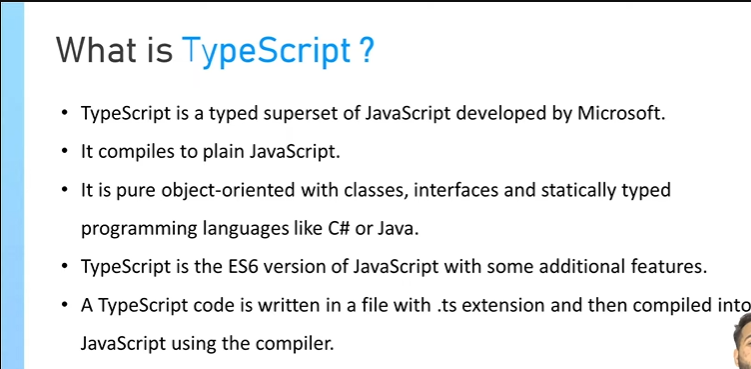
**Typescript**

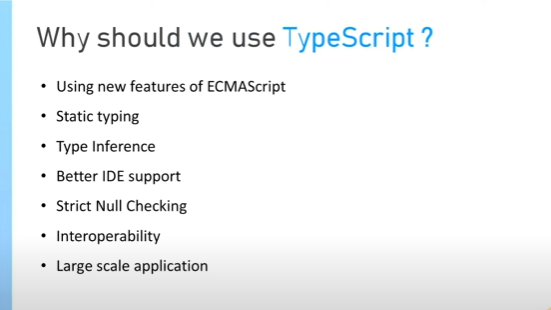


**Note:-**js does not on browser but I complie in js

**static** type means we have explicity mention the type of data.

**Features of ts**

It consist types, next-gen javscript Feature,non-js feature like interfaces or generics,meta programming like decorators, and rich configuration options



**Type inference:-**

Type inference in TypeScript is the ability of the TypeScript compiler to automatically determine the data types of variables when there is no explicit type annotation provided by the developer

// Example 1

let num = 5; // TypeScript infers the type of 'num' as number based on the assigned value.

console.log(typeof num); // Output: "number"

// Example 2

let message = "Hello, TypeScript!"; // TypeScript infers the type of 'message' as string.

console.log(typeof message); // Output: "string"

// Example 3

let array = [1, 2, 3, 4, 5]; // TypeScript infers the type of 'array' as number[].

console.log(Array.isArray(array)); // Output: true

**Better IDE support:-**

**->Type Inference and Autocompletion:** TypeScript's static typing allows IDEs to offer more accurate autocompletion suggestions and intelligent code analysis. With type information readily available, IDEs can provide context-aware suggestions, reducing the likelihood of errors and enhancing developer productivity.

**->Type Checking and Error Highlighting:** TypeScript's compiler performs static type checking, which enables IDEs to highlight potential type-related errors in real-time, even before the code is executed. This feature helps developers identify and rectify issues quickly, leading to more robust and reliable code.

**Strict Null Chechking:-** When strict null checking is enabled in TypeScript, variables are not automatically assigned the value of null or undefined.

if there is no key present and we are trying to access that then it throws error(while checking)

Let name:{

First:’alok’,

Last:’shakya’

}

Console.log(name.title) // it shows the red line

// Without strict null checking

let a: string;

// a = null; // No error without strict null checking

// With strict null checking

let b: string;

// b = null; // This will result in a compilation error

// Using union types to allow null or undefined

let c: string | null;

c = null; // No error with strict null checking

// Using the nullable modifier

let d: string | null = null; // No error with strict null checking

**Interoperabitlity:-** Interoperability in TypeScript refers to the ability of TypeScript to seamlessly work with existing JavaScript code and libraries. TypeScript is designed to be a superset of JavaScript, meaning that any valid JavaScript code is also valid TypeScript code. This feature allows developers to gradually adopt TypeScript within their projects without the need for a complete rewrite.

Suppose you have an existing JavaScript function that performs a simple calculation. You want to utilize this function in a new TypeScript file while taking advantage of TypeScript's type system.

// calc.js(js file)

function add(x, y) {

return x + y;

}

// app.ts(ts file)

// Using the JavaScript function in TypeScript

const result = add(5, 3); // Utilizing the 'add' function from calc.js

console.log(result); // Output: 8

**Q->Why do we need typescript compiler?**

We need the TypeScript compiler to convert TypeScript code into JavaScript code so that it can be executed in any JavaScript runtime environment, including web browsers and server-side environments. The TypeScript compiler, also known as tsc

**Q-> What happens when you compile?**

It will create a new file .js

**Q->What are the different types available in typescript?**

Boolean,Number,string,array,tuple(it is an array of fixed size where each element may have a differenct data type)

Enum(Represents a set of named constants, providing more expressive names to numeric values.)

**Q->Why does add('1', '2') start showing error when the trainer has defined types?**

Because we have set the data type number and we are passing the string that’s why it is showing error

**Q->What do you mean typescript can infer the types? What is the advantage of this?**

Type inference in TypeScript is the ability of the TypeScript compiler to automatically determine the data types of variables when there is no explicit type annotation provided by the developer

**Q->What is the problem with type inferenece sometime and how do you solve with type casting?**

Suppose if we do multiplication and we are passing the string then it will throw an error so we can avoid

It by passing the type of data

For example

// Example without type casting

let value = '100'; // TypeScript infers the type of 'value' as string

let result = value \* 2; // Error: Operator '\*' cannot be applied to types 'string' and 'number'

// Example with type casting

let value2 = '100'; // TypeScript infers the type of 'value2' as string

let result2 = Number(value2) \* 2; // Using type casting to convert 'value2' to a number

console.log(result2); // Output: 200

solution

let value:number = 100;

**Q->What does tsc init do?**

It gives the file tsconfig.json

**Q->What does strict mode do?**

In TypeScript, the "strict" mode refers to a set of compiler options that enforce stricter type-checking rules and coding practices during the compilation process. When the strict mode is enabled, the TypeScript compiler applies additional checks and constraints to ensure that the code adheres to more rigorous type safety and best coding practices, leading to more robust and maintainable codebases.

**Q->What does buttonElement ! exactly do ? Why do we put EXCLAMATION MARK( ! ).**

If we do not use buttonElement! Then it will suggest that this might be empty or null but when we use

! then this varaiable is non-nullable

**Q->Why shouldnt we use any type?**

The any type removes all the benefits of TypeScript's strong type system, leading to a loss of type safety and type checking capabilities. This can make the code more error-prone and difficult to maintain, as the TypeScript compiler cannot provide meaningful feedback or detect potential type-related errors.

**Q-> what is union type**

When we are passing the type of data more than 1 one and ww have to use | pipe in between

**Q->Type aliases and interfaces** are both used to define custom types in TypeScript, but they serve different purposes and have distinct characteristics. Understanding their differences and use cases can help in choosing the appropriate approach for defining custom types based on specific requirements.

**Type Aliases:**

Type aliases in TypeScript allow you to create a new name for a type. They are especially useful when dealing with complex or lengthy type definitions that need to be reused. Type aliases can represent a union, intersection, tuple, or any other custom type definition.

type Point = {

x: number;

y: number;

};

**Interfaces:**

Interfaces in TypeScript are used to define the structure of an object, specifying the names and types of its properties, as well as defining function types. They are suitable for defining contracts within your code, ensuring that objects adhere to a specific shape or structure.

interface Point {

x: number;

y: number;

}

**When to Use:**

Type Aliases: Use type aliases when you need to create a name for a complex type or when you want to create reusable type definitions. They are particularly useful for creating descriptive names for types that might be used in multiple places within your codebase.

Interfaces: Use interfaces when you need to define the structure of an object, including its properties and method signatures. Interfaces are helpful for establishing contracts that objects must adhere to, promoting consistency and interoperability within your codebase.

**Q->Why do we need to install types/node? What does it do?**

**Basic of types**

@types allows the ts complier to understand the structure AND types of external library

We are using and when we use @types/nodes then it enables the typescript to provide the better support, improved automcompletion and improved type checking

@types/node🡪 when we install it then ts get to know that we are using nodes so it will start to provide the autocompletion and improved type checking related to node

**Q->Why do we need to install types/express? What does it do?**

@types/express🡪 when we install it then ts get to know that we are using express so it will start to provide the autocompletion and improved type checking related to express

**Q->What is the need for changing moduleresolution to es6?**

It tells the ts complier to use node.js style module resolution

**Q->Import keyword is very similar to require keyword. What do you think?**

Yes , working is same for both but there are some difference between

|  |  |
| --- | --- |
| **import** | **require** |
| Import syntax is part of es6 Module | Require is a part of commonJs Module |
| Import statmnet is hoisted at top and its processed before the code executing. It allows for named imports and default imports | The require function is a synchronous call that loads modules at runtime. It only supports the import of the entire module or its specific parts/variable |
| It supports in modern browser that supports Es6 Modules | It is used in nodejs environment and is not supported in web browser |

**Q->How to use import keyword?**

Import keyword is used to use the another file that is present in our project’

Import anynameforfile from ‘./file we want to use’ // if exports default exportfilename

Import exact{nameoffilewhichweexport} from ‘./file we want to use’ // named exports

**Q->How does import { Router } from 'express'. work. What does { } this do?**

By using {} we are extracting that file which is Router from express.

**Q->What do you mean by interface?Why do we have to create it for todos type?**

Interface is the structure that defines the behaviour that an object must implement

**Q->What are named exports and default exports?**

When we export the functions, variables then we can import those files with that name which we provide during export

// math.js

export const add = (a, b) => a + b;

export const subtract = (a, b) => a - b;

// main.js

import { add, subtract } from './math.js';

console.log(add(5, 3)); // Output: 8

console.log(subtract(5, 3)); // Output: 2

in case of exports default

When we export the functions, variables then we can import that file and we can give any name in case of export default

// math.js

const add = (a, b) => a + b;

export default add;

// main.js

import myAddFunction from './math.js';

console.log(myAddFunction(5, 3)); // Output: 8

**Q->What is type casting**

Type casting is a feature in TypeScript that allows developers to explicitly change the type of a value from one type to another

**Q->What is the need of RequestBody and RequestParams?**

We use RequesBody and RequrestParams to let know the ts what type of data will be in RequestBody and RequestParams and in what key will be that data is.

Like if we define

Type RequestBody = {text:string};

In Api we do like that

Const body = req.body as RequestBody

And we can access it like 🡪 body.text

But if we do like body.texts then it show an error

And same for params also

**Q->What is the problem we are trying to solve here?**

The problem is that a js script is also created when we complie our code using TypeSCripy comlier

**Q->How do we solve this problem?What does outdir: '/dist' do exactly?**

We solve this problem by outdir and all the (complied code)javascript code will be move in dist folder

**Q->Why do we have to change the rootDir to './src'?**

By using rootDir we are mention that in src folder contains our typescript file on which we are working

**Q->Why do we add node dist/app.js in package.json ? What does it exactly do?**

Node.js is a runtime environment for executing JavaScript code, but it does not natively support the execution of TypeScript code.

Node is not able to execute the typescript code

So we we have to add in script that our javascript whihch is complied file of ts have to run the app.js file whihcis present in dist folder

“script”:”node dist/app.js”

**Disadvatage of ts**

The introduction of static typing and other advanced features can sometimes lead to increased complexity, especially for small projects or projects that do not require strict typing. This additional complexity can result in longer development cycles and more effort spent on understanding and implementing TypeScript-specific features.

**Q🡪 if there are lot of advantage of ts then why should we use js**

Ease of Learning and Use: JavaScript has a relatively low entry barrier, making it more accessible for beginners and developers new to programming. Its simplicity and flexibility allow for quick prototyping and experimentation, making it an ideal choice for smaller projects or rapid development cycles.

**compilation error** is that error which occurs before runtime when it finds some type-error,syntax error and some other issue

Like in typescript if we define

Let num:number = ‘hello’ // This would result in a compile-time error in TypeScript

**Runtime-error** is that error which occurs when the program is running

Like

Let x = 10;

Let y = 0;

x/y gives run-time error because // This would result in a runtime error since dividing by zero is not allowed.

**// when we connect nodejs with typescript**

And we want to use

import express from 'express'

then steps:-

install-> npm i @types/express

When we passing the objects in port**(app.listen({port:3000})** so this must give error but to show/indicates those error

We have to perform those steps

first-1-> make some changes in tsconfig.json

   "moduleResolution": "node10",

Node is capable of run the javascript code not typescript code

error: remote origin already exists.

Steps:-

git add .

git commit -m "Your commit message here"

git remote -v( error: remote origin already exists.)important

git push