TypeScript

- Introduction
- Basics Features & Types
- Classes, Interfaces, Modules & Namespaces
- Development Environment
- Decorators

I

Introduction and installation

?

Types in typescript.

What is TypeScript ?

Why TypeScript is developed while having JavaScript ?

What are the **features** of typescript ?

What is the **difference** between TypeScript and JavaScript ?

What are the **advantages** of using TypeScript over JavaScript ?

What are the **disadvantages** of using TypeScript over JavaScript ?

Whether you are doing front end development Or backend with node

Using Typescript will boost your productivity, code quality, readability.

Prerequisites

Basic javascript concepts

OOP concepts

What's wrong with javascript

Developed many years ago for very basic dom manipulation

Lack Of Types - Dynamic typing

Lack of modularity

Objects are loosely coupled - anything can be added

There is really no class in javascript *[but technically there is]*

Application complexity

But
Why do we
need
javascript

Because of Browsers...

What is TypeScript

Fig: TypeScript is a SuperSet of JavaScript

TypeScript is an open-source **programming language.**

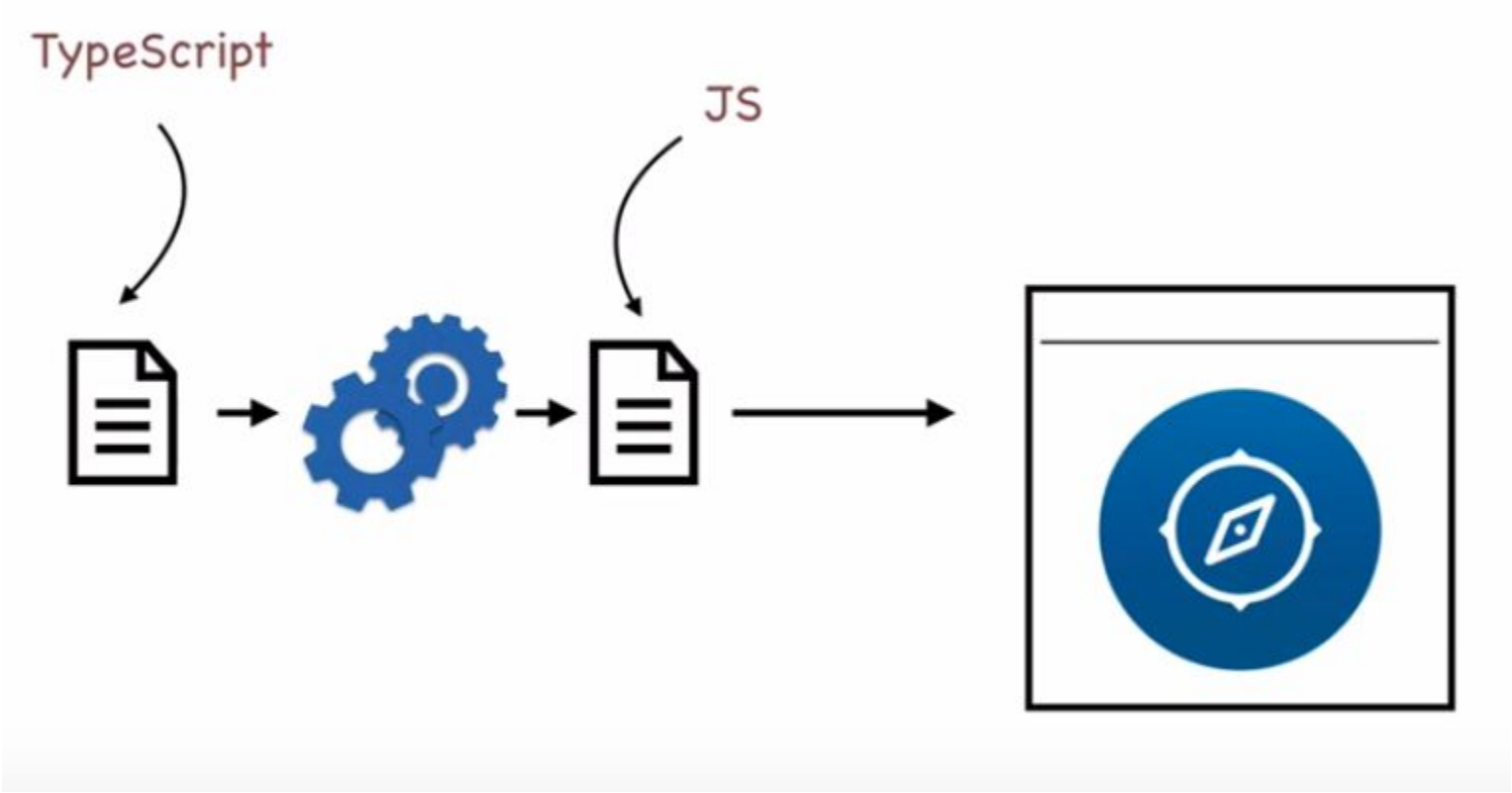
It is developed and maintained by **Microsoft.**

TypeScript follows javascript syntactically but adds more features to it.

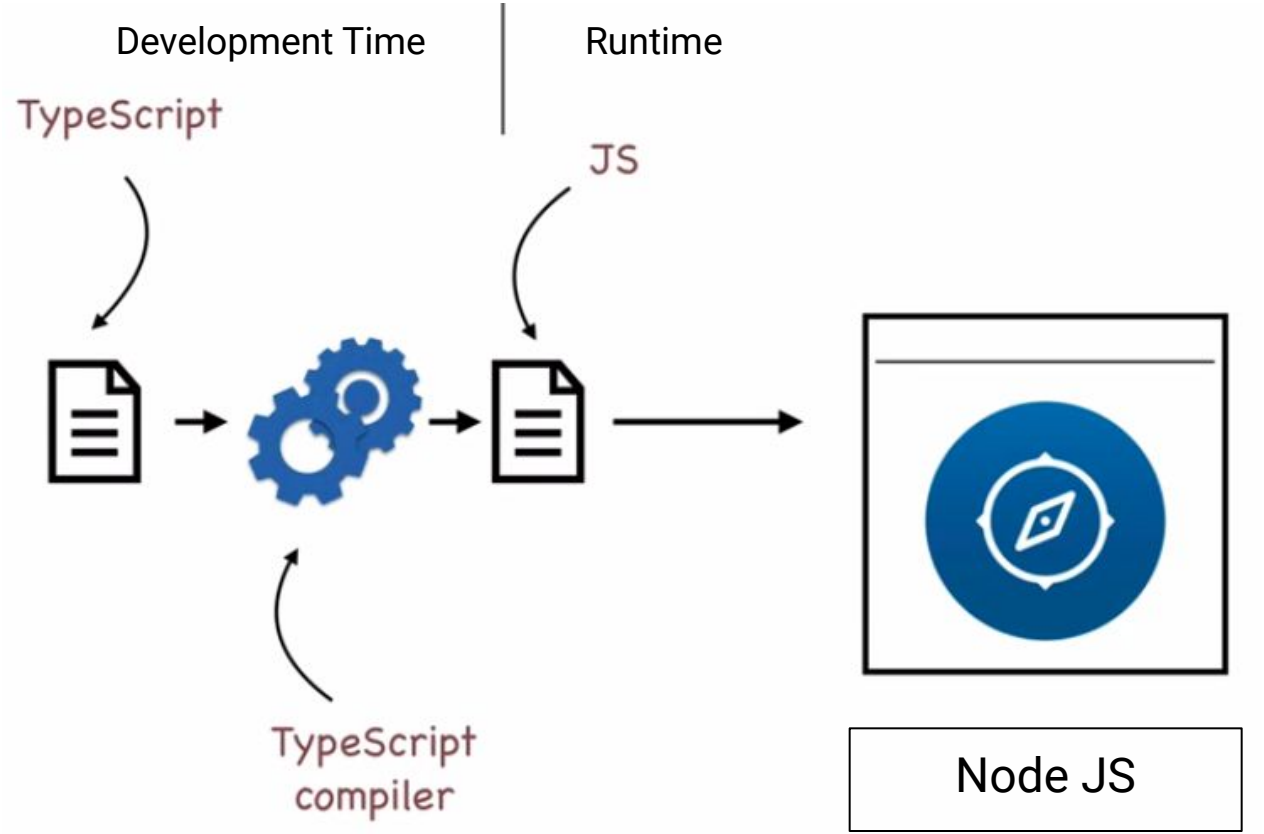
Typescript is a typed **superset of JavaScript** that compiles to plain JavaScript

Typescript is purely **object-oriented** with features like
classes, objects and interfaces just like Java

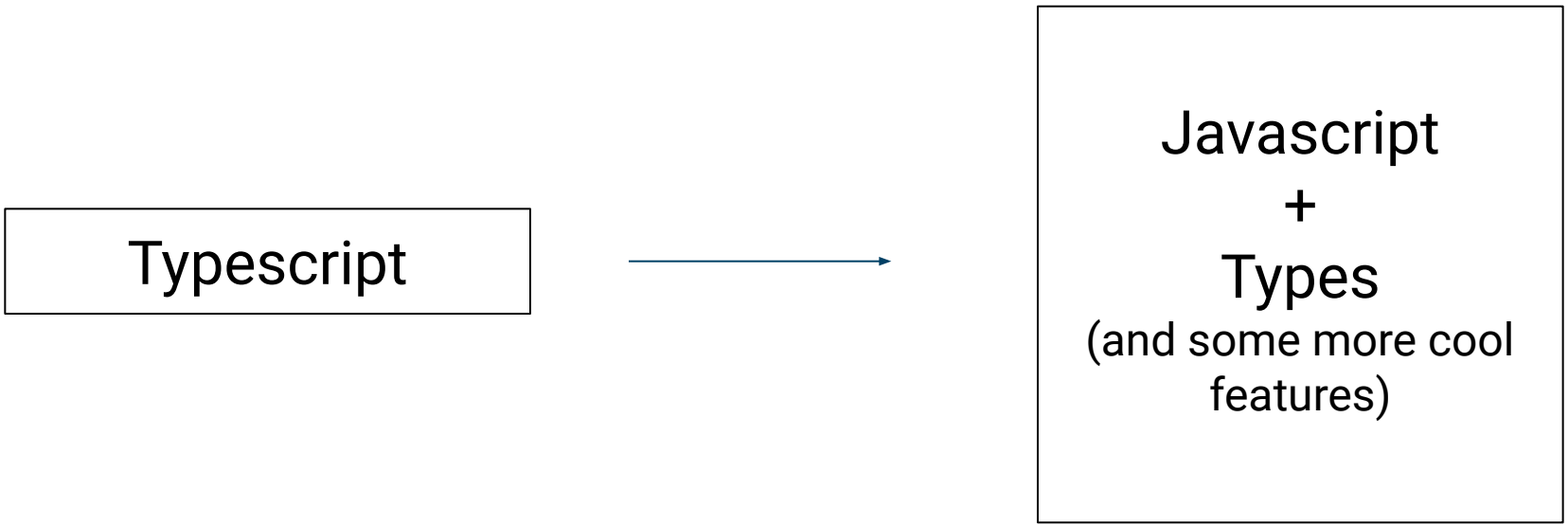
~~Compilation~~ Transpilation



Transpilation



Typescript



Typescript

Let keyword , Arrow function, etc.. some of new features in js

BUT will all browsers supports this ?

Typescript takes care of this as well

Why Typescript

Provides a way to state the data types for variables and objects.

JavaScript is introduced as a client-side programming language but JavaScript can be used as a server-side programming language also.

JavaScript became complex and heavy for usage other than client-side or frontend. Because of this, **JavaScript was even not able to fulfill the requirement of Object-oriented programming language.**

This prevents JavaScript from succeeding at the enterprise level as a server-side technology. Then TypeScript was developed by the development team to bridge this gap

Where typescript is being/can be used

Typescript makes complex apps manageable…

Web	Angular, React, Vue…
Node	Nest JS, GraphQL, Feathers JS…
Apps	React Native, NativeScript, Electron...
Tooling	Webpack, Babel...

Features

Code easier to understand

TypeScript Code is converted into Plain JavaScript Code: TypeScript code is not understandable by the browsers. Code is compiled and converted into JavaScript.
The above process is known as **Trans-piled**.

JavaScript is TypeScript: Whatever code is written in JavaScript can be converted to TypeScript by changing the extension from .js to .ts.

Use TypeScript anywhere: TypeScript code can be run on any browser, devices or in any operating system. TypeScript is not specific to any Virtual-machine etc.

TypeScript supports JS libraries: Existing JavaScript code can be used, incorporate popular JavaScript libraries, and can be called from other JavaScript code.

Advantages

TypeScript always **point out the compilation errors** at the time of development only. Because of this at the run-time the chance of getting errors are very less whereas JavaScript is an interpreted language.

TypeScript has a feature which is **strongly-typed or supports static typing**. That means Static typing allows for checking type correctness at compile time. This is not available in JavaScript.

TypeScript is nothing but JavaScript and some additional features
All features from modern JS may not be supported in your target browser but TypeScript compiler **can compile the .ts files into ES3,ES4 and ES5 also**.

Disadvantages

...

Generally TypeScript takes time to compile the code.

Installation

Prerequisites

- Nodejs

```
> npm install -g typescript
```

IDE - Visual Studio Code

II

Basic Features and Types

Building your first TypeScript file

Create file *first.ts*

```
function first(name) {
        console.log("hello " + name);
}
```

```
> tsc first.ts
```

```
> tsc first.ts --watch
```

```
> tsc first.ts --out name.js
```

Code will get compiled and .js file will be created

Type Annotations

Type annotations in TypeScript are lightweight ways to **record the intended contract of the function or variable.**

- Official Documentation

```
function first(name: string): string {
        return "hello " + name;
}
```

TypeScript can offer static analysis based on both the structure of your code, and the type annotations you provide.

Types

Any
Number
String
Boolean
Array
Object Types : class, interface, module
Void
Enum
Tuple [string, number]
Union -> number | string
Null
Undefined

Data types

Some of the built-in types that typescript provides are:

number : 64-bit double precision numbers for both integers and fractions.

string : a sequence of characters or string type data.

void : used for functions that return nothing.

null : represents no value or null value

boolean : represents a boolean value either true or false

https://www.typescriptlang.org/docs/handbook/basic-types.html

Variables

`let` and `const` are two relatively new types of variable declarations in JS

`let` is similar to `var` in some respects, but allows users to avoid some of the common “gotchas” that users run into in JavaScript.

`const` is an augmentation of `let` in that it prevents re-assignment to a variable.

TypeScript being a superset of JavaScript, the language naturally supports `let` and `const`.

NOTE: prefer let and const over var

https://www.typescriptlang.org/docs/handbook/variable-declarations.html

Operators

In TypeScript, an operator can be classified into the following ways.

- Arithmetic operators
- Comparison (Relational) operators
- Logical operators
- Bitwise operators
- Assignment operators
- Ternary/conditional operator
- Concatenation operator
- Type Operator

https://www.javatpoint.com/typescript-operators

Type Operators

in	It is used to check for the existence of a property on an object. `console.log('make' in Bike);`
delete	It is used to delete the properties from the objects. `delete Bike.Company1;`
typeof	It returns the data type of the operand `console.log(typeof message);`
instanceof	It is used to check if the object is of a specified type or not. `console.log( arr instanceof Array );`

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];
```

The second way uses a generic array type, Array<elemType>:

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

difference between Arrays in JavaScript and TypeScript

Array *difference between Arrays in JavaScript and TypeScript*

JavaScript Arrays are flexible

```
var myArr = [ 1 , ‘abc’ , true]
```

useful in many cases

How do we have such thing in ts

Tuple

It's like an array but just set of elements

Tuple types allow you to express an array with a fixed number of elements whose types are known

```
var myTuple : [number, string] =  [1,  'abc'];
```

For array declaration you specify the data type before [ ].
For tuple data types are specified inside [ ]

Implicit typing

If you don’t explicitly declare a variable type, but you assign a value with the declaration, TypeScript implicitly assumes the type from the value being assigned

```
var  n = 1

n = ‘abc’  //not possible
```

Applies to functions as well

Functions

TypeScript adds some new capabilities to the standard JavaScript functions to make them easier to work with

Types with function

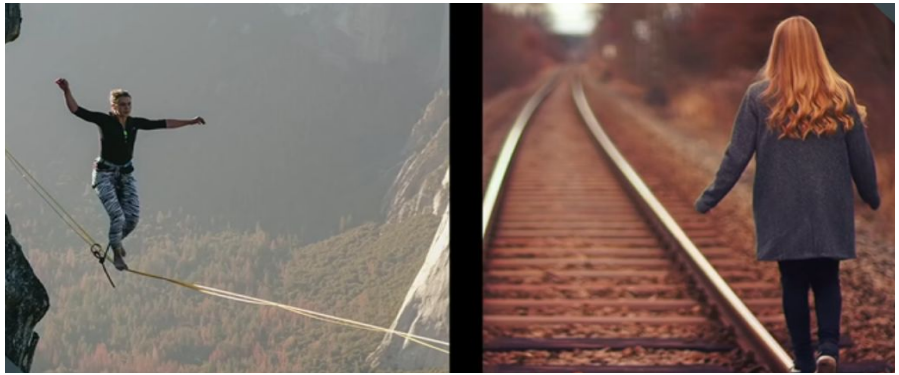
- Function arguments
  - JS accepts any number of args, TS does not
- Optional and default arguments
  - Optional args should be at very end of list ?
  - Optional with default value =
  - Can be used with types as well i.e. a?:number
- Specifying return type :

Type erasure and error behaviour

Typescripts compiler flags error during errors during development time only

Traditionally in other compilation process compilation will stop on error but typescript will continue and generate js file even with error

Development with Typescript vs Javascript



Typescript helps you stay on faster track

III

Classes, Interfaces, Modules & Namespaces

Classes, Objects, Constructor and Methods

No classes in javascript but in javascript
IIFE -*Class in javascript are annotated immediately invoked functions expressions*

Member variables can be Typed

Classes can act as a Type

Constructor->special function with name constructor
Multiple constructors not allowed (No Constructor overloading)

Methods

Instance member variable referenced using this keyword

Inheritance & Polymorphism

Extends keyword

Single, multilevel inheritance supported in TypeScript

This and Super Keyword

You can override and overload methods

Tricky overloading because of type erasure

When both functions are compiled to JavaScript, their signature is totally identical. As JavaScript doesn't have types, we end up creating two functions taking same number of arguments. So, TypeScript restricts us from creating such functions.

Overloads are recognized as possible signatures for method calls

actual implementation is not recognized as overloading

Interfaces & Duck Typing

Interfaces are Typescript only concept
Not preserved after transpilation
Also Useful to define shape of the expected data i.e. Template for Objects

The **duck-typing** feature provides type safety in TypeScript code.

Through the duck-typing rule TypeScript ***compiler checks that an object is same as other object or not.***

According to the duck-typing method, both objects must have the same properties/variables types.

Generics

Generics give us the ability to pass in a range of types to a component, adding an extra layer of abstraction and re-usability to your code.

Generic Function

```
function printData<T>( args: T) {
    console.log(args)
}
```

```
function printInfo<T extends Person>(person: T){
    console.log(person)
}
```

Generic Classes

Generic Interfaces

Modules

Modules are executed within their own scope, **not in the global scope;**

this means that variables, functions, classes, etc. declared in a module are not visible outside the module

unless they are explicitly exported using the **export**

Namespaces

It is nothing but a single global object which will contain all our functions, methods, variables

Namespace.ts

```
export var CALCULATOR = {
    ADD: function (a: number, b : number): number {},
}
```

Test.ts

```
import { CALCULATOR } from "./namespace"
console.log(CALCULATOR.ADD(1, 2))
```

IV

Development Environment

tsconfig JSON File

Configuration file which is referred by typescript compiler.

No need to manually provide command line args

`tsc --init` generates sample tsconfig file

Now providing only `>tsc` command without any file name will generate the js file for all `.ts` file in folder

Some useful configs
strict, outDir, noEmitOnError

Node Project, NPM and Type Definitions

`npm init` creates package.json File

Add script to package.json `tsc && node index.js`

`npm install lodash --save`

Type definitions `npm i @types/lodash --save-dev`

Several type definition libraries are available under @types/----- package

TypeScript Declaration File　　d.ts

TypeScript Declaration Files describes the shape of an existing JavaScript codebase to TypeScript.

The "d.ts" file is used to provide typescript type information about an API that's written in JavaScript

By using declaration files (also called .d.ts files), you can avoid misusing libraries and get things like completions in your editor.

TypeScript Declaration File　　d.ts

The idea is that you're using something like jQuery or underscore, an existing javascript library. You want to consume those from your typescript code.

Rather than rewriting jquery or underscore or whatever in typescript,
you can instead write the d.ts file, which contains only the type annotations.

Then from your typescript code you get the typescript benefits of static type checking while still using a pure JS library.

d.ts

It is nothing but a single global object which will contain all our functions, methods, variables

test.js

```
Function printString(a){}
```

test.d.ts

```
export as namespace Test
export function printString(a:string);
```

file.ts

```
import * as Test from './test'
Test.printString(“abc”)
```

V

Decorators

Decorators

Decorators are just JavaScript functions that allow us to annotate our code or hook into its behavior

Decorators let you **add information and behavior** to a class, method, property, or accessor **at runtime.**
Decorators can hook into your code and alter the behavior

Decorators are an experimental feature that may change in future releases.

Decorators

Decorators use the form **@expression**, where **expression must evaluate to a function** that will be **called at runtime** with information about the decorated declaration.

Decorators provide a way to add both annotations and a meta-programming syntax for class declarations and members.

Where can we use decorators

1. Class definitions
2. Properties
3. Methods
4. Accessors (getters/setters)
5. Parameters

We see Decorators implemented by the **Angular** Framework for classes like @Component, properties like @ViewChild,

Class Decorator

A class decorator makes it possible to intercept the constructor of class.

They are called when the class is declared, not when a new instance is instantiated.

Most powerful characteristics of a decoractor is its ability to reflect metadata, but the casual user will rarely need this feature.

It is more suitable for use in frameworks, like the Angular Compiler for example, that need to to analyze the codebase to build the final app bundle.

Class Decorator example

```
@freezeClass
class Test {}

function freezeClass(constructor: Function) {
    Object.freeze(constructor)
    Object.freeze(constructor.prototype)
}
console.log(Object.isFrozen(Test));    // true
class Test2 extends Test{              // not possible
}

var obj = new Test()
```

Decorator Factories

A Decorator Factory is simply a function that returns the expression that will be called by the decorator at runtime.

```
function decoratorFactoryFun(value:string){
    return function decoratorFun(params:any) {
        // do something with 'target' and 'value'...
    }
}
```

This just means the decorator itself is wrapped in a function so we can pass custom arguments to it

Decorator Evaluation

There is a **well defined order** to how decorators applied to various declarations inside of a class are applied:

1. Parameter Decorators, followed by Method, Accessor, or Property Decorators are applied for each instance member.
2. Parameter Decorators, followed by Method, Accessor, or Property Decorators are applied for each static member.
3. Parameter Decorators are applied for the constructor.
4. Class Decorators are applied for the class

References

https://www.typescriptlang.org/docs

https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/

Youtube - Typescript Basics

https://www.youtube.com/playlist?list=PLqq-6Pq4lTTanfgsbnFzfWUhhAz3tIezU