1. Explain the key differences between Windows services and Web services in C#.

Windows Service

- Runs as a background process on Windows (no UI) and can start automatically at system boot or manually.

- Suited for long-running tasks (e.g., monitoring, scheduled background jobs, listeners).

- Installed and managed using service control manager (SCM). Runs under a Windows account (LocalSystem, NetworkService, custom).

- Lifecycle managed via SCM (OnStart, OnStop, OnPause, OnContinue).

- Typically built using System.ServiceProcess.ServiceBase.

- Not directly reachable over HTTP unless the service opens a network endpoint itself.

Web Service (SOAP / REST)

- A service accessible over network protocols (HTTP typically), meant to be consumed by clients (web apps, other services).

- SOAP Web Service (ASMX/WCF) uses XML envelopes and contracts. REST uses HTTP verbs and typically JSON.

- Runs inside a web host (IIS, Kestrel), not as a Windows service (though a service could host a web server).

- Stateless (usually) and designed for request/response interaction.

- Focus is interoperability, contracts (WSDL for SOAP), security over web protocols.

Summary (side-by-side)

- Purpose: background system tasks (Windows Service) vs. network API (Web Service).

- Hosting: SCM / OS service vs. Web host (IIS/Kestrel).

- Interaction model: event/continuous vs. request/response.

- Deployment/management: Windows Service Manager vs. web server / deployment pipeline.

---

2. Describe the Windows service and explain the purpose of OnStart and OnStop methods.

A Windows Service is an application that runs in the background as a Windows system service. It can start before any user logs in, run under specific accounts, and respond to SCM commands. It is useful for persistent background processing such as monitoring, job scheduling, or providing system-level functionality.

OnStart method

- Called by the Service Control Manager (SCM) when the service starts.

- Should contain code to initialize the service (start timers, spawn worker threads, initialize resources).

- Must return quickly; long operations should be offloaded to worker threads to avoid SCM timeouts.

OnStop method

- Called when the service is stopped by SCM.

- Should stop background activities, gracefully shutdown worker threads, release resources, and persist any required state.

- Also should return promptly after initiating shutdown.

Other lifecycle methods (brief)

- OnPause / OnContinue for pausing/resuming.

- OnShutdown for graceful shutdown at system shutdown.

---

3. Create a basic Windows service that logs the current date and time to a text file. Include the steps to install and start the service.

Below is a minimal Windows Service in C# (.NET Framework). This writes the current date/time to a text file every minute.

Service code (Service1.cs)

```csharp
using System;
using System.IO;
using System.ServiceProcess;
using System.Timers;

public class TimeLoggingService : ServiceBase
{
    private Timer _timer;
    private string _logFilePath;

    public TimeLoggingService()
    {
        ServiceName = "TimeLoggingService";
        _logFilePath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "TimeLog.txt");
    }

    protected override void OnStart(string[] args)
    {
        // Initialize timer to tick every 60 seconds
        _timer = new Timer(60 * 1000); // 60 seconds
        _timer.Elapsed += Timer_Elapsed;
        _timer.AutoReset = true;
        _timer.Start();

        // Log start
        Log("Service started at " + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"));
    }

    private void Timer_Elapsed(object sender, ElapsedEventArgs e)
    {
        Log("Current time: " + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"));
    }

    protected override void OnStop()
    {
        if (_timer != null)
        {
            _timer.Stop();
            _timer.Dispose();
            _timer = null;
        }
        Log("Service stopped at " + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"));
    }

    private void Log(string message)
    {
        try
        {
            Directory.CreateDirectory(Path.GetDirectoryName(_logFilePath));
            File.AppendAllText(_logFilePath, message + Environment.NewLine);
        }
        catch
        {
            // Avoid throwing from service; optionally log to EventLog
        }
    }

    // For debug as console app
    public void DebugStart() => OnStart(null);
    public void DebugStop() => OnStop();
}
```

Project notes

- Create a Windows Service project (or Class Library with a ServiceBase derived class) in Visual Studio targeting .NET Framework.

- Add an installer (ServiceProcessInstaller and ServiceInstaller) or install with sc create.

Installer class (ProjectInstaller.cs)

```csharp
using System.ComponentModel;
using System.ServiceProcess;

[RunInstaller(true)]
public class ProjectInstaller : System.Configuration.Install.Installer
{
    private ServiceProcessInstaller processInstaller;
    private ServiceInstaller serviceInstaller;

    public ProjectInstaller()
    {
        processInstaller = new ServiceProcessInstaller();
        processInstaller.Account = ServiceAccount.LocalSystem;

        serviceInstaller = new ServiceInstaller();
        serviceInstaller.ServiceName = "TimeLoggingService";
        serviceInstaller.DisplayName = "Time Logging Service";
        serviceInstaller.StartType = ServiceStartMode.Automatic;

        Installers.Add(processInstaller);
        Installers.Add(serviceInstaller);
    }
}
```

Build & Install (two common ways)

A — Using sc.exe (recommended for .exe already built)

1. Build the project — you'll have TimeLoggingService.exe in bin\Release.

2. Open an elevated command prompt (Run as Administrator).

3. Create service:

```
sc create TimeLoggingService binPath= "C:\Path\To\TimeLoggingService.exe" start= auto
```

Note: There must be a space after binPath= and start= in sc syntax.

4. Start the service:

```
sc start TimeLoggingService
```

5. Stop and delete:

```
sc stop TimeLoggingService
sc delete TimeLoggingService
```

B — Using InstallUtil.exe (older .NET Framework approach)

1. From an elevated Developer Command Prompt:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe "C:\Path\To\TimeLoggingService.exe"
```

2. Start service via Services MMC or net start TimeLoggingService.

3. To uninstall:

```
InstallUtil.exe /u "C:\Path\To\TimeLoggingService.exe"
```

Important

- For debugging locally, it's common to add a #if DEBUG console runner that calls DebugStart/DebugStop.

- Running as LocalSystem gives broad privileges; pick the least privilege account required.

---

4. Design a website to consume a SOAP-based Web service and display the result in the textbox. Use an example of currency conversion or any other method.

I'll show a simple **ASP.NET Web Forms** example that consumes a SOAP ASMX web service. (SOAP example is more straightforward with ASMX for teaching; WCF is another option.)

Assumption: There is a SOAP service with a method Convert (string fromCurrency, string toCurrency, decimal amount) that returns decimal.

Steps

1. Add a Service Reference to the SOAP service (in Visual Studio: Add Service Reference → enter WSDL URL).

2. In Default.aspx, put a TextBox to enter amount, dropdowns for currencies, a Button, and a TextBox for result.

Default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html>
<html>
<head runat="server"><title>Currency Converter</title></head>
<body>
    <form id="form1" runat="server">
        <asp:DropDownList ID="ddlFrom" runat="server" />
        <asp:DropDownList ID="ddlTo" runat="server" />
        <asp:TextBox ID="txtAmount" runat="server" />
        <asp:Button ID="btnConvert" runat="server" Text="Convert" OnClick="btnConvert_Click" />
        <asp:TextBox ID="txtResult" runat="server" ReadOnly="true" />
    </form>
</body>
</html>
```

Default.aspx.cs

```csharp
using System;
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            ddlFrom.Items.Add("USD");
            ddlFrom.Items.Add("EUR");
            ddlTo.Items.Add("USD");
            ddlTo.Items.Add("EUR");
        }
    }

    protected void btnConvert_Click(object sender, EventArgs e)
    {
        decimal amount;
        if (!decimal.TryParse(txtAmount.Text, out amount))
        {
            txtResult.Text = "Invalid amount.";
            return;
        }

        // Assume the service reference was added and named CurrencyServiceRef
        var client = new CurrencyServiceRef.CurrencyConverterSoapClient();
        try
        {
            decimal result = client.Convert(ddlFrom.SelectedValue, ddlTo.SelectedValue, amount);
            txtResult.Text = result.ToString("F2");
        }
        catch (Exception ex)
        {
            txtResult.Text = "Error: " + ex.Message;
        }
        finally
        {
            if (client != null && client.State == System.ServiceModel.CommunicationState.Opened)
                client.Close();
        }
    }
}
```

If using ASMX consumer (older style)

You'd add a web reference and Visual Studio generates a proxy class. Usage pattern is similar.

Note

- For modern projects, consider SOAP via WCF client proxies or use RESTful APIs (Web API) which are simpler to call with HttpClient.

- The example shows data posted to server side and response shown in txtResult.

---

5. What are attributes in C#? Explain their purpose and provide an example of a custom attribute.
Attributes are metadata attached to program entities (classes, methods, properties, assemblies, parameters, etc.). At runtime (or compile time) they can be read via reflection to influence behavior, provide hints to tools/compilers, or configure frameworks (e.g., serialization, ORM mapping, security).

Purpose

- Attach declarative information to program elements.
- Used by serializers, ORM frameworks, code analysis tools, compiler, runtime behaviors.
- Can provide additional info without modifying code logic.

Example of a custom attribute

```csharp
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, Inherited = true, AllowMultiple = false)]
public class DocumentationAttribute : Attribute
{
    public string Author { get; }
    public string Description { get; set; }

    public DocumentationAttribute(string author)
    {
        Author = author;
    }
}

// Usage:
[Documentation("Rahul Agrawal", Description = "Handles employee operations")]
public class EmployeeService
{
    // ...
}
```

You can later reflect on EmployeeService to inspect DocumentationAttribute.

---

6. Explain the use of the Obsolete attribute in C#. What is its impact during compilation?

[Obsolete] attribute marks program elements that are outdated or should not be used.

Forms

- [Obsolete] — default: causes a compiler warning when used.
- [Obsolete("message")] — includes a message explaining replacement or reason.
- [Obsolete("message", true)] — the true flag makes usage an error (compiler error) instead of a warning.

Example

```csharp
[Obsolete("Use NewMethod() instead", false)] // warning
public void OldMethod() { }

[Obsolete("Do not use this", true)] // error
public void VeryOldMethod() { }
```

Impact during compilation

- If error flag = false or omitted: compiler emits a warning when the member is referenced.
- If error flag = true: compiler emits an error and compilation fails if the member is used.

---

7. Write a program in C# to create a custom attribute Author with properties like Name and Version.

AuthorAttribute.cs

```csharp
using System;

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method | AttributeTargets.Assembly, AllowMultiple = true)]
public class AuthorAttribute : Attribute
{
    public string Name { get; }
    public string Version { get; set; }

    public AuthorAttribute(string name)
    {
        Name = name;
    }
}
```

Usage

```csharp
[Author("Rahul Agrawal", Version = "1.0")]
public class SampleClass
{
    [Author("Rahul Agrawal", Version = "1.1")]
    public void DoWork() { }
}
```

Reading via reflection

```csharp
using System;
using System.Reflection;

var attrs = (AuthorAttribute[])typeof(SampleClass).GetCustomAttributes(typeof(AuthorAttribute), true);
foreach (var a in attrs)
    Console.WriteLine($"Author: {a.Name}, Version: {a.Version}");
```

8. What is an assembly in C#? Explain the difference between private and shared assemblies.
An assembly is the compiled output (DLL or EXE) that contains IL code, metadata, and resources. It is the unit of deployment, versioning, and security in .NET. Assemblies contain types, resources, and a manifest.

Private assembly

- Placed in the application's folder.

- Used only by that application.

- No global registration needed.

Shared assembly

- Strong-named (signed with a key).

- Installed into the Global Assembly Cache (GAC) so multiple applications can share it.

- Versioning and GAC allow central management and reuse.

When to use each

- Private assembly: app-specific, simpler deployment.

- Shared assembly: common libraries used by multiple apps; requires careful version management.

---

9. Describe the structure of an assembly. Highlight the role of the manifest and metadata.

Typical parts of an assembly

- **Manifest**: describes the assembly identity (name, version, culture, public key token), lists files in the assembly, referenced assemblies, required permissions, and exported types. The manifest is the authoritative descriptor.

- **Metadata**: stored with IL; describes types, members, references, attributes — used by CLR for type loading and reflection.

- **Intermediate Language (IL)**: compiled MSIL code for types/methods.

- **Resources**: embedded resources (images, strings) and satellite assemblies for localization.

- **Native headers** / optional debugging information.

Role of manifest

- Acts like a table of contents and identity for the assembly.

- Ensures CLR can locate dependencies, enforce version policy, and security checks.

- Used by loader to bind and locate referenced assemblies.

Metadata

- Holds type signatures, method signatures, attributes, and other compile-time info that the CLR uses at runtime.

- Enables reflection, JIT compilation, and type safety.

---

10. Explain the steps to create a shared assembly and register it in the Global Assembly Cache (GAC).

1. Create the library

- Create Class Library project and implement types.

2. Strong-name sign the assembly

- Generate a key pair:

```
sn -k MyKeyFile.snk
```

- In Visual Studio: Project Properties → Signing → Select Strong name key file → choose MyKeyFile.snk.

3. Build the assembly (DLL)

---

4. Install into GAC

- Historically, developers used gacutil.exe:

```
gacutil -i MyLibrary.dll
```

Note: gacutil is part of Visual Studio SDK; not present on all machines.

- On modern Windows with .NET Framework, GAC is managed by the OS; gacutil is developer tool. For production, use an installer (MSI) that places assembly in the GAC via Windows Installer.

5. Verify

- gacutil -l MyLibrary or check C:\Windows\assembly (older shell) or C:\Windows\Microsoft.NET\assembly (newer layout).
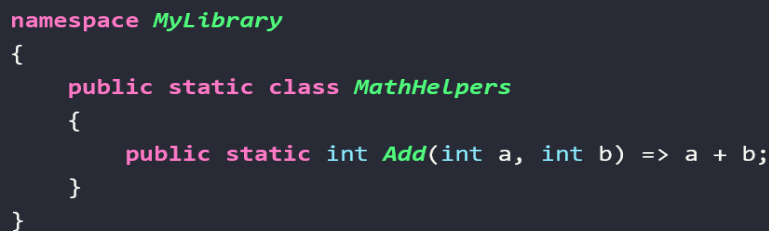
Important

- Assembly must be strong-named and versioned.

- For .NET Core / .NET 5+ there is no GAC; shared frameworks are handled differently.

---

11. Create a class library in C# and compile it as a DLL. Demonstrate its use in a console application.
MyLibrary/MathHelpers.cs

```csharp
namespace MyLibrary
{
    public static class MathHelpers
    {
        public static int Add(int a, int b) => a + b;
    }
}
```

Build produces MyLibrary.dll.

Console app that references the DLL

- In Visual Studio: Add Reference → Browse to MyLibrary.dll.

- Or with dotnet CLI:

    o   dotnet new classlib -o MyLibrary

    o   dotnet new console -o MyConsoleApp

    o   In console app: dotnet add reference ../MyLibrary/MyLibrary.csproj

    o   dotnet run in MyConsoleApp

Program.cs

```csharp
using System;
using MyLibrary;

class Program
{
    static void Main()
    {
        Console.WriteLine("2 + 3 = " + MathHelpers.Add(2,3));
    }
}
```

12. You are developing a desktop application to manage employee data. How would you use ADO.NET to fetch employee records from a SQL Server database and display them in a DataGridView?

Add a DataGridView to the form (e.g., dataGridView1).

1. Use SqlConnection, SqlDataAdapter, DataSet or DataTable to fetch data.

2. Bind the DataTable to DataGridView.

Sample code

```csharp
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

public void LoadEmployees()
{
    string connString = "Server=YOUR_SERVER;Database=YOUR_DB;Trusted_Connection=True;";
    string sql = "SELECT EmployeeID, Name, Department, Salary FROM Employees";

    using (SqlConnection conn = new SqlConnection(connString))
    using (SqlDataAdapter adapter = new SqlDataAdapter(sql, conn))
    {
        DataTable dt = new DataTable();
        adapter.Fill(dt);
        dataGridView1.DataSource = dt;
    }
}
```

Make columns editable and save changes

- If you want to allow editing and persist back to DB, use SqlDataAdapter with appropriate SelectCommand, InsertCommand, UpdateCommand, DeleteCommand or use SqlCommandBuilder.

```
SqlDataAdapter adapter = new SqlDataAdapter(selectCmdText, conn);
SqlCommandBuilder builder = new SqlCommandBuilder(adapter);


DataTable dt = new DataTable();
adapter.Fill(dt);
dataGridView1.DataSource = dt;


// When saving:
adapter.Update(dt);
```

Notes

- Configure AutoGenerateColumns or create typed columns.

- Use transactions or validations for data integrity on update.

- Use parameterized commands for the adapter to avoid SQL injection.

---

13. A client requires the application to work offline. Explain how you would use a DataSet and DataTable to enable this functionality in ADO.NET.

Idea

- DataSet/DataTable provide an in-memory, disconnected representation of data.

- You fetch data once from the DB into a DataSet and can work offline (view/edit/search) without a DB connection.

- Changes are tracked in the DataSet (RowState) and can be synchronized back with SqlDataAdapter.Update() when back online.

Example workflow

```
DataSet ds = new DataSet();
SqlDataAdapter adapter = new SqlDataAdapter("SELECT * FROM Employees", conn);
SqlCommandBuilder builder = new SqlCommandBuilder(adapter);
adapter.Fill(ds, "Employees");

// Work offline - manipulate ds.Tables["Employees"]

// When online again:
adapter.Update(ds, "Employees");
```

Benefits

- Allows disconnected editing and batching updates.

- Supports relations (multiple tables) and constraints in-memory.

- Useful for applications that require offline CRUD operations.

---

14. How would you implement parameterized queries in ADO.NET to prevent SQL injection when retrieving data from a table?

Bad (vulnerable) example

```
string sql = "SELECT * FROM Users WHERE Username = '" + username + "' AND Password = '" + password + "'";
```

Good (parameterized) example

```
using (SqlConnection conn = new SqlConnection(connString))
using (SqlCommand cmd = new SqlCommand("SELECT * FROM Users WHERE Username = @user AND Password = @pass", conn))
{
    cmd.Parameters.Add(new SqlParameter("@user", SqlDbType.NVarChar, 50) { Value = username });
    cmd.Parameters.Add(new SqlParameter("@pass", SqlDbType.NVarChar, 50) { Value = password });

    conn.Open();
    using (SqlDataReader reader = cmd.ExecuteReader())
    {
        // process results
    }
}
```

Notes

- Use parameterized commands for every value coming from user input.

- Prefer stored procedures with parameters for additional safety, but parameterized queries are sufficient to avoid SQL injection.

- Do not concatenate or interpolate user input into SQL strings.

---

15. You are tasked with developing an inventory management system that requires performing the following operations:

Requirements & approach

A. **Display a list of products with details in a DataGridView, allowing users to edit the data and save changes back to the database.**
   Use DataSet (or DataTable) with SqlDataAdapter:

   o Fill DataSet → bind to DataGridView.

   o User edits rows in the grid (in-memory).

   o Use SqlDataAdapter.Update(dataSet, "Products") to persist changes.

- Why DataSet?

   o Disconnected model supports editing, relationships, and change tracking (RowState).

   o Allows validation and batching updates.

Code (simplified):

```csharp
SqlDataAdapter adapter;
DataTable productTable;

void LoadProducts()
{
    string sql = "SELECT ProductId, Name, Quantity, Price FROM Products";
    adapter = new SqlDataAdapter(sql, conn);
    SqlCommandBuilder builder = new SqlCommandBuilder(adapter);

    productTable = new DataTable();
    adapter.Fill(productTable);
    dataGridView1.DataSource = productTable;
}

void SaveChanges()
{
    adapter.Update(productTable);
}
```

B. Retrieve the total number of products and their combined value from the database quickly, without loading all the product details into memory.

- Use a **single SQL aggregate query** executed on the server:

```sql
SELECT COUNT(*) AS TotalCount, SUM(Quantity * Price) AS TotalValue FROM Products;
```

- Retrieve results using ExecuteReader or ExecuteScalar:

  - ExecuteScalar works if you ask for a single scalar (e.g., count or sum individually).

  - For both values, use ExecuteReader or a SqlCommand with ExecuteReader and read two columns.

- Why not DataSet?

  - DataSet would require loading all rows into memory which is inefficient.

  - Aggregation on the DB server is far more efficient.

Code (example using ExecuteReader):

```csharp
string sql = "SELECT COUNT(*) AS TotalCount, SUM(Quantity * Price) AS TotalValue FROM Products";
using (SqlConnection conn = new SqlConnection(connString))
using (SqlCommand cmd = new SqlCommand(sql, conn))
{
    conn.Open();
    using (SqlDataReader reader = cmd.ExecuteReader())
    {
        if (reader.Read())
        {
            int totalCount = reader.IsDBNull(0) ? 0 : reader.GetInt32(0);
            decimal totalValue = reader.IsDBNull(1) ? 0m : reader.GetDecimal(1);
            // use values
        }
    }
}
```

C. Explain how you would utilize DataSet and DataReader in these scenarios, and justify why each is more suitable for its respective operation.DataSet / DataTable

- o Use when you need disconnected access, editing, binding to UI (DataGridView), relationships, and change-tracking.

- o Good for CRUD operations where the UI will modify data and later commit changes.

- **DataReader**

  - o Use when you need fast, forward-only, read-only access to rows (e.g., streaming large result sets, reporting).

  - o Lowest memory overhead and fastest for sequential reads.

- **Aggregate queries + ExecuteScalar/ExecuteReader**

  - o Use server-side aggregation to compute counts/sums quickly without transferring row data.

  - o Best for summary statistics.

Putting it together (inventory example)

- Display/edit/save: DataAdapter.Fill(DataTable) → bind → DataAdapter.Update.

- Get totals: SELECT COUNT(*), SUM(...) + ExecuteReader or ExecuteScalar.

Final tips & best practices

- Always use parameterized queries for DB operations.

- For services: design for proper shutdown and error handling; log important events to EventLog or files.

- Dispose ADO.NET objects (using) to avoid connection leaks.

- For production shared assemblies use signed keys and proper installers; for .NET Core, follow the framework packaging model (no GAC).

- When binding to DataGridView and allowing edits, handle concurrency (timestamps/rowversion columns) to avoid update conflicts.