**TypeScript by Sir Daniyal Nagori**

**Lecture # 1**

“TypeScript is a statically/strongly typed open source programming language that builds on JavaScript, giving you better tooling at any scale.”

**Why We should use TypeScript**

1. Static typing
   1. TypeScript is a superset of JavaScript that adds optional static typing and other features such as classes and modules.
   2. TypeScript checks a program for errors before execution, and does so based on the kinds of values, it’s a static type checker.
   3. Once TypeScript’s compiler is done with checking your code, it erases the types to produce the resulting “compiled” code. This means that once your code is compiled, the resulting plain JS code has no type information.
2. Code Completion and Intellisense
   1. Object Oriented Programming : Its helps programmers to write code in a object oriented manner if required. Thus helps users to jump into TS
3. IDE Support
   1. It is super well-supported by text editors including (VS Code, Atom, Sublime, Eclips, etc.)
4. Large Community/Adoption
   1. TypeScript is made for creating large, complex systems that the modern Web abounds with.

**Setting up your** **Environment**

1. Download the latest version of the Node.js from the URL <https://nodejs.org>
2. Now install the Node.js setup
3. After installation you can check the version installed on your system using command : *node* -v
4. Second Step is to download and Install Visual Studio Code from the Microsoft website
5. Now you can install type script by writing following command on the command prompt *: npm install -g typescript*

**Setting Script Execution Policy for Windows**

You need to set the Execution policy so that you can execute scripts in windows. By default you cannot execute scripts. To allow permission you can use command Set-ExecutionPolicy. So run command from the VSCode Terminal Window :

*Set-ExecutionPolicy Bypass -Scope CurrentUser*

**Typescript Compilation Commands**

When you write code in TypeScript, running *tsc* will transform or compile your code into JavaScript.   
E.g: tsc index.ts . This will create a index.js file in the current folder. We can use the command *tsc --init* to initialize our project by creating a tsconfig. json file in your project project directory. This tsconfig.json file will allow you to configure further and customize how TypeScript and the tsc compiler interact. You can remove, add, and change configurations in this file to best meet your needs.

**Creating , Compiling and Executing Your First Project**

1. Start the VS Code
2. Select the folder where you want to create project in VSCode using Menu File-Open Folder option
3. Create a new file in that folder using the create new file icon
4. You can rename it as index.ts
5. Now write console.log(“Hello World”)
6. You also need to create tsconfig.json file for project as: ***tsc --init***

tsconfig.json file is a file of JSON format which allows us to point the root level files and different compiler options to setup that require to compile a TypeScript based projects. The existence of this file in a project specifies that the given directory is the TypeScript project folder root. The tsconfig.json file mainly consists of the information for the following:

* Compiler Options
* Compile OnSave
* Files
* Include
* Exclude

1. Next step is to create package.json file we use npm command as : ***npm init -y*** to create file with default options. We can also use the ***npm init*** to create package.json file if we want to create file by giving values.

The package.json file is the heart of any Node project. It records important metadata about a project which is required before publishing to NPM, and also defines functional attributes of a project that npm uses to install dependencies, run scripts, and identify the entry point to our package.

1. Now to compile this file from VSCode write on its terminal window: ***tsc index.ts***
2. This will create a js file of same name. So index.js file will be created in the same folder
3. Now you can run this js file from VSCode terminal window using command : ***node index.js***

**Fundamentals of Type Script**

Variables : Variable means anything that can vary. A TypeScript variable is simply a name of storage location. A variable must have a unique name. Variables are values in your code that can represent different values each time the code runs. The first time you create a variable, you declare it. And you need a special word for that which are let , var , or const . For Example: *let firstname = "Ali";*

The commonly used naming conventions used for variables are camel-case: Example: *let firstName = "Ali";*

**Rules for Varibale Names**

* ﻿A variable name can't contain any spaces
* ﻿﻿﻿A variable name can contain only letters, numbers, dollar signs, and underscores.
* The first character must be a letter, or an underscore (-), or a dollar sign ($).
* ﻿﻿﻿Subsequent characters may be letters, digits, underscores, or dollar signs.
* Numbers are not allowed as the first character of variable.

**Type Annotation on Variables**

* When you declare a variable using const, var, or let, you can optionally add a type annotation to explicitly specify the type of the variable: *let myName: string = "Alice";*
* TypeScript doesn’t use “types on the left”-style declarations like int x = 0; Type annotations will always go after the thing being typed.
* In most cases, though, this isn’t needed. Wherever possible, TypeScript tries to automatically infer the types in your code.
* // No type annotation needed -- 'myName' inferred as type 'string'
* let myName = "Alice";

**Comments**

* Single line TypeScript comments start with two forward slashes (//).
* All text after the two forward slashes until the end of a line makes up a comment
* Even when there are forward slashes in the commented text.
* Multi-line comments start with /\* and end with \*/.
* Any text between /\* and \*/ will be ignored by JavaScript.

**Primitives Data Types**

* String : A string is used to store a text value. Example: *let firstName = "Ali";*
* Number : A number is used to store a numeric value. Example: *let score = 25*;
* Boolean : A boolean is used to store a value that is either true or false. Example: *let isMarried = false*;
* Undefined : An undefined type is either when it has not been defined or it has not been assigned a value.  
  Example: *let unassigned*;
* Null : null is a special value for saying that a variable is empty or has an unknown value.  
  Example: *let empty = null*;

Note : A variable is undefined when it's not assigned any value after being declared. Null refers to a value that is either empty or doesn't exist. null means no value. To make a variable null we must assign null value to it as by default in typescript unassigned values are termed undefined.

**Statements**

A computer program is a list of "instructions" to be "executed" by a computer. In a programming language, these programming instructions are called statements. A JavaScript program is a list of programming statements.

TypeScript applications consist of statements with an appropriate syntax. A single statement may span multiple lines. Multiple statements may occur on a single line if each statement is separated by a semicolon.

**Declaration using Let , Var & Const**

var and let are both used for variable declaration in TypeScript but the difference between them is that var is function scoped and let is block scoped. Means variable declared using var can be accessed any where in the function whereas the variable declared using let is only accessible within that block. Variable declared by let cannot be redeclared and must be declared before use whereas variables declared with var keyword are hoisted.

const is an augmentation(rise) of let in that it prevents re-assignment to a variable.

Note: Dont use var, use let and const

**Template Literals / Template String**

* Template literals are literals delimited with backtick (`) characters, allowing for multi-line strings, string interpolation (to create strings by doing substitution of placeholders) with embedded expressions, and special constructs called tagged templates.
* Its a new way to deal with strings and specially dynamic strings. Template literals are enclosed by backtick (`) characters
* For Example:
  + Two write multi line string we can use Template literal as

*console.log(`string text line 1*

*string text line 2`);*

* + String interpolation : With template literals, you can avoid the concatenation operator and improve the readability of your code, by using placeholders of the form ${expression} to perform substitutions for embedded expressions:

*console.log("\n Example of String interpolation using Template Literals")*

*const a = 5;*

*const b = 10;*

*console.log(`Fifteen is ${a + b} and*

*not ${2 \* a + b}.`);*

**Analyzing and Modifying Data Types**

* You can check the type of a variable by entering typeof.  
  Example: *let testVariable = 1;   
   console.log("\nDatatype of variable testVariable is : ");  
   console.log(typeof testVariable);*
* The variables in TypeScript cannot change types. Example:   
   *let a = 2;   
   a = “2”;* // This statement will generate error as we are now assigning string type value

**Operators**

* **Arithmetic Operators:** An operator that performs arithmetic operations on groups and numbers is known as arithmetic operators. Typescript supports following Arithmatic Operators : Addition(+), Subtraction(-), Division(/), Multiplication(\*) , Exponential(\*\*) and modulus(%) operator. Consider following Examples

*let n1 = 11;*

*let n2 = 2;*

*console.log(`\nSum of two variable n1=${n1} and n2=${n2} is = ${n1 + n2}`);*

*console.log(`\nSubtraction of two variable n1=${n1} and n2=${n2} is = ${n1 - n2}`);*

*console.log(`\nMultiplication of two variable n1=${n1} and n2=${n2} is = ${n1 \* n2}`);*

*console.log(`\nDivision of two variable n1=${n1} and n2=${n2} is = ${n1 / n2}`);*

*console.log(`\nExponent of two variable n1=${n1} and n2=${n2} is = ${n1 \*\* n2}`);*

*console.log(`\nModulus/Remainder of two variable n1=${n1} and n2=${n2} is = ${n1 % n2}`);*

Note: + operator behaves differently if it is used with two string type variables. It concatenates both strings.

*let str1 = "11";*

*let str2 = "2";*

*console.log(`\nIn case of String + with two string variable str1=${str1} and str2=${str2} is = ${str1 + str2}`);*

* **Assignment Operators:** Assignment operators are used to assign values to variables. The assignment operators are = += -= /= \*=. For example : We can use assignment operators as :

*console.log(`\n We can assign value to a variable using = Operator as n1=12`);*

*console.log(` Using Addition Assignment operation we can do n1 = n1 + 20 as n1+=20 : so if n1=${n1} then n1+=20 will be equal to ${n1+=20} `)*

*console.log(` Using Subtraction Assignment operation we can do n1 = n1 - 20 as n1-=20 : so if n1=${n1} then n1-=20 will be equal to ${n1-=20} `)*

*console.log(` Using Multiplication Assignment operation we can do n1 = n1 \* 20 as n1\*=20 : so if n1=${n1} then n1\*=20 will be equal to ${n1\*=20} `)*

*console.log(` Using Division Assignment operation we can do n1 = n1 / 20 as n1/=20 : so if n1=${n1} then n1/=20 will be equal to ${n1/=20} `)*

* **Comparison Operators:** Comparison operators can compare numbers or strings and perform evaluations. Expressions that use comparison operators do not return a number value as do arithmetic expressions. Comparison expressions return either 1/true , or 0/false . For example :

*let n = 5;*

*console.log('\n Comparison Operators ')*

*console.log(`if value of n is ${n} then n==5 will return ${n==5}`);*

*console.log(`if value of n is ${n} then n===5 will return ${n===5}`);*

*console.log(`if value of n is ${n} then n!=5 will return ${n!=5}`);*

*console.log(`if value of n is ${n} then n > 8 will return ${n>8}`);*

*console.log(`if value of n is ${n} then n >= 8 will return ${n>=8}`);*

*console.log(`if value of n is ${n} then n < 8 will return ${n<8}`);*

*console.log(`if value of n is ${n} then n <= 8 will return ${n<=8}`);*

* **Logical Operators:** Logical operator are used to combine multiple conditions in one. The logical Boolean operators perform logical operations with bool operands.