**Type Script**

**What is TypeScript ?**

It is is a open source programming language developed and maintained by Microsoft.

It is a strict syntactical superset of JavaScript and adds optional static typing to the language.

TypeScript is designed for the development of large applications and transcompiles to JavaScript.

Most modern frameworks are built using TypeScript. TypeScript overcomes the loosely typed issues of JavaScript.This helps in less code, less debugging and resources.

**Install TypeScript**

You can install TypeScript via npm,

npm install typescript

then run the compiler via tsc,

npx tsc

**Variable Declarations :**

[var/let/const] [identifier] : [type] = value;

TypeScript helps us classify the type of the declared variable.This helps us in restricting the amount of errors caused by type casting. This makes the TypeScript strictly typed

**Example:**

let name: string = ‘Ravi’;

let age: number = 20;

let name = ‘Ravi’; // this is same as the first statement but is implicitly referred as a string

## **Type Assertion in TypeScript**

var str = ‘1’;

var str2:number = <number><any>str;

**Variable scope:**

* **Global Scope** − Global variables are declared outside the programming constructs. These variables can be accessed from anywhere within your code.
* **Class Scope** − These variables are also called fields. Fields or class variables are declared within the class but outside the methods. These variables can be accessed using the object of the class. Fields can also be static. Static fields can be accessed using the class name.
* **Local Scope** − Local variables, as the name suggests, are declared within the constructs like methods, loops etc. Local variables are accessible only within the construct where they are declared

**Variable types:**

**Boolean**

The most basic datatype is the simple true/false value, which JavaScript and TypeScript call a boolean value.

Ex : let isDone: boolean = false;

**Number**

As in JavaScript, all numbers in TypeScript are either floating point values or BigIntegers.

let decimal: number = 6;

**String**

Another fundamental part of creating programs in JavaScript for webpages and servers alike is working with textual data. As in other languages, we use the type string to refer to these textual datatypes.

let color: string = "blue";

**Array**

TypeScript, like JavaScript, allows you to work with arrays of values. Array types can be written in one of two ways. In the first, you use the type of the elements followed by [] to denote an array of that element type:

let list: number[] = [1, 2, 3];

let list: Array<number> = [1, 2, 3];

**Tuple**

Tuple types allow you to express an array with a fixed number of elements whose types are known, but need not be the same.

let x: [string, number] = ["hello", 10];

**Enum**

A helpful addition to the standard set of datatypes from JavaScript is the enum

enum Color {

Red,

Green,

Blue,

}

let c: Color = Color.Green;

**Unknown**

We may need to describe the type of variables that we do not know when we are writing an application. These values may come from dynamic content – e.g. from the user – or we may want to intentionally accept all values in our API.

let notSure: unknown = 4;

notSure = "maybe a string instead";

**Any**

In some situations, not all type information is available or its declaration would take an inappropriate amount of effort.

let looselyTyped: any = {};

**Void**

void is a little like the opposite of any: the absence of having any type at all.

function warnUser(): void {

console.log("This is my warning message");

}

## **Null and Undefined**

In TypeScript, both undefined and null actually have their types named undefined and null respectively.

let u: undefined = undefined;

let n: null = null;

## **Never**

The never type represents the type of values that never occur.

function error(message: string): never {

throw new Error(message);

}

## **Object**

object is a type that represents the non-primitive type, i.e. anything that is not number, string, boolean, bigint, symbol, null, or undefined.

## **Functions**

To begin, just as in JavaScript, TypeScript functions can be created both as a named function or as an anonymous function. This allows you to choose the most appropriate approach for your application, whether you’re building a list of functions in an API or a one-off function to hand off to another function.

function add(x: number, y: number): number {

return x + y;

}

let myAdd = function (x: number, y: number): number {

return x + y;

};

**Classes :**

In javascript, it uses functions and prototype-based inheritance to build up reusable components which is a lot hectic to do and effort.

**Example in TypeScript:**

class Greeter {

greeting: string;

constructor(message: string) {

this.greeting = message;

}

greet() {

return "Hello, " + this.greeting;

}

}

let greeter = new Greeter("world");

**Access Modifiers:**

TypeScript provides public, private, protected modifiers

Also Type Script provides regular oops concepts like **inhertance, interface..**

By, L. N. S. S. Ravi Teja