Introduction:

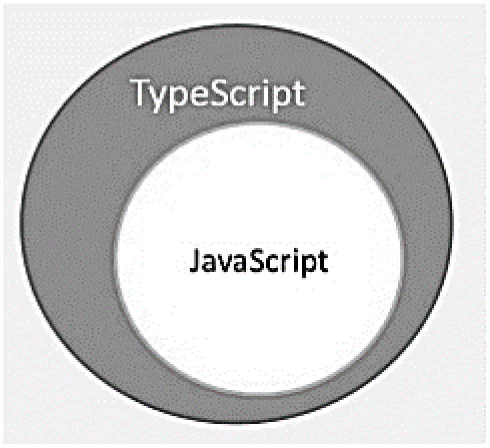
- TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.
- TypeScript is pure object oriented with classes, interfaces and statically typed like C# or Java.
- The popular JavaScript framework **Angular 2.0** is written in TypeScript.

Hello World:

```
var message = "Hello World";

console.log(message);
```

TypeScript:

- "TypeScript is JavaScript for application-scale development."
- TypeScript is a strongly typed, object oriented, compiled language. It was designed by **Anders Hejlsberg** (designer of C#) at Microsoft.
- TypeScript is a typed superset of JavaScript compiled to JavaScript. In other words, TypeScript is JavaScript plus some additional features.



Feature:

- **TypeScript is just JavaScript**: TypeScript starts with JavaScript and ends with JavaScript. Typescript adopts the basic building blocks of your program from JavaScript. Hence, you only need to know JavaScript to use TypeScript. All TypeScript code is converted into its JavaScript equivalent for the purpose of execution.
- **TypeScript supports other JS libraries**. Compiled TypeScript can be consumed from any JavaScript code. TypeScript-generated JavaScript can reuse all of the existing JavaScript frameworks, tools, and libraries.
- **JavaScript is TypeScript**. This means that any valid **.js** file can be renamed to **.ts** and compiled with other TypeScript files.
- **TypeScript is portable**. TypeScript is portable across browsers, devices, and operating systems. It can run on any environment that JavaScript runs on. Unlike its counterparts, TypeScript doesn’t need a dedicated VM or a specific runtime environment to execute.

Why Should We Use TypeScript:

- **Compilation** − JavaScript is an interpreted language. Hence, it needs to be run to test that it is valid. It means you write all the codes just to find no output, in case there is an error. Hence, you have to spend hours trying to find bugs in the code. The TypeScript transpiler provides the error-checking feature. TypeScript will compile the code and generate compilation errors, if it finds some sort of syntax errors. This helps to highlight errors before the script is run.
- **Strong Static Typing** − JavaScript is not strongly typed. TypeScript comes with an optional static typing and type inference system through the TLS (TypeScript Language Service). The type of a variable, declared with no type, may be inferred by the TLS based on its value.
- TypeScript **supports type definitions** for existing JavaScript libraries. TypeScript Definition file (with **.d.ts** extension) provides definition for external JavaScript libraries. Hence, TypeScript code can contain these libraries.
- TypeScript **supports Object Oriented Programming** concepts like classes, interfaces, inheritance, etc.

TypeScript and Object Orientation

- TypeScript is Object-Oriented JavaScript. Object Orientation is a software development paradigm that follows real-world modelling. Object Orientation considers a program as a collection of objects that communicate with each other via mechanism called methods. TypeScript supports these object oriented components too.

Object − An object is a real time representation of any entity. According to Grady Brooch, every object must have three features –

- **State** − described by the attributes of an object
- **Behavior** − describes how the object will act
- **Identity** − a unique value that distinguishes an object from a set of similar such objects.
- **Class** − A class in terms of OOP is a blueprint for creating objects. A class encapsulates data for the object.
- **Method** − Methods facilitate communication between objects.

Example:

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

Variable Declaration In TypeScript:

S.No.	Variable Declaration Syntax & Description
1.	**var name:string = ”mary”** The variable stores a value of type string
2.	**var name:string;** The variable is a string variable. The variable’s value is set to undefined by default
3.	**var name = ”mary”** The variable’s type is inferred from the data type of the value. Here, the variable is of the type string
4.	**var name;** The variable’s data type is any. Its value is set to undefined by default.

TypeScript Variable Scope:

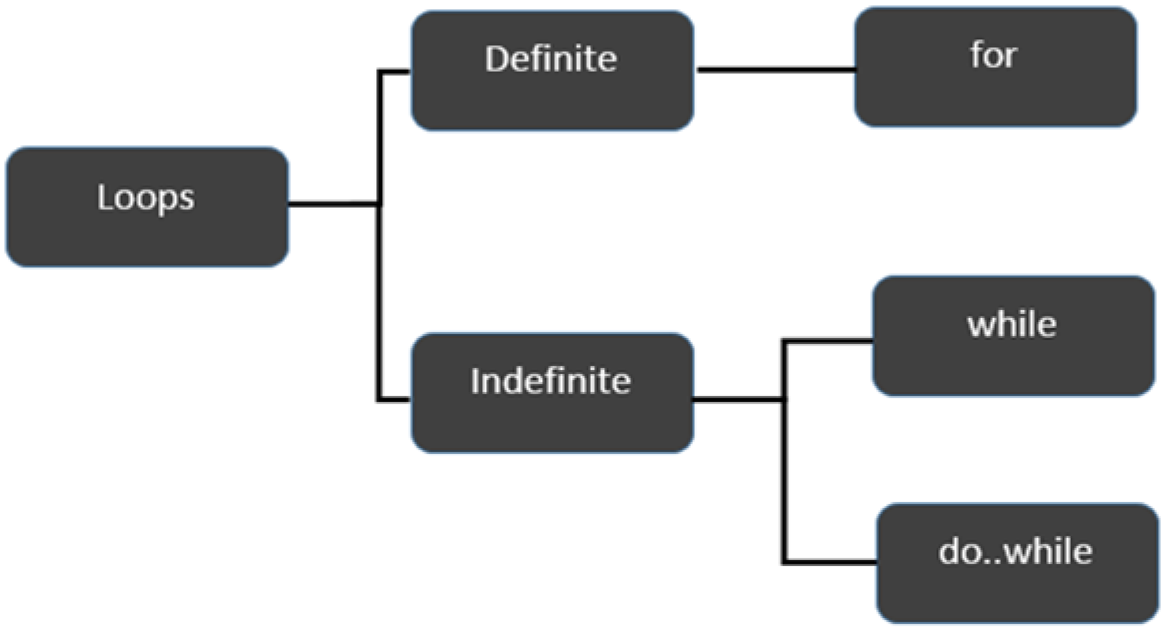
The scope of a variable specifies where the variable is defined. The availability of a variable within a program is determined by its scope. TypeScript variables can be of the following scopes –

- **Global Scope** − Global variables are declared outside the programming constructs. These variables can be accessed from anywhere within your code.

- **Class Scope** − These variables are also called **fields**. Fields or class variables are declared within the class but outside the methods. These variables can be accessed using the object of the class. Fields can also be static. Static fields can be accessed using the class name.
- **Local Scope** − Local variables, as the name suggests, are declared within the constructs like methods, loops etc. Local variables are accessible only within the construct where they are declared.

```
var global_num = 12          //global variable

class Numbers {

   num_val = 13;             //class variable

   static sval = 10;         //static field



   storeNum():void {

      var local_num = 14;    //local variable

   }

}

console.log("Global num: "+global_num)

console.log(Numbers.sval)   //static variable

var obj = new Numbers();

console.log("Global num: "+obj.num_val)
```

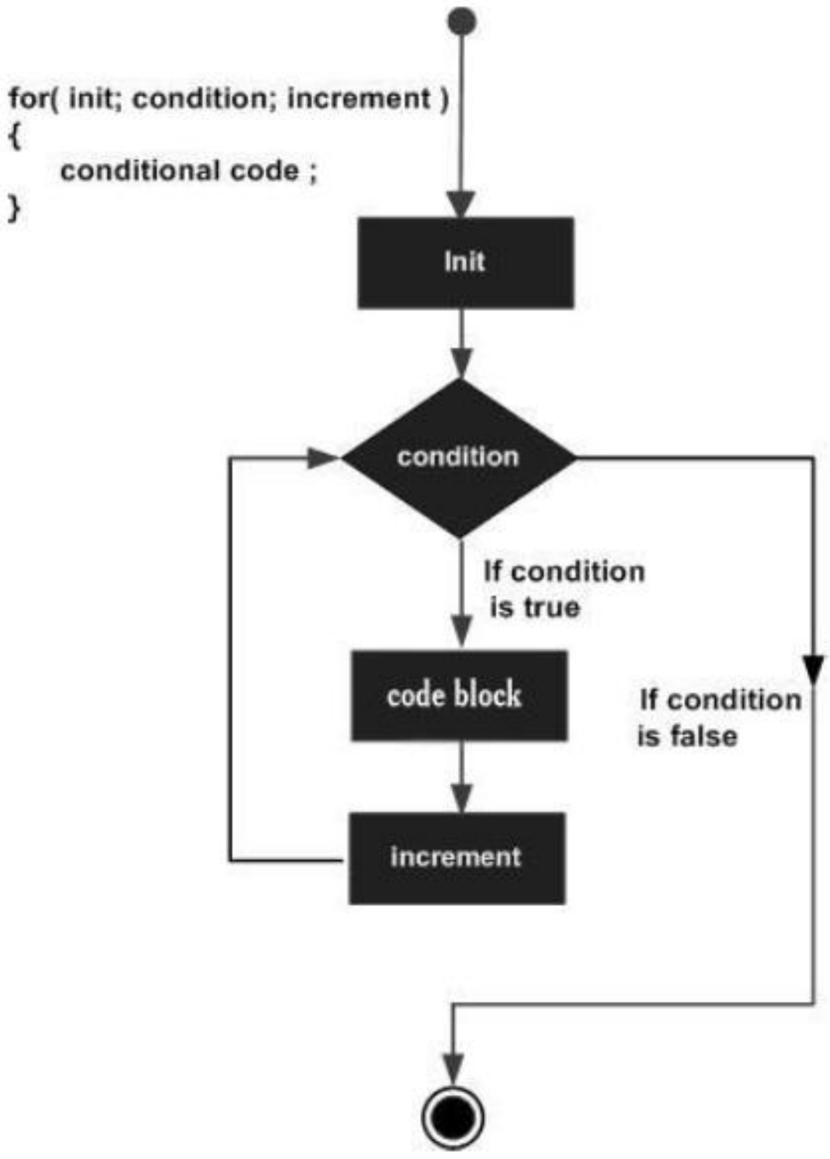
Loops In TypeScript:



TypeScript - For Loop:

The **for** loop executes the code block for a specified number of times. It can be used to iterate over a fixed set of values, such as an array**.**

Flow Diagram:



Example:

```
var num:number = 5;

var i:number;

var factorial = 1;



for(i = num;i>=1;i--) {
   factorial *= i;

}

console.log(factorial)
```

The for...in loop:

Another variation of the *for* loop is the *for... in* loop. The *for… in* loop can be used to iterate over a set of values as in the case of an array or a tuple. The syntax for the same is given below –

```
for (var val in list) {
   //statements
}
```

Let’s take a look at the following example –

Example:

```
var j:any;

var n:any = "a b c"



for(j in n) {

   console.log(n[j])

}
```

It will produce the following output –

```
a
b
c
```

TypeScript - While Loop:

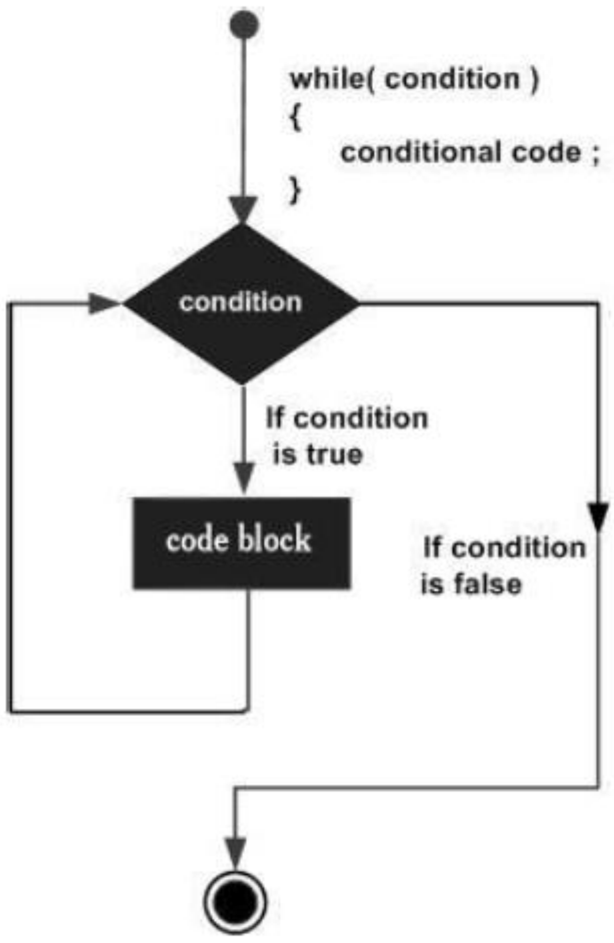
The **while** loop executes the instructions each time the condition specified evaluates to true. In other words, the loop evaluates the condition before the block of code is executed.

Syntax:

```
while(condition) {
   // statements if the condition is true
}
```

Flow Diagram:

-



Example:

```
var num:number = 5;

var factorial:number = 1;



while(num >=1) {

   factorial = factorial * num;

   num--;

}
```

```
console.log("The factorial  is "+factorial);
```

TypeScript – Functions:

Functions are the building blocks of readable, maintainable, and reusable code. A function is a set of statements to perform a specific task. Functions organize the program into logical blocks of code. Once defined, functions may be called to access code. This makes the code reusable. Moreover, functions make it easy to read and maintain the program's code.

Defining a Function:

A function definition specifies what and how a specific task would be done. Before using a function, it must be defined. Functions are defined using the **function** keyword. The syntax for defining a standard function is given below −

Syntax

```
function  function_name() {
   // function body
}
```

TypeScript – Arrays:

An array is a homogenous collection of values. To simplify, an array is a collection of values of the same data type.

Features of an Array:

Here is a list of the features of an array −

- An array declaration allocates sequential memory blocks.
- Arrays are static. This means that an array once initialized cannot be resized.

- Each memory block represents an array element.
- Array elements are identified by a unique integer called as the subscript / index of the element.
- Like variables, arrays too, should be declared before they are used. Use the var keyword to declare an array.
- Array initialization refers to populating the array elements.
- Array element values can be updated or modified but cannot be deleted.

Arrays may be declared and initialized in a single statement. The syntax for the same is −

```
        var array_name[:data type] = [val1,val2…valn]
```

Note − The pair of [] is called the dimension of the array.

Accessing Array Elements:

The array name followed by the subscript is used refer to an array element. Its syntax is as follows −

```
array_name[subscript] = value
```

Example: Simple Array

```
var alphas:string[];

alphas = ["1","2","3","4"]

console.log(alphas[0]);

console.log(alphas[1]);
```

Array Object

An array can also be created using the Array object. The Array constructor can be passed.

- A numeric value that represents the size of the array or
- A list of comma separated values.

The following example shows how to create an array using this method.

Example:

```
var arr_names:number[] = new Array(4)



for(var i = 0;i<arr_names.length;i++;) {

   arr_names[i] = i * 2

   console.log(arr_names[i])

}
```

TypeScript – Classes:

TypeScript is object oriented JavaScript. TypeScript supports object-oriented programming features like classes, interfaces, etc. A class in terms of OOP is a blueprint for creating objects. A class encapsulates data for the object. Typescript gives built in support for this concept called class. JavaScript ES5 or earlier didn't support classes. Typescript gets this feature from ES6.

Creating classes:

Use the class keyword to declare a class in TypeScript. The syntax for the same is given below −

Syntax:

```
class class_name {
   //class scope
}
```

The class keyword is followed by the class name. The rules for identifiers must be considered while naming a class.

A class definition can include the following −

- **Fields** − A field is any variable declared in a class. Fields represent data pertaining to objects
- **Constructors** − Responsible for allocating memory for the objects of the class
- **Functions** − Functions represent actions an object can take. They are also at times referred to as methods

Example: Declaring a class:

```
class Car {

   //field

   engine:string;



   //constructor

   constructor(engine:string) {

      this.engine = engine

   }

   //function

   disp():void {

      console.log("Engine is  :   "+this.engine)

   }
}
```

Creating Instance objects

To create an instance of the class, use the **new** keyword followed by the class name. The syntax for the same is given below −

Syntax

```
var object_name = new class_name([ arguments ])
```

- The **new** keyword is responsible for instantiation.
- The right-hand side of the expression invokes the constructor. The constructor should be passed values if it is parameterized.

Example: Instantiating a class

```
var obj = new Car("Engine 1")
```

Example: Putting them together

```
class Car {

   //field

   engine:string;



   //constructor

   constructor(engine:string) {

      this.engine = engine

   }
   //function

   disp():void {

      console.log("Function displays Engine is  :   "+this.engine)

   }

}

//create an object

var obj = new Car("XXSY1")



//access the field

console.log("Reading attribute value Engine as :  "+obj.engine)



//access the function

obj.disp()
```

Class Inheritance:

TypeScript supports the concept of Inheritance. Inheritance is the ability of a program to create new classes from an existing class. The class that is extended to create newer classes is called the parent class/super class. The newly created classes are called the child/sub classes.

A class inherits from another class using the ‘extends’ keyword. Child classes inherit all properties and methods except private members and constructors from the parent class.

Syntax:

```
class child_class_name extends parent_class_name
```

However, TypeScript doesn’t support multiple inheritance.

Example: Class Inheritance:

```
class Shape {

   Area:number



   constructor(a:number) {

      this.Area = a

   }

}

class Circle extends Shape {

   disp():void {

      console.log("Area of the circle:  "+this.Area)

   }

}

Var obj = new Circle(223);

obj.disp()
```

Types of Inheritance in TypeScript:

- **Single** – Every class can at the most extend from one parent class
- **Multiple** – A class can inherit from multiple classes. TypeScript doesn't support multiple inheritance.
- **Multi-level** – The following example shows how multi-level inheritance works.

Method Overriding:

Method Overriding is a mechanism by which the child class redefines the superclass's method.

```
class PrinterClass {

   doPrint():void {

      console.log("doPrint() from Parent called…")

   }

}



class StringPrinter extends PrinterClass {

   doPrint():void {

      super.doPrint()

      console.log("doPrint() is printing a string…")

   }

}

var obj = new StringPrinter()

obj.doPrint()
```

The access modifiers supported by TypeScript are:

S.No.	Access Specifier & Description
1.	**public** A public data member has universal accessibility. Data members in a class are public by default.
2.	**private** Private data members are accessible only within the class that defines these members. If an external class member tries to access a private member, the compiler throws an error.
3.	**protected** A protected data member is accessible by the members within the same class as that of the former and also by the members of the child classes.

Interface in TypeScript:

An interface is a syntactical contract that an entity should conform to. In other words, an interface defines the syntax that any entity must adhere to. Interfaces define properties, methods, and events, which are the members of the interface. Interfaces contain only the declaration of the members. It is the responsibility of the deriving class to define the members.

Example: Interface and Objects:

```
interface IPerson {

   firstName:string,
   lastName:string,

   sayHi: ()=>string

}



var customer:IPerson = {

   firstName:"Tom",

   lastName:"Hanks",

   sayHi: ():string =>{return "Hi there"}

}



console.log("Customer Object ")

console.log(customer.firstName)

console.log(customer.lastName)

console.log(customer.sayHi())



var employee:IPerson = {

   firstName:"Jim",

   lastName:"Blakes",

   sayHi: ():string =>{return "Hello!!!"}

}



console.log("Employee  Object ")
console.log(employee.firstName) console.log(employee.lastName)
```