - `let`: Declares a mutable variable scoped to the block in which it is defined.
   - `const`: Declares an immutable variable whose value cannot be changed once assigned. It's also block-scoped.
   - `var`: Declares a variable with function-scoped or globally-scoped access. However, it's best to avoid using `var` due to its lack of block scope and potential hoisting issues.

   Example:
   ```typescript
   let num: number = 10;
   const message: string = "Hello, TypeScript!";
   var flag: boolean = true;
   ```

3. **Type Annotations**:
   TypeScript allows you to specify the type of a variable explicitly using type annotations.

   Example:
   ```typescript
   let count: number = 5;
   let name: string = "John";
   let isActive: boolean = true;
   ```

4. **Type Inference**:
   TypeScript can automatically infer the type of a variable based on its initialization value if a type annotation is not provided.

   Example:
   ```typescript
   let age = 25; // TypeScript infers age as number
   ```

```
let city = "New York"; // TypeScript infers city as string
```

5. **Type Union**:
   TypeScript allows variables to have multiple types using type union syntax.

   Example:
   ```typescript
   let result: string | number;
   result = "Success"; // Valid
   result = 100; // Also valid
   ```

6. **Any Type**:
   TypeScript provides the `any` type, which disables type checking for a variable, allowing it to hold values of any type.

   Example:
   ```typescript
   let dynamic: any = "Hello";
   dynamic = 10; // Valid
   dynamic = true; // Also valid
   ```

7. **Arrays**:
   TypeScript supports arrays with type annotations to specify the type of elements they contain.

   Example:
   ```typescript
   let numbers: number[] = [1, 2, 3, 4, 5];
   let names: string[] = ["Alice", "Bob", "Charlie"];
   ```

8. **Tuples**:
   Tuples allow you to express an array where the type of a fixed number of elements is known, but they don't need to be the same type.

   Example:
   ```typescript
   let tuple: [number, string] = [1, "hello"];
   ```

9. **Enums**:

Enums allow you to define a set of named constants, making it easier to work with a set of related values.

Example:
```typescript
enum Color {
    Red,
    Green,
    Blue
}
let color: Color = Color.Green;
```

These are some of the key data types and variable declarations in TypeScript, offering strong typing and flexibility for building robust applications.