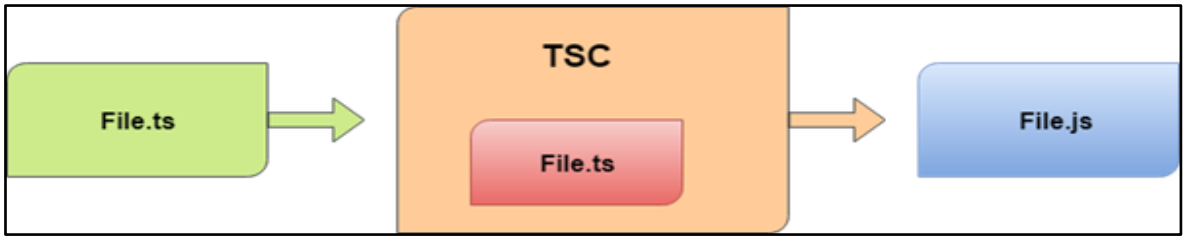
✅TypeScript:

TypeScript is an open-source pure **object-oriented programming language**. It is a strongly typed superset of **JavaScript whi**ch compiles to plain JavaScript. It contains all elements of the JavaScript. It is a language designed for large-scale JavaScript application development, which can be executed on any browser, any Host, and any Operating System. The TypeScript is a language as well as a set of tools. TypeScript is the ES6 version of JavaScript with some additional features.

TypeScript cannot run directly on the browser. It needs a compiler to compile the file and generate it in JavaScript file, which can run directly on the browser. The TypeScript source file is in ".ts" extension. We can use any valid ".js" file by renaming it to ".ts" file. TypeScript uses TSC (TypeScript Compiler) compiler, which convert Typescript code (.ts file) to JavaScript (.js file).



🔰Why use TypeScript?

We use TypeScript because of the following benefits.

- TypeScript supports Static typing, Strongly type, Modules, Optional Parameters, etc.
- TypeScript supports object-oriented programming features such as classes, interfaces, inheritance, generics, etc.
- TypeScript is fast, simple, and most importantly, easy to learn.
- TypeScript provides the error-checking feature at compilation time. It will compile the code, and if any error found, then it highlighted the mistakes before the script is run.
- TypeScript supports all JavaScript libraries because it is the superset of JavaScript.
- TypeScript support reusability because of the inheritance.
- TypeScript make app development quick and easy as possible, and the tooling support of TypeScript gives us autocompletion, type checking, and source documentation.
- TypeScript has a definition file with. d.ts extension to provide a definition for external JavaScript libraries.
- TypeScript supports the latest JavaScript features, including ECMAScript 2015.
- TypeScript gives all the benefits of ES6 plus more productivity.
- Developers can save a lot of time with TypeScript.

⭕Advantage of TypeScript over JavaScript:

- TypeScript always highlights errors at compilation time during the time of development, whereas JavaScript points out errors at the runtime.
- TypeScript supports strongly typed or static typing, whereas this is not in JavaScript.
- TypeScript runs on any browser or JavaScript engine.
- Great tooling supports with IntelliSense which provides active hints as the code is added.
- It has a namespace concept by defining a module.

⭕Disadvantage of TypeScript over JavaScript:

- TypeScript takes a long time to compile the code.
- TypeScript does not support abstract classes.
- If we run the TypeScript application in the browser, a compilation step is required to transform TypeScript into JavaScript.

☢Features of TypeScript;

Object-Oriented language: TypeScript provides a complete feature of an object-oriented programming language such as classes, interfaces, inheritance, modules, etc. In TypeScript, we can write code for both client-side as well as server-side development.

TypeScript supports JavaScript libraries: TypeScript supports each JavaScript elements. It allows the developers to use existing JavaScript code with the TypeScript. Here, we can use all of the JavaScript frameworks, tools, and other libraries easily.

JavaScript is TypeScript: It means the code written in JavaScript with valid .js extension can be converted to TypeScript by changing the extension from .js to .ts and compiled with other TypeScript files.

TypeScript is portable: TypeScript is portable because it can be executed on any browsers, devices, or any operating systems. It can be run in any environment where JavaScript runs on. It is not specific to any virtual-machine for execution.

DOM Manipulation: TypeScript can be used to manipulate the DOM for adding or removing elements similar to JavaScript.

♋Components of TypeScript:

The TypeScript language is internally divided into three main layers. Each of these layers is divided into sublayers or components. In the following diagram, we can see the three layers and each of their internal components. These layers are:

- Language
- The TypeScript Compiler
- The TypeScript Language Services

☣TypeScript Installation: {see particle}

TypeScript First Program:

Let us write a program in the text editor, save it, compile it, run it, and display the output to the console. To do this, we need to perform the following steps.

Step-1 Open the Text Editor and write/copy the following code.

```
let message: string ="Hello world ";

console.log(message)
```

Step-2 Save the above file as ".ts" extension.

Step-3 Compile the TypeScript code. To compile the source code, open the command prompt, and then goes to the file directory location where we saved the above file. For example, if we save the file on the desktop, go to the terminal window and type: - cd Desktop/folder_name. Now, type the following command tsc filename.ts for compilation and press Enter.

```
javatpoint@root: ~/Desktop/TypeScriptFiles
javatpoint@root:~$ cd Desktop/TypeScriptFiles
javatpoint@root:~/Desktop/TypeScriptFiles$ tsc FirstCode.ts
javatpoint@root:~/Desktop/TypeScriptFiles$
```

Step-4 Now, to run the above JavaScript file, type the following command in the terminal window: node filename.js and press Enter. It gives us the final output as:

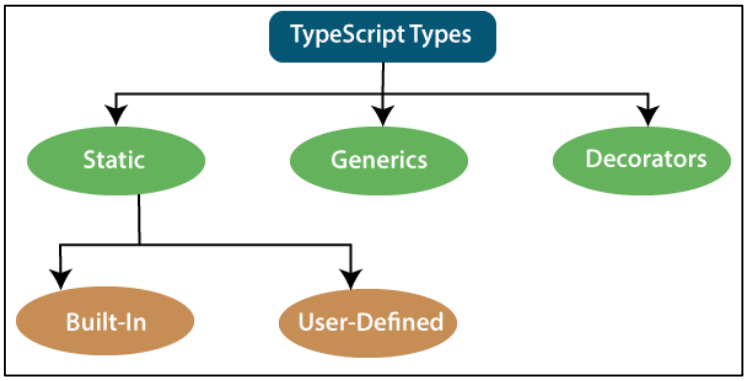
```
javatpoint@root: ~/Desktop/TypeScriptFiles
javatpoint@root:~$ cd Desktop/TypeScriptFiles
javatpoint@root:~/Desktop/TypeScriptFiles$ tsc FirstCode.ts
javatpoint@root:~/Desktop/TypeScriptFiles$ node FirstCode.js
Hello, JavaTpoint
javatpoint@root:~/Desktop/TypeScriptFiles$
```

TypeScript Type:

The TypeScript language supports different types of values. It provides data types for the JavaScript to transform it into a strongly typed programming language. JavaScript doesn't support data types, but with the help of TypeScript, we can use the data types feature in JavaScript.

TypeScript plays an important role when the object-oriented programmer wants to use the type feature in any scripting language or object-oriented programming language. The Type System checks the validity of the given values before the program uses them. It ensures that the code behaves as expected.

TypeScript provides data types as an optional Type System. We can classify the TypeScript data type as following:



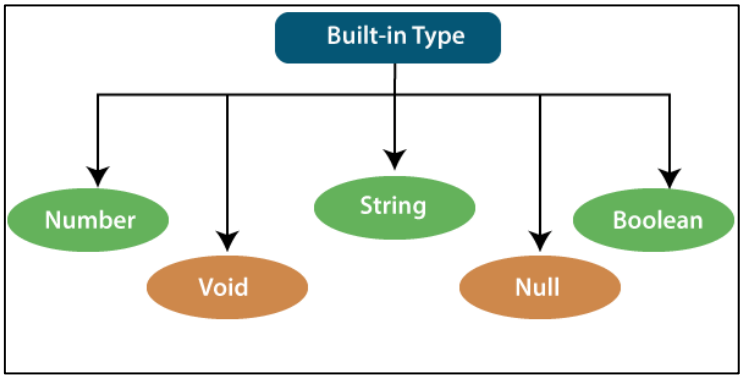
1. Static Types:

In the context of type systems, static types mean "at compile time" or "without running a program." In a statically typed language, variables, parameters, and objects have types that the compiler knows at compile time. The compiler used this information to perform the type checking.

Static types can be further divided into two sub-categories:

Built-in or Primitive Type:

The TypeScript has five built-in data types, which are given below:



Number:

Like JavaScript, all the numbers in TypeScript are stored as floating-point values. These numeric values are treated like a number data type. The numeric data type can be used to represents both integers and fractions. TypeScript also supports Binary (Base 2), Octal (Base 8), Decimal (Base 10), and Hexadecimal (Base 16) literals.

Syntax: let identifier: number = value;

Ex .

```
1. let first: number = 12.0;          // number
2. console.log(first);
```

String:

We will use the string data type to represents the text in TypeScript. String type work with textual data. We include string literals in our scripts by enclosing them in single or double quotation marks. It also represents a sequence of Unicode characters. It embedded the expressions in the form of $ {expr}.

Syntax:

```
let identifier: string = " ";          Or

let identifier: string = ' ';
```

```
1. let empName: string = "Rohan";
2. let empDept: string = "IT";
3. // Before-ES6
4. let output1: string = employeeName + " works in the " + employeeDept + " department.";
5. // After-ES6
6. let output2: string = `${empName} works in the ${empDept} department.`;
7. console.log(output1);//Rohan works in the IT department.
8. console.log(output2);//Rohan works in the IT department.
```

Boolean:

The string and numeric data types can have an unlimited number of different values, whereas the Boolean data type can have only two values. They are "true" and "false." A Boolean value is a truth value which specifies whether the condition is true or not.

Syntax

```
1. let identifier: BooleanBoolean = Boolean value;
```

Examples

```
1. let isDone: boolean
```

Void:

A void is a return type of the functions which do not return any type of value. It is used where no data type is available. A variable of type void is not useful because we can only assign undefined or null to them. An undefined data type denotes uninitialized variable, whereas null represents a variable whose value is undefined.

Syntax

```
let unusable: void = undefined;
```

```
1. 1. function helloUser(): void {
2.         alert("This is a welcome message");
3.         }
```

Null:

Null represents a variable whose value is undefined. Much like the void, it is not extremely useful on its own. The Null accepts the only one value, which is null. The Null keyword is used to define the Null type in TypeScript, but it is not useful because we can only assign a null value to it.

Examples

```
1. let num: number = null;
2. let bool: boolean = null;
3. let str: string = nul
```

Undefined:

The Undefined primitive type denotes all uninitialized variables in TypeScript and JavaScript. It has only one value, which is undefined. The undefined keyword defines the undefined type in TypeScript, but it is not useful because we can only assign an undefined value to it.

Example

```
1. let num: number = undefined;
2. let bool: boolean = undefined;
3. let str: string = undefined;
```

Any Type:

It is the "super type" of all data type in TypeScript. It is used to represents any JavaScript value. It allows us to opt-in and opt-out of type-checking during compilation. If a variable cannot be represented in any of the basic data types, then it can be declared using "Any" data type. Any type is useful when we do not know about the type of value (which might come from an API or 3rd party library), and we want to skip the type-checking on compile time.

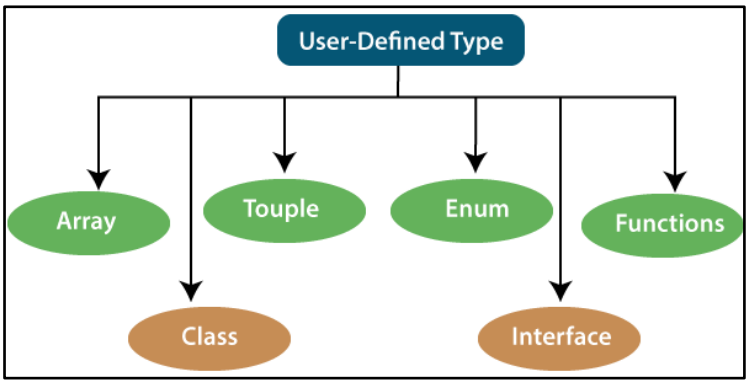
Syntax:

```
let identifier: any = value
```

```
1. let val: any = 'Hi';
2.      val = 555;   // OK
3.      val = true;   // OK
```

User-Defined Data Type:

TypeScript supports the following user-defined data types:



Array

An array is a collection of elements of the same data type. Like JavaScript, TypeScript also allows us to work with arrays of values. An array can be written in two ways:

1. Use the type of the elements followed by [] to denote an array of that element type:

```
var list : number[] = [1, 3, 5];
```

2. The second way uses a generic array type:

```
var list : Array<number> = [1, 3, 5];
```

Tuple:

The Tuple is a data type which includes two sets of values of different data types. It allows us to express an array where the type of a fixed number of elements is known, but they are not the same. For example, if we want to represent a value as a pair of a number and a string, then it can be written as:

```
// Declare a tuple
let a: [string, number];
// Initialize it
a = ["hi", 8, "how", 5]; // OK
```

Interface:

An Interface is a structure which acts as a contract in our application. It defines the syntax for classes to follow, means a class which implements an interface is bound to implement all its members. It cannot be instantiated but can be referenced by the class which implements it. The TypeScript compiler uses interface for type-checking that is also known as "duck typing" or "structural subtyping."

```
interface Calc {
    subtract (first: number, second: number): any;
}

let Calculator: Calc = {
    subtract(first: number, second: number) {
        return first - second;
    }
}
```

Class:

Classes are used to create reusable components and acts as a template for creating objects. It is a logical entity which store variables and functions to perform operations. TypeScript gets support for classes from ES6. It is different from the interface which has an implementation inside it, whereas an interface does not have any implementation inside it.

```
class Student
{
    RollNo: number;
    Name: string;
    constructor(_RollNo: number, Name: string)
    {
        this.RollNo = _rollNo;
        this.Name = _name;
    }
    showDetails()
    {
        console.log(this.rollNo + " : " + this.name);
    }
}
```

Enums:

Enums define a set of named constant. TypeScript provides both string-based and numeric-based enums. By default, enums begin numbering their elements starting from 0, but we can also change this by manually setting the value to one of its elements. TypeScript gets support for enums from ES6.

```
enum Color {
        Red, Green, Blue
};
let c: Color;
ColorColor = Color.Green;
```

Functions:

A function is the logical blocks of code to organize the program. Like JavaScript, TypeScript can also be used to create functions either as a named function or as an anonymous function. Functions ensure that our program is readable, maintainable, and reusable. A function declaration has a function's name, return type, and parameters.

```
1. //named function with number as parameters type and return type
2. function add(a: number, b: number): number {
3.         return a + b;
4. }
5.
6. //anonymous function with number as parameters type and return type
7. let sum = function (a: number, y: number): number {
8.         return a + b;
9. };
```

Variables:

A variable is the storage location, which is used to store value/information to be referenced and used by programs. It acts as a container for value in code and must be declared before the use. We can declare a variable by using the var keyword. In TypeScript, the variable follows the same naming rule as of JavaScript variable declaration. These rules are-

- The variable name must be an alphabet or numeric digits.
- The variable name cannot start with digits.
- The variable name cannot contain spaces and special character, except the underscore(_) and the dollar($) sign.

In ES6, we can define variables using **let and const** keyword. These variables have similar syntax for variable declaration and initialization but differ in scope and usage. In TypeScript, there is always recommended to define a variable using **let** keyword because it provides the **type safety**.

The **let** keyword is similar to **var** keyword in some respects, and **const** is an let which prevents prevents re-assignment to a variable.

Variable Declaration

We can declare a variable in one of the four ways:

1. Declare type and value in a single statement

var [identifier] : [type-annotation] = value;

2. Declare type without value. Then the variable will be set to undefined.

var [identifier] : [type-annotation];

3. Declare its value without type. Then the variable will be set to any.

var [identifier] = value;

4. Declare without value and type. Then the variable will be set to any and initialized with undefined.

var [identifier];

Let's understand all the three variable keywords one by one.

var keyword

Generally, var keyword is used to declare a variable in JavaScript.

var x = 50;

We can also declare a variable inside the function:

```
function a() {
    var msg = " Welcome to JavaTpoint !! ";
    return msg;
}
a();
```

TypeScript Operators:

An Operator is a symbol which operates on a value or data. It represents a specific action on working with data. The data on which operators operates is called operand. It can be used with one or more than one values to produce a single value. All of the standard JavaScript operators are available with the TypeScript program.

Example: 10 + 10 = 20;

In the above example, the values '10' and '20' are known as an operand, whereas '+' and '=' are known as operators.

Operators in TypeScript

In TypeScript, an operator can be classified into the following ways.

- Arithmetic operators
- Comparison (Relational) operators
- Logical operators
- Bitwise operators
- Assignment operators
- Ternary/conditional operator
- Concatenation operator
- Type Operator

2.7.1 Arithmetic Operators

Arithmetic operators take numeric values as their operands, performs an action, and then returns a single numeric value. The most common arithmetic operators are addition(+), subtraction(–), multiplication(*), and division(/).

Operator	Operator_Name	Description	Example

+	Addition	It returns an addition of the values.	let a = 10;
			let b = 20;
			let c = a + b;
			console.log(c);
			//Output30

-	Subtraction	It returns the difference of the values.	let a = 20;
			let b = 10;
			let c = a - b;
			console.log(c);
			//Output10

*	Multiplication	It returns the product of the values.	let a = 30;
			let b = 20;
			let c = a * b;
			console.log(c);
			//Output600

/	Division	It performs the division operation, and returns the quotient.	let a = 100;
			let b = 20;
			let c = a / b;
			console.log(c);
			//Output5

Operator	Operator_Name	Description	Example

%	Modulus	It performs the division operation and returns the remainder.	let a = 75;
			let b = 20;
			let c = a % b;
			console.log(c);
			//Output15

++	Increment	It is used to increments the value of the variable by one.	let a = 15;
			a++;
			console.log(a);
			//Output16

--	Decrement	It is used to decrements the value of the variable by one.	let a = 15;
			a--;
			console.log(a);
			//Output14

2.7.2 Comparison (Relational) Operators

The comparison operators are used to compares the two operands. These operators return a Boolean value true or false. The important comparison operators are given below.

Operator	Operator_Name	Description	Example

==	Is equal to	It checks whether the values of the two operands are equal or not.	let a = 10;
			let b = 20;
			console.log(a==b); //false
			console.log(a==10); //true
			console.log(10=='10'); //true

===	Identical(equal and of the same type)	It checks whether the type and values of the two operands are equal or not.	let a = 10;
			let b = 20;
			console.log(a===b); //false
			console.log(a===10); //true
			console.log(10==='10'); //false

!=	Not equal to	It checks whether the values of the two operands are equal or not.	let a = 10;
			let b = 20;
			console.log(a!=b); //true
			console.log(a!=10); //false
			console.log(10!='10'); //false

!==	Not identical	It checks whether the type and values of the two operands are equal or not.	let a = 10;
			let b = 20;
			console.log(a!==b); //true
			console.log(a!==10); /false
			console.log(10!=='10'); //true

>	Greater than	It checks whether the value of the left operands is greater than the value of the right operand or not.	let a = 30;
			let b = 20;
			console.log(a>b); //true
			console.log(a>30); //false
			console.log(20> 20'); //false

Operator	Operator_Name	Description	Example

>=	Greater than or equal to	It checks whether the value of the left operands is greater than or equal to the value of the right operand or not.	let a = 20;
			let b = 20;
			console.log(a>=b); //true
			console.log(a>=30); //false
			console.log(20>='20'); //true

<	Less than	It checks whether the value of the left operands is less than the value of the right operand or not.	let a = 10;
			let b = 20;
			console.log(a<b); //true
			console.log(a<10); //false
			console.log(10<'10'); //false

<=	Less than or equal to	It checks whether the value of the left operands is less than or equal to the value of the right operand or not.	let a = 10;
			let b = 20;
			console.log(a<=b); //true
			console.log(a<=10); //true
			console.log(10<='10'); //true

2.7.3 Logical Operators

Logical operators are used for combining two or more condition into a single expression and return the Boolean result true or false. The Logical operators are given below.

Operator	Operator_Name	Description	Example

&&	Logical AND	It returns true if both the operands(expression) are true, otherwise returns false.	let a = false;
			let b = true;
			console.log(a&&b); /false
			console.log(b&&true); //true
			console.log(b&&10); //10 which is also 'true'
			console.log(a&&'10'); //false

||	Logical OR	It returns true if any of the operands(expression) are true, otherwise returns false.	let a = false;
			let b = true;
			console.log(a||b); //true
			console.log(b||true); //true
			console.log(b||10); //true
			console.log(a||'10'); //'10' which is also 'true'

!	Logical NOT	It returns the inverse result of an operand(expression).	let a = 20;
			let b = 30;
			console.log(!true); //false
			console.log(!false); //true
			console.log(!a); //false
			console.log(!b); /false
			console.log(!null); //true

2.7.4 Bitwise Operators

The bitwise operators perform the bitwise operations on operands. The bitwise operators are as follows.

Operator	Operator_Name	Description	Example

&	Bitwise AND	It returns the result of a Boolean AND operation on each bit of its integer arguments.	let a = 2;
			let b = 3;
			let c = a & b;
			console.log(c);
			//Output2

|	Bitwise OR	It returns the result of a Boolean OR operation on each bit of its integer arguments.	let a = 2;
			let b = 3;
			let c = a | b;
			console.log(c); // Output 3

^	Bitwise XOR	It returns the result of a Boolean Exclusive OR operation on each bit of its integer arguments.	let a = 2;
			let b = 3;
			let c = a ^ b;
			console.log(c); //Output1

~	Bitwise NOT	It inverts each bit in the operands.	let a = 2;
			let c = ~ a;
			console.log(c); //Output-3

>>	Bitwise Right Shift	The left operand's value is moved to the right by the number of bits specified in the right operand.	let a = 2;
			let b = 3;
			let c = a >> b;
			console.log(c); //Output0

<<	Bitwise Left Shift	The left operand's value is moved to the left by the number of bits specified in the right operand. New bits are filled with zeroes on the right side.	let a = 2;
			let b = 3;
			let c = a << b;
			console.log(c); //Output16

>>>	Bitwise Right Shift with Zero	The left operand's value is moved to the right by the number of bits specified in the right operand and zeroes are added on the left side.	let a = 3;
			let b = 4;
			let c = a >>> b;
			console.log(c); //Output0

2.7.5 Assignment Operators

Assignment operators are used to assign a value to the variable. The left side of the assignment operator is called a variable, and the right side of the assignment operator is called a value. The data-type of the variable and value must be the same otherwise the compiler will throw an error. The assignment operators are as follows.

Operator	Operator_Name	Description	Example

=	Assign	It assigns values from right side to left side operand.	let a = 10;
			let b = 5;
			console.log("a=b:" +a); //
			Output 10

+=	Add and assign	It adds the left operand with the right operand and assigns the result to the left side operand.	let a = 10;
			let b = 5;
			let c = a += b;
			console.log(c); //Output15

-=	Subtract and assign	It subtracts the right operand from the left operand and assigns the result to the left side operand.	let a = 10;
			let b = 5;
			let c = a -= b;
			console.log(c); //Output 5

*=	Multiply and assign	It multiplies the left operand with the right operand and assigns the result to the left side operand.	let a = 10;
			let b = 5;
			let c = a *= b;
			console.log(c); //Output 50

/=	Divide and assign	It divides the left operand with the right operand and assigns the result to the left side operand.	let a = 10;
			let b = 5;
			let c = a /= b;
			console.log(c); //Output2

%=	Modulus and assign	It divides the left operand with the right operand and assigns the result to the left side operand.	let a = 16;
			let b = 5;
			let c = a %= b;
			console.log(c); //Output1

Ternary/Conditional Operator

The conditional operator takes three operands and returns a Boolean value based on the condition, whether it is true or false. Its working is similar to an if-else statement. The conditional operator has right-to-left associativity. The syntax of a conditional operator is given below.

expression ? expression-1 : expression-2;

- expression: It refers to the conditional expression.
- expression-1: If the condition is true, expression-1 will be returned.
- expression-2: If the condition is false, expression-2 will be returned.

Example

```
1. let num = 16;
2. let result = (num > 0) ? "True":"False"
3. console.log(result);
```

Concatenation Operator:

The concatenation (+) operator is an operator which is used to append the two string. In concatenation operation, we cannot add a space between the strings. We can concatenate multiple strings in a single statement. The following example helps us to understand the concatenation operator in TypeScript.

```
let message = "Welcome to " + "JavaTpoint";
console.log("Result of String Operator: " +message);
```

Type Operators

There are a collection of operators available which can assist you when working with objects in TypeScript. Operators such as typeof, instanceof, in, and delete are the examples of Type operator. The detail explanation of these operators is given below.

Operator_Name	Description	Example
In	It is used to check for the existence of a property on an object.	let Bike = {make: 'Honda', model: 'CLIQ', year: 2018}; console.log('make' in Bike); // Output:true

delete	It is used to delete the properties from the objects.	let Bike = { Company1: 'Honda',
		Company2: 'Hero',
		Company3: 'Royal Enfield'
		};
		delete Bike.Company1;
		console.log(Bike);
		//Output:{ Company2: 'Hero', Company3: 'Royal Enfield' }

typeof	It returns the data type of the operand.	let message = "Welcome to " + "Techneo";
		console.log(typeof message);
		//Output:String

instanceof	It is used to check if the object is of a specified type or not.	let arr = [1, 2, 3];
		console.log(arrinstanceof Array); // true
		console.log(arrinstanceof String); // false

💎 Decision Making:

The decision making in a programming language is similar to decision making in real life. In a programming language, the programmer uses decision making for specifying one or more conditions to be evaluated by the program. The decision making always returns the Boolean result true or false.

if statement

It is a simple form of decision making. It decides whether the statements will be executed or not, i.e., it checks the condition and returns true if the given condition is satisfied.

Syntax:

```
if(condition) {
    // code to be executed
}
```

```
let a = 10, b = 20;
if (a < b)
{
    console.log('a is less than b.');
}
```

if-else statement

The if statement only returns the result when the condition is true. But if we want to returns something when the condition is false, then we need to use the if-else statement. The if-else statement tests the condition. If the condition is true, it executes if block and if the condition is false, it executes the else block.

```
if(condition) {
  // code to be executed
} else {
   // code to be executed
}
```

```
let n = 10
if (n > 0) {
    console.log("The input value is positive Number: " +n);
} else {
     console.log("The input value is negative Number: " +n);
}
```

if-else-if ladder

Here a user can take decision among multiple options. It starts execution in a top-down approach. When the condition gets true, it executes the associated statement, and the rest of the condition is bypassed. If it does not find any condition true, it returns the final else statement.

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
else{
//code to be executed if all the conditions are false
}
```

```
let marks = 95;
if(marks<50){
    console.log("fail");
}
else if(marks>=50 && marks<60){
    console.log("D grade");
}
else if(marks>=60 && marks<70){
    console.log("C grade");
}
else if(marks>=70 && marks<80){
    console.log("B grade");
}
else if(marks>=80 && marks<90){
    console.log("A grade");
}else if(marks>=90 && marks<100){
    console.log("A+ grade");
}else{
    console.log("Invalid!");
}
```

Nested if statement

Here, the if statement targets another if statement. The nested if statement means if statement inside the body of another if or else statement.

Syntax

```
if(condition1) {
    //Nested if else inside the body of "if"
    if(condition2) {
        //Code inside the body of nested "if"
    }
    else {
        //Code inside the body of nested "else"
    }
}
else {
    //Code inside the body of "else."
}
```

```
let n1 = 10, n2 = 22, n3 = 25
if (n1 >= n2) {
    if (n1 >= n3) {
        console.log("The largest number is: " +n1)
    }
    else {
        console.log("The largest number is: " +n3)
    }
}
else {
    if (n2 >= n3) {
        console.log("The largest number is: " +n2)
    }
    else {
        console.log("The largest number is: " +n3)
    }
}
```

⚙TypeScript Inheritance:

Inheritance is an aspect of OOPs languages, which provides the ability of a program to create a new class from an existing class. It is a mechanism which acquires the properties and behaviours of a class from another class. The class whose members are inherited is called the base class, and the class that inherits those members is called the derived/child/subclass. In child class, we can override or modify the behaviours of its parent class.

Before ES6, JavaScript uses functions and prototype-based inheritance, but TypeScript supports the class-based inheritance which comes from ES6 version. The TypeScript uses class inheritance through the extends keyword. TypeScript supports only single inheritance and multilevel inheritance. It doesn't support multiple and hybrid inheritance.

```
SYNTAX:
class sub_class_name extends super_class_name
{
        // methods and fields
{
```

🔨Why use inheritance?

We can use it for Method Overriding (so runtime polymorphism can be achieved).

We can use it for Code Reusability.

Types of Inheritance:

We can classify the inheritance into the five types. These are:

- Single Inheritance
- Multilevel Inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Single Inheritance:

Single inheritance can inherit properties and behaviour from at most one parent class. It allows a derived/subclass to inherit the properties and behaviour of a base class that enable the code reusability as well as we can add new features to the existing code. The single inheritance makes the code less repetitive.

```
1.  class Shape {
2.      Area:number
3.      constructor(area:number) {
4.          this.Area = area
5.      }
6.  }
7.  class Circle extends Shape {
8.      display():void {
9.          console.log("Area of the circle: "+this.Area)
10.     }
11. }
12. var obj = new Circle(320);
13. obj.display()
```

Multilevel Inheritance:

When a derived class is derived from another derived class, then this type of inheritance is known as multilevel inheritance. Thus, a multilevel inheritance has more than one parent class. It is similar to the relation between Grandfather, Father, and Child.

```
1.  class Animal {
2.      eat():void {
3.          console.log("Eating")
4.      }
5.  }
6.  class Dog extends Animal {
7.      bark():void {
8.          console.log("Barking")
9.      }
10. }
11. class BabyDog extends Dog{
12.     weep():void {
13.         console.log("Weeping")
14.     }
15. }
16. let obj = new BabyDog();
17. obj.eat();
18. obj.bark();
19. obj.weep()
```

Multiple Inheritance:

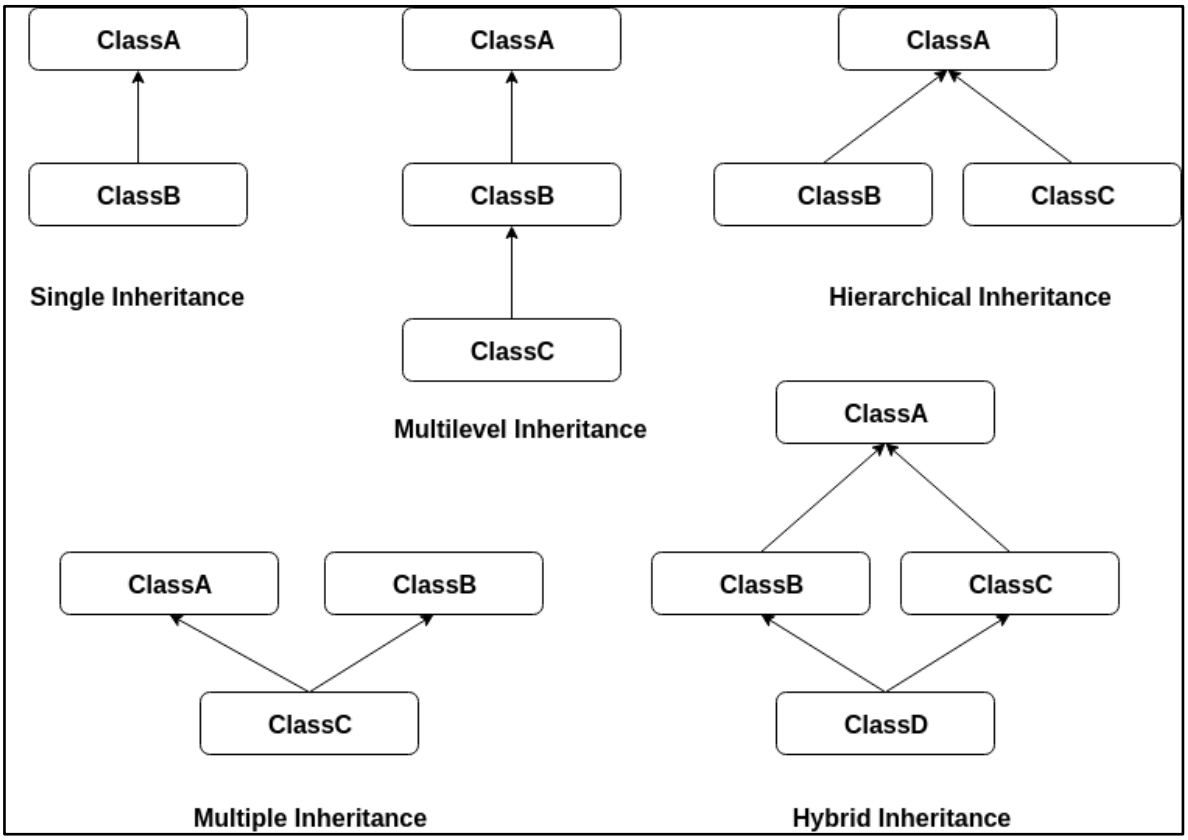
When an object or class inherits the characteristics and features form more than one parent class, then this type of inheritance is known as multiple inheritance. Thus, a multiple inheritance acquires the properties from more than one parent class. TypeScript does not support multiple inheritance.

Hierarchical Inheritance:

When more than one subclass is inherited from a single base class, then this type of inheritance is known as hierarchical inheritance. Here, all features which are common in sub-classes are included in the base class. TypeScript does not support hierarchical inheritance.

Hybrid Inheritance:

When a class inherits the characteristics and features from more than one form of inheritance, then this type of inheritance is known as Hybrid inheritance. In other words, it is a combination of multilevel and multiple inheritance. We can implement it by combining more than one type of inheritance. TypeScript does not support hybrid inheritance.



TypeScript Vs. JavaScript

SN	JavaScript	TypeScript
1.	It doesn't support strongly typed or static typing.	It supports strongly typed or static typing feature.
2.	Netscape developed it in 1995.	Anders Hejlsberg developed it in 2012.
3.	JavaScript source file is in ".js" extension.	TypeScript source file is in ".ts" extension.
4.	It is directly run on the browser.	It is not directly run on the browser.
5.	It is just a scripting language.	It supports object-oriented programming concept like classes, interfaces, inheritance, generics, etc.
6.	It doesn't support optional parameters.	It supports optional parameters.
7.	It is interpreted language that's why it highlighted the errors at runtime.	It compiles the code and highlighted errors during the development time.
8.	JavaScript doesn't support modules.	TypeScript gives support for modules.
9.	In this, number, string are the objects.	In this, number, string are the interface.
10.	JavaScript doesn't support generics.	TypeScript supports generics.

11.	**Example:**
	`<script>`
	`function addNumbers(a, b) {`
	`  return a + b;`
	`}`
	`var sum = addNumbers(15, 25);`
	`document.write('Sum of the numbers is: ' + sum);`
	`</script>`
		Example:
		`function addNumbers(a, b) {`
		`  return a + b;`
		`}`
		`var sum = addNumbers(15, 25);`
		`console.log('Sum of the numbers is: ' + sum);`