UNIT-I: Introduction to .NET

1. Concept and Features of .NET

- **Definition**: .NET is a software framework developed by Microsoft to create, deploy, and run applications across platforms such as Windows, web, and mobile.
- **Features**:
  - **Cross-language interoperability**: Supports multiple programming languages.
  - **Unified programming model**: Provides a consistent development experience.
  - **Rich class libraries**: Predefined classes and methods for application development.
  - **Automatic memory management**: Includes garbage collection.
  - **High security**: Includes Code Access Security (CAS) and role-based security.
  - **Portability**: Runs on various platforms with .NET Core/.NET 5+.

2. Microsoft Intermediate Language (MSIL)

- **Definition**: MSIL is a low-level, CPU-independent set of instructions that .NET applications are compiled into before execution.
- **Features**:
  - Platform-independent execution.
  - Just-In-Time (JIT) compilation converts MSIL into native code.
- **Example**:

```
// Sample MSIL code
ldstr "Hello, World!"
call void [mscorlib]System.Console::WriteLine(string)
```

3. Metadata

- **Definition**: Metadata is data about the code, such as definitions and references, stored within assemblies.
- **Features**:
  - Enables reflection (inspection of code at runtime).
  - Facilitates cross-language interoperability.

- **Example**: Contains details like class name, methods, and properties.

4. .NET Namespaces

- **Definition**: Namespaces in .NET organize classes and other data types for easier management and avoidance of name conflicts.
- **Examples**:
  - `System`: Core functionalities.
  - `System.IO`: File handling.
  - `System.Net`: Networking functionalities.
- **Usage**:

```
using System;
using System.IO;
```

5. Common Language Runtime (CLR)

- **Definition**: The CLR is the execution environment of .NET applications, responsible for managing code execution.
- **Features**:
  - Memory management.
  - Thread management.
  - Exception handling.
  - Security enforcement.
- **Example**:
  - Converts MSIL to native code for the host machine.

6. Common Type System (CTS)

- **Definition**: CTS standardizes the data types used in .NET to ensure consistency across programming languages.
- **Examples**:
  - `System.Int32` (C# `int`) is the same as `Integer` in VB.NET.
- **Types**:
  - **Value types**: Stored directly in memory (e.g., `int`, `float`).

- **Reference types**: Hold references to memory locations (e.g., `string`, `object`).

7. Common Language Specification (CLS)

- **Definition**: CLS is a subset of CTS, specifying the basic rules and constructs for .NET languages to ensure interoperability.
- **Example**:
  - A CLS-compliant language does not allow multiple inheritance.

8. Overview of .NET Applications

- **Definition**: .NET applications are software programs developed using the .NET framework for various platforms like Windows, web, and mobile.
- **Types**:
  - **Windows Applications**: GUI-based apps using Windows Forms or WPF.
  - **Web Applications**: ASP.NET for dynamic web content.
  - **Mobile Applications**: Xamarin/MAUI for cross-platform mobile apps.

UNIT-II: Introduction to C# Programming with Respect to ASP.NET

1. Basics of ASP.NET

- **Definition**: ASP.NET is a server-side framework for building dynamic web pages and applications using .NET technologies. It enables developers to create rich, interactive, and data-driven web applications.
- **Features**:
  - **Server-side execution**: Code runs on the server, ensuring better security and performance.
  - **Event-driven model**: Simplifies handling of user actions like button clicks.
  - **State management**: Includes session, view state, cookies, and application state.
  - **Built-in caching**: Improves application performance by temporarily storing data.
- **Advantages**:

    - Enhanced scalability.
    - Rich set of controls for faster development.

- **Example**:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="WebApp.Default" %>
<html>
<body>
    <form runat="server">
        <asp:Label ID="Label1" runat="server" Text="Hello, ASP.NET!">
</asp:Label>
    </form>
</body>
</html>
```

2. Creating and Deploying ASP.NET Applications

- **Steps**:
    1. **Set up development environment**: Install Visual Studio and .NET SDK.
    2. **Create a new project**: Use Visual Studio to create an "ASP.NET Web Application" project.
    3. **Develop application**:
        - Design UI using Web Forms or Razor pages.
        - Write backend logic in C#.
    4. **Test application**: Run the application locally.
    5. **Deploy application**:
        - Use IIS for on-premise deployment.
        - Use Azure or AWS for cloud deployment.
- **Deployment Tools**:
    - **IIS (Internet Information Services)**: A web server for hosting ASP.NET applications.
    - **Azure App Service**: A cloud-based platform for hosting web applications.

3. Web Forms

- **Definition**: Web Forms is a part of ASP.NET that simplifies the creation of dynamic, data-driven web pages using event-driven programming.

- **Components**:

- **Pages**: `.aspx` files that define the structure and layout of the web page.
- **Code-behind files**: Contain the server-side logic in C# or VB.NET.

- **Features**:
  - Supports drag-and-drop controls in Visual Studio.
  - Automatically generates client-side HTML and JavaScript.

- **Example**:

```
<asp:Button ID="Button1" runat="server" Text="Submit"
OnClick="Button1_Click" />
```

4. Web Controls

- **Definition**: Reusable UI components in ASP.NET for creating web pages.
- **Types**:
  - **Standard Controls**: Basic controls like `TextBox`, `Label`, `Button`.
  - **Data Controls**: For displaying and managing data (e.g., `GridView`, `Repeater`).
  - **Navigation Controls**: For menus and site navigation (e.g., `Menu`, `TreeView`).
  - **Validation Controls**: To ensure proper user input (e.g., `RequiredFieldValidator`).
- **Example**:

```
<asp:TextBox ID="TextBox1" runat="server" />
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
    ControlToValidate="TextBox1" ErrorMessage="Field is required." />
```

5. Working with Events

- **Definition**: Events are user actions (like clicks or text changes) that trigger server-side logic in ASP.NET.
- **Common Events**:
  - `Click` for buttons.
  - `TextChanged` for text boxes.
- **Example**:

```
protected void Button1_Click(object sender, EventArgs e)
{
    Label1.Text = "Button was clicked!";
}
```

UNIT-III: Advanced ASP.NET Concepts

1. Rich Web Controls

- **Definition**: Advanced ASP.NET controls with enhanced functionality, interactivity, and styling, designed to create feature-rich web applications.
- **Examples and Features**:
  - **Calendar**:
    - Displays a graphical calendar.
    - Users can select dates for form inputs.
    - Example:

```
<asp:Calendar ID="Calendar1" runat="server" />
```

  - **AdRotator**:
    - Rotates ads based on an XML file or database.
    - Supports weighted rotation for better ad targeting.
    - Example:

```
<asp:AdRotator ID="AdRotator1" runat="server"
AdvertisementFile="Ads.xml" />
```

  - **FileUpload**:
    - Allows users to upload files to the server.
    - Example:

```
<asp:FileUpload ID="FileUpload1" runat="server" />
<asp:Button ID="Button1" runat="server" Text="Upload"
OnClick="UploadFile" />
```

2. Custom Web Controls

- **Definition**: Controls designed and created by developers for specific functionalities, often used to encapsulate reusable logic and UI.
- **Advantages**:
  - Reusable across multiple projects.
  - Encapsulation of complex functionality in a single component.
- **Steps to Create Custom Controls**:
  1. Derive the control from a base class, like `Control` or `WebControl`.
  2. Override the `Render` method to define how the control will appear on the client.
  3. Add properties and methods as needed.
- **Example**:

```
public class CustomLabel : Label
{
    public string Prefix { get; set; }
    protected override void Render(HtmlTextWriter writer)
    {
        writer.Write($"{Prefix}{Text}");
    }
}
```

3. Validation Controls

- **Definition**: Built-in controls to ensure that the data entered by the user is valid before processing.
- **Common Validation Controls**:
  - **RequiredFieldValidator**:
    - Ensures that a field is not empty.

- Example:

```
<asp:TextBox ID="TextBox1" runat="server" />
<asp:RequiredFieldValidator ID="Validator1" runat="server"
    ControlToValidate="TextBox1" ErrorMessage="This field is
required." />
```

- **CompareValidator**:
  - Compares the value of one control to another or a specific value.
  - Example:

```
<asp:TextBox ID="TextBox1" runat="server" />
<asp:CompareValidator ID="CompareValidator1" runat="server"
    ControlToValidate="TextBox1" ValueToCompare="100"
Operator="GreaterThan"
    ErrorMessage="Value must be greater than 100." />
```

- **RangeValidator**:
  - Ensures that input falls within a specific range.
- **RegularExpressionValidator**:
  - Validates input using a regular expression.
  - Example:

```
<asp:TextBox ID="EmailTextBox" runat="server" />
<asp:RegularExpressionValidator ID="Validator1" runat="server"
    ControlToValidate="EmailTextBox"
    ValidationExpression="\w+@\w+\.\w+" ErrorMessage="Invalid email
format." />
```

4. Debugging

- **Definition**: The process of identifying and resolving errors or bugs in an application.
- **Tools for Debugging**:
  - **Visual Studio Debugger**:
    - Set breakpoints to pause execution and inspect variable values.

        - Step through code line by line to find issues.
    - **Trace Logs**:
        - Use the `Trace` class for logging events.
        - Example:

```
Trace.WriteLine("Page Loaded");
```

- **Best Practices**:
    - Use meaningful variable names.
    - Keep breakpoints only during debugging.
    - Test edge cases thoroughly.

5. Deploying Projects with Business Objects

- **Definition**: Deployment of ASP.NET applications while using business objects to encapsulate the business logic.
- **Steps**:
    1. Design business objects with methods that handle business rules.
    2. Separate business logic from UI logic for better maintainability.
    3. Deploy the application using IIS or cloud platforms.
- **Example of Business Object**:

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    public decimal GetDiscountedPrice(decimal discountPercentage)
    {
        return Price - (Price * discountPercentage / 100);
    }
}
```

UNIT-IV: ADO.NET

1. Basics of ADO.NET

- **Definition**: ADO.NET is a data access technology in the .NET framework that facilitates communication between applications and data sources like SQL Server or Oracle.
- **Features**:
  - **Disconnected Data Access**: Works with in-memory data structures like `DataSet`.
  - **Integration with XML**: Supports operations on XML data.
  - **Optimized Providers**: Includes providers like SQL Server and OLEDB for efficient database access.

2. ADO.NET Objects

2.1 Data Table

- **Definition**: Represents a table of in-memory data with rows and columns.
- **Features**:
  - Contains schema information and data.
  - Can be used independently or within a `DataSet`.

2.2 Data View

- **Definition**: Provides a customizable and filterable view of a `DataTable`.
- **Usage**:
  - Sort or filter rows without modifying the underlying `DataTable`.

2.3 Data Set

- **Definition**: An in-memory representation of a collection of related tables and relationships.
- **Features**:
  - Supports disconnected operations.
  - Can work with multiple tables simultaneously.

2.4 Data Adapter

- **Definition**: Acts as a bridge between a database and a `DataSet`.
- **Usage**:

- Fills the `DataSet` with data and updates the database with changes.

3. OLEDB and SQL Managed Providers

3.1 OLEDB Provider

- **Definition**: A provider for accessing data from various sources like Microsoft Access, Excel, etc.
- **Example**:

```
using (OleDbConnection connection = new
OleDbConnection("Connection_String"))
{
    OleDbCommand command = new OleDbCommand("SELECT * FROM Table",
connection);
    connection.Open();
    OleDbDataReader reader = command.ExecuteReader();
    while (reader.Read())
    {
        Console.WriteLine(reader["ColumnName"]);
    }
}
```

3.2 SQL Managed Provider

- **Definition**: A provider specifically designed for SQL Server for high performance.
- **Example**:

```
using (SqlConnection connection = new SqlConnection("Connection_String"))
{
    SqlCommand command = new SqlCommand("SELECT * FROM Students",
connection);
    connection.Open();
    SqlDataReader reader = command.ExecuteReader();
    while (reader.Read())
    {
        Console.WriteLine(reader["Name"]);
    }
}
```

4. Data Operations Example

Fetching Data:

```
using (SqlConnection conn = new SqlConnection("Connection_String"))
{
    string query = "SELECT * FROM Employees";
    SqlDataAdapter adapter = new SqlDataAdapter(query, conn);
    DataSet ds = new DataSet();
    adapter.Fill(ds, "Employees");
    foreach (DataRow row in ds.Tables["Employees"].Rows)
    {
        Console.WriteLine(row["Name"]);
    }
}
```

Updating Data:

```
using (SqlConnection conn = new SqlConnection("Connection_String"))
{
    string query = "UPDATE Employees SET Name = 'John' WHERE Id = 1";
    SqlCommand cmd = new SqlCommand(query, conn);
    conn.Open();
    cmd.ExecuteNonQuery();
}
```