TypeScript

TypeScript is an open-source object-oriented language developed and maintained by Microsoft, licensed under Apache 2 license. It is a typed superset of Javascript that compiles to plain JavaScript. TypeScript was developed under Anders Hejlsberg, who also led the creation of the C# language. TypeScript was first released in October 2012.

npm install -g typescript

tsc -v

npm install typescript --save-dev

## 

## TypeScript Features

* **Cross-Platform:** TypeScript runs on any platform that JavaScript runs on. The TypeScript compiler can be installed on any Operating System such as Windows, macOS, and Linux.
* **Object-Oriented Language:** TypeScript provides powerful features such as Classes, Interfaces, and Modules. You can write pure object-oriented code for client-side as well as server-side development.
* **Static type-checking:** TypeScript uses static typing. This is done using type annotations. It helps type checking at compile time. Thus, you can find errors while typing the code without running your script each time. Additionally, using the type inference mechanism, if a variable is declared without a type, it will be inferred based on its value.
* **Optional Static Typing:** TypeScript static typing is optional, if you prefer to use JavaScript's dynamic typing.
* **DOM Manipulation:** Like JavaScript, TypeScript can be used to manipulate the DOM.
* **ES 6 Features:** TypeScript includes most features of planned [ECMAScript](https://www.tutorialsteacher.com/articles/what-is-ecmascript) 2015 (ES 6, 7) such as class, interface, Arrow functions etc.

**TypeScript Advantages**

1. TypeScript is an open-source language with continuous development and maintenance by Microsoft.
2. TypeScript runs on any browser or JavaScript engine.
3. TypeScript is similar to JavaScript and uses the same syntax and semantics. All of TypeScript's code finally gets converted into JavaScript. This allows a quicker learning curve for front-end developers currently coding in JavaScript.
4. TypeScript is also closer in syntax to backend languages like Java and Scala. This helps backend developers write front-end code faster.
5. TypeScript code can be called from an existing JavaScript code. TypeScript also works with existing JavaScript frameworks and libraries without any issues.
6. The TypeScript Definition file, with .d.ts extension, provides support for existing JavaScript libraries like Jquery, D3.js, etc. So, TypeScript code can add JavaScript libraries using type definitions to avail the benefits of type-checking, code autocompletion, and documentation in existing dynamically-typed JavaScript libraries.
7. TypeScript has support for the latest JavaScript features from [ECMAScript](https://www.tutorialsteacher.com/articles/what-is-ecmascript) 2015 . It includes features from ES6 and ES7 that can run in ES5-level JavaScript engines like Node.js. This offers a massive advantage of using features from future JavaScript versions in current JavaScript engines.
8. TypeScript has easy integration with task runner tools like Grunt and Gulp to automate the workflow.

# TypeScript - Type Annotations

TypeScript is a typed language, where we can specify the type of the variables, function parameters and object properties. We can specify the type using :Type after the name of the variable, parameter or property. There can be a space after the colon. TypeScript includes all the primitive types of JavaScript- number, string and boolean.

id:number,

name:string

# TypeScript Data Type

# Number

let first:number = 123; // number

let second: number = 0x37CF; // hexadecimal

let third:number=0o377 ; // octal

let fourth: number = 0b111001;// binary

# String

let employeeName:string = 'John Smith';

# Boolean

let isActive:boolean = true;

# Arrays

let fruits: string[] = ['Apple', 'Orange', 'Banana'];

let fruits: Array<string> = ['Apple', 'Orange', 'Banana'];

let fruits: Array<string>;

fruits = ['Apple', 'Orange', 'Banana'];

# Multi Type Arrays:

let values: (string | number)[] = ['Apple', 2, 'Orange', 3, 4, 'Banana'];

// or

let values: Array<string | number> = ['Apple', 2, 'Orange', 3, 4, 'Banana'];

# Tuples

TypeScript introduced a new data type called Tuple. Tuple can contain two values of different data types.

var empId: number = 1;

var empName: string = "Steve";

// Tuple type variable

var employee: [number, string] = [1, "Steve"];

var employee: [number, string] = [1, "Steve"];

var person: [number, string, boolean] = [1, "Steve", true];

var user: [number, string, boolean, number, string];// declare tuple variable

user = [1, "Steve", true, 20, "Admin"];// initialize tuple variable

var employee: [number, string][];

employee = [[1, "Steve"], [2, "Bill"], [3, "Jeff"]];

**accessing:**

var employee: [number, string] = [1, "Steve"];

employee[0]; // returns 1

employee[1]; // returns "Steve"

# Enum

enums allow us to declare a set of named constants i.e. a collection of related values that can be numeric or string values.

There are three types of enums:

1. Numeric enum
2. String enum
3. Heterogeneous enum

enum PrintMedia {

Newspaper = 1,

Newsletter = getPrintMediaCode('newsletter'),

Magazine = Newsletter \* 3,

Book = 10

}

function getPrintMediaCode(mediaName: string): number {

if (mediaName === 'newsletter') {

return 5;

}

}

enum PrintMedia {

Newspaper = "NEWSPAPER",

Newsletter = "NEWSLETTER",

Magazine = "MAGAZINE",

Book = "BOOK"

}

// Access String Enum

PrintMedia.Newspaper; //returns NEWSPAPER

PrintMedia['Magazine'];//returns MAGAZINE

enum Status {

Active = 'ACTIVE',

Deactivate = 1,

Pending

}

# Union

let code: (string | number);

code = 123; // OK

code = "ABC"; // OK

code = false; // Compiler Error

let empId: string | number;

empId = 111; // OK

empId = "E111"; // OK

empId = true; // Compiler Error

# Any

let something: any = "Hello World!";

something = 23;

something = true;

# Void

function sayHi(): void {

console.log('Hi!')

}

let speech: void = sayHi();

console.log(speech); //Output: undefined

# Never

function throwError(errorMsg: string): never {

throw new Error(errorMsg);

}

function keepProcessing(): never {

while (true) {

console.log('I always does something and never ends.')

}

}

default parameters:

// Default Parameters  
sayHello(hello: string = 'hello') {   
 console.log(hello);   
}sayHello(); // Prints 'hello'sayHello('world'); // Prints 'world'

optional parameters:

// Optional Parameters  
sayHello(hello?: string) {   
 console.log(hello);   
}sayHello(); // Prints 'undefined'sayHello('world'); // Prints 'world'

# Nullish coalescing operator (??)

The **nullish coalescing operator (??)** is a logical operator that returns its right-hand side operand when its left-hand side operand is [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/null) or [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined), and otherwise returns its left-hand side operand.

This can be contrasted with the [logical OR (||) operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_OR), which returns the right-hand side operand if the left operand is any [falsy](https://developer.mozilla.org/en-US/docs/Glossary/Falsy) value, not only null or undefined. In other words, if you use || to provide some default value to another variable foo, you may encounter unexpected behaviors if you consider some falsy values as usable (e.g., '' or 0). See below for more examples.

The nullish coalescing operator has the fifth-lowest [operator precedence](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence), directly lower than || and directly higher than the [conditional (ternary) operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator).

const foo = null ?? 'default string';

console.log(foo);

// expected output: "default string"

const baz = 0 ?? 42;

console.log(baz);

// expected output: 0

# Logical nullish assignment (??=)

The logical nullish assignment (x ??= y) operator only assigns if x is [nullish](https://developer.mozilla.org/en-US/docs/Glossary/Nullish) (null or undefined).

x ?? (x = y);

x = x ?? y;