



1-0 Intro to typescript technocrat mission

What we will be learning & doing



1. Introduction of typescript
2. Basic and advanced types of typescript.
3. Normal function & arrow function.
4. Generic and Interface
5. Modules & namespaces.
6. Object Oriented Typescript.

1-1 Introduction to Typescript

What is Typescript

Typescript is an Object Oriented Programming Language that is built on top of **JS**
With Extra Fetcher

Why Typescript?

Lacking in JavaScript

- Dynamically Typed Language

```
1 // String
2 let a = 'Hello JavaScript!';
3
4 // Number
5 a = 100;
6
7 // Array
8 a = [1, 2, 3, 4, 5, 6];
9
10 // Object
11 a = {
12     name : "SIFAT ULLAH SHOYON",
13     profession: 'Full Stack Web Developer'
14 };
15
16 // Function
17 a = function (){
18     return 'Hello JavaScript!';
19 };
```

আমরা সবাই জানি JavaScript হল একটি ডায়নামিক টাইপ ল্যাংগুয়েজ। এই ডায়নামিক টাইপ ল্যাংগুয়েজ হওয়ার কাড়নে আমরা যে, কোন সময় যে কোন টাইপ এর ভেরিয়েবল ডিকলার্ড করতে পারি। আমরা চাইলে একটি string variable এ (number, array, object & function) ও রাখতে পারি এতে JavaScript মাইন্ড করে না। ছোট্ট স্কেল এর প্রজেক্ট এ এই **type tacking error** টা তেমন একটা সমস্যা না হলেও বড় প্রজেক্ট এ এই টাইপ error গুলো সহজে ধরা যায় না যতক্ষণ পর্যন্ত কোড গুলো রান টাইমে না জায় (ব্রাউজারে বা নোডজেএসে)। যার জন্য বলা হয় বড় প্রজেক্টে এ JavaScript দিয়ে কোড করা খুবই বেশি ডিফিকাল্ট। তাই TypeScript দিয়ে কাজ করলে এই type tacking error গুলো শলভ করার জন্য run time এ যাওয়ার প্রয়োজন পড়ে না বাগ গুলো খুবই সহজে ফিক্সড করা যায়।

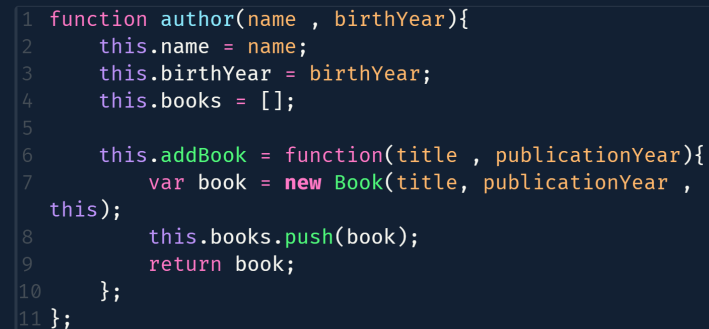
JavaScript এ বাই ডিফল্ট object oriented programming (

OOP) সাপোর্ট করে না আমরা **syntactic sugar** এর মাধ্যমে JavaScript এ আমরা object oriented programming এর কোড লিখি।

When Working on a large Application With Multiple Developers

- Very difficult to maintain a Large Codebase
- Hard to find bugs
- Catch errors only in runtime

Need es3 support in older browsers but do you like modern syntax?



```
1 function author(name , birthYear){
2   this.name = name;
3   this.birthYear = birthYear;
4   this.books = [];
5
6   this.addBook = function(title , publicationYear){
7     var book = new Book(title, publicationYear ,
8     this);
9     this.books.push(book);
10    return book;
11  };
12 }
```

Solution:

TypeScript code can be transpiled into older versions of JavaScript

Can Browser Really Recognized TypeScript

- No
- Converting JS to Ts

দিন শেষে typescript তো JavaScript এ রূপান্তরিত হয়। typescript এ বলে দেওয়া যায় যে, typescript এর কোন ভার্সন এ রূপান্তরিত হবে। যদি বলা হয় তুমি es3 বা es5 এ রূপান্তরিত হবে।

তাহলে সে ওই ভার্সনেই রূপান্তর হবে। এই পদ্ধতিকে বলা হয় **Task File**।

Benefits of Using TypeScript

- Supports Older Browser
- Type Safety
- Increase Your Productivity
- Less Bugs and Less Testing

JS Types in TS	TS Own Types
Number	Interface
String	Void
Boolean	Array
Null	Tuple
Undefined	Enum
Object	Union
Symbol	Intersection

Drawbacks of using Typescript

- Type Complexities
- Limited Library Support
- Over Engineering
- Migration Challenges

1-2 install typescript and fast node version manager

Install Node Updated Version using NVM & Also Typescript [**npm install -g typescript**]

1-3 Write your first typescript program

প্রথমেই একটি ফাইল নিতে হবে যে কোনো নামই দেওয়া জেতে পারে উদাহরণস্বরূপ বলা যায় index.ts। .ts হচ্ছে Typescript ফাইল এর extension। Typescript এর কোড **node দিয়ে run করা যায় না**। Typescript এর কোড js এর কোড এ রূপান্তর করে তার পর কোড রান করতে হবে এর জন্য দরকার হবে একটি **Typescript কম্পাইলারের**। ts file এ কোড লিখার পর commend box এ গিয়ে Typescript এর কোড কম্পাইল করে নিতে হবে tsc index.ts(file name) এই কমান্ড দিয়ে। এই কমান্ড দেওয়ার পরে সাথে সাথে একই পথে একটি JavaScript file তৈরি হবে ওই খানে Typescript এর কোড কম্পাইল হয়ে JavaScript কোড এ রূপান্তর হয়ে গেছে।

আমি যদি না চাই যে, সেম ফোল্ডার এ js file & ts file এক সাথে থাকুক এবং ২ টা ফাইল এক সাথে রাখা ঠিক ও না তাহলে error দিবে। নতুন একটা ফোল্ডার তৈরি করবো module1 নামে এবং এর ভিতরে আড়ও একটি ফোল্ডার করবো src নামে (**module1/src/index.ts**) ওই ফোল্ডার এর ভিতরে সকল ts file থাকবে।

আমি যদি চাই যে, আমার সকল ts file গুলো ও আলাদা থাকবে তাহলে আমাকে Typescript এর configuration file এর দরকার হবে। Typescript এর configuration file তৈরি করতে হলে commend box এ গিয়ে **tsc --init** এই কমান্ড টি দিতে হবে। এই configuration file এ অনেক কিছুই করা যায়। যেহেতু বর্তমানে আমি পাথ টি ঠিক করবো সেই জন্যে আমি rootDir টি uncommand করে pathname select করে দিবো (**"rootDir" : "./module1/src/"**)। file অবশ্যই save করতে হবে।

আমি যদি চাই যে, আমার সকল js file গুলো compile হয়ে আলাদা থাকবে তাহলে আমাকে Typescript এর configuration file এর দরকার হবে। Typescript এর configuration file তৈরি করতে হলে commend box এ গিয়ে tsc --init এই কমান্ড টি দিতে হবে। এই configuration file এ অনেক কিছুই করা যায়। যেহেতু বর্তমানে আমি পাথ টি ঠিক করবো সেই জন্যে আমি outDir টি uncommand করে pathname select করে দিবো (**"outDir" : "./module1/dist/"**)। file অবশ্যই save করতে হবে।

এর পর commend box এ গিয়ে tsc এই কমান্ডটি দিলেই কনফিগারেশন ফাইলটি আমাদের কথা মতো সব কাজ করে দিবে। আলাদা আলাদা .ts file & .js file তৈরি হয়ে যাবে কথা মতো ফোল্ডারে।

আমি যদি চাই আমি আমার js code old ES5 বা তারও আগের version এর কোড এ রূপান্তর করবো তা ও করা যাবে শুধুমাত্র Typescript এর configuration file গিয়ে **"target" : "ES5"** করে দিলেই হয়ে যাবে। এইভাবে চাইলে সব ভার্সনেই কোড রূপান্তর করা যাবে।

1-4 Basic data types

Primitive Types & Non Primitive Types

Primitive	Non Primitive
number	Array
string	Tuple
Boolean	Object
null	
undefined	
symbol	

Implicit Data Type কি?

⇒ Typescript Compiler যদি নিজ থেকে variable এর ডাটা Assume করতে পারে তখন তাকে বলা হয় Implicit Data type ।

```
let fullName = 'Sifat Ullah Shoyon';
```

Explicit Data Type কি?

⇒ আমার যদি নিজ থেকে variable এর ডাটা টাইপ কি হবে তা বলে দেই তখন তাকে বলা হয় Explicit Data Type ।

```
let fullName : string = 'Sifat Ullah Shoyon';
```

Any Type Data বলতে কি বোঝায়?

⇒ একটি Variable এর যদি কোনো Type Define করা না থাকে তাহলে Typescript Compiler তাকে Assume করে নিবে Any Type Data হিসাবে। Any Type Data তে চাইলে যে কোন ডাটাই রাখা যায় যেমনঃ (number, string, Boolean, array, object, etc.) Any Type Data ব্যবহার করা উচিত নয় । এই ডাটা ব্যবহার করা মানে typescript off করে দিয়ে JavaScript এর কোড লিখা।

```
let d;

d = 123;
d = 'sifat';
d = false;
```

আমি যদি জানি যে, আমার variable এর ডাটা নাই কিন্তু variable এর টাইপ কি হবে তখন চাইলে variable এর টাইপ টি বলে দেওয়া যায় -

```
let d : number;
```

Array Data Type:

```
let friends : string[] = ['shohan' , 'mridul' , 'sabin' , 'alam'];
// friends.push(9); Error: Argument of type 'number' is not assignable to type 'string'.

let friendsRollNumber : number[] = [5 , 6, 9, 7, 3, 2, 1];
// friendsRollNumber.push('shoyon'); Argument of type 'string' is not assignable to type 'number'.
```

Tuples কি?

Tuples অনেকটা array এর মতোই কিন্তু এইখানে order টা গুরুত্বপূর্ণ type এর ক্ষেত্রে। Mix data নিয়ে কাজ করতে হলে এই Tuples টি ব্যবহৃত হয় যেমন: key , value; Tuples এ push() method টি ব্যবহার করা উচিত নয়। tuple multiple জিনিস নিয়ে থাকতে পারে -

```
// tuple --> array --> order --> type of values

let coordinates : [number , number] = [1,5];

let ageName : [number , string, boolean] = [50, 'Mr.x', true];

ageName[0] = 67;

// ageName[1] = 96; error : Type 'number' is not assignable to type 'string'.

ageName[1] = 'Sifat';

ageName[2] = false;
```

1-5 Object, Optional, and Literal Types

JavaScript এর মতো typescript এ ও object ডিকলার করা যায়। যদি obj এর key variable এর ডেটা টাইপ বলে দেওয়া হয় তখন প্রত্যেকটি type এর পরে দিতে হয় **সেমিকোলন(;)** কিন্তু value এর ক্ষেত্রে দিতে হয় **কমা(,)**।

এমন হতে পারে যে, একটা property এর মান সব ইউজার এর কাছে না ও থাকতে পারে তখন error এর হাত থেকে বাঁচার জন্য ব্যবহার করতে হয় **অপশনাল টাইপ(?)**।

যদি কখনো এমন কোন সিচুয়েশন তৈরি হয় যে, আমার value এর মান পরিবর্তন করা যাবে না তখন চাইলে type (string , number , Boolean , etc.) ব্যবহার না করে সরাসরি type ব্যবহার করা যায় যেননঃ company : "Day Dynamo's" । যখন একটা value type হিসাবে আঁচান করবে তখন তাকে বলা হয় **literal types**

আমরা যদি চাই কোন key এর type (string , number , Boolean , etc.) থাকবে কিন্তু আমার তার value পরিবর্তন করবো তা সেক্ষেত্রে আমার **readonly (property or access modifier)** ব্যবহার করতে পারি তাহলে আর value পরিবর্তন করা যাবে না যেমন:

```
readonly company: string;
```

```
// Reference Type --> Object
```

```
const user : {  
    // company: 'Dev Daynamos'; // type --> literal types  
    readonly company: string; // type --> literal types  
    firstName: string;  
    middleName?: string; //optional type  
    lastName: string;  
    isMarried: boolean;  
} = {  
    company: 'Dev Daynamos',  
    firstName : 'Sifat',  
    // middleName: 'Ullah',  
    lastName: 'Shoyon',  
    isMarried: false,  
};
```

```
// user.company = 'Daynamic Codars'; // Error Cannot assign to
```

```
// Readonly
```

```
const user1 : {  
    company: string; // type --> literal types  
    firstName: string;  
    middleName?: string; //optional type  
    lastName: string;  
    isMarried: boolean;
```



```

} = {
  company: 'Dev Daynamos',
  firstName : 'Sifat',
  // middleName: 'Ullah',
  lastName: 'Shoyon',
  isMarried: false,
};

user1.company = 'PH';

```

1-6 Function in typescript

function 2 ধরনের যথাঃ

- ১) normal function
- ২) arrow function

object এর ভিতরে যদি কোন একটি function থাকে সেটিকে আমরা বলি **Method**। নরমাল js এর ফাংশন এ যেই কাজ গুলো করা যেতো Typescript এ ও সেই সকল ধরনের কাজ করা যায়। কোন object এর ভিতরের কোন property কে যদি access করতে হয় function এর ভিতরে তাহলে **this keyword** ব্যবহার করতে হয় আর this keyword normal function এ কাজ করে arrow function এ কাজ করে না।

```

// normal function
function add(num1 : number, num2 : number = 12) : number {
  return num1 + num2;
};

// console.log(add(5,7));

// arrow function
const addArrow = (num1 : number, num2 : number) : number => num1 + num2;

// console.log(addArrow(20,24));

```

```
// object --> function --> method

const poorUser = {
  name : 'sifat',
  balance : 0,
  addBalance(balance : number) : number{
    return this.balance + balance;
  },
};

// console.log('poor user' , poorUser);

const poorUser1 = {
  name : 'sifat',
  balance : 0,
  addBalance(balance : number) : string {
    return `This is my new ${this.balance + balance} balance`;
  },
};

// console.log('poor user one' , poorUser1.addBalance(50));

const array : number[] = [5,9,42,15];

const newArray : number[] = array?.map((element : number) : number);

// console.log('array' , array);

// console.log('new array' , newArray);
```

1-7 Spread and Rest Operator

Rest Operator সব গুলো element কে নিয়ে একটি নতুন array তৈরি করে এবং **Spread Operator** array থেকে element গুলো কে বাদ দেয়।

```

// spread operator

// array
const friends1: string[] = ["mridul", "shohan", "sabin", "alar"];

const friends2: string[] = ["billa", "nazmul", "shuvo"];

friends1.push(...friends2);

// console.log(friends1);

// object
const mentors1 = {
  typeScript: "Mezba",
  redax: "Mir",
  dbms: "mizan",
};

const mentors2 = {
  prisma: "Firoz",
  next: "tanmoy",
  cloud: "nahid",
};

const mentorList = {
  ...mentors1,
  ...mentors2,
};

// console.log('mentors 1' , mentors1);
// console.log('mentors 2' , mentors2);
// console.log('mentors list' , mentorList);

// rest operator

```

```
const myFriends = (frend1: string, frend2: string, frend3: string) => {
    // return [frend1, frend2, frend3];
    console.log(frend1, frend2, frend3);
};

// console.log('my friend' , myFriends("a" , "b" , "c"));

const myFriends1 = (...rest: string[]) => {
    rest.forEach((friend: String) => {
        console.log("Hello", friend);
    });
};

// console.log("my friend", myFriends1("a", "b", "c", "d", "e"));
```

1-8 Destructuring in typescript

JavaScript এ সেই ভাবে Destructuring করা যায় একই ভাবে typescript ও করা যায়।

```

1 // Destructuring
2
3 // Object Destructuring
4
5 const users = {
6   _id: 569,
7   name: {
8     firstName: "SIFAT",
9     middleName: "ULLAH",
10    lastName: "SHOYON",
11  },
12  isDeveloper: true,
13  address: "Dhaka, BD",
14  contactNo: "0190000000",
15 };
16
17 const {
18   _id,
19   name: { firstName, middleName: midName, lastName },
20   isDeveloper,
21   address,
22   contactNo,
23 } = users;
24
25 // console.log(midName);
26
27 // Array Destructuring
28
29 const myFriends: string[] = [
30   "mridul",
31   "alamin",
32   "sabin",
33   "shohan",
34   "billa",
35   "nazmul",
36   "shuvo",
37 ];
38
39 const [a, , , bestFriend, masum, ...rest] = myFriends;
40
41 const bstFrnd = bestFriend;
42
43 console.log(bstFrnd);

```

object এ {address : string} property এর পরে value এর type দেওয়া যায় না কাড়ন এইটাকে js/ts **name aliasing** মনে করে, মানে একটি Value এর নাম নতুন করে re declared করাকে বোঝায়।

একই ভাবে array তে object এর মতো value name aliasing করা যায় না এক লাইন এ । যদি একসুই দরকার পরে তখন নতুন একটি variable নিয়ে পরের লাই এ করতে হবে। masum বা billa (billa এর name aliasing হচ্ছে masum) এর পরে যতগুলো বস্তু আছে সব গুলো একত্রে আনার জন্য short cut হচ্ছে এই **...rest** operator ।

```
// Destructuring

// Object Destructuring

const users = {
  _id: 569,
  name: {
    firstName: "SIFAT",
    middleName: "ULLAH",
    lastName: "SHOYON",
  },
  isDeveloper: true,
  address: "Dhaka, BD",
  contactNo: "01900000000",
};

const {
  _id,
  name: { firstName, middleName: midName, lastName },
  isDeveloper : sting, // এই ভাবে টাইপ ডিকলার্ড করা যাবে না
  address : sting, // এই ভাবে টাইপ ডিকলার্ড করা যাবে না
  contactNo,
} = users;

// console.log(midName);

// Array Destructuring
```

```
const myFriends: string[] = [
  "mridul",
  "alamin",
  "sabin",
  "shohan",
  "billa",
  "nazmul",
  "shuvo",
];

const [a, , , bestFriend, masum, ...rest] = myFriends;

const bstFrnd = bestFriend; // object এর মতো একলাইন এ করা যাবে

console.log(bstFrnd);
```

1-9 Type alias in typescript

Aliasing কি?

⇒ কোন একটি property এর value এর নাম re assign করাকে সাধারনত Aliasing বলা হয়।

Aliasing কয় প্রকার?

⇒ Aliasing সাধারনত ০২ প্রকার যথাঃ

1. Name Aliasing
2. Type Aliasing

. **Name Aliasing** মানে হলো value এর নাম re assign করা যেমনঃ

```
const {_id, fullName : fName} = user;
```

. **Type Aliasing** মানে হলো value এর type assign করা যেমনঃ

```
// Type Aliasing
```

```
// variable types
type Student = {
  name: string;
  age: number;
  gender: string;
  contactNo?: string;
  addressNo: string;
};

const student1: Student = {
  name: "Sifat",
  age: 36,
  gender: "Male",
  contactNo: "01600000000",
  addressNo: "ctg",
};

const student2: Student = {
  name: "Ullah",
  age: 21,
  gender: "Male",
  addressNo: "Dhk",
};

const student3: Student = {
  name: "shoyon",
  age: 40,
  gender: "Male",
  contactNo: "01400000000",
  addressNo: "ctg",
};

// string , boolean , function Types Aliasing

// Types Aliasing
type UserName = string;
```



```
// variable
const userName: UserName = "sifat ullah shoyon";

// Types Aliasing
type IsAdmin = boolean;
// Variable
const isAdmin: IsAdmin = true;

// Types Aliasing
type Add = (num1: number, num2: number) => number;
// Function
const add: Add = (num1, num2) => num1 + num2;
```

বার বার type declared করা ঝামেলা এড়ানোর জন্য এবং code কে re useable করার জন্য type কে সবার উপরে declared করতে হয় এ ক্ষেত্রে কিছু Rules মেনে কাজ টি করতে হয় যেমানঃ type এর পরে যেই মান টি থাকবে তার নাম শুরু করতে হবে **Capital Latter** দিয়ে।

```
type UserName = string;
```

1-10 Union and Intersection types

Union Types কি?

⇒ JavaScript এ or logic (||) বুঝার জন্য যে, sign ব্যবহৃত হতো ওই sign কেই বলা হয় Union Types কিন্তু typescript এ Union types এ ব্যবহৃত হয় মূলত একটি সিংগেল or (||) । মূলত একটি **single or sign (|)** কেই বলা হয় Union Types ।

String Literal Types কি?

⇒ Type এর ক্ষেত্রে Value হিসাবে String ব্যবহার করলে তাকে String Literal Types বলা হয়।

Intersection Types কি?

⇒ Intersection মানে হচ্ছে common type । JavaScript এ and logic (&&) বুঝার জন্য যে, sign ব্যবহৃত হতো ওই sign কেই বলা হয় Intersection Type কিন্তু typescript এ Intersection Type এ ব্যবহৃত হয় মূলত একটি সিংগেল and (&) । মূলত একটি **single and (&)** কেই বলা হয় Intersection Types ।

```
// Union And Intersection

// Union Type
type FrontEndDeveloper = "React Js" | "Angular Js";
type FullStackDeveloper = "Mern" | "Python";

type Developer = FrontEndDeveloper | FullStackDeveloper;

const newDeveloper: FrontEndDeveloper = "React Js";

type User = {
  name: string;
  email?: string;
  age: number;
  gender: "Male" | "Female";
  bloodGroup: "A+" | "B+" | "O+" | "AB+" | "AB-";
};

const user1: User = {
  name: "SIFAT",
  age: 28,
  gender: "Male",
  bloodGroup: "A+",
};

// Intersection Type

type OrdinaryGpDesigner = {
  skills: string[];
  designation1: "ordinaryGpDesigner";
};

type UiUxDesigner = {
  skills: string[];
  designation2: "uiUxDesigner";
};
```

```
};

type GrapichDesigner = OrdinaryGpDesigner & UiUxDesigner;

const grapichDesigner: GrapichDesigner = {
  skills: ["ps", "ai", "xd", "in"],
  designation1: "ordinaryGpDesigner",
  designation2: "uiUxDesigner",
};
```

1-11 Ternary, optional chaining & nullish coalescing operator

Question Mark (?) কে এক এক use case এ এক এক নামে ডাকা হয় এবং এদের কাজ ও আলাদা আলাদা হয়। যেমনঃ

Ternary Operator : শর্ত প্রয়োগ এর ক্ষেত্রে (if/else) এর short cut version হচ্ছে ternary operator ।

```
let age = 18;

const isAdult = age >= 18 ? "adult" : "not adult";
```

Optional Chaining : এমন অনেক সময় থাকে যে, আমার ওয়েবসাইটে এমন value দেখাই যা সব ইউজার এর থাকে না তখন error থেকে বাঁচতে আমার এই optional chaining কে ব্যবহার করে থাকি।

```
<p>{user?.emai}</p>
```

Nullish Coalescing Operator : এই operator টি null or undefined এর উপর ভিত্তি করে কোন একটা result return করে। যদি result null or undefined না হয় তখন সে কিছুই result output দেয় না। nullish coalescing operator ব্যবহার করার জন্য ০২ টি question mark (??) দিতে হয়।

ternary operator এবং nullish coalescing operator পরস্পর দেখতে একই মনে হলে ও তারা এক নয় । ternary operator সবসময় falsy value এর উপরে কাজ করে আর nullish

coalescing operator কাজ করে শুধু null or undefined এর উপরে।

```
// ternary operator || optional chaining || nullish coalescing

const age: number = 18;

if (age >= 18) {
  console.log("Adult");
} else {
  console.log("Not Adult");
}

// Ternary Operator
const isAdult = age >= 18 ? "Adult " : "NotAdult";

// console.log({isAdult});

const isAuthenticated = "";
// const isAuthenticated = null;
// const isAuthenticated = undefined;

const result = isAuthenticated ?? "Guest";

const result1 = isAuthenticated ? isAuthenticated : "user";

// console.log({result});
// console.log({result1});

// optional chaining operator

type UserInfo = {
  name: string;
  age: number;
  address: {
    city: string;
    street: string;
  };
}
```

```

        houseNumber?: string;
        presentAddress: string;
        permanentAddress?: string;
    };
    isDeveloper: boolean;
};

const userInfo: UserInfo = {
    name: "sifat",
    age: 23,
    address: {
        city: "Dhk",
        street: "Bamoil",
        presentAddress: "Dhaka",
    },
    isDeveloper: true,
};

const permanentAddress = userInfo?.address?.permanentAddress

// console.log({ permanentAddress: permanentAddress });

```

Tips & Trick:

Typescript এর কোড যেহেতু Node Envirment এ run করা যায় না তাই বার বার Typescript এর কোড কমপাইল করে JS এর কোডে রূপান্তর করে তার পর code run করতে হয়। বার বার tsc দাও এবং তার পরে node filePathName

এই কাজ টি সহজ করতে একটি packages ব্যবহার করা যায় যথাঃ-

```

npm i -g ts-node-dev
ts-node-dev --respawn --transpile-only filePathName

// Emample: ts-node-dev --respawn --transpile-only ./module-one/src/1.11.ts

```

1-12 Never, unknown and nullable type1-12

Never, unknown and nullable type

Nullable Types:

কোন একটি variable এর টাইপ যদি null হয় তখন তাকে nullable types বলা হয়।

Unknown Types:

বর্তমানে টাইপটির অবস্থা কি আছে তা আমরা জানি না কিন্তু রান টাইম এ গিয়ে জানা যাবে ওই রকম টাইপকে বলা হয় unknown types । উদাহরনঃ `typeof`

Never Types:

যেই function কোখনো কোন কিছু return করবে না সেই function এর type কে বলা হয় never types

```
{
  // Start

  // nullable types / unknown types

  // nullable types
  const searchName = (value: string | null) => {
    if (value) {
      console.log("Searching");
    } else {
      console.log("There is nothing to search");
    }
  };

  searchName(null);

  // unknown types

  const getSpeedInMeterPerSecond = (value: unknown) => {
    if (typeof value === "number") {
      const convertedSpeed = (value * 1000) / 3600;
      console.log(`The Speed is ${convertedSpeed} ms-1`);
    }
  };
}
```

```

    } else if (typeof value === "string") {
        const [result, unit] = value.split(" ");
        const convertedSpeed = (parseFloat(result) * 1000) / 3600;
        console.log(`The Speed is ${convertedSpeed} ms-1`);
    } else {
        console.log("Wrong Input");
    }
};

getSpeedInMeterPerSecond(true);

// never types

const throwError = (message: string): never => {
    throw new Error(message);
};

throwError("something went wrong");

// End
}

```

Tips & Trick:

Typescript শিখার শুরুর অবস্থায় সব গুলো Typescript file একই সাথে একই ফোল্ডারে থাকে। দিন শেষে সব গুলো ts file তো js file এ রূপান্তরিত হয়। index.ts file এ একটি user নামক variable আছে (const userName = 'Sifat'); কিন্তু অন্য একটি file যেমনঃ test.ts file এ ও user নামক variable থকে তখন আসলে js বুঝতে পারে না, সে এটি কে global scope মনে করে একটি ফাইল এর ভিতরেই সব আছে তাই error দেয়। তাই এই error থেকে বাঁচতে প্রত্যেকটি ts file এ কোড করার আগে একটি { } দিয়ে নিলে তখন আর error দেয় না তখন সে ভাবে এখন এটি একটি block scope এর ভিতরে আছে।