**Type scripting features**

browsers transpiles typescript code into javascript

type script is super script of javascript

**Static Typing:** This means that you can declare the types of variables, and the

compiler will make sure that they aren't assigned the wrong types of values

var burger: string = 'hamburger', // String

calories: number = 300, // Numeric

tasty: boolean = true; // Boolean

**Interfaces**

Interfaces are used to type-check whether an object fits a

certain structure.

interface Food {

name: string;

calories: number;

}

**Classes**

TypeScript offers a class system that is very similar to the one in these

languages, including inheritance, abstract classes, interface implementations,

setters/getters, and more.

class Menu {

// Our properties:

// By default they are public, but can also be private or protected.

items: Array<string>; // The items in the menu, an array of strings.

pages: number; // How many pages will the menu be, a number.

// A straightforward constructor.

constructor(item\_list: Array<string>, total\_pages: number) {

// The this keyword is mandatory.

this.items = item\_list;

this.pages = total\_pages;

}

// Methods

list(): void {

console.log("Our menu for today:");

for(var i=0; i<this.items.length; i++) {

console.log(this.items[i]);

}

}}

**Generics**

Generics are templates that allow the same function to accept arguments of various different types.

function genericFunc<T>(argument: T): T[] {

var arrayOfT: T[] = []; // Create empty array of type T.

arrayOfT.push(argument); // Push, now arrayOfT = [argument].

return arrayOfT;

}

**Modules**

TypeScript introduces a syntax for exporting and importing modules,

### exporter.ts

var sayHi = function(): void {

console.log("Hello!");

}

export = sayHi;

### importer.ts

import sayHi = require('./exporter');

sayHi();

async/await

### **Async** - declares an asynchronous function

Automatically transforms a regular function into a Promise

### - pauses the execution of async functions

* When placed in front of a Promise call, await forces the rest of the code to wait until that Promise finishes and returns a result.

async function getJSONAsync(){

// The await keyword saves us from having to write a .then() block.

let json = await axios.get('https://tutorialzine.com/misc/files/example.json'); **Await**

// The result of the GET request is available in the json variable.

// We return it just like in a regular synchronous function.

return json;

}

**var vs let**

var will scope to nearest function

let will scope to nearest function cant be used outside scope

type annotation

let a: number;

**Type assertions:** tell compilor abt type of variable(as string)

**Arrow functions/lambda functions**

let dolog = (message) =>console.log(message);

Whats the diff between ECMA5 and ECMA6 Script?

What is the difference between angular and angular2?

What you are using for testing in your ang application?

Jamine and karma

Karma is test runner

Jasmine is framework

Protactor -- like Coded UI -- ENd to end testing

Why you use arrow functions in typescripts

Closure in Javascript?

Hoisting and scoping of variables in javascript?

let keyword in ES6/Typescript?(used for scoping inside a function)

export of function is possible in Angular....

Diff between Deferred execution vs Immediate execution in LINQ

The basic difference between a Deferred execution vs Immediate execution is that

Deferred execution of queries produce a sequence of values,

The query is actually executed when the query variable is iterated over,

not when the query variable is created.This is useful when, for example,

you have a database that is being updated by other applications

whereas Immediate execution of queries return a singleton value

and is executed immediately.

Examples are using Count(), Average(), Max() etc.

**IEnumerable** execute select query on server side, load data in-memory on client side

and then filter data while **IQueryable** execute select query on server side with all filters.

Inner Join in **LINQ**

var orderForBooks = from bk in bookList

join ordr in bookOrders

on bk.BookID equals ordr.BookID

select new

{

bk.BookID,

Name = bk.BookNm,

ordr.PaymentMode

};