Introduction to Typescript

TypeScript

JavaScript that scales.

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

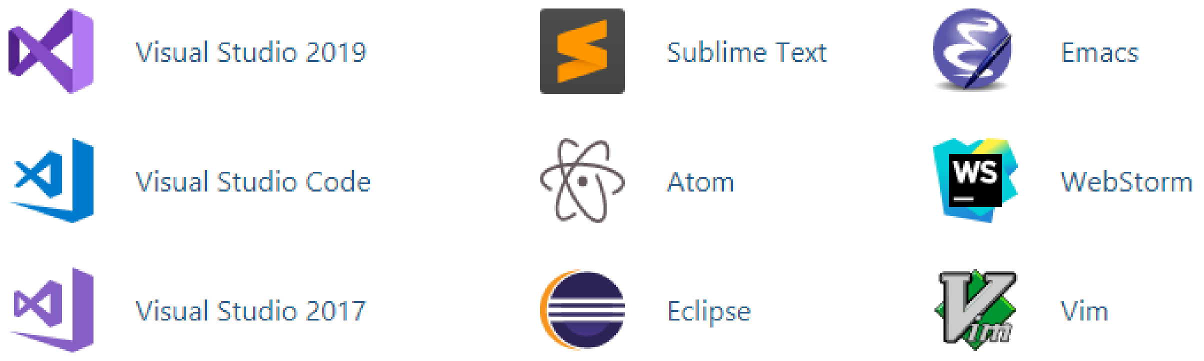
Any browser. Any host. Any OS. Open source.

- Typescript is a **typed superset of JavaScript**.

  TypeScript = **JavaScript + Typed Static + Data Types + Classes + Interfaces + Misc. Concepts**

  Misc. Concepts like Arrow Functions, Multiline Strings, String Interpolation, De-structuring, Modules etc.

- Typescript is developed by **Microsoft Corporation in 2012.**
- Current version is 4.x.x
- Typescript supported by all modern code editors / IDE's such as **Visual Studio Code, Atom, Sublime Text, Web Storm, Eclipse etc.**

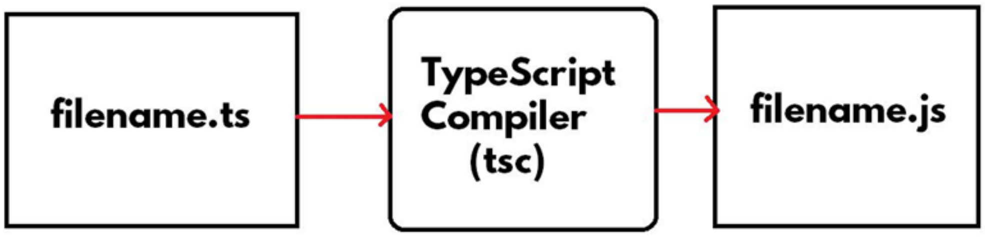


- **Typescript code** can’t be executed directly by the browser.
  So, Typescript code should be converted into JavaScript code.
  For this conversion we require **Transpiler.**

- The process of conversion of one source to another source is called **Transpilation**.

- Typescript provide **tsc** (**Type Script Compiler)** for Transpilation.

  Eg: "filename.ts (TypeScript file)" to "filename.js (JavaScript file)".



```
>tsc filename.ts

filename.js
```

Benefits of Typescript:

Optional Static Typing:

- The data type of the variable is fixed at the time of declaration. The type of a variable can't be changed throughout the program, then it is said to be "**Static Typing**".

  Eg: C, C++, Java, and C#.NET are Static Typing Programming Languages.

- The data type of the variable is not fixed while declaration, and the data type will be taken by the runtime engine automatically at the time of program execution, and possible to assign any type of value to the variable, then it is said to be "**Dynamic typing**".

  Eg: JavaScript, Python, etc.

- Typescript supports "**Optional Static Typing**". That means you can use both "**Dynamic typing**" and "**Static typing**" in Typescript.
  Eg:

  ```
  let name:string = 'Jones';

  let name = 'Jones';
  ```

Type Safety:

- If we specify data type while declaring the variable and when we assign wrong type of value into the variable, the compiler shows errors. This feature is called Type Safety.

  Eg:

  ```
  let x: number = 10;

  x = "Jones"; //error
  ```

Identification of Errors:

- Typing mistakes and syntax errors are displayed as errors in the compilation time itself rather than at runtime.

Environment Setup for Typescript

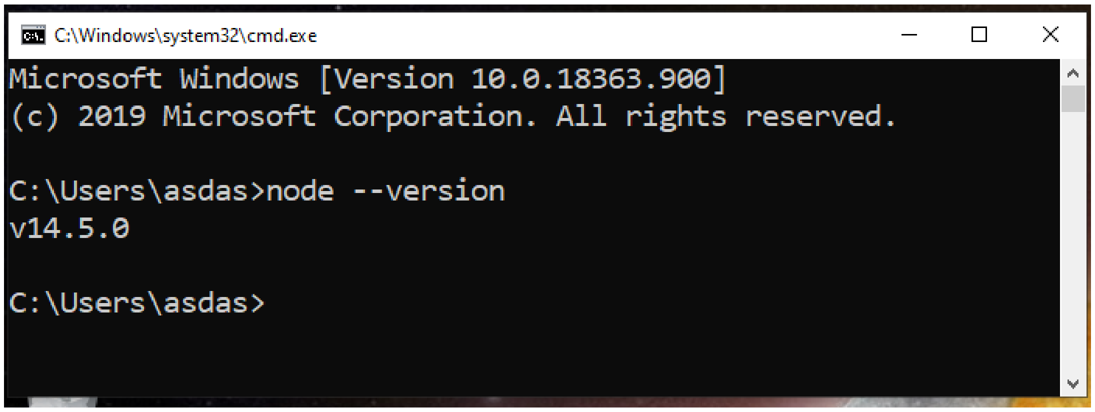
Steps:

1. Installation of NodeJS
2. Installation of Typescript package
3. Installation of Visual Studio Code
4. Create Typescript Workspace
5. Create Typescript Program
6. Trans compile the **first.ts** file in to **first.js** file
7. Execute the **first.js** file using **node** command.

Installation of Typescript

1. Check whether **node.js** is installed or not.

2. Open "**Command Prompt**" and type follow command.
   >node –v
   or
   >node --version



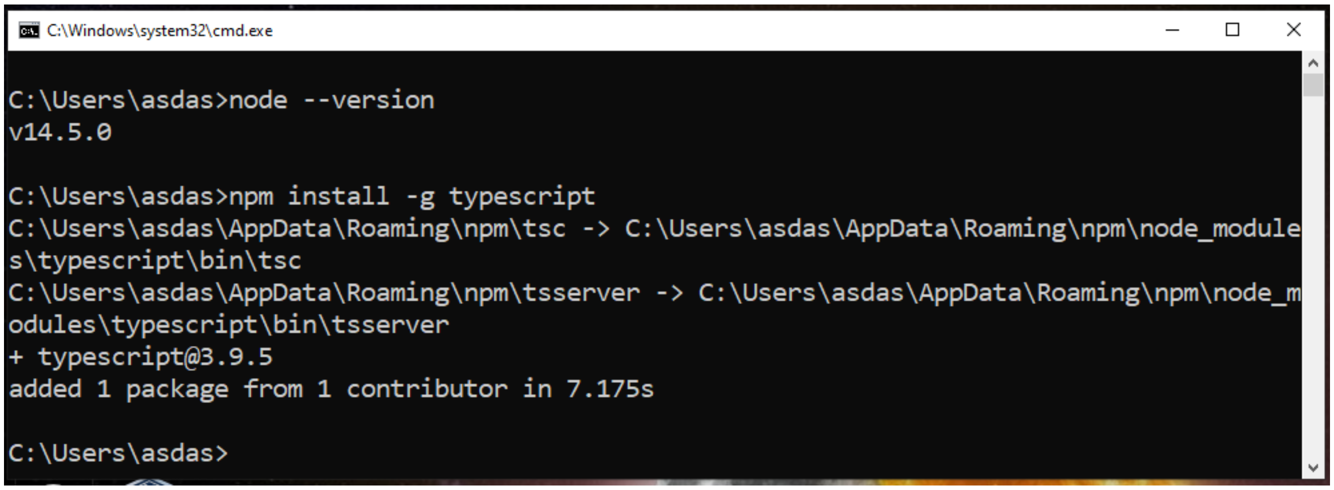
```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\asdas>node --version
v14.5.0

C:\Users\asdas>
```

Note: If NodeJs is not installed, install it first

3. Type the fallowing command
   npm install -g typescript



```
C:\Windows\system32\cmd.exe

C:\Users\asdas>node --version
v14.5.0

C:\Users\asdas>npm install -g typescript
C:\Users\asdas\AppData\Roaming\npm\tsc -> C:\Users\asdas\AppData\Roaming\npm\node_module
s\typescript\bin\tsc
C:\Users\asdas\AppData\Roaming\npm\tsserver -> C:\Users\asdas\AppData\Roaming\npm\node_m
odules\typescript\bin\tsserver
+ typescript@3.9.5
added 1 package from 1 contributor in 7.175s

C:\Users\asdas>
```

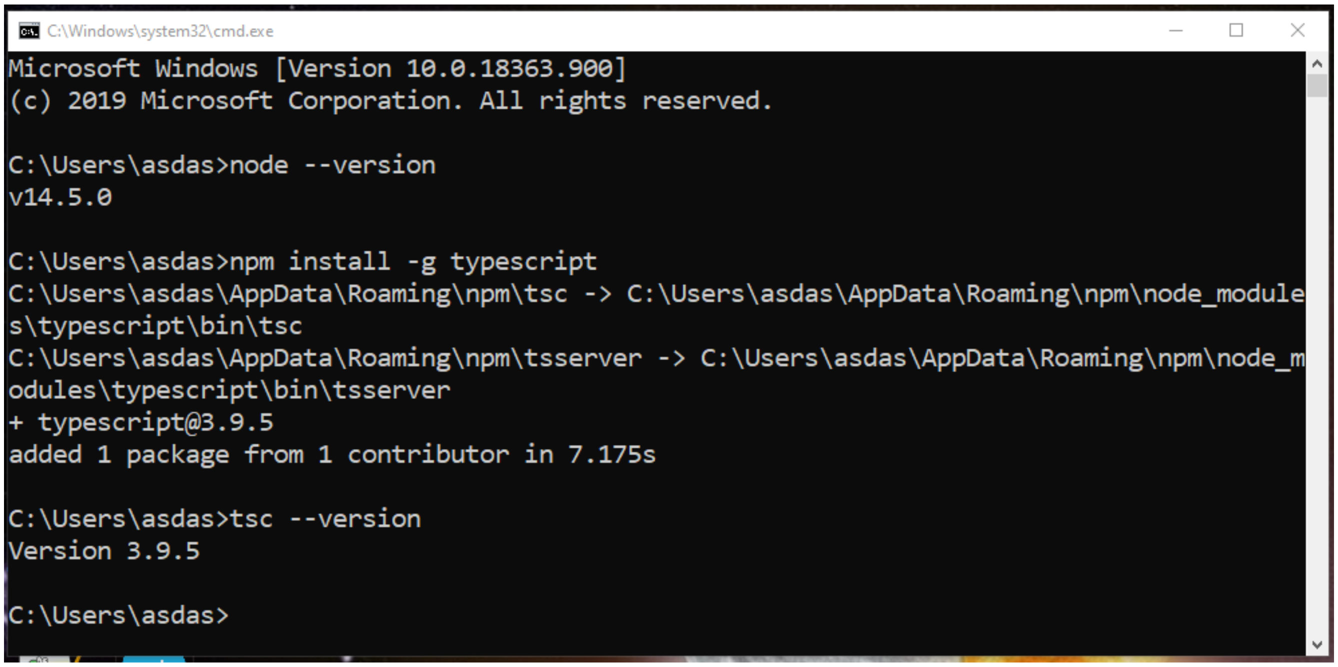
npm:

- npm is the world's largest Software **Registry**.
- This registry contains over 800,000 code packages.
- Open-source developers use npm to share software.
- Many organizations also use npm to manage private development.
- It is installed automatically with nodejs.

Syntax:

npm install <package> options

4. Now type the fallowing command to see Typescript version.
   tsc - -version


C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\asdas>node --version
v14.5.0

C:\Users\asdas>npm install -g typescript
C:\Users\asdas\AppData\Roaming\npm\tsc -> C:\Users\asdas\AppData\Roaming\npm\node_module
s\typescript\bin\tsc
C:\Users\asdas\AppData\Roaming\npm\tsserver -> C:\Users\asdas\AppData\Roaming\npm\node_m
odules\typescript\bin\tsserver
+ typescript@3.9.5
added 1 package from 1 contributor in 7.175s

C:\Users\asdas>tsc --version
Version 3.9.5

C:\Users\asdas>
```

Developing First Typescript Program

Step 1: Creating a file.

- Open “Visual Studio Code”, by clicking on “Start” – “Visual Studio Code” – "Visual Studio Code".
- Visual Studio Code opened.
- Go to “File” – “New File”.
- Type the program as follows

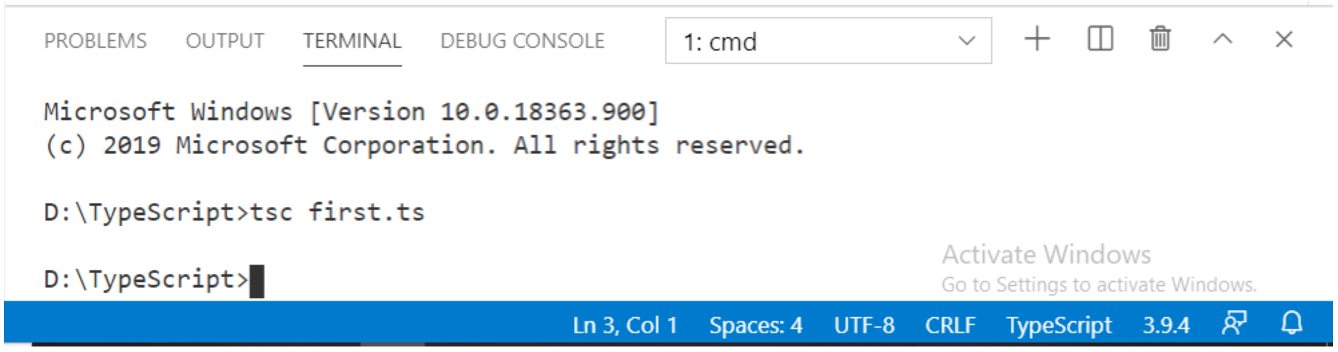
```
var wish:string = "Welcome to TypeScript";

console.log(wish);
```

- Go to “File” menu – “Save” (or) Press Ctrl+S.
- Create a folder as “**Typescript**“ folder in any drive (Here we used **D:\ drive)** and save the file as “first.ts”.
- Click on “Save”.
- Now the typescript file (**D:\TypeScript\first.ts**) is ready.

Step 2: Transpile the Typescript Program

- Open the Terminal => New Terminal

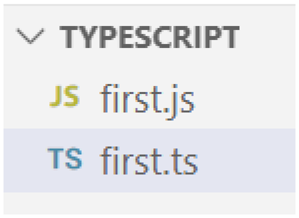


```
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\TypeScript>tsc first.ts

D:\TypeScript>
```

- Now the “first.ts” has been transpiled into “first.js”;
- So “**first.js**” file has been generated in the work space.

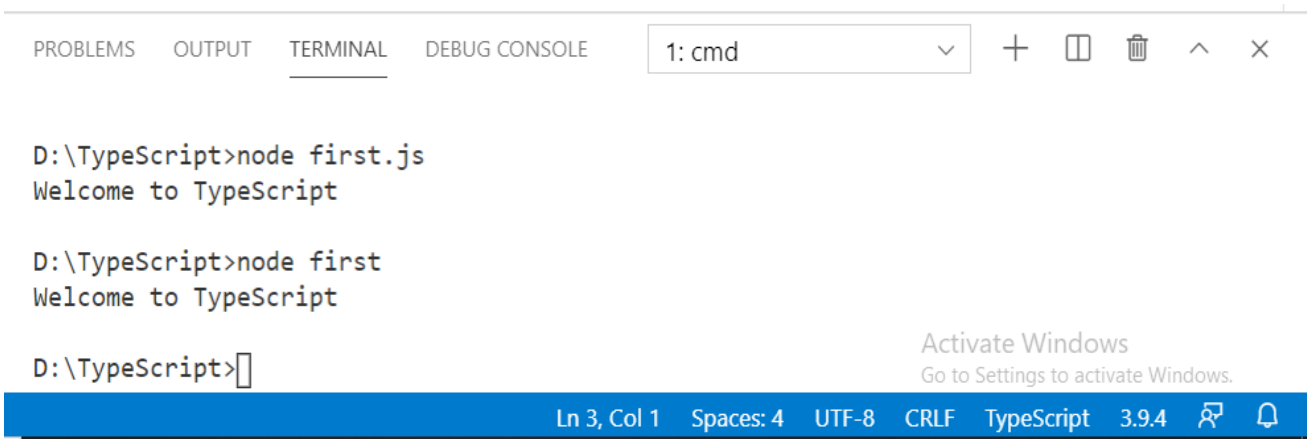


Step 3: Execute the first.js file

>node first.js

or

>node first



```
D:\TypeScript>node first.js
Welcome to TypeScript

D:\TypeScript>node first
Welcome to TypeScript

D:\TypeScript>
```

Note:
The "**node**" command (Provided by Node.js) is used to execute the JavaScript file at Command Prompt.

Basics of Typescript

Variables:

- Variable is a named memory location in RAM, to store a value at run time.

Syntax:

let variableName :DataTypeName = value;

Example:

let a :number = 100;
let s :string = “Jones”

Static Typing	Dynamic Typing
➢ The data type of the variable is fixed at the time of declaration. The type of a variable can’t be changed throughout the program, then it is said to be “**Static Typing**”. Eg: C, C++, Java, and C#.NET are Static Typing Programming Languages.	➢ The data type of the variable is not fixed at the time of declaration, and the data type will be taken by the runtime engine automatically at the time of program execution, and possible to assign any type of value to the variable, then it is said to be “**Dynamic typing**”. Eg: JavaScript, Python, etc.

Note:

- Typescript supports "**Optional Static Typing**".

```
let x:number = 10;//Static Type
x = "Jones";//Error, can't assign different type value

let y = 20;//Optional Static Type
y = "Anna";//Error, can't assign different type value
```

Data Types:

- "**Data type**" specifies what type of value that can be stored in a variable.

number	All types of numbers (integer / floating-point numbers). Eg: 10, 10.3498
string	Collection of characters in double quotes or single quotes. Eg: 'Sai' or "Sai"
boolean	To represent boolean values either **true or false**
any	To represent any type of value
object	To represent an object

Eg:

```
export {};

let customerId: number = 1234;
let customerName: string = "Jones";
let isMarried: boolean = true;
let customerAddress: object = {
  city: "Hyderabad",
  state: "TS",
  country: "India",
};
let customerContact: any = "jones@gmail.com";

console.log("CustomerId: ", customerId);
console.log("CustomerName: ", customerName);
console.log("Married Status: ", isMarried);
console.log("Customer Address: ", customerAddress);
console.log("Customer Contact: ", customerContact);
```

```
typescript>tsc datatypes.ts
typescript>node datatypes.js
CustomerId: 1234
CustomerName: Jones
Married Status: true
Customer Address: {"city":"Hyderabad","state":"TS","country":"India"}
Customer Contact: 8833224455
```

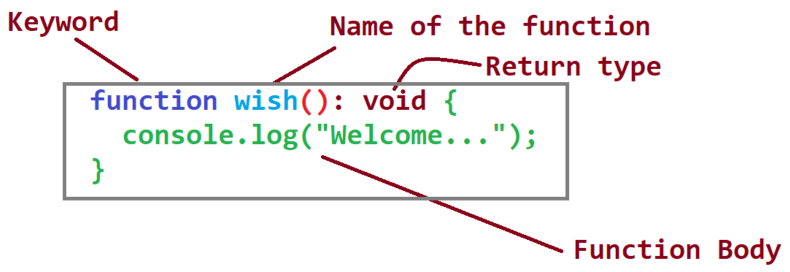
Working with Functions:

- A function is a block of reusable code.
- A function allows us to write code once and use it as many times as we want just by calling the function.

Example:

```
//Function Definition
function wish(): void {
  console.log("Welcome...");
}

//Function Calling
wish();
```



```
function wish(): void {
  console.log("Welcome...");
}
```

Importance of Return type:

- Sometimes a function may return value or not.
- If function does not return any value, then we have to use '**void**' as return type.

```
function sqr(num: number): number {
  return num * num;
}

let sqrOfTen: number = sqr(10);
console.log(sqrOfTen);
```

OOPS Concepts in Type Script

What is Object?

- Object is the primary concept in OOP (Object Oriented Programming).
- "Object" represents a physical item that represents a **person or a thing**.
- Object is a collection of **properties** (details) and **methods** (manipulations).

  For example,

  Student, Employee, Product, etc
- We can create any no. of objects inside the program.

What is Property?

- Properties are the details about the object.
- Properties are used to store a value.

  For example,

  studentName:string = "Scott" is a property in the "student object".
- Properties are stored inside the object.
- The value of property can be changed any no. of times.

What is Method?

- Methods are the operations / tasks of the object, which manipulates / calculates the data and do some process.
- Methods are specified inside the object.

What is Class?

- “Class” represents a model of the object, which defines list of **properties** and **methods** of the object.

  Eg:

  “Student” class represents structure (list of properties and methods) of every student object.

  ```
  class ClassName{
      properties
      methods
      constructor
  }
  ```

- We can create any no. of objects based on a class, using "**new**" keyword.
- All the objects of a class, shares same set of properties & methods of the class.
- We can store the reference of the object in "**reference variable**"; using which you can access the object.

Steps for working with classes and objects

1. **Create a Class:**

   Syntax

   ```
   class ClassName
   {
       property: datatype = value;
       …
       methodName( parameter1: datatype, parameter2: datatype, … ) : return_type   {
           //Method logic
       }
       …
   }
   ```

2. **Perform Object Instantiation or Object Creation:**

   Syntax:

   ```
   let referenceVariableName :ClassName = new ClassName( );
   ```

   Reference Variable: The variable that can store the reference of an object.

3. **Access Properties and Methods using Reference Variable**

   Syntax:

   ```
   referenceVariableName.property
   referenceVariableName.method( );
   ```

Example:

```
export{}
class Student
{
    studentId: number = 101;
    studentName:string = "Jones";
    studentMarks: number = 55;

    getResult():string{
        if (this.studentMarks >= 35) {
            return "Pass";
        }
        else {
            return "Fail";
        }
    }
}
let s1 = new Student();
console.log(s1);
console.log(s1.studentId);
console.log(s1.studentName);
console.log(s1.studentMarks);
console.log(s1.getResult());
```

```
D:\MyTypeScript>tsc test.ts
D:\MyTypeScript>node test
Student { studentId: 101, studentName: 'Jones', studentMarks: 55 }
101
Jones
55
Pass
```

Working with Constructors:

- Constructor is a special function which is a part of the class.
- Constructor is used to **initialize properties** of the class.
- Constructor will be called automatically when we create a new object for the class.
- If you create multiple objects for the same class, the constructor will be called each time when you create new object.
- In Typescript, constructor's name should be always "**constructor**" only.
- Constructor can receive arguments; but can't return any value.
- In TypeScript, we can't define multiple constructors (Overloaded Constructors)

Syntax of Constructor

```
constructor( parameter1: dataType, parameter2: dataType, … ){
    code here
}
```

Constructor - Example

```
export{}
class Student{
    studentId: number;
    studentName:string;
    studentMarks: number;

    constructor(studentId:number,studentName:string,studentMarks:number){
        this.studentId = studentId;
        this.studentName = studentName;
        this.studentMarks = studentMarks;
    }
    getResult():string{
        if (this.studentMarks >= 35) {
            return "Pass";
        }
        else {
            return "Fail";
        }
    }
}
let s1 = new Student(101,"Sai",90);
```

```
console.log(s1);
console.log(s1.getResult());

let s2 = new Student(102,"Ram",65);
console.log(s2);
console.log(s2.getResult());
```

```
D:\TypeScriptDemo>node test
Student { studentId: 101, studentName: 'Sai', marks: 90 }
Pass
Student { studentId: 102, studentName: 'Ram', marks: 65 }
Pass
```

```
export {};

class Student {
  studentId: number;
  studentName: string;
  studentMarks: number;

  constructor(
    studentId: number = 0,
    studentName: string = "Guest",
    studentMarks: number = 0
  ) {
    this.studentId = studentId;
    this.studentName = studentName;
    this.studentMarks = studentMarks;
  }
  getResult(): string {
    if (this.studentMarks >= 40) {
      return "Pass";
    } else {
      return "Fail";
    }
  }
}
let s1 = new Student(101);
console.log(s1);
let s2 = new Student(102, "Smith");
console.log(s2);
let s3 = new Student(103, "Smith",66);
console.log(s3);
```

Composition Or Has-A Relationship:

- By using Class Name or by creating object we can access members of one class inside another class is nothing but composition. Also known as Has-A Relationship.
  Eg:
  Customer Has Address
  Employee Has Address
  Student Has Address

- The main advantage of Has-A Relationship is **Code Reusability**.

Example:

```
export{}
class Address{
    city:string;
    state:string;
    country:string;

    constructor(city:string,state:string,country:string) {
        this.city = city;
        this.state = state;
        this.country = country;
    }
}
class Customer{
    customerId: number;
    customerName: string;
    customerAddress: Address;

    constructor(customerId:number, customerName:string, customerAddress:Address){
        this.customerId = customerId;
        this.customerName = customerName;
        this.customerAddress = customerAddress;
    }
}

let customerAddress:Address = new Address('Hyderabad','Telangana','India');
let customer:Customer = new Customer(101,'Jones',customerAddress);
console.log(customer);
console.log(customer.customerId);
console.log(customer.customerName);
console.log(customer.customerAddress);
```

```
Customer {
  customerId: 101,
  customerName: 'Jones',
  customerAddress: Address { city: 'Hyderabad', state: 'Telangana', country: 'India' }
}
101
Jones
Address { city: 'Hyderabad', state: 'Telangana', coun
```