TypeScript Mastery

A Step-by-Step Learning Experience

Presented by Sir Ameen Alam

(Part 2)

Step 00 Introduction to TypeScript

TypeScript Mastery: A Step-by-Step Learning Experience (Part1)

https://docs.google.com/presentation/d/1eepDn27eQ8DbRJy7LHTkrIwYx8hQwuo0aDKGS_x10yc

Hello World Program

Introduction to TypeScript, setting up the environment, and writing a simple Hello World program.

Follow TypeScript Mastery: A Step-by-Step Learning Experience (Part1)

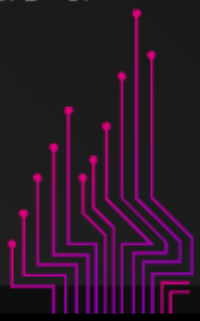
https://docs.google.com/presentation/d/1eepDn27eQ8DbRJy7LHTkrIwYx8hQwuo0aDKGS_x10yc

Step 01
TypeScript Basics

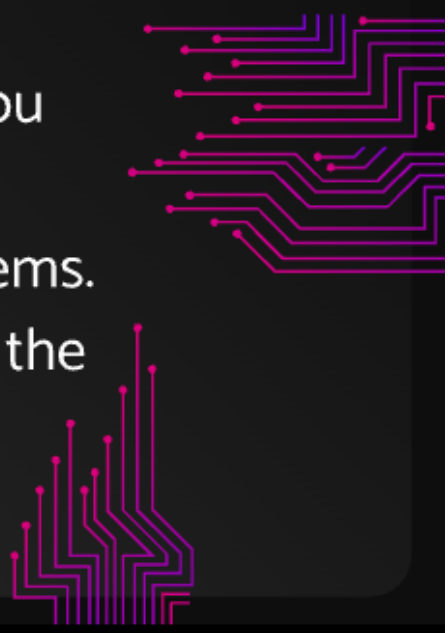
Variables

Variable in Non-Programming

Imagine you have a box where you can store your toys. You can take toys out, put different toys in, or even check to see which toy is currently inside. The box serves as a container for whatever toy you decide to place in it at any time.



Similarly, a variable in programming acts like this box. It's a container in your computer's memory where you can store information, such as numbers, text, or more complex items. Just like you can change the toy in the box, you can also change the information stored in a variable.

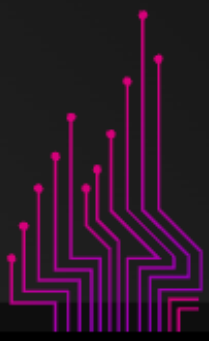


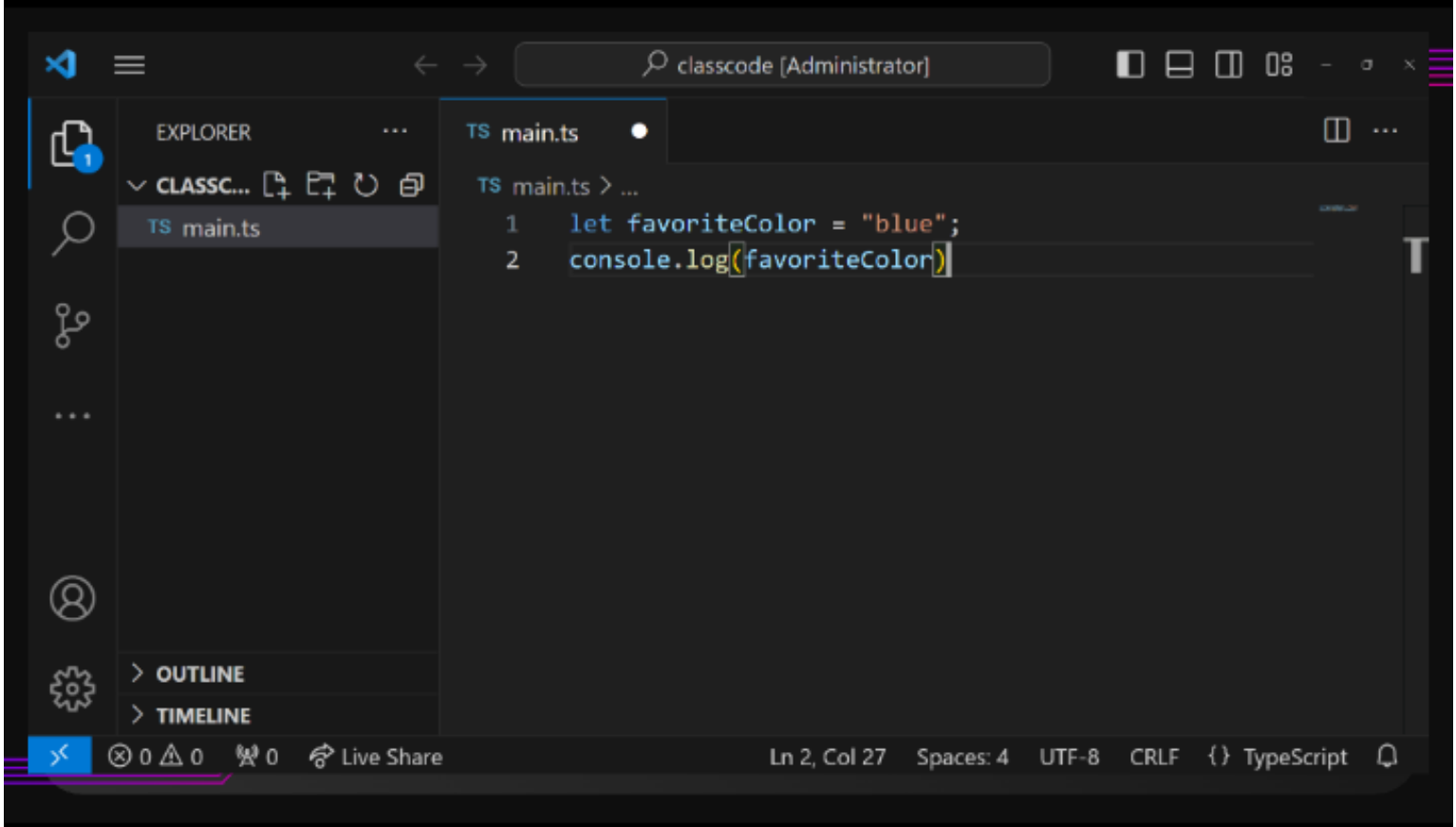
Variable in Programming

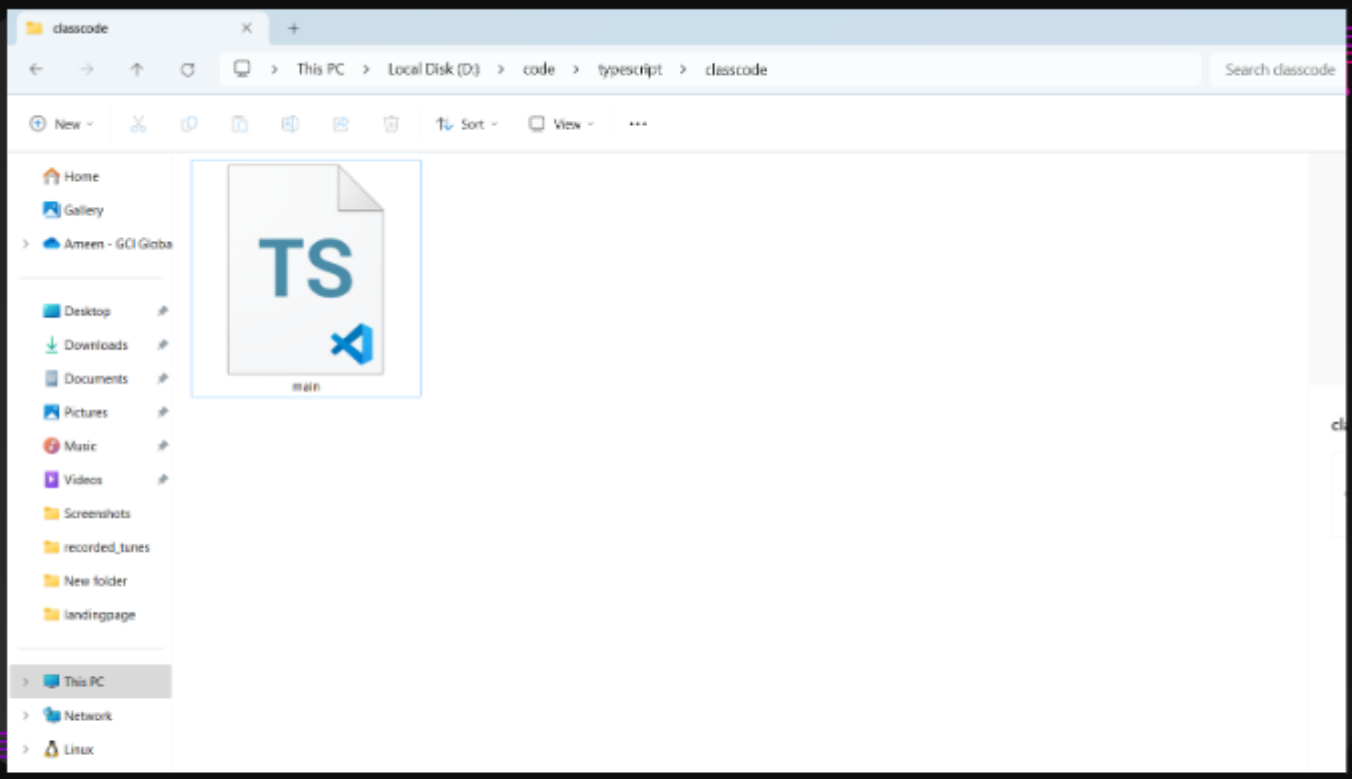
In programming, a variable is used to store data that can be changed or manipulated throughout the execution of a program. Variables have names, so you can refer to them and manage their contents easily.

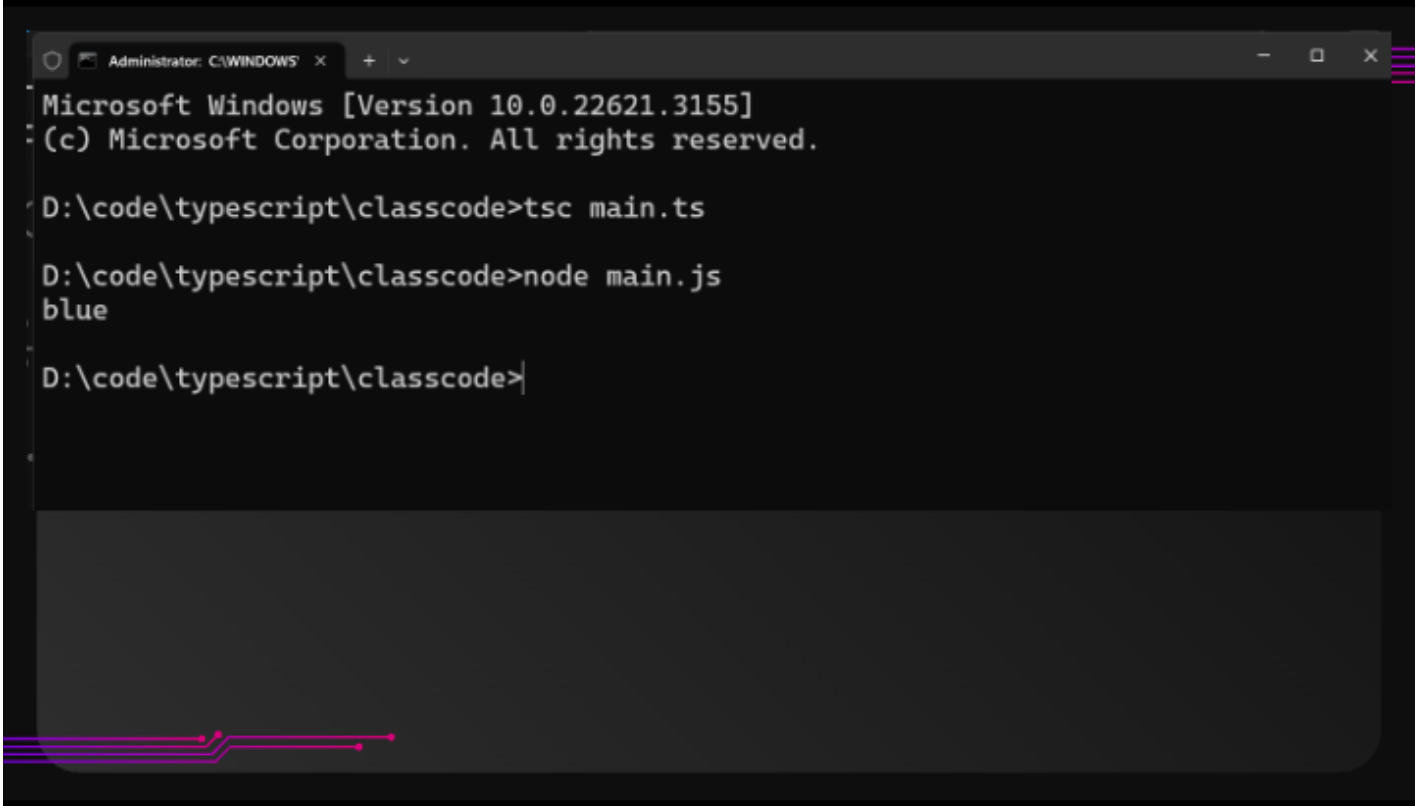
Declares a variable named favoriteColor and assigns it the value ”blue”

```
let favoriteColor = ”blue”;
```



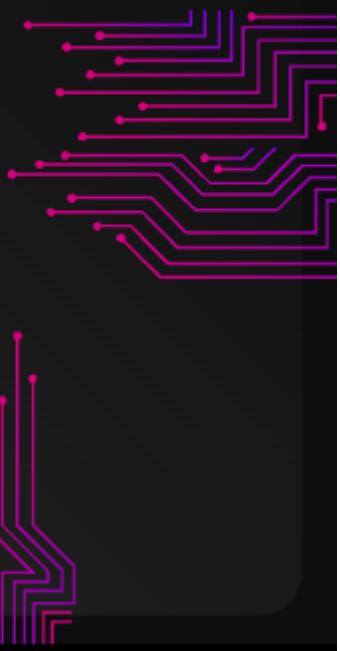

classcode [Administrator]
EXPLORER
CLASSC...
main.ts
main.ts
main.ts > ...
1 let favoriteColor = "blue";
2 console.log(favoriteColor)
OUTLINE
TIMELINE
0 0
0
Live Share
Ln 2, Col 27
Spaces: 4
UTF-8
CRLF
TypeScript

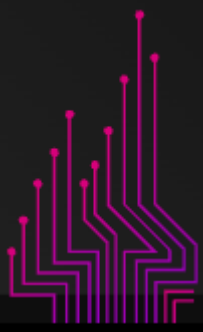

classcode
This PC
Local Disk (D:)
code
typescript
classcode
Search classcode
New
Sort
View
Home
Gallery
Ameen - GCI Globa
Desktop
Downloads
Documents
Pictures
Music
Videos
Screenshots
recorded_tunes
New folder
landingpage
This PC
Network
Linux
TS
main

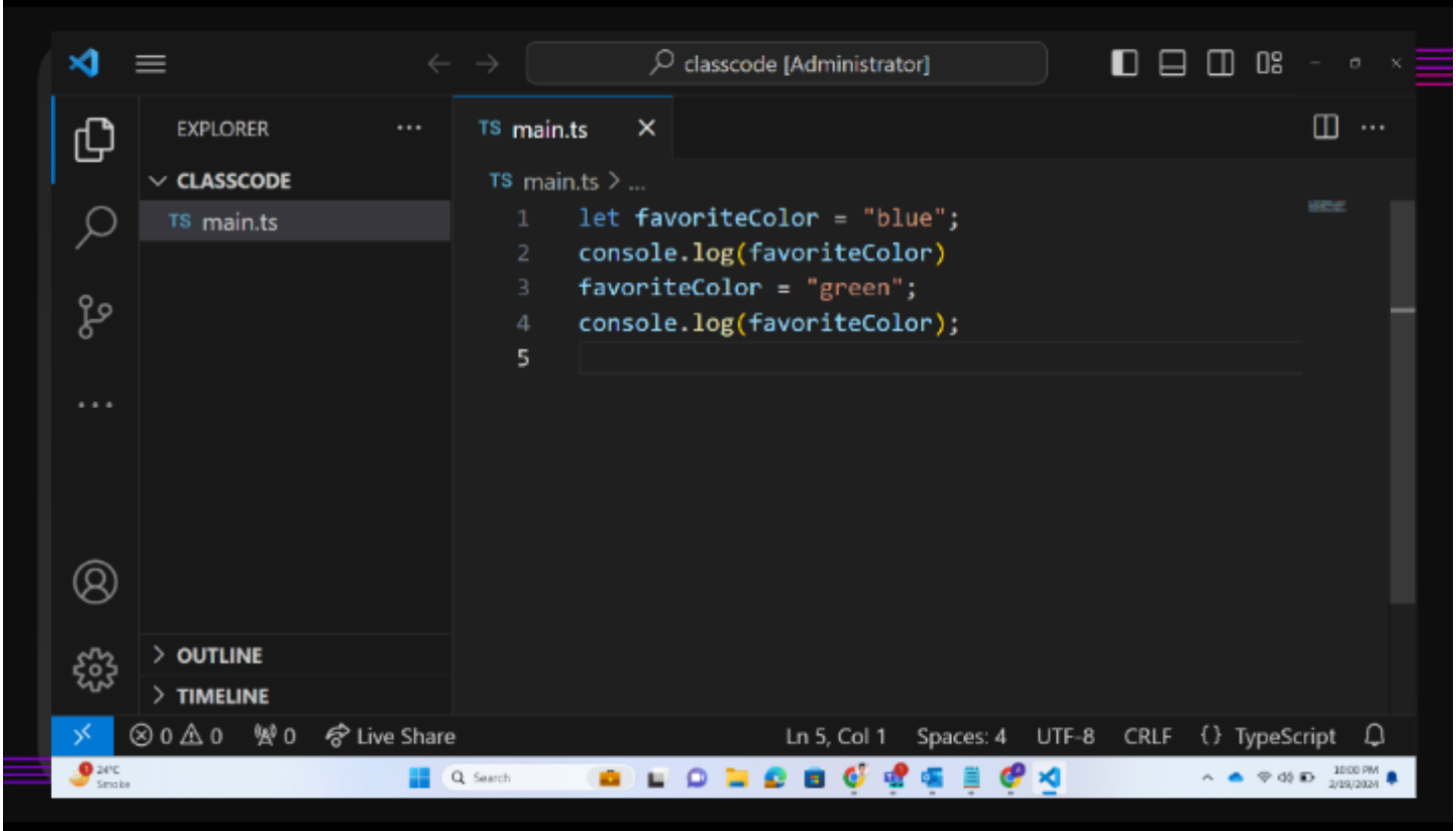

Administrator: C:\WINDOWS\
Microsoft Windows [Version 10.0.22621.3155]
(c) Microsoft Corporation. All rights reserved.
D:\code\typescript\classcode>tsc main.ts
D:\code\typescript\classcode>node main.js
blue
D:\code\typescript\classcode>

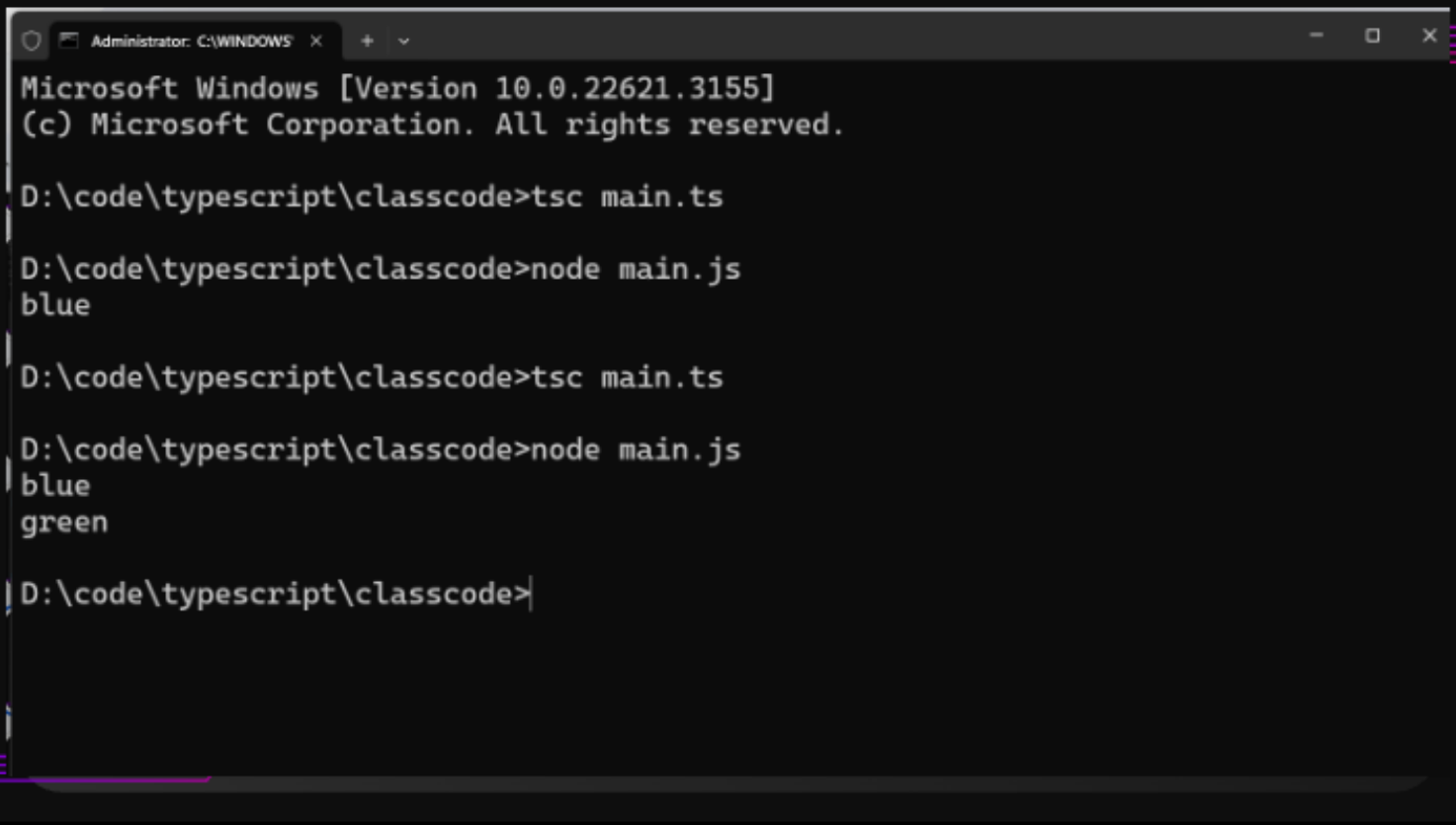
Changes the value of favoriteColor to ”green”

```
let favoriteColor = ”blue”;
favoriteColor = ”green”;
```



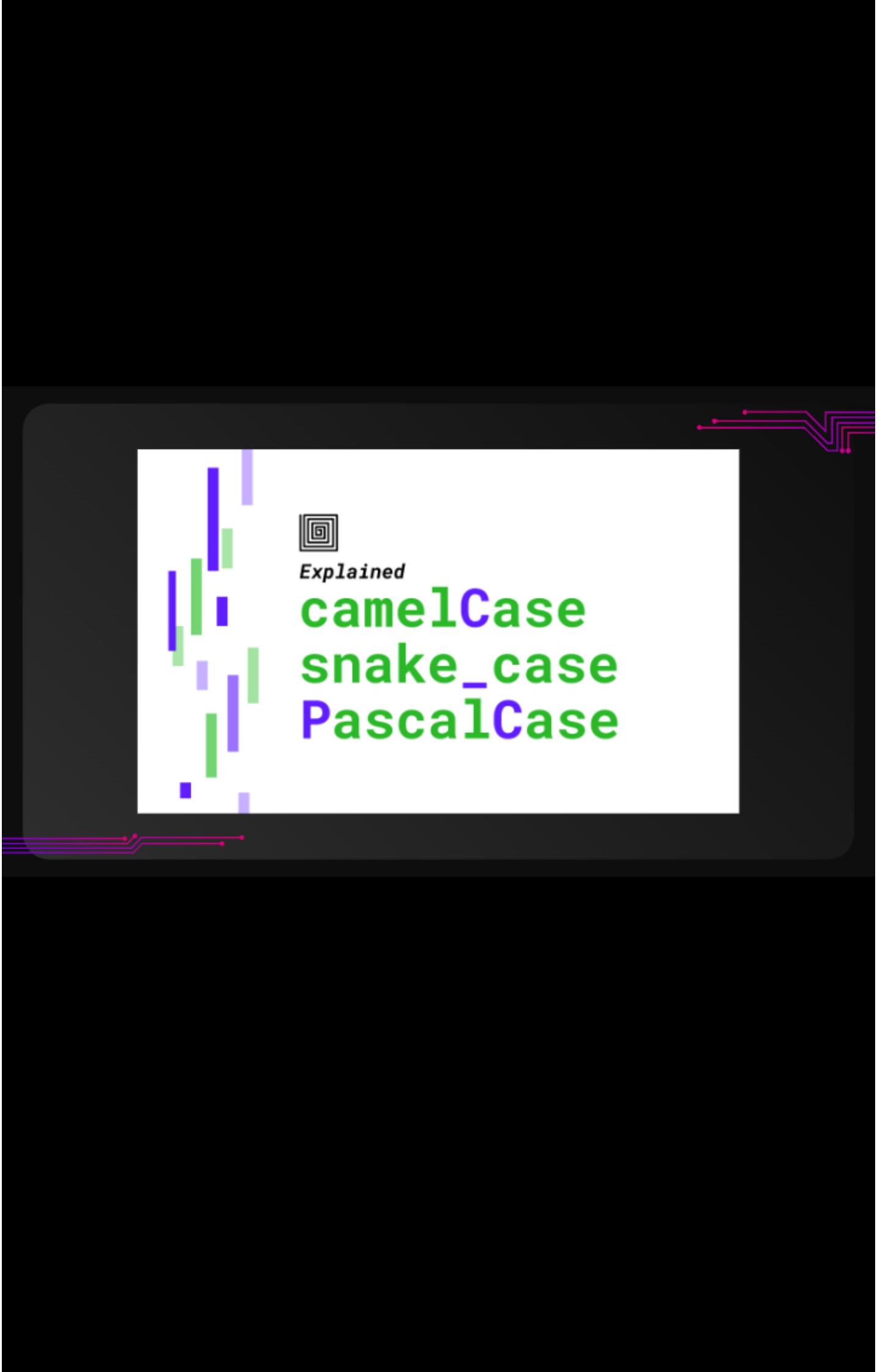



classcode [Administrator]
EXPLORER
CLASSCODE
main.ts
OUTLINE
TIMELINE
main.ts
main.ts > ...
1 let favoriteColor = "blue";
2 console.log(favoriteColor)
3 favoriteColor = "green";
4 console.log(favoriteColor);
5
Live Share
Ln 5, Col 1
Spaces: 4
UTF-8
CRLF
TypeScript

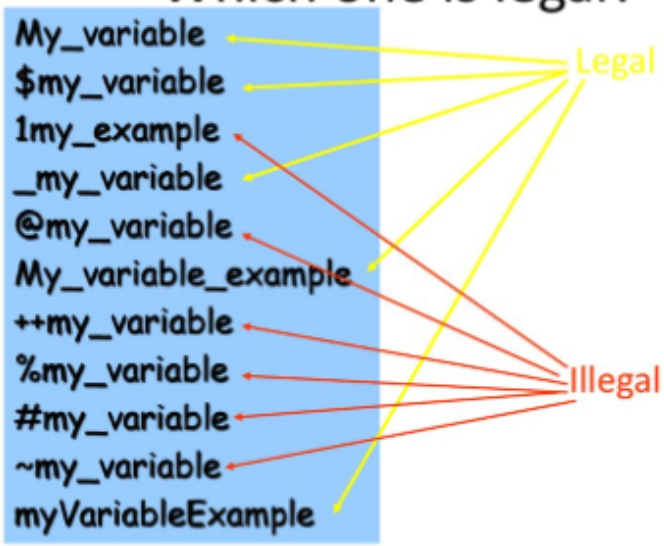

Administrator: C:\WINDOWS
Microsoft Windows [Version 10.0.22621.3155]
(c) Microsoft Corporation. All rights reserved.
D:\code\typescript\classcode>tsc main.ts
D:\code\typescript\classcode>node main.js
blue
D:\code\typescript\classcode>tsc main.ts
D:\code\typescript\classcode>node main.js
blue
green
D:\code\typescript\classcode>

In the example above:
We declare a variable favoriteColor and initially assign it the value ”blue”.
Later, we change the value of favoriteColor to ”green”.

case sensitive


Explained
camelCase
snake_case
PascalCase

Which one is legal?



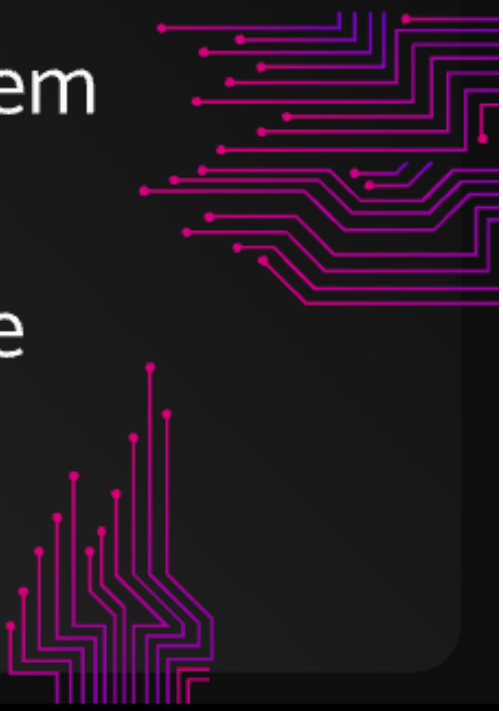
Strong Typing

Non-Programming

Imagine you're organizing a big event and you've assigned specific roles to your team members:
photographers, decorators, and security personnel.
Each role has clear expectations and tasks that cannot be exchanged with others without causing confusion or issues.

For example, asking a photographer to handle security would not be ideal because each person's skills and tools are suited to their specific role.

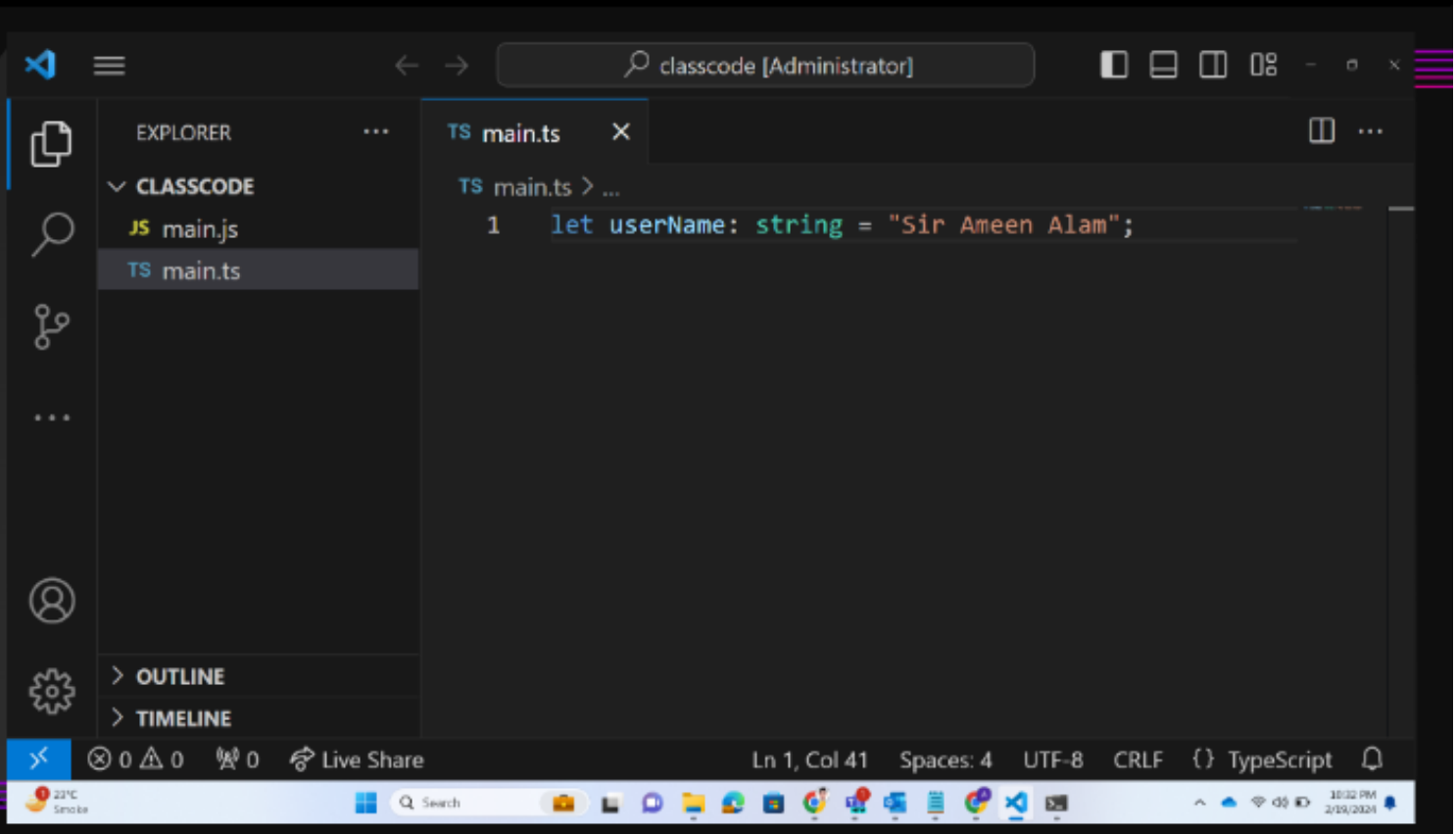
This scenario is similar to how TypeScript's type system works. Just like assigning specific roles helps manage your event smoothly.



TypeScript assigns types to variables to ensure they are used correctly throughout your code. This helps prevent errors, such as mixing up numbers with text or trying to perform operations on incompatible data types.

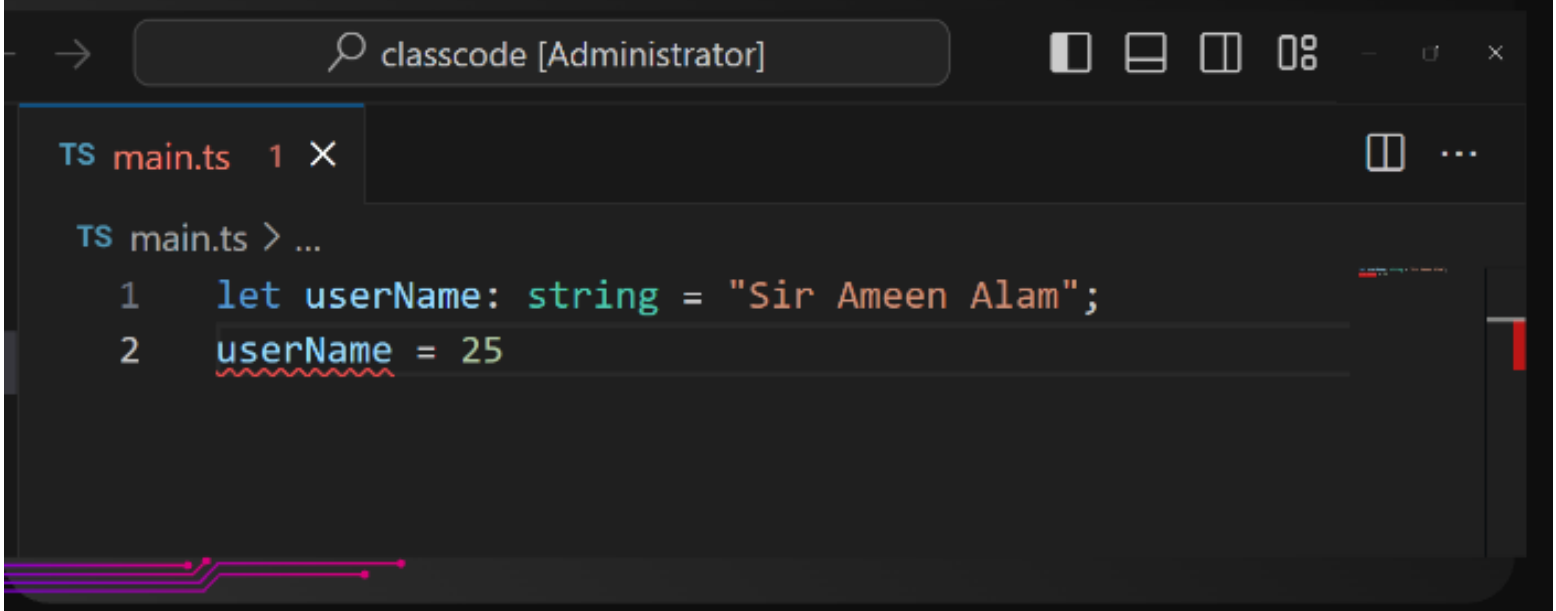
Strong Typing in TypeScript

TypeScript enhances JavaScript by adding a type system. This means you can specify what type of data (such as numbers, strings, or more complex objects) can be stored in a variable. This specification helps catch errors during development, making your code more robust and maintainable.

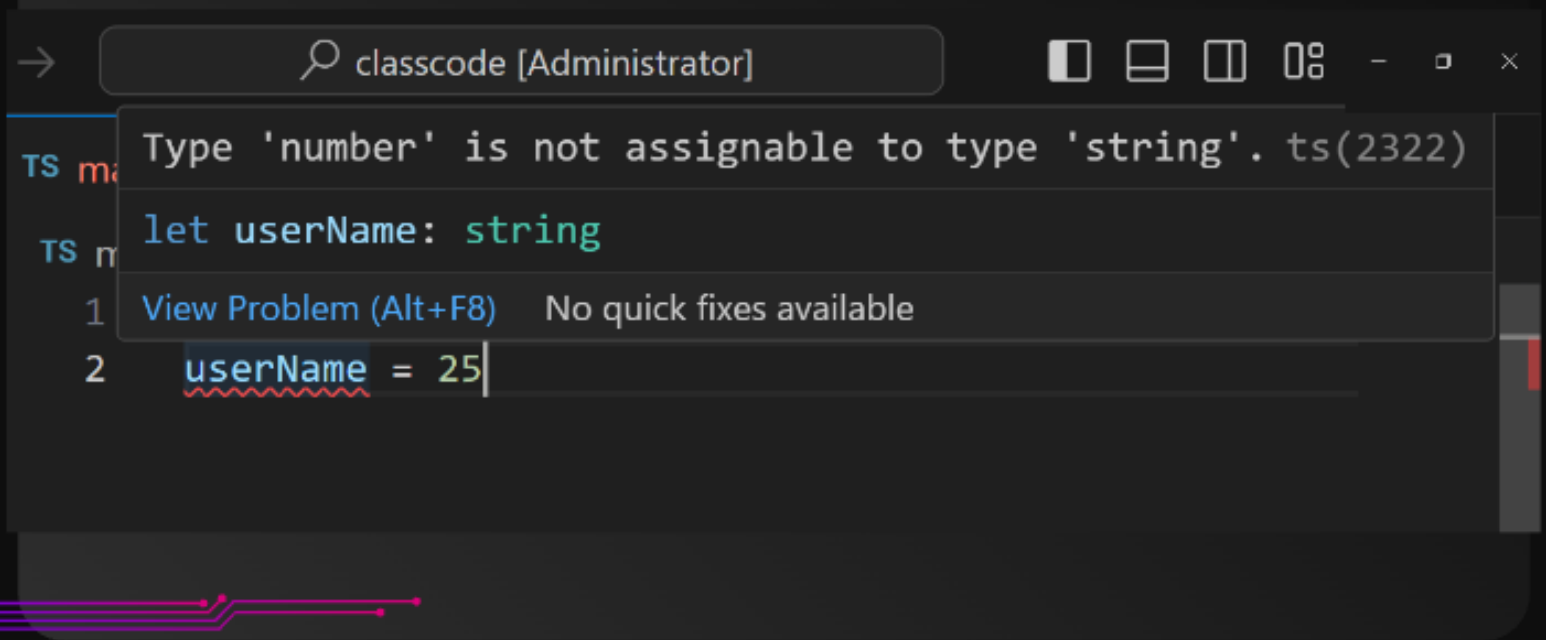

classcode [Administrator]
EXPLORER
CLASSCODE
main.js
main.ts
main.ts
main.ts > ...
1 let userName: string = "Sir Ameen Alam";
OUTLINE
TIMELINE
0 0
0
Live Share
Ln 1, Col 41
Spaces: 4
UTF-8
CRLF
TypeScript

Hover your mouse over the Variable

Error: Type 'number' is not assignable to type 'string'.

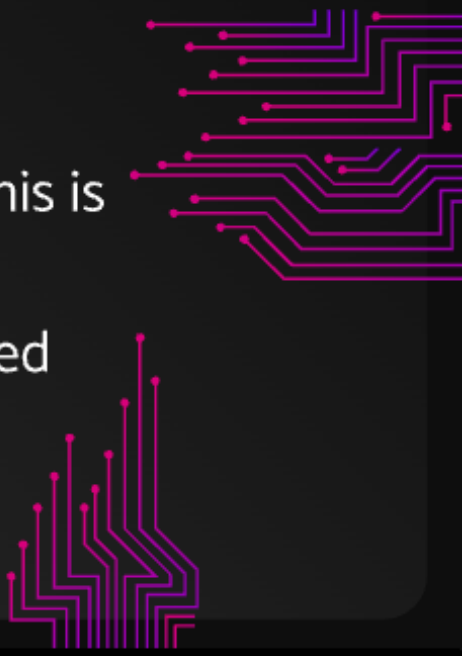


Hover your mouse over the Variable



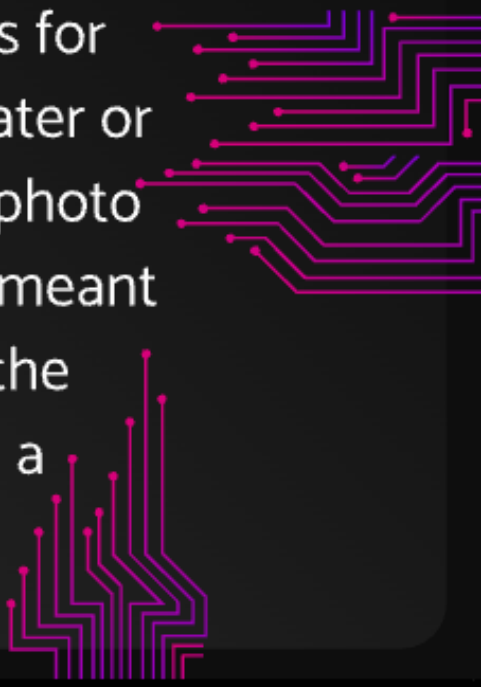
In the example above:

Attempting to assign a number to userName would result in an error. This is TypeScript enforcing strong typing, ensuring that variables are always used with the correct type of data



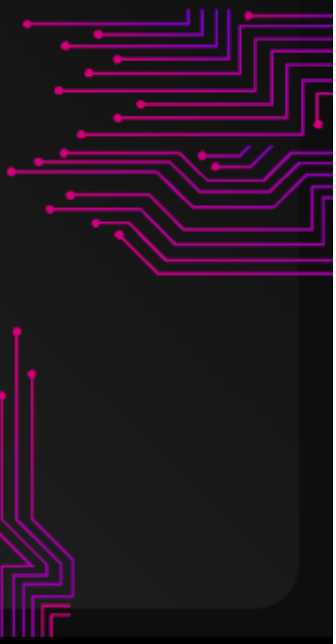
Non-Programming

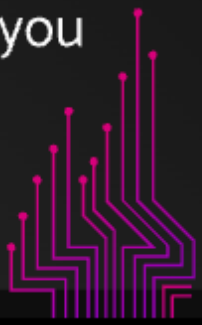
Imagine you're packing for a vacation, and you have different types of containers for your items: a bottle for liquids (like water or shampoo), a wallet for money, and a photo album for pictures. Each container is meant for a specific type of item, and using the wrong container (like putting liquid in a photo album) wouldn't make sense.



TypeScript uses data types to know what kind of data is being stored and manipulated.

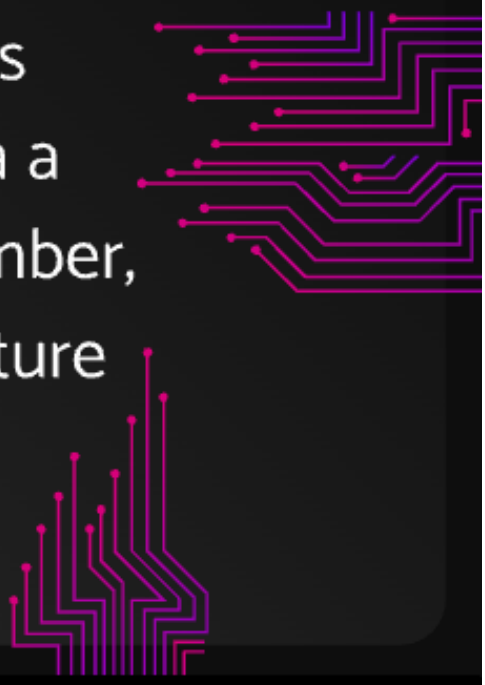
For example, you wouldn't store text in a variable meant for numbers, just like you wouldn't put water in a photo album.





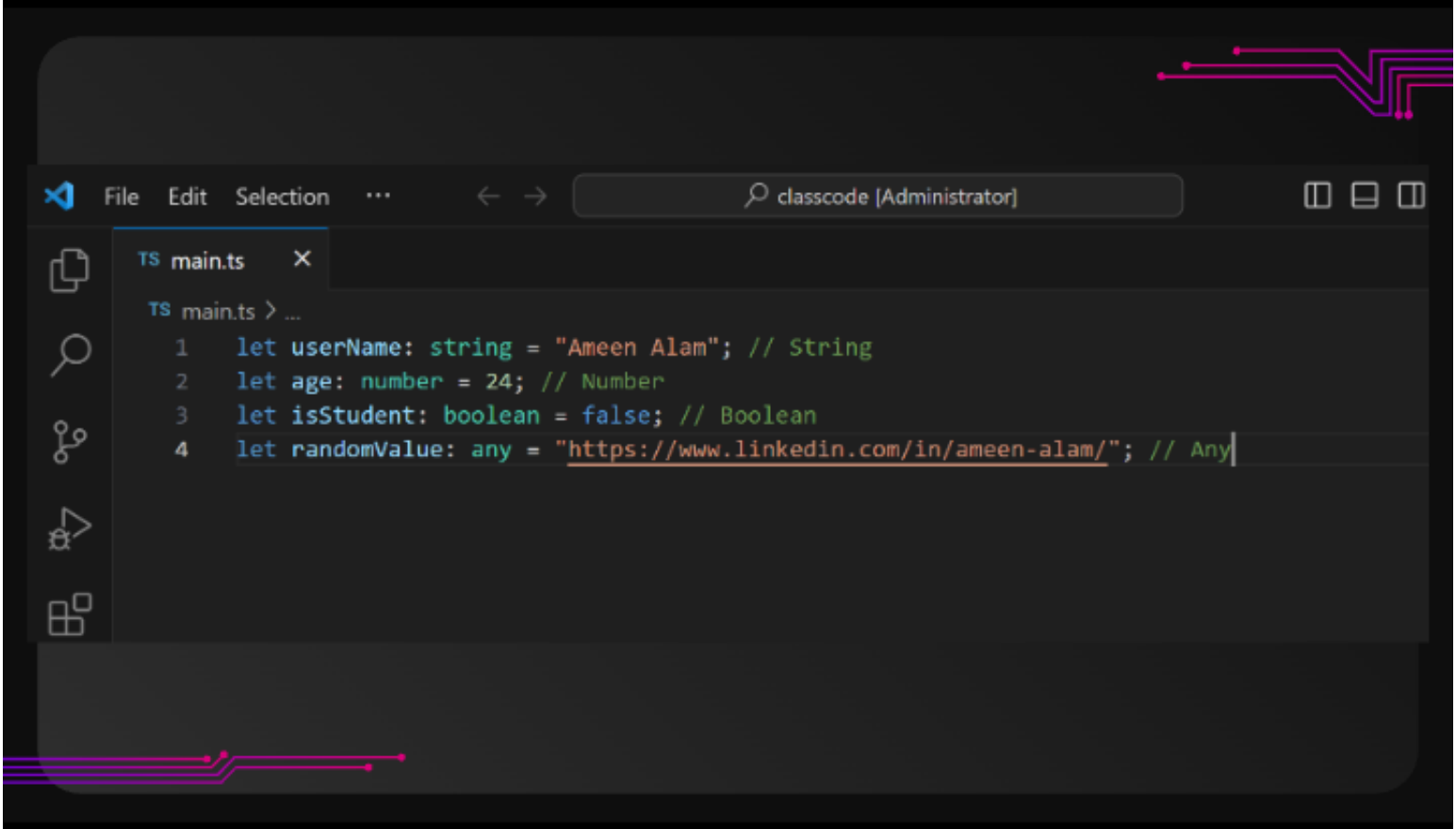
Data Types in TypeScript

TypeScript enhances JavaScript by adding explicit types. This allows you to specify what kind of data a variable can hold, such as a number, string, or a more complex structure like an array or object.



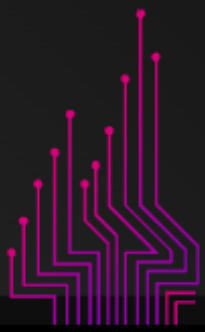
Here are some basic data types in TypeScript

- **String:** For textual data
- **Number:** For numerical values (integers and floating-point numbers)
- **Boolean:** For true/false values
- **Any:** For variables that can hold any type of data

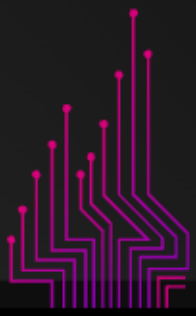

File
Edit
Selection
classcode [Administrator]
main.ts
main.ts > ...
1 let userName: string = "Ameen Alam"; // String
2 let age: number = 24; // Number
3 let isStudent: boolean = false; // Boolean
4 let randomValue: any = "https://www.linkedin.com/in/ameen-alam/"; // Any

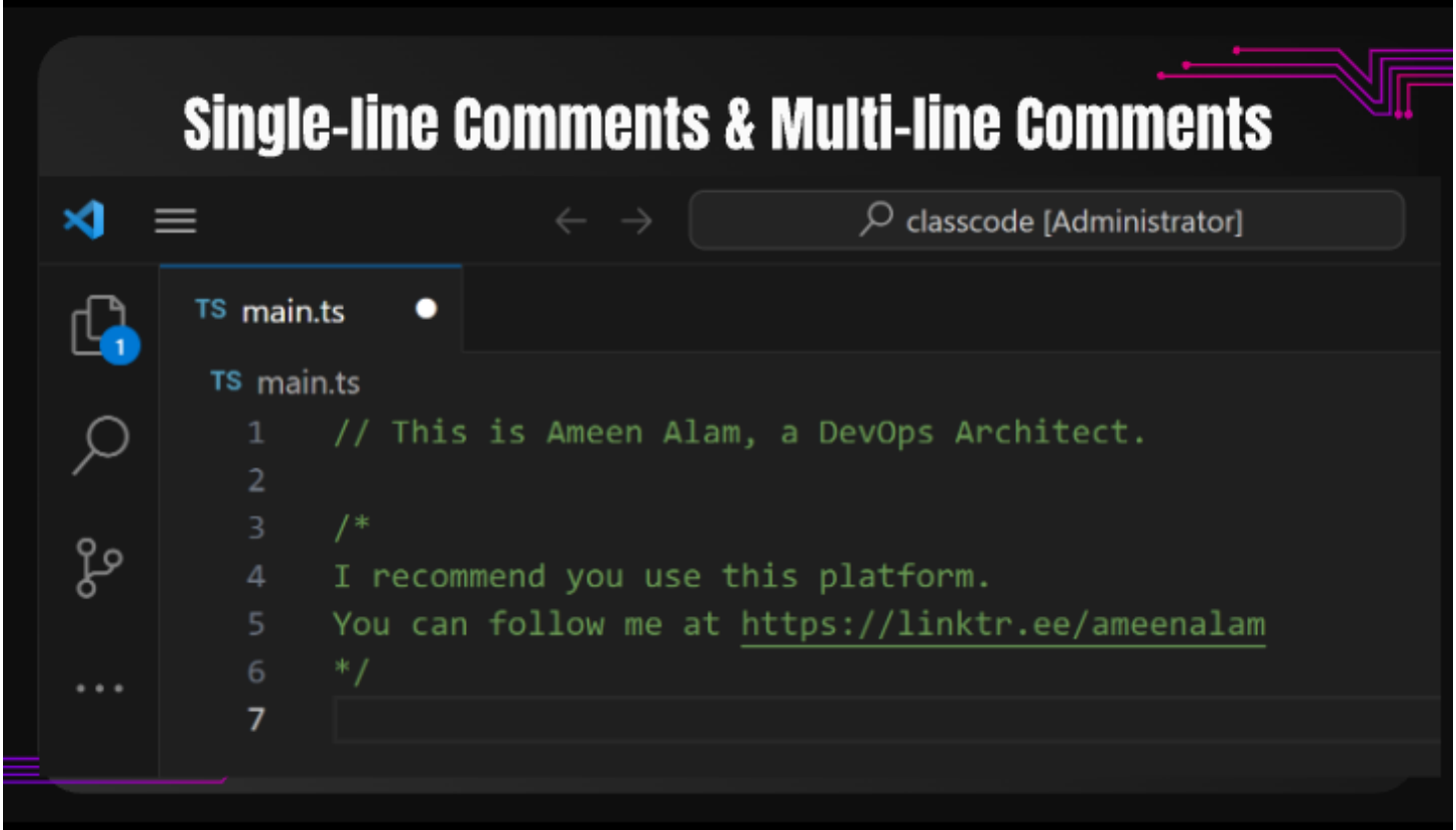
Comments in Code

Comments in code are annotations added to the source code of a program to provide explanations, clarifications, or notes to anyone reading the code.

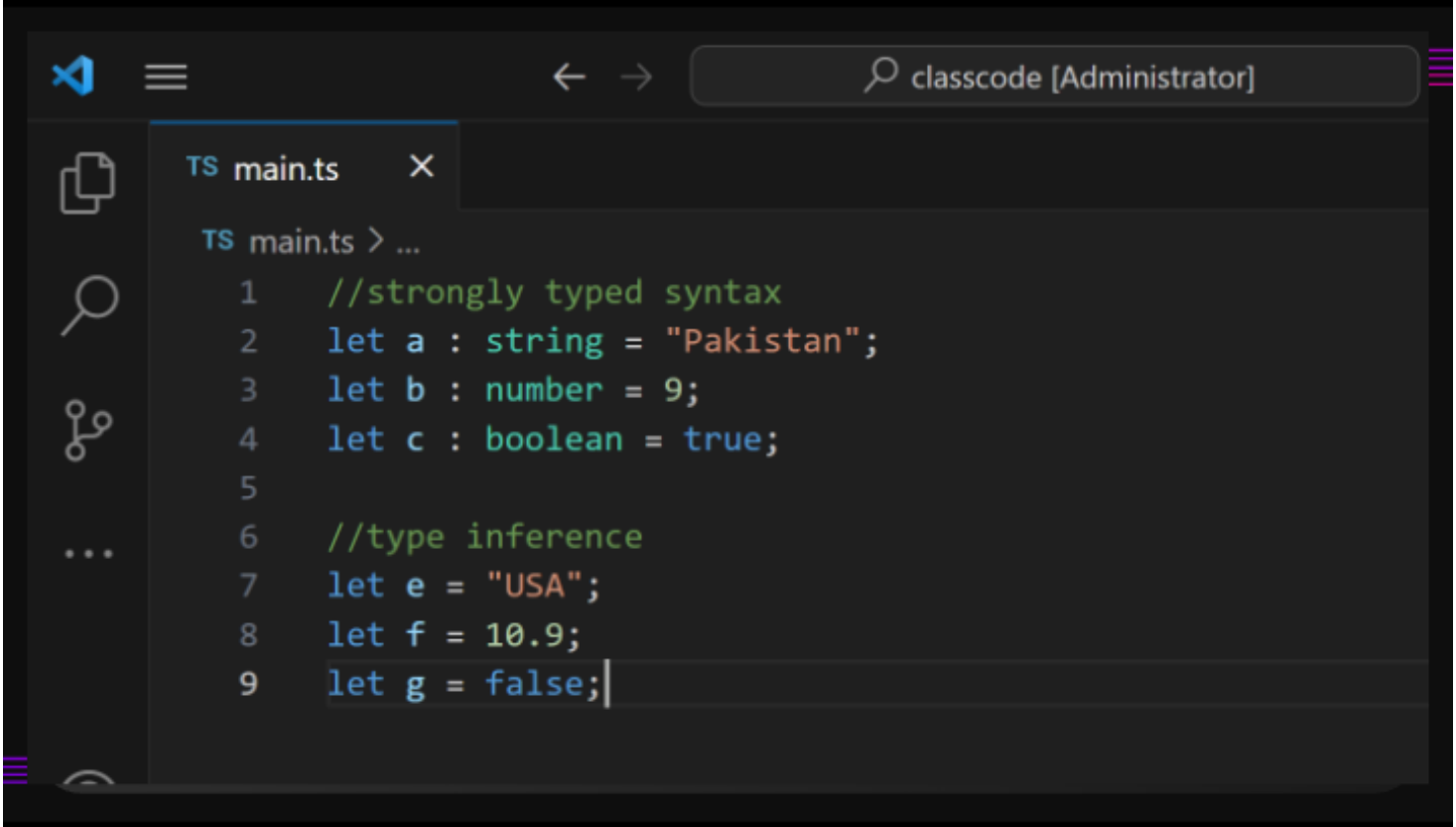


These comments are ignored by the compiler or interpreter, so they do not affect the execution of the program.




Single-line Comments & Multi-line Comments
classcode [Administrator]
TS main.ts
TS main.ts
1 // This is Ameen Alam, a DevOps Architect.
2
3 /*
4 I recommend you use this platform.
5 You can follow me at https://linktr.ee/ameenalam
6 */
7

Type Inference


classcode [Administrator]
TS main.ts
TS main.ts > ...
1 //strongly typed syntax
2 let a : string = "Pakistan";
3 let b : number = 9;
4 let c : boolean = true;
5
6 //type inference
7 let e = "USA";
8 let f = 10.9;
9 let g = false;

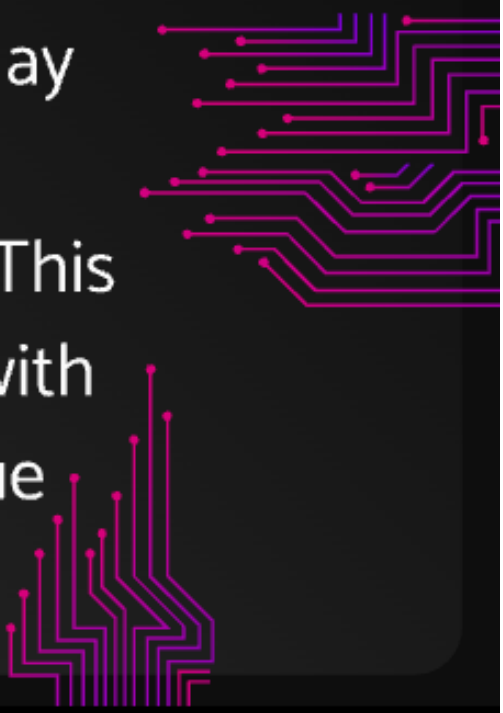
Variables Advance

Const and Let

Non-Programming

Consider two scenarios involving money in your wallet:

Let: The amount of money you have changes frequently. One day you might have $50, and after shopping, you might have $20. This is similar to variables declared with let in TypeScript, where the value can change over time.

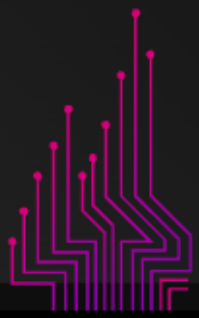


Const: Your bank account number, on the other hand, is a constant. Once it's set, it doesn't change. This mirrors the const declaration in TypeScript, signifying that once a variable is assigned a value, it cannot be reassigned to something else.

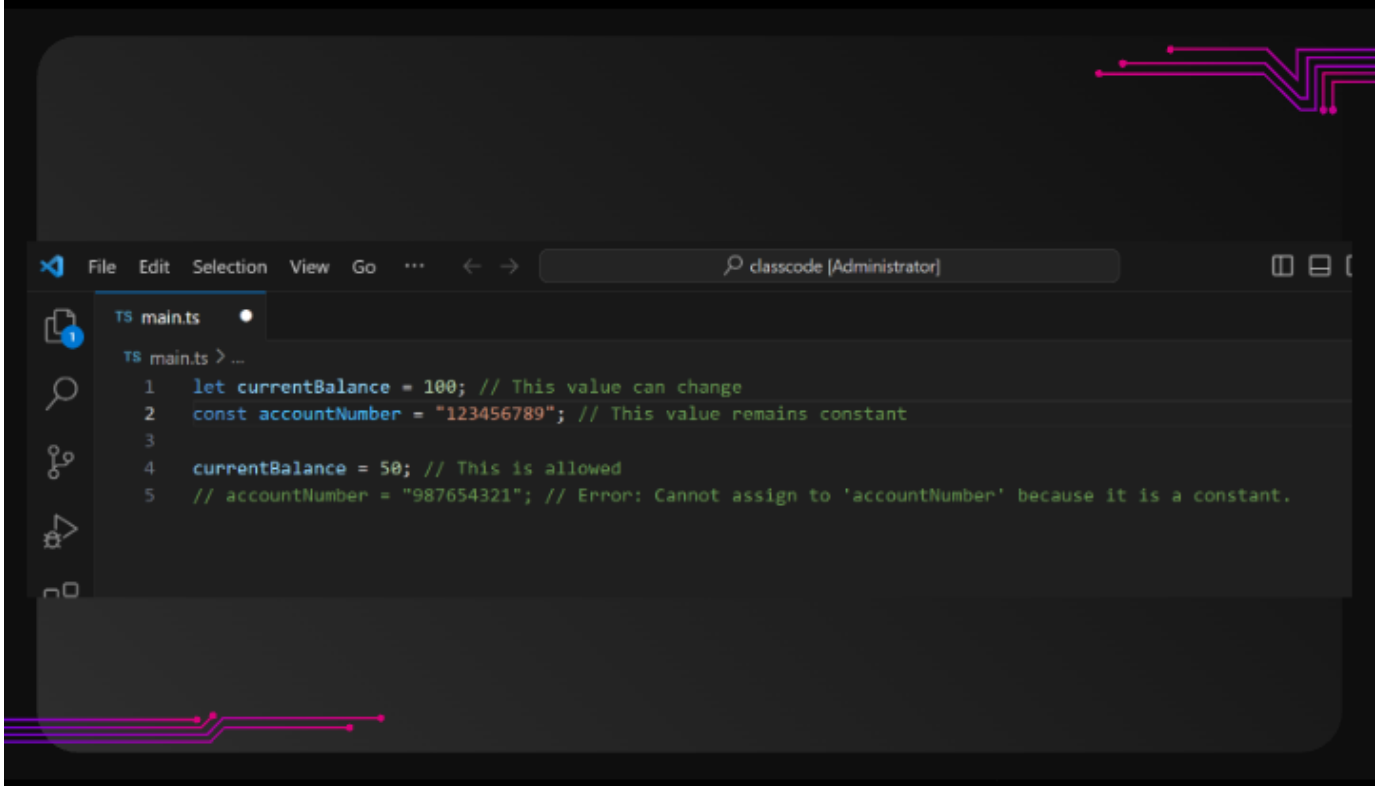
Const and Let in TypeScript

In TypeScript, let and const are two ways to declare variables, with key differences in their mutability and scope.

Using let: This declares a variable that can be reassigned to a different value. It's useful for values that change over time, like counters or values that depend on conditions.



Using const: This declares a constant that cannot be reassigned once set. It's perfect for values that should remain the same throughout the execution of your program, like configuration settings or important identifiers.


File Edit Selection View Go
classcode [Administrator]
main.ts
main.ts > ...
1 let currentBalance = 100; // This value can change
2 const accountNumber = "123456789"; // This value remains constant
3
4 currentBalance = 50; // This is allowed
5 // accountNumber = "987654321"; // Error: Cannot assign to 'accountNumber' because it is a constant.

Additional Data Types

Undefined: Represents a variable that has not been assigned a value, or not initialized. It is one of JavaScript's primitive types that TypeScript adopts.

Unknown: Used for variables where the type is not known at the time of writing the code. It's a safer alternative to any, as it requires the type to be determined before it can be used.

BigInt: A data type that can store numbers larger than the maximum limit for the number type. It allows representation of very large integers.

Symbol: A unique and immutable primitive value that can be used as the key of an Object property. Symbols are often used to add unique property keys to an object that won't collide with keys any other code might add to the object, and which are hidden from any mechanisms other code will typically use to access the object.

Null: Represents the intentional absence of any object value. It is another primitive type in JavaScript used in TypeScript to signify that a variable intentionally has no value.

Hoisting


JS
HOISTING WITH
LET AND CONST

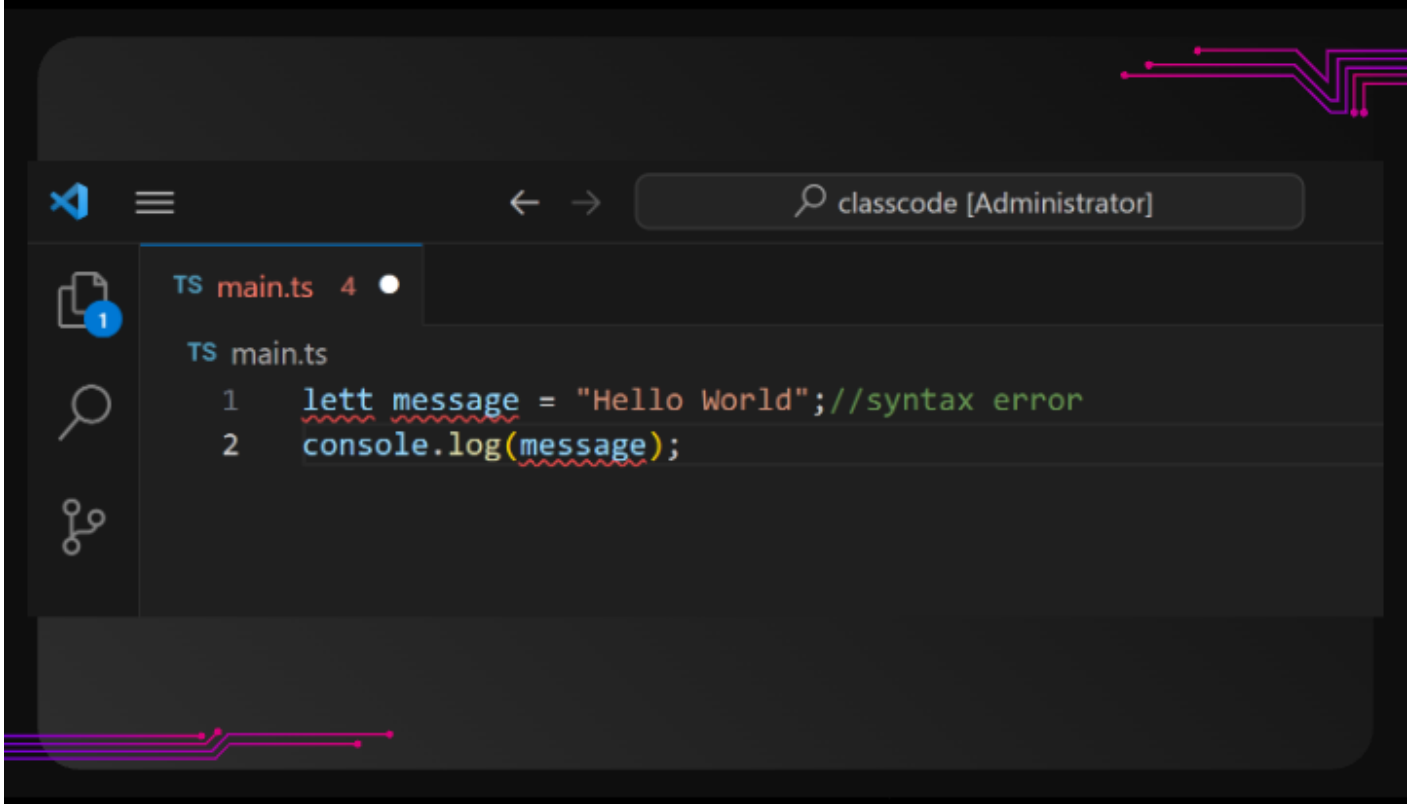
Hoisting in JavaScript with let and const – and How it Differs from var

https://www.freecodecamp.org/news/javascript-let-and-const-hoisting

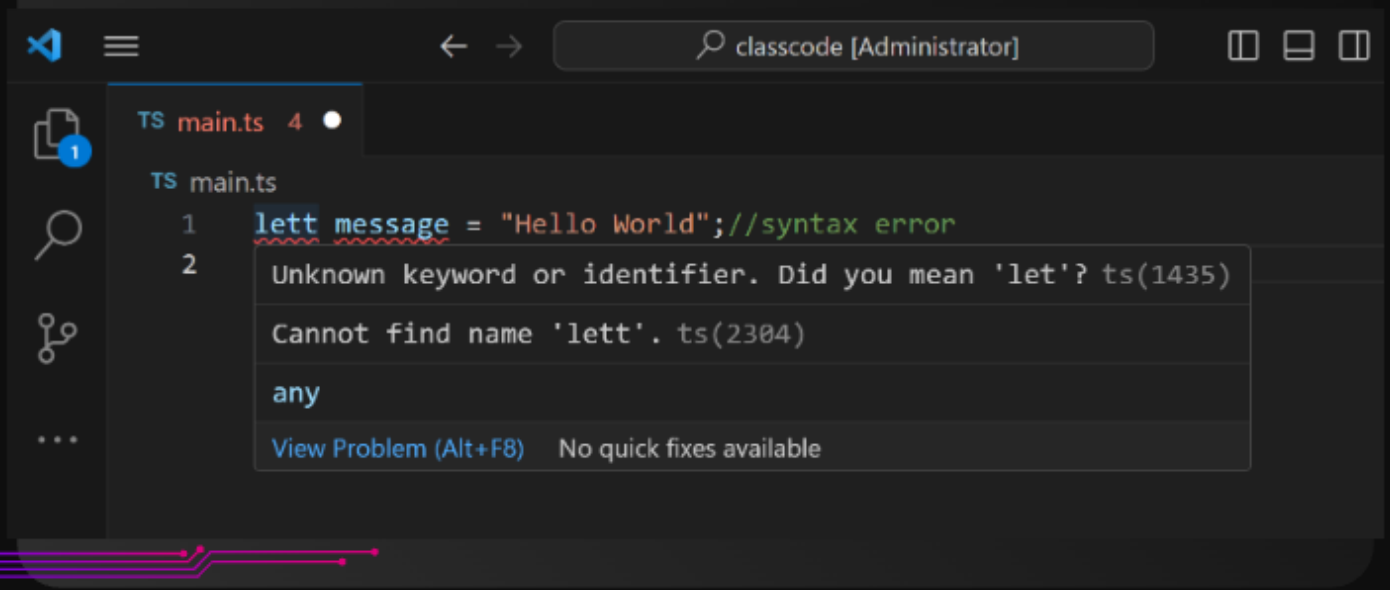
Errors

Syntax Error

Identifying and resolving syntax errors in TypeScript code.


classcode [Administrator]
TS main.ts 4
TS main.ts
1 lett message = "Hello World";//syntax error
2 console.log(message);

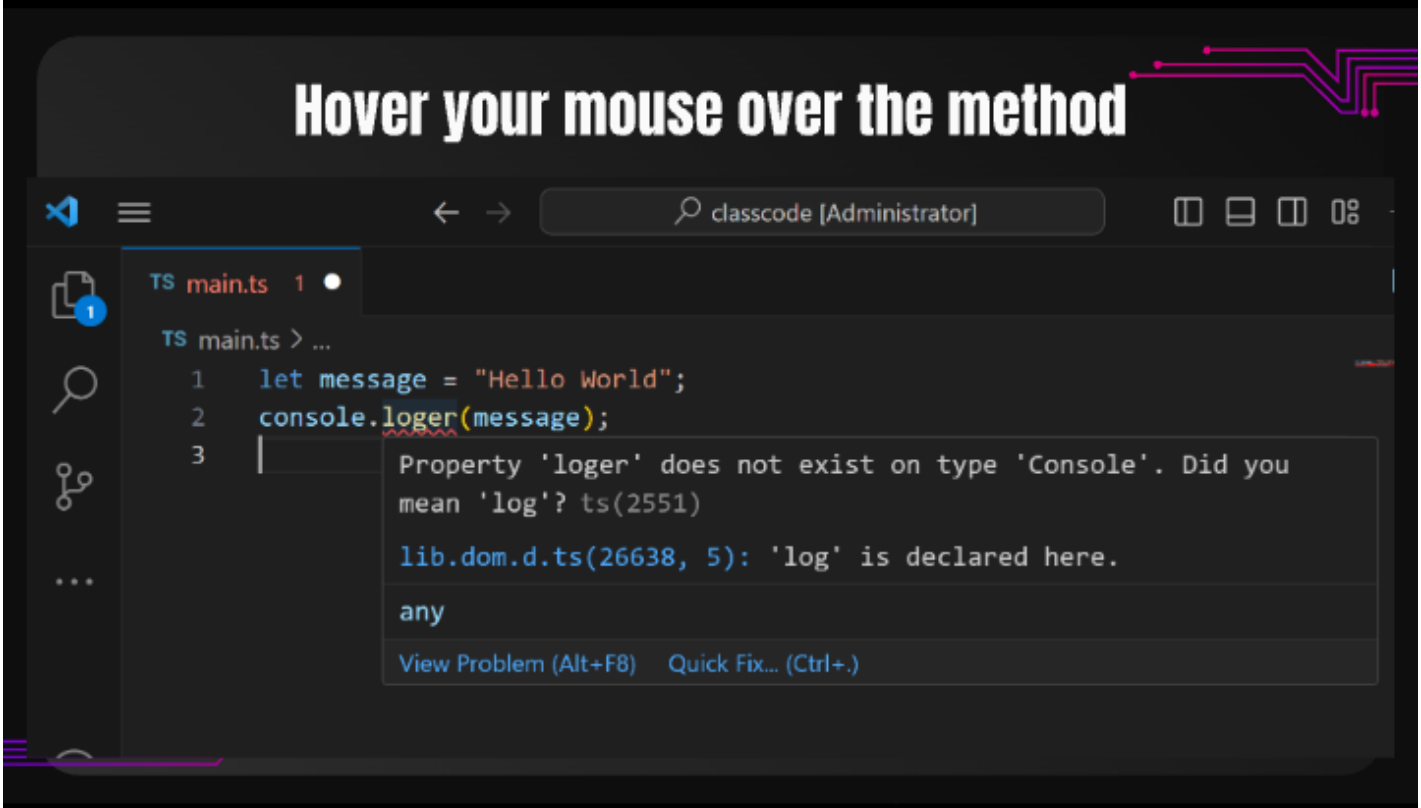
Hover your mouse over the Keyword



Type Error

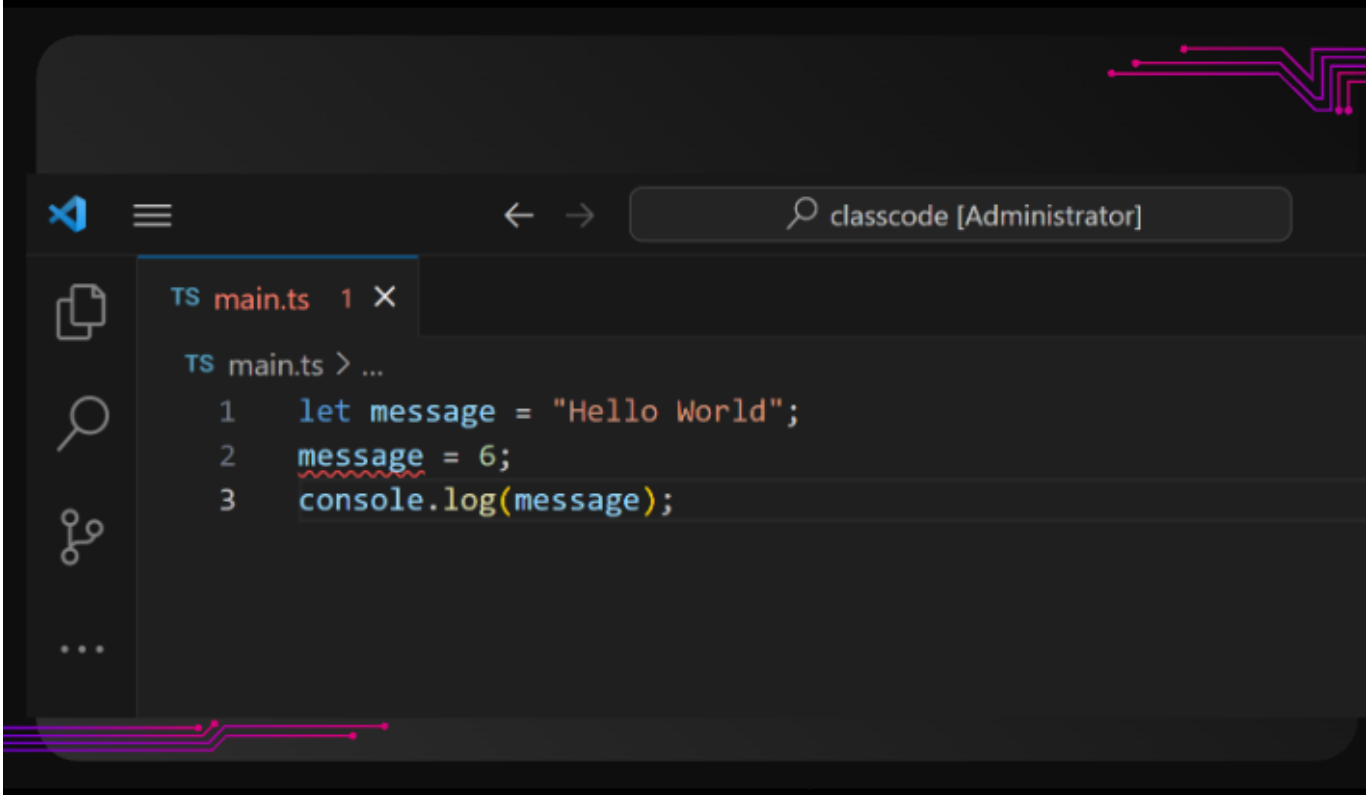
Understanding type errors and how TypeScript's type system helps prevent them.

Hover your mouse over the method

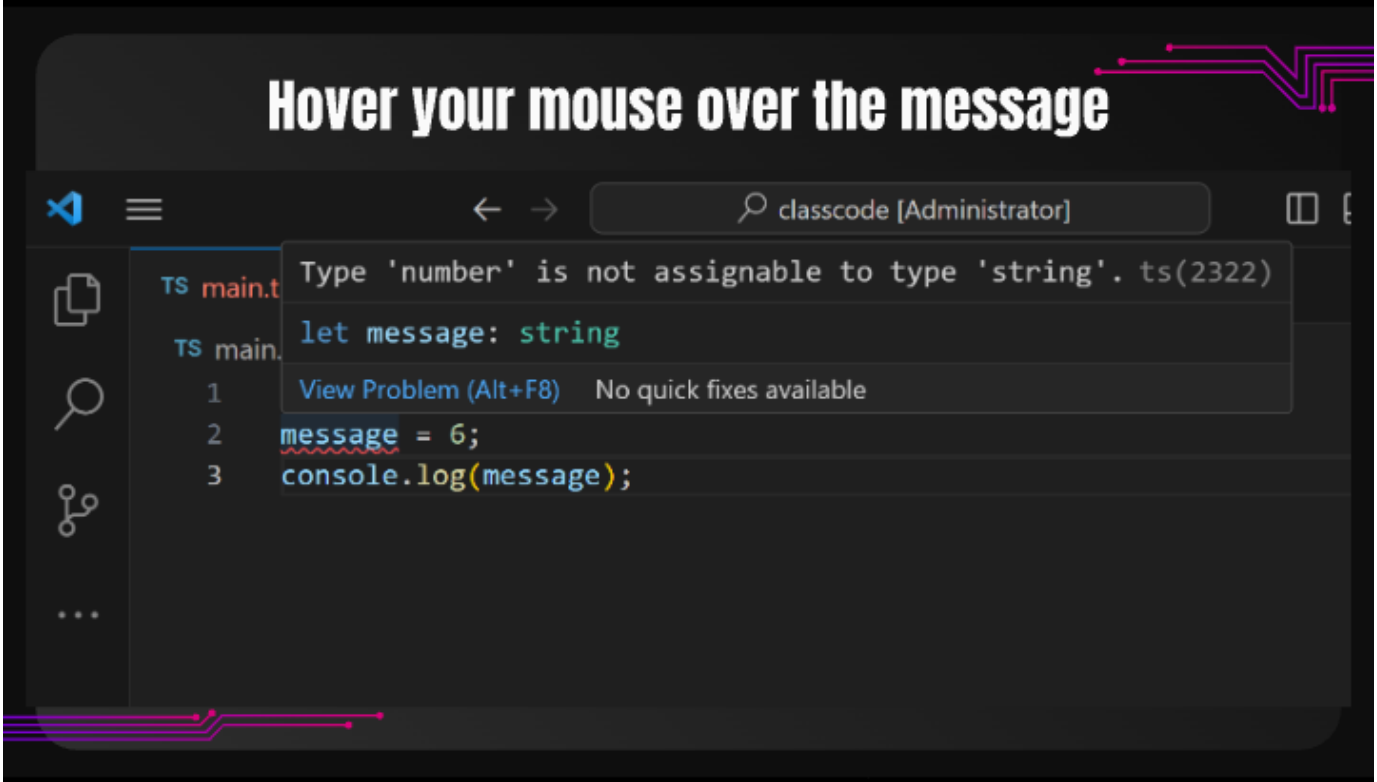


Assignability Error

Learning about assignability errors and how to address them through type compatibility.


classcode [Administrator]
TS main.ts 1
TS main.ts > ...
1 let message = "Hello World";
2 message = 6;
3 console.log(message);

Hover your mouse over the message

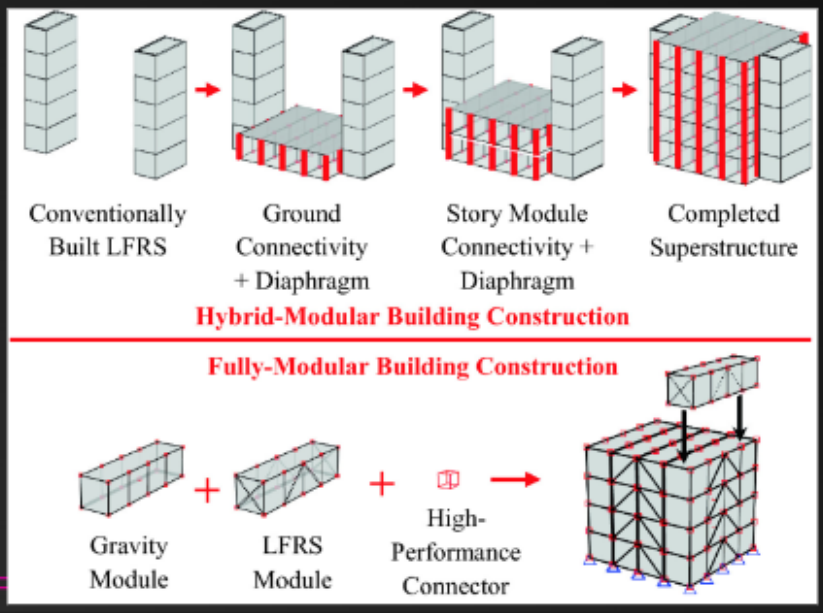


Step 03
Modules

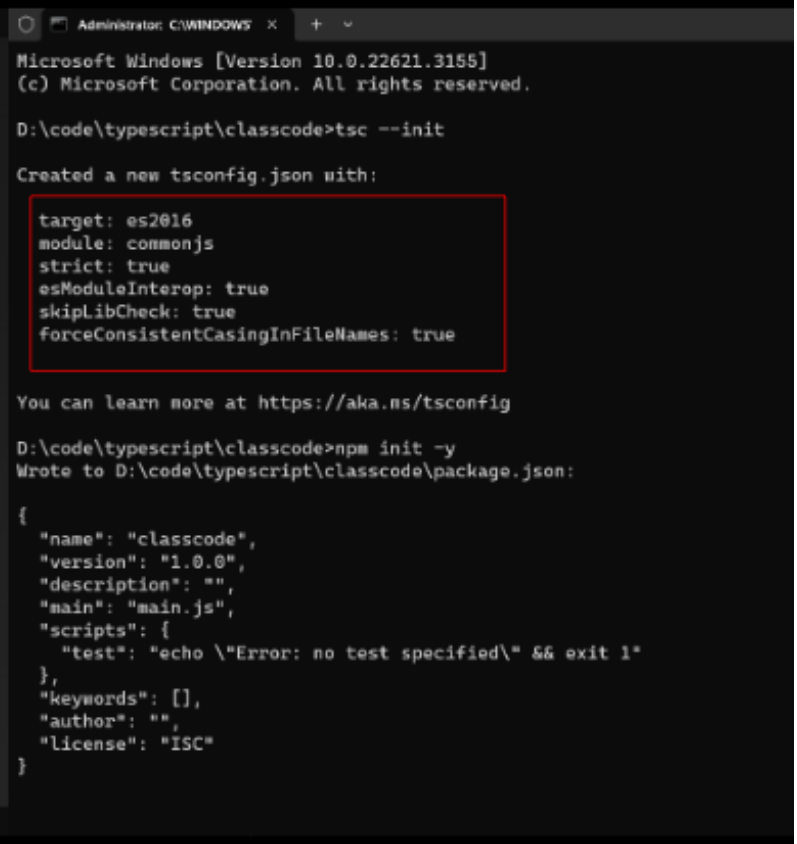
Basics of modules, exporting, and importing modules.

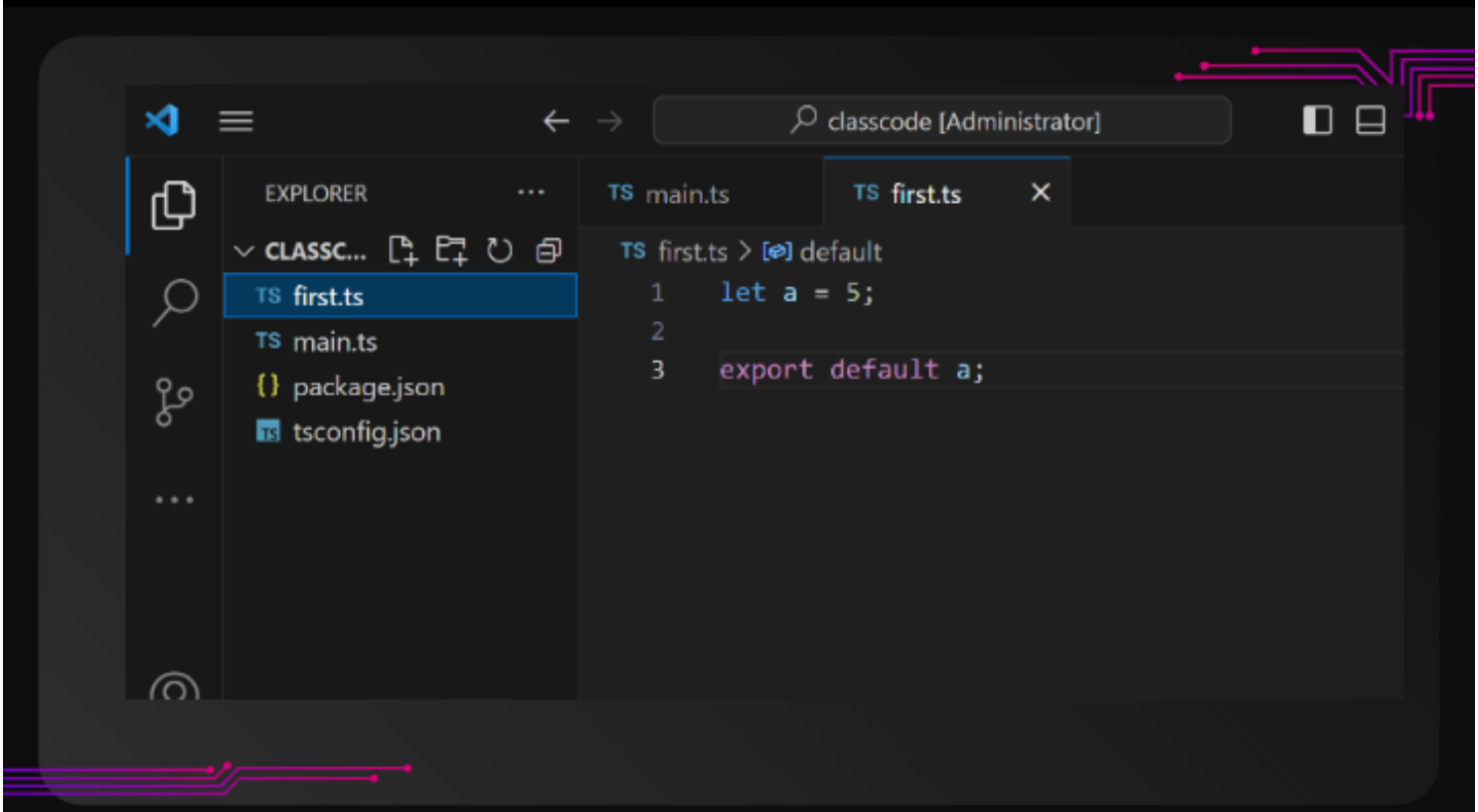
Modular Architecture

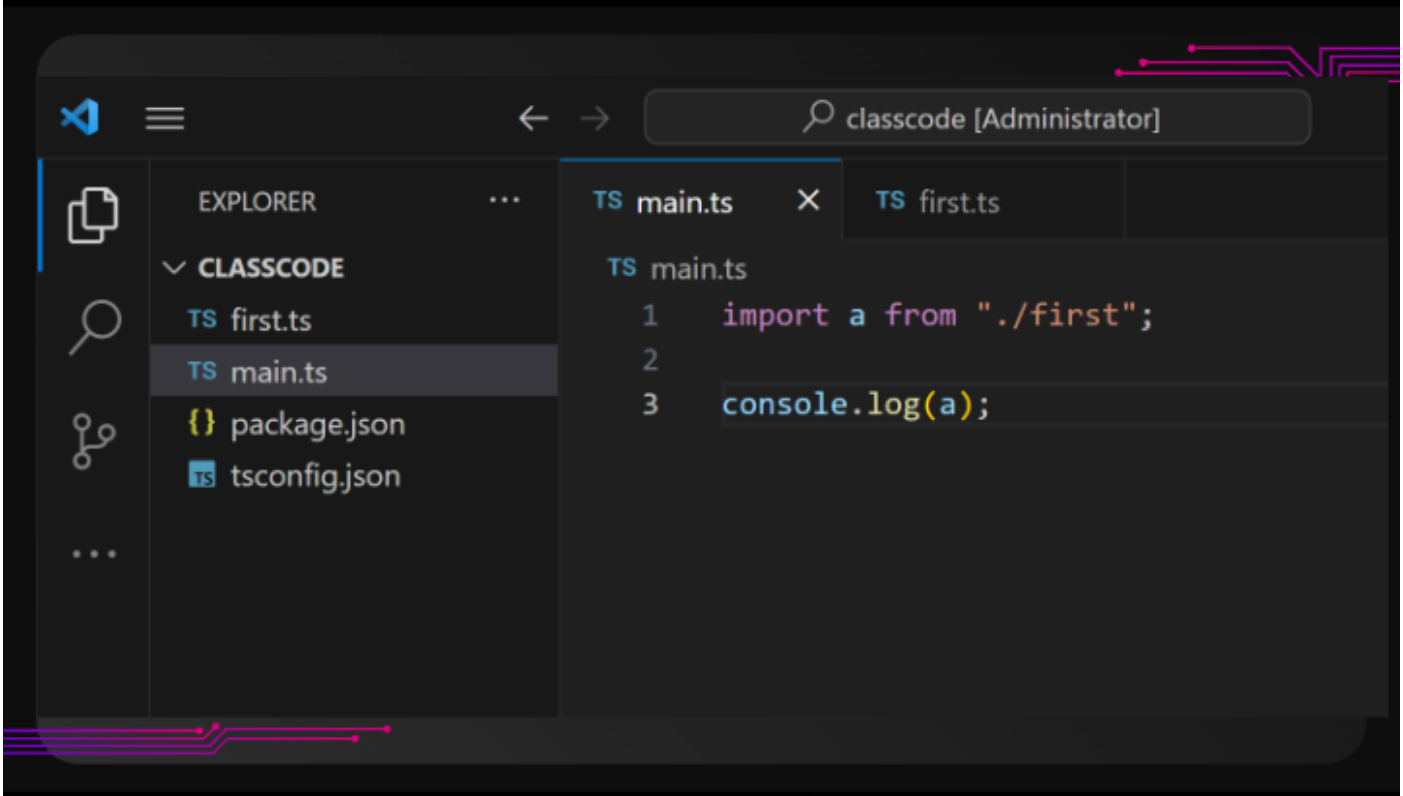
Fully-Modular Buildings

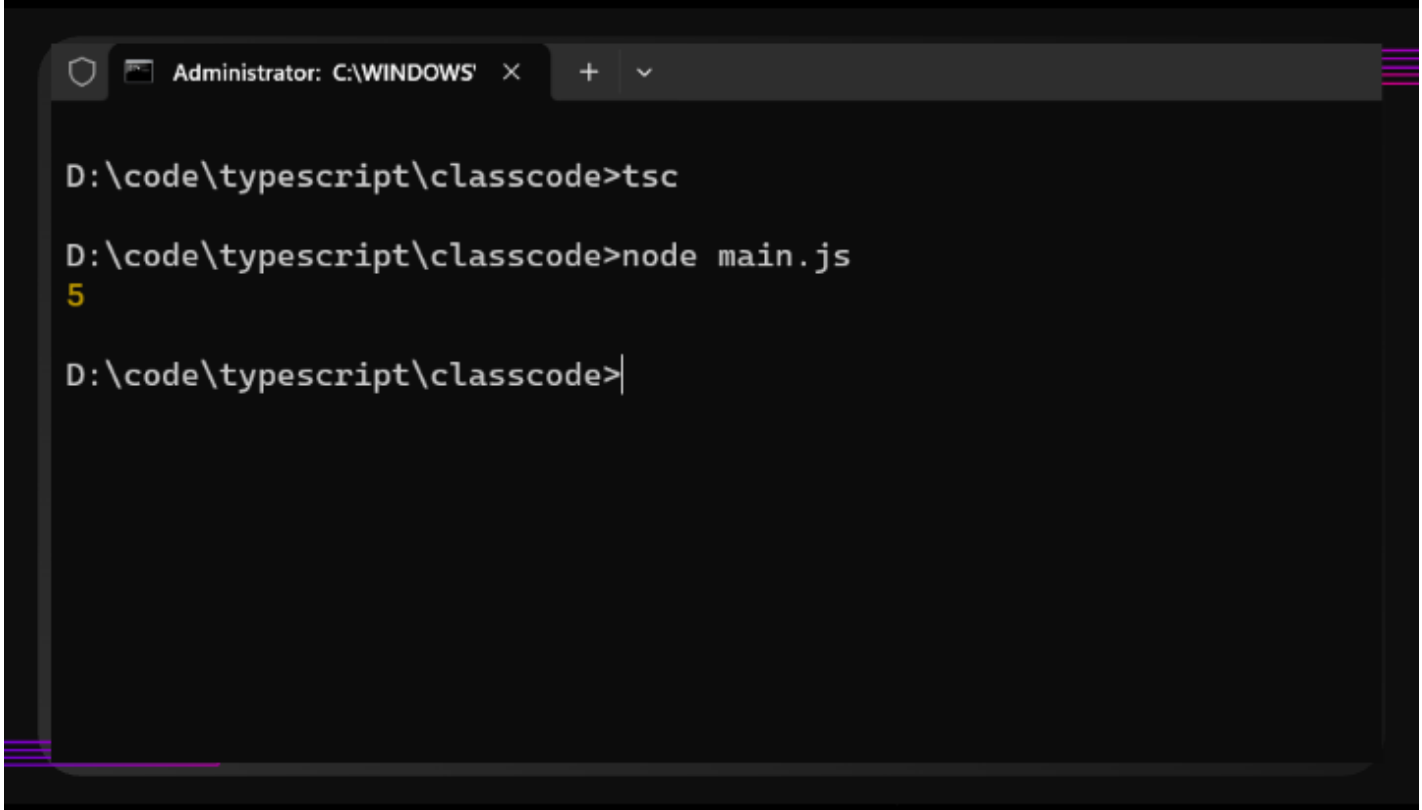


Create a New Project




classcode [Administrator]
EXPLORER
CLASSC...
first.ts
main.ts
package.json
tsconfig.json
main.ts
first.ts
first.ts > default
1 let a = 5;
2
3 export default a;


classcode [Administrator]
EXPLORER
CLASSCODE
first.ts
main.ts
package.json
tsconfig.json
main.ts
first.ts
main.ts
1 import a from "./first";
2
3 console.log(a);


Administrator: C:\WINDOWS'
D:\code\typescript\classcode>tsc
D:\code\typescript\classcode>node main.js
5
D:\code\typescript\classcode>

When we transpile this program it runs correctly.

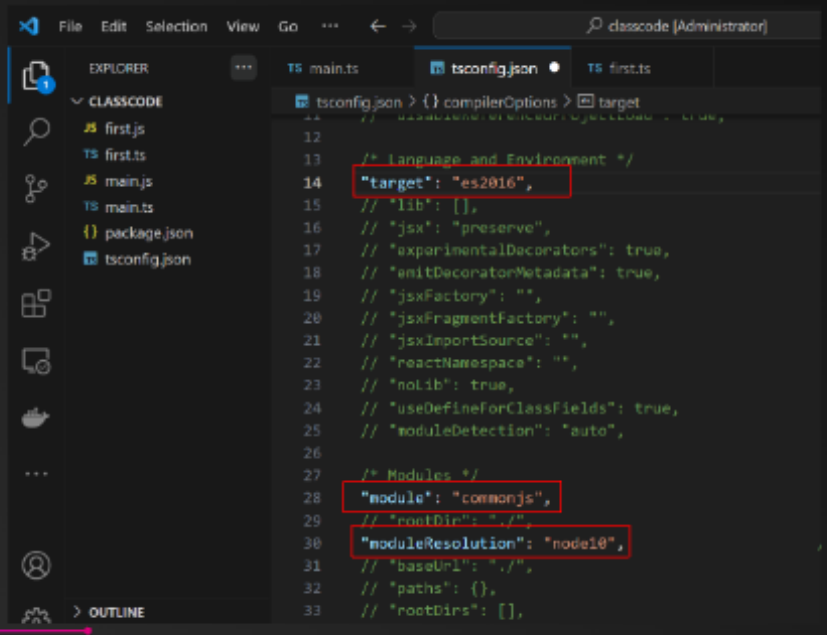
However, note that the transpiled JavaScript code does not use the ES Module syntax but rather the old commonjs syntax.

Modules in Learn-TypeScript Repo

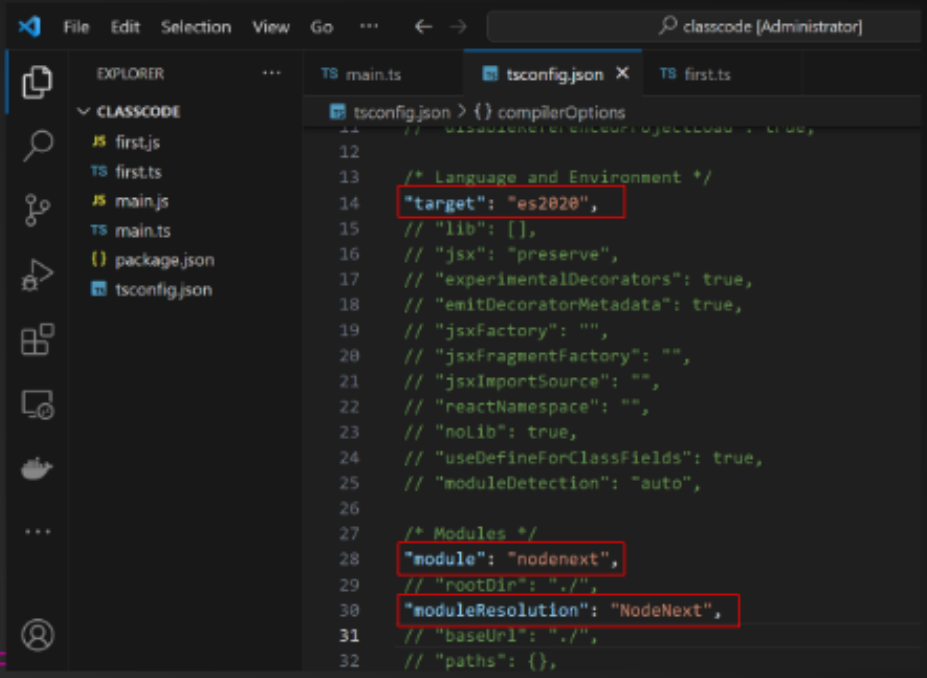
Native ECMAScript Modules

Using ES modules in TypeScript.

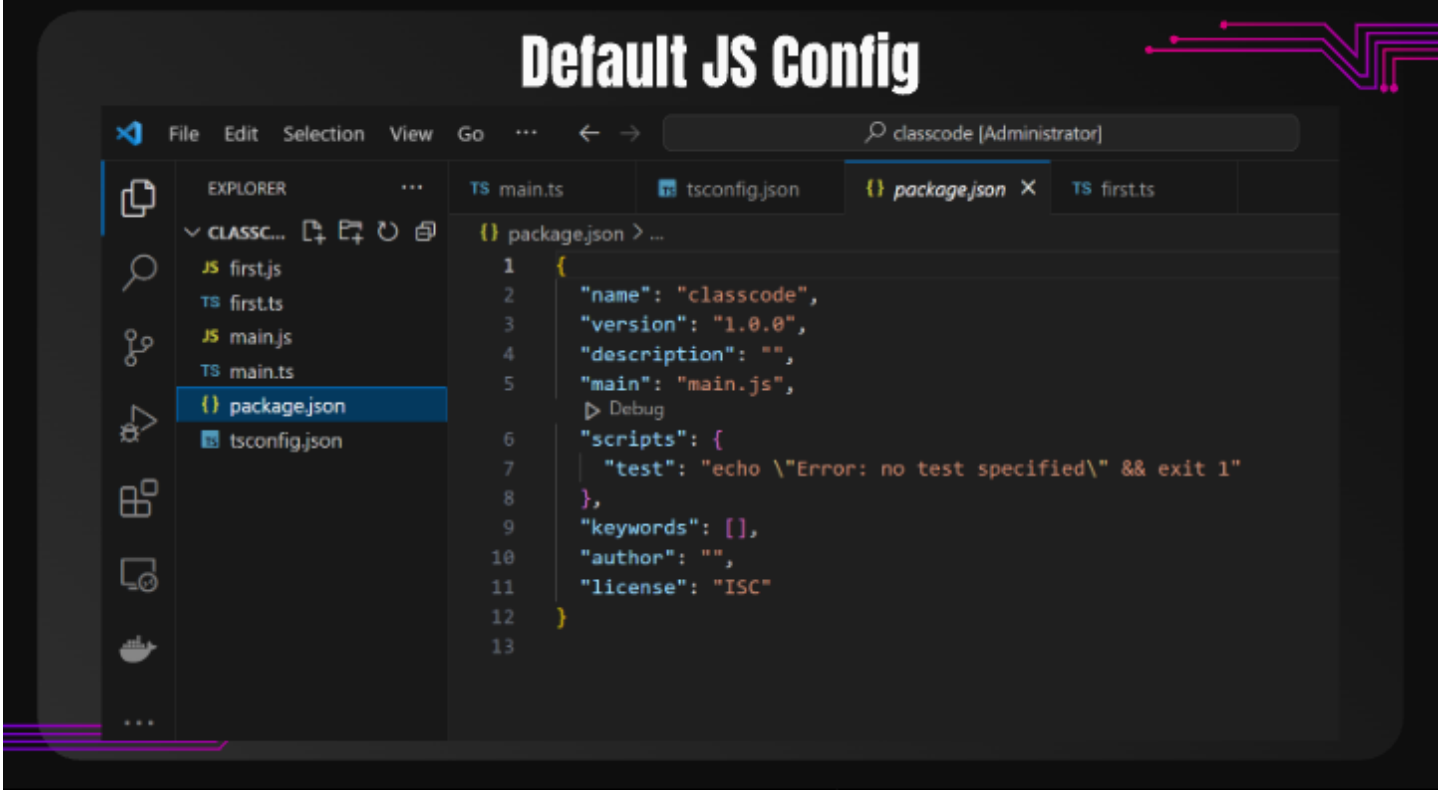
Default TS Config



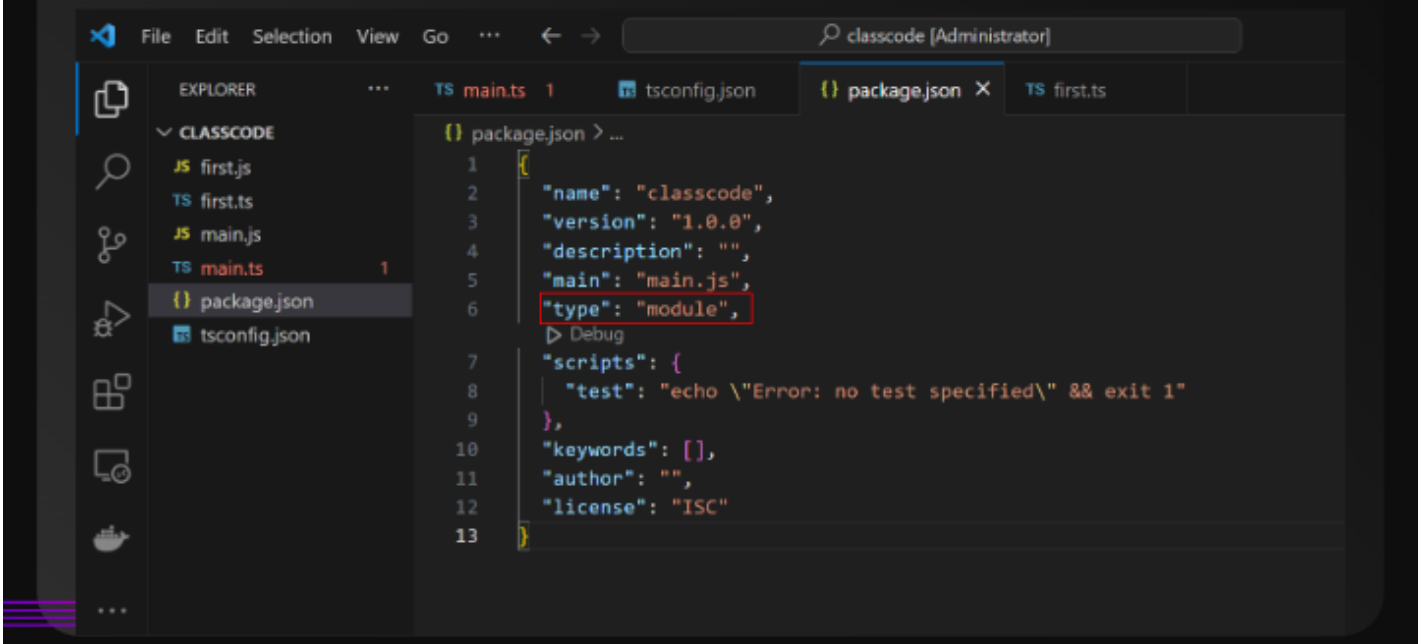
The program we will have make some changes



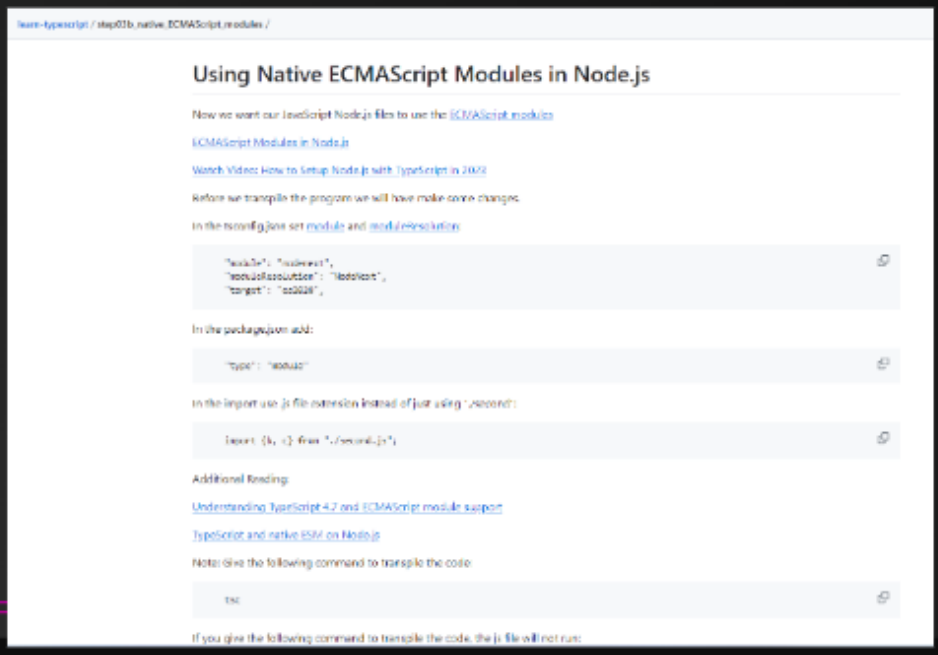
Default JS Config



The program we will have make some changes



Using Native ECMAScript Modules in Node.js



learn-typescript / step03b_native_ECMAScript_modules /

Using Native ECMAScript Modules in Node.js

Now we want our JavaScript Node.js files to use the ECMAScript modules

ECMAScript Modules in Node.js

Watch Videos How to Setup Node.js with TypeScript in 2022

Before we transpile the program we will have make some changes.

In the tsconfig.json set module and moduleResolution

```
"module": "nodenext",
"moduleResolution": "nodenext",
"target": "es2020",
```

In the package.json add:

```
"type": "module"
```

In the import use .js file extension instead of just using "./second":

```
import {h, c} from "./second.js";
```

Additional Reading:

Understanding TypeScript 4.7 and ECMAScript module support

TypeScript and native ESM on Node.js

Note: Give the following command to transpile the code:

```
tsc
```

If you give the following command to transpile the code, the .js file will not run:

Importing Third-party Modules

Example with importing the inquirer module.

NPM - Node Package Manager

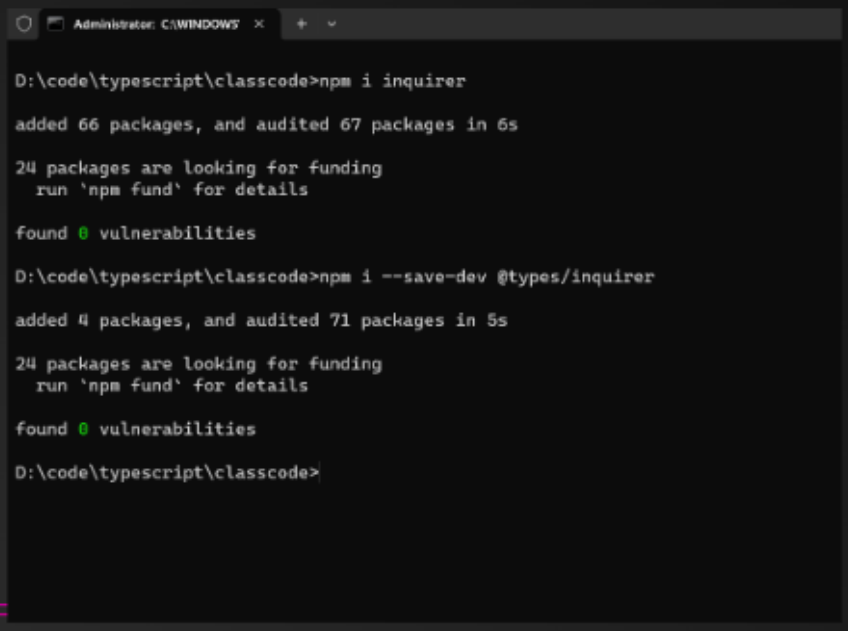
Install Dependencies

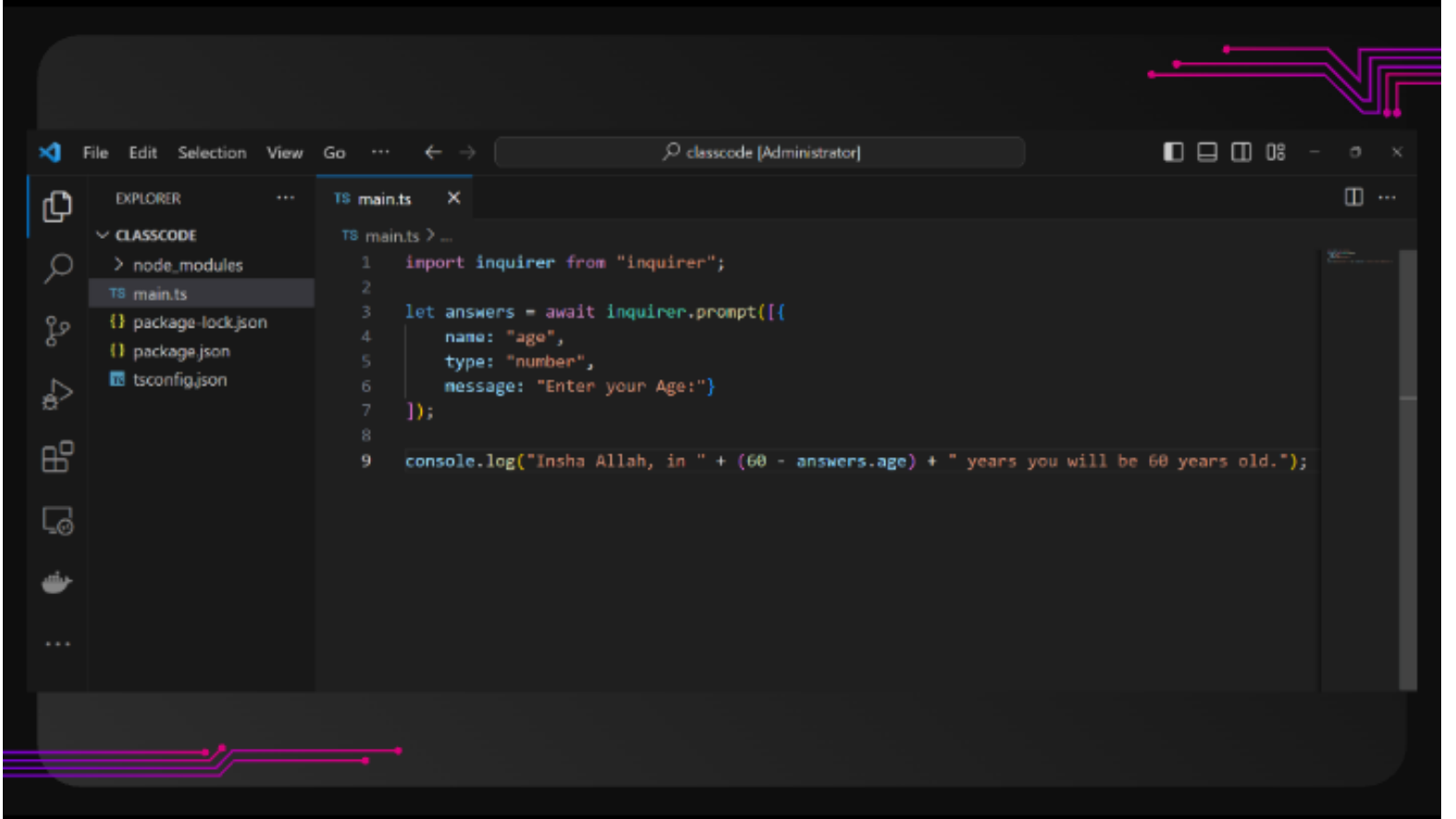
Using Native ECMAScript Modules in Node.js
Using Inquirer Package

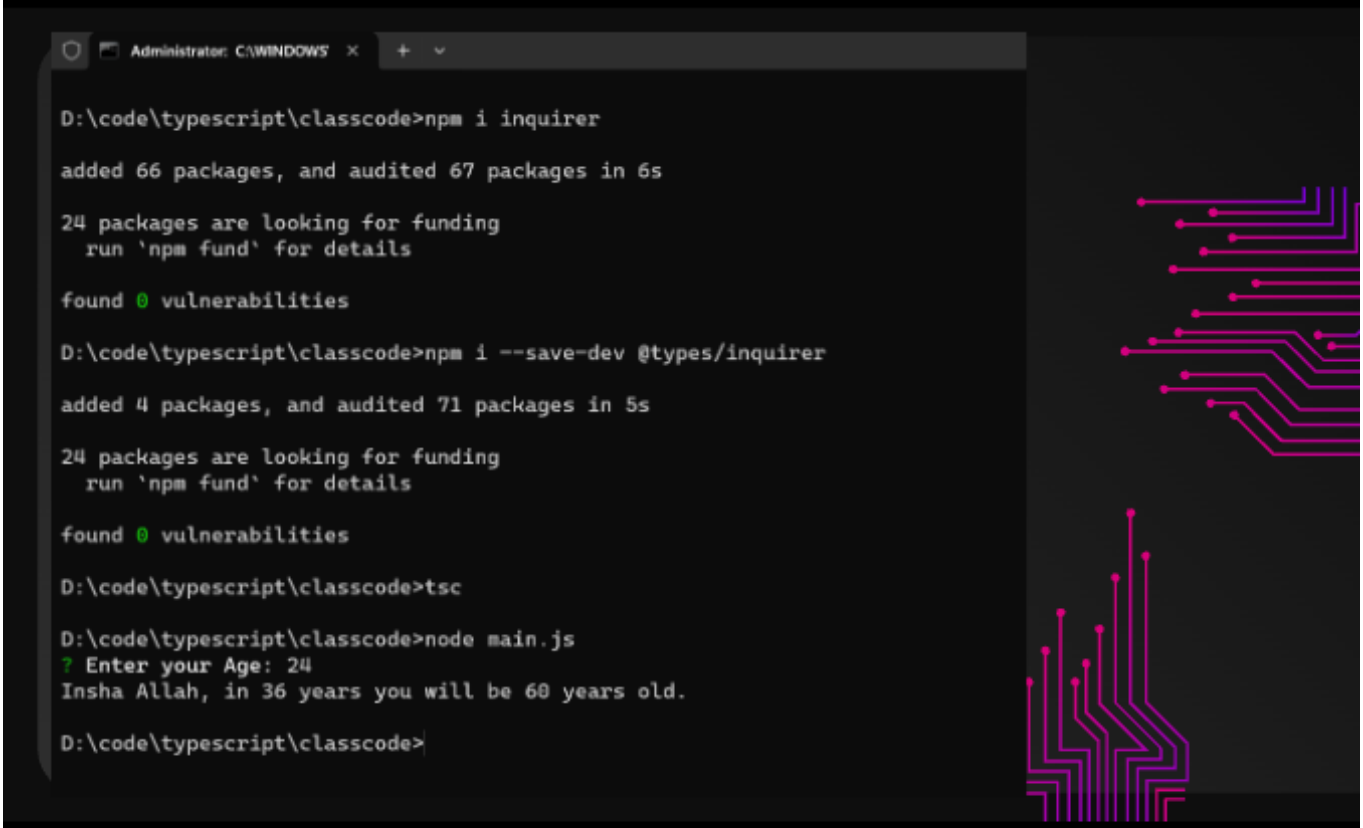
npm i inquirer

npm i --save-dev @types/inquirer

Install Dependencies

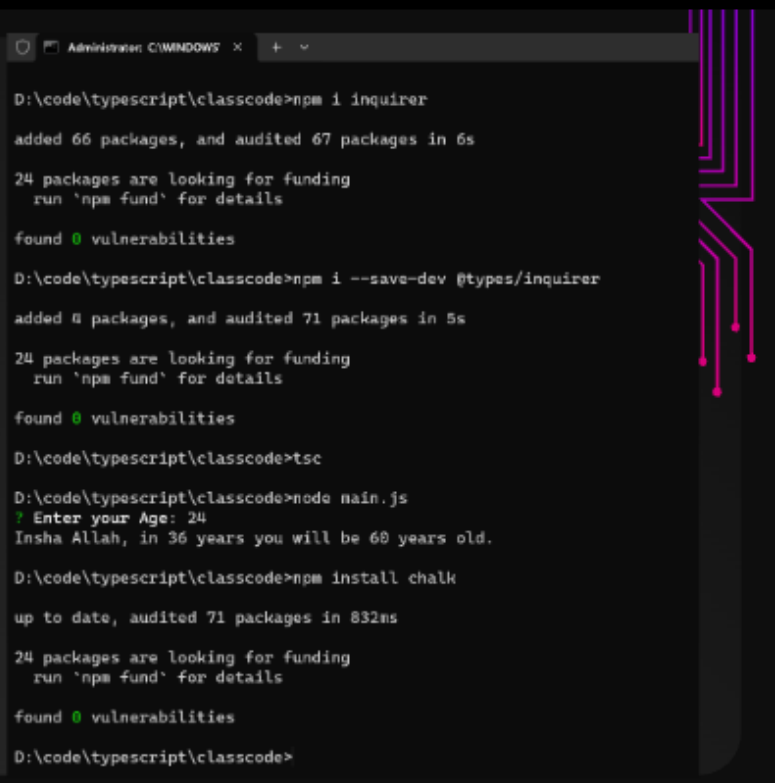


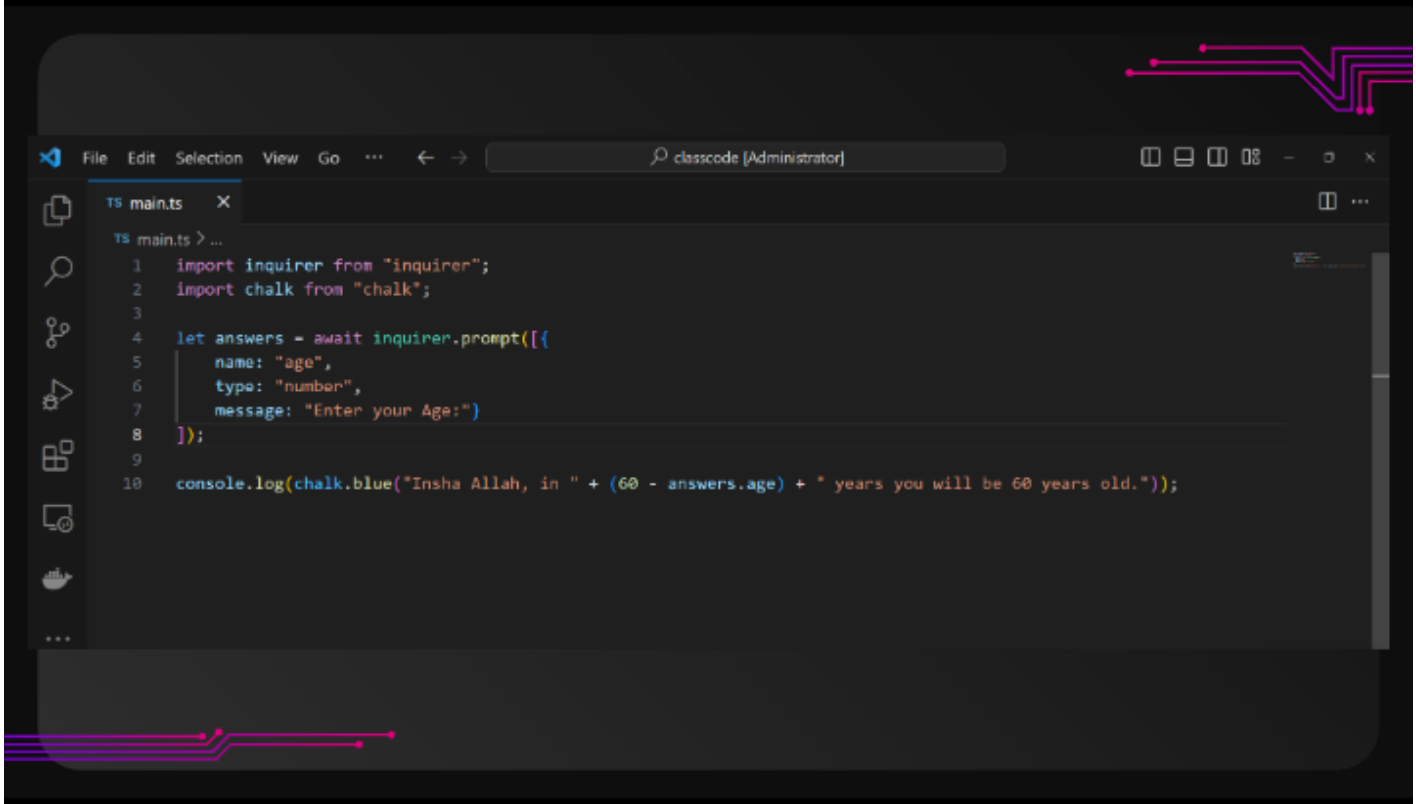

File Edit Selection View Go
classcode [Administrator]
EXPLORER
CLASSCODE
node_modules
main.ts
package-lock.json
package.json
tsconfig.json
main.ts
import inquirer from "inquirer";
let answers = await inquirer.prompt([{
name: "age",
type: "number",
message: "Enter your Age:"}
]);
console.log("Insha Allah, in " + (60 - answers.age) + " years you will be 60 years old.");

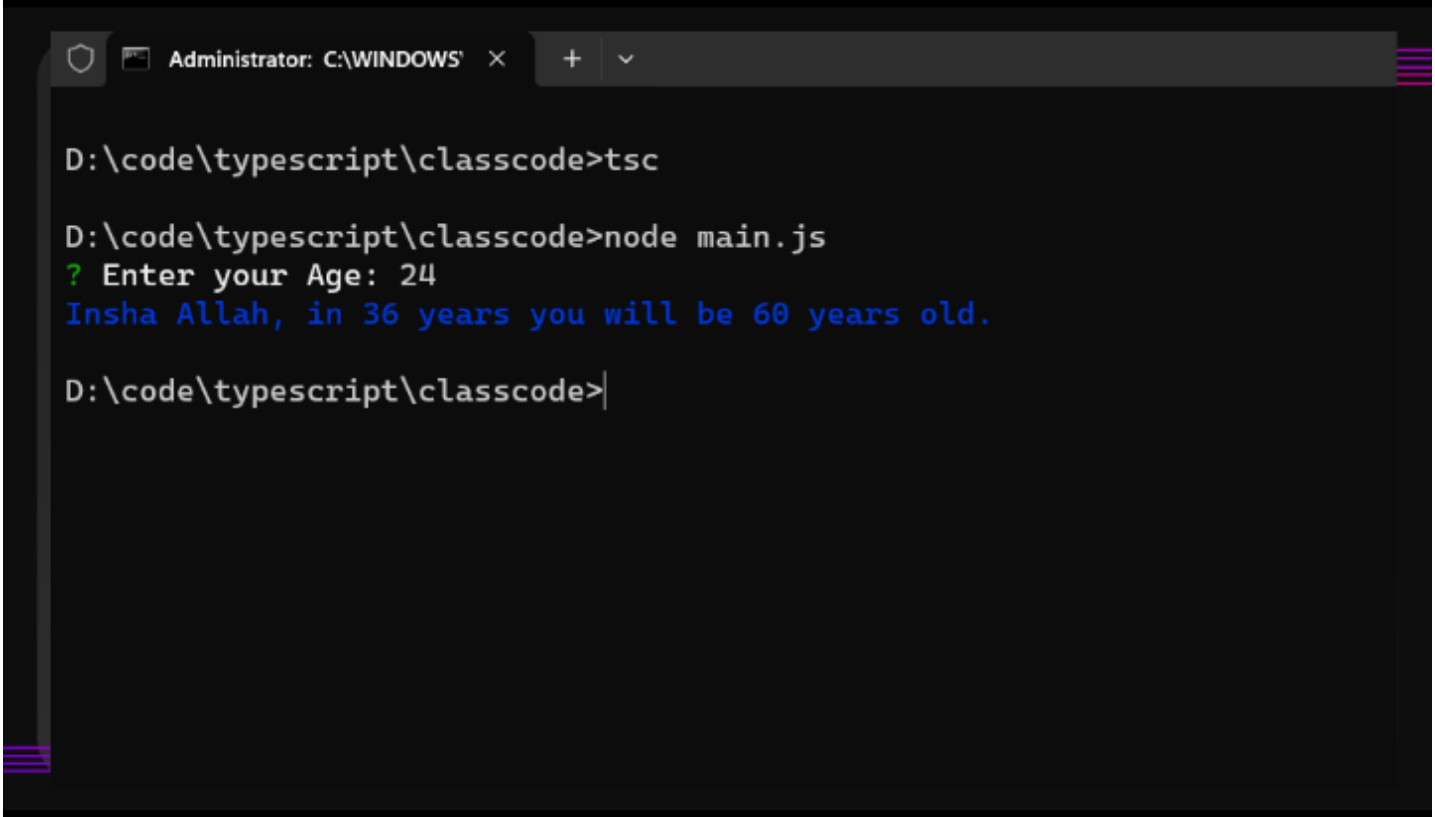

Administrator: C:\WINDOWS
D:\code\typescript\classcode>npm i inquirer
added 66 packages, and audited 67 packages in 6s
24 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
D:\code\typescript\classcode>npm i --save-dev @types/inquirer
added 4 packages, and audited 71 packages in 5s
24 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
D:\code\typescript\classcode>tsc
D:\code\typescript\classcode>node main.js
? Enter your Age: 24
Insha Allah, in 36 years you will be 60 years old.
D:\code\typescript\classcode>

Install Chalk Dependency

Practical example of third-party module usage.




File Edit Selection View Go
classcode [Administrator]
main.ts
main.ts > ...
import inquirer from "inquirer";
import chalk from "chalk";
let answers = await inquirer.prompt([{
name: "age",
type: "number",
message: "Enter your Age:"}
]);
console.log(chalk.blue("Insha Allah, in " + (60 - answers.age) + " years you will be 60 years old."));


Administrator: C:\WINDOWS\
D:\code\typescript\classcode>tsc
D:\code\typescript\classcode>node main.js
? Enter your Age: 24
Insha Allah, in 36 years you will be 60 years old.
D:\code\typescript\classcode>

Step 04
Advanced Types

Unions and Literals

- Understanding union types and literal types.

Objects and Typing

Creating and Typing Objects

- Basics of object types in TypeScript

Aliased Types

- Using type aliases for objects.

Structural Typing and Object Literals

- How TypeScript uses structural typing for objects.

Nested Objects

- Working with objects within objects.

Intersection Types

- Combining types using intersection types.

Special Types

- Introduction to any, unknown, and never types.

JSON Objects

Understanding JSON format, creating, and using JSON objects in TypeScript.

Step 03 Type Casting and Enums

Explicit Casting

- Techniques for type casting in TypeScript.

Enums

- Introduction to Enums: Using enums for a set of named constants.
- Const Enums: Understanding the benefits of const enums.

Step 04: Collections and Functions

Arrays

- Basics of arrays in TypeScript, typing arrays.

Functions

Function Basics

- Writing typed functions.

Optional, Default, and Rest Parameters:

- Enhancing function flexibility

Async Functions

- Introduction to asynchronous programming with async/await.

Function Overloads

- Using function overloads for multiple type signatures

Tuples

Object-Oriented Programming in TypeScript

Introduction to OOP

- Basics of classes and objects.

Classes and Inheritance

Defining Classes

- Creating classes, properties, and methods

Inheritance

- Extending classes to create subclasses

Abstract Classes

- Using abstract classes to define base classes

Class Modifiers

Private and Protected Modifiers

- Controlling access to class members.

Accessors

- Using getters and setters.

Static Properties

- Understanding static members of a class.

Interfaces

Introduction to Interfaces

- Defining contracts for objects.

Optional Properties and Index Signatures

- Enhancing interface flexibility.

Extending Interfaces

- Inheritance among interfaces

Implementing Interfaces in Classes

- Using interfaces to define class structures.

Advanced OOP Features

Decorators

- Introduction to decorators for class modification.

Generics

- Using generics for type-safe data structures and functions.

Class Expressions

- Defining classes in expressions.

Generators

- Understanding generators and iterators.

Step 06
Applying OOP Concepts

Practical OOP Applications

- Applying OOP concepts in real-world TypeScript projects.

TypeScript Project Structure

- Organizing and structuring TypeScript projects for maintainability.

Topics

- Variables
- Variables and Data Types
- Type Annotations on Variables
- Comments
- Statements
- Let, Var, Const
- Primitive data types
- Template Literals
- Analyzing and modifying data types
- Operators
  Operators and Expressions

Topics

- If, Else, Else If Statements
- If Statement Nested
- Functions
- Return Type Annotations
- Arrow Functions
- Array
- For Loops
- Nested For Loops
- Control Structures (if, else, switch, loops)
  Functions and Scope

Topics

- Understanding Classes and Objects
- Properties and Methods
- Encapsulation: Keeping data safe

Thanks!

https://linktr.ee/ameenalam

fb.com/SheikhAmeenAlam

linkedin.com/in/ameen-alam

instagram.com/sheikhameenalam

yt.com/ameenalamofficial

Presented by Sir Ameen Alam