****

**UNIVERSITY OF GONDAR**

**FACULTY OF INFORMATICS**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**SEMINAR TITLE: - TYPSCRIPT LANGUAGE**

**By**

|  |  |
| --- | --- |
| Full Name | ID NO |
| DESTAW ESUBALEW TAYE | GUS/00140/11 |
| ABEBE LEGESE GOSHU | GUS/00352/11 |
| BIRHAN FENTAW MERSHA | GUS/92386/11 |
| ABEBA WORKNEH | GUS/00033/10 |

UNDER THE GUIDANCE OF

Mr. TSHEHAY

AUGEST 30/2022

# **CERTIFICATE**

We certify that this MSC seminar report entitled <<Typescript Language>>

by:

<<ABEBE LEGESE

DESTAW ESUBALEW

BIRHAN FENTAW

ABEBA WORKNEH>>

is approved by us for submission. We certify further that, to the best of Our knowledge, the report represents work carried out by the students.

20/12/2014(2022) Mr. Tsehay

Date Name and Signature of Supervisor

# Acknowledgement

We sincerely express our gratitude to Department of Information Technology providing necessary help and assistance. We are very much grateful to Candidate. Mr. **Tsehay**, Assistant professor of University of Gondar for giving opportunity to work in the field of Typescript Language.

Table of Contents

[**CERTIFICATE** I](#_Toc112513321)

[Acknowledgement II](#_Toc112513322)

[**Acronym** VI](#_Toc112513323)

[**Abstract** VII](#_Toc112513324)

[**CHAPTER ONE** 1](#_Toc112513325)

[**1. Typescript language** 1](#_Toc112513326)

[1.1 Definition of typescript language 1](#_Toc112513327)

[1.2 Objective of the Typescript language 2](#_Toc112513328)

[1.3 Why Use Typescript? 2](#_Toc112513329)

[1.4 components of typescript language 2](#_Toc112513330)

[1.4.1 Language 2](#_Toc112513331)

[1.4.2 The Typescript Compiler 3](#_Toc112513332)

[1.4.3 The Typescript Language Services 3](#_Toc112513333)

[**Chapter 2** 4](#_Toc112513334)

[**2.Types of typescript languages** 4](#_Toc112513335)

[2.1 static types 4](#_Toc112513336)

[2.2 Generic 8](#_Toc112513337)

[2.3 Decorators 8](#_Toc112513338)

[2.2 Features of typescript language 11](#_Toc112513339)

[**Chapter 3** 13](#_Toc112513340)

[**3. Application of Typescript Language** 13](#_Toc112513341)

[3.1. Reasons to use typescript languages 13](#_Toc112513342)

[3.1.1 Better developer experience 13](#_Toc112513343)

[3.1.2 Expanding your programming knowledge 13](#_Toc112513344)

[3.1.3 Easy to start and adopt 13](#_Toc112513345)

[3.1.4 Active community 14](#_Toc112513346)

[3.1.5 Runs everywhere 14](#_Toc112513347)

[3.1.6 Prevents bugs 14](#_Toc112513348)

[3.1.7 Documentation + communication 14](#_Toc112513349)

[3.1.8 Code quality 14](#_Toc112513350)

[3.1.9 Easier refactoring 14](#_Toc112513351)

[3.1.10 AOT compilation 15](#_Toc112513352)

[3.2 CHARACTERISTICS OF TYPESCRIPT LANGUAGE 15](#_Toc112513353)

[3.3 Advantages of Typescript 17](#_Toc112513354)

[3.4 Disadvantages of Typescript 17](#_Toc112513355)

[**CHAPTER FOUR** 18](#_Toc112513356)

[**Conclusion** 18](#_Toc112513357)

[**References** 19](#_Toc112513358)

**Table of figure**

[Figure 1 type script compiler 3](#_Toc112510395)

[Figure 2 Typescript language types 4](#_Toc112510396)

[Figure 3 Built in data types 5](file:///C:\Users\SET\Desktop\seminar%20Ass\Seminar%20Ass%20Final.docx#_Toc112510397)

[Figure 4 user defined types 7](#_Toc112510398)

[Figure 5 types of decorates 9](file:///C:\Users\SET\Desktop\seminar%20Ass\Seminar%20Ass%20Final.docx#_Toc112510399)

# **Acronym**

TS= Typescript

JS = JavaScript

IDE = integrated development environment

TSC = Typescript compiler

UI = User Interface

ECMA = European Computer Manufacturer's Association

AMD = Advanced Micro Devices

ES = Extra Segment

VM = Virtual Machine

# **Abstract**

**Typescript** is a [free and open source](https://en.wikipedia.org/wiki/Free_and_open_source) [programming language](https://en.wikipedia.org/wiki/Programming_language) developed and maintained by [Microsoft](https://en.wikipedia.org/wiki/Microsoft). It is a strict syntactical [superset](https://en.wikipedia.org/wiki/Superset) of [JavaScript](https://en.wikipedia.org/wiki/JavaScript) and adds optional [static typing](https://en.wikipedia.org/wiki/Static_typing) to the language. It is designed for the development of large applications and [trans piles](https://en.wikipedia.org/wiki/Source-to-source_compiler) to JavaScript.As it is a superset of JavaScript, existing JavaScript programs are also valid Typescript programs. [[1]](#one)

TypeScript is an extension of JavaScript intended to enable easier development of large-scale JavaScript applications. While every JavaScript program is a TypeScript program, TypeScript offers a module system, classes, interfaces, and a rich gradual type system. The intention is that TypeScript provides a smooth transition for JavaScript programmers—well-established JavaScript programming idioms are supported without any major rewriting or annotations [[2].](#two) This means that the compiled TypeScript code contains no type information at all. There is therefore no run-time type checking. A programmer has to resort to the built in JavaScript type checking abilities, which is considered by some to be good enough  [[3]](#three)

# **CHAPTER ONE**

# **1. Typescript language**

## 1.1 Definition of typescript language

**Background**

Typescript is a typed superset of JavaScript that compiles directly to JavaScript code. Typescript files commonly use the .ts extension. Many IDEs support Typescript without any other setup required, but Typescript can also be compiled with the Typescript Node.JS package from the command line. TypeScript is a language created by Microsoft  [[4].](#four) It is superset of JavaScript that adds things such as a module system, classes, interfaces and a static type system. The aim of the language is to help developers by enabling the ability to catch mistakes statically. It also allows for other development features that can be provided by IDEs such as the ability to list what properties and methods that exist on an object

The Typescript language is defined in a careful, clear, but informal document  [[5].](#five) Naturally, this document contains certain ambiguities. For example, the language permits subtyping recursive types; the literature contains several rules for subtyping recursive types, not all sound, and the document does not say exactly which is employed. Therefore, it may be difficult to know exactly what is the type system, and in what ways it is sound or unsound.

Typescript is a free and open source programming language developed and maintained by Microsoft. It is a strict superset of JavaScript. JavaScript remains a poor language for developing and maintaining large applications. Typescript is an extension of JavaScript intended to address this deficiency. Typescript is an extension of JavaScript intended to enable easier development of large-scale JavaScript applications. Typescript adds optional static typing and class-based object-oriented programming to the java script language. The Typescript programming language adds optional types to JavaScript, with support for interaction with existing JavaScript libraries via interface declarations. While every JavaScript program is a Typescript program, Typescript offers a module system, classes, interfaces, and a rich gradual type system. The intention is that Typescript provides a smooth transition for JavaScript programmers— well-established JavaScript programming idioms are supported without any major rewriting or annotations. One interesting consequence is that the Typescript type system is not statically sound by design[[6].](#six)

Typescript enriches JavaScript with a module system, classes, interfaces, and a static type system. As Typescript aims to provide lightweight assistance to programmers, the module system and the type system are flexible and easy to use. In particular, they support many common JavaScript programming practices. They also enable tooling and IDE experiences previously associated with languages such as C# and Java.

Typescript is a syntactic sugar for JavaScript. Typescript is a compiled language; it’s not interpreted at run-time. The Typescript compiler takes Typescript files (.ts) and compiles them in to JavaScript files (.js). Typescript code is not processed by browsers that work with JavaScript code. Therefore, to be executed, Typescript code has to be translated into JavaScript. This operation is referred to as Transpiration and the tools that perform it are called Transpolar.

## 1.2 Objective of the Typescript language

The primary goal of Typescript language is to give a **statically** typed experience to JavaScript development. A syntactic superset of JavaScript, it adds syntax for declaring and expressing types, for annotating properties, variables, parameters and return values with types, and for asserting the type of an expression.

The main aim of this paper is to formalize these type-system extensions. Typescript also adds a number of new language constructs, such as classes, modules, and lambda expressions. The Typescript compiler implements these constructs by translation to JavaScript (ECMAScript 5).

## 1.3 Why Use Typescript?

Typescript builds on JavaScript to effectively fill in the gaps and give developers better tooling at any scale. Weighed against its competitors, Typescript is both easier and more efficient.

## 1.4 components of typescript language

### 1.4.1 Language

It features the Typescript language elements. It comprises elements like syntax, keywords, and type annotations.

### 1.4.2 The Typescript Compiler

The Typescript compiler (TSC) transform the Typescript program equivalent to its JavaScript code. It also performs the parsing, and type checking of our Typescript code to JavaScript code.

Components of TypeScript

Figure 1 type script compiler

Browser doesn't support the execution of Typescript code directly. So the program written in Typescript must be re-written in JavaScript equivalent code which supports the execution of code in the browser directly. To perform this, Typescript comes with Typescript compiler named "tsc." The current version of Typescript compiler supports ES6, by default. It compiles the source code in any module like ES6, SystemJS, AMD, etc. We can install the Typescript compiler by locally, globally, or both with any **npm** package. Once installation completes, we can compile the Typescript file by running "tsc" command on the command line.

### 1.4.3 The Typescript Language Services

The language service provides information which helps editors and other tools to give better assistance features such as automated refactoring and IntelliSense [[7].](#seven) It exposes an additional layer around the core-compiler pipeline. It supports some standard typical editor operations like code formatting and outlining, colorization, statement completion, signature help, etc.

# **Chapter 2**

# **2.Types of typescript languages**

The Typescript language supports different types of values. It provides data types for the JavaScript to transform it into a strongly typed programing language. JavaScript doesn't support data types, but with the help of Typescript, we can use the data types feature in JavaScript. Typescript plays an important role when the object-oriented programmer wants to use the type feature in any scripting language or object-oriented programming language. The Type System checks the validity of the given values before the program uses them. It ensures that the code behaves as expected [[8].](#eight)

Typescript provides data types as an optional Type System. We can classify the Typescript data type as following.

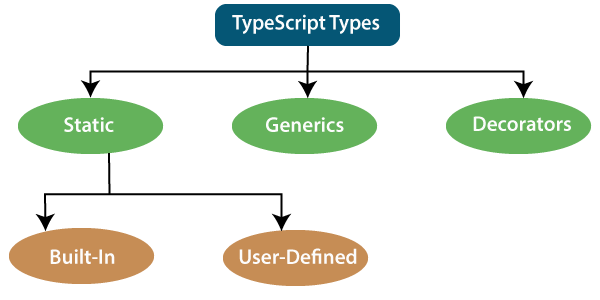


Figure 2 Typescript language types

### 2.1 static types

In the context of type systems, static types mean "at compile time" or "without running a program." In a statically typed language, variables, parameters, and objects have types that the compiler knows at compile time. The compiler used this information to perform the type checking.

Static types can be further divided into two sub-categories:

#### **2.1.1 Built-in or Primitive Type**

The Typescript has five built-in data types, which are given below

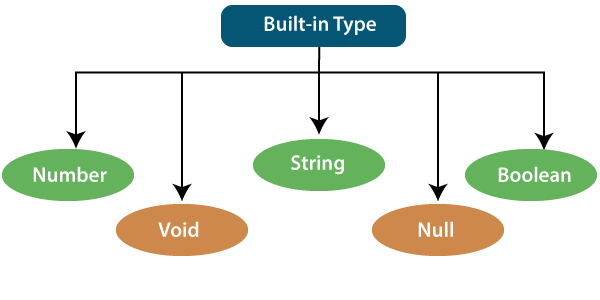


Figure Built in data types

**Number**: Like JavaScript, all the numbers in Typescript are stored as floating-point values. These numeric values are treated like a number data type.

**Syntax:**

let identifier: number = value;

**Examples:-**

let first: number = 12.0;             // number

let second: number = 0x37CF;          // hexadecimal

let third: number = 0o377 ;           // octal

let fourth: number = 0b111001;        // binary

console.log(first);           // 123

console.log(second);          // 14287

console.log(third);           // 255

console.log(fourth);          // 57

**String:** We will use the string data type to represents the text in Typescript. String type work with textual data. We include string literals in our scripts by enclosing them in single or double quotation marks.

**Syntax**

let identifier: string = " ";                Or

let identifier: string = ' ';

**Boolean:** The string and numeric data types can have an unlimited number of different values, whereas the Boolean data type can have only two values.

**Syntax**

let identifier: BooleanBoolean = Boolean value;

**Examples**

let isDone: boolean = false;

**Void:** A void is a return type of the functions which do not return any type of value. It is used where no data type is available. A variable of type void is not useful because we can only assign undefined or null to them. An undefined data type denotes uninitialized variable, whereas null represents a variable whose value is undefined.

Syntax:

Unsable: void=undefined;

Example:

function helloUser(): void {

alert("This is a welcome message");

 }

let tempNum: void = undefined;

  tempNum = null;

 tempNum = 123;      //Error

**Null:** Null represents a variable whose value is undefined. Much like the void, it is not extremely useful on its own. The Null accepts the only one value, which is null. The Null keyword is used to define the Null type in Typescript, but it is not useful because we can only assign a null value to it.

let num: number = null;

let bool: boolean = null;

let str: string = null;

**Difference between Null and Undefined**

NULL: Null is used to represent an intentional absence of value. It represents a variable whose value is undefined. It accepts only one value, which is null. The Null keyword is used to define the Null type in Typescript, but it is not useful because we can only assign a null value to it.

Undefined :It represents uninitialized variables in Typescript and JavaScript. It has only one value, which is undefined. The undefined keyword defines the undefined type in Typescript, but it is not useful because we can only assign an undefined value to it.

**2.1.2 User-Defined Datatype**

Typescript supports the following user-defined data types:

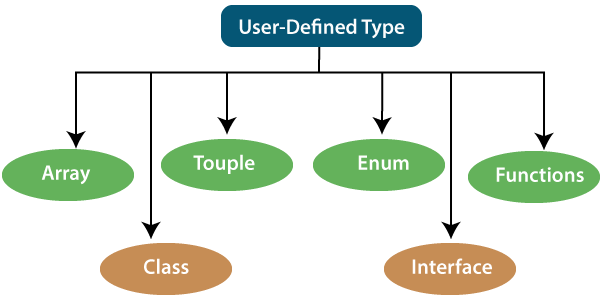


Figure 4 user defined types

**Array:** An array is a collection of elements of the same data type.

1. Use the type of the elements followed by [] to denote an array of that element type:

var list : number[] = [1, 3, 5];

The second way uses a generic array type

var list : Array**<number>** = [1, 3, 5];

**Class:** Classes are used to create reusable components and acts as a template for creating **objects.** It is a logical entity which store variables and functions to perform operations.

class Student  {

RollNo: number;

Name: string;

    constructor(\_RollNo: number, Name: string)

    {

        this.RollNo = \_rollNo;

        this.Name = \_name;

    }

    showDetails()

    {

        console.log(this.rollNo + " : " + this.name);

    }

}

**Touple:** The Tuple is a data type which includes two sets of values of different data types. It allows us to express an array where the type of a fixed number of elements is known, but they are not the same

Declare a tuple

let a: [string, number];

// Initialize it

a = ["hi", 8, "how", 5]; // OK

**Interface:** An Interface is a structure which acts as a contract in our application. It defines the syntax for classes to follow, means a class which implements an interface is bound to implement all its members. It cannot be instantiated but can be referenced by the class which implements it.

**Enums:** Enums define a set of named constant. TypeScript provides both string-based and numeric-based enums. By default, enums begin numbering their elements starting from 0, but we can also change this by manually setting the value to one of its elements. TypeScript gets support for enums from ES6.

**Functions:** A function is the logical blocks of code to organize the program. Like JavaScript, Typescript can also be used to create functions either as a **named function** or as an **anonymous function**. Functions ensure that our program is readable, maintainable, and reusable.

### 2.2 Generic

Generic is used to create a component which can work with a variety of data type rather than a single one. It allows a way to create reusable components. It ensures that the program is flexible as well as scalable in the long term. Typescript uses generics with the type variable <T> that denotes types. The type of generic functions is just like non-generic functions, with the type parameters listed first, similarly to function declarations [[9].](#nine)

### 2.3 Decorators

A Decorator is a special kind of declaration that can be applied to classes, methods, accessor, property, or parameter. Decorators are simply functions that are prefixed **@expression** symbol, where expression must evaluate to a function that will be called at runtime with information about the decorated declaration. Note: Decorators are an experimental feature proposed for ES7. It is already in use by some of the JavaScript frameworks including Angular 2. The Decorators may change in future releases [[10].](#ten)

**Example**

function f() {

    console.log("f(): evaluated");

    return function (target, propertyKey: string, descriptor: PropertyDescriptor) {

        console.log("f(): called");  }  }

Typescript uses the following types of Decorators:

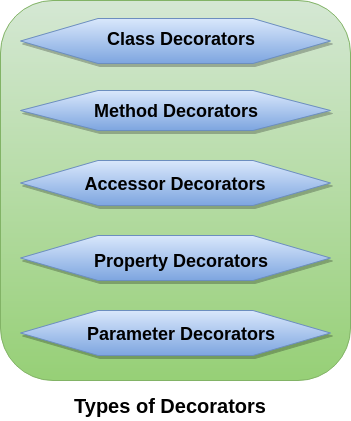


Figure types of decorates

1. [Class Decorators](https://www.javatpoint.com/typescript-decorators#class-decorators)
2. [Method Decorators](https://www.javatpoint.com/typescript-decorators#method-decorators)
3. [Accessor Decorators](https://www.javatpoint.com/typescript-decorators#accessor-decorators)
4. [Property Decorators](https://www.javatpoint.com/typescript-decorators#property-decorators)
5. [Parameter Decorators](https://www.javatpoint.com/typescript-decorators#parameter-decorators)
6. Class Decorator

A class decorator is defined just before the class declaration, and it tells about the class behaviors. A class decorator is applied to the constructor of the class. A class decorator can be used to observe, modify, or replace a class definition.

class Company {

 @ReadOnly

 name: string = "JavaTpoint.com";

}

let comp = new Company ();

comp.name = 'SSSIT.com'; // Here, we can't change company name.

console.log(comp.name); // 'JavaTpoint.com'

1. A Method Decorator is defined just before a method declaration. It is applied to a property descriptor for the method. It can be used to observe, modify, or replace a method definition. We cannot use method decorator in a declaration file.

In the below example, the **@log** decorator will log the new item entry.

class Item {

    itemArr: Array;

    constructor() {

        this.itemArr = [];

        }

    @log

    Add(item: string): void {

       this.itemArr.push(item);

       }

    GetAll(): Array {

       return this.itemArr;

       }

}

1. An Accessor Decorator is defined just before an accessor declaration. It is applied to the property descriptor for the accessor. It can be used to observe, modify, or replace an accessor's definitions

class Person {

    msg: string;

    constructor(message: string) {

        this.msg = message;      }

    @validate

    show(@required name: string) {

        return "Hello " + name + ", " + this.msg;

    }  }

1. A property decorator is defined just before a property declaration. It is similar to the method decorators. The only difference between property decorators and method decorators is that they do not accept property descriptor as an argument and do not return anything.

The expression for the property decorator function accepts two arguments. They are:

1. Either the constructor function of the class for a static member or the prototype of the class for an instance member.
2. The member name.

**Example**:

In the below example, the **@ReadOnly** decorator will make the name property as read-only, so we can't change its value.

class Company {

 @ReadOnly

 name: string = "JavaTpoint.com";  }

let comp = new Company();

comp.name = 'SSSIT.com'; // Here, we can't change company name.

console.log(comp.name); // 'JavaTpoint.com'

1. A parameter decorator is defined just before a parameter declaration. It is applied to the function for a class constructor or method declaration. It cannot be used in a declaration file or in any other ambient context (such as in a declared class).

The expression for the parameter decorator function accepts three arguments. They are:

1. Either the constructor function of the class for a static member or the prototype of the class for an instance member.
2. The member name.
3. The index of the parameter in the function?s arguments list.

**Example**:

In the below example, a parameter decorator **(@required)** is applied to the parameter of a member of the **Person** class.

class Person {

    msg: string;

    constructor(message: string) {

        this.msg = message;    }

    @validate

    show(@required name: string) {

        return "Hello " + name + ", " + this.msg;   }  }

## 2.2 Features of typescript language

**Typescript is just JavaScript**. Typescript starts with JavaScript and ends with JavaScript. Typescript adopts the basic building blocks of your program from JavaScript. Hence, you only need to know JavaScript to use Typescript. All Typescript code is converted into its JavaScript equivalent for the purpose of execution.

**Typescript supports other JS libraries**. Compiled Typescript can be consumed from any JavaScript code. Typescript-generated JavaScript can reuse all of the existing JavaScript frameworks, tools, and libraries.

**JavaScript is Typescript**. This means that any valid **.js** file can be renamed to **.ts** and compiled with other Typescript files.

**Typescript is portable**. Typescript is portable across browsers, devices, and operating systems. It can run on any environment that JavaScript runs on. Unlike its counterparts, Typescript doesn’t need a dedicated VM or a specific runtime environment to execute.

# **Chapter 3**

# **3. Application of Typescript Language**

## 3.1. Reasons to use typescript languages

Typescript introduces strong typing. Most of the additional syntax structures like type annotations, interfaces, generics are to ensure program semantic consistency. They are used only when developing and removed during the compilation time.

So what are the reasons why you should use Typescript?

### 3.1.1 Better developer experience

Thanks to functionalities that Typescript comes with, it’s easier and more intuitive for a software developer to write a program. One of the main conveniences is that you know not only the name of the variable, but also **what kind of data it stores**. You can focus more on big stuff like logic in the application, and care less about small details.

Typescript also saves your time because instead of having to search into documentation, when typing **TS can suggest all of the available options like props in components or functions/classes in libraries.** It provides you with a higher chance of using them as intended and passing correct parameters. For example, you don’t have to look into the documentation to see that you can add checked, checkedIcon props and see what type they should be to <Checkbox/> component from Material UI. Without typing, it’s easier to forget to wrap variable in an array or pass parameters in the wrong order [[11].](#eleven)

### 3.1.2 Expanding your programming knowledge

By learning Typescript, you get to know **new programming concepts**, that you may haven’t had contact with, like types, generics, interfaces, enums, namespaces. All of these structures were inspired by static-typed languages like C#, Java, and C++, so if you use Typescript, it may benefit in the future when you will want to try new programming language.

### 3.1.3 Easy to start and adopt

If you want to use TypeScript, it’s very easy to get started with. Or maybe you already are developing an application for some time that’s already written in plain JavaScript? You can introduce TS to your existing project incrementally because it compiles also .js files. No need to rewrite your whole code. [Here](https://basarat.gitbooks.io/typescript/content/docs/types/migrating.html) you can find an easy unofficial guide on how to migrate

### 3.1.4 Active community

ypeScript is getting more and more popular. It’s used by the top tech companies like **Google, Microsoft, Airbnb, Shopify, Asana, Adobe, and Mozilla** so we can assume that it reaches their expectations in terms of scalability - as they are developing large and complex applications.

TypeScript has also a lot of amazing open-source contributors, like DefinitelyTyped and TypeStrong, which are constantly developing TypeScript’s tooling

### 3.1.5 Runs everywhere

As we said, TypeScript compiles to pure JavaScript - it means that it can run everywhere. It can be used either on the front-end of the application (for example with [React, Angular](https://www.merixstudio.com/blog/angular-vs-react-choosing-right-technology-your-next-project/) or even without any framework) and with the [Node.js](https://www.merixstudio.com/blog/what-you-need-know-about-nodejs-short-technology-guide/) on the back-end.

Another advantage of TypeScript is that it compiles to any JS versions, ES6, ES5 and down to… ES3.

### 3.1.6 Prevents bugs

Typescript won't make your software bug free. **But it can prevent a lot of type-related errors**. For example, if you set strictNullChecks option to true when you have a variable that can be an object or null/undefined, you can't get property of this object without checking if it isn't null/undefined.

### 3.1.7 Documentation + communication

Types act like another layer of documentation. They store valuable information in the code, which is extremely helpful for future software developers that will use your code. It’s a different way to communicate with your co-workers about the meaning and enforce them on how to use things that we build. Interfaces tell other people your intentions without them, for example, having to open a separate test file. From my experience, coming into the project with types was much more painless and faster in comparison to just JavaScript one. So, another time-saving.

Unlike any other documentation, TypeScript is always in sync with the newest changes.

### 3.1.8 Code quality

Defining data structures in the beginning, using types and interfaces, forces you to think about your app’s data structure from the start and **make better design decisions**.

### 3.1.9 Easier refactoring

When you want to rename a variable with IDE, it can only make guesses - so sometimes it ends up with unwanted changes. However, when IDE will be provided with the instrumentation of Typescript types, then it can be 100% accurate. In pure JS it’s necessary to do a regression test of the whole system after such a refactor, which may take lots of time.

### 3.1.10 AOT compilation

Typescript does everything during compilation time, so it doesn’t have a negative effect on runtime performance. One of the examples of this can be protected and private fields introduced into the standard by [ECMA Script](https://www.merixstudio.com/blog/ecmascript-exciting-changes-for-javascript-developers/), which has been around in Typescript for some time. The difference is that TS is checking it at compilation, and ES during runtime and this can affect application performance.

## 3.2 CHARACTERISTICS OF TYPESCRIPT LANGUAGE

Typescript originated from the perceived shortcomings of JavaScript for the development of large-scale applications[[12].](#tewelve)

1. **Type annotations** Type signature or type annotation defines the inputs and outputs for a function, subroutine or method. A type signature includes the function's return type, the number of arguments, the types of arguments, or errors it may pass back.

2. **compile-time** type checking During compilation the typescript language compiler checks the data type.

3. **Type inference** Type inference refers to the automatic deduction of the data type of an expression in a programming language.

4. **Type erasure** In programming languages, type erasure refers to the compile-time process by which explicit type annotations are removed from a program, before it is executed at run-time. Operational semantics that do not require programs to be accompanied by types are called type-erasure semantics, to be contrasted with type-passing semantics. The possibility of giving type-erasure semantics is a kind of abstraction principle, ensuring that the run-time execution of a program does not depend on type information. In the context of generic programming, the opposite of type erasure is called reification.

5. **Interfaces** In object-oriented programming, a protocol or interface is a common means for unrelated objects to communicate with each other. These are definitions of methods and values which the objects agree upon in order to co-operate.

6. **Enumerated** type In computer programming, an enumerated type (also called enumeration or enum, or factor in the R programming language, and a categorical variable in statistics) is a data type consisting of a set of named values called elements, members or enumerators of the type. The enumerator names are usually identifiers that behave as constants in the language. A variable that has been declared as having an enumerated type can be assigned any of the enumerators as a value. In other words, an enumerated type has values that are different from each other, and that can be compared and assigned, but which are not specified by the programmer as having any particular concrete representation in the computer's memory; compilers and interpreters can represent them arbitrarily.

7. **Mixin** In object-oriented programming languages, a mixin is a class that contains a combination of methods from other classes. How such a combination is done depends on the language. If a combination contains all methods of combined classes, it is equivalent to multiple inheritance. Mixins are sometimes described as being "included" rather than "inherited". Mixins encourage code reuse and can be used to avoid the inheritance ambiguity that multiple inheritance can cause (the "diamond problem"), or to work around lack of support for multiple inheritance in a language. A mixin can also be viewed as an interface with implemented methods[[13].](#thirteen)

8. G**eneric**: Generic programming is a style of computer programming in which algorithms are written in terms of types to-be specified-later that are then instantiated when needed for specific types provided as parameters.

9. **Namespaces** Namespace is a set of symbols that are used to organize objects of various kinds, so that these objects may be referred to by name. Prominent examples include: File systems are namespaces that assign names to files. Programming languages organize their variables and subroutines in namespaces. Computer networks and distributed systems assign names to resources, such as computers, printers, websites, (remote) files, etc.

10. **Tuple** A tuple is a finite ordered list of elements. In mathematics, an n-tuple is a sequence (or ordered list) of n elements, where n is a non-negative integer.

## ****3.3**** Advantages of Typescript

* Typescript features mistakes at arrangement time while JavaScript, at the runtime.
* Typescript gives the advantages of discretionary static composing: TS types can be added to factors, capacities, properties, and so forth.
* Typescript underpins specifically or static composing. Static composing can be valuable to help archive capacities, explain utilization, and diminish psychological overhead (interface type clues and getting expected mistakes continuously programming).
* Typescript runs in any program or JavaScript motor.
* Extraordinary tooling with IntelliSense which gives dynamic clues as the code’s additional.
* Typescript helps in code organizing.
* Typescript has a namespace idea by characterizing a module.
* Typescript’s explanations can be discretionary.
* Typescript upholds interfaces.
* Typescript’s manager modules give a standout amongst other IDE engineer insight.
* Typescript has better documentation for APIs which is in a state of harmony with a source code. A few organizations report a decrease in bugs when they change to Typescript. [[14]](#ten)

## ****3.****4 Disadvantages of Typescript

* Typescript sets aside a long effort to incorporate the code.
* Typescript doesn’t uphold theoretical classes.
* When utilizing an outsider library, there should be a definition document, and now and again it’s not generally accessible.
* Nature of type definition documents is a worry.
* At whatever point Typescript needs to run in a program, there should even now be a gathering step to change Typescript to JavaScript.
* Enrolling Typescript engineers may be a genuine annoyance, since a portion of the designers are not keen on learning Typescript, being entirely alright with utilizing JavaScript. Groups that as of now use JavaScript, become profitable with Typescript after around 2-3 months, and conversant in Typescript simply after about a large portion of a year. So instructing staff can likewise be an obstacle.
* Typescript isn’t completely **coexpressive** with JavaScript. Missing highlights include: HOFs, Composition, Generics with Higher key.

# **CHAPTER FOUR**

# **Conclusion**

This paper describes and analyses the core of the TypeScript language, and in particular its type system. The work that it represents has been useful in resolving ambiguities in the language definition, and in identifying minor unintended inconsistencies and mistakes in the language implementation. It provides a basis for partial soundness theorems, and it isolates and accounts for sources of unsoundness in the type system. Beyond the details of this work (which are specific to TypeScript, and which may perhaps change, as TypeScript develops further), we hope that our results will contribute to the principled study of deliberate unsoundness. In this direction, we believe that there are various opportunities for intriguing further research.

In particular, to the extent that any type system expresses programmer intent, we would expect that it could be useful in debugging, despite its unsoundness. Research on blame might be helpful in this respect. It may also be worthwhile to codify programmer guidance that would, over time, reduce the reliance on dangerous typing rules. Static analysis tools may support this guidance and complement a type system. These and related projects would aim to look beyond sound language fragments: the principles of programming languages may also help us understand and live with unsoundness.

# **References**

|  |  |
| --- | --- |
| [1] | M.Metey and V.P Veiko, Typescript Language Specification 0.9.5 edition, 2014, Berlin, Germany: Springer Verlag, 1998. |
| [2] | G.Bierman M. Parkinson and A.pitts MJ, An imperative core calculus for Java and with effects. Technical Report563, University of Cambrige: computer laboratory, 2003. |
| [3] | R-Chugh D-Herman and R Jhala, DCRockford , Dependent types for Javascript, in proceedings of OOSLA., Berlin: University of Cambrige, 2008. |
| [4] | C. Anderson and L. Cardelli , A theory of objects, Germany: Springer Verlag, 1996. |
| [5] | Saptarshi Bhattacharyya, Asoke Nath, Application of TypeScript Language : A Brief overview, Kolkata, Kolkata: St. Xavier’s College, 2016. |
| [6] | P. Minder, Introducing modified TypeScript in an exisƟng framework to im prove error handling, Linköping: Linköping University, 2016. |
| [7] | S. Fenton, Pro TypeScript Application, Germany: Trí Dũng Lê, 2009. |
| [8] | A. Igarashi, B. Pierce, and P. Wadler, . Featherweight Java: A minimal core calculus for Java and GJ. ACM, TOPLAS,: TOPLAS,, 2001. |
| [9] | J. Turner, Announcing TypeScript, Neyork: TypeScript Language team blo, 204. |
| [10] | P. Thiemann., Towards a type systems for analyzing JavaScript programs, lendon: Proceedings of ESOP, 2005. |
| [11] | Barbara Kitchenham and Stuart Charters, typescript language engineeering, Berlin: university of berlin, 2009. |
| [12] | Barbara Kitchenham and Stuart Charters., : Engineering 2: : Engineering 2, 2007. |
| [13] | Gavin Bierman, Martín Abadi, and Mads Torgersen, berlin: berlin university. |
| [14] | .. M. Corporation., TypeScript Language Specification, 0.9.5 edition, Washington: Microsoft, 2014. |
| [15] | B. Peter, microsoft type script:the Javascript we need, or a solution looking for a problem?, Ars Technica conde Nast, 2015. |