

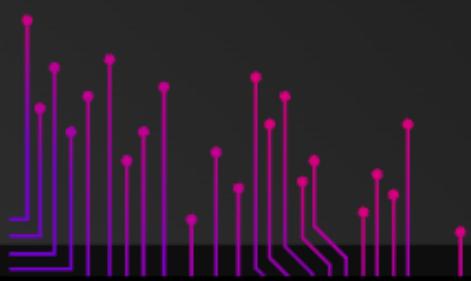
TypeScript Mastery

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/1eepDn27eQ8DbRJy7LHTkrIwYx8hQwuooaDKGS_x1oyc

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/1eepDn27eQ8DbRJy7LHTkrlwYx8hQwu0oaDKGS_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.





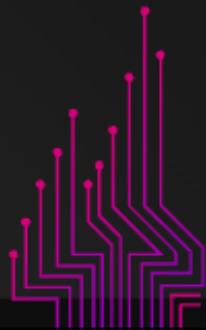
Variable in Programming

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"

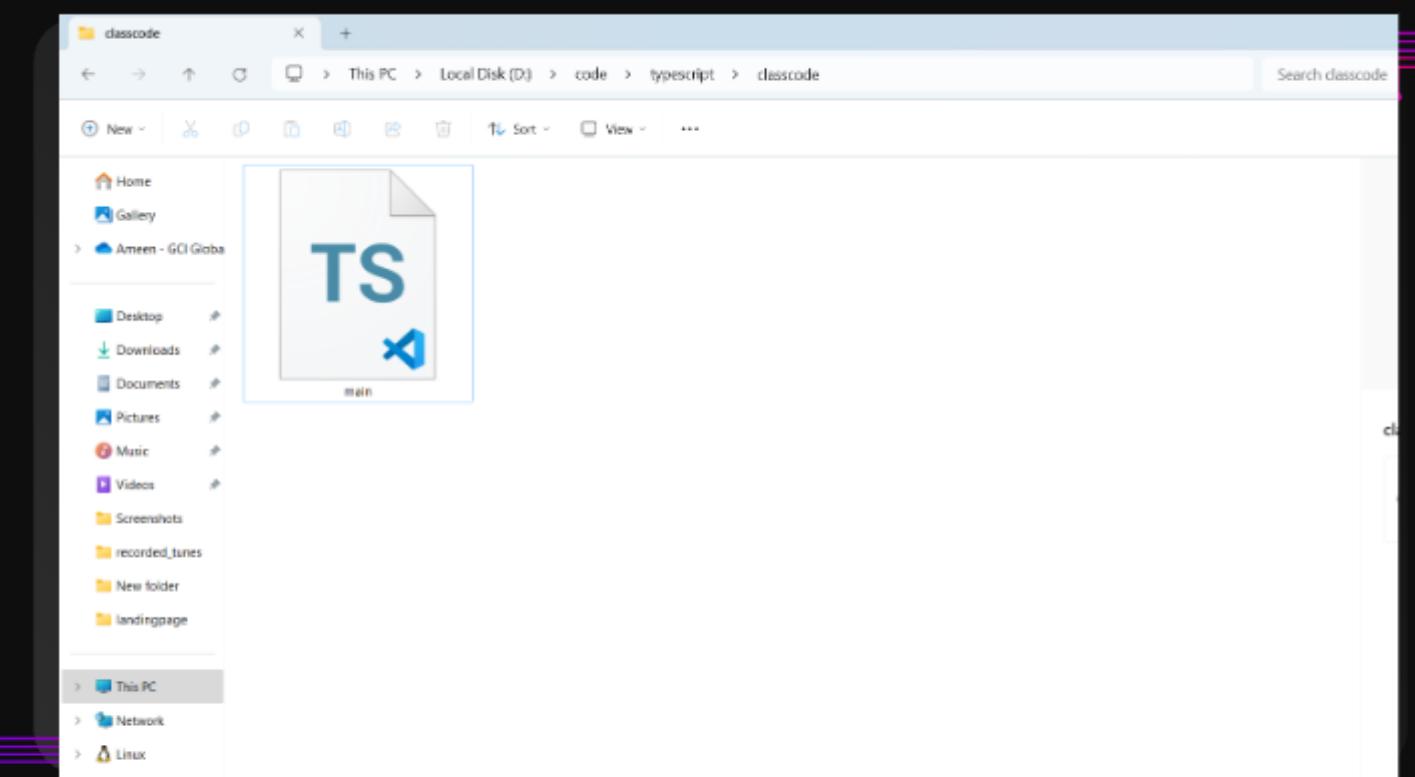
```
let favoriteColor = "blue";
```



The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The title bar displays the path "classcode [Administrator]". The Explorer sidebar on the left shows a folder named "CLASSC..." containing a file "main.ts". The main editor area shows the following TypeScript code:

```
1 let favoriteColor = "blue";
2 console.log(favoriteColor)
```

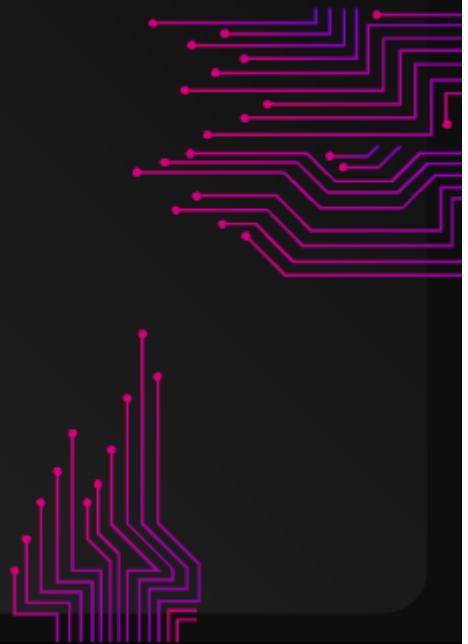
The status bar at the bottom provides information about the current file: "Ln 2, Col 27" and "Spaces: 4". It also shows the encoding as "UTF-8", the line separator as "CRLF", and the language as "TypeScript".



```
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>
```

Changes the value of favoriteColor
to "green"

```
let favoriteColor = "blue";  
favoriteColor = "green";
```



The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The title bar displays the path "classcode [Administrator]". The left sidebar has icons for File, Explorer, Search, and others. The "EXPLORER" section shows a folder named "CLASSCODE" containing a file "main.ts". The main editor area shows the following TypeScript code:

```
TS main.ts > ...
1 let favoriteColor = "blue";
2 console.log(favoriteColor)
3 favoriteColor = "green";
4 console.log(favoriteColor);
5
```

The status bar at the bottom shows "Ln 5, Col 1" and other settings like "Spaces: 4", "UTF-8", "CRLF", and "TypeScript". There are also icons for weather (24°C), battery, and network.

```
Administrator: C:\WINDOWS > + - X
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".

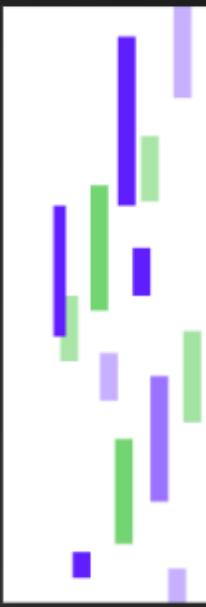


case sensitive

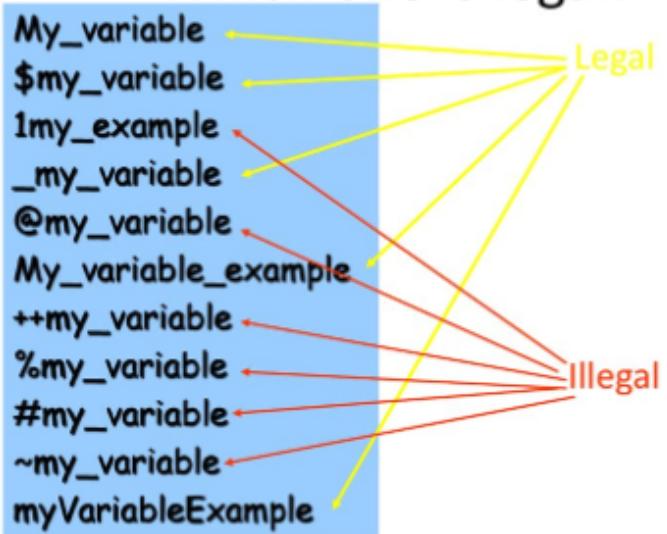


Explained

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.

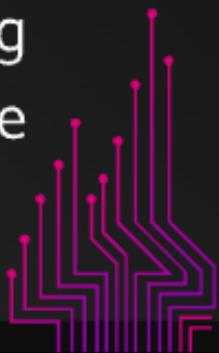


TypeScript assigns types to variables to ensure they are used correctly throughout your code. This helps prevent errors, such as mixing up numbers with text or trying to perform operations on incompatible data types.



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.

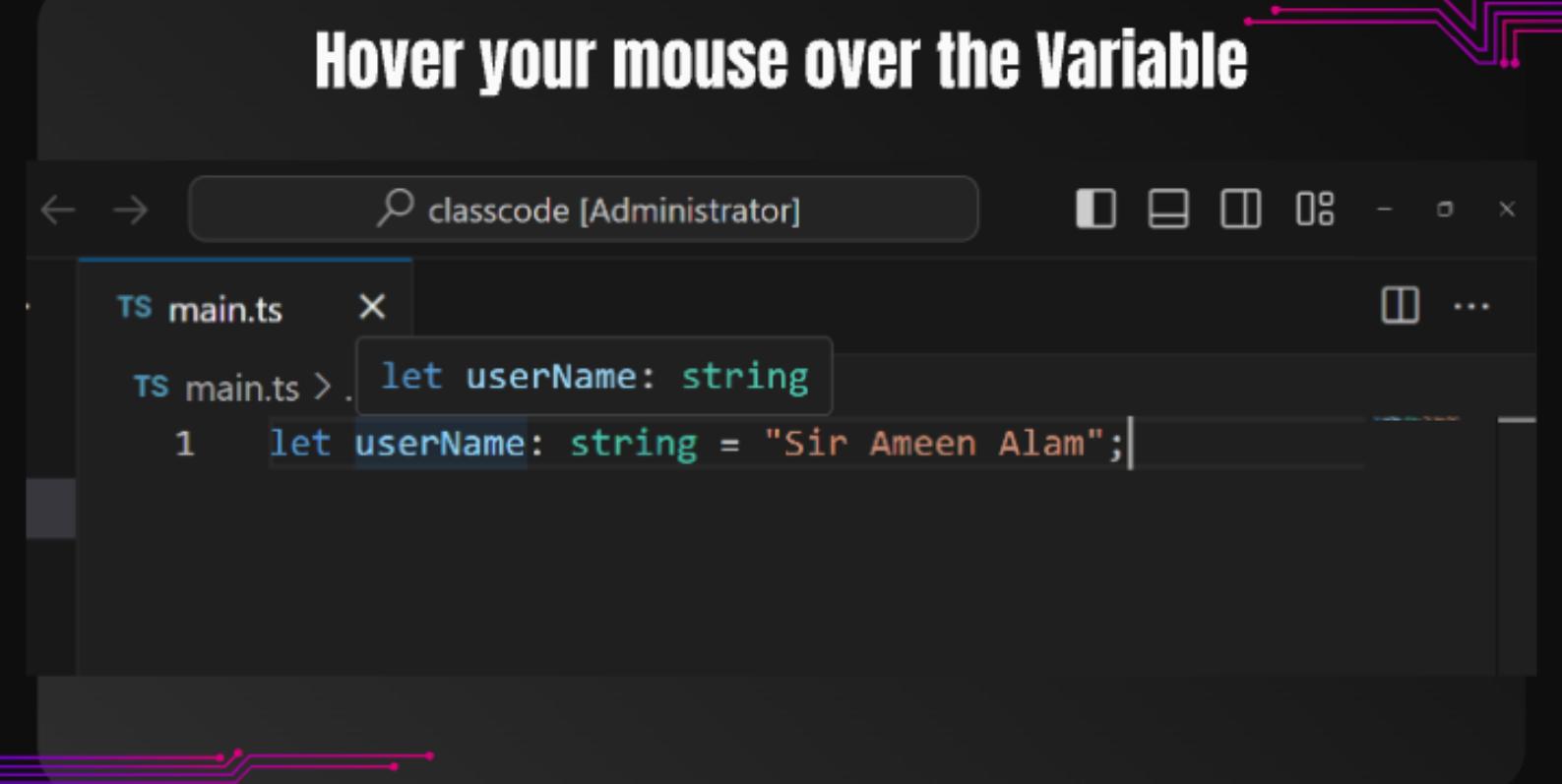


The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The title bar displays "classcode [Administrator]". The left sidebar has icons for Explorer, Search, Problems, and Outline/Timeline. The Explorer view shows a folder named "CLASSCODE" containing files "main.js" and "main.ts", with "main.ts" selected. The main editor area shows the following TypeScript code:

```
1 let userName: string = "Sir Ameen Alam";
```

The status bar at the bottom shows "Ln 1, Col 41" and "Spaces: 4". The taskbar at the bottom includes icons for Weather (22°C), Live Share, and various system icons like battery, signal, and date/time.

Hover your mouse over the Variable



A screenshot of a dark-themed code editor window. The title bar shows the path "classcode [Administrator]". The main area displays a file named "main.ts" with the following content:

```
TS main.ts
TS main.ts > let userName: string
1 let userName: string = "Sir Ameen Alam";
```

The word "userName" in the second line is highlighted with a light blue box, indicating it is the variable being hovered over. A tooltip or status bar at the bottom of the editor also displays "let userName: string". The interface includes standard window controls (minimize, maximize, close) and a tab bar.

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

A screenshot of a code editor window titled "classcode [Administrator]". The status bar shows "TS main.ts 1 X". The code editor displays the following TypeScript code:

```
TS main.ts > ...
1 let userName: string = "Sir Ameen Alam";
2 userName = 25
```

The line "2 userName = 25" contains a red underline under "userName", indicating a TypeScript error: "Type 'number' is not assignable to type 'string'." The code editor interface includes a search bar, a toolbar with icons, and a file tab bar.

Hover your mouse over the Variable

A screenshot of a code editor window titled "classcode [Administrator]". The status bar shows "Type 'number' is not assignable to type 'string'. ts(2322)". The code editor displays the following TypeScript code:

```
TS 1 let userName: string
TS 2 userName = 25
```

Line 1 has a tooltip "View Problem (Alt+F8) No quick fixes available". The variable "userName" in line 2 is underlined with a red wavy line, indicating a TypeScript error.

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] TS main.ts X 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

The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with icons for file operations, search, and other tools. The main area displays a file named 'main.ts' with the following content:

```
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

The screenshot shows a code editor interface with a dark theme. On the left, there is a vertical toolbar with icons for file operations (New, Open, Save, etc.), search, and other functions. The main area displays a file named "main.ts". The code within the file is as follows:

```
TS main.ts  X
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] TS main.ts ●  
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](https://www.freecodecamp.org/news/javascript-let-and-const-hoisting)



Errors

Syntax Error

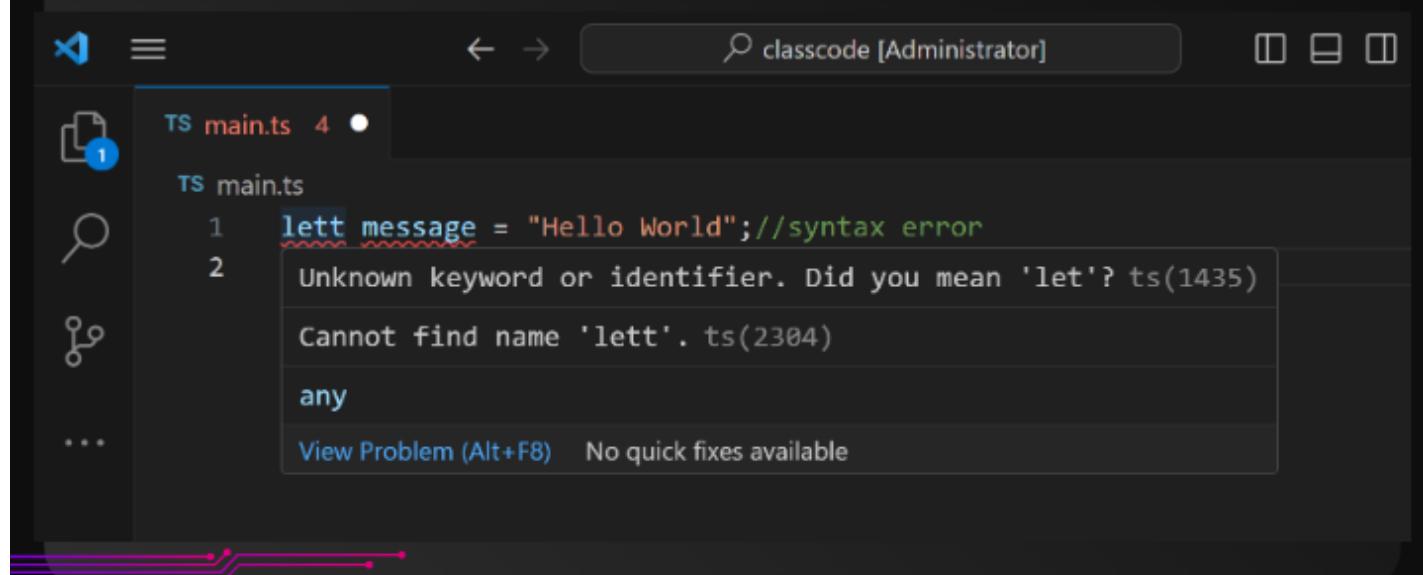
Identifying and resolving syntax errors in TypeScript code.

The screenshot shows a dark-themed code editor window. At the top, there are icons for file operations (New, Open, Save, etc.) and a search bar containing the text "classcode [Administrator]". Below the toolbar, the file path "TS main.ts" is displayed, followed by the number "4" and a circular icon with a dot. The main editor area contains the following TypeScript code:

```
TS main.ts 4 ●
TS main.ts
1  lett message = "Hello World"; //syntax error
2  console.log(message);
```

The word "lett" in the first line is underlined with a red squiggly line, indicating a syntax error. The code editor has decorative purple wavy lines at the bottom.

Hover your mouse over the Keyword



A screenshot of the Visual Studio Code interface. The title bar shows the file path "classcode [Administrator]". The left sidebar has icons for file, search, and other settings. The main editor window displays a TypeScript file named "main.ts" with the following code:

```
1  lett message = "Hello World"; //syntax error
2
```

The word "lett" is underlined with a red squiggly line, indicating a syntax error. A tooltip is open over the first occurrence of "lett" at line 1, showing the following information:

- Unknown keyword or identifier. Did you mean 'let'? ts(1435)
- Cannot find name 'lett'. ts(2304)
- any

At the bottom of the tooltip, there are links for "View Problem (Alt+F8)" and "No quick fixes available".

Type Error

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

Hover your mouse over the method

The screenshot shows a dark-themed code editor window. At the top, there's a navigation bar with icons for back, forward, search, and zoom, followed by a search bar containing "classcode [Administrator]" and some status indicators. The main area displays a file named "main.ts" with the following content:

```
TS main.ts 1 ●
TS main.ts > ...
1 let message = "Hello World";
2 console.loger(message);
3 |
```

A tooltip has appeared over the misspelled "loger" method, displaying the following error message:

Property 'loger' does not exist on type 'Console'. Did you mean 'log'? ts(2551)

Below the tooltip, there are two suggestions:

- lib.dom.d.ts(26638, 5): 'log' is declared here.
- any

At the bottom of the editor window, there are links for "View Problem (Alt+F8)" and "Quick Fix... (Ctrl+.)".

Assignability Error

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

```
TS main.ts 1 X
TS main.ts > ...
1 let message = "Hello World";
2 messsage = 6;
3 console.log(message);
```

Hover your mouse over the message

A screenshot of a code editor interface, likely Visual Studio Code, displaying a TypeScript file named `main.ts`. The code contains a simple logging statement:let message: string
message = 6;
console.log(message);The word `message` is underlined with a red squiggly line, indicating a type error. A tooltip has appeared over the error, containing the following text:

Type 'number' is not assignable to type 'string'. ts(2322)

Below the tooltip, the code editor shows the line numbers 1, 2, and 3 corresponding to the code lines above.

Step 03

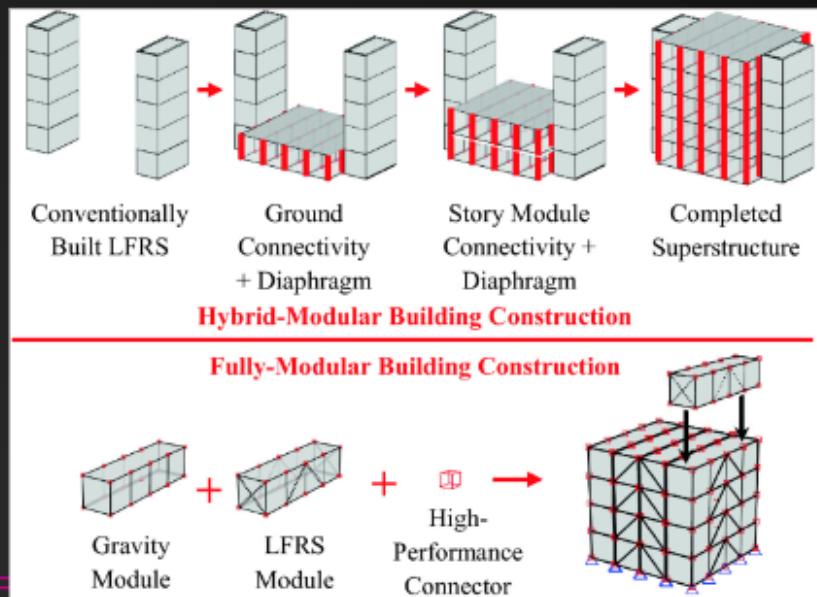
Modules

Basics of modules, exporting, and
importing modules.

Modular Architecture



Fully-Modular Buildings



Create a New Project

```
Administrator: C:\WINDOWS > tsc --init
Microsoft Windows [Version 10.0.22621.3155]
(c) Microsoft Corporation. All rights reserved.

D:\code\typescript\classcode>tsc --init
Created a new tsconfig.json with:
{
  "target": "es2016",
  "module": "commonjs",
  "strict": true,
  "esModuleInterop": true,
  "skipLibCheck": true,
  "forceConsistentCasingInFileNames": true
}

You can learn more at https://aka.ms/tsconfig
D:\code\typescript\classcode>npm init -y
Wrote to D:\code\typescript\classcode\package.json:
{
  "name": "classcode",
  "version": "1.0.0",
  "description": "",
  "main": "main.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

The screenshot shows a dark-themed code editor interface, likely Visual Studio Code. The top bar includes a search bar with the text "classcode [Administrator]" and several icons. The left sidebar has sections for "EXPLORER", "... (three dots)", and "CLASSC...". The "EXPLORER" section lists files: "first.ts" (selected), "main.ts", "package.json", and "tsconfig.json". The main editor area shows the content of "first.ts":

```
TS first.ts > [✖] default
1 let a = 5;
2
3 export default a;
```

The screenshot shows the Visual Studio Code (VS Code) interface on a dark-themed background. At the top, there is a search bar with the text "classcode [Administrator]" and a magnifying glass icon. Below the search bar, the title bar displays the project name "classcode". On the far left, the Explorer sidebar is visible, showing a file tree with the following structure:

- EXPLORER
- ...
- CLASSCODE
 - first.ts
 - main.ts**
 - package.json
 - tsconfig.json
- ...

The "main.ts" file is currently selected in the Explorer sidebar and is also the active editor tab in the main workspace. The code editor window shows the following TypeScript code:

```
TS main.ts
1 import a from "./first";
2
3 console.log(a);
```

```
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

learn-typescript / step03a_modules / readme.md 

 ziaukhan add step 03

967

[Preview](#) [Code](#) | [Blame](#) 18 lines (6 loc) • 358 Bytes

Modules in TypeScript

[Modules](#)

[ECMAScript Modules in Node.js](#)

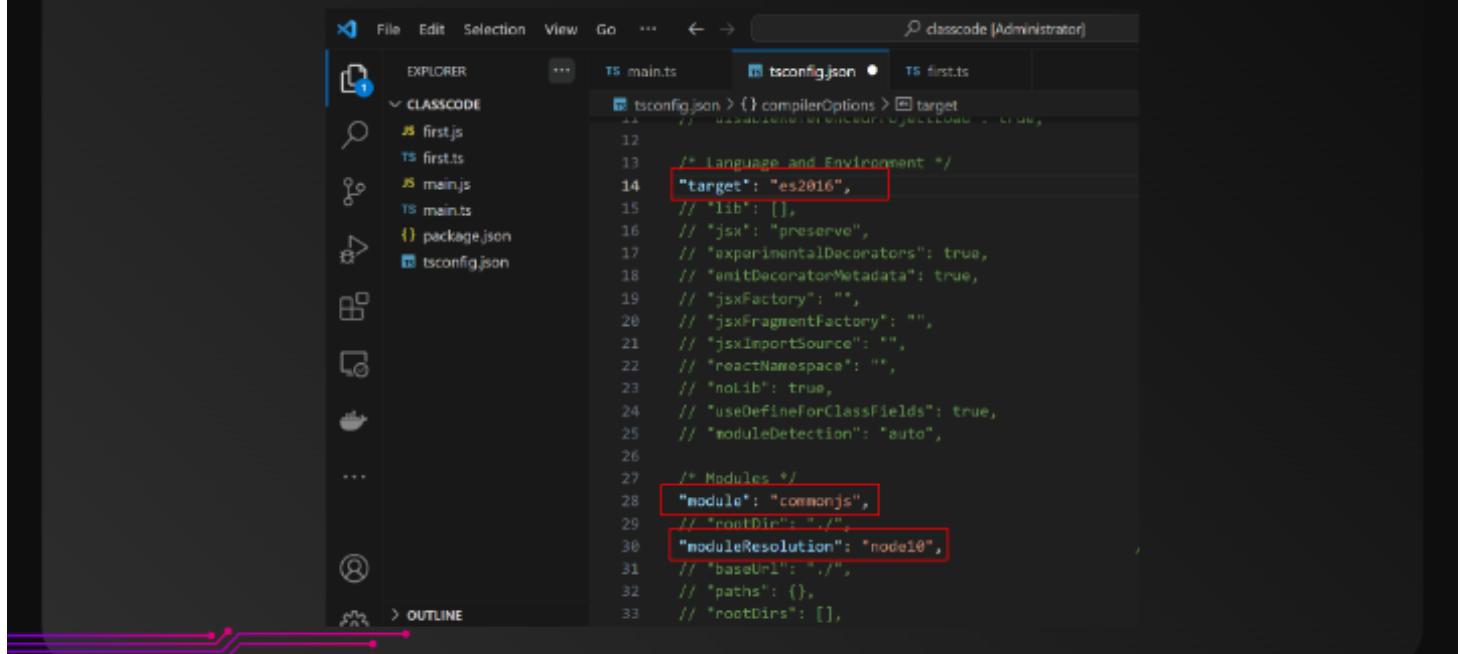
When we transpile this program it runs correctly.

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

Native ECMAScript Modules

Using ES modules in TypeScript.

Default TS Config



A screenshot of the Visual Studio Code interface. The title bar shows "dsscode (Administrator)". The Explorer sidebar on the left lists files: first.js, first.ts, main.js, main.ts, package.json, and tsconfig.json. The main editor area displays the contents of tsconfig.json. Several sections of the JSON code are highlighted with red boxes:

```
tsconfig.json > {} compilerOptions > target
12 // "allowJs": true,
13 /* Language and Environment */
14 "target": "es2016",
15 // "lib": [],
16 // "jsx": "preserve",
17 // "experimentalDecorators": true,
18 // "emitDecoratorMetadata": true,
19 // "jsxFactory": "",
20 // "jsxFragmentFactory": "",
21 // "jsxImportSource": "",
22 // "reactNamespace": "",
23 // "noLib": true,
24 // "useDefineForClassFields": true,
25 // "moduleDetection": "auto",
26
27 /* Modules */
28 "module": "commonjs",
29 // "rootDir": "./",
30 "moduleResolution": "node10",
31 // "baseUrl": "./",
32 // "paths": {},
33 // "rootDirs": [],
```

The program we will have make some changes

```
File Edit Selection View Go ... ⏪ ⏩ classcode [Administrator]

EXPLORER CLASSCODE
first.js
first.ts
main.js
main.ts
package.json
tsconfig.json

tsconfig.json > {} compilerOptions
  11 // Visual Studio Code Object Model - Editor,
  12
  13 /* Language and Environment */
  14 "target": "es2020",
  15 // "lib": [],
  16 // "jsx": "preserve",
  17 // "experimentalDecorators": true,
  18 // "emitDecoratorMetadata": true,
  19 // "jsxFactory": "",
  20 // "jsxFragmentFactory": "",
  21 // "jsxImportSource": "",
  22 // "reactNamespace": "",
  23 // "noLib": true,
  24 // "useDefineForClassFields": true,
  25 // "moduleDetection": "auto",
  26
  27 /* Modules */
  28 "module": "nodenext",
  29 // "rootDir": "./",
  30 "moduleResolution": "NodeNext",
  31 // "baseUrl": "./",
  32 // "paths": {}
```

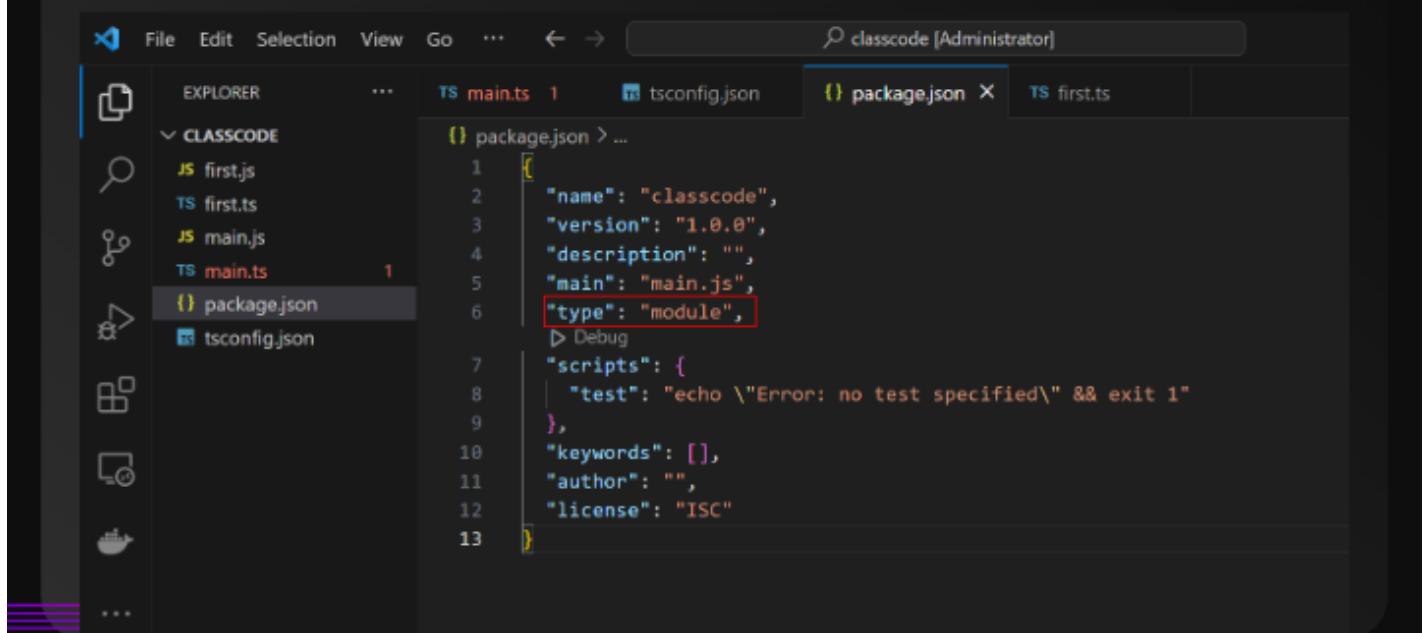
Default JS Config

A screenshot of a code editor interface, likely Visual Studio Code, displaying a file named `package.json`. The file contains the following JSON configuration:

```
1  {
2    "name": "classcode",
3    "version": "1.0.0",
4    "description": "",
5    "main": "main.js",
6    "scripts": {
7      "test": "echo \\\"Error: no test specified\\\" && exit 1"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC"
12 }
13 
```

The code editor has a dark theme. The sidebar on the left shows files: `first.js`, `first.ts`, `main.js`, `main.ts`, and `package.json` (which is selected). The top bar shows tabs for `main.ts`, `tsconfig.json`, `package.json`, and `first.ts`. The status bar at the bottom indicates the file is named `classcode [Administrator]`.

The program we will have make some changes



A screenshot of a code editor interface, likely Visual Studio Code, showing a project named "CLASSCODE". The left sidebar shows files: first.js, first.ts, main.js, main.ts (with a red underline), package.json (selected and highlighted with a yellow background), and tsconfig.json. The right pane displays the contents of the package.json file:

```
1  {
2    "name": "classcode",
3    "version": "1.0.0",
4    "description": "",
5    "main": "main.js",
6    "type": "module",
7    "scripts": {
8      "test": "echo \\\"Error: no test specified\\\" && exit 1"
9    },
10   "keywords": [],
11   "author": "",
12   "license": "ISC"
13 }
```

The line "type": "module" is highlighted with a red rectangle.

Using Native ECMAScript Modules in Node.js

Learn TypeScript / Step 01b: 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 Video: How to Setup Node.js with TypeScript in 2020](#)

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

In the `tsconfig.json` set `module` and `moduleResolution`:

```
'compilerOptions': {  
    'module': 'esNext',  
    'moduleResolution': 'NodeNext',  
    'target': 'es2019',  
},
```

In the `package.json` add:

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

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

```
import {k} from './second.js'
```

Additional Reading:

[Understanding TypeScript 4.2 and ECMAScript module support](#)

[TypeScript and native ES6 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



```
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>
```

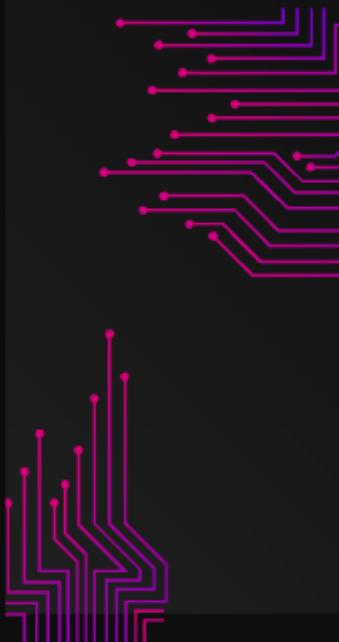
```
File Edit Selection View Go ... ← → ⌂ classcode [Administrator] □ □ □ 08 - ⌂ ...  
EXPLORER CLASSCODE  
node_modules  
main.ts package-lock.json package.json tsconfig.json  
...  
TS main.ts X  
TS main.ts > ...  
1 import inquirer from "inquirer";  
2  
3 let answers = await inquirer.prompt([ {  
4   name: "age",  
5   type: "number",  
6   message: "Enter your Age:"  
7 }]);  
8  
9 console.log("Insha Allah, in " + (60 - answers.age) + " years you will be 60 years old.");
```

```
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.

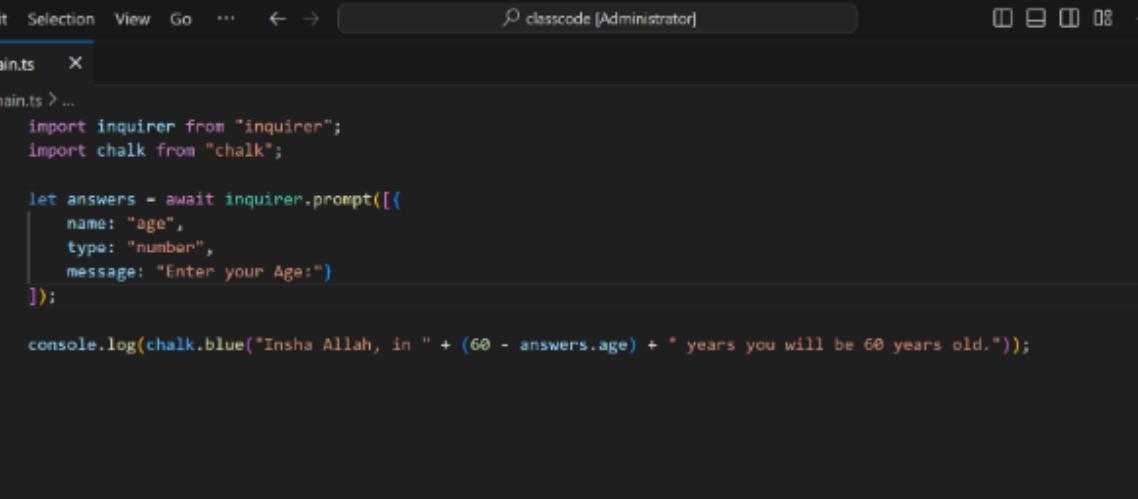
```
D:\code\typescript\classcode>npm i inquirer
added 66 packages, and audited 67 packages in 6s
24 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities

D:\code\typescript\classcode>npm i --save-dev @types/inquirer
added 0 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>npm install chalk
up to date, audited 71 packages in 832ms
24 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities

D:\code\typescript\classcode>
```



```
File Edit Selection View Go ... ← → classcode [Administrator] □ □ □ 08 - ⌂ x  
ts main.ts X  
ts main.ts > ...  
1 import inquirer from "inquirer";  
2 import chalk from "chalk";  
3  
4 let answers = await inquirer.prompt([  
5   name: "age",  
6   type: "number",  
7   message: "Enter your Age:")  
8 ];  
9  
10 console.log(chalk.blue("Insha Allah, in " + (60 - answers.age) + " years you will be 60 years old."));
```

```
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://linktree/ameenalam>



fb.com/SheikhAmeenAlam



linkedin.com/in/ameen-alam



instagram.com/sheikhameenalam



yt.com/ameenalamofficial

Presented by Sir Ameen Alam

