Basic Syntax in TypeScript

1. Problem Statement

Imagine you're joining a new team to build a large, important software system.

- Everyone needs to write code that is clear, consistent, and easy to understand.
- The language you use must have well-defined rules, so that the code is easy to read and maintain, even as the team grows.
- You want to avoid confusion and mistakes that come from inconsistent naming, formatting, or unclear program structure.

The challenge:
How do you learn and apply the basic rules and structure of TypeScript so your code is professional, readable, and reliable?

2. Learning Objectives

By the end of this lesson, you will be able to:

- Understand the basic building blocks of TypeScript syntax.
- Write and organize code using modules, functions, variables, statements, and expressions.
- Use identifiers, keywords, and comments correctly.
- Apply object-oriented principles with classes, objects, and methods.
- Compile and run TypeScript code with different compiler options.

3. Concept Introduction with Analogy

Analogy: Learning a New Language

Learning TypeScript syntax is like learning the grammar and punctuation of a new spoken language:

- **Words and sentences** = Variables, functions, and statements.
- **Grammar rules** = Syntax rules for how you write code.
- **Punctuation** = Semicolons, line breaks, and indentation.
- **Comments** = Notes to yourself or others, like margin notes in a book.

Just as clear grammar makes conversation easy to follow, clear syntax makes your code easy to read and understand.

4. Technical Deep Dive

TypeScript Program Structure

A TypeScript program is made up of:

- **Modules**: Organize code into separate files or sections.
- **Functions**: Blocks of code that perform specific tasks.
- **Variables**: Named storage for values.
- **Statements and Expressions**: Instructions and calculations.
- **Comments**: Notes that explain the code.

Your First TypeScript Code

```
let message: string = "Hello World";
console.log(message);
```

- The first line declares a variable named `message` that holds text.
- The second line prints the value of `message` to the screen.

Compiling and Running TypeScript

1. **Save your code** in a file with a `.ts` extension, e.g., `Test.ts`.
2. **Open your terminal** in the folder containing your file.
3. **Compile the file** using the TypeScript compiler:

```
tsc Test.ts
```

```
This creates a file named  `Test.js`.
```

4. **Run the compiled code** using:

    ```
    node Test.js
    ```

Compiler Flags

Compiler flags let you change how TypeScript compiles your code.
Some useful flags:

Flag	Description
–help	Shows help manual
–module	Load external modules
–target	Set the output language version
–declaration	Generate a type definition file
–removeComments	Remove comments from output
–out	Combine files into one output file
–sourcemap	Generate source map files
–noImplicitAny	Disallow variables with no type
–watch	Watch for file changes and recompile

You can compile multiple files at once:

```
tsc file1.ts file2.ts file3.ts
```

Identifiers in TypeScript

Identifiers are names for variables, functions, classes, etc.
Rules:

- Can include letters, numbers, `_`, or `$`.
- Cannot start with a number.
- Cannot use spaces or special symbols (except `_` and `$`).
- Cannot be a reserved keyword.
- Are case-sensitive.

Examples:

Valid	Invalid
firstName	first name
first_name	1number
num1	first-name
$result	Var

TypeScript Keywords

Reserved words that have special meaning.
Some examples:

`break`, `case`, `const`, `continue`, `do`, `else`, `enum`, `export`, `false`, `for`, `function`, `if`, `import`, `in`, `let`, `new`, `null`, `private`, `public`, `return`, `static`, `super`, `switch`, `this`, `throw`, `true`, `try`, `typeof`, `var`, `void`, `while`, `yield`

Whitespace and Line Breaks

- Spaces, tabs, and newlines are ignored by TypeScript.
- Use them to format and indent your code for readability.

Case Sensitivity

TypeScript is case-sensitive:
`message` and `Message` are different identifiers.

Semicolons

- Semicolons are optional at the end of statements.
- Multiple statements on one line must be separated by semicolons.

```
console.log("Hello");  console.log("World")
```

Comments in TypeScript

- **Single-line:** `// this is a comment`
- **Multi-line:**
  `/* This is a multi-line comment */`

Object Orientation in TypeScript

TypeScript supports object-oriented programming:

- **Object:** Represents a real-world entity with state, behavior, and identity.
- **Class:** Blueprint for creating objects.

- **Method:** Function defined inside a class.

Example:

```
class Greeting {
  greet(): void {
    console.log("Hello World!!!");
  }
}

let obj = new Greeting();
obj.greet();
```

- This defines a class with a method, creates an object, and calls the method.

5. Step-by-Step Data Modeling & Code Walkthrough

1. **Declare a variable:**

   ```
   let productName:  string  =  "Bananas";
   ```

2. **Write a function:**

```
function printProduct(name: string): void {
  console.log("Product: " + name);
}
printProduct(productName);
```

3. **Add a comment:**

   ```
   // This prints the product name
   printProduct(productName);
   ```

4. **Define a class:**

```
class Store {
 open(): void {
   console.log("Store is open!");
 }
}
```

```
let store = new Store();
store.open();
```

6. Interactive Challenge

Your Turn!

- Create a variable for your favorite fruit and print it.
- Write a function that takes a number and prints double its value.
- Add a single-line and a multi-line comment to your code.
- Define a class called `Person` with a method `sayHello` that prints a greeting.

7. Common Pitfalls & Best Practices

- **Don't use invalid identifiers** (e.g., starting with a number or using spaces).
- **Use comments** to explain tricky code.
- **Indent and format** your code for readability.
- **Don't use reserved keywords as names.**
- **Use classes and methods** to organize related code.

8. Quick Recap & Key Takeaways

- TypeScript syntax defines the rules for writing clear, structured programs.
- Use variables, functions, classes, and comments to organize your code.
- Follow identifier and keyword rules to avoid errors.
- Use whitespace and indentation to make your code easy to read.

9. Optional: Programmer’s Workflow Checklist

- Use valid identifiers for all variables and functions.
- Add comments to explain code where needed.
- Organize code into functions and classes.
- Use the TypeScript compiler with helpful flags.
- Format and indent code for readability.
- Test your code by compiling and running it.
- Avoid using reserved keywords as names.