

# TypeScript Course

Learn Strongly Typed JavaScript Development Using TypeScript — a comprehensive journey from fundamentals to real-world application, designed for developers ready to write safer, more scalable code.

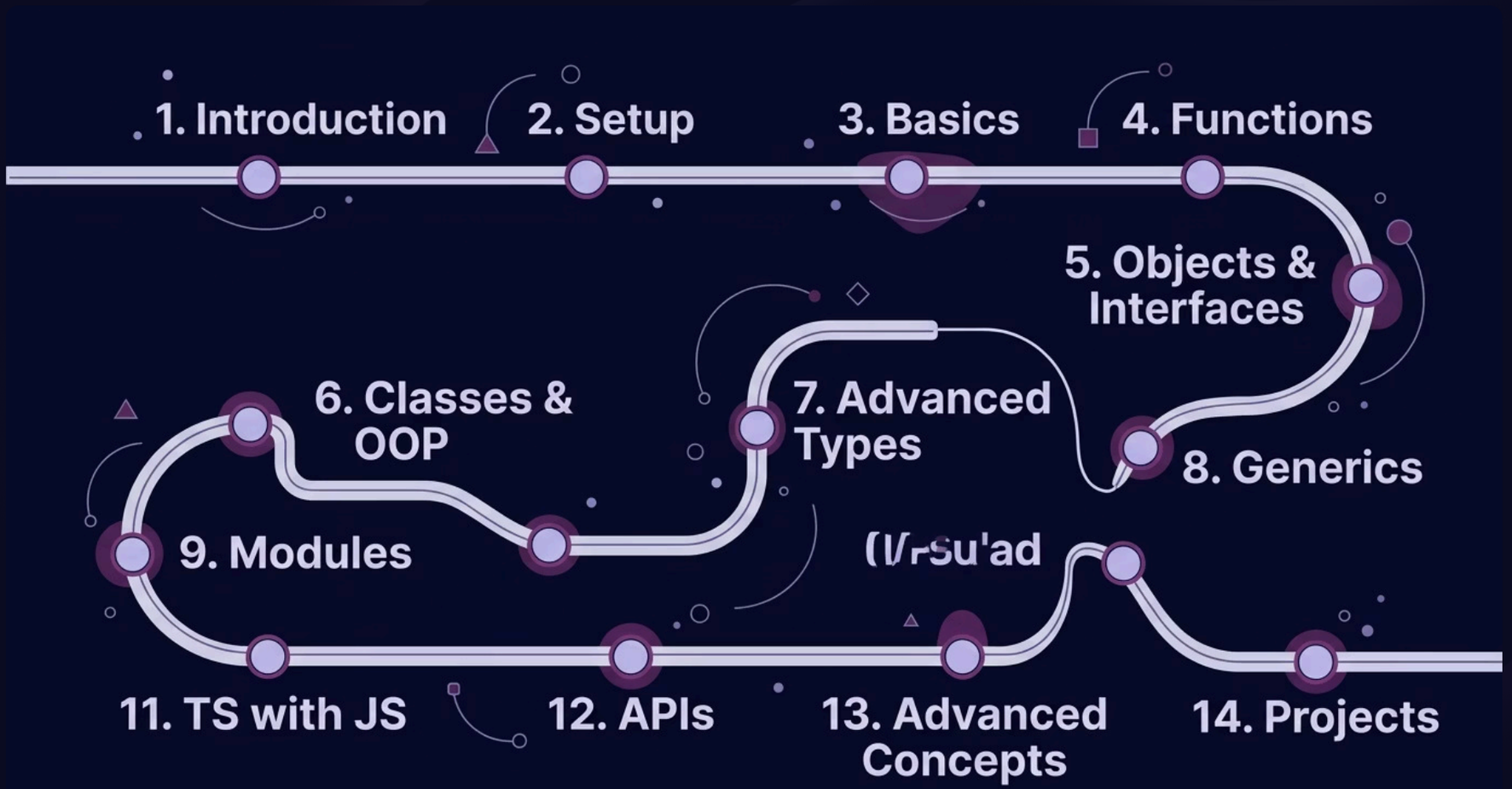
14 MODULES

BEGINNER TO INTERMEDIATE

HANDS-ON PROJECTS

# What You'll Learn

This course is structured as a progressive journey — each module builds on the last, taking you from TypeScript fundamentals all the way to advanced types, decorators, and full project builds.



MODULE 1

# Introduction to TypeScript

## What is TypeScript?

TypeScript is a strongly typed superset of JavaScript developed by Microsoft. It compiles down to plain JavaScript and runs anywhere JS runs — browsers, Node.js, and beyond.

### Key Features

Static typing, interfaces, generics, enums, and advanced tooling support that JavaScript simply doesn't offer out of the box.

### TypeScript vs JavaScript

TypeScript catches errors at compile time rather than runtime — a game-changer for large codebases and team collaboration.

### When to Use It

Ideal for large-scale applications, API-driven projects, team environments, and anywhere code maintainability matters most.

MODULE 2

# Setting Up TypeScript

Before writing a single line of TypeScript, you need the right environment. This module walks through every step — from installing Node.js to writing and compiling your very first `.ts` file.

### Install Node.js & npm

Download from nodejs.org and verify with `node -v` and `npm -v`.

### Install TypeScript

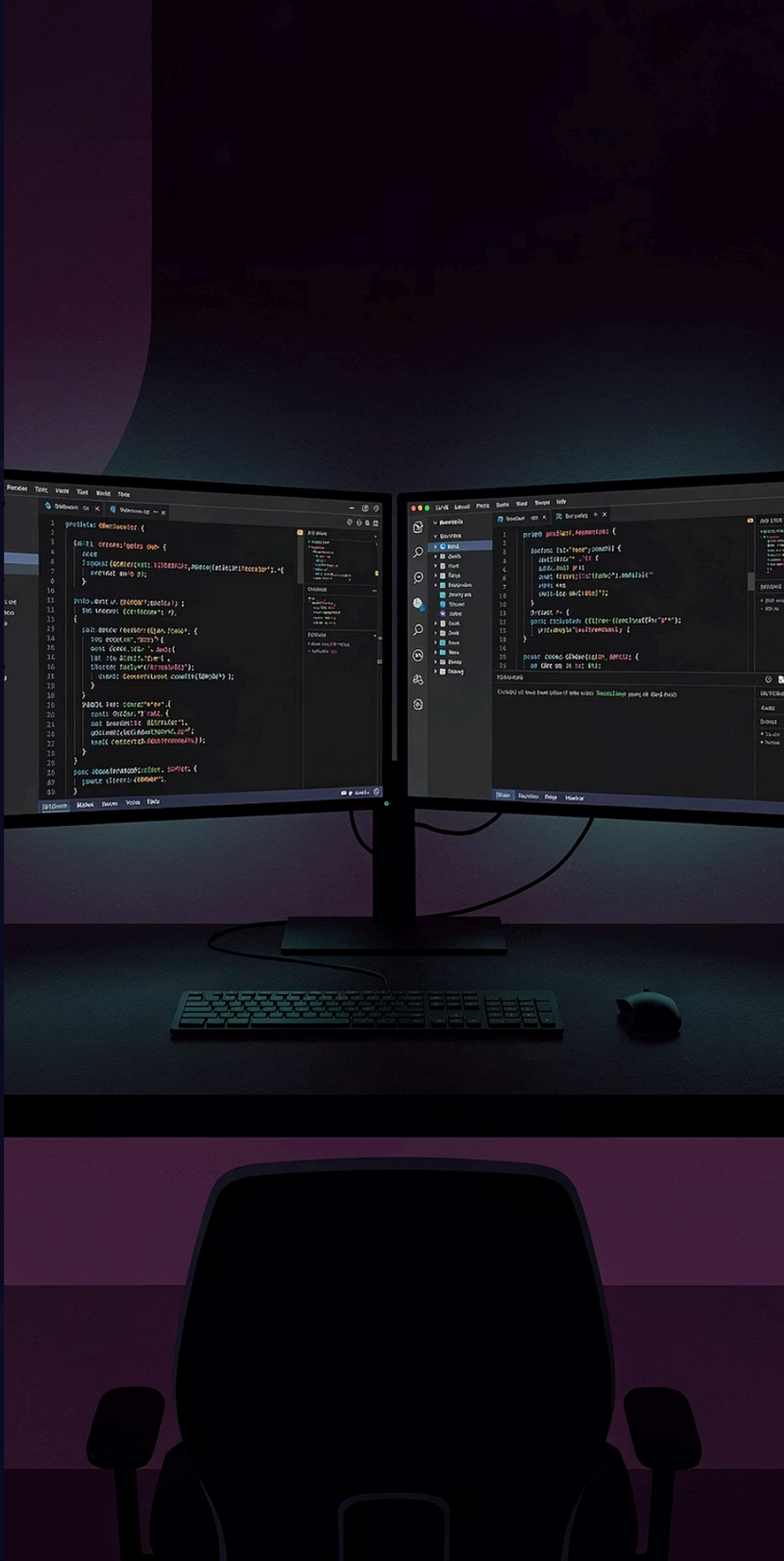
Run `npm install -g typescript` to install the TypeScript compiler globally.

### Use the tsc Compiler

Compile any `.ts` file with `tsc filename.ts` — it outputs clean JavaScript.

### Configure VS Code

Install the TypeScript extension for IntelliSense, error highlighting, and auto-completion.



MODULE 3

# TypeScript Basics — Variables & Data Types

TypeScript's type system is its superpower. Understanding the full range of built-in types gives you control over how data flows through your application and prevents entire classes of bugs before they happen.

### Primitives

`string`, `number`, `boolean` — the everyday building blocks of typed variables.

### Collections

`array` for ordered lists, `tuple` for fixed-length, mixed-type sequences like `[string, number]`.

### Enum

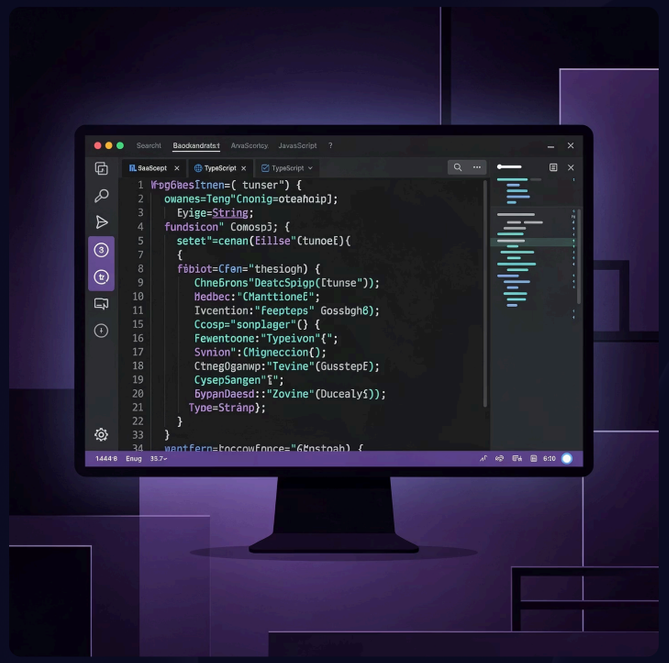
Named constants like `Direction.Up` — readable, self-documenting alternatives to magic numbers or strings.

### Special Types

`any`, `unknown`, `void`, `never`, `null`, and `undefined` — each serves a specific purpose in the type system.

MODULE 4

# Functions in TypeScript



TypeScript supercharges JavaScript functions with explicit parameter types, return type annotations, and flexible parameter patterns that make your function signatures self-documenting.

### Optional & Default Parameters

Mark parameters optional with `?` or provide defaults directly in the signature.

### Rest Parameters

Accept any number of arguments using `...args: string[]` syntax — clean and type-safe.

### Return Types & Arrow Functions

Explicitly annotate return types and use concise arrow functions with full type inference support.

MODULE 5

# Objects and Interfaces

Interfaces are one of TypeScript's most powerful tools for defining the shape of data. They act as contracts — ensuring that objects in your application consistently follow a defined structure.



## Object Types & Aliases

Use `type` aliases to name complex object shapes and reuse them across your codebase for consistency.



## Interfaces

Define the structure of objects with `interface` — interfaces can be extended and merged, making them ideal for scalable architectures.



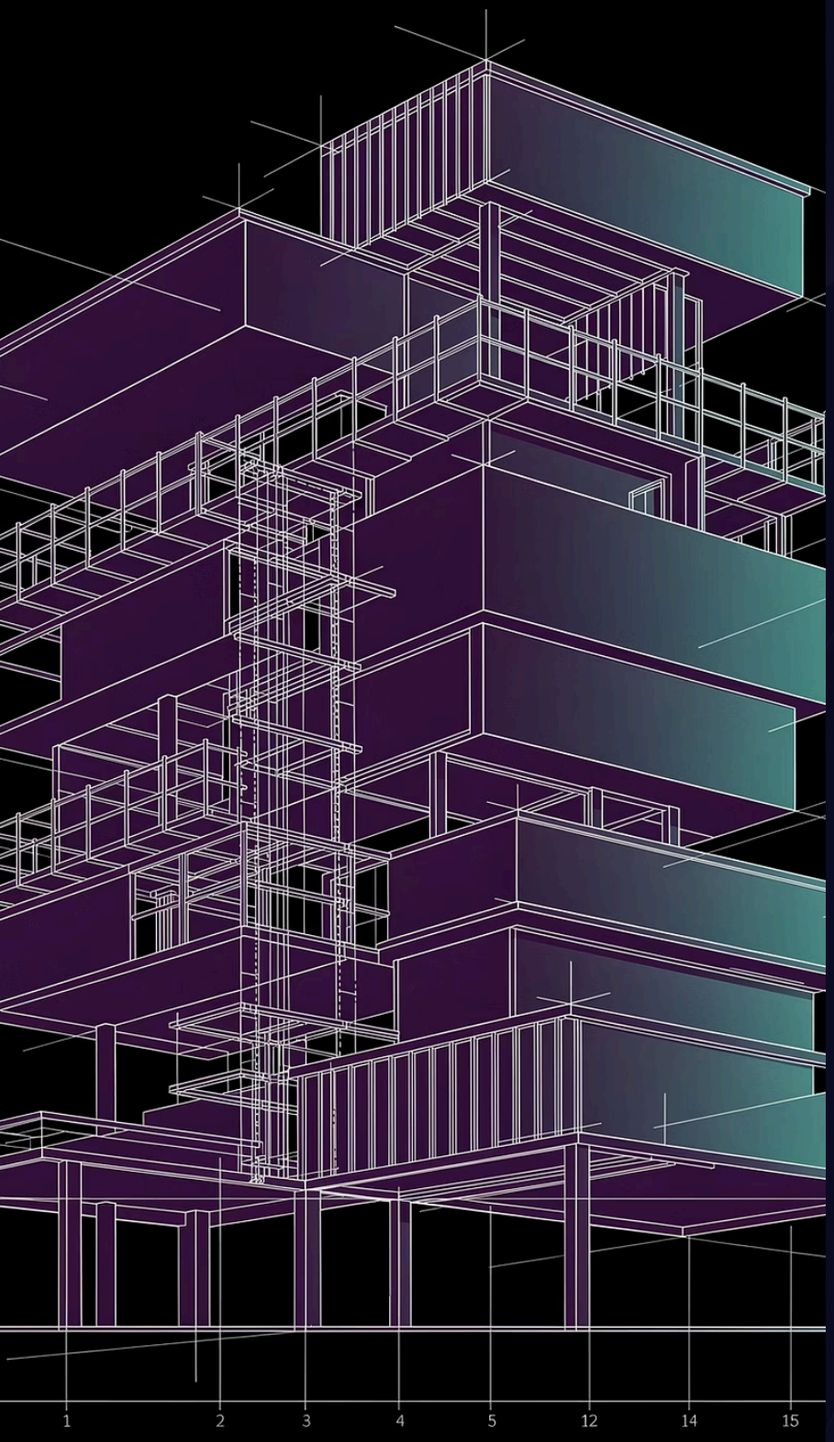
## Optional & Readonly

Use `?` for optional properties and `readonly` to prevent accidental mutation after object creation.



## Function Types in Interfaces

Interfaces can describe callable signatures, allowing you to type functions as first-class members of your objects.




MODULE 6

# Classes and Object-Oriented Programming

TypeScript brings full OOP capabilities to JavaScript — with classes, access modifiers, inheritance, and abstract patterns that make your code modular, reusable, and easy to reason about.

### Classes & Constructors

Define structured blueprints for objects with typed properties and constructor initialization.

### Access Modifiers

`public`, `private`, and `protected` control visibility and encapsulate implementation details.

### Inheritance & Override

Extend base classes and override methods to build specialized behavior without duplicating code.

### Abstract Classes & Accessors

Define incomplete base classes with abstract methods, and use getters/setters for controlled property access.

MODULE 7

# Advanced Types

1

## Union Types

A value that can be `string | number` — flexibly typed without losing safety.

2

## Intersection Types

Combine multiple types into one using `&` — perfect for composing complex shapes.

3

## Literal Types

Restrict a value to exact strings or numbers, like `"success" | "error"`.

4

## Type Narrowing & Guards

Use `typeof`, `instanceof`, and custom guards to narrow types inside conditionals safely.

## Type Assertions

When you know more than the compiler, use `as` to override inferred types — with great power comes great responsibility.

Advanced types are where TypeScript truly shines. Mastering union types, type guards, and literal types unlocks elegant solutions to complex data modeling problems.

MODULE 8

# Generics — Write Once, Use Everywhere

Generics allow you to write reusable, flexible components that work with *any* type while still preserving full type safety. They are the cornerstone of robust TypeScript library design.

1

### Generic Functions

Write a function once with a type parameter `<T>` and call it with any type — the compiler infers the rest.

2

### Generic Classes

Build data structures like stacks, queues, and repositories that work across multiple data types.

3

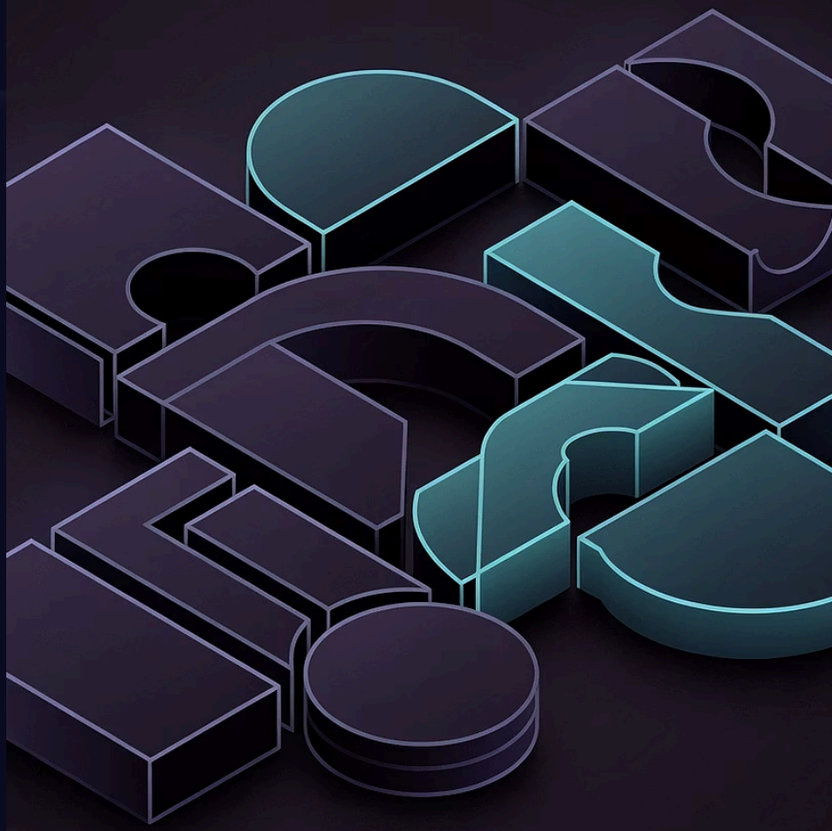
### Generic Interfaces

Define contracts that accept type parameters, making your interfaces adaptable and highly reusable.

4

### Constraints

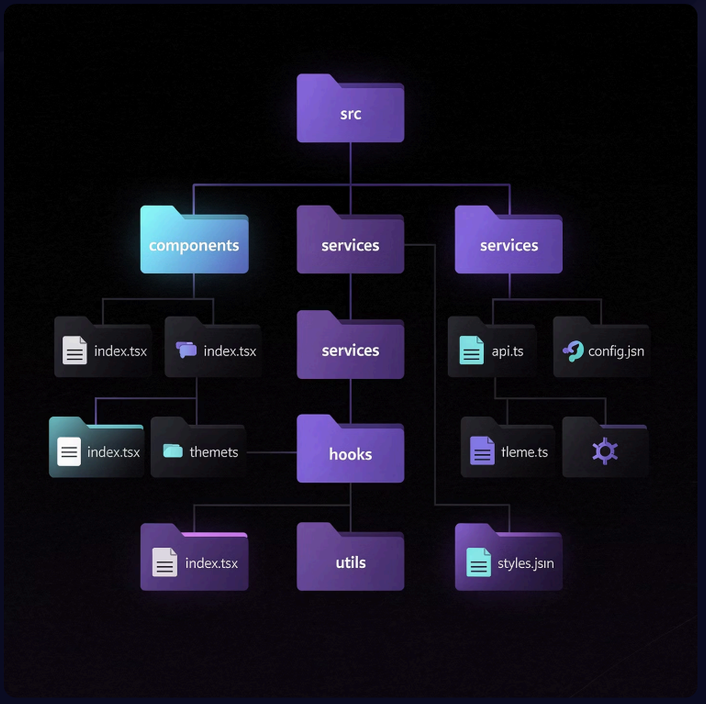
Use `extends` to restrict what types a generic can accept — flexibility with guardrails.



MODULE 9

# Modules and Namespaces

Well-organized code is maintainable code. TypeScript's module system gives you fine-grained control over what your files expose and consume — essential for any serious project.




## Import & Export

Use ES module syntax to share code across files. Both named and default exports are fully supported with type safety intact.

## Namespaces

Group related code under a logical namespace to avoid naming collisions in larger applications.

## File Organization

Structure your project with clear folder hierarchies — separate concerns, co-locate related logic, and keep things easy to navigate.

MODULE 10

# Decorators

Decorators are a powerful, expressive meta-programming feature in TypeScript that let you annotate and modify classes, methods, properties, and parameters — keeping your logic clean and declarative.




## Class Decorators

Apply decorators directly to a class declaration to observe, modify, or replace the class definition — widely used in frameworks like Angular and NestJS.



## Method & Property Decorators

Wrap methods with logging, validation, or access control logic without polluting the method body itself.



## Parameter Decorators

Inspect and annotate individual function parameters — commonly used in dependency injection systems and metadata reflection.





MODULE 11

# TypeScript with Existing JavaScript Projects

You don't have to rewrite everything from scratch. TypeScript is designed to be adopted incrementally — you can introduce it into an existing JavaScript project one file at a time.

### Gradual Adoption

Rename `.js` files to `.ts` gradually. TypeScript's `allowJs` option lets both coexist during migration.

### Type Declaration Files

`.d.ts` files describe the types of JavaScript modules without converting them — giving you type safety for existing code.

### External Libraries

Install `@types/` packages from DefinitelyTyped to get typings for popular libraries like React, Lodash, and Express.

### tsconfig.json

The project configuration file — controls compiler options, include/exclude patterns, module resolution, and strict mode settings.

MODULE 12

# Working with APIs in TypeScript



Fetching and consuming APIs is one of the most common real-world TypeScript tasks. Proper typing of async operations and API responses transforms fragile data handling into robust, predictable code.

01

## Fetch Data with Typed Responses

Define interfaces that mirror your API's JSON shape, then cast fetch results to those types for full IntelliSense support.

02

## Async/Await & Promises

TypeScript fully supports `async/await` with typed return values — `Promise<User[]>` is both readable and safe.

03

## Error Handling

Handle network and parsing failures gracefully with typed error patterns and try/catch blocks.

MODULE 13

# Advanced TypeScript Concepts

These advanced type utilities unlock the full expressive power of the TypeScript type system — letting you transform, filter, and construct new types programmatically rather than defining them manually.



### Utility Types

`Partial<T>`, `Pick<T, K>`, `Omit<T, K>`, and `Record<K, V>` — built-in type transformers for common patterns.



### Indexed Access Types

Look up the type of a specific property using `Type["key"]` — great for deriving types from existing structures.



### Conditional Types

Express type logic with `T extends U ? X : Y` — enabling highly dynamic and context-aware type definitions.



### Mapped Types

Iterate over all keys of an existing type to construct a new one — the engine behind `Partial`, `Readonly`, and more.

MODULE 14

# Practical Projects — Build Real Applications

The best way to solidify TypeScript skills is to build things. This final module puts everything together through five hands-on projects that cover real-world patterns you'll use every day.



### Student Management App

Manage student records with full CRUD operations, typed data models, and interface-driven design.



### Calculator

Build a robust calculator with typed inputs, error handling, and operator logic using functions and enums.



### Todo App

A classic project elevated with TypeScript — typed task models, state management, and filtering logic.



### Employee Record System

Apply classes, access modifiers, and generics to build a realistic HR data management system.



### API Data Viewer

Fetch, type, and display live API data — bringing together async patterns, interfaces, and DOM interaction.



# Your TypeScript Journey Starts Now

From understanding what TypeScript is to building full applications with generics, decorators, and live API data — this course gives you everything you need to write **safer, smarter, more professional JavaScript**.

### 14 Structured Modules

A clear, progressive learning path from zero to advanced TypeScript mastery.

### Real-World Projects

Five hands-on builds that mirror actual industry use cases and reinforce every concept.

### Job-Ready Skills

TypeScript is now standard in modern frontend and backend development — this course makes you hireable.

**Ready to level up?** Start with Module 1 and work through each module in sequence. The projects in Module 14 will make everything click into place.