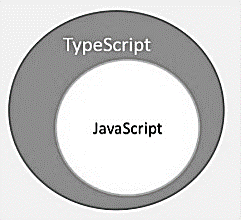
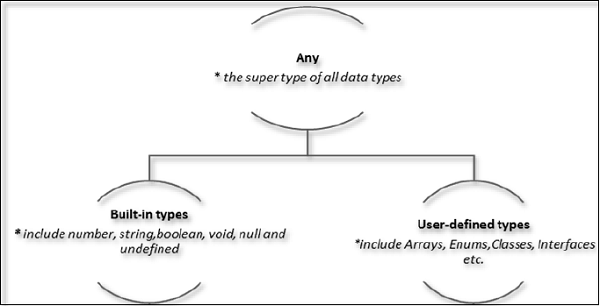
**TypeScript**

What is TypeScript?

By definition, “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 both a language and a set of tools. TypeScript is a typed superset of JavaScript compiled to JavaScript. In other words, TypeScript is JavaScript plus 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.

|  |  |  |
| --- | --- | --- |
| **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); |

============== First example on .ts ===============================

var message:string = "Hello World"

console.log(message)

On compiling, it will generate following JavaScript code.

//Generated by typescript 1.8.10

var message = "Hello World";

console.log(message);

Example: Variables in TypeScript

var name:string = "John";

var score1:number = 50;

var *score2*:number = 42.50

var sum = score1 + score2

console.log("name"+name)

console.log("first score: "+score1)

console.log("second score: "+score2)

console.log("sum of the scores: "+sum)

Type Assertion in TypeScript

TypeScript allows changing a variable from one type to another. TypeScript refers to this process as Type Assertion. The syntax is to put the target type between < > symbols and place it in front of the variable or expression. The following example explains this concept −

Example

var str = '1'

var str2:number = <number> <any> str //str is now of type number

console.log(typeof(str2))

Inferred Typing in TypeScript

Given the fact that, Typescript is strongly typed, this feature is optional. TypeScript also encourages dynamic typing of variables. This means that, TypeScript encourages declaring a variable without a type. In such cases, the compiler will determine the type of the variable on the basis of the value assigned to it. TypeScript will find the first usage of the variable within the code, determine the type to which it has been initially set and then assume the same type for this variable in the rest of your code block.

The same is explained in the following code snippet −

Example: Inferred Typing :- Loose couple data type

var num = 2; // data type inferred as number

console.log("value of num "+num);

num = "12";

console.log(num);

=============== Operators

The negation operator (-)

Changes the sign of a value. Let’s take an example.

var x:number = 4

var y = -x;

console.log("value of x: ",x); //outputs 4

console.log("value of y: ",y); //outputs -4

unary operator :- the operator that works with one operand only.

X++ y++ ++y --z z—

Binary operator :- the operator that works with two operands

X+y;

Arithmetic Assignment operator

X+=y; x=x+y;

Ternary operator :- shortcut of if-else

bal=amt>5000?amt-2000:amt;

if(amt>5000 ){ bal=amt-2000;} else{bal=amt;}

Conditional Operator (?)

This operator is used to represent a conditional expression. The conditional operator is also sometimes referred to as the ternary operator. The syntax is as given below ?

Test ? expr1 : expr2

Test :- refers to the conditional expression

? expr1 :- value returned if the condition is true

: expr2 value returned if the condition is false

var num:number = -2

var result = num > 0 ?"positive":"non-positive"

console.log(result)

======================= Functions

Example: Optional Parameters

function disp\_details(id:number,name:string,mail\_id?:string) {

console.log("ID:", id);

console.log("Name",name);

if(mail\_id!=undefined)

console.log("Email Id",mail\_id);

}

disp\_details(123,"nandini");

disp\_details(111,"vinayak","vinu@gmail.com");

Rest Parameters : param arrays, varargs… is called in Java.

Rest parameters are similar to variable arguments in Java. Rest parameters don’t restrict the number of values that you can pass to a function. However, the values passed must all be of the same type. In other words, rest parameters act as placeholders for multiple arguments of the same type.

To declare a rest parameter, the parameter name is prefixed with three periods. Any nonrest parameter should come before the rest parameter.

function addNumbers(int[] nums…) {} //declaration of varargs in java

Example: Rest Parameters

function addNumbers(...nums:number[]) {

var i;

var sum:number = 0;

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

sum = sum + nums[i];

}

console.log("sum of the numbers",sum)

}

addNumbers(4);

addNumbers(1,2,3)

addNumbers(10,10,10,10,10)

Example ─ A Simple Anonymous function

var msg = function() {

return "hello world";

}

console.log(msg())

## Lambda Functions

Lambda refers to anonymous functions in programming. Lambda functions are a concise mechanism to represent anonymous functions. These functions are also called as **Arrow functions**.

### Lambda Function - Anatomy

There are 3 parts to a Lambda function −

* **Parameters** − A function may optionally have parameters
* **The fat arrow notation/lambda notation (=>)** − It is also called as the goes to operator
* **Statements** − represent the function’s instruction set

### Lambda Expression

It is an anonymous function expression that points to a single line of code. Its syntax is as follows −

( [param1, parma2,…param n] )=>statement;

### Example: Lambda Expression

var msg = (x:number)=>10 + x

console.log(msg(100)) //outputs 110

## Function Overloads

Functions have the capability to operate differently on the basis of the input provided to them. In other words, a program can have multiple methods with the same name with different implementation. This mechanism is termed as Function Overloading. TypeScript provides support for function overloading.

function disp(s1:string):void;

function disp(n1:number,s1:string):void;

function disp(x:any,y?:any):void {

console.log(x);

console.log(y);

}

disp("abc")

disp(1,"xyz");

============================================================

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: TypeScript and Object Orientation

class Greeting {

greet():void {

console.log("Hello World!!!")

}

}

var obj = new Greeting();

obj.greet();