📘 CHAPTER 2: TypeScript Setup - Installing, Compiling, and tsconfig.json

***1. What Is tsc?***

tsc is the TypeScript compiler. It takes your .ts files and converts them to .js files , the only format browsers understand.

Why?  
TypeScript adds types for development only. Browsers can't run TypeScript directly.

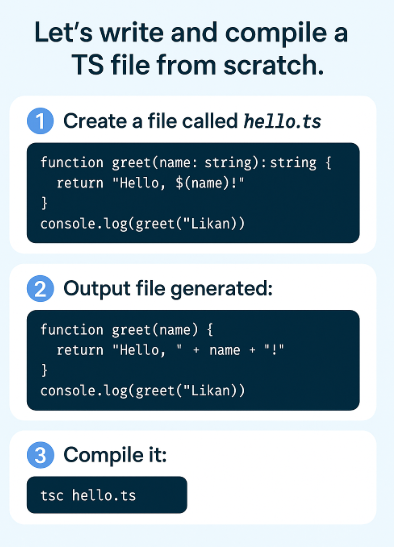
Analogy - tsc is like a translator, it takes a structured typed language (TS) and produces vanilla JavaScript (JS) for runtime.

***2. How to Install TypeScript***



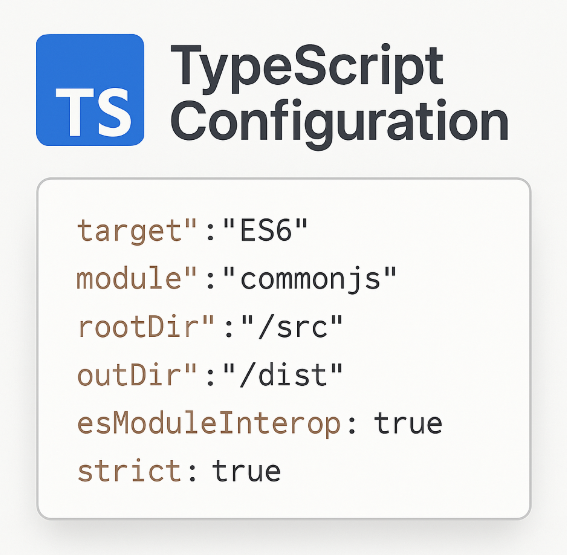
***3. Your First TypeScript File***

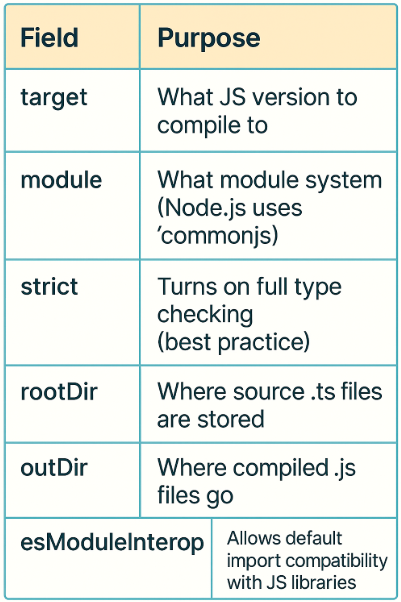
Let’s write and compile a TS file from scratch.



***🗂️ 4. What is tsconfig.json?***

This file configures how the compiler behaves for your project. To create it, run command tsc --init. This generates a tsconfig.json file like this -



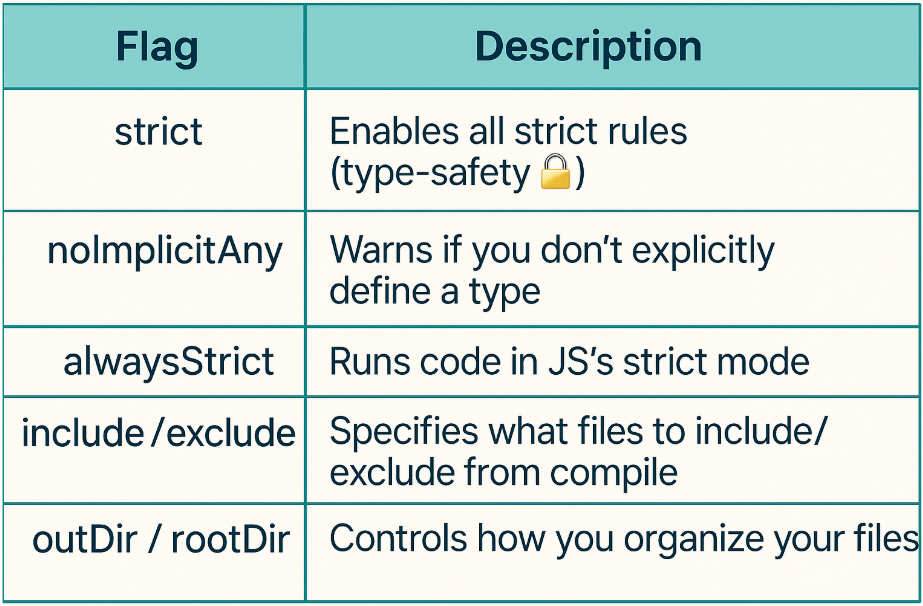
***Explanation of important fields -***

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

***5.Your First TypeScript Project Structure***



***6. Important Compiler Flags (in tsconfig.json)***



🧠 Best Practice for Interviews: Always turn on strict and noImplicitAny in real projects.

Interview Insights -

***Q: What does the TypeScript compiler do?***

A: It checks your code for type errors and converts .ts into browser-compatible .js files.

***Q: What is tsconfig.json used for?***

A: It configures how your TypeScript project compiles: target version, output folder, strictness, etc.

***Q: Can the browser run TypeScript directly***?

A: ❌ No. Only JavaScript runs in the browser. You need to compile TS first.

📁 rootDir vs outDir in tsconfig.json

***rootDir (Source folder)***

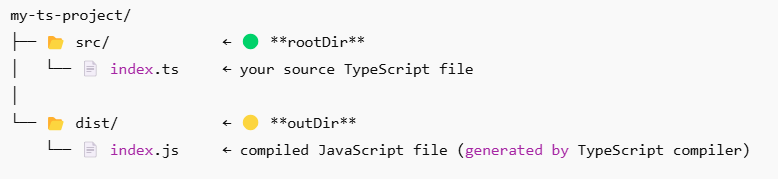
* This is where your TypeScript (.ts) source files live
* It tells the compiler: Start reading files from here

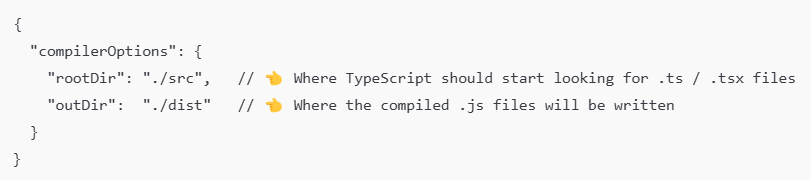
***outDir (Output folder)***

* This is where your compiled JavaScript (.js) files will go
* It tells the compiler: Place the compiled code here

***🔧 Example Setup***

Suppose you have this project structure

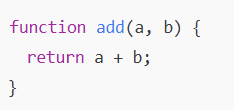
In your tsconfig.json



noImplicitAny -

What happens normally in JavaScript?

In JavaScript, you can write:



That works fine - JS doesn’t care about the types of a and b. They can be numbers, strings, anything.

✅ Example:



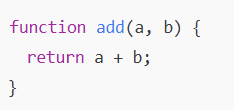
JS says I don’t care what type you pass.

**⚙️ What TypeScript tries to do**

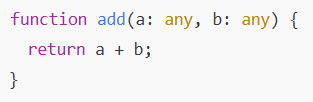
TypeScript wants to know the type of everything (number, string, boolean, etc.)  
If you don’t tell it, it tries to guess (infer).

But if it can’t guess, it gives the variable a default type called any – meaning, I have no idea what this is.

🧩 Example:



Here TypeScript doesn’t know the type of a or b.  
So it secretly thinks:

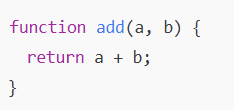


That’s risky because now TypeScript can’t check mistakes - it’s like turning off type safety.

**What noImplicitAny does?**

When you turn on: "noImplicitAny": true

TypeScript will show an error whenever a variable’s type is not explicitly defined or can’t be inferred.

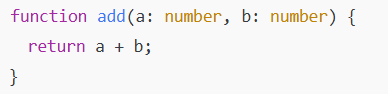


We will get some errors here.

Parameter 'a' implicitly has an 'any' type.

Parameter 'b' implicitly has an 'any' type.

✅ You fix it by adding explicit types:



Now TypeScript knows what you meant.

🧩When noImplicitAny = **false,** TypeScript will **not complain**, and will silently use any (unsafe).

🧩 When noImplicitAny = **true**, TypeScript will **force you to write types** if it can’t figure them out. (safe)

**What is esModuleInterop in TypeScript?**

"esModuleInterop": true  
→ allows **better compatibility between CommonJS and ES Modules**.

**⚙️ Meaning -** It lets you **import old-style modules (like in Node.js)** using **modern ES6 import syntax**.

Without it → you must use the old require () style.  
With it → you can use clean import syntax.

Example (with esModuleInterop: true)

import express from "express"; // ✅ Works fine

Example (with esModuleInterop: false

import express from "express"; // ❌ Error)