**Type Script**

Super set of Javascirpt

Developed by Microsoft

It is compiled language

It is a strongly typed language (defines variable with a specific type, functions return type )

Supports OOPS

Converts the OOPS Programming language code into JavaScript

We can easy know the type mismatch errors easily at time of compilation

Used for developing almost all modern web apps which are using js framework

Latest version 2.5 and extension is .ts

**Java Script**

**(.js)**

Employee = function(){

}

**TypeScript Code**

**(.ts)**

Class Employee{

}

**TsConfig file**

{ Config Settings}

Not Mandatory

When compilation done from TypeScript to Javascript, it generates additional file .map which is used for debugging purpose

* Javascript compile time errors cannot be shown
* Still In ES6 datatypes is not avaible where as available in TypeScript
* It is a strongly typed language (defines variable with a specific type, functions return type )
* Supports OOPS
* Default config for tsconfig we will get from vsCode, if want some other additional config setting then we need to include tsconfgi file and maully write the configs

Install and configure

Npm install –g typescript

[www.typescriptlang.org](http://www.typescriptlang.org) for reference and documentation

**DataTypes**

Number

String

Object

Enum - strongly indexed constant string values which can be represented with number values

Tuple - any in from [xdatatype,ydatatype], mostly used for graphs

Array new Array<number>()

Number[]

Date

Any

Undefined

**Define variables**

We can use var or let

Var x:string =””;

Var age :number;

Var names : string[];

**Defining function**

Function test(): void{

}

**Steps**

1. Open a new Folder in vs Code
2. Install the typescript Globally
   1. Npm install typescript –g
3. To Set the tsConfig.Json use
   1. Tsc –-init
4. Write the typescript code
5. To compile into Java Script
   1. Tsc filename.ts
6. Auotmatically typescript file will be compiled into es5 format and .js file will be generated in the same path

Explain frist Example

We can config TS by using tsc –-init at command prompt

{

"compilerOptions": {

"target": "es5",

"module": "commonjs",

"sourceMap": true,

"outDir": "./js”,

"removeComments": true,

"strict": true ,

"baseUrl": "./",

"paths": {}

// other Options

}

}

Example

function doSomething(argv:any):string|number|undefined

{

if(typeof argv == "string"){

return argv.toUpperCase();

}

else if (typeof argv == "number"){

return argv\*argv

}

else{

return undefined;

}

}

console.log(doSomething("rama"));

console.log(doSomething({name:"snou", age:"30"}))

// type assertions

var info:any = "Hello rama ";

// with info. we dont have intellisens e

var Result = <string>info;

console.log(Result.toUpperCase());

var now : any = new Date();

console.log((<Date> now).getFullYear());

**Use to the tsConfig.Json to set setting like OutDir, removeComments ,removeComments**

Optional params in TypeScript

In Javascript generally all the params to a function are optional

Inorder to implement the same in typescript we use ?

Function MyFunct(fname : string , lname?:string)

{

}

Will takes null value for lname if no value is provided for lname , where if we want to provide some default value

Function MyFunct(fname : string , lname:string=”ABC”)

{

}

**Working with Classes**

/\* general way of creating class ..

this is ES6 Style class creation

class Employee{

// dont use let or var for member variables of the class

private name: string;

public salary: number;

protected joinDate: Date;

// Access specifiers will ensure the meberscope at the time of compilation

// we will not find any accessspeciers in java script

constructor(){

this.name ="abc"; this.salary=3000; this.joinDate = new Date();

}

}

will work perfect

\*/

// TypeScript Style class creation

class Employee{

private readonly id : string;

constructor(id:string, private name:string, public salry: number, public joinDate: Date ){

this.id = id;

}

Display() we can write if want to dispaly the data

// must use this for accessing the member variables

}

**Working with Interfaces**

//interfaces will provide a comman structure across the elements

interface User {

name: string;

pwd: string;

validate(): boolean

}

//interfaces will not be shown in javascript transpiled Files

function loginDemo(userObj: User) {

if (user.validate()) {

console.log('Succesful login ..' + user.name)

}

else {

console.log('Invalid login...' + user.name)

}

}

let user = {

name: "Rama",

pwd: "123",

// we can add some additioinal members but it contain all the interface members

lastLogin : new Date(),

validate: function () {

if (this.name === "Rama" && this.pwd === "123") {

return true;

}

else {

return false;

}

}

}

loginDemo(user);