**Javascript** Demerits:

* No DataType for varaibles
* Any no. of variables can be passed to a function. That function will not enforce the user to enter required parameters only to that function. The no. of arguments to functions in Javascript fo not have to match the function signature. You can pass in more or lesser arguments than what’s declared in the function. It would call the function.
* Objects are loosely structured. That is we can add member variables to the object even it is already created. Ex. Var person(“firstName”: “Raj”, “lastName”:”Sarma”}. For this object we can add person.age = 10; This should not be allowed. There is no concept of template in javascript which will help in restricting to add member variables.

**Typescript**: (JavaScript + types and other stuff)

* **tsc** is typescript compiler
* Conversion of Typescript to Javascript happenes during development time. Typescript features exists in development time, when you compile typescript to javascript is pure javascript which doesnot have any features of Typescript that is not datatypes (concept called Type Erasure).
* It is a superset of Javascript.
* Let keyword which addresses the scoping problem in variable declaration, Arrow functions.
* Frameworks like Angular 2, React are using
* Uses NodeJs for compiling.
* tsc filename.ts for compiling
* Datatypes declaration is postfix i.e, declaring a variable and specifying datatype.
* Number, Boolean, string -> primitive datatypes
* undefined, null, symbol, Array, Tuple.
* For datatypes undefined, null, Only values we can assign to them are undefined, null respectively.
* For mixed data types in an array, we can use a Tuple. Ex:

var myArr : [number, boolean];

myArr1 = [2, false];

* You are forced to create an array which is of same size that you have declared.
* The types you are assigned are for the purposes of checking during development time, not at runtime.
* Even though we have a compilation error (it warns the developers) in typescript file (that is like assigning different type of data to other data type variable), It will compile and produces javascript file.
* By default, Typescript enforces that the argument counts in function calls exactly match the function signature.
* By using optional (which should be a last argument), we can provide a parameter value to the function. We can have any number of optional arguments

function optionalAdd(a:number, b:number, c?) {

return a+b+c;

}

* We can also add parameter value = 0 for optional.

function optionalAdd(a:number, b:number, c=0) {

return a+b+c;

}

* Return type of the function can also be specified.

function addingTs(a:number, b: number): number {

return a + b;

}

* If you don’t explicitly declare a variable type, but you assign a value with the declaration, Typescript uses implicitly assumes the type from the value being assigned.
* **any** datatype specifies compiler not to do type checking for a variable. If a variable is specified as any datatype then we can assign any value for that variable. Later we can assign any other datatype value .
* we can declare a variable with more than one datatype then we can use **union** for type checking

var variable : number | boolean | string;

variable = 'test';

* Constructor of a class need not be a name of the class. Keyword **constructor** is used as a constructor.
* We can not have multiple (overloaded) constructors.
* If an object is having same property and behavior of any instance of a class, then this concept is called as **Duck Typing**.

class Per implements Person {

firstName: string;

lastName: string;

getFullName(): string {

throw this.firstName + " "+ this.lastName;

}

}

let aPerson: Person = new Per();

let someObj = {

firstName : "Test",

lastName : "Test",

getFullName : () => "Test"

};

aPerson = someObj;

* The object must have all the data members, methods of the class. The object may contain extra variables. But we cannot access the data members from the object if we assign it with the instance of the class.
* If we don’t specify any visibility the member variable is assumed to be as public.
* In a class if we don’t want to declare variables, we can create them in the constructor itself using any modifier like (public, private, protected).
* Using readonly modifier, we can not change the value of the variable, but assignment can be done only at the time of declaring it.

readonly name = "Sarma";

* Readonly modifier can also be used in constructor for declaring a variable.

readonly name;

constructor(name : string) {

this.name = name;

}

constructor(readonly name : string) {

}

* Modules allows us to create a large file into separate files so that we can use the existing functionality into other files. This is done by using **export** keyword placed before a class, so that the respective class is eligible for importing in another class.

export class Person1 {

firstName : string;

lastName : string;

greet() {

console.log("Hey There");

}

}

* To Import respective class we use import keyboard, specify the class name in curly braces and specify the file name(.ts is not required) which we are importing.

import {Person1} from './classes1';

import \* from 'library\_name'; // For Node Modules

import \* as \_ from loadash'; // For Node Modules

* At the time of importing, we can use alias name for using that alias name in the application.
* While compiling from type script file to java script, to specify a customized file name, we can use **--out filename.js**
* To monitor a type script file for any change after compiling, that is It has to generate respective java script file, we can use **--watch**. It will continuously monitors for changes. Now if we change the code in ts file, then **watch** will automatically compiles again.
* Type script compiler has an option of specifying Configuration file, which contains all those arguments register on those commands. It refers that configuration file and gets configuration. This file is called **tsconfig.json** (used by the type script compiler looking for) which is placed in the root of the project.
* Type script compiler has an option of creating Configuration file for us using **tsc --init**.
* Once we have the **tsconfig.json** file, then we can use **tsc** command to compile the ts files.
* In the configuration file, **“strict”: true** specifies thatwe mustsettype for every variable. If we don’t specify data type to that variable then it shows error.
* In the configuration file, **“outDir”: “./folder”** specifies thatoutput folder for creating javascript files.
* In the configuration file, **“noEmitOnError”: true** don’t want to generate javascript file onlyif there are errors.
* In the configuration file, **“noEmit”: true** does notgenerates javascript files.
* To initialize a node project, we use **npm –init** which creates a package.json file for the purposes of creating a node artifact which contains name, version, publishing it **npm** repository.
* Rather than typing the typescript compiler command and executing resulting javascript file In package.json, scripts -> “start” : “tsc && node out/ index.js”. index.js js file is the main javascript file.
* To download and install the package from the repository needed in the applicationuse **npm install**.
* Use **npm install package\_name** **--save** in order to save it to local directory where you are running and making an entry into package and it gets registered as dependency. It will be saved in node\_modules folder. It will be available in our application.
* To get information from the respective library and also save type definitions in @types folder use **npm install @types/package\_name –save-dev**
* To make a project use **tsc --init** and **npm init.**

svc.getUserInfo('rajesarma', (user : User) => {

console.log(user);

}); // usnig call backs

getUserInfo(userName : string, callBack : (user: User) => any ) {

let options : any = {

headers: {

'User-Agent' : 'request'

},

json : true

}

request.get('http://localhost:8091/url\_data/'+userName, options, (error :any,response: any, body : any) => {

let user = new User(body);

callBack(user);

})

}

* **request.get(url, options, response)**. We can not have return type for an asynchronous method. We can not return the result of the asynchronous operation of the result itself from a function. In javascript, we can have promises to handle asynchronous output back to the consumer (Observables also) . But the basic way to dealing asynchronous methods by using **callbacks**.
* Call back is a function that takes user as an argument.
* The map function executes the function passed to it for each element of the array and returns a new array with the corresponding return values.
* To pass command line arguments, we use **process.argv**