

# TypeScript Mastery

## A Step-by-Step Learning Experience

Presented by Sir Ameen Alam

◀ Parts of this slide didn't load. Try reloading Reload

x



( Part 2 )

# Meet Ameen Alam

An accomplished professional with extensive expertise in Cloud Computing and DevOps. He holds multiple certifications, including AWS Developer Associate and Kubernetes Application Developer, and has over 8 years of experience in the IT, finance, and banking industry. Ameen is currently the Founder & CTO at Doblier Inc.



[fb.com/SheikhAmeenAlam](https://fb.com/SheikhAmeenAlam)



[instagram.com/sheikhameenalam](https://instagram.com/sheikhameenalam)



[linkedin.com/in/ameen-alam](https://linkedin.com/in/ameen-alam)



[yt.com/ameenalamofficial](https://yt.com/ameenalamofficial)



# Content

<https://github.com/panaverse/learn-typescript>

<https://linktr.ee/giaic>

# Step 00 Introduction to TypeScript

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

[https://docs.google.com/presentation/d/1eepDn27eQ8DbRJy7LHTkrlwYx8hQwuoOaDKGS\\_x1Oyc](https://docs.google.com/presentation/d/1eepDn27eQ8DbRJy7LHTkrlwYx8hQwuoOaDKGS_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/1ieepDn27eQ8DbRJy7LHTkrlwYx8hQwuoOaDKGS\\_x1Oyc](https://docs.google.com/presentation/d/1ieepDn27eQ8DbRJy7LHTkrlwYx8hQwuoOaDKGS_x1Oyc)

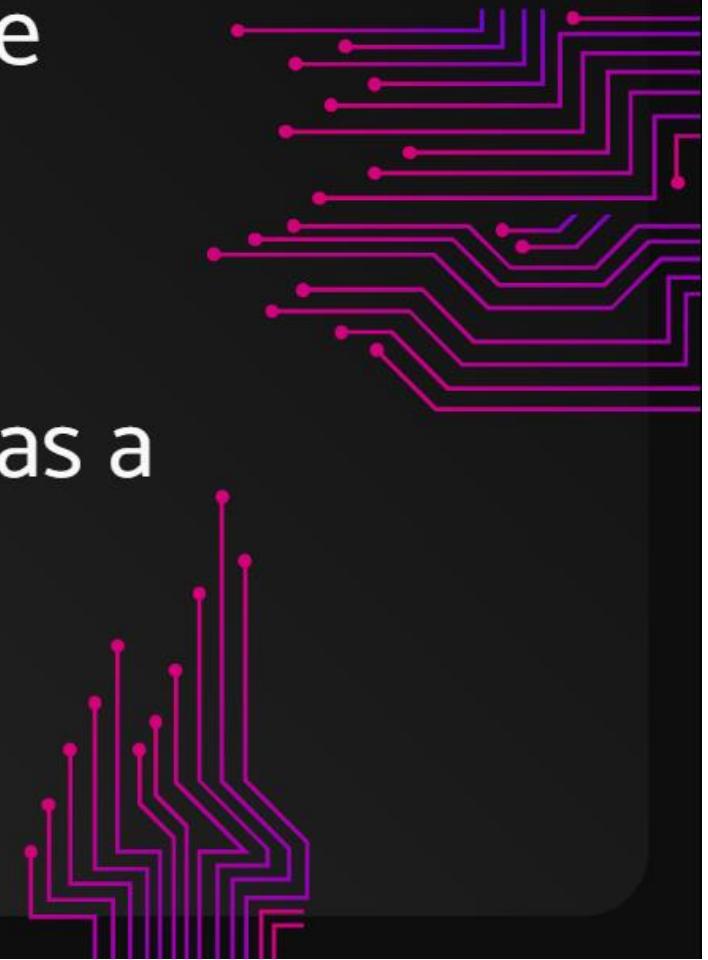
# Step 01

# 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"

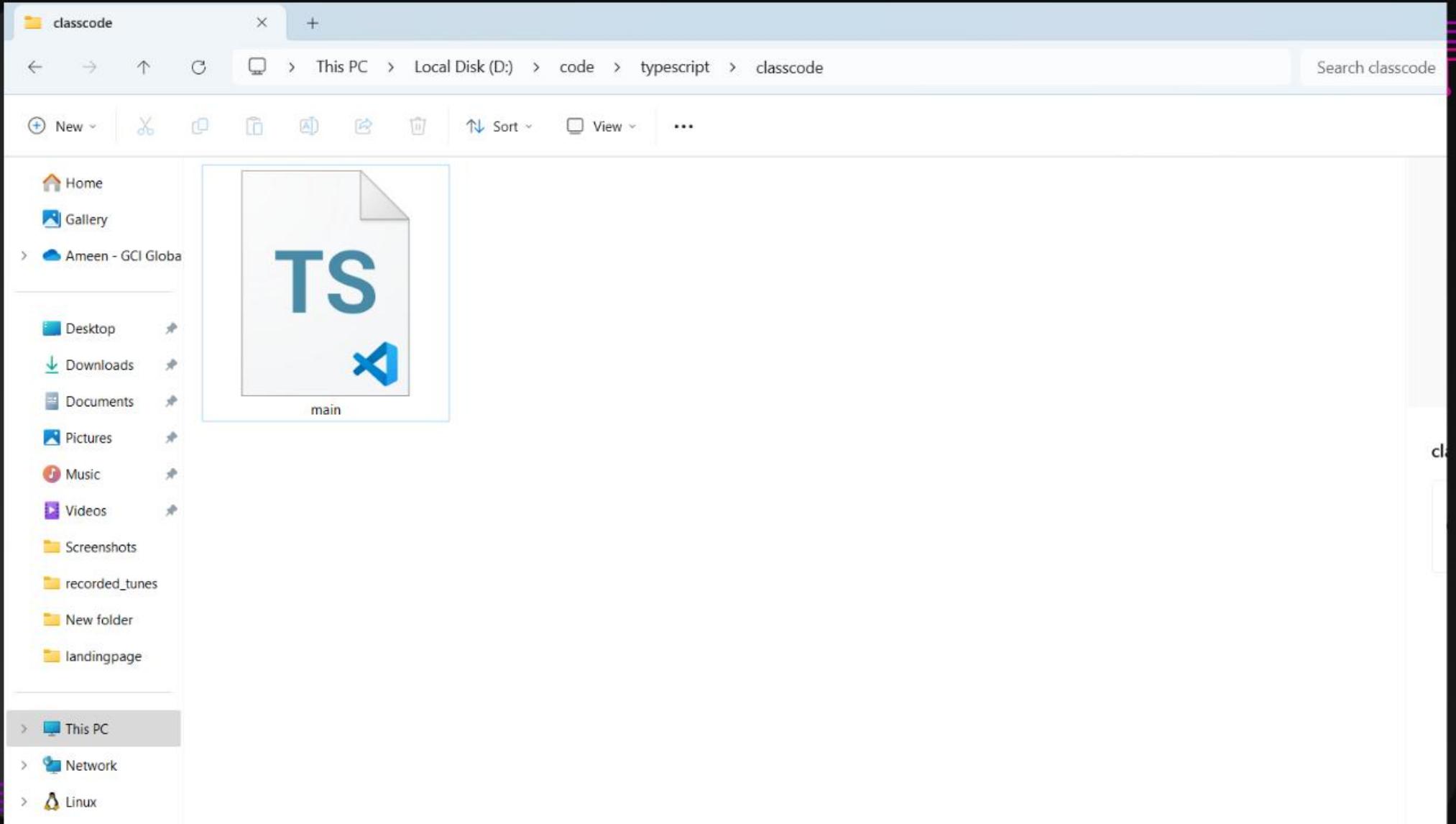
```
let favoriteColor = "blue";
```

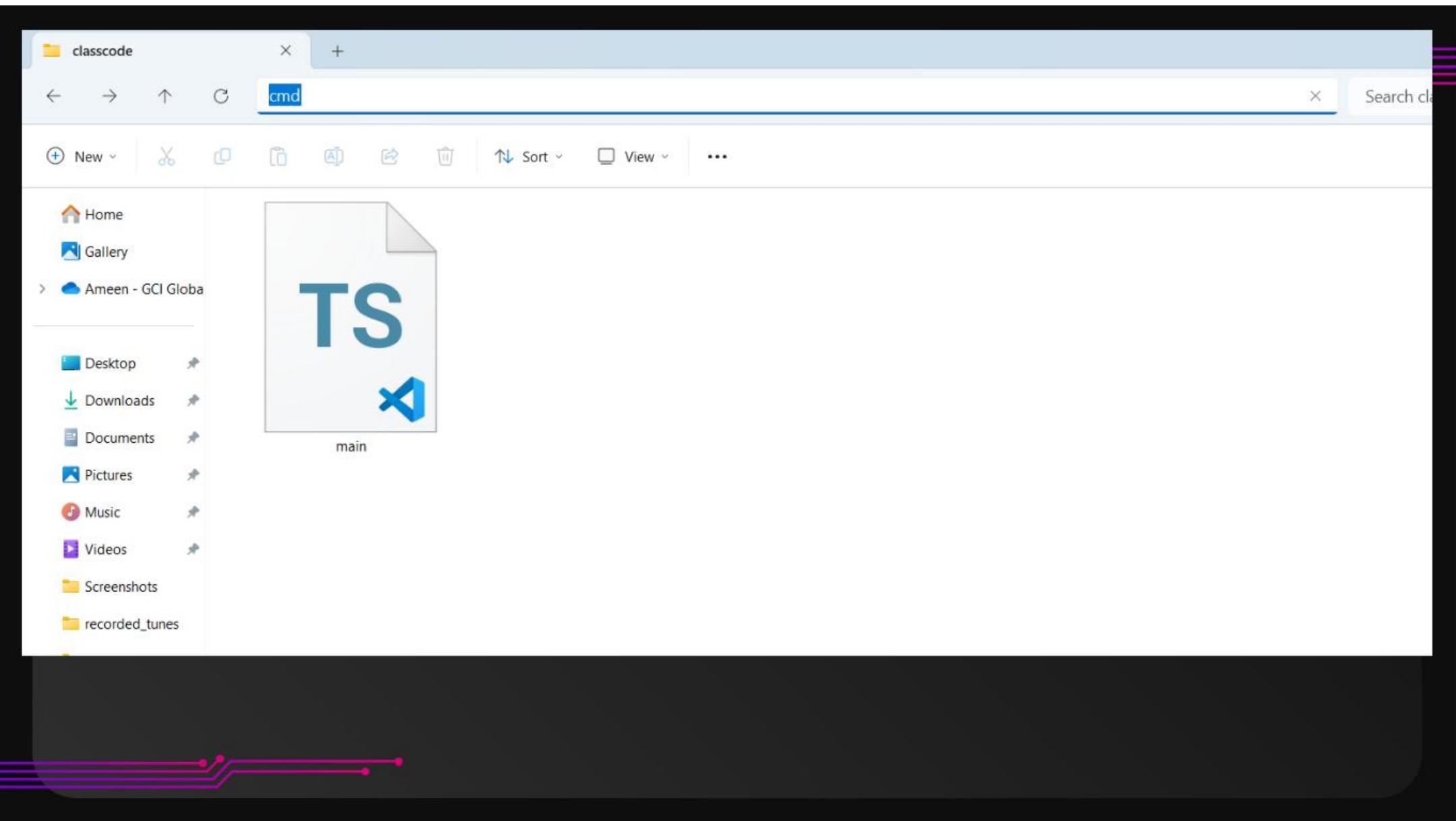


A screenshot of the Microsoft Visual Studio Code (VS Code) interface. The window title bar shows "classcode [Administrator]". The left sidebar includes icons for Explorer, Search, Problems, Outline, and Timeline. The Explorer view shows a folder named "CLASSC..." containing a file "main.ts". The main editor area displays the following TypeScript code:

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

The status bar at the bottom provides information about the current file: "Ln 2, Col 27", "Spaces: 4", "UTF-8", "CRLF", and "TypeScript". There are also icons for closing the editor, live share, and notifications.





Administrator: C:\WINDOWS' + ^



Microsoft Windows [Version 10.0.22621.3155]  
(c) Microsoft Corporation. All rights reserved.

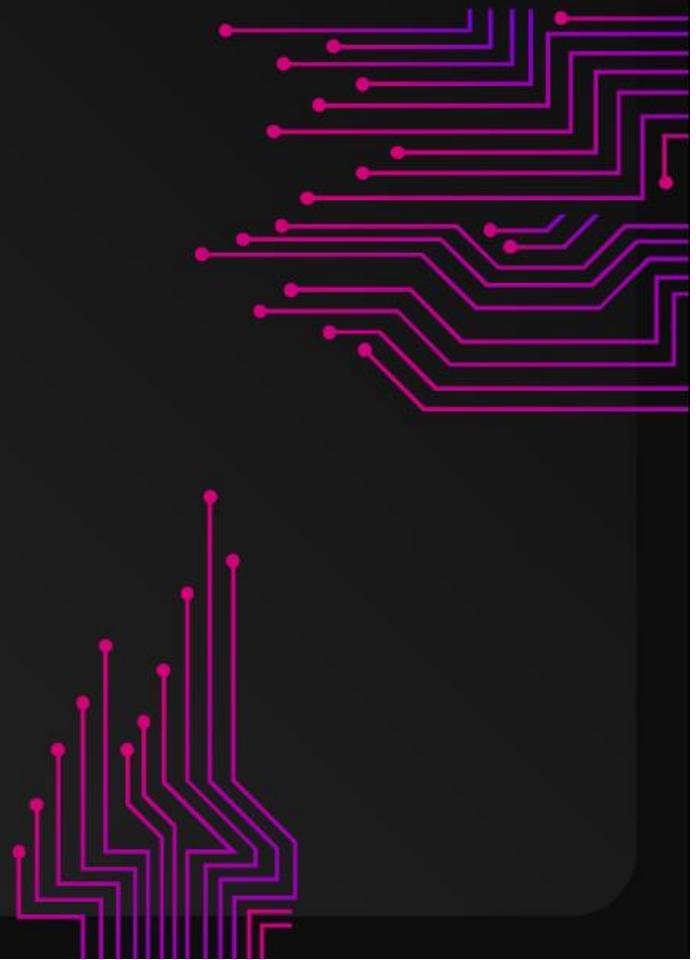
D:\code\typescript\classcode>tsc main.ts

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 "classcode [Administrator]". The left sidebar has icons for Explorer, Search, Problems, Outline, and Timeline. The Explorer section shows a folder named "CLASSCODE" containing a file "main.ts". The main editor area shows the following TypeScript code:

```
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 "Spaces: 4". There are also icons for Live Share, CRLF, and 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

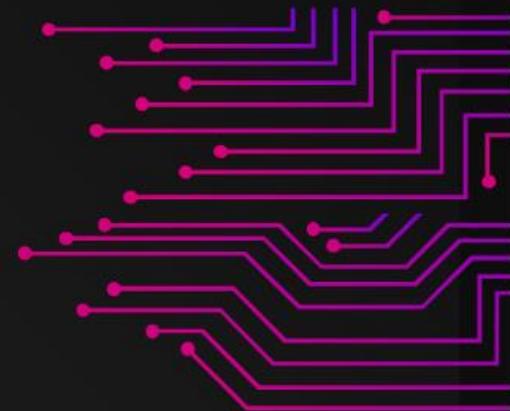
D:\code\typescript\classcode>node main.js  
blue  
green

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

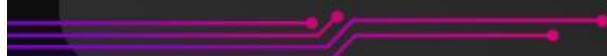


*Explained*

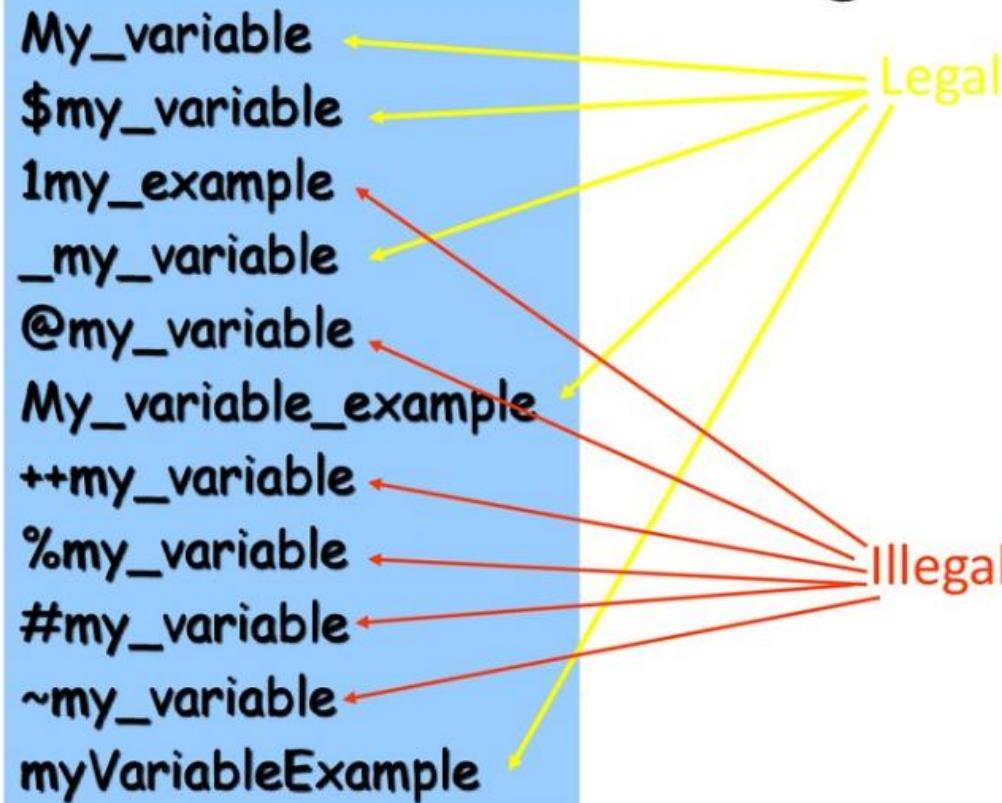
**camelCase**

**snake\_case**

**PascalCase**



# Which one is legal?



# Strong Typing

# Non-Programming

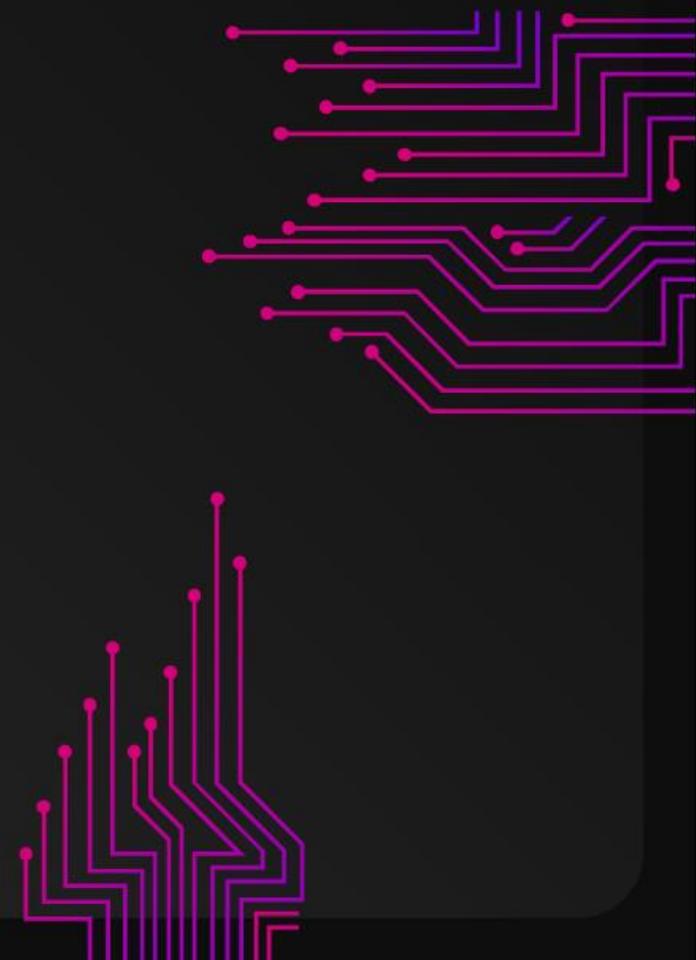
Imagine you're organizing a big event and you've assigned specific roles to your team members:

photographers, decorators, and security personnel.

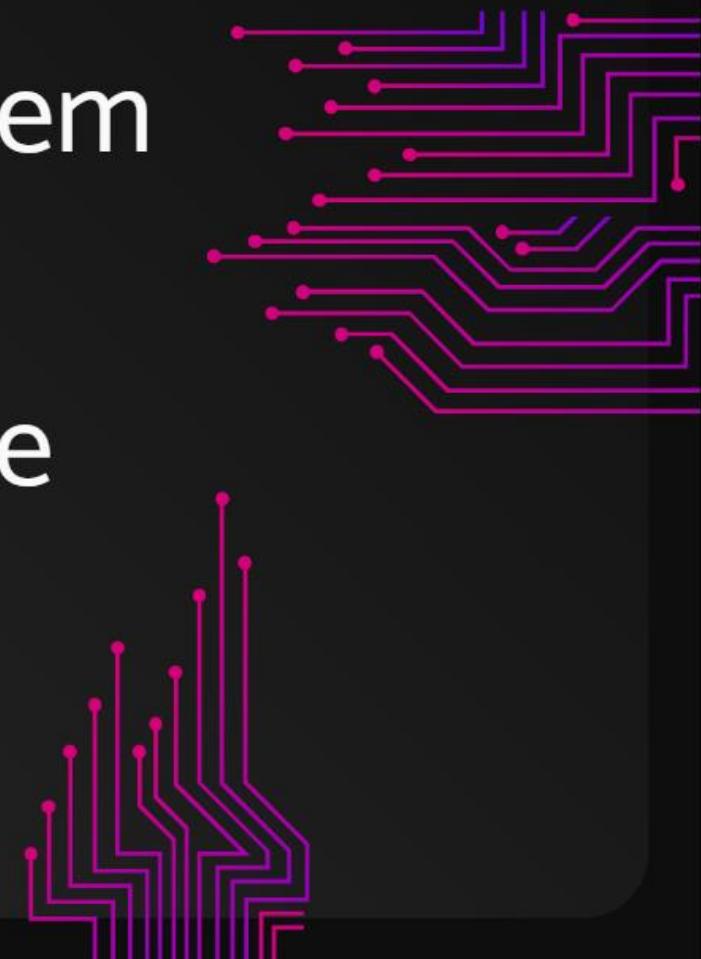
Each role has clear expectations and tasks that cannot be exchanged with others without causing confusion or issues.



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.

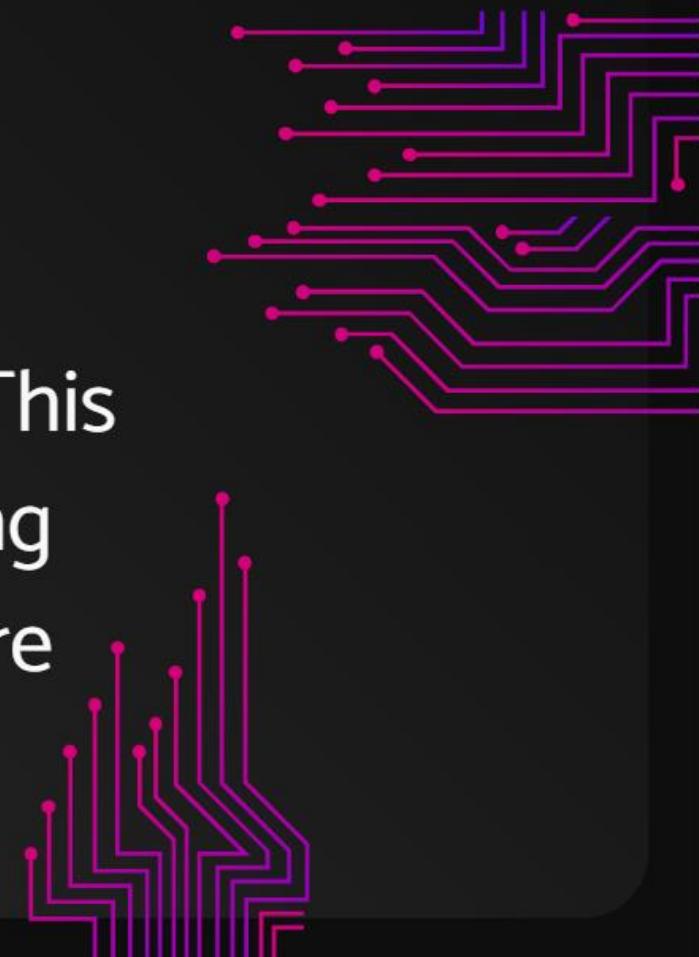


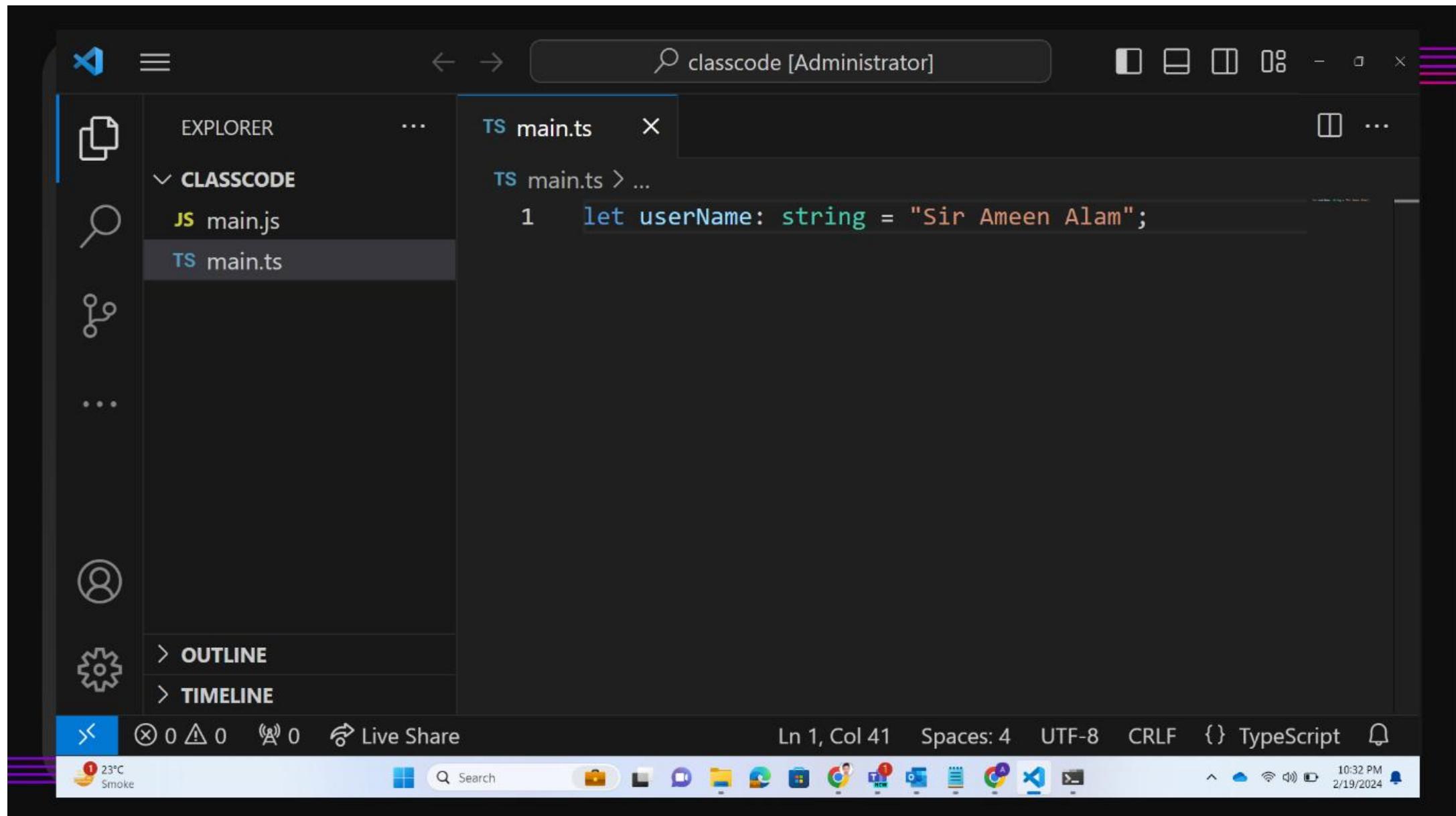
TypeScript assigns types to variables to ensure they are used correctly throughout your code. This helps prevent errors, such as mixing up numbers with text or trying to perform operations on incompatible data types.



# **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.



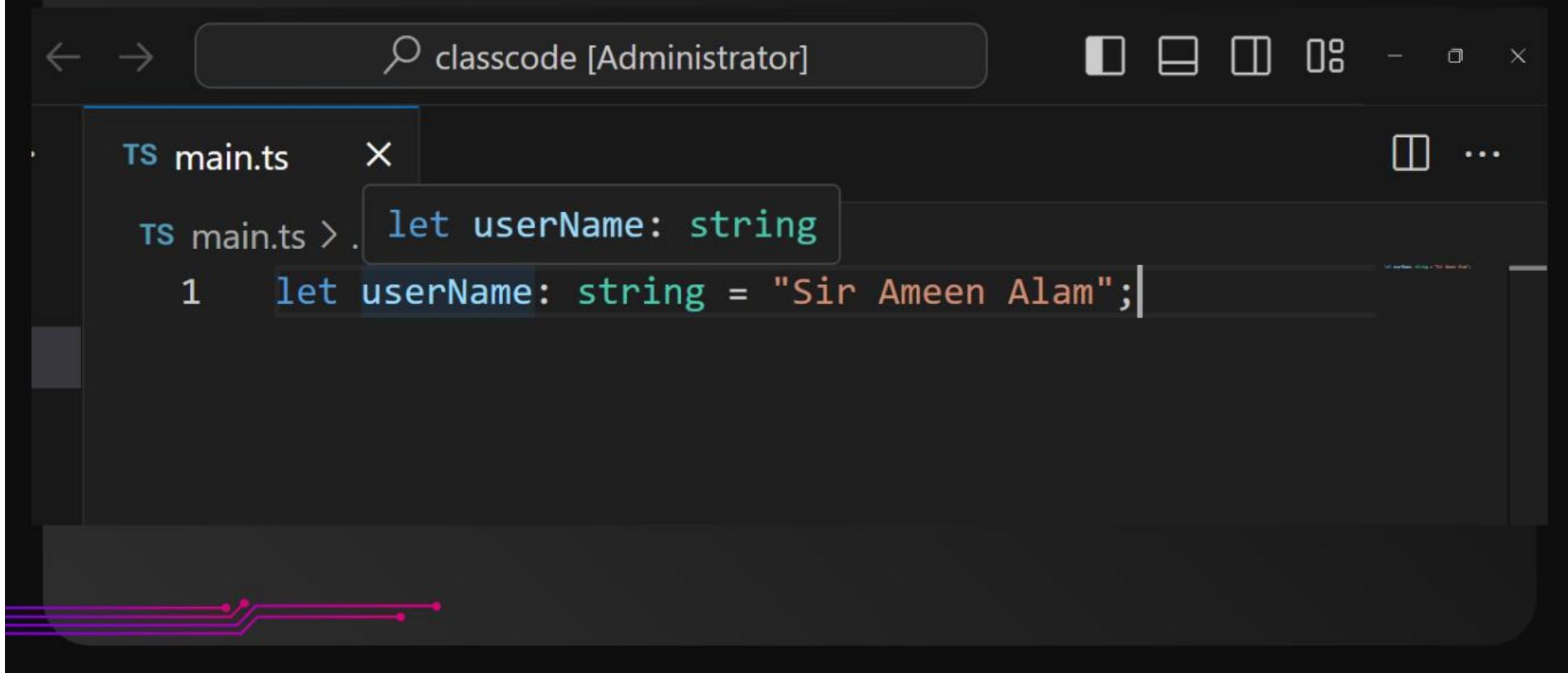


The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The title bar displays the workspace name as "classcode [Administrator]". The left sidebar features the Explorer, Search, Problems, Outline, and Timeline sections. The main editor area shows a TypeScript file named "main.ts" with the following content:

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

The status bar at the bottom provides information about the current file: Line 1, Column 41, 4 spaces, UTF-8 encoding, CRLF line endings, and the language is set to TypeScript. The system tray at the bottom right shows the date and time as 10:32 PM on 2/19/2024.

# Hover your mouse over the Variable



A screenshot of a dark-themed code editor window titled "classcode [Administrator]". The window has a search bar at the top with a magnifying glass icon. Below the search bar are several control icons: a left arrow, a right arrow, a square, a double square, a double square with a dot, a minus sign, a square with a dot, and an 'X'. The main area shows a file named "main.ts". A tooltip is displayed over the line of code "let userName: string". The tooltip contains the text "let userName: string". The code in the editor is as follows:

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

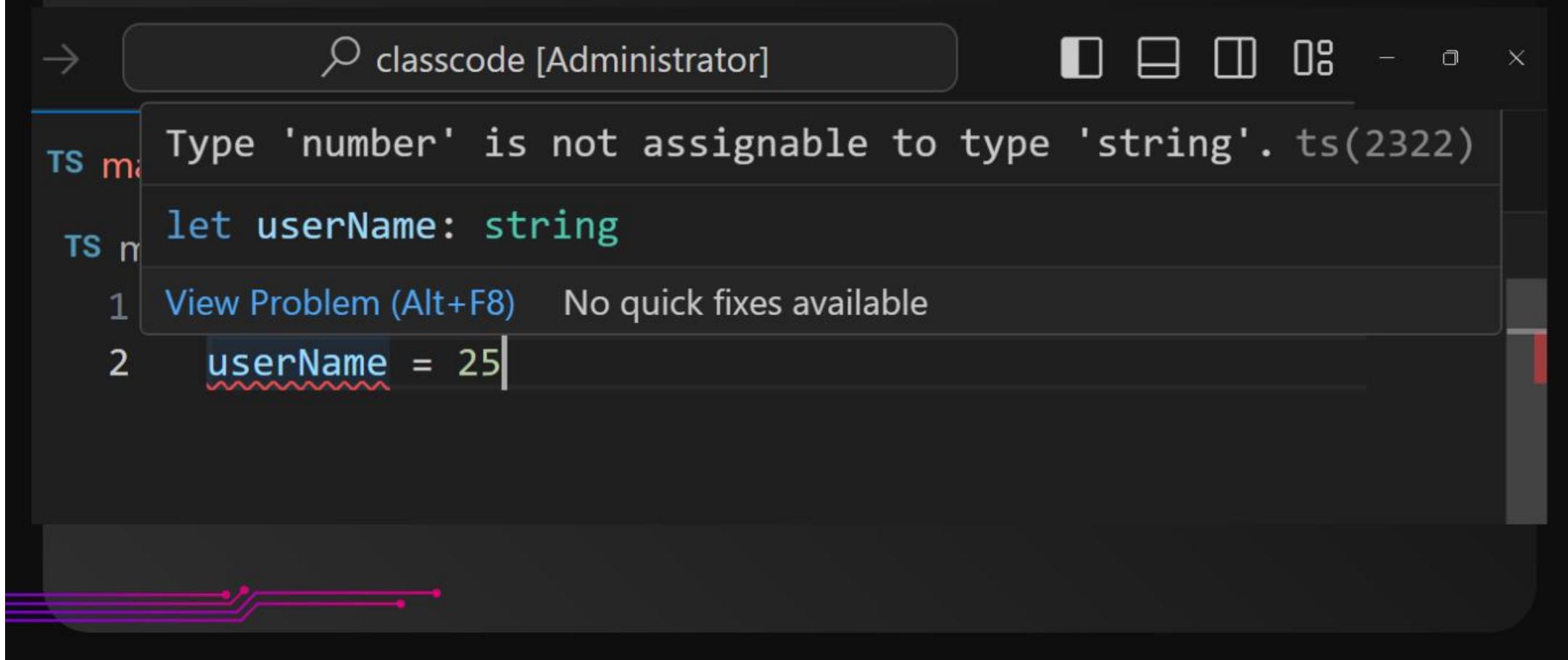
# Error: Type 'number' is not assignable to type 'string'.

A screenshot of a dark-themed code editor window titled "classcode [Administrator]". The window contains a single file named "main.ts". The code is as follows:

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

The variable declaration "let userName: string = "Sir Ameen Alam";" is shown in blue. The assignment statement "userName = 25" is shown in red, with the identifier "userName" underlined by a red squiggle. This indicates a TypeScript error where a number value is being assigned to a string variable. The status bar at the bottom right shows "No errors or warnings in 'main.ts'".

# Hover your mouse over the Variable



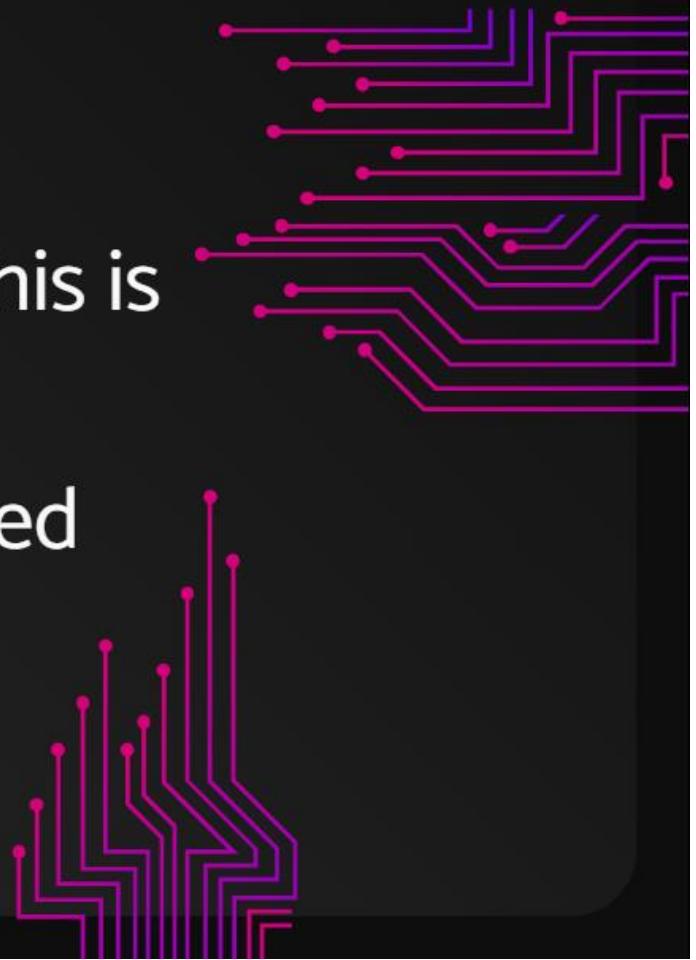
A screenshot of a code editor interface, likely Visual Studio Code, displaying a TypeScript error. The status bar at the top shows the path 'classcode [Administrator]'. The main area shows the following code:

```
TS M Type 'number' is not assignable to type 'string'. ts(2322)
let userName: string
1 View Problem (Alt+F8) No quick fixes available
2 userName = 25
```

The variable 'userName' is highlighted with a red wavy underline, indicating a TypeScript error. A tooltip or status message above the code states: 'Type 'number' is not assignable to type 'string''. Below the code, there is a link to 'View Problem (Alt+F8)' and a note that 'No quick fixes available'.

In the example above:

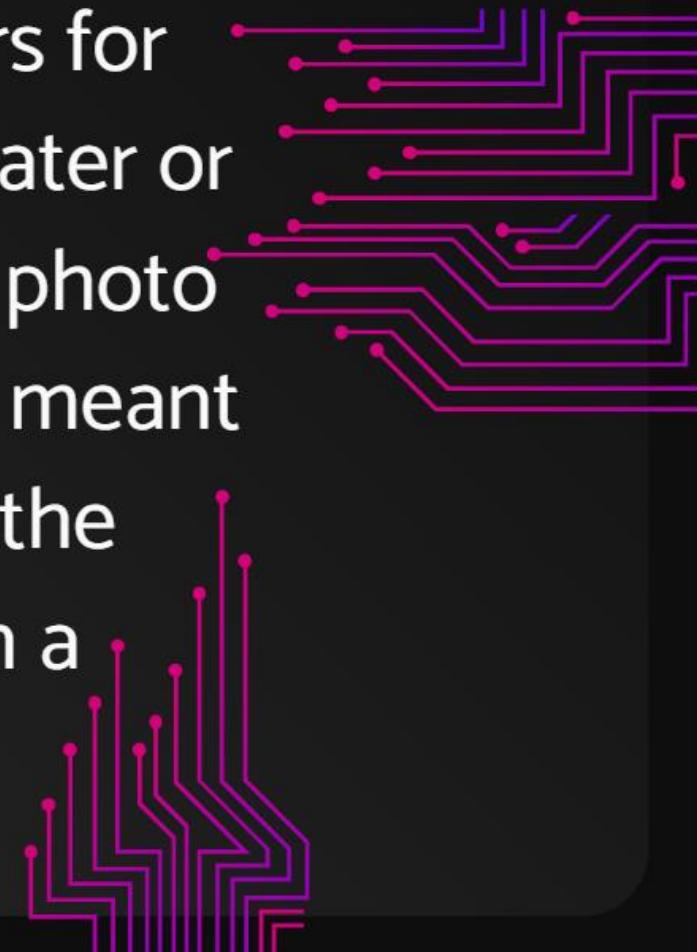
Attempting to assign a number to `userName` would result in an error. This is TypeScript enforcing strong typing, ensuring that variables are always used with the correct type of data



# Primitive Data Types

# 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

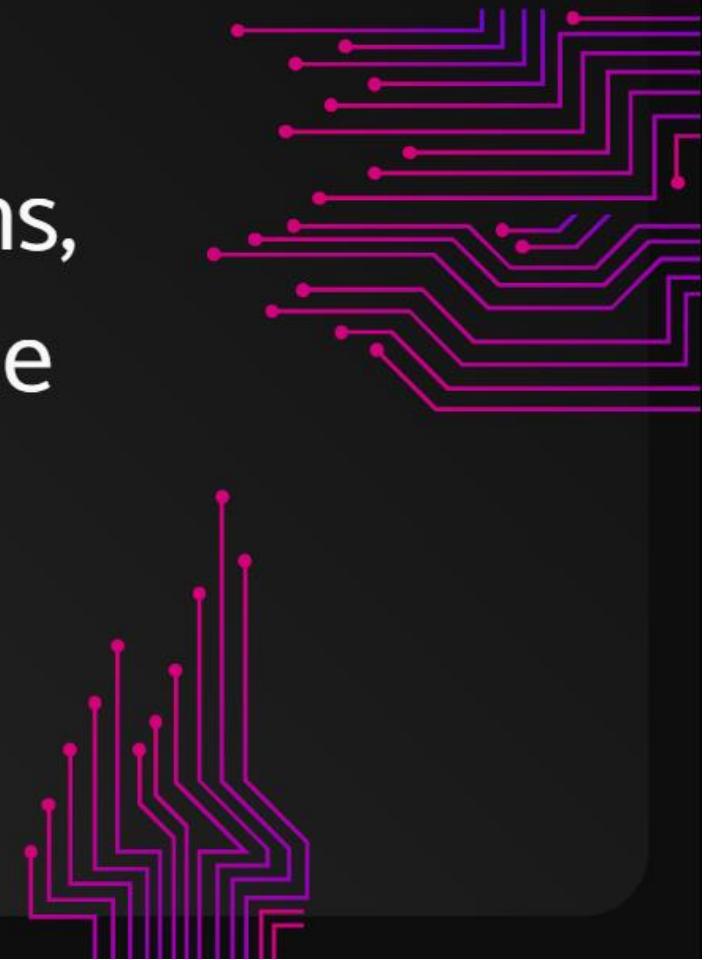
- **String:** For textual data
- **Number:** For numerical values (integers and floating-point numbers)
- **Boolean:** For true/false values
- **Any:** For variables that can hold any type of data

The screenshot shows a dark-themed code editor window. At the top, there's a navigation bar with icons for File, Edit, Selection, and a search bar containing the text "classcode [Administrator]". On the far right of the top bar are three small square icons. The left sidebar features several icons: a file icon, a magnifying glass, a gear, and a grid. The main workspace displays a file named "main.ts". The code within the file is as follows:

```
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



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 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

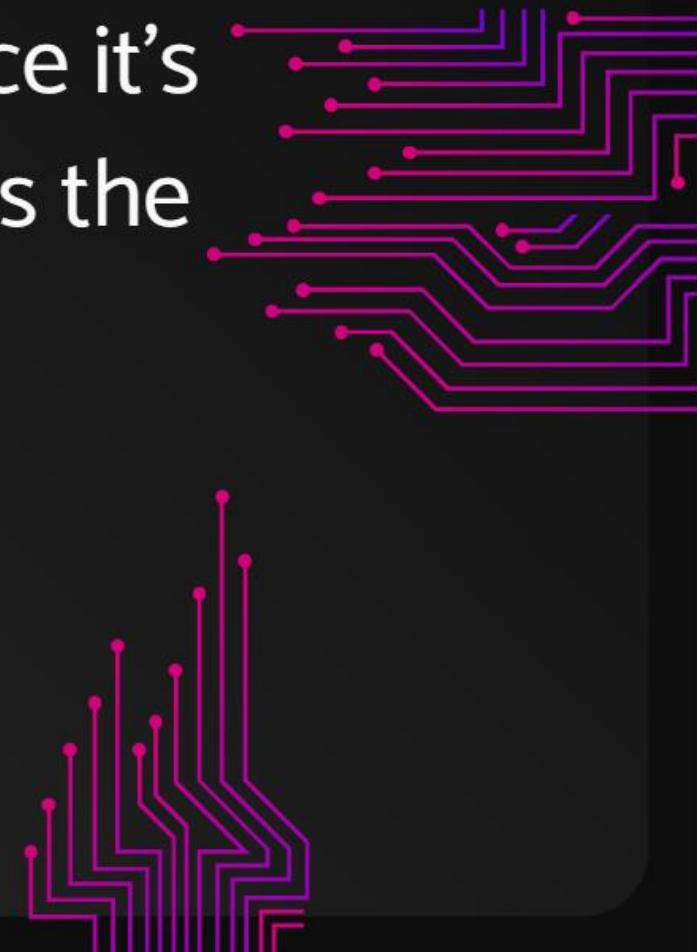
# Non-Programming

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

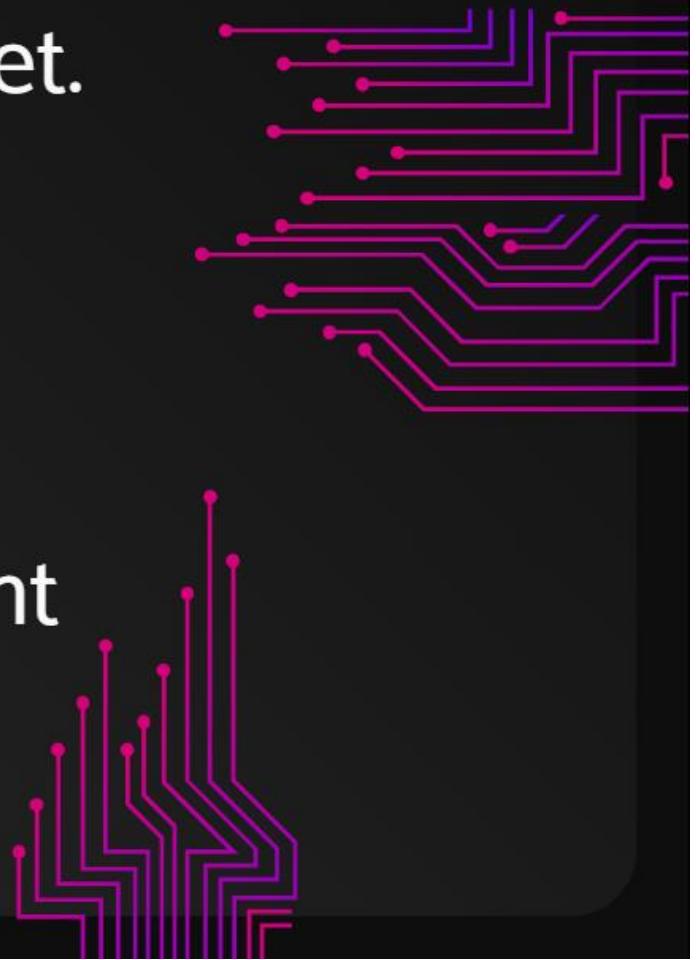
In TypeScript, let and const are two ways to declare variables, with key differences in their mutability and scope.

**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 Primitive 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.

# Errors

# Syntax Error

Identifying and resolving syntax errors in TypeScript code.

A screenshot of the Microsoft Visual Studio Code (VS Code) interface. The title bar shows the VS Code logo, a search bar containing "classcode [Administrator]", and a back/forward navigation bar. The left sidebar features icons for files (with a blue circle containing the number 1), search, and connections. The main editor area displays a file named "main.ts". The code contains two lines:

```
1 lett message = "Hello World"; //syntax error
2 console.log(message);
```

The word "lett" in the first line and "message" in the second line are underlined with red squiggly lines, indicating syntax errors. The status bar at the bottom shows "main.ts 4".

# Hover your mouse over the Keyword

The screenshot shows a code editor interface with a dark theme. At the top, there's a toolbar with icons for file operations and a search bar containing "classcode [Administrator]". Below the toolbar, a sidebar on the left displays a file tree with "main.ts" selected. The main editor area shows the following code:

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

A tooltip has appeared over the word "lett", which is underlined with a red squiggly line indicating a syntax error. The tooltip contains 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 Microsoft Visual Studio Code interface. The title bar includes icons for file, search, and other functions, followed by the text "classcode [Administrator]". 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 is displayed over the misspelled method "loger". The tooltip contains the following text:

Property 'loger' does not exist on type 'Console'. Did you mean 'log'? ts(2551)  
lib.dom.d.ts(26638, 5): 'log' is declared here.  
any

At the bottom of the tooltip, there are two links: "View Problem (Alt+F8)" and "Quick Fix... (Ctrl+.)".

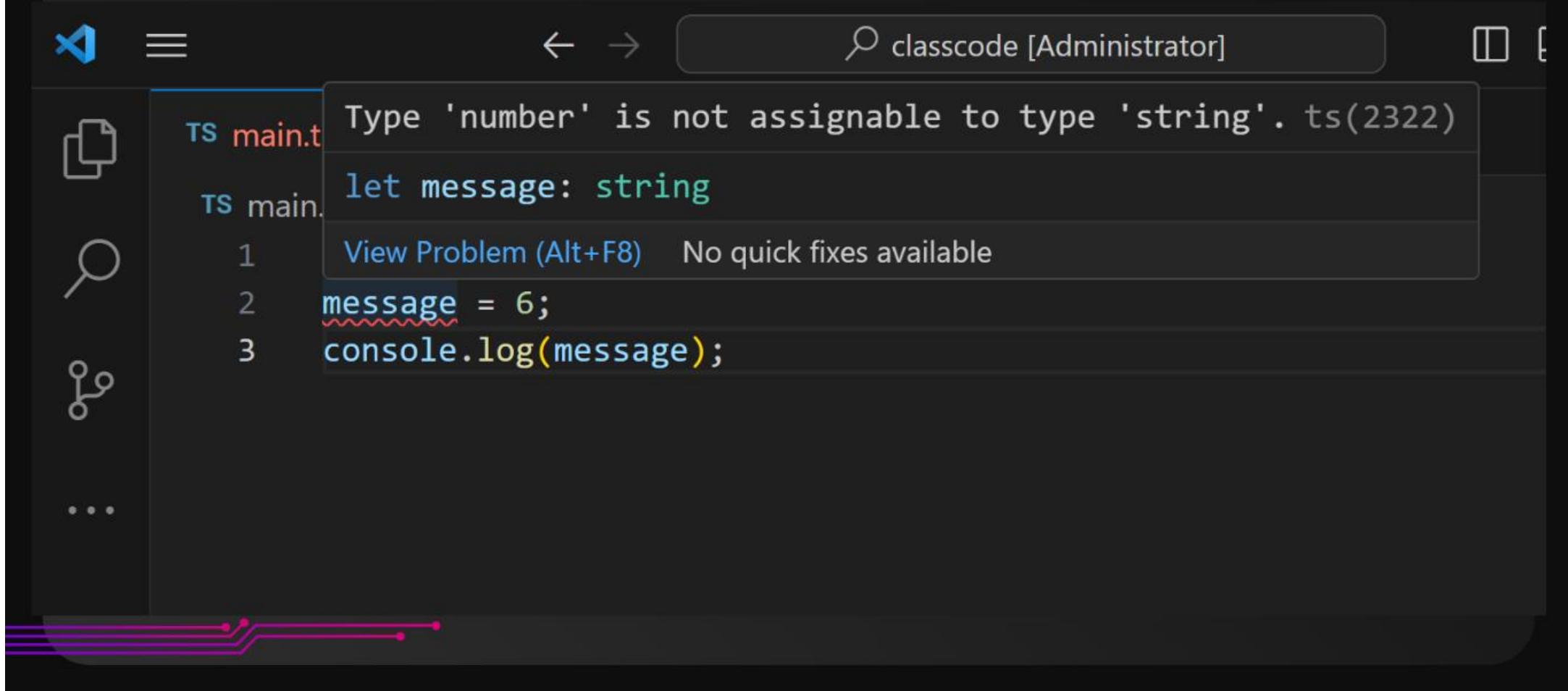
# Assignability Error

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

A screenshot of a dark-themed code editor interface. The top bar features a blue 'VS' logo, a three-line menu icon, a back arrow, a forward arrow, and a search bar containing the text 'classcode [Administrator]'. The left sidebar includes icons for file operations, search, and other tools, with three dots at the bottom indicating more options. The main workspace displays a TypeScript file named 'main.ts' with the extension 'TS'. The code contains three numbered lines:

```
1 let message = "Hello World";
2 message = 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 assignment statement:

```
let message: string  
message = 6;  
console.log(message);
```

A tooltip is displayed over the assignment statement `message = 6;`, indicating a TypeScript error:

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

The tooltip also includes links to "View Problem (Alt+F8)" and "No quick fixes available".

# String Concatenation



TS main.ts X

TS main.ts > [e] fullName

```
1 let firstname: string = 'Ameen';
2 let lastname: string = 'Alam';
3 let fullName: string| = firstname + ' ' + lastname;
4 console.log(fullName)
```





# Template Literals

TS main.ts > ...

```
1 let firstname: string = 'Ameen';
2 let lastname: string = 'Alam';
3 let fullName: string = `${firstname} ${lastname}`;
4 console.log(fullName);
5
```



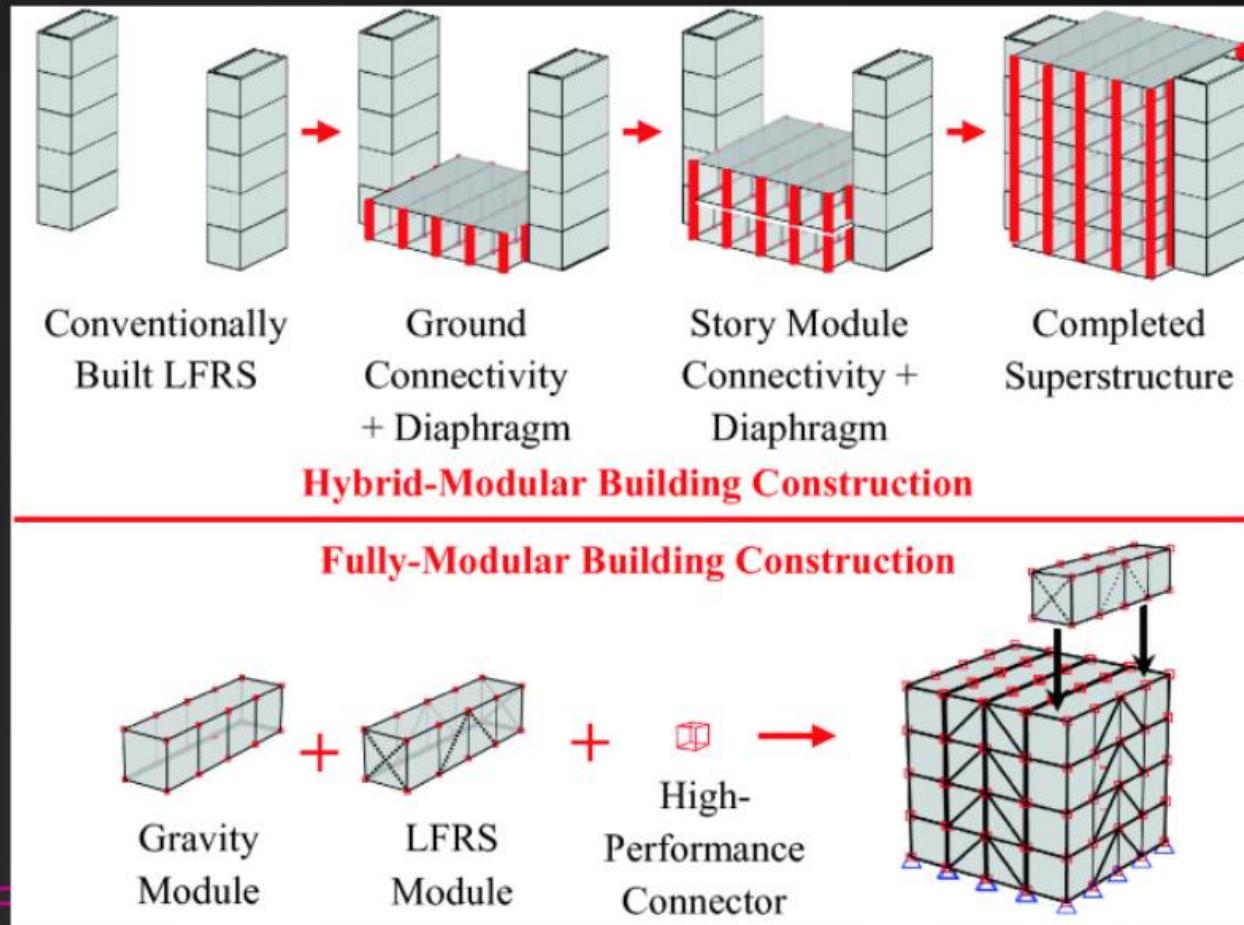
# Modules

Basics of modules, exporting, and  
importing modules.

# Modular Architecture



# Fully-Modular Buildings



## **Home Work:**

We will code Module later in our advanced session; for now, go to the advanced slides, comprehend the module, and utilize the Inquirer.

# Operators in TypeScript

Operators allow us to perform operations  
on variables and values.

**Imagine you're at a  
grocery store**

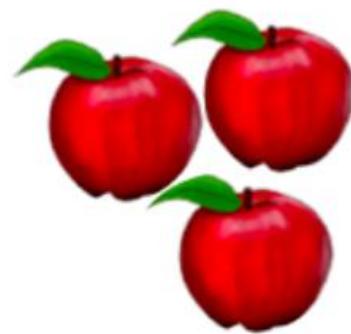
# Addition (+)

Addition (+): You add apples to your cart. 2 apples and then 3 more apples give you 5 apples.

```
console.log(2 + 3)
```

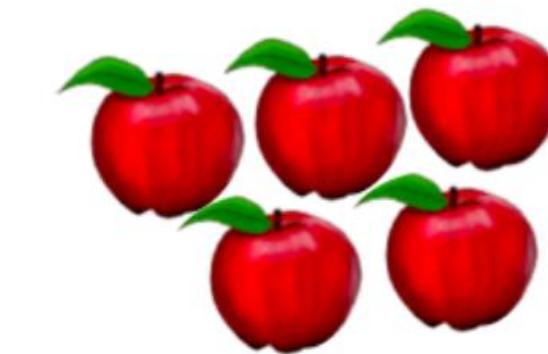


**2**



**+**

**3**



**=**

**5**

```
let num1: number = 2;
```

```
let num2: number = 3;
```

```
let cart: number = num1 + num2;
```

```
console.log(cart)
```

TS main.ts



...

TS main.ts > ...

```
1 let num1: number = 2;  
2 let num2: number = 3;  
3 let cart: number = num1 + num2;  
4 console.log(cart) // 5  
5
```

# Subtraction (-)

Subtraction (-): You have 5 apples in cart and you eaten 2 apples, leaving you with 3 apples.

```
console.log(5 - 2)
```

5 Apples



Eat 2



2 Left



```
let cart: number = 5;
```

```
let num3: number = 2;
```

```
let total: number = cart - num3
```

```
console.log(total)
```

Explorer (Ctrl+Shift+E)

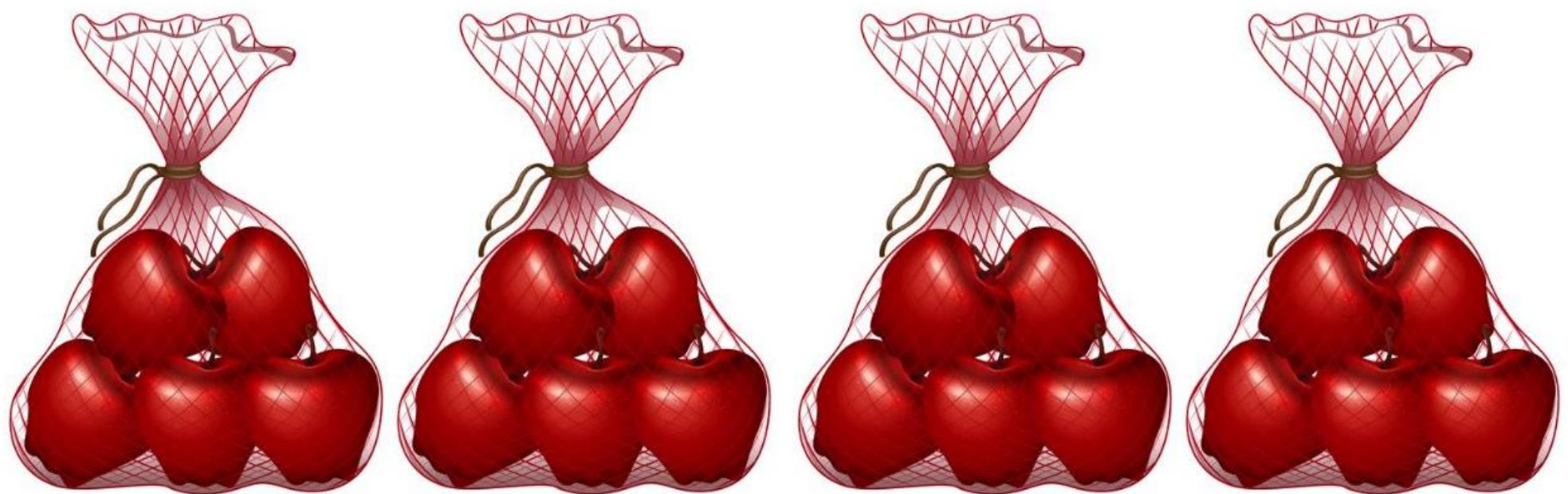
TS main.ts > ...

```
1 let cart: number = 5;
2 let num3: number = 2;
3 let total: number = cart - num3
4 console.log(total) // 3
```

# Multiplication (\*)

Multiplication (\*): You decide you need 4 bags of 5 apples. You now have 20 apples.

```
console.log(4 * 5)
```



```
let bags: number = 4;
```

```
let apples: number = 5;
```

```
let totalApples: number = bags * apples
```

```
console.log(totalApples)
```

TS main.ts X

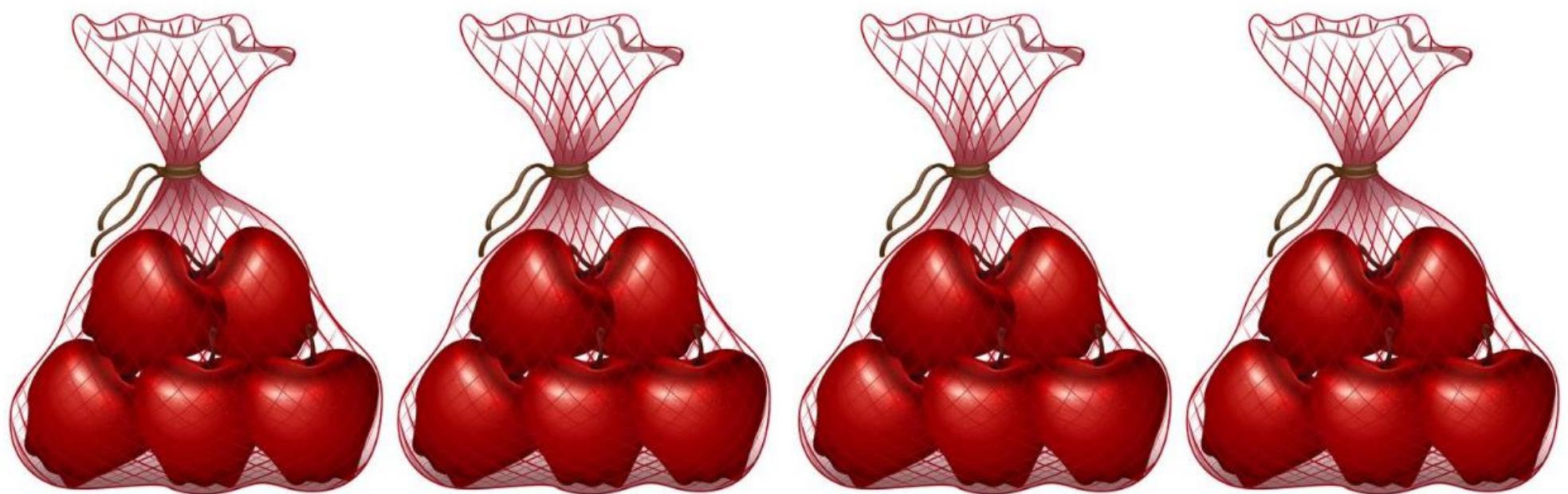
TS main.ts > ...

```
1 let bags: number = 4;
2 let apples: number = 5;
3 let totalApples: number = bags * apples
4 console.log(totalApples) // 20
```

# Division (/)

Division (/): You decide to distribute these 20 apples equally into 4 bags. Each bag gets 5 apples.

```
console.log(20 / 4)
```



```
let totalApples: number = 20;  
let bags: number = 4;  
let eachBags: number = totalApples / bags  
console.log(eachBags)
```

TS main.ts X

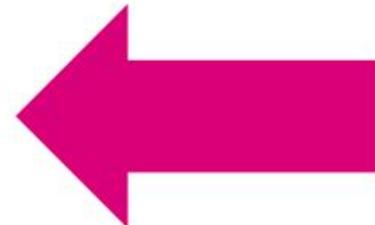
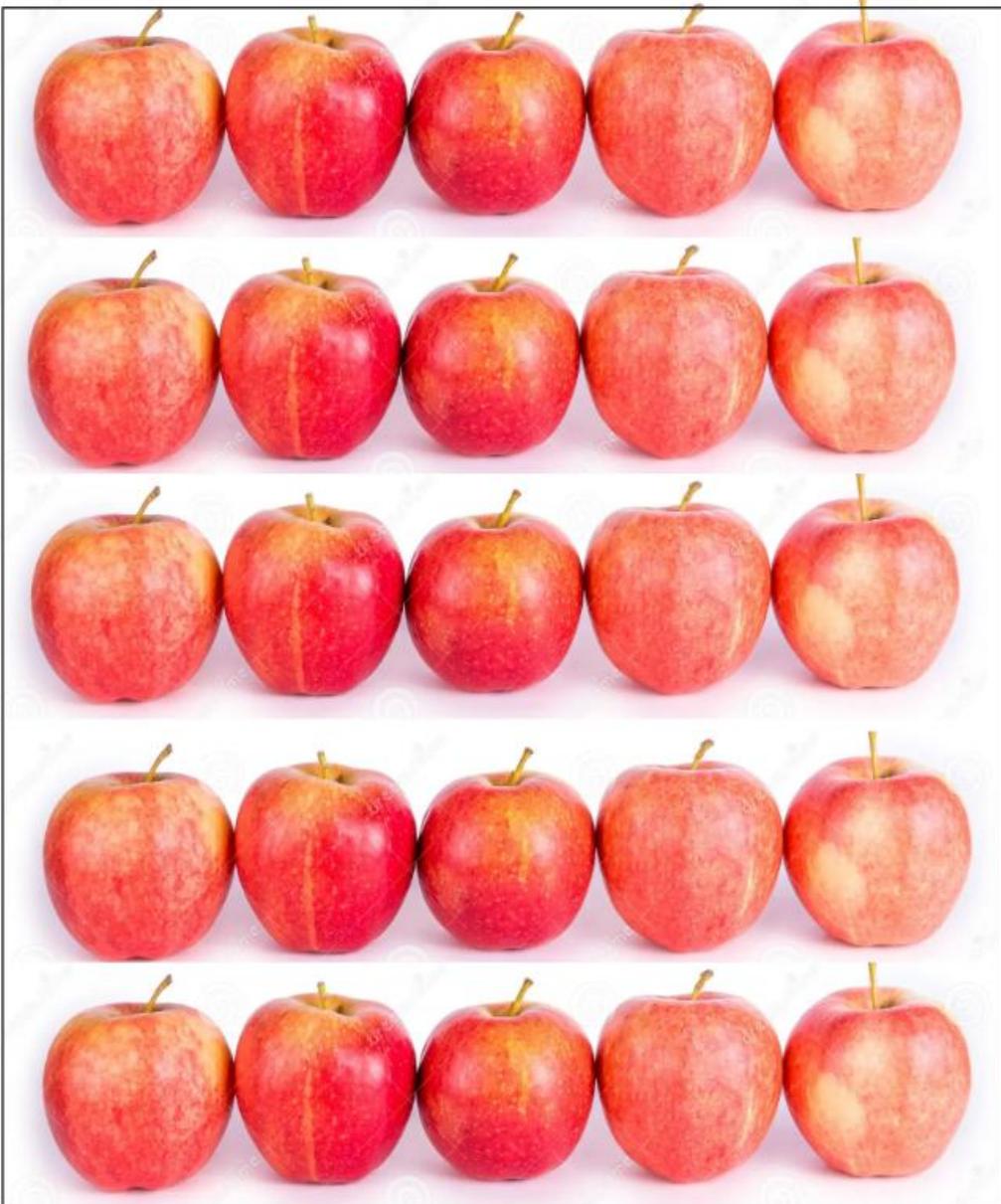
TS main.ts > ...

```
1 let totalApples: number = 20;
2 let bags: number = 4;
3 let eachBags: number = totalApples / bags
4 console.log(eachBags) // 5
```

# Exponentiation (\*\*)

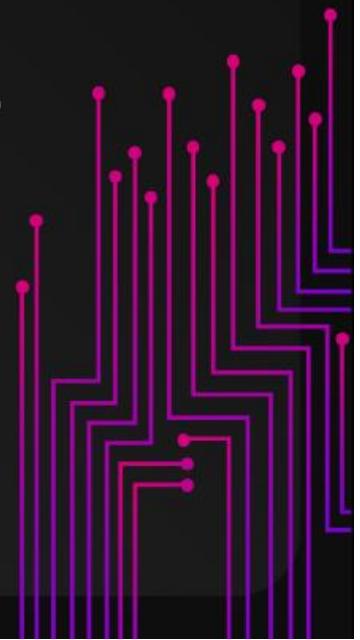
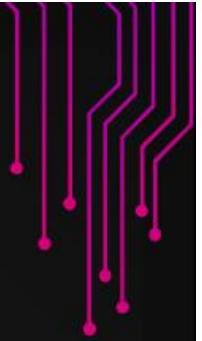
Exponentiation ()**\*\***: You decide to buy boxes of apples, each box contains 5 layers of apples, and each layer has 5 apples. So, 5 boxes will have  $5^2 = 25$  apples each.

```
console.log(5 ** 2)
```

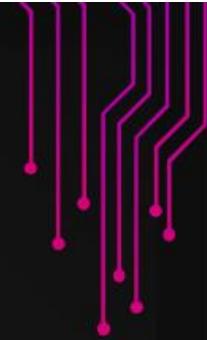


**Square  
25 Apples  
Each Box**

Each layer holds 5 apples, and there are 5 layers, so I multiply 5 (apples per layer) by 5 (layers) to get the total number of apples.



If you decide that the number of apples in each layer should be squared (meaning you multiply the number of apples by itself), with 5 layers, the calculation would be "5 layers squared," which is 5 times 5, giving you 25 apples in total according to this specific calculation method.



```
let layer = 5
```

```
let apple = 5
```

```
let power = layer ** 2;
```

```
console.log(power)
```



main.ts > [➊] power

```
1 let layer:number = 5
2 let apple:number = 5
3 let power:number = layer ** 2
4 console.log(power) // 25
```



# Modulus (%)

Modulus (%): You have 5 apples; you distribute them into 2 bags. Each bag gets 2 apples, and 1 apple remains ( $5 \% 2 = 1$ ).

```
console.log(5 % 2)
```

5 Apples



I am a  
Remainder



5 Apples



```
let apple = 5
```

```
let bags = 2
```

```
let remainder = apple % bags;
```

```
console.log(remainder)
```

TS main.ts



TS main.ts > ...

```
1 let apple = 5
2 let bags = 2
3 let remainder = apple % bags;
4 console.log(remainder) // 1
```

5

# Unary operators

(++) (--)

Prefix and postfix operators

Unary operators: Imagine you have a loyalty card that you use once (++), and then a coupon that you use once and throw away (--).

```
let a=0
```

```
console.log(a++);
```

TS main.ts

X



TS main.ts > ...

```
1 let a=5
2 let b=2
3 a++; // a becomes 6
4 b--; // b becomes 1
```

TS main.ts

X

TS main.ts > ...

1 let a=5

2 let b=2

3 ++a; // a becomes 6

4 --b; // b becomes 1



# Home Work

```
let a: number = 5; let b: number = 2;  
let c: number;  
c = ++a + a++ + --b + b-- + a + b++ + b;  
console.log(c);
```



TS main.ts X

TS main.ts > ...

```
1 let a:number=5;
2 let b:number=2;
3 let c:number;
4 c = ++a + a++ + --b + b-- + a + b++ + b;
5 console.log(c);
```

# Combining Operators

TS main.ts X

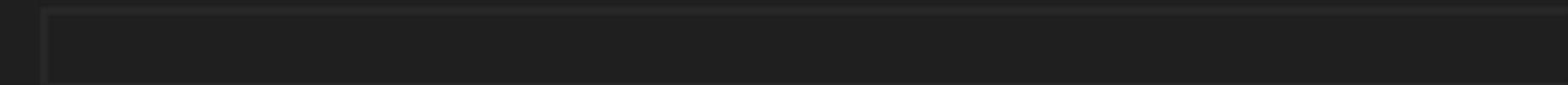
TS main.ts > ...

```
1 let result = 3 + 4 * 5;  
2 console.log(result)  
3 // Answer will be 23 or 35 or ??  
4
```



TS main.ts > ...

```
1 let result = 3 + 4 * 5;  
2 result++  
3 console.log(result)  
4 // Answer will be 23 or 24 or 35 or 36 or ??  
5
```



# Addition Calculator

$$2 + 3 = ?$$

TS main.ts > ...

```
1 import inquirer from "inquirer";
2 const input1 = await inquirer.prompt({
3     name: "num1",
4     type: "number",
5     message: "kindly enter your first no:"
6 });
7 const input2 = await inquirer.prompt({
8     name: "num2",
9     type: "number",
10    message: "kindly enter your second no:"
11 });
12 let total: number = input1.num1 + input2.num2
13 console.log(total);
```



Administrator: C:\WINDOWS

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

```
D:\code\typescript\classcode>node main.js
```

```
? kindly enter your first no: 2
```

```
? kindly enter your second no: 3
```

```
5
```

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

# BMI Calculator

```
ts main.ts > ...
```

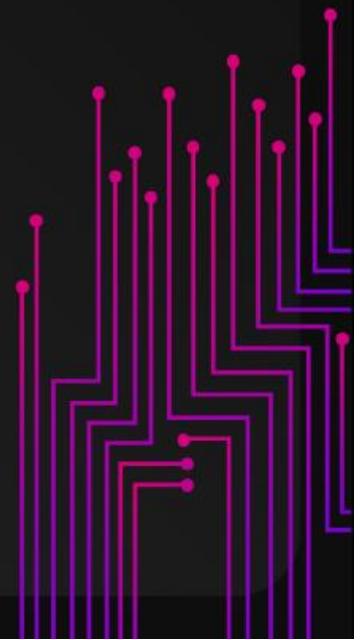
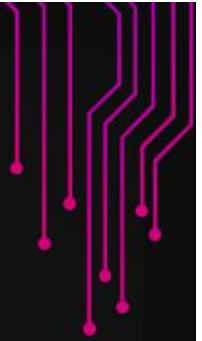
```
1 let weightInKg = 70; // 70kg
2 let heightInMeters = 1.75; // 1.75m
3 let bmi = weightInKg / (heightInMeters * heightInMeters)
4 console.log(`Your BMI is ${bmi}`);
5 |
```

# Home Work

Create Addition, Subtraction,  
Multiplication, Division,  
Exponentiation, Modulus and  
BMI Calculator using inquirer.

# Assignment Operators (=)

Assignment Operators (=): You assign tasks to your family members. For example, you assign the task of washing dishes to your sibling.



TS main.ts X

TS main.ts > ...

```
1 let c = 10;
```

```
2 c += 5;
```

```
3 // equivalent to c = c + 5, c is now 15
```

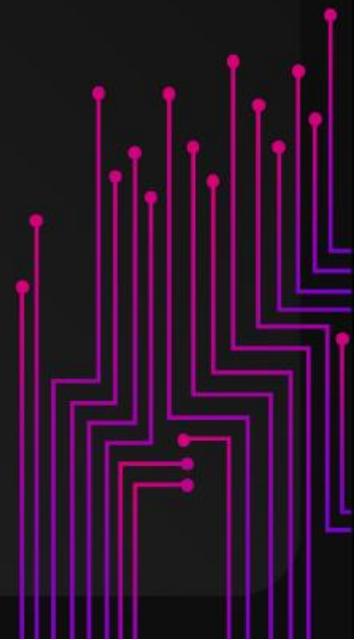
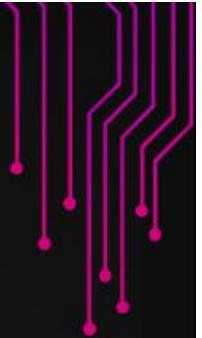
# Comparison Operators

`5 == 2` is false

`5 != 2` is true

`5 > 2` is true

`5 < 2` is false



```
main.ts > [o] b  
1 let a=5  
2 let b=2  
3 let isEqual = (a == b); // false  
4 let isNotEqual = (a != b); // true  
5 let isGreaterThan = (a > b); // true  
6 let isLessThan = (a < b); // false
```

# Logical Operators

$(5 > 0) \&\& (2 > 0)$  is true

$(5 < 0) \parallel (2 > 0)$  is true

$!(5 > 0)$  is false

TS main.ts X

TS main.ts > ...

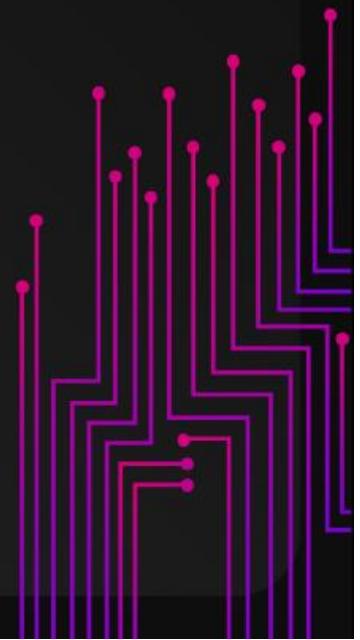
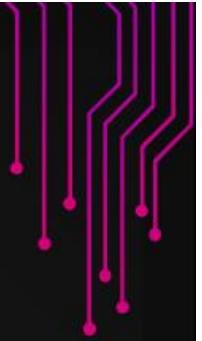
```
1 let a=5
2 let b=2
3 let logicalAnd = (a > 0) && (b > 0); // true
4 let logicalOr = (a < 0) || (b > 0); // true
5 let logicalNot = !(a > 0); // false
6
```

# Logic Statements

# If and If-Else Statements

## If and If-Else Statements:

Imagine deciding what to wear based on the weather. If it's raining, you wear a raincoat. Otherwise (else), you wear sunglasses.



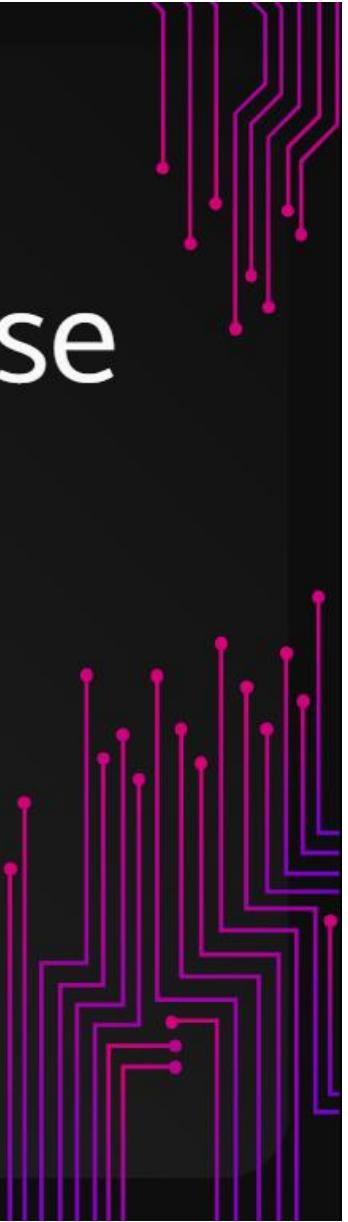
main.ts > ...

```
1 let isRaining = true;
2 if (isRaining) {
3     console.log("Wear a raincoat.");
4 } else {
5     console.log("Wear sunglasses.");
6 }
```

```
let isRaining = false;  
if (isRaining) {  
  console.log("Wear a raincoat.");  
} else {  
  console.log("Wear sunglasses.");  
}
```

# Else If Statements

Extending the above, if it's raining, you wear a raincoat. Else if it's cloudy, you wear a light jacket. Otherwise, you wear sunglasses.



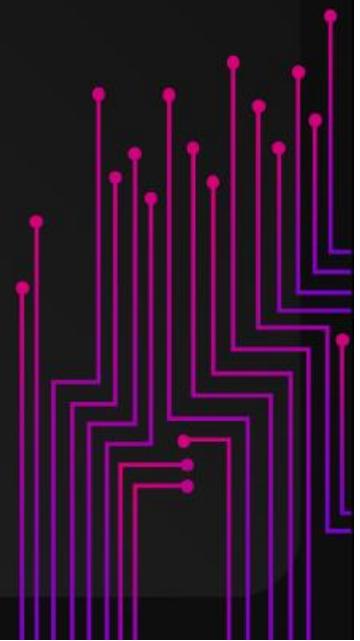
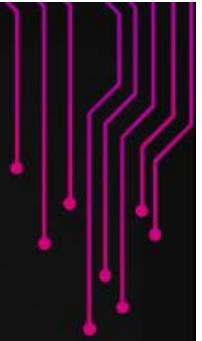
main.ts > ...

```
1 let weather = "cloudy";|
2 if (weather === "raining") {
3     console.log("Wear a raincoat.");
4 } else if (weather === "cloudy") {
5     console.log("Wear a light jacket.");
6 } else {
7     console.log("Wear sunglasses.");
8 }
```

# Conditional Ternary Operators

# Conditional Ternary Operators:

When deciding on a snack, you think, "Am I hungry?" If yes, you eat an apple; if not, you drink water.





```
let isHungry = true;  
let snack = isHungry ? "apple" : "water";  
console.log(`You should have some ${snack}.`);
```





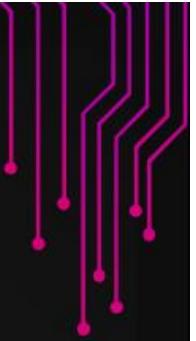
```
let isHungry = false;  
let snack = isHungry ? "apple" : "water";  
console.log(`You should have some ${snack}.`);
```

|



# Switch Statements

Switch Statements: Choosing what to do on your day off based on the day. If it's Saturday, you go hiking. If it's Sunday, you read a book. Otherwise, you work on a hobby.



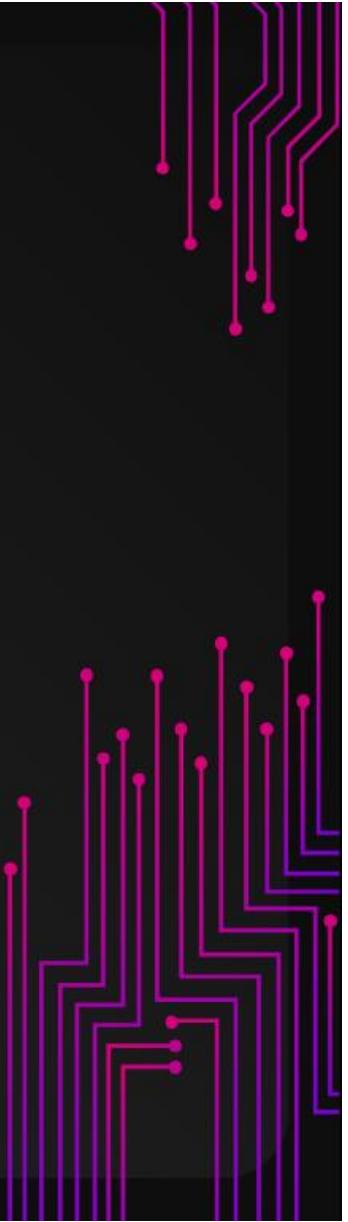


```
let dayOff = "Sunday";  
switch (dayOff) {  
  case "Saturday":  
    console.log("Go hiking.");  
    break;  
  case "Sunday":  
    console.log("Read a book.");  
    break;  
  default:  
    console.log("Work on a hobby.");  
}
```

# Self-Check Quiz

A simple quiz that evaluates answers  
using if-else statements:

A simple quiz that  
evaluates answers using  
if-else statements



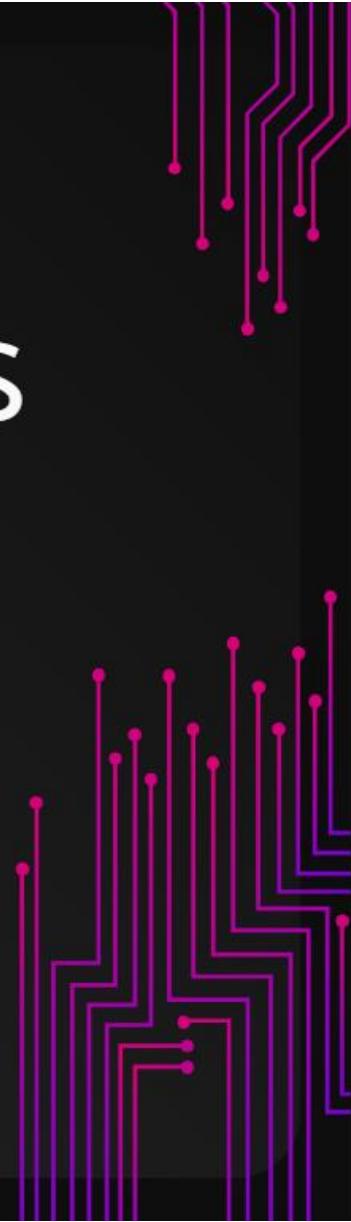
```
let answer: string = "correct"
if (answer === "correct") {
    console.log("You got it right!");
} else {
    console.log("Sorry, that's not correct.");
}
```



# Evaluating a Number Game

A simple game where the user guesses if  
a number is high, low, or equal to a target  
number

A simple game where the user guesses if a number is high, low, or equal to a target number

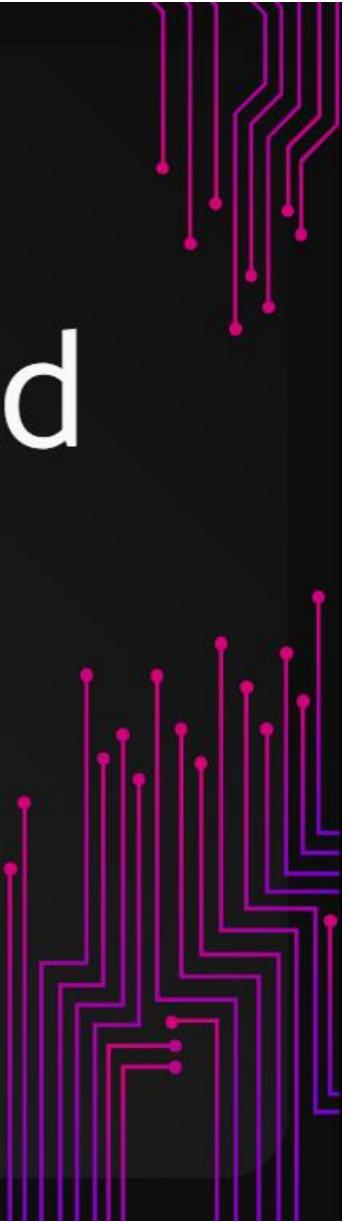


```
let guess: number = 7;
let target: number = 5;
if (guess < target) {
    console.log("Your guess is too low.");
} else if (guess > target) {
    console.log("Your guess is too high.");
} else {
    console.log("You guessed correctly!");
}
```

# Friend Checker Game

A game to check if someone is a friend  
based on their name

A game to check if  
someone is a friend based  
on their name





main.ts > ...

```
1  let isFriend: string = "Ameen"
2  if (isFriend === "Ameen" || isFriend === "Daniyal") {
3      console.log(` ${isFriend} is your friend.`);
4  } else {
5      console.log(` ${isFriend} is not your friend.`);
6 }
```



```
let isFriend: string = "Hamzah"
if (isFriend === "Ameen" || isFriend === "Daniyal") {
    console.log(` ${isFriend} is your friend.`);
} else {
    console.log(` ${isFriend} is not your friend.`);
}
```

```
import inquirer from "inquirer";
let isFriend = await inquirer.prompt([
  name: "name",
  type: "string",
  message: "Enter your friend name:"
]);
if (isFriend.name === "Ameen" || isFriend.name === "Daniyal") {
  console.log(` ${isFriend.name} is your friend.`);
} else {
  console.log(` ${isFriend.name} is not your friend.`);
}
```

Administrator: C:\WINDOWS

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

```
D:\code\typescript\classcode>node main.js
? Enter your friend name: Ameen
Ameen is your friend.
```

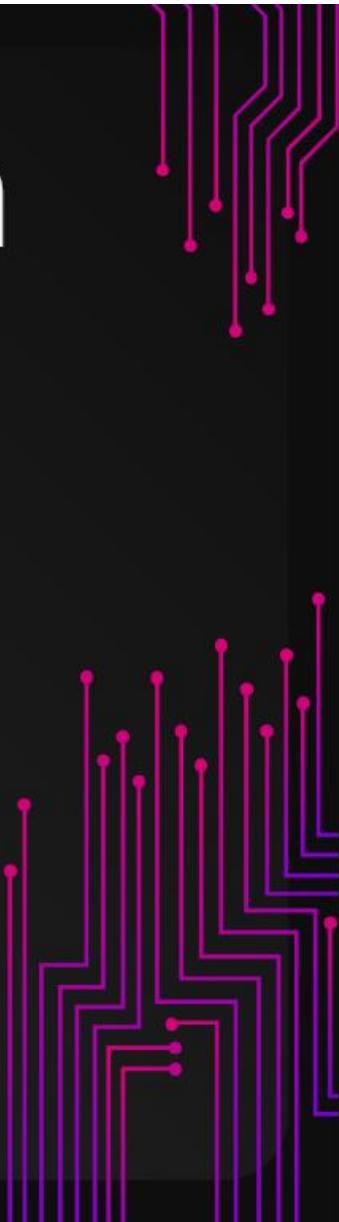
```
D:\code\typescript\classcode>node main.js
? Enter your friend name: Hamzah
Hamzah is not your friend.
```

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

# Rock Paper Scissors Game

A simple implementation of the Rock,  
Paper, Scissors game

# A simple implementation of the Rock, Paper, Scissors game



```
let player1: string = "Rock"
let player2: string = "Scissors"
if (player1 === player2) {
    console.log("It's a tie!");
} else if ((player1 === "Rock" && player2 === "Scissors") ||
    (player1 === "Scissors" && player2 === "Paper") ||
    (player1 === "Paper" && player2 === "Rock")) {
    console.log("Player 1 wins!");
} else {
    console.log("Player 2 wins!");
}
```

# Home Work

## Create a Calculator

Create a calculator using condition statement, operator, template literals, inquirer, and chowk.

Calculator buttons = + - \* / \*\* %

# Functions

Functions are the fundamental building block of  
any application in JavaScript

# **Basic Functions**





```
function halfFryEgg() {  
    let cocked = 1 + 1.5 + 2;  
    // Egg + Butter + Salt  
    console.log(cocked)  
}  
halfFryEgg() // Invoking the function
```



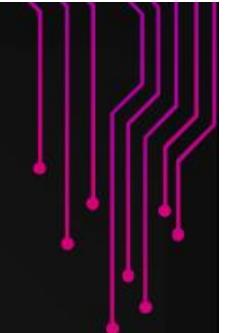
# Returning Function Values

Functions can return the result of their operations, like how a recipe yields a Half Fry Egg.

```
function halfFryEgg(): number {  
    let cocked = 1 + 1.5 + 2;  
    // Egg + Butter + Salt  
    return cocked  
}  
// Invoking the function  
let response: number = halfFryEgg()  
console.log(response)
```

# Parameters and Arguments

- Think of a function as a recipe.
- Each recipe (function) has a name
- Ingredients (parameters),
- Steps to follow (code inside the function).



Parameters are like Ingredients in a  
recipe, such as "butter" and "salt"

When you actually make the Half  
Fry Egg, the butter and salt you use  
are the arguments (actual values).



```
function halfFryEgg(egg: number, butter: number, salt: number): number
    return egg + butter + salt;
}

let response: number = halfFryEgg(1, 1.5, 2)
console.log(response)
```





```
function addNumbers(a: number, b: number): number {  
    return a + b;  
}  
let response = addNumbers(5, 3) // Invoking the function  
console.log(response);
```



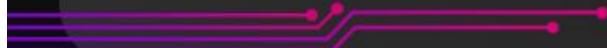
```
function calculateArea(width: number, height: number): number {  
    return width * height;  
}  
  
let response: number = calculateArea(100, 50)  
console.log(response)
```

# Default Parameters

If you forget an ingredient, you have a backup.



```
function halfFryEgg(  
    egg: number = 1, butter: number = 1.5, salt: number = 2  
): number {  
    return egg + butter + salt;  
}  
let response: number = halfFryEgg()  
console.log(response)
```





```
function halfFryEgg(  
    egg: number = 1, butter: number = 1.5, salt: number = 2  
): number {  
    return egg + butter + salt;  
}  
let response: number = halfFryEgg(1, 1.5, 3)  
console.log(response)
```



# Rest Parameters

Accepting an unknown number of ingredients



```
function halfFryEgg(egg: number = 1, ...ingredients: number[]) {  
    console.log(egg)  
    console.log(ingredients)  
}  
halfFryEgg(1, 1.5, 2, 5)
```



# Spread Operator

Like spreading out ingredients on the counter



```
function halfFryEgg(egg: number = 1, ...ingredients: number[]) {  
    console.log(egg)  
    console.log(...ingredients)  
}  
halfFryEgg(1, 1.5, 2, 5)
```

**What happens if I  
call function  
twice?**





# Arrow Functions

A shorthand way of writing a recipe



```
let halfFryEgg = () => 1 + 1.5 + 3;  
| | | | | // egg + butter + salt  
let response: number = halfFryEgg()  
console.log(response)
```





```
let halfFryEgg = (egg: number, butter: number, salt: number): number => (
    egg + butter + salt
)

let response: number = halfFryEgg(1, 1.5, 2)
console.log(response)
```





```
let halfFryEgg = () => { return 1 + 1.5 + 3 };  
                           // egg + butter + salt  
let response: number = halfFryEgg()  
console.log(response)
```





```
let halfFryEgg = () => {  
    // egg + butter + salt  
    let cocked = 1 + 1.5 + 3;  
    return cocked  
};  
let response: number = halfFryEgg()  
console.log(response)
```





```
let halfFryEgg = (egg: number, butter: number, salt: number): number => {  
    let cocked = egg + butter + salt;  
    return cocked  
};  
let response: number = halfFryEgg(1, 1.5, 2)  
console.log(response)
```



# Variable Scope

The current context of code, which determines  
the accessibility of variables to JavaScript

# Global Variables

Ingredients available in your entire kitchen

# Local Variables

Ingredients used within a recipe



```
let globalVar = "Accessible everywhere";
function showExample() {
  let localVar = "Accessible only inside this function";
  console.log(globalVar); // Works
}
console.log(localVar); // Error: localVar is not defined
```



# 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)



# **Anonymous Function**

main.ts > ...

```
1 let halfFryEgg = function () {  
2     let cocked = 1 + 1.5 + 2;  
3     console.log(cocked);  
4 }  
5 halfFryEgg()  
6
```

# Immediately Invoked Function Expression (IIFE)

Think of it as a recipe you make right after reading it, without planning to make it again.

```
(function() {  
    console.log("Runs immediately");  
})();
```



# Recursive Functions

A recipe that says to repeat a step (the function)  
until the dish is done.

```
function countdown(number: number): void {  
    if (number <= 0) {  
        console.log("Done!");  
        return;  
    }  
    console.log(number);  
    countdown(number - 1);  
}  
countdown(5);|
```

Administrator: C:\WINDOWS' X +

D:\code\typescript\classcode>tsc

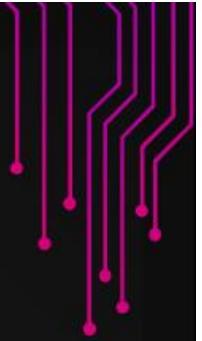
D:\code\typescript\classcode>node main.js

5  
4  
3  
2  
1

Done!

# Create a Recursive Function

Creating a function that calls itself  
to solve a problem.



```
function factorial(n: number): number {  
    if (n === 0) {  
        return 1;  
    }  
    return n * factorial(n - 1);  
}  
let response = factorial(5);  
console.log(response);
```

Administrator: C:\WINDOWS

D:\code\typescript\classcode>tsc

D:\code\typescript\classcode>node main.js  
120

# Nested Functions

A recipe within a recipe

```
function outerFunction() {  
    const innerFunction = function() {  
        console.log("Hello from inside!");  
    }  
    innerFunction();  
}  
outerFunction();
```



# Function Callbacks

A step in a recipe that says to perform another specific recipe now.

```
function processUserInput(callback:any) {  
    let name = "Sir Ameen Alam";  
    callback(name);  
}  
processUserInput(function(name:string) {  
    console.log("Hello, " + name);  
});
```



```
function processUserInput(callback: (name: string) => void) {  
    let name = "Sir Ameen Alam";  
    callback(name);  
}  
processUserInput(function (name: string) {  
    console.log("Hello, " + name);  
});
```



# **Set Timeout Order**

Scheduling tasks like setting a timer for baking.

```
setTimeout(() => {
  console.log("Cake is ready!");
}, 2000); // Waits 2 seconds
```



# Self-Check Quiz

Consider creating a function that quizzes the user and checks the answer.

```
import inquirer from "inquirer";
let input1 = await inquirer.prompt([
    name: "userAnswer",
    type: "string",
    message: "What is the capital of France?"
]);
function quiz(question: string, correctAnswer: string) {
    if (input1.userAnswer.toLowerCase() === correctAnswer.toLowerCase())
        console.log("Correct!");
    } else {
        console.log("Wrong answer. Try again.");
    }
}
quiz("What is the capital of France?", "Paris");
```



Administrator: C:\WINDOWS

+ | -



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

```
D:\code\typescript\classcode>node main.js
```

? What is the capital of France? Paris

Correct!

```
D:\code\typescript\classcode>node main.js
```

? What is the capital of France? London

Wrong answer. Try again.

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

# **Homework Assignments**

## 1. Basic Function Creation

Create a function named calculateProduct that takes two parameters, multiplies them together, and returns the result.

```
const result = calculateProduct(5, 10);
console.log(result); // Should print 50
```

## 2. Using Default Parameters

Define a function greet that takes two parameters: name and greeting, where greeting has a default value of "Hello". The function should return a greeting message.

```
console.log(greet("Ameen")); // Should print "Hello,  
Ameen!"
```

```
console.log(greet("Zia", "Hi")); // Should print "Hi,  
Zia!"
```

### 3. Arrow Function Conversion

Convert the following traditional function into an arrow function:

```
function add(a: number, b: number): number {  
    return a + b;  
}
```

## 4. Implementing a Rest Parameter

Write a function `sumAll` that uses a rest parameter to take any number of arguments and returns their sum.

```
console.log(sumAll(1, 2, 3)); // Should print 6
```

```
console.log(sumAll(10, 20, 30, 40, 50)); //  
Should print 150
```

## 5. Function Returning Another Function

Create a function `multiplier` that takes a number as its argument and returns another function. The returned function should take a single number as its argument and return the product of its argument and the argument of the first function.

```
const triple = multiplier(3);
```

```
console.log(triple(5)); // Should print 15
```

## 6. Recursive Function - Factorial

Write a recursive function to calculate the factorial of a number. The factorial of a number  $n$  is the product of all positive integers less than or equal to  $n$ . For example, the factorial of 5 ( $5!$ ) is  $5 * 4 * 3 * 2 * 1 = 120$ .

```
console.log(factorial(5)); // Should print 120
```

## 7. Nested Functions - Scoping

Write a function that contains two nested functions. The outer function should accept a parameter x, and its nested functions should increment and then triple x. The outer function should return the result of the tripled value after incrementing.

```
console.log(outerFunction(4)); // Should first increment  
4 to 5, then triple 5 to 15, and finally return 15
```

## 8. Anonymous Function and Callbacks

Create an anonymous function that takes an array of numbers and a callback function. The anonymous function should apply the callback function to each element of the array and return a new array with the results.

```
const numbers = [1, 2, 3];
```

```
const doubledNumbers = anonymousFunction(numbers, (x)  
=> x * 2);
```

```
console.log(doubledNumbers); // Should print [2, 4, 6]
```

## 9. Set Timeout Exercise

Use `setTimeout` within a function to simulate a delay in processing (e.g., retrieving data from a database). The function should accept a callback and invoke it after a delay of 2 seconds.

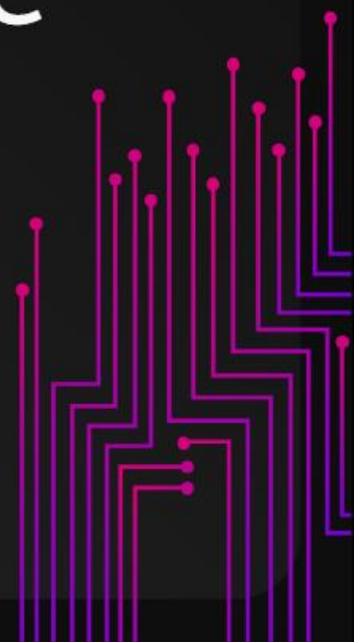
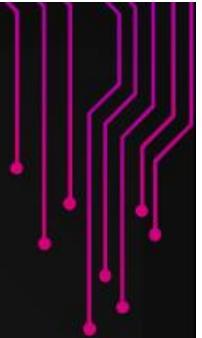
```
simulateDelay(() => console.log("Data  
retrieved"));
```

# Submission Instructions

For each exercise, write your TypeScript code in a clear and organized manner. Comment on your code to explain your logic and the steps you've taken to solve each problem. After completing the exercises, review your solutions to ensure they meet the requirements and test them to ensure they work as expected.

# **Basic Objects in TypeScript**

Think of an object as a file cabinet where each drawer is labeled and contains specific information.



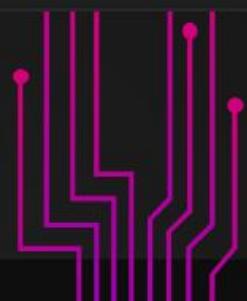
In TypeScript, an object is a collection of key-value pairs, where each key (also known as a property) is associated with a value. This structure is similar to the file cabinet, where each drawer's label is a key, and the contents of the drawer are the value.





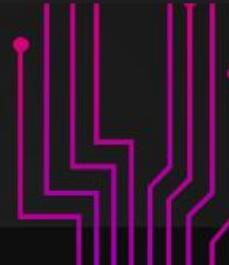
```
// Defining a basic object in TypeScript
let person = {
    name: "Ameen Alam"
};

// Accessing properties of the object
console.log(person.name); // Output: Ameen Alam
```





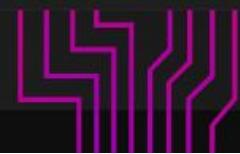
```
// Defining a basic object in TypeScript
let person: { name: string; } = {
    name: "Ameen Alam",
};
console.log(person.name);
// Output: Ameen Alam
```





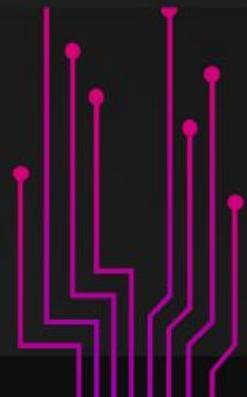
```
// Defining a basic object in TypeScript
let person: { name: string; age: number; address: string;
  name: "Ameen Alam",
  age: 24,
  address: "123 ABC Street"
};

// Accessing properties of the object
console.log(person.name); // Output: Ameen Alam
console.log(person.age); // Output: 24
console.log(person.address); // Output: 123 ABC Street
```





```
// Modifying an object's properties
person.age = 18;
console.log(person.age); // Output: 18
```



```
// Modifying an object's properties
```

```
person.age = 18;
```

```
console.log(person.age); // Output: 18
```

```
// Adding a new property
```

```
person.email = "alice@example.com";
```

```
// Error: Property 'email' does not exist on type...
```

# Type Alias in TypeScript

Imagine you have a favorite smoothie recipe that includes a specific combination of ingredients: bananas, strawberries, and almond milk. Instead of listing these ingredients every time you talk about your favorite smoothie, you simply start calling it "MySmoothie." Here, "MySmoothie" is a nickname or alias for the combination of bananas, strawberries, and almond milk.



# Type Aliases: TypeScript Code

In TypeScript, a type alias allows you to create a new name (alias) for an existing type or a complex type. This can make complex types easier to work with and can be especially handy when dealing with objects or function types that are used in multiple places.

```
type User = {  
    name: string;  
    age: number;  
    hasPet: boolean;  
};  
// Now you can use the 'User' type alias to define objects  
let user1: User = {  
    name: "Ameen Alam",  
    age: 24,  
    hasPet: true  
};
```



```
// Another example with a complex type for a function
type Operation = (x: number, y: number) => number;
const add: Operation = (x, y) => x + y;
const subtract: Operation = (x, y) => x - y;
console.log(add(5, 3)); // 8
console.log(subtract(10, 4)); // 6
```

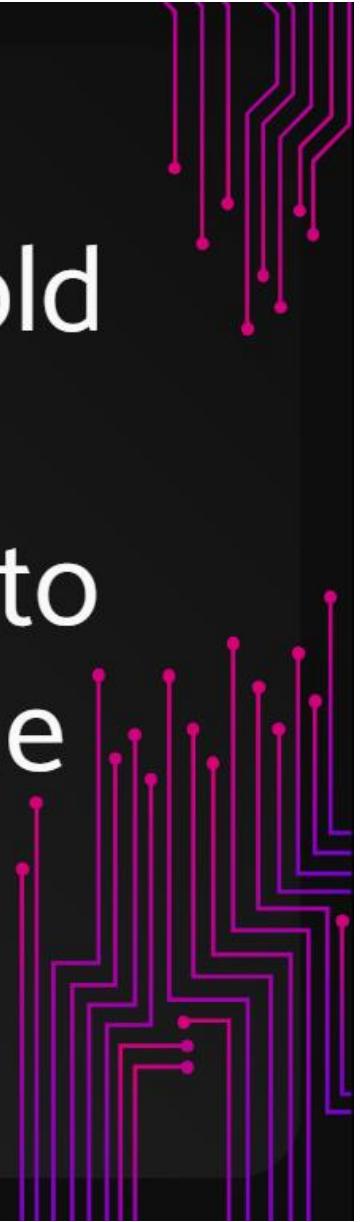


# Type Literal

Imagine you go to a coffee shop where you can only order drinks in three specific sizes: "Small", "Medium", or "Large". You cannot order a "Mega" or "Mini" because the shop simply does not recognize those sizes. In this scenario, the drink sizes offered by the shop are like type literals in programming—specific, predefined values that are acceptable.



TypeScript allows you to specify exact values that a variable can hold using type literals. This can be particularly useful when you want to restrict a variable to a specific value or a set of values.



||||

```
let drinkSize: "Small" | "Medium" | "Large";  
drinkSize = "Medium"; // Valid  
drinkSize = "Small"; // Valid
```



```
let drinkSize: "Small" | "Medium" | "Large";  
drinkSize = "Medium"; // Valid  
drinkSize = "Small"; // Valid  
drinkSize = "Mega";  
// Error: Type  
//'"Mega"' is not assignable to type '"Small" | "Medium" | "Large"'.
```



main.ts

```
Type '"Mega"' is not assignable to type '"Small" | "Medium" |
| "Large"'. ts(2322)
```

```
1 let drinkSize: "Small" | "Medium" | "Large"
2
```

```
3 View Problem (Alt+F8) No quick fixes available
```

```
4 drinkSize = "Mega";
```

```
5 // Error: Type
```

```
6 //'"Mega"' is not assignable to type '"Small" | "Medium" | "Large"'.
```

```
7
```



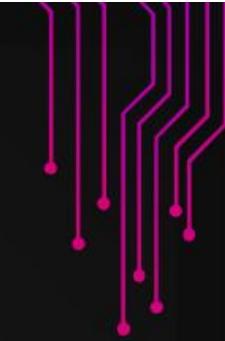
# Type Unions

Unions: A Real-World Analogy

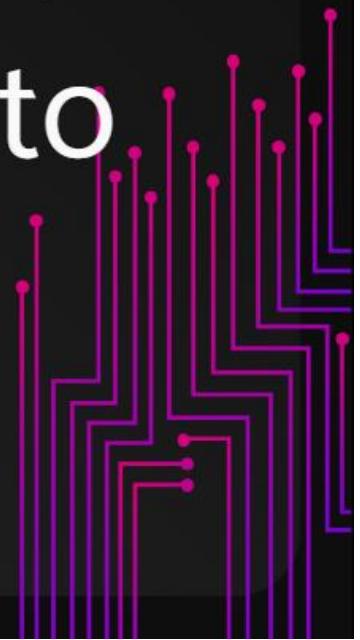
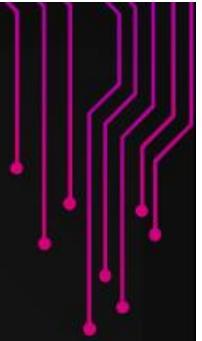
Imagine you're packing for a vacation where you plan to spend time both at the beach and hiking in the mountains. When packing your bag, you decide to bring items that could be useful for either activity.



So, you pack sunglasses (useful at the beach), sunscreen (useful at both beach and mountains), and a water bottle (useful in the mountains).



This bag represents a Union type: it can contain items useful for the beach, the mountains, or both, giving you the flexibility to use it in different scenarios.



# Union Types: TypeScript Code

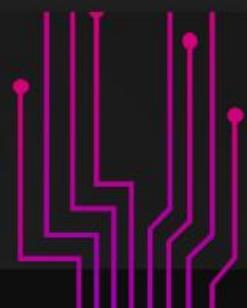
In TypeScript, a union type allows a variable to hold values of multiple types. It's like saying a variable can be one type OR another type.



```
let mixedBag: string | number;

mixedBag = "Sunscreen"; // OK
mixedBag = 30; // OK, maybe representing the SPF of the sunscreen

// mixedBag = true;
// Error: Type 'boolean' is not assignable to type 'string | number'.
```



# Type Intersection

Intersection: A Real-World Analogy

Now, think about a multi-tool that you're bringing along on your trip. This multi-tool includes a knife for hiking and a bottle opener for the beach. This tool is not useful just for the beach or just for the mountains; it must serve a purpose in both scenarios simultaneously.



This represents an Intersection type: it combines features or functionalities from multiple sources into one entity, ensuring it meets several criteria at once.



# Intersection Types: TypeScript Code

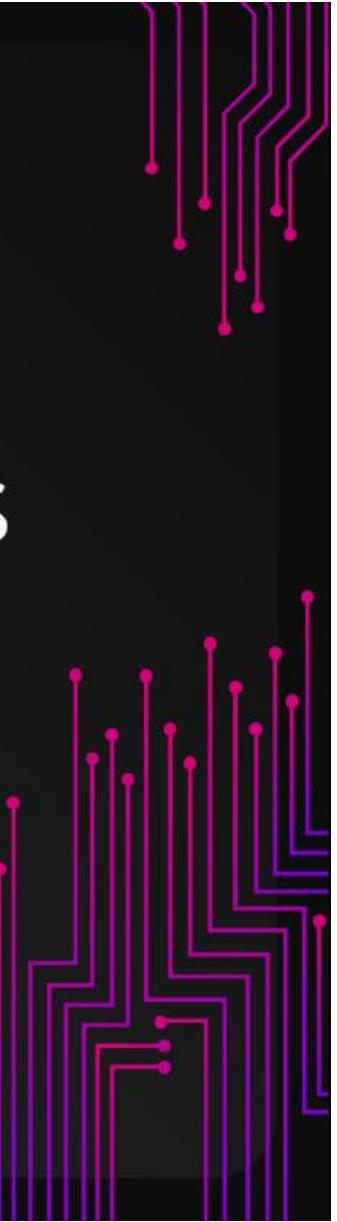
Intersection types allow you to combine multiple types into one.

This means a variable of an intersection type will have all the properties of all the types it intersects.

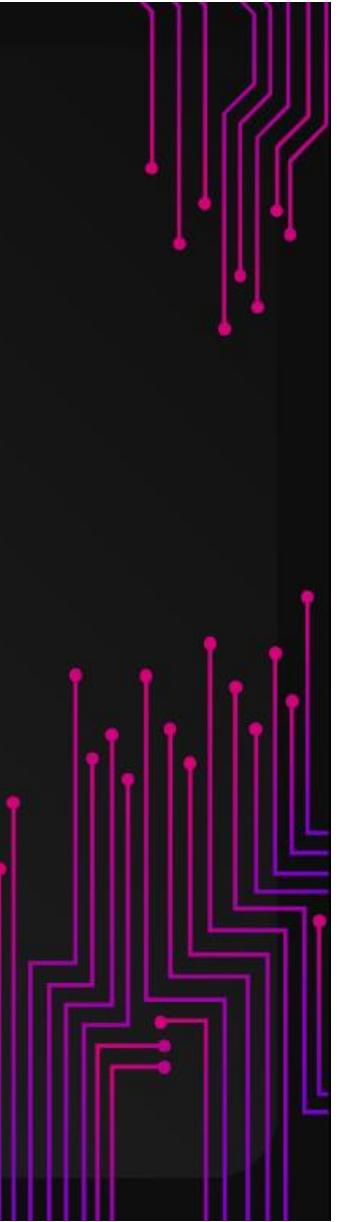
```
type BeachGear = {  
    sunscreen: boolean;  
    towel: boolean;  
}  
type MountainGear = {  
    waterBottle: boolean;  
    map: boolean;  
}  
type AdventureGear = BeachGear & MountainGear;  
let myGear: AdventureGear = {  
    sunscreen: true,  
    towel: true,  
    waterBottle: true,  
    map: true  
};|
```

# Arrays and Their Properties

Imagine an array as a row of mailboxes along a street. Each mailbox is assigned a number (its index) and contains mail (values). Just as you can add, remove, or check mail in these mailboxes



You can do the same with values in an array. The properties of an array, such as its length, help you understand how many mailboxes are in the row or how many items are in the array.



```
let fruits = ["Apple", "Banana", "Cherry"];
console.log(fruits.length); // 3
console.log(fruits[1]);
// "Banana" - Accessing the second element (index starts at 0)
```

```
let fruits: string[] = ["Apple", "Banana", "Cherry"];
console.log(fruits.length); // 3
console.log(fruits[1]);
// "Banana" - Accessing the second element (index starts at 0)
```

# Array Methods

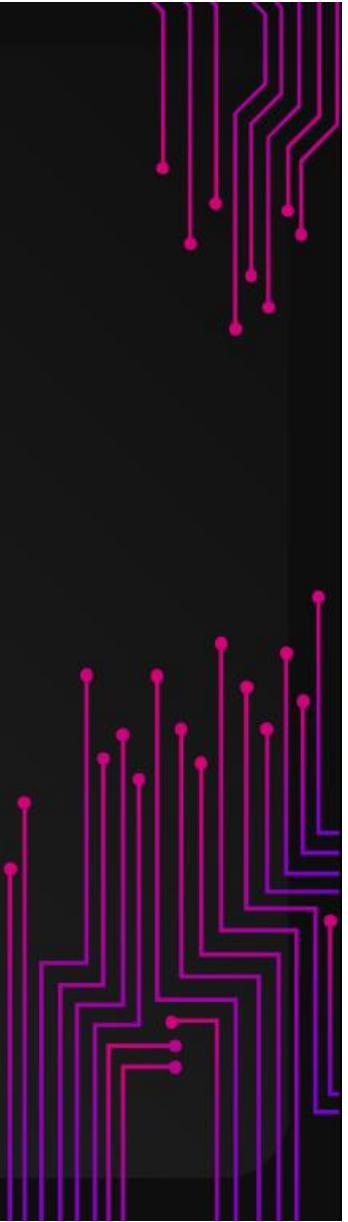
Arrays come with a toolbox of methods to manipulate their contents, like adding or removing items.

```
let colors: string[] = ["Red", "Green", "Blue"];  
  
colors.push("Yellow"); // Adds "Yellow" to the end  
colors.pop(); // Removes the last element ("Yellow")  
colors.shift(); // Removes the first element ("Red")  
colors.unshift("Purple"); // Adds "Purple" to the start  
  
console.log(colors); // ["Purple", "Green", "Blue"]
```

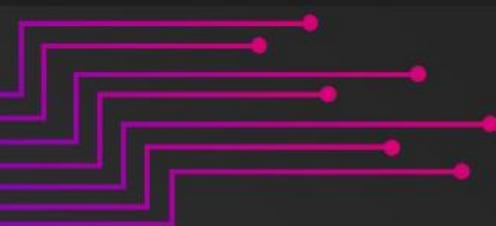


# Multidimensional Arrays

Think of a multidimensional array like a chest of drawers, where each drawer contains another set of compartments. For instance, one drawer might be for socks, and within this drawer, each compartment has pairs of socks of different colors.

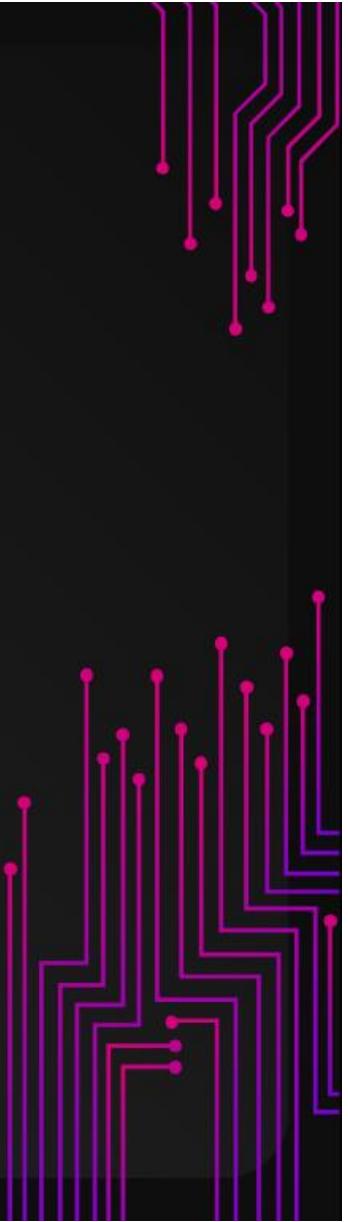


```
let matrix: number[][] = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];
console.log(matrix[1][2]);
// 6 - Accessing the third element in the second array
```



# **Working with Objects and Arrays**

Imagine a library system where each book is represented by a card that contains details about the book (like an object) such as its title, author, and ISBN. All these cards are then organized into a card catalog (an array), allowing you to find any book by looking through the cards.



```
type Book = {  
    title: string;  
    author: string;  
    isbn: string;  
};  
let library: Book[] = [  
    { title: "The Hobbit", author: "J.R.R. Tolkien", isbn: "123456789"  
    { title: "1984", author: "George Orwell", isbn: "987654321" }  
];
```

```
type Book = {  
    title: string;  
    author: string;  
    isbn: string;  
};  
let library: Book[] = [  
    { title: "The Hobbit", author: "J.R.R. Tolkien", isbn: "123456789" },  
    { title: "1984", author: "George Orwell", isbn: "987654321" }  
];  
// Adding a new book to the array  
library.push({  
    title: "The Catcher in the Rye",  
    author: "J.D. Salinger",  
    isbn: "1112131415"  
});
```



```
// Finding a book by author
let foundBook = library.find(book => book.author === "George Orwell");
console.log(foundBook);
// Output { title: "1984", author: "George Orwell", isbn: "987654321" }
```



# **Homework Assignments**

## 1. Basic Array Operations

Create an array called fruits that contains the names of four different fruits. Perform the following operations:

- Add a new fruit to the end of the array.
- Remove the first fruit from the array.
- Add a new fruit to the beginning of the array.
- Find the index of a fruit and remove that fruit using the index.

```
// Example usage and operations on the 'fruits' array
```

## 2. Working with Multidimensional Arrays

Define a  $3 \times 3$  matrix of numbers as a multidimensional array. Write functions to:

- Print the diagonal elements of the matrix.
- Calculate the sum of all elements in the matrix.

```
// Define the matrix and implement the functions
```

### 3. Filtering and Searching in Arrays of Objects

Given an array of objects where each object represents a book with properties title, author, and yearPublished, write functions to:

- Filter books published after the year 2000.
- Search for books by a specific author.

```
// Define the array of book objects and implement the  
functions
```

## 4. Using Array Methods

Create an array of numbers. Using array methods, perform the following tasks:

- Create a new array with the squares of each number.
- Filter out all numbers greater than 50.
- Use the reduce method to find the sum of all numbers in the array.

```
// Example usage of array methods for manipulation
```

## 5. Advanced: Working with Nested Arrays and Objects

Consider an array of objects where each object represents a student. Each student object has a name, id, and an array of grades. Write a function that calculates the average grade for each student and adds it as a new property `averageGrade` to each student object.

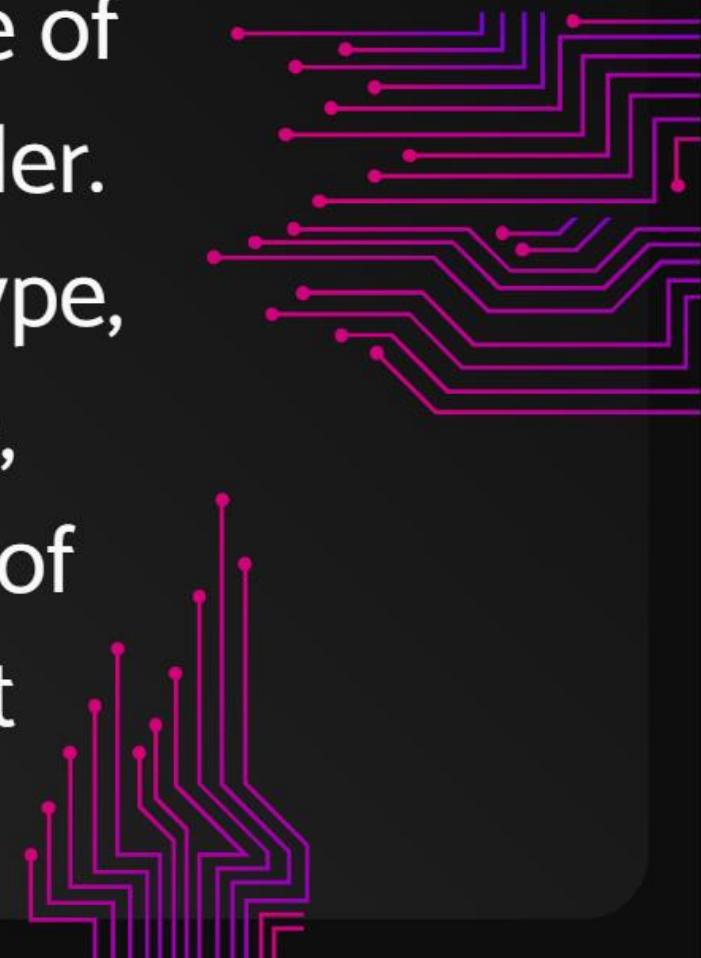
```
// Define the array of student objects and implement  
the function
```

# Submission Instructions

For each exercise, write your TypeScript code in a clear and organized manner. Comment on your code to explain your logic and the steps you've taken to solve each problem. After completing the exercises, review your solutions to ensure they meet the requirements and test them to ensure they work as expected.

# Tuples

Imagine you're ordering a coffee and you need to specify both the type of coffee and the size in a single order. This pair of information (coffee type, size) can be thought of as a tuple, which is simply a fixed collection of elements that may be of different types.





```
let coffeeOrder: [string, string] = ["Cappuccino", "Medium"];
| | | | | | // Tuple: [Coffee Type, Size]
```



# Enums

```
enum CoffeeType {  
    Espresso,  
    Latte,  
    Cappuccino,  
    Americano  
}
```

```
let myCoffee: CoffeeType = CoffeeType.Latte;  
console.log(myCoffee);
```



||||

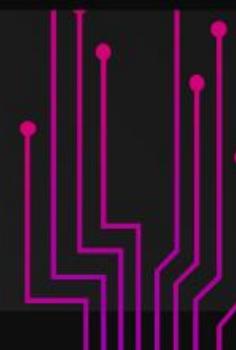
Administrator: C:\WINDOWS'

+ | ▾

D:\code\typescript\classcode>tsc

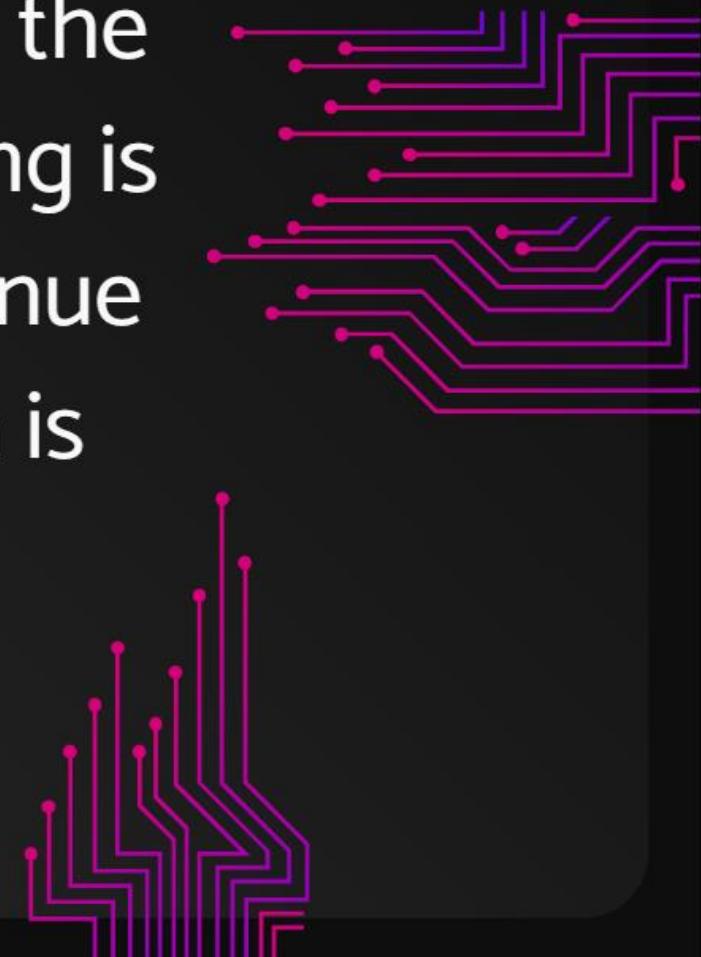
D:\code\typescript\classcode>node main.js  
1

D:\code\typescript\classcode>



# While Loop

Imagine you're waiting for a bus. You keep checking every minute until the bus arrives. This repeated checking is like a while loop, where you continue to do something until a condition is met.



```
let minutesUntilBusArrives = 5;

while (minutesUntilBusArrives > 0) {
  console.log(`Bus arrives in ${minutesUntilBusArrives} minutes.`);
  minutesUntilBusArrives--;
}
```





Administrator: C:\WINDOWS

+ ▾

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

```
D:\code\typescript\classcode>node main.js
```

```
Bus arrives in 5 minutes.
```

```
Bus arrives in 4 minutes.
```

```
Bus arrives in 3 minutes.
```

```
Bus arrives in 2 minutes.
```

```
Bus arrives in 1 minutes.
```

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



# **Do-While Loop**

Even if you arrive just as the bus pulls up, you still check at least once to see if the bus is there. A do-while loop ensures the action is performed at least once, and then it continues if the condition is true.



```
let minutesUntilBusArrives = 5;

do {
  console.log("Checking for the bus...");
  /* Assume checkBusArrival() decreases minutesUntilBusArrives
   and returns false when the bus arrives */
} while (minutesUntilBusArrives > 0);
```



D:\code\typescript\classcode>tsc

D:\code\typescript\classcode>node main.js

Checking for the bus...

**Checking for the bus...**

Checking for the bus...

Checking for the bus...

```
let minutesUntilBusArrives = 5;

do {
    console.log("Checking for the bus...");
    minutesUntilBusArrives--;
    /* Assume checkBusArrival() decreases minutesUntilBusArrives
       and returns false when the bus arrives */
} while (minutesUntilBusArrives > 0);
```



Administrator: C:\WINDOWS' + ▾

D:\code\typescript\classcode>tsc

D:\code\typescript\classcode>node main.js

Checking for the bus...

D:\code\typescript\classcode>



```
let minutesUntilBusArrives = 5;
function checkBusArrival() {
    minutesUntilBusArrives--;
    if (minutesUntilBusArrives != 0) {
        return "Checking for the bus..."
    } else { return false }
}
do {
    let response = checkBusArrival()
    console.log(response)
    /* Assume checkBusArrival() decreases minutesUntilBusArrives
       and returns false when the bus arrives */
} while (minutesUntilBusArrives > 0);
```





Administrator: C:\WINDOWS' X

+ ▾

D:\code\typescript\classcode>tsc

D:\code\typescript\classcode>node main.js

Checking for the bus...

Checking for the bus...

Checking for the bus...

Checking for the bus...

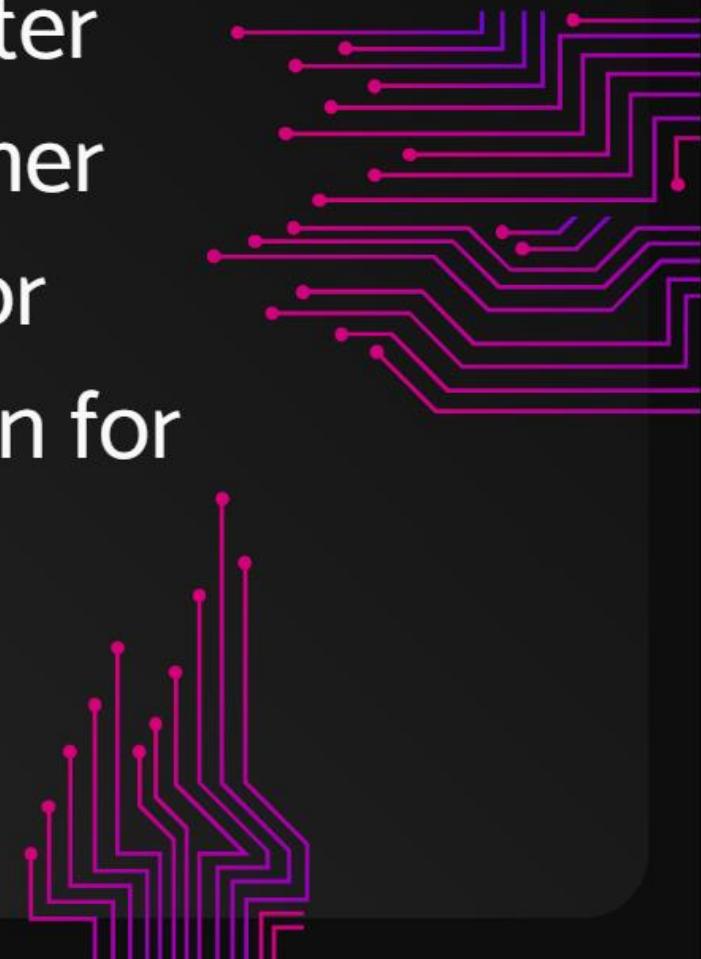
false

D:\code\typescript\classcode>



# For Loop

Suppose you're watering plants in a row. You start at one end and water each plant until you reach the other end. This process is similar to a for loop, where you perform an action for each item in a sequence.



```
for (let i = 0; i < 5; i++) {  
  console.log(`Watering plant ${i + 1}`);  
}  
|
```





Administrator: C:\WINDOWS

+ ▾

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

```
D:\code\typescript\classcode>node main.js
```

```
Watering plant 1
```

```
Watering plant 2
```

```
Watering plant 3
```

```
Watering plant 4
```

```
Watering plant 5
```

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

# For-In Loop

Imagine you have a keyring with keys for different doors. You look at each key on the keyring to find the one you need. A for-in loop in programming is like examining each key (property) in an object.



```
let person:any = {  
    name: "Alice", age: 30, city: "Wonderland"  
};
```

```
for (let key in person) {  
    console.log(`${key}: ${person[key]}`);  
}
```





Administrator: C:\WINDOWS' X



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

```
D:\code\typescript\classcode>node main.js
```

```
name: Alice
```

```
age: 30
```

```
city: Wonderland
```

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



# How can I define this object type?

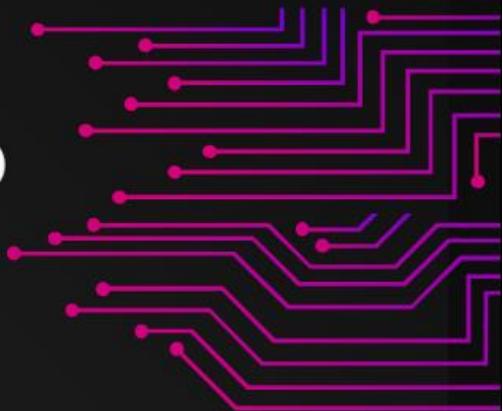
Element implicitly has an 'any' type because expression of type 'string' can't be used to index type '{ name: string; age: number; city: string; }'. ts(7053)

```
1 No index signature with a parameter of type 'string' was
2 found on type '{ name: string; age: number; city: string;
3 }'. ts(7053)
4
5 let person: {
6   name: string;
7   age: number;
8   city: string;
9 }
```

[View Problem \(Alt+F8\)](#) No quick fixes available

# For-Of Loop

Consider having a box of chocolates. You take out each chocolate one by one to see what kind it is. A for-of loop is used to go through each item in an array (or any iterable), similar to checking each chocolate.



```
let flavors = [  
    "Vanilla", "Chocolate", "Strawberry", "Mint"  
];  
  
for (let flavor of flavors) {  
    console.log(flavor);  
}
```





Administrator: C:\WINDOWS

+ ▾

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

```
D:\code\typescript\classcode>node main.js
```

Vanilla

Chocolate

Strawberry

Mint

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

# **Homework Assignments**

## 1. Loop through an Array with For-Of

Create an array of your favorite movies.

Write a function that uses a for-of loop to print each movie to the console.

```
// Example function that prints each  
movie
```

## 2. Enumerate Properties with For-In Loop

Given an object representing a car with properties like make, model, and year, write a function that uses a for-in loop to print each property name and its value.

```
// Example function that prints car properties
```

### 3. Basic For Loop Exercise

Write a function that uses a for loop to print the numbers from 1 to 100. However, for numbers divisible by 3, print "Fizz" instead of the number, and for numbers divisible by 5 (and not 3), print "Buzz". For numbers divisible by both 3 and 5, print "FizzBuzz".

```
// Example FizzBuzz function
```

## 4. Practicing Do-While Loop

Create a function that simulates a simple guessing game. This function should generate a random number between 1 and 10 and then prompt the user to guess the number. Use a do-while loop to keep asking them to guess again until they get it right.

```
// Note: Since TypeScript runs on Node.js for this task,  
consider a pseudo-code or describe the logic, as  
'prompt' is not available in standard Node.js without  
additional packages.
```

## 5. While Loop for a Countdown

Write a function that takes a number as an argument and counts down to zero using a while loop, printing each number to the console.

```
// Example countdown function
```

## 6. Enums for Days of the Week

Define an enum for days of the week. Write a function that takes a day as an argument and returns "Weekend" if it's Saturday or Sunday, and "Weekday" for other days.

```
// Example enum for days and function
```

## 7. Tuples for RGB Colors

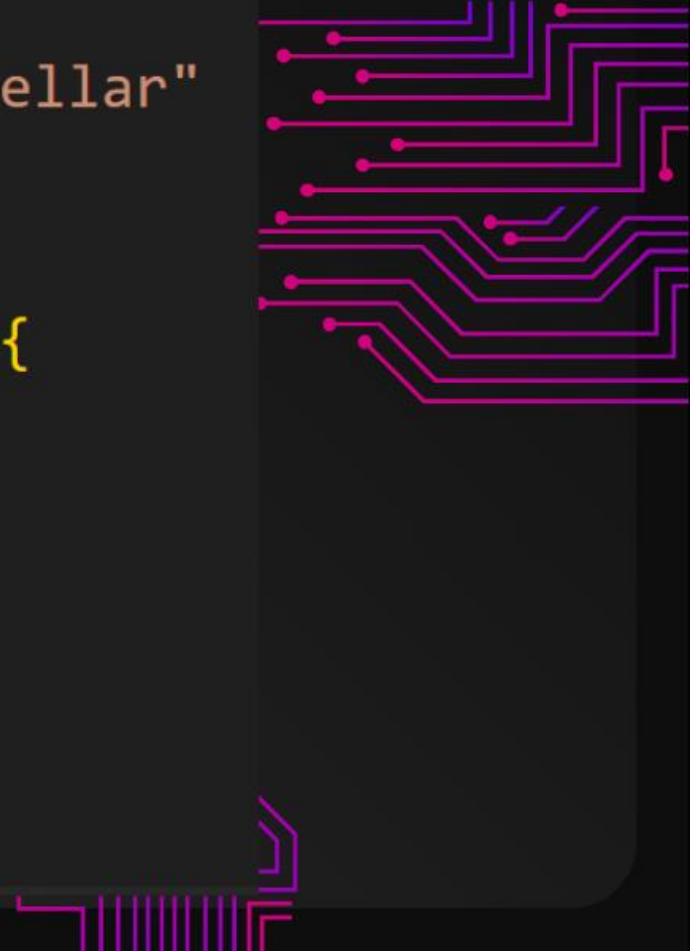
Define a tuple type for RGB color values. Write a function that takes an RGB tuple as an argument and returns a string describing the color.

```
// Example RGB tuple and function
```

# Example Solutions

# 1. Loop through an Array with For-Of

```
const favoriteMovies: string[] = [  
    "Inception", "The Matrix", "Interstellar"  
];  
  
function printMovies(movies: string[]) {  
    for (const movie of movies) {  
        console.log(movie);  
    }  
}  
  
printMovies(favoriteMovies);
```





## 2. Enumerate Properties with For-In Loop

```
const car = {  
    make: "Toyota",  
    model: "Camry",  
    year: 2020  
};  
function printCarDetails(car: { [key: string]: string | number }) {  
    for (const key in car) {  
        console.log(` ${key}: ${car[key]}`);  
    }  
}  
printCarDetails(car);
```



### 3. Basic For Loop Exercise (FizzBuzz)

```
function fizzBuzz() {  
    for (let i = 1; i <= 100; i++) {  
        let output = '';  
        if (i % 3 === 0) output += 'Fizz';  
        if (i % 5 === 0) output += 'Buzz';  
        console.log(output || i);  
    }  
  
    fizzBuzz();
```



# Submission Instructions

For each exercise, ensure your TypeScript code is well-commented to explain your logic. Adhere to best practices such as using `const` for variables that won't change and `let` for those that will. Test your functions to make sure they work as expected. This set of homework assignments is designed to solidify your understanding of TypeScript's control flow mechanisms, enums, and tuples, as well as to encourage thoughtful problem-solving and code organization.

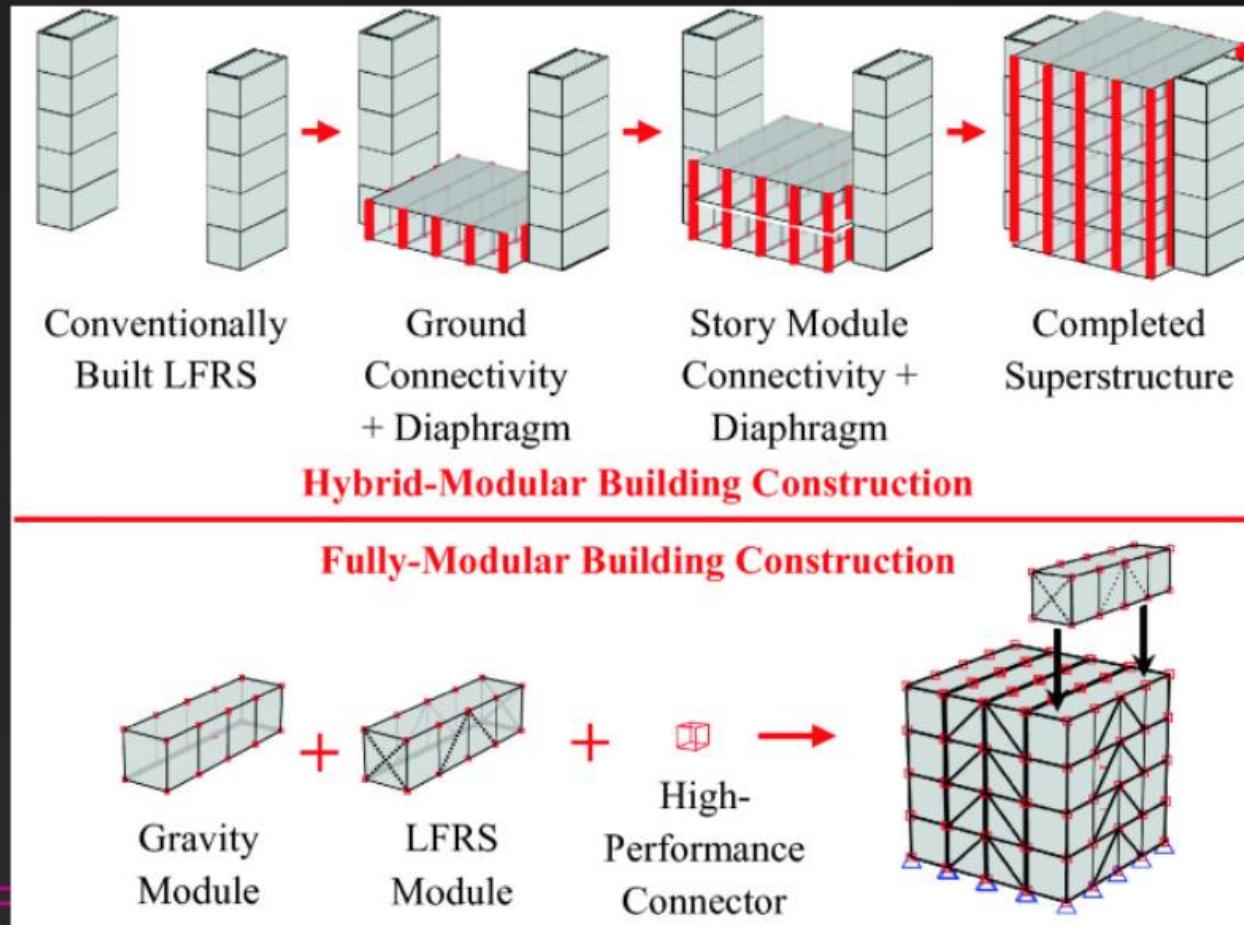
# Modules

Basics of modules, exporting, and  
importing modules.

# Modular Architecture



# Fully-Modular Buildings



# Create a New Project

```
Administrator: C:\WINDOWS' × + ▾
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"
}
```

A screenshot of the Microsoft Visual Studio Code (VS Code) interface. The window title bar at the top displays "classcode [Administrator]". The left sidebar contains icons for File, Explorer, Search, and others, with "EXPLORER" currently selected. The main workspace shows a file tree under "CLASSC..." with files: "TS first.ts" (selected), "TS main.ts", "{} package.json", and "tsconfig.json". The right-hand editor pane displays the contents of "first.ts":

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

The screenshot shows the Visual Studio Code interface with a dark theme. In the Explorer sidebar on the left, there is a tree view under the 'CLASSCODE' folder containing files: 'first.ts', 'main.ts', 'package.json', and 'tsconfig.json'. The 'main.ts' file is selected. The top bar has a search bar with the text 'classcode [Administrator]'. The main editor area displays the 'main.ts' file with the following code:

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



Administrator: C:\WINDOWS' X



```
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

9e7

[Preview](#) [Code](#) | [Blame](#) 10 lines (6 loc) • 358 Bytes

## Modules in TypeScript

[Modules](#)

[ECMAScript Modules in Node.js](#)

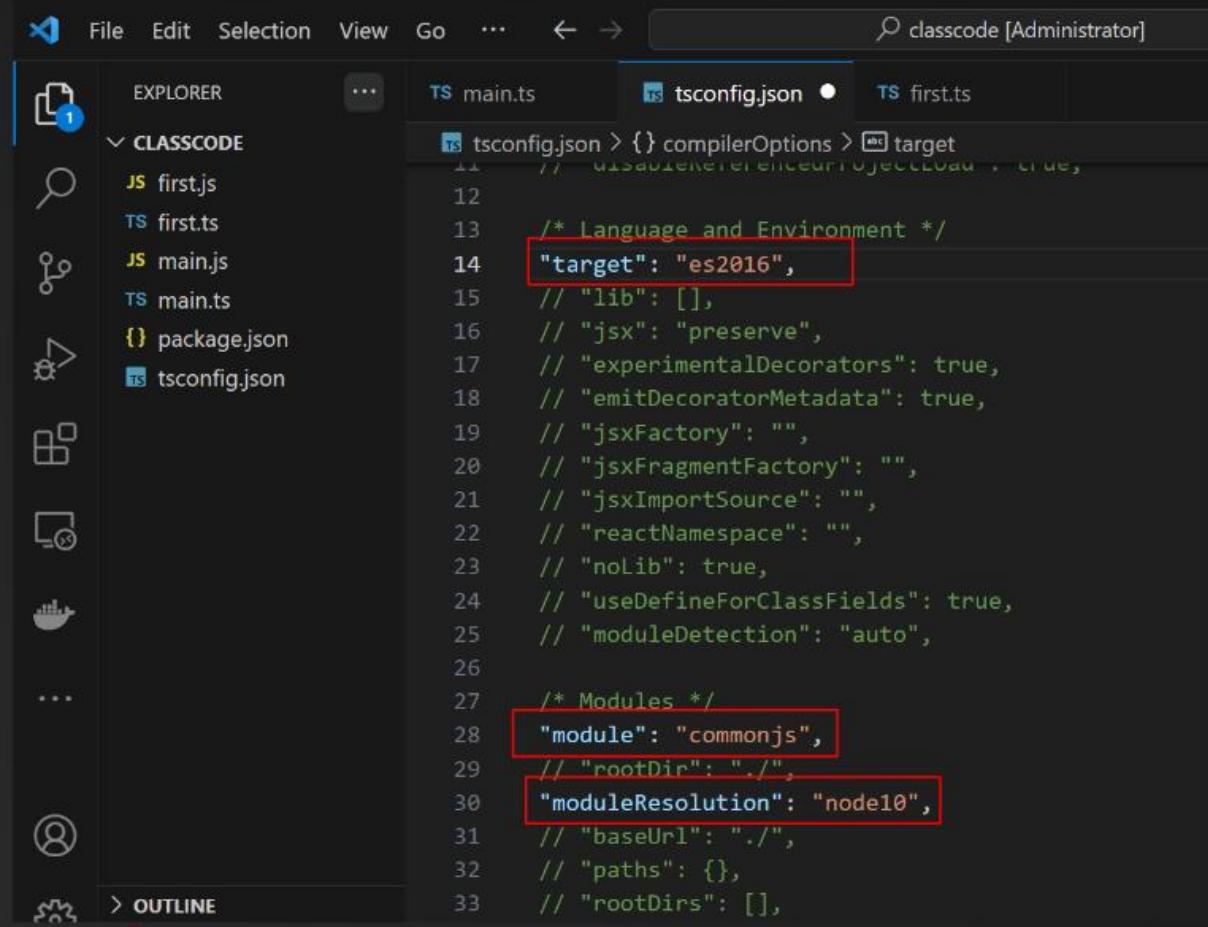
When we transpile this program it runs correctly.

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.

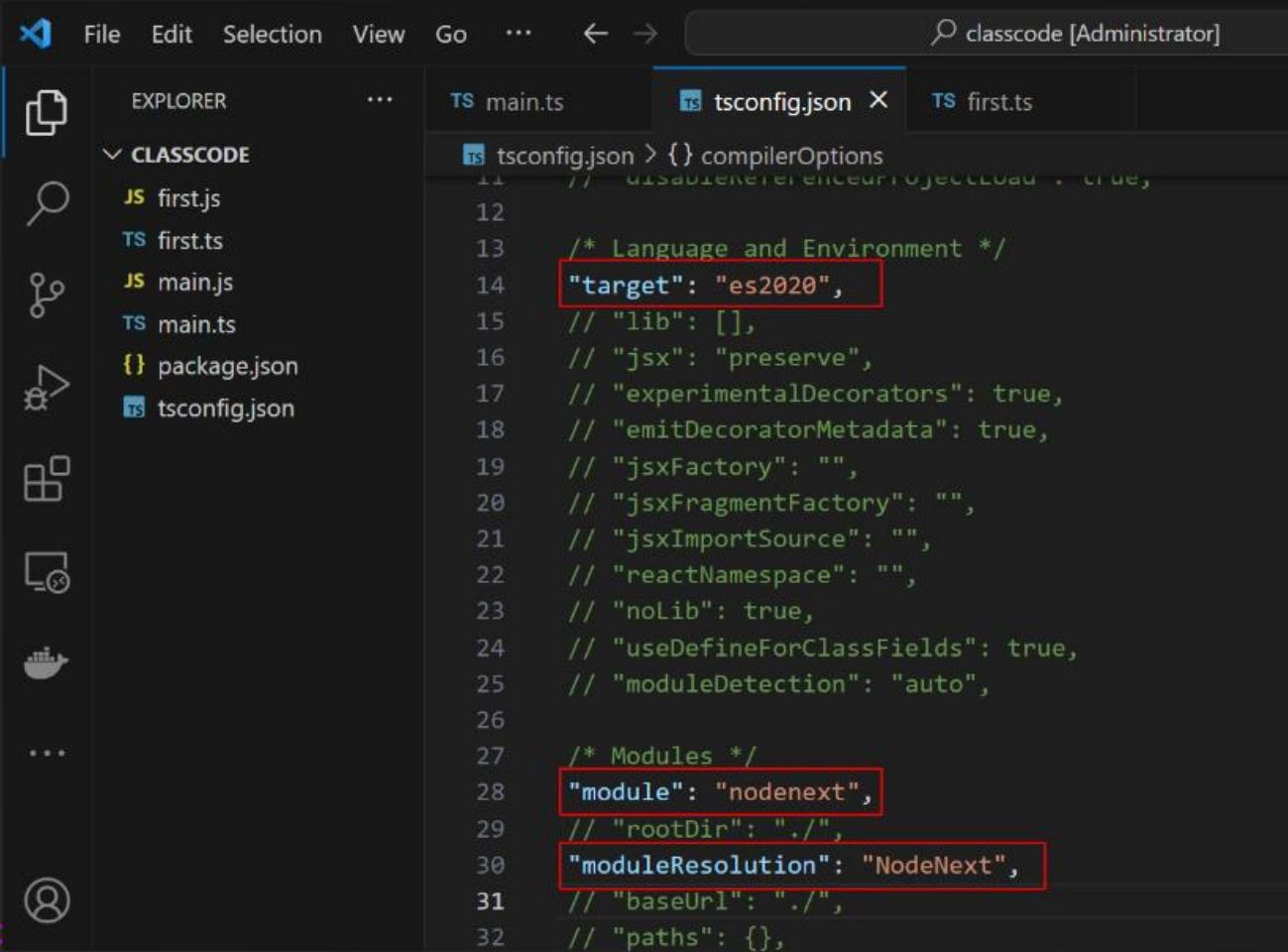
# Default TS Config



The screenshot shows the Visual Studio Code interface with the title bar "classcode [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 the tsconfig.json file. Two specific sections of the JSON object are highlighted with red boxes:

```
 12
 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



A screenshot of the Visual Studio Code interface. The title bar shows "classcode [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 shows the contents of the tsconfig.json file. Several lines of code are highlighted with red boxes: "target": "es2020", "module": "nodenext", and "moduleResolution": "NodeNext".

```
tsconfig.json > {} compilerOptions
  ...
  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": {},
```

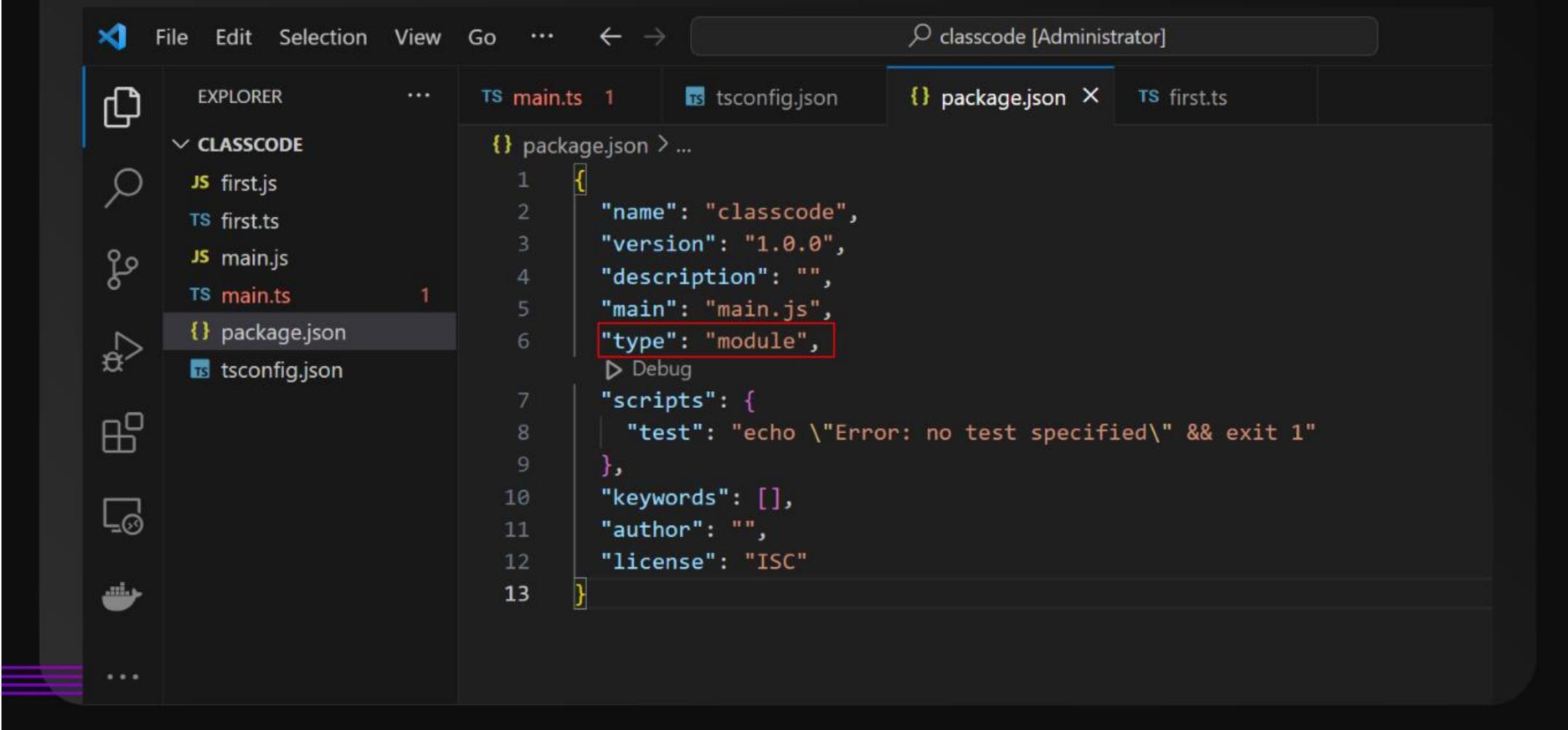
# Default JS Config



A screenshot of the Visual Studio Code (VS Code) interface. The title bar shows the path "classcode [Administrator]". The menu bar includes File, Edit, Selection, View, Go, and a separator (...). The top right features a search bar with a magnifying glass icon. Below the menu is the Explorer sidebar, which displays a tree view of files and folders under "CLASSC...". The "package.json" file is selected, highlighted with a blue border. The main editor area shows the contents of the "package.json" file:

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

# The program we will have make some changes



A screenshot of the Visual Studio Code (VS Code) interface. The title bar shows the path "classcode [Administrator]". The left sidebar has icons for Explorer, Search, Problems, and others. The Explorer view shows a folder named "CLASSCODE" containing files: first.js, first.ts, main.js, main.ts (with a red '1' badge), package.json (selected and highlighted in yellow), and tsconfig.json. The main editor area displays the "package.json" file with the following content:

```
{} package.json > ...
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 / 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 Video: How to Setup Node.js with TypeScript in 2023](#)

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 {b, 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
```

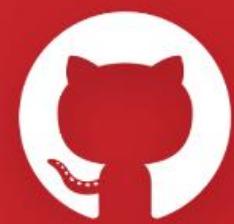


If you give the following command to transpile the code, the js file will not run:

# Importing Third-party Modules

Example with importing the `inquirer` module.

# NPM - Node Package Manager

 + **npm**

# Install Dependencies

# Using Native ECMAScript Modules in Node.js

## Using Inquirer Package

```
npm i inquirer
```

```
npm i --save-dev @types/inquirer
```



# Using Native ECMAScript Modules in Node.js

## Using Inquirer Package

```
npm i inquirer
```

```
npm i --save-dev @types/inquirer
```



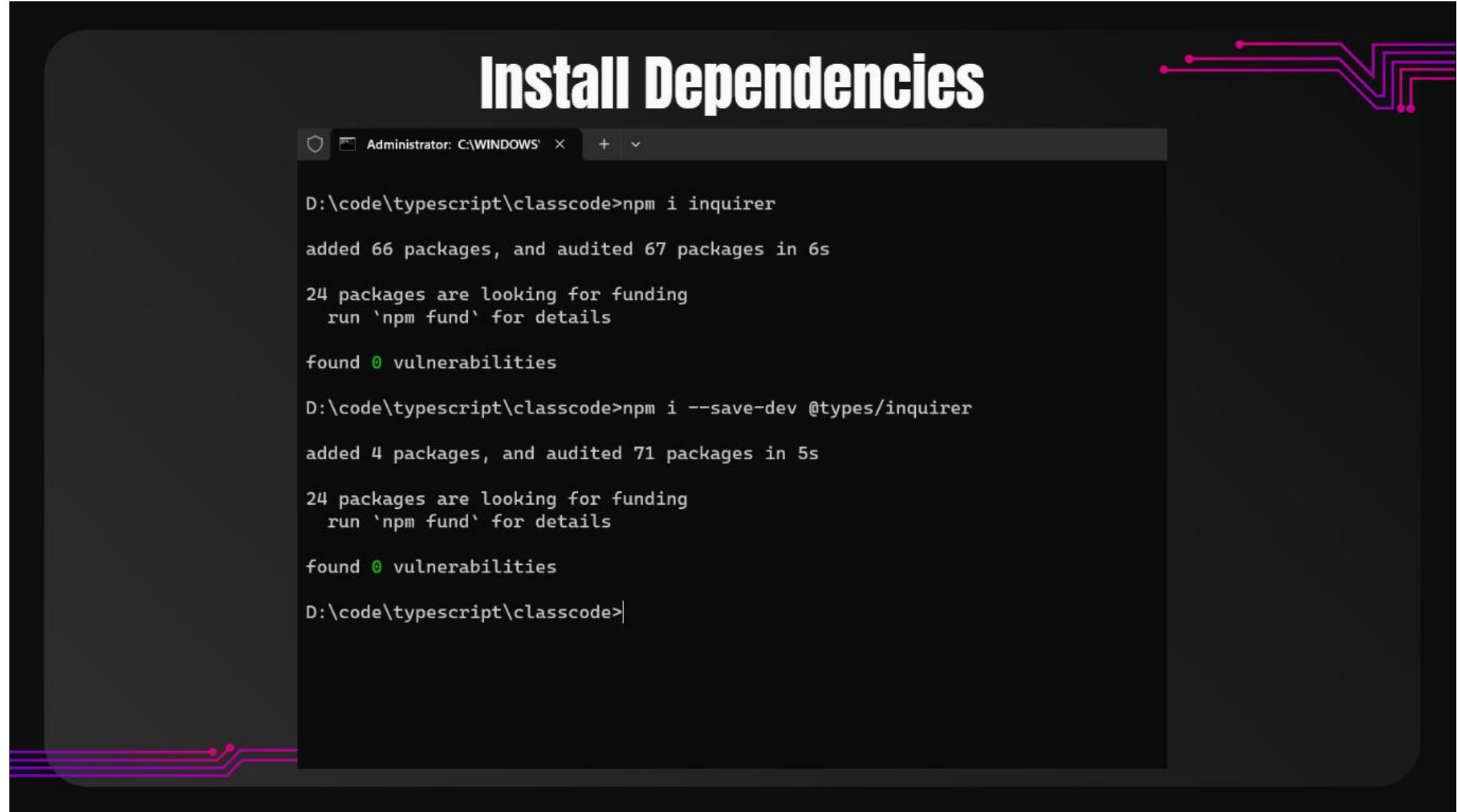
# Install Dependencies



```
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>
```





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.");
10 |
```



```
Administrator: C:\WINDOWS' <-- + ^
```

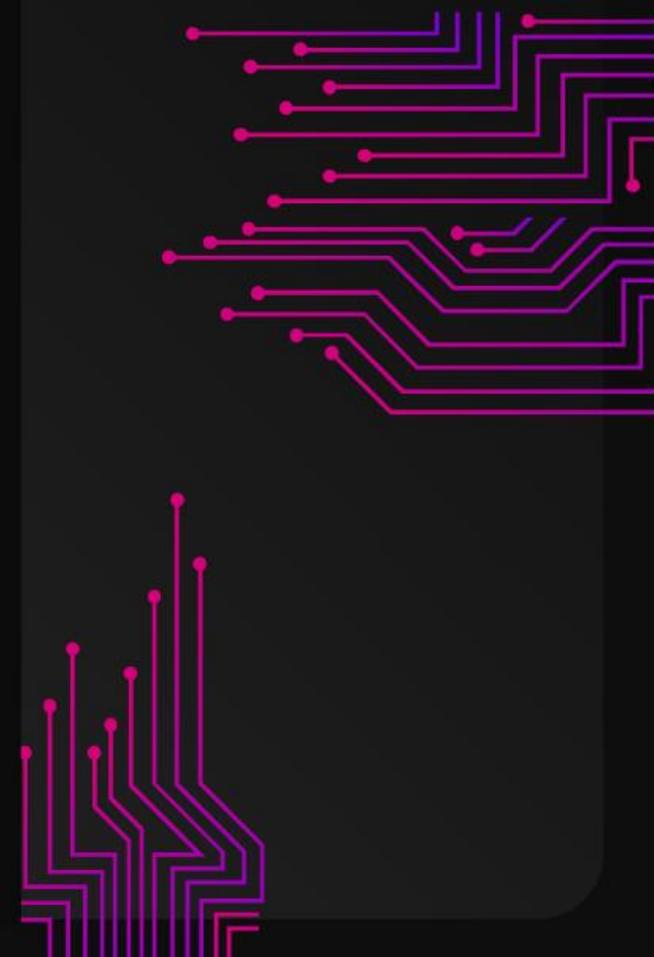
```
D:\code\typescript\classcode>npm i inquirer
added 66 packages, and audited 67 packages in 6s
24 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities

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.

```
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>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] ⏹ ⏷ ⏸ ⏹ ⏷ ⏸ ⏹ ⏷ ⏸
```

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   {
6     name: "age",
7     type: "number",
8     message: "Enter your Age:"
9   }
10]);
11 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>
```

# Thanks!

<https://linktr.ee/ameenalam>



[fb.com/SheikhAmeenAlam](https://fb.com/SheikhAmeenAlam)



[linkedin.com/in/ameen-alam](https://linkedin.com/in/ameen-alam)



[instagram.com/sheikhameenalam](https://instagram.com/sheikhameenalam)



[yt.com/ameenalamofficial](https://yt.com/ameenalamofficial)

Presented by Sir Ameen Alam

