TypeScript Fundamentals – Beginner Guide

For JavaScript Developers

Chapter 1: Introduction to TypeScript

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. It adds optional static typing and supports modern JavaScript features.

Basic Example

```
let message: string = "Hello TypeScript";
console.log(message);
```

Chapter 2: Installation & Setup

To use TypeScript, install Node.js and then TypeScript via npm. You can compile TypeScript files using 'tsc' or run directly with 'ts-node'.

Setup Example

```
npm install -g typescript
tsc --init
tsc index.ts
node index.js
```

Chapter 3: Basic Types

TypeScript provides static typing for variables using types like string, number, boolean, any, and void.

Type Example

```
let name: string = "John";
let age: number = 25;
let isActive: boolean = true;
```

Chapter 4: Arrays & Tuples

TypeScript supports typed arrays and fixed-length tuples with specified element types.

Array & Tuple Example

```
let numbers: number[] = [1, 2, 3];
let user: [string, number] = ["Alice", 30];
```

Chapter 5: Enums

Enums allow you to define a set of named constants, either numeric or string-based.

Enum Example

```
enum Color {
  Red,
  Green,
  Blue
}
let c: Color = Color.Green;
```

Chapter 6: Functions

Functions in TypeScript can have typed parameters and return values. Optional and default parameters are also supported.

Function Example

```
function add(a: number, b: number): number {
  return a + b;
}
console.log(add(5, 10));
```

Chapter 7: Interfaces & Types

Interfaces define the structure of an object. Type aliases allow alternative names for types.

Interface Example

```
interface Person {
  name: string;
  age: number;
}
let user: Person = { name: "John", age: 25 };
```

Chapter 8: Classes & Access Modifiers

TypeScript supports object-oriented programming with classes, constructors, and access modifiers like public, private, and protected.

Class Example

```
class Car {
  private model: string;
  constructor(model: string) {
    this.model = model;
  }
  drive() {
    console.log(this.model + " is driving");
  }
}
let car = new Car("Toyota");
car.drive();
```

Chapter 9: Generics

Generics provide a way to create reusable components by allowing types as parameters.

Generic Example

```
function identity<T>(arg: T): T {
  return arg;
}
console.log(identity<string>("Hello"));
```

Chapter 10: Modules

Modules organize code by splitting it across multiple files using export and import keywords.

Module Example

```
export const PI = 3.14;
// In another file: import { PI } from "./module";
```

Chapter 11: TypeScript in React & Node

TypeScript integrates with React using .tsx files and works in Node with type definitions.

React Example

```
type ButtonProps = { label: string };
const Button = ({ label }: ButtonProps) => {
  return <button>{label}</button>;
};
```

Final Exercises

1. Build a Todo app using classes and interfaces.
2. Create a generic API response handler function.
3. Define a User interface and implement it in a class.
4. Create and import a module with a utility function.
5. Build a shopping cart using enums and tuples.