**Common**

1. Introduce yourself and explain about your recent project
2. Explain about your application architecture of your recent project
3. Software design patterns. What are the design patterns used in your project and explain any of the patterns?
4. SOLID Principles
5. Cognitive Services

**.NET Core**

1. **.NET Core Versions –** 2.0, 2.1, 2.2, 3.0, **3.1**, **5**, 6, 7, 8.
2. **What is new in .NET Core 3.1**

This release contains minor improvements to .NET Core 3.0, focusing on small, but important, fixes. The most important feature about .NET Core 3.1 is that it's a [long-term support (3 yrs) (LTS)](https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-core-3-1#long-term-support) release.

1. **What is new in .NET Core 5**
2. **What is new in .NET Core 6**
3. **Advantages of API**

The major benefit of using an API is accessing and consuming data and services from thousands of independent sources. This means organizations of all sizes can access these functionalities without developing their own specialized applications. Other major benefits to using APIs include APIs increase business agility.

1. **What is Dot NET Core used for?**

* .NET Core is useful in the server application creations, that run on various operating systems like Windows, Mac, and Linux. Using this, developers can write libraries as well as applications in C#, F#, and VB.NET in both runtimes.
* Generally, it is used for cloud applications or for modifying large enterprise applications into microservices.
* .NET Core 3.0 supports cross-development between WPF, UWP, and Windows Forms.
* .NET Core supports microservices, which permits cross-platform services to work with the .NET Core framework including services developed with .NET Framework, Ruby, Java, etc.
* .NET Core’s features like lightweight, modularity, and flexibility make it easier to deploy .NET Core applications in containers. These containers can be deployed on any platform, Linux, cloud, and Windows.

1. **Why .Net Core & its advantages over .Net framework**

| **Features** | **.NET Core** | **.NET framework** |
| --- | --- | --- |
| **Compatibility** | It works based on the principle of “build once, run anywhere”. It is cross-platform, so it is compatible with different operating systems such as Linux, Windows, and Mac OS. | This framework is compatible with the Windows operating system only. Even though, it was developed for supporting software and applications on all operating systems. |
| **Installation** | Since it is cross-platform, it is packaged and installed independently of the OS. | It is installed in the form of a single package for Windows OS. |
| **Application Models** | It does not support developing the desktop application and it focuses mainly on the windows mobile, web, and windows store. | It is used for developing both desktop and web applications, along with that it also supports windows forms and WPF applications. |
| **Performance and Scalability** | It provides high performance and scalability. | It is less effective compared to .Net Core in terms of performance as well as scalability of applications. |
| **Support for Micro-Services and REST Services** | It supports developing and implementing the micro-services and the user is required to create a REST API for its implementation. | It does not support the microservices’ development and implementation, but it supports REST API services. |
| **Packaging and Shipping** | It is shipped as a collection of Nugget packages. | All the libraries that belong to the .Net Framework are packaged and shipped all at once. |
| **Android Development** | It is compatible with open-source mobile app platforms like Xamarin, via .NET Standard Library. Developers can make use of tools of Xamarin for configuring the mobile application for particular mobile devices like Android, iOS, and Windows phones. | It does not support the development of mobile applications. |
| **CLI Tools** | For all supported platforms, it provides lightweight editors along with command-line tools. | This framework is heavy for CLI(Command Line Interface) and developers usually prefer to work on the lightweight CLI. |
| **Deployment Model** | Updated version of the .NET Core gets initiated on one machine at a time, which means it gets updated in new folders/directories in the existing application without affecting it. Thus, we can say that .NET Core has a very good flexible deployment model. | When the updated version is released, it is deployed only on the Internet Information Server at first. |

1. **Dependency Injection & its Lifetime (Transient, Scoped, Singleton). When to use Singleton lifetime.**

What is dependency injection in .NET core?

ASP.NET Core supports the dependency injection (DI) software design pattern, which is a technique for achieving Inversion of Control (IoC) between classes and their dependencies. For more information specific to dependency injection within MVC controllers, see Dependency injection into controllers in ASP.NET Core.

Why is dependency injection important in .NET core?

beasily. You can use mock databases with dependency injection, and test your application without affecting the actual database.

Which principle is an implementation of dependency injection?

**The Inversion of Control (IoC)**

Dependency injection is a subset of the Inversion of Control (IoC) principle. The DI principle is a technique that is used to remove internal dependencies from the implementation by enabling these dependencies to be injected externally.

Is dependency injection a container?

What is DI Container.

A DI Container is a framework to create dependencies and inject them automatically when required. It automatically creates objects based on the request and injects them when required. DI Container helps us to manage dependencies within the application in a simple and easy way.

What is singleton vs transient vs scoped?

Singleton is a single instance for the lifetime of the application domain. Scoped is a single instance for the duration of the scoped request, which means per HTTP request in ASP.NET. Transient is a single instance per code request.

Dependency injection Life Time

The below three methods define the lifetime of the services,

***AddTransient***  
Transient lifetime services are created each time they are requested. This lifetime works best for lightweight, stateless services.

***AddScoped***  
Scoped lifetime services are created once per request.

***AddSingleton***  
Singleton lifetime services are created the first time they are requested (or when ConfigureServices is run if you specify an instance there) and then every subsequent request will use the same instance.

|  |  |  |
| --- | --- | --- |
| Service Type | In the scope of a given http request | Across different http requests |
| Transient | New Instance | New Instance |
| Scoped | Same Instance | New Instance |
| Singleton | Same Instance | Same Instance |

**Summary**

Let’s summarize what we discussed so far,

* With a transient service, a new instance is provided every time an instance is requested whether it is in the scope of same http request or across different http requests.
* With a scoped service we get the same instance within the scope of a given http request but a new instance across different http requests.
* With Singleton service, there is only a single instance. An instance is created, when service is first requested and that single instance single instance will be used by all subsequent http request throughout the application.

‘

<https://www.c-sharpcorner.com/article/understanding-addtransient-vs-addscoped-vs-addsingleton-in-asp-net-core/>

1. **Which methods are dependency injection?**

There are three types of dependency injection — constructor injection, method injection, and property injection.

* **Constructor Injection:** Dependency is passed to the object via its constructor that accepts an interface as an argument.
* **Method Injection:** A.k.a. interface-based injection.
* **Property Injection:** A.k.a. setter injection.

https://dotnettutorials.net/lesson/setter-dependency-injection-design-pattern-csharp/

1. **What is middleware in .NET Core?**

Middleware is software that's assembled into an app pipeline to handle requests and responses. Each component: Chooses whether to pass the request to the next component in the pipeline. Can perform work before and after the next component in the pipeline.

What are the 4 functions of middleware systems?

Data management, application services, messaging, authentication, and application programming interface (API) management are all commonly handled by middleware.

1. **What is Controller?**

Controller implements an application's business logic and acts as a bridge between the HTTP/REST API and domain/database models.

1. **What is HttpContextAccessor used for?**

It stores the request and response information, such as the properties of request, request-related services, and any data to/from the request or errors, if there are any. ASP.NET Core applications access the HTTPContext through the IHttpContextAccessor interface. The HttpContextAccessor class implements it.

1. **What is HttpContext in Web API controller?**

HttpContext encapsulates all information about an individual HTTP request and response. An HttpContext instance is initialized when an HTTP request is received. The HttpContext instance is accessible by middleware and app frameworks such as Web API controllers, Razor Pages, SignalR, gRPC, and more.

1. **What is HttpContext vs HttpApplication?**

HttpApplication is the pipeline implementation which loads the HTTP modules and handlers. Incoming requests are processed by its instances. HttpContext is just the context object, which holds the request and response messages.

1. **What are the 2 types of HTTP requests?**

The two most common HTTP methods are: GET and POST.

1. **What are the 4 main methods in HTTP?**

The primary or most commonly used HTTP methods are POST, GET, PUT, PATCH, and DELETE. These methods correspond to create, read, update, and delete (or CRUD) operations, respectively**.**

1. **What are the 8 methods of HTTP?**

|  |  |
| --- | --- |
| **SNO** | **Method and Description** |
| 1 | **GET**  The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data. |
| 2 | **HEAD**  Same as GET but transfers the status line and header section only. |
| 3 | **POST**  A POST request is used to send data to the sp;xxerver, for example, customer information, file upload, etc. using HTML forms. |
| 4 | **PUT**  Replaces all current representations of the target resource with the uploaded content. |
| 5 | **DELETE**  Removes all current representations of the target resource given by a URI. |
| 6 | **CONNECT**  Establishes a tunnel to the server identified by a given URI. |
| 7 | **OPTIONS**  Describes the communication options for the target resource. |
| 8 | **TRACE**  Performs a message loop-back test along the path to the target resource. |

[**https://www.tutorialspoint.com/http/http\_methods.htm**](https://www.tutorialspoint.com/http/http_methods.htm)

1. **What are the 3 parts of HTTP request?**

An HTTP request is made out of three components: request line, headers and message body.

1. **What are action methods in .NET Core?**

An action (or action method) is a method on a controller which handles requests. Controllers logically group similar actions together. This aggregation of actions allows common sets of rules, such as routing, caching, and authorization, to be applied collectively. Requests are mapped to actions through routing.

1. **What is the difference between @controller and @service?**

@ Service: Used to define a service class that contains business logic. @ Controller: Used to define a controller class that handles user requests and returns responses. @ Repository: Used to define a repository class that interacts with a database or persistence layer.

1. **Which API is used in controller?**
2. What is the purpose of API controller?
3. Startup class methods (Configure & Configure Service)

What is startup time limit in .NET Core?

The ASP.NET Core Module is configured with a default startupTimeLimit of 120 seconds.

1. What is the difference between ConfigureServices and configure?

***ConfigureServices():*** Registers the services that your application will need. ***Configure():*** Configures the middleware pipeline that controls how the application processes the HTTP requests and sends the response.

1. **Why is ConfigureServices used?**

The ConfigureServices method is a place where you can register your dependent classes with the built-in IoC container. After registering dependent class, it can be used anywhere in the application. You just need to include it in the parameter of the constructor of a class where you want to use it.

1. Authorization / Authentication filters
2. **What is the maximum query length in .NET Core?**

Property Value. The maximum length of the query string, in number of characters. The default is **2048**.

What is the maximum GET request size?

Internet Explorer also has a maximum path length of **2,048** characters. This limit applies to both POST request and GET request URLs. If you are using the GET method, you are limited to a maximum of 2,048 characters, minus the number of characters in the actual path.

1. **What is the maximum length of URL in asp net core 6?**

It's **4096** characters in ASP.Net but it can be configured using Web.

1. **What is the request size limit in NET Core?**

ASP.NET Core 2.0 enforces 30MB (~28.6 MiB) max request body size limit, be it Kestrel and HttpSys. Under normal circumstances, there is no need to increase the size of the HTTP request. But when you are trying to upload large files (> 30MB), there is a need to increase the default allowed limit.

1. **What is the default HTTP timeout in .NET Core?**

The default value is **100,000 milliseconds (100 seconds).**

1. **What is the maximum JSON length in .NET core?**

The maximum length of JSON strings. The default is **2097152 characters**, which is equivalent to 4 MB of Unicode string data.

1. **What is the default request body size?**

The Request Body Size Limit policy, which by default is set to **1 MB,** is applied to all API gateway proxy requests. If the request exceeds this limit, it will be blocked and an error code will be returned.

1. **How do I increase API timeout in net core?**

There is no way to increase the Timeout period on your web api using code, with . Net Core. The only solution is to provide a web. config, that you add manually to your project.

1. **What is the default timeout for REST API call?**

The default value of the REST client response timeout is 120 seconds. You can increase this time if an adapter that you use has longer than normal response times.

1. **What is the maximum timeout for Httpclient?**

The default value is currently 100000 ms (100 seconds).

1. **How do I expire a session in .NET Core?**

The default Session Timeout in ASP.Net Core is 20 minutes.

public void ConfigureServices(IServiceCollection services)

{

services. AddMvc();

// Set Session Timeout. Default is 20 minutes.

services.AddSession(options =>

{

options.IdleTimeout = TimeSpan.FromMinutes(30);

});

1. **What is the difference between user roles and user claims?**

Role-based authorization requires first identifying the user, then ascertaining the roles to which the user is assigned, and finally comparing those roles to the roles that are authorized to access a resource. In contrast, a claim is not group based, rather it is identity based.

1. **How does HttpContext current user identity name work?**

When the HttpContext.Current.User.Identity.Name property is called, . NET uses the authentication information stored in the cookie to identify the current user and retrieve their name.

1. **What is the difference between middleware and Httpmodule in .NET core?**

The MiddleWare components, we use to set up the request processing pipe line for ASP.net core application. The HttpModules , we use to set up the request processing pipe line for ASP.net Application. It is the pipeline which determines how to process the incoming httprequest and out going httpresponse.

1. **How do I add cookies to .NET Core?**

Creating a cookie in ASP.NET Core is simple. First, create a new CookieOptions object as shown in the code example given below: var cookieOptions = new CookieOptions();

1. **Where you will implement singleton pattern? and In .Net core we will include singleton, where will place that code?**

In Start Up.

public Startup(IConfiguration configuration)

{

var config = new ConfigurationBuilder()

.SetBasePath(System.IO.Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true).Build();

LogManager.Configuration = new

NLogLoggingConfiguration(config.GetSection("NLog"));

Configuration = configuration;

}

public static IConfiguration Configuration { get; set; }

public void ConfigureServices(IServiceCollection services)

{

services.TryAddSingleton<IHttpContextAccessor, HttpContextAccessor>();

services.AddMvc(config => { config.Filters.Add(typeof(ExceptionFilter)); });

services.AddSingleton<IConfiguration>(Configuration);

services.AddSingleton<ISmartEnggLogging, SmartEnggLogging>();

services.AddSingleton(CommonSettingsSingleton.GetInstance(Configuration));

CommonProperties obj = CommonSettingsSingleton.GetSettingValues();

//services.AddCors(); //commented for CORS issue

services.AddHsts(option => {

option.Preload = false;

option.IncludeSubDomains = true;

option.MaxAge = System.TimeSpan.FromDays(365);

});

services.AddControllers();

//Dependency Injection with Addscoped

services.AddScoped<CacheManager, CacheManager>();

services.AddScoped<IBotManagementUtilities, BotManagementUtilitiesBL>();

services.AddScoped<BotManagementUtilitiesDAL, BotManagementUtilitiesDAL>()

// SAST - HttpOnlyCookies and Insecure Cookie

services.AddCookiePolicy(opts =>

{

opts.HttpOnly = Microsoft.AspNetCore.CookiePolicy.HttpOnlyPolicy.Always;

opts.Secure = CookieSecurePolicy.Always;

});

string filePath = Configuration.GetValue<string>("commonfilepath");

if (obj.KeycloakDetails.KeycloakEnabled != "Yes")

{

services.AddAuthentication(IISDefaults.AuthenticationScheme);

}

else

{

TokenValidation.Validate(services, filePath);

}

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseRouting();

//RP Issue - Fix

//Global response Header filter

app.Use(async (context, next) =>

{

context.Response.OnStarting(() =>

{

context.Response.Headers.Add("X-XSS-Protection", "1; mode = block");

context.Response.Headers.Add("X-Content-Type-Options", "nosniff");

return Task.FromResult(0);

});

await next();

});

//RP Issue - Fix

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

if (!env.IsDevelopment())

{

app.UseHsts();

}

}

Is serializable a singleton class?

Suppose if you serialize the Singleton class, and then again de-serialize that object, it will create a new instance, hence deserialization will break the Singleton pattern. The below code is used to illustrate how the Singleton pattern breaks with deserialization. Implements Serializable interface for Singleton Class

1. **Is DbContext singleton or scoped?**

The DbContext is a singleton class that represents the gateway to all data access, and therefore should not be instantiated more than once.

1. **What is the use of Appsettings.json in Web API?**

The appsettings. json file is generally used to store the application configuration settings such as database connection strings, any application scope global variables, and much other information.

1. How to implement two different URL API call in single controller method API.
2. **What is routing? How it is setup in routing file?**

Routing is responsible for matching incoming HTTP requests and dispatching those requests to the app's executable endpoints. Endpoints are the app's units of executable request-handling code. Endpoints are defined in the app and configured when the app starts.

1. **What is change token in ASP.NET Core?**

Change tokens are a helpful feature in. NET 7 for tracking changes to a particular resource or piece of data. This can be useful for implementing caching or for other scenarios where you want to be notified when a resource has been modified.

1. **What is scaffolding in .NET core?**

Scaffolding in ASP.NET Core is a technique used to generate code at design time to support a number of common application scenarios when working with Entity Framework Core. The code generation tool is available as a Nuget package.

1. **What are extension methods in .NET core?**

Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. Extension methods are static methods, but they're called as if they were instance methods on the extended type.

1. **What is the maximum URL length for Web API?**

The REST API supports Uniform Resource Locators (URLs) with a length of up to 6000 characters. To avoid exceeding this limit, it is important to be aware of URL encoding. Some frameworks and HTTP clients automatically encode URLs.

1. **‘What is CORS in .NET Core?**

CORS means cross-origin resource sharing. CORS is a mechanism—an HTTP protocol, to be exact—that allows web applications to access resources hosted on different domains (or origins.)

Why is CORS used?

CORS provides a number of benefits: It allows browsers to enforce the same-origin policy, which is a security measure that prevents a malicious script from accessing resources that it should not have access. It allows restricted resources on a web page to be requested from another domain.

1. **How do I resolve a CORS problem?**

To get rid of a CORS error, you can download a browser extension like *CORS Unblock*. The extension appends Access-Control-Allow-Origin: \* to every HTTP response when it is enabled. It can also add custom Access-Control-Allow-Origin and Access-Control-Allow-Methods headers to the responses.

1. **How do I activate CORS?**

* In middleware using a named policy or default policy.
* Using endpoint routing.
* With the [EnableCors] attribute.

1. **What are the three types of CORS?**

CORS requests are of three types: Simple, Preflight, and Request with Credentials. Simple requests are used to perform safe operations like an HTTP GET method. Preflight requests are for performing operations with side-affects like PUT and DELETE methods.

1. **Filters in .NET Core**

Filters in ASP.NET Core allow code to run before or after specific stages in the request processing pipeline. 1. Built-in filters handle tasks such as: Authorization, preventing access to resources a user isn't authorized for. 2. Response caching, short-circuiting the request pipeline to return a cached response.

Custom filters can be created to handle cross-cutting concerns. Examples of cross-cutting concerns include error handling, caching, configuration, authorization, and logging. Filters avoid duplicating code. For example, an error handling exception filter could consolidate error handling.

**How filters work**

Filters run within the *ASP.NET Core action invocation pipeline*, sometimes referred to as the *filter pipeline*. The filter pipeline runs after ASP.NET Core selects the action to execute:

**Filter Types**

Each filter type is executed at a different stage in the filter pipeline:

* [*Authorization filters*](https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-7.0#authorization-filters)*:*
  + Run first.
  + Determine whether the user is authorized for the request.
  + Short-circuit the pipeline if the request is not authorized.
* [*Resource filters*](https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-7.0#resource-filters)*:*
  + Run after authorization.
  + [OnResourceExecuting](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.filters.iresourcefilter.onresourceexecuting) runs code before the rest of the filter pipeline. For example, OnResourceExecuting runs code before model binding.
  + [OnResourceExecuted](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.filters.iresourcefilter.onresourceexecuted) runs code after the rest of the pipeline has completed.
* [*Action filters*](https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-7.0#action-filters)*:*
  + Run immediately before and after an action method is called.
  + Can change the arguments passed into an action.
  + Can change the result returned from the action.
  + Are **not** supported in Razor Pages.
* [*Endpoint filters*](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis/min-api-filters)*:*
  + Run immediately before and after an action method is called.
  + Can change the arguments passed into an action.
  + Can change the result returned from the action.
  + Are **not** supported in Razor Pages.
  + Can be invoked on both actions and route handler-based endpoints.
* [*Exception filters*](https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-7.0#exception-filters)
  + Apply global policies to unhandled exceptions that occur before the response body has been written to.
* [*Result filters*](https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-7.0#result-filters)*:*
  + Run immediately before and after the execution of action results.
  + Run only when the action method executes successfully.
  + Are useful for logic that must surround view or formatter execution.

<https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-7.0>

**C# with OOPs**

1. **What is Class?**

A Class define the structure using methods and properties/fields that resemble real-world entity.

1. **What are Objects?**

Objects are instances of the class that holds different data in properties/fields and can interact with other objects.

1. **What are Object-oriented Design Principles?**

There are various object-oriented principles and techniques using which you can develop applications that are maintainable and extendable.

The followings are four main principles of object-oriented programming:

* **Abstraction**

Data abstraction is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either abstract classes or interfaces. Abstraction in C# is the process to hide the internal details and showing functionality only.

* **Encapsulation**

Encapsulation is the concept of wrapping data into a single unit. It collects data members and member functions into a single unit called class. The purpose of encapsulation is to prevent alteration of data from outside. This data can only be accessed by getter functions of the class.

* **Inheritance**

Inheritance allows you to define a child class that reuses (inherits), extends, or modifies the behavior of a parent class. The class whose members are inherited is called the base class.

* **Polymorphism**

Polymorphism allows a class to have multiple implementations with the same name.

There are two types of polymorphism in C#:

* + **Static / Compile Time Polymorphism – Method & Operator Overloading.** It is also known as Early binding. It is also known as Compile Time Polymorphism because the decision of which method is to be called is made at compile time.

Here C# compiler checks the number of parameters passed and the parameter type, decides which method to call, and throws an error if no matching method is found.

* + **Dynamic / Runtime Polymorphism** – Virtual / Overriding Method.

Dynamic/runtime polymorphism is also known as late binding. Here, the method name and the method signature (the number of parameters and parameter type must be the same and may have a different implementation). Method overriding is an example of dynamic polymorphism.

Method overriding can be done using inheritance. With method overriding, it is possible for the base class and derived class to have the same method name and the same something. The compiler would not be aware of the method available for overriding the functionality, so the compiler does not throw an error at compile time. The compiler will decide which way to call at runtime, and if no method is found, it throws an error.

**Other points about polymorphism:**

* Method Overriding differs from shadowing.
* Using the "new" keyword, we can hide the base class member."
* We can prevent a derived class from overriding virtual members.

public class Y : X { public **sealed** override void A()}

* We can access a base class virtual member from the derived class.

1. **What is namespace in C#?**  
   The namespace keyword is used to declare a scope that contains a set of related objects. You can use a namespace to organize code elements and to create globally unique types.
2. **What are Methods?**

A method represents a particular behavior. It performs some action and might return information about an object or update an object’s data.

1. **What are Properties?**

A property is a member that provides a flexible mechanism to read, write, or compute the value of a private field. Properties can be used as if they're public data members, but they're special methods called accessors.

Properties hold the data temporarily during the execution of an application.

1. **What is Shadowing?**

Shadowing refers to the ability of a derived class to define a member with the same name as a member in its base class. This is achieved by using the new keyword in the derived class member definition.

1. **What is Method Shadowing?**

When superclass and subclass contain the same method including parameters and if they are static. The method in the superclass will be hidden by the one that is in the subclass. This mechanism is known as method shadowing.

1. **What is the difference between variable hiding and shadowing?**

Variable Hiding happens when a variable declared in the child class has the same name as the variable declared in the parent class. In contrast, variable shadowing happens when a variable in the inner scope has the same name as the variable in the outer scope.

1. **What is the difference between overriding and shadowing in C#?**

Shadowing redefines the complete method, whereas overriding redefines only the implementation of the method. In Overriding, you can access the base class using the child class’s object overridden method.

1. **How to implement shadowing in C#?**

The shadowing and method overriding both can be achieved together by using the virtual modifier with the new keyword i.e we can hide the base class normal method as virtual. We can hide base class virtual methods by using the "new" keyword. ie we can use the new keyword with virtual keywords together.

1. **What is method overriding C#?**

An override method provides a new implementation of the method inherited from a base class. The method that is overridden by an override declaration is known as the overridden base method. An override method must have the same signature as the overridden base method.

1. **What are constructors in C#?**

A constructor is a special method that is used to initialize objects. The advantage of a constructor is that it is called when an object of a class is created.

How many types of constructors are there in C#?

5 Types of Constructors in C#. They are

* ***Default Constructor***

A constructor with no parameters is called a default constructor. A default constructor has every instance of the class to be initialized to the same values. The default constructor initializes all numeric fields to zero and all string and object fields to null inside a class.

***Example***

// C# Program to illustrate calling a Default constructor

using System;

namespace DefaultConstructorExample {

class Geek {

int num;

string name;

// this would be invoked while the object of that class created.

Geek()

{

Console.WriteLine("Constructor Called");

}

// Main Method

public static void Main()

{

// this would invoke default constructor.

Geek geek1 = new Geek();

// Default constructor provides the default values to the int and object as 0, null

// Note: It Give Warning because Fields are not assign

Console.WriteLine(geek1.name);

Console.WriteLine(geek1.num);

} } }

* ***Static Constructor***

A static constructor is used to initialize any [static](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static) data, or to perform a particular action that needs to be performed only once. It is called automatically before the first instance is created or any static members are referenced. A static constructor will be called at most once.

**Points To Remember:**

* It can’t be called directly.
* When it is executing then the user has no control.
* It does not take access modifiers or any parameters.
* It is called automatically to initialize the class before the first instance created.

***Example***

class SimpleClass

{

// Static variable that must be initialized at run time.

static readonly long baseline;

// Static constructor is called at most one time, before any

// instance constructor is invoked or member is accessed.

static SimpleClass()

{

baseline = DateTime.Now.Ticks;

}

}

There are several actions that are part of static initialization. Those actions take place in the following order:

* *Static fields are set to 0*. This is typically done by the runtime.
* *Static field initializers run*. The static field initializers in the most derived type run.
* *Base type static field initializers run*. Static field initializers starting with the direct base through each base type to [System.Object](https://learn.microsoft.com/en-us/dotnet/api/system.object).
* *Base static constructors run*. Any static constructors, starting with [Object.Object](https://learn.microsoft.com/en-us/dotnet/api/system.object.-ctor) through each base class to the direct base class.
* *The static constructor runs*. The static constructor for the type runs.
* ***Private Constructor***

A private constructor is a special instance constructor. It is generally used in classes that contain static members only. If a class has one or more private constructors and no public constructors, other classes (except nested classes) cannot create instances of this class.

**Points To Remember:**

* It is the implementation of a singleton class pattern.
* use private constructor when we have only static members.
* Using private constructor, prevents the creation of the instances of that class.

***Example***

public class Counter

{

private Counter() { }

public static int currentCount;

public static int IncrementCount()

{

return ++currentCount;

}

}

class TestCounter

{

static void Main()

{

// If you uncomment the following statement, it will generate

// an error because the constructor is inaccessible:

// Counter aCounter = new Counter(); // Error

Counter.currentCount = 100;

Counter.IncrementCount();

Console.WriteLine("New count: {0}", Counter.currentCount);

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

// Output: New count: 101

* ***Parameterized Constructor*** and

A constructor having at least one parameter is called as parameterized constructor. It can initialize each instance of the class to different values.

***Example***

// C# Program to illustrate calling of parameterized constructor.

using System;

namespace ParameterizedConstructorExample {

class Geek {

// data members of the class.

String name;

int id;

// parameterized constructor would initialized data members with

// the values of passed arguments while object of that class created.

Geek(String name, int id)

{

this.name = name;

this.id = id;

}

// Main Method

public static void Main()

{

// This will invoke parameterized constructor.

Geek geek1 = new Geek("GFG", 1);

Console.WriteLine("GeekName = " + geek1.name +

" and GeekId = " + geek1.id);

}}}

* ***Copy Constructor***

This constructor creates an object by copying variables from another object. Its main use is to initialize a new instance to the values of an existing instance.

***Example:***

// C# Program to illustrate calling a Copy constructor

using System;

namespace copyConstructorExample {

class Geeks {

private string month;

private int year;

// declaring Copy constructor

public Geeks(Geeks s) {

month = s.month;

year = s.year;

}

// Instance constructor

public Geeks(string month, int year) {

this.month = month;

this.year = year;

}

// Get details of Geeks

public string Details {

get {

return "Month: " + month.ToString() +

"\nYear: " + year.ToString();

}}

// Main Method

public static void Main() {

// Create a new Geeks object.

Geeks g1 = new Geeks("June", 2018);

// here is g1 details is copied to g2.

Geeks g2 = new Geeks(g1);

Console.WriteLine(g2.Details);

} } }

Can we overload constructor in C#?

We can overload constructors in C# just like methods. We can do so by changing the signatures by using a different number, or type of parameters. Given that we have more than one type of parameter in the constructor, we can also change their order and achieve constructor overloading. Depending upon the number and type of arguments passed, the corresponding constructor is called.

Can we override constructor in C#?

No, you can't override constructors. The concept makes no sense in C#, because constructors simply aren't invoked polymorphically. You always state which class you're trying to construct, and the arguments to the constructor.

Can a constructor be inherited?

Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

Which constructor is called first in C#?

A static constructor is called automatically. It initializes the class before the first instance is created or any static members declared in that class (not its base classes) are referenced. A static constructor runs before an instance constructor.

What if copy constructor is private?

The copy constructor can be defined as private. If we make it private, then the objects of the class can't be copied. It can be used when a class used dynamic memory allocation.

Is class static or non-static?

The difference between a static class and a non-static class is that a static class cannot be instantiated or inherited and that all of the members of the class are static in nature. To declare a class as static, you should mark it with the static keyword in the class declaration.

Why class is not static?

Because the static keyword is meant for providing memory and executing logic without creating Objects, a class does not have a value logic directly, so the static keyword is not allowed for outer class and mainly as mentioned above static can't be used at Package level. It only used within the Class level.

What is the difference between constructor and destructor?

A constructor allows an object to initialize some of its value before it is used. A destructor allows an object to execute some code at the time of its destruction.

Can we call constructor without creating object?

NO. You can't invoke a constructor without creating an object.

Can we declare constructor as private?

A private constructor in Java is used in restricting object creation. It is a special instance constructor used in static member-only classes. If a constructor is declared as private, then its objects are only accessible from within the declared class. You cannot access its objects from outside the constructor class.

Which constructor has no return type?

A constructor cannot have a return type (not even a void return type). A common source of this error is a missing semicolon between the end of a class definition and the first constructor implementation. The compiler sees the class as a definition of the return type for the constructor function and generates C2533.

Can we declare constructor as final?

No, a constructor can't be made final. A final method cannot be overridden by any subclasses.

How many constructors can we declare inside a class?

There's no limit on the number of constructors a class can have.

1. **What are unmanaged objects in C#**

Un-Managed objects are created outside the control of .NET libraries and are not managed by CLR, example of such unmanaged code is COM objects, file streams, connection objects, Interop objects. (Basically, third party libraries that are referred in .NET code.)

1. **What are Finalizers and Dispose?**

Finalizers (historically referred to as **destructors**) are used to perform any necessary final clean-up when a class instance is being collected by the garbage collector. In most cases, you can avoid writing a finalizer by using the [System.Runtime.InteropServices.SafeHandle](https://learn.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.safehandle) or derived classes to wrap any unmanaged handle.

Clean up un-managed resources

The [Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) method is primarily implemented to release unmanaged resources. When working with instance members that are [IDisposable](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable) implementations, it's common to cascade [Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) calls. There are additional reasons for implementing [Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose), for example, to free memory that was allocated, remove an item that was added to a collection, or signal the release of a lock that was acquired.

The [.NET garbage collector](https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/) does not allocate or release unmanaged memory.

Dispose() and Dispose(bool)

The [IDisposable](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable) interface requires the implementation of a single parameterless method, [Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose). Also, any non-sealed class should have an additional Dispose(bool) overload method.

***Method signatures are:***

public non-virtual (NotOverridable in Visual Basic) ([IDisposable.Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) implementation).

protected virtual (Overridable in Visual Basic) Dispose(bool).

The Dispose() method

Because the public, non-virtual (NotOverridable in Visual Basic), parameterless Dispose method is called when it is no longer needed (by a consumer of the type), its purpose is to free unmanaged resources, perform general cleanup, and to indicate that the finalizer, if one is present, doesn't have to run. Freeing the actual memory associated with a managed object is always the domain of the [garbage collector](https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/). Because of this, it has a standard implementation:

public void Dispose()

{

// Dispose of unmanaged resources.

Dispose(true);

// Suppress finalization.

GC.SuppressFinalize(this);

}

The Dispose method performs all object cleanup, so the garbage collector no longer needs to call the objects' [Object.Finalize](https://learn.microsoft.com/en-us/dotnet/api/system.object.finalize) override. Therefore, the call to the [SuppressFinalize](https://learn.microsoft.com/en-us/dotnet/api/system.gc.suppressfinalize) method prevents the garbage collector from running the finalizer. If the type has no finalizer, the call to [GC.SuppressFinalize](https://learn.microsoft.com/en-us/dotnet/api/system.gc.suppressfinalize) has no effect. Note that the actual cleanup is performed by the Dispose(bool) method overload.

The Dispose(bool) method overload

In the overload, the disposing parameter is a [Boolean](https://learn.microsoft.com/en-us/dotnet/api/system.boolean) that indicates whether the method call comes from a [Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) method (its value is true) or from a finalizer (its value is false).

protected virtual void Dispose(bool disposing)

{

if (\_disposed)

{

return;

}

if (disposing)

{

// TODO: dispose managed state (managed objects).

}

// TODO: free unmanaged resources (unmanaged objects) and override a finalizer below.

// TODO: set large fields to null.

\_disposed = true;

}

***Important***

The disposing parameter should be false when called from a finalizer, and true when called from the [IDisposable.Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) method. In other words, it is true when deterministically called and false when non-deterministically called.

The body of the method consists of three blocks of code:

* A block for conditional return if object is already disposed.
* A block that frees unmanaged resources. This block executes regardless of the value of the disposing parameter.
* A conditional block that frees managed resources. This block executes if the value of disposing is true. The managed resources that it frees can include:
* Managed objects that implement [IDisposable](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable). The conditional block can be used to call their [Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) implementation (cascade dispose). If you have used a derived class of [System.Runtime.InteropServices.SafeHandle](https://learn.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.safehandle) to wrap your unmanaged resource, you should call the [SafeHandle.Dispose()](https://learn.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.safehandle.dispose#system-runtime-interopservices-safehandle-dispose) implementation here.
* Managed objects that consume large amounts of memory or consume scarce resources. Assign large managed object references to null to make them more likely to be unreachable. This releases them faster than if they were reclaimed non-deterministically.

If the method call comes from a finalizer, only the code that frees unmanaged resources should execute. The implementer is responsible for ensuring that the false path doesn't interact with managed objects that may have been disposed. This is important because the order in which the garbage collector disposes managed objects during finalization is non-deterministic.

1. **What are Interfaces?**

An interface is a contract that defines the set of rules for a particular functionality. They are used effectively with classes using OOP principles like inheritance and polymorphism to make applications more flexible.

Why use interface in C#?

That capability is important in C# because the language doesn't support multiple inheritance of classes. In addition, you must use an interface if you want to simulate inheritance for structs, because they can't actually inherit from another struct or class.

Can interfaces have fields?

An interface can't contain instance fields, instance constructors, or finalizers. Interface members are public by default, and you can explicitly specify accessibility modifiers, such as public, protected, internal, private, protected internal, or private protected .

**Interface IIntf**

**{**

**Method 1**

**}**

**Class Cls : IIntf**

**{**

**Method 1() { }**

**}**

**How to call interface methods**

private readonly IIntf \_iIntf;

public ClassContructor(IIntf \_intf)

{

\_iIntf = \_intf;

}

\_iIntf.Method1();

**Use Cases:**

1. **In interface we have 5 methods, do we implement all the 5 methods in child class or only required method we will implement?**

We need to implement all 5 methods in child class. Otherwise it will throw run time errors.

1. **Abstract**

Why abstract class is used in C#?

The purpose of an abstract class is to provide a common definition of a base class that multiple derived classes can share.

What is abstract class in C#?

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class). Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the derived class (inherited from).

Can abstract class be sealed in C#?

When a class is declared sealed, it cannot be inherited, abstract classes cannot be declared sealed. To prevent being overridden, use the sealed in C#. When you use sealed modifiers in C# on a method, then the method loses its capabilities of overriding.

Can we inherit abstract class?

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

Can we extend abstract class?

An abstract class can extend another abstract class. And any concrete subclasses must ensure that all abstract methods are implemented. Abstract classes can themselves have concrete implementations of methods. These methods are inherited just like a method in a non-abstract class.

Can we have constructor in abstract class C#?

Yes, we can have constructor in Abstract class, and we can call this constructor from derived class.

Is abstract class static or non static?

If you declare a method in a class abstract to use it, you must override this method in the subclass. But, overriding is not possible with static methods. Therefore, an abstract method cannot be static.

1. **What is the difference between Interface and abstract class?**

|  |  |  |
| --- | --- | --- |
| **S.- No.** | **Abstract Class** | **Interface** |
| 1 | The special class which cannot be instantiated is known as abstract class. | The interface enables us to determine the functionality or functions but cannot implement that. |
| 2 | Abstract classes have static members. | Interface does not have static members. |
| 3 | They have a constructor. | They don’t have a constructor. |
| 4 | It includes both a declaration and an explanation. | It includes only a declaration. |
| 5 | Here the performance is fast. | Here the performance is slow. |
| 7 | A class has the liberty to only utilize a single abstract class. | Here a class has the liberty to utilize multiple interfaces. |
| 8 | It is utilized to execute the core identity of class. | It is utilized to execute the peripheral skills of the class. |
| 9 | It includes methods, fields, constants, etc. | It only includes methods . |
| 10 | Abstract class can be fully, partially or not implemented. | Interfaces can be fully implemented. |

What is abstract and interface in C#?

The special class which cannot be instantiated is known as abstract class, whereas the interface enables us to determine the functionality or functions but cannot implement that.

1. Explain with real time experience how you implement using interface class and abstract class?
2. **Static**

**Static Class**

A static class cannot be instantiated. All members of a static class are static and are accessed via the class name directly, without creating an instance of the class.

public static class CSharpCorner

{

// Static fields

public static string Name = "C# Corner";

public static string Founder = "Mahesh Chand";

public static DateTime YearFounded = new DateTime(2000, 01, 01);

public static string Location = "Downingtown, PA";

public static string Description =

"Online community of software and data developers";

public static int GetAgeOfWebsite()

{

return 1;

}

}

Static classes have the following characteristics:

* Static classes cannot contain Instance Constructors.
* Static classes contain only static members.
* Static classes cannot be instantiated.
* Static classes are sealed. That means you cannot inherit other classes from instance classes.

**Static Members**

A static or non-static class static constructors, properties, methods, fields, operators, and events. Static properties and static methods are the most-used static members.

**Static Constructor**

* Static constructors can't be parameterized.
* A static constructor has no access modifier because it doesn't have message passing and is used during domain processing.
* Static Constructor is used to initializing static data members of the class.

**Static Method**

Static methods are shared methods. They can be called with the class name and static method name only. You cannot instantiate a static method.

Static methods only use static data members to perform calculations or processing.

Can we create object for static class?

You cannot create an object of the static class; therefore the members of the static class can be accessed directly using a class name like ClassName.

Can we inherit static class?

Static classes are sealed and therefore cannot be inherited. They cannot inherit from any class or interface except Object. Static classes cannot contain an instance constructor. However, they can contain a static constructor.

Can we extend static class in C#?

No. Static classes are sealed classes so we can't inherit that.

Can we call static class?

A static method can be called directly from the class, without having to create an instance of the class. A static method can only access static variables; it cannot access instance variables. Since the static method refers to the class, the syntax to call or refer to a static method is: class name. method name.

Is a static class a singleton?

The single instance of a singleton class is static and hence an instance of the singleton class is stored in the high frequency heap.

Can we override static method?

Static methods cannot be overridden since they are bonded at compile time and method overriding relies on dynamic binding at runtime.

**Few use cases of static classes.**

* App Configuration class that has all static settings about an app, and the values of settings don’t change based on the objects or users.
* A DatabaseConfig class may have members such as database name, server name, port number, and even a connection string. We know that these values will not change for objects.
* A Math class with all static methods. Static classes are useful and provide an easy way to access their members that do not need to work differently for different objects.
* Above listed CSharpCorner class. We know the value of CSharpCorner class members, such as its founder, launch date, location, and description, will never change regardless of its objects.
* Static classes and static members are useful because they do not require instances created for each new object. That means, they consume fewer resources, and no duplication of the same class or member is needed in memory.

Static members make code cleaner.

1. **Virtual**

What is virtual method in C#?

A virtual method is one that is declared as virtual in the base class. A method is declared as virtual by specifying the keyword "virtual" in the method signature. A virtual method may or may not have a return type. Virtual methods allow subclasses of the type to override the method.

What is the difference between abstract and virtual methods?

The abstract method is a method that is declared in an abstract class but is not implemented. The virtual method is declared in a base class and has an implementation, but the child class may override the default implementation.

Can we override virtual method?

By default, methods are non-virtual, and they cannot be overridden. Virtual modifiers cannot be used with static, abstract, private, and override modifiers.

1. **What is the delegate in C#?**

A delegate is a type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type. You can invoke (or call) the method through the delegate instance.

Why use delegate in C#

Delegates can be used to invoke static and non-static methods. A delegate can be used to call one or more methods having identical signatures. Delegates can be used to define callback methods and invoke event handlers.

What are the two types of delegates in C#?

There are two types of delegates in C#, Singlecast delegates, and Multiplecast delegates. Singlecast delegate point to a single method at a time. The delegate is assigned to a single method at a time. They are derived from System

How many types of delegates are there in C#?

There are three types of delegates that can be used in C#.

What is the difference between delegate and action C#?

An action is a delegate whose signature has no parameters and has no return value. You can't use Action without using a delegate at the same time, since Action is a specific delegate type.

What is the difference between func and delegate?

Both are reference types that encapsulate a method. The Func delegate points to a method that accepts parameters and returns a value; the Action delegate points to a method that accepts parameters but does not return a value (i.e., returns void).

What is difference between events and delegates?

Events are typically public class members. By comparison, delegates are often passed as parameters and stored as private class members, if they are stored at all.

What is delegate and event in C#?

A delegate is a way of telling C# which method to call when an event is triggered. For example, if you click a Button on a form, the program would call a specific method. It is this pointer that is a delegate. Delegates are good, as you can notify several methods that an event has occurred, if you wish so.

What is alternative for delegate in C#?

Instead of using a delegate, the subscriber can implement an interface ISubscriber that has a method SomeEvent , and pass itself to the publisher (to a method with a signature Subscribe(ISubscriber subscriber) ). The publisher then will store this reference to the subscriber and call subscriber.

Can a delegate be static?

Any method from any accessible class or struct that matches the delegate type can be assigned to the delegate. The method can be either static or an instance method.

Types of Delegates

* ***Func Delegate***

Func is a generic delegate included in the System namespace. It has zero or more input parameters and one out parameter. The last parameter is considered as an out parameter.

The Func delegate that takes one input parameter and one out parameter is defined in the System namespace, as shown below:

namespace System

{

public delegate TResult Func<in T, out TResult>(T arg);

}

***Example:***

class Program

{

static int Sum(int x, int y)

{

return x + y;

}

static void Main(string[] args)

{

Func<int,int, int> add = Sum;

int result = add(10, 10);

Console.WriteLine(result);

}

}

**Points to Remember**

* Func is built-in delegate type.
* Func delegate type must return a value.
* Func delegate type can have zero to 16 input parameters.
* Func delegate does not allow ref and out parameters.
* Func delegate type can be used with an [anonymous method](https://www.tutorialsteacher.com/csharp/csharp-anonymous-method) or [lambda expression](https://www.tutorialsteacher.com/linq/linq-lambda-expression).
* ***Action Delegate***

It is built-in generic delegate types. Action is a delegate type defined in the System namespace. An Action type delegate is the same as [Func delegate](https://www.tutorialsteacher.com/csharp/csharp-func-delegate) except that the Action delegate doesn't return a value. In other words, an Action delegate can be used with a method that has a void return type.

***Example:***

public delegate void Print(int val);

static void ConsolePrint(int i)

{

Console.WriteLine(i);

}

static void Main(string[] args)

{

Print prnt = ConsolePrint;

prnt(10);

}

**Points to Remember**

* Action delegate is same as func delegate except that it does not return anything. Return type must be void.
* Action delegate can have 0 to 16 input parameters.
* Action delegate can be used with [anonymous methods](https://www.tutorialsteacher.com/csharp/csharp-anonymous-method) or [lambda expressions](https://www.tutorialsteacher.com/linq/linq-lambda-expression).
* ***Predicate Delegate***

Predicate is the delegate like [Func](https://www.tutorialsteacher.com/csharp/csharp-func-delegate) and [Action](https://www.tutorialsteacher.com/csharp/csharp-action-delegate) delegates. It represents a method containing a set of criteria and checks whether the passed parameter meets those criteria. A predicate delegate methods must take one input parameter and return a boolean - true or false.

The Predicate delegate is defined in the System namespace, as shown below:

**Predicate signature:** public delegate bool Predicate<in T>(T obj);

Same as other delegate types, Predicate can also be used with any method, anonymous method, or lambda expression.

***Example:***

static bool IsUpperCase(string str)

{

return str.Equals(str.ToUpper());

}

static void Main(string[] args)

{

Predicate<string> isUpper = IsUpperCase;

bool result = isUpper("hello world!!");

Console.WriteLine(result);

}

An anonymous method can also be assigned to a Predicate delegate type as shown below.

static void Main(string[] args)

{

Predicate<string> isUpper = delegate(string s)

{ return s.Equals(s.ToUpper());};

bool result = isUpper("hello world!!");

}

A lambda expression can also be assigned to a Predicate delegate type as shown below.

static void Main(string[] args)

{

Predicate<string> isUpper = s => s.Equals(s.ToUpper());

bool result = isUpper("hello world!!");

}

**Points to Remember:**

* Predicate delegate takes one input parameter and boolean return type.
* [Anonymous method](https://www.tutorialsteacher.com/csharp/csharp-anonymous-method) and [Lambda expression](https://www.tutorialsteacher.com/linq/linq-lambda-expression) can be assigned to the predicate delegate.
* ***Anonymous Methods.***

An anonymous method is a method without a name. Anonymous methods in C# can be defined using the delegate keyword and can be assigned to a variable of delegate type.

public delegate void Print(int value);

static void Main(string[] args)

{

Print print = delegate(int val) {

Console.WriteLine("Inside Anonymous method. Value: {0}", val);

};

print(100);

}

Anonymous methods can access variables defined in an outer function.

public delegate void Print(int value);

static void Main(string[] args)

{

int i = 10;

Print prnt = delegate(int val) {

val += i;

Console.WriteLine("Anonymous method: {0}", val);

};

prnt(100);

}

Anonymous methods can also be passed to a method that accepts the delegate as a parameter.

In the following example, PrintHelperMethod() takes the first parameters of the Print delegate:

public delegate void Print(int value);

class Program

{

public static void PrintHelperMethod(Print printDel,int val)

{

val += 10;

printDel(val);

}

static void Main(string[] args)

{

PrintHelperMethod(delegate(int val) { Console.WriteLine("Anonymous method: {0}", val); }, 100);

}

}

Anonymous methods can be used as event handlers:

saveButton.Click += delegate(Object o, EventArgs e)

{

System.Windows.Forms.MessageBox.Show("Save Successfully!");

};

**Anonymous Method Limitations**

* It cannot contain jump statement like goto, break or continue.
* It cannot access ref or out parameter of an outer method.
* It cannot have or access unsafe code.
* It cannot be used on the left side of the is operator.

**Points to Remember:**

* Anonymous method can be defined using the delegate keyword
* Anonymous method must be assigned to a delegate.
* Anonymous method can access outer variables or functions.
* Anonymous method can be passed as a parameter.
* Anonymous method can be used as event handlers

**Advantages of Action and Func Delegates**

* Easy and quick to define delegates.
* Makes code short.
* Compatible type throughout the application.

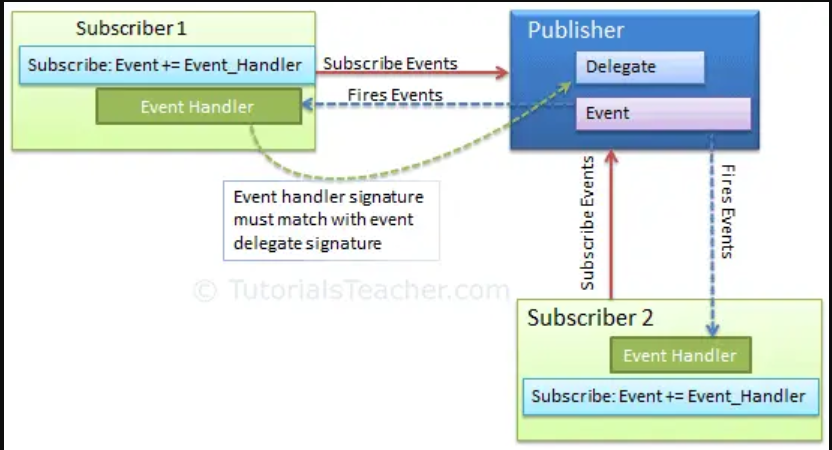
1. **What are Events?**

An event is a notification sent by an object to signal the occurrence of an action. Events in .NET follow the [observer design pattern](https://docs.microsoft.com/en-us/dotnet/standard/events/observer-design-pattern).

The class who raises events is called Publisher, and the class who receives the notification is called Subscriber. There can be multiple subscribers of a single event. Typically, a publisher raises an event when some action occurred. The subscribers, who are interested in getting a notification when an action occurred, should register with an event and handle it.

In C#, an event is an encapsulated [delegate](https://www.tutorialsteacher.com/csharp/csharp-delegates). It is dependent on the delegate. The [delegate](https://www.tutorialsteacher.com/csharp/csharp-delegates) defines the signature for the event handler method of the subscriber class.

The following figure illustrates the event in C#.



1. **Can we declare class as private?**

No, we cannot declare a top-level class as private or protected. It can be either public or default (no modifier).

1. **What is the difference between delegate and observer pattern in C#?**

In the delegate pattern, only one object can directly listen to another object's events. In the observer pattern, any number of objects can listen to a particular object's events.

1. **What is delegate vs thread C#?**

A delegate is a like a variable for a function. In other languages they are called function pointers. These variables can reference one or more functions. Threads allow for multiple execution paths to in application to exist at the same time.

1. **What is reflection in C#?**

Reflection provides objects (of type Type) that describe assemblies, modules, and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties.

1. **What is boxing and unboxing in C#?**

Boxing is the process of converting a value type to the type object or to any interface type implemented by this value type. When the common language runtime (CLR) boxes a value type, it wraps the value inside a System. Object instance and stores it on the managed heap. Unboxing extracts the value type from the object.

Why do we use boxing and unboxing?

Boxing and unboxing are both important concepts in the field of object-oriented programming. They enable a primitive type of data to be handled as an object, and vice versa. This is one of the building blocks of the unification of the Type System, which was implemented for the C# programming language.

What is the disadvantage of boxing in C#?

Boxing is an expensive process, since it copies an object from a stack to a heap which requires a number of processor as well as space on the heap. Another disadvantage of using boxing is that the same object appears at two different places in memory which can have contradictory state.

Why is boxing slow in C#?

Boxed values require an additional read  
  
Values on the stack are right there. Stick and move, stick and move! To fetch a boxed value you must first get the pointer, then look up the object. This means boxed values are slower, in addition to being larger.

1. **What is metadata in C#?**

Metadata refers to binary information saved in memory or a language runtime portable executable file. When you compile code from a portable executable file, data is added to another file section. The code is converted to MSIL (Microsoft Intermediate Language) before moving to another file partition.

1. **What is the main use of properties in C#?**

Properties enable a class to expose a public way of getting and setting values, while hiding implementation or verification code. A get property accessor is used to return the property value, and a set property accessor is used to assign a new value.

Can properties be private in C#?

Properties can be private, public, protected, or internal. In addition, properties can be any valid C# type.

1. **Difference between Field and Property in C#?**

The key difference between field and property in C# is that a field is a variable of any type that is declared directly in the class while property is a member that provides a flexible mechanism to read, write or compute the value of a private field.

1. **What is Value vs Ref C#?**

Unlike value types, a reference type doesn't store its value directly. Instead, it stores the address where the value is being stored. In other words, a reference type contains a pointer to another memory location that holds the data.

What are reference types in C#?

In C#, classes and interfaces are reference types. Variables of reference types store references to their data (objects) in memory, and they do not contain the data itself. An object of type Object, string, or dynamic is also a reference type.

* **Class types** − This reference type points to an object of a class.
* **Array types** − This reference type points to an array.
* **Interface types** − This reference type points to an object of a class which implements an interface.

1. **When to use REF vs OUT vs PARAM in C#?**

**REF** is used to state that the parameter passed may be modified by the method. **IN** is used to state that the parameter passed cannot be modified by the method. **OUT** is used to state that the parameter passed must be modified by the method.

1. **Is C# call by reference or value?**

In C#, arguments can be passed to parameters either by value or by reference. Remember that C# types can be either reference types ( class ) or value types ( struct ): Pass by value means passing a copy of the variable to the method. Pass by reference means passing access to the variable to the method.

What is the difference between copy by value and reference?

In pass-by value in JavaScript, a copy of the original variable is created, so any changes made to the copied variable do not affect the original variable. In pass-by reference in JavaScript, we pass the reference of the actual parameter. No copy is created in the memory.

1. **What is the difference between a struct and a class C#?**

Struct are value types whereas Classes are reference types. Structs are stored on the stack whereas Classes are stored on the heap. Value types hold their value in memory where they are declared, but a reference type holds a reference to an object in memory.

Which is faster class or struct?

Struct is faster than Class because: To store class, Apple first finds memory in Heap, then maintain the extra field for RETAIN count. Also, store reference of Heap into Stack. So when it comes to access part, it has to process stack and heap.

1. **What is the difference between var and dynamic in C#?**

In case of var the variable is initialized at the time of its declaration so that the compiler comes to know the type of variable according to the value assign to it. On other hand in case of dynamic it is not mandatory to be get initialized at the time of declaration.

1. **Which is read only variable?**

Read-only variables can be used to gather information about the current template, the user who is currently logged in, or other current settings. These variables are read-only and cannot be assigned a value.

Is read only immutable?

A readonly type would be defined as an immutable type or a type from a list of known readonly types. Thus, we would need to have a list of readonly types. This list would have to be user configurable so more readonly types can be added.

What is the difference between readonly and private in C#?

Use private set when you want setter can't be accessed from outside. Use readonly when you want to set the property only once.

1. **What is readonly and static in C#?**

Constant and ReadOnly keyword is used to make a field constant which value cannot be modified. The static keyword is used to make members static that can be shared by all the class objects.

What is static vs read only?

Static members can only be accessed within the static methods. The non-static methods cannot access static members. Readonly fields can be initialized at declaration or in the constructor. Therefore, readonly variables are used for the run-time constants.

1. **What is difference between constant and readonly in C#?**

The main difference between const and readonly keywords in C# is that const need to be defined at the time of assignment, while readonly field can be defined at runtime. Const's are implicitly static, while readonly values don't have to be static.

1. **What are immutable objects in C#?**

An immutable object is defined as an object that cannot be changed after it has been created. For many use cases, such as Data Transfer Objects, immutability is a desirable feature.

1. **What is difference between immutable and mutable in C#?**

A mutable object can be changed after it's created, and an immutable object can't. That said, if you're defining your own class, you can make its objects immutable by making all fields final and private. Strings can be mutable or immutable depending on the language.

1. **Are strings mutable in C#?**

String objects are immutable: they can't be changed after they've been created. All of the String methods and C# operators that appear to modify a string actually return the results in a new string object.

1. **What is the difference between immutable and constant in C#?**

A constant variable is assigned a value at the time of declaration. On the other hand, immutable is also used for the same purpose but is a little flexible as it can be assigned inside a constructor at the time of its construction.

1. **What is stack and heap in C#?**

Stack provides static memory allocation, i.e., it is used to store the temporary variables. Heap provides dynamic memory allocation. By default, all the global variables are stored in the heap. It is a linear data structure means that elements are stored in the linear manner, i.e., one data after another.

What are the 3 differences between stack and heap?

The major difference between Stack memory and heap memory is that the stack is used to store the order of method execution and local variables while the heap memory stores the objects, and it uses dynamic memory allocation and deallocation.

Why stack is faster than heap?

Stack memory size is very less when compared to Heap memory. Because of simplicity in memory allocation (LIFO), stack memory is very fast when compared to heap memory.

1. **What is lazy loading in C#?**

Lazy Loading is a technique that delays the initialization of an object. This is a new feature of C# 4.0. The basic idea of lazy loading is to load objects or data only when they are needed. A lazy loading pattern is also called Object on Demand. In other words, initialization of the object happens only on demand.

https://studysection.com/blog/lazy-loading-in-c/

1. **What is Lazy and eager in C#?**

Lazy Loading vs. Eager Loading. While lazy loading delays the initialization of a resource, eager loading initializes or loads a resource as soon as the code is executed. Eager loading also involves pre-loading related entities referenced by a resource.

1. **What is serialization in C#?**

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file. This is beneficial in terms of transmitting the converted data.

1. **What are indexes in C#?**

Indexers allow instances of a class or struct to be indexed just like arrays. The indexed value can be set or retrieved without explicitly specifying a type or instance member. Indexers resemble properties except that their accessors take parameters.

What is the difference between indexer and property in C#?

Indexers are like properties but that are accessed using index. While properties can be both static and non-static, indexers are always the instance members of a class. Hence, indexers of a class can never be static.

1. **Can I index a list in C#?**

The IndexOf method returns the first index of an item if found in the List. C# List class provides methods and properties to create a list of objects (types). For example, the IndexOf method returns the first index of an item if found in the List.

1. **Can final class be inherited?**

Final classes cannot be extended or inherited. If we try to inherit a final class, then the compiler throws an error during compilation. We can simply define a final class using the final keyword and can write the class body code according to our needs.

1. [**Shallow Copy and Deep Copy in C#**](https://www.geeksforgeeks.org/shallow-copy-and-deep-copy-in-c-sharp/)
2. **What are Generics**

Generics allow you to write a class or method that can work with any data type.

T is called type parameter, which can be used as a type of fields, properties, method parameters, return types, and delegates in the DataStore class. For example, Data is generic property because we have used a type parameter T as its type instead of the specific data type.

Why would you use generics C#?

Use generic types to maximize code reuse, type safety, and performance. The most common use of generics is to create collection classes. The . NET class library contains several generic collection classes in the System.

1. **What is Collections**

A collection is a class, so you must declare an instance of the class before you can add elements to that collection. If your collection contains elements of only one data type, you can use one of the classes in the System. Collections. Generic namespace.

## Generic Collections

## System.Collections.Generic

| Generic Collections | Description |
| --- | --- |
| [List<T>](https://www.tutorialsteacher.com/csharp/csharp-list) | Generic List<T> contains elements of specified type. It grows automatically as you add elements in it. |
| [Dictionary<TKey,TValue>](https://www.tutorialsteacher.com/csharp/csharp-dictionary) | Dictionary<TKey,TValue> contains key-value pairs. |
| [SortedList<TKey,TValue>](https://www.tutorialsteacher.com/csharp/csharp-sortedlist) | SortedList stores key and value pairs. It automatically adds the elements in ascending order of key by default. |
| [Queue<T>](https://www.tutorialsteacher.com/csharp/csharp-queue) | Queue<T> stores the values in FIFO style (First In First Out). It keeps the order in which the values were added. It provides an Enqueue() method to add values and a Dequeue() method to retrieve values from the collection. |
| [Stack<T>](https://www.tutorialsteacher.com/csharp/csharp-stack) | Stack<T> stores the values as LIFO (Last In First Out). It provides a Push() method to add a value and Pop() & Peek() methods to retrieve values. |
| Hashset<T> | Hashset<T> contains non-duplicate elements. It eliminates duplicate elements. |

## Non-generic Collections

| Non-generic Collections | Usage |
| --- | --- |
| [ArrayList](https://www.tutorialsteacher.com/csharp/csharp-arraylist) | ArrayList stores objects of any type like an array. However, there is no need to specify the size of the ArrayList like with an array as it grows automatically. |
| [SortedList](https://www.tutorialsteacher.com/csharp/csharp-sortedlist) | SortedList stores key and value pairs. It automatically arranges elements in ascending order of key by default. C# includes both, generic and non-generic SortedList collection. |
| [Stack](https://www.tutorialsteacher.com/csharp/csharp-stack) | Stack stores the values in LIFO style (Last In First Out). It provides a Push() method to add a value and Pop() & Peek() methods to retrieve values. C# includes both, generic and non-generic Stack. |
| [Queue](https://www.tutorialsteacher.com/csharp/csharp-queue) | Queue stores the values in FIFO style (First In First Out). It keeps the order in which the values were added. It provides an Enqueue() method to add values and a Dequeue() method to retrieve values from the collection. C# includes generic and non-generic Queue. |
| [Hashtable](https://www.tutorialsteacher.com/csharp/csharp-hashtable) | Hashtable stores key and value pairs. It retrieves the values by comparing the hash value of the keys. |
| BitArray | BitArray manages a compact array of bit values, which are represented as Booleans, where true indicates that the bit is on (1) and false indicates the bit is off (0). |

1. **Singleton Pattern**

The singleton design pattern is a creational design pattern. The purpose of the singleton design pattern is to ensure that a class only has one instance and provide a global point of access to it throughout the life of an application. Access of one instance is preferred to avoid unexpected results.

**Use Cases:** DB Connection & Logging

## Singleton Class Structure

A class should have the following structure for singleton pattern:

* Should have a private or protected constructor. No public and parameterized constructors.
* Should have a static property (with a private backing field) to return an instance of a class. A static method can also be used to return an instance.
* At least have one non-static public method for a singleton operation.

The following is the basic structure of the singleton class in C#.

Example: Singleton Class Structure

public class Singleton

{

private static Singleton \_instance;

private Singleton()

{

}

public static Singleton Instance

{

get

{

if (\_instance == null)

\_instance = new Singleton();

return \_instance;

}

}

public void DoSingletonOperation()

{

Console.WriteLine("singleton operation");

}

}

The above singleton class uses the static property to return an instance of the class. It has a private parameter less constructor which will restrict the creation of an object using the new keyword. You must use the Instance property to get its object. You can make the constructor protected if you want to allow it to be inherited in a subclass.

## Singleton Class using Static Constructor

You can create a singleton class by using the static constructor. The static constructor runs only once per app domain when any static member of a class is accessed.

## Singleton Class with Lazy Instantiation

If you use .NET 4 or higher, use [Lazy<T>](https://docs.microsoft.com/en-us/dotnet/api/system.lazy-1) to create an instance only when needed.

Example: Singleton Class with Lazy Instantiation

public sealed class VoteMachine

{

private static readonly Lazy<VoteMachine> \_instance = new L Lazy<VoteMachine>(() => new VoteMachine());

private int \_totalVotes = 0;

private VoteMachine()

{

}

public static VoteMachine Instance

{

get

{

return \_instance.Value;

}

}

public void RegisterVote()

{

\_totalVotes += 1;

Console.WriteLine("Registered Vote #" + \_totalVotes);

}

public int TotalVotes

{

get

{

return \_totalVotes;

}

}

}

The code above implicitly uses LazyThreadSafetyMode.ExecutionAndPublication as the thread safety mode for the Lazy<VoteMachine>>. The Lazy<T> make the lazy instantiation simple and performs well. It also allows you to check whether or not the instance has been created yet with the IsValueCreated property.

[**https://www.tutorialsteacher.com/csharp/singleton#:~:text=You%20can%20create%20a%20singleton,of%20a%20class%20is%20accessed.&text=The%20above%20VoteMachine%20is%20a%20singleton%20class%20with%20the%20static%20constructor**](https://www.tutorialsteacher.com/csharp/singleton#:~:text=You%20can%20create%20a%20singleton,of%20a%20class%20is%20accessed.&text=The%20above%20VoteMachine%20is%20a%20singleton%20class%20with%20the%20static%20constructor)**.**

public sealed static DBConnect()

{

**Private DBConnect() – Why Private is used**

{

}

}

Private parameter less constructor which will restrict the creation of an object using the new keyword.

What is singleton vs private constructor?

In singleton class, we use private constructor so that any target class could not instantiate our class directly by calling constructor, however, the object of our singleton class is provided to the target class by calling a static method in which the logic to provide only one object of singleton class is written.

What is lazy singleton?

One of the use cases for the Singleton pattern is lazy instantiation. For example, in the case of module imports, we may accidently create an object even when it's not needed. Lazy instantiation makes sure that the object gets created when it's actually needed.

Can we inherit singleton class in C#?

Why singleton class is always sealed in C#?

The sealed keyword means that the class cannot be inherited from. Declaring constructors private means that instances of the class cannot be created.

Why singleton class is always sealed in C#?

The sealed keyword means that the class cannot be inherited from. Declaring constructors’ private means that instances of the class cannot be created.

1. **What are tuples in C#?**

The word "tuple" in C# refers to a data structure which may consist of multiple parts. This data structure may or may not be a dataset with multiple values. Tuples were first introduced with . NET framework 4.0, and they allow a maximum of 8 elements. More than that will give you a compiler error.

Tuples are an ordered collection of data values, a tuple is an unordered collection of labeled data values.

Tuple<string> MyStringTuple = new Tuple<string>("Pluralsight");

10 Tuple<string,int> MyCustomTuple = new Tuple<string,int>("Daniel",28);

11 Tuple<int, int, int, int, int, int, int, Tuple<int>> MyMaxTuple = new Tuple<int, int, int, int, int, int, int, Tuple<int>>(1, 2, 3, 4, 5, 6, 7, new Tuple<int>(8));

Are tuples immutable?

Tuples and lists are the same in every way except two: tuples use parentheses instead of square brackets, and the items in tuples cannot be modified (but the items in lists can be modified). We often call lists mutable (meaning they can be changed) and tuples immutable (meaning they cannot be changed).

**LINQ**

1. **What is LINQ in C#?**

Language-Integrated Query (LINQ) is the integration of query capabilities directly into the C# language. Traditionally, queries against data are expressed as simple strings without type checking at compile time or IntelliSense support.

1. **What are the two types of LINQ?**

Query Syntax and Method Syntax

1. **What is IEnumerable in C#?**

IEnumerable in C# is an interface that defines one method, GetEnumerator which returns an IEnumerator interface. This allows readonly access to a collection then a collection that implements IEnumerable can be used with a for-each statement.

IEnumerable<Customer> customers = \_context.Customers.AsEnumerable().Where(c => c.IsActive);

1. **What is the use of IQueryable in C#?**

IQueryable<T> is a C# interface that lets you query different data sources. The type T specifies the type of the data source that you're querying. Under the hood, IQueryable uses expression trees that translate LINQ queries into the query language for the data provided.

IQueryable<Customer> customers = \_context.Customers.Where(c => c.IsActive);

1. **Difference between IEnumerable and IQueryable in C#?**

The main difference between IEnumerable and IQueryable in C# is that IQueryable queries out-of-memory data stores, while IEnumerable queries in-memory data. Moreover, IQueryable is part of .NET's System.LINQ namespace, while IEnumerable is in System.Collections namespace.

IQueryable contains methods for constructing expression trees. IQueryable inherits IEnumerable, so IQueryable does everything that IEnumerable does. IQueryable extends the IEnumerable with logic for querying data.

1. **What is the difference between IList and IEnumerable in C#?**

IEnumerable uses IEnumerator internally and is the recommended way to enumerate the contents of a collection. On the other hand, List uses IList, a non-generic interface, internally. IList allows for more direct manipulation of the contents of a list and is the recommended way to manipulate a list.

1. **What is the difference between list and IEnumerable?**

IEnumerable is a deferred execution while List is an immediate execution. IEnumerable will not execute the query until you enumerate over the data, whereas List will execute the query as soon as it's called. Deferred execution makes IEnumerable faster because it only gets the data when needed.

What is the difference between ToList and AsEnumerable?

The advantage of using AsEnumerable vs. ToList is that AsEnumerable does not execute the query. AsEnumerable preserves deferred execution and does not build an often useless intermediate list. On the other hand, when forced execution of a LINQ query is desired, ToList can be a way to do that.

1. **What is the difference between ObservableCollection and IEnumerable**

An IEnumerable is typically something you can enumerate - such as collections of items or generators. An ObservableCollection is a list that raises events when it's being modified. One is abstract and broad, the other concrete and specific.

1. **What is the difference between ICollection and IQueryable?**

ICollection is an actual collection of objects. when you iterate it (for each) it just walks the collection. IQueryable is an expression tree, that when iterated, executes the query (typically against a database) to get a collection of objects to return.

What is the difference between ICollection and list?

ICollection inherits IEnumerable. IList inherits the ICollection. And List is a class implementation of the IList.

1. **What is the difference between string array and IEnumerable in C#?**

Array is a reference type that provides methods to access elements by index and the property called Length that gives the number of elements in the array. IEnumerable is an interface that defines a single method, GetEnumerator, to go through all the items in a collection.

1. **Which is faster list or IEnumerable?**

IEnumerable is more efficient and faster when you only need to enumerate the data once. The List is more efficient when you need to enumerate the data multiple times because it already has all of it in memory.

1. **Which is faster IEnumerable or IQueryable?**

As IEnumerable performs a lot more work than IQuerytable, it is much slower. In the process of querying data from databases, IQueryable can be seen executing a select query on the server-side with the help of its filters. In comparison to IEnumerable, it does less work and therefore showcases faster performance.

1. **What is the difference between AsEnumerable and IEnumerable?**

The difference is that you can call AsEnumerable on an IEnumerable<T> where T is an anonymous type, and you couldn't perform such a cast on such a sequence, since you can't type out the name of a type that has no name. This ability of AsEnumerable to infer the generic argument is the reason for it to exist.

1. **What is the difference between ICollection and enum?**

An IEnumerable supports filtering elements using where clause. ICollection is another type of collection, which derives from IEnumerable and extends it's functionality to add, remove, update element in the list

1. **What is the difference between list select and ConvertAll in C#?**

The ConvertAll method runs through the list and creates a new list in one go, while the Select method uses lazy execution and only processes the items as you need them. If you don't need all the item, the Select method is more efficient.

1. **What does ToList () do?**

The tolist() function is used to convert a given array to an ordinary list with the same items, elements, or values.

Why ToList is used in C#?

The ToList<TSource>(IEnumerable<TSource>) method forces immediate query evaluation and returns a List<T> that contains the query results. You can append this method to your query in order to obtain a cached copy of the query results. ToArray has similar behavior but returns an array instead of a List<T>.

**Angular**

1. **What is Angular? Why was it introduced?**

[Angular was introduced](https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular) to create Single Page applications. This framework brings structure and consistency to web applications and provides excellent scalability and maintainability.

Angular is an open-source, JavaScript framework wholly written in [TypeScript.](https://www.simplilearn.com/tutorials/programming-tutorial/advanced-typescript) It uses HTML's syntax to express your application's components clearly.

1. **What is TypeScript?**

TypeScript is a superset of JavaScript that offers excellent consistency. It is highly recommended, as it provides some syntactic sugar and makes the code base more comfortable to understand and maintain. Ultimately, TypeScript code compiles down to [JavaScript](https://www.simplilearn.com/tutorials/javascript-tutorial/introduction-to-javascript) that can run efficiently in any environment.

1. Whats new in Angular 10?
2. How to call an API from angular application?
3. How to implement AuthGuard in Angular?
4. How to validate the user is authorized to access the API
5. Can you explain how JWT token implemented?
6. In Angular, how routing works?
7. Do you have an experience to deploy angular application? How you do that?
8. What are the data bindings using in Angular?
9. How to pass the data from child to child. Behavior subject

https://arulkumarsivaraj.blogspot.com/2021/07/angular-topics-to-learn.html

**SQL Server**

1. Explain Indexing
2. What is the difference between cluster index and non-cluster index?
3. When the cluster index will be created?
4. Why is cluster index being faster than non-cluster index?
5. **What does index seek and index scan do?**

Index scan means it retrieves all the rows from the table and index seek means it retrieves selective rows from the table.

1. What is transaction in SQL?
2. During the transaction I want to retrieve the data? only the committed transaction retrieves the query? When inserting the record and want to retrieve that data only that transaction?