UNIT I

.NET Architecture

- **Common Language Runtime (CLR):** The CLR is the execution engine for .NET applications. It manages memory, thread execution, code safety, compilation, and other system services.
    - **Common Type System (CTS):** This standard ensures that data types defined in two different languages (e.g., C# and VB.NET) can interact with each other.
    - **Common Language Specification (CLS):** This is a set of rules that language designers must follow to ensure their language interoperates with other .NET languages.
    - **Execution Engine:** The component responsible for loading classes, managing memory layout, and executing the code instructions.
- **Framework Class Library (FCL):** A massive collection of reusable classes, interfaces, and value types that provide access to system functionality.
    - **Base Class Library (BCL):** The core subset of the FCL providing fundamental types available to all languages using the .NET Framework.
- **Compilation Process:**
    - **Source Code to MSIL:** When you compile code written in C# or VB.NET, it is not compiled into machine code immediately but into Microsoft Intermediate Language (MSIL), which is CPU-independent.
    - **JIT (Just-In-Time) Compiler:** At runtime, the JIT compiler translates the MSIL code into native machine code specific to the computer's processor architecture.
    - **Native Code:** This is the final machine-readable binary code executed by the processor after JIT compilation.

Assemblies

- **Structure of Assembly:** An assembly is the fundamental unit of deployment and versioning in .NET, typically a `.dll` or `.exe` file.
    - **Manifest:** Contains metadata describing the assembly itself, including its name, version, culture, and a list of all files in the assembly.
    - **Type Metadata:** Describes the data types defined in the assembly, allowing the CLR to locate and load classes.
    - **MSIL Code:** The intermediate language code generated from the source code.

    - **Resources:** Non-executable data such as images or strings embedded within the assembly.
- **Types of Assemblies:**
    - **Private Assemblies:** Intended for use by a single application and typically stored in the application's directory.
    - **Shared Assemblies (Global Assembly Cache - GAC):** Intended to be shared by multiple applications; they must be strongly named and installed in the GAC.
- **Versioning:**
    - **Major.Minor.Build.Revision:** The four-part version number used by the CLR to enforce version compatibility (e.g., 1.0.2.5).

Garbage Collection

- **Garbage Collection (GC):** An automatic memory management system that reclaims memory occupied by objects that are no longer in use, preventing memory leaks.
    - **Generations (Gen 0, Gen 1, Gen 2):** The heap is divided into three generations to optimize performance. New objects are Gen 0; if they survive a collection, they move to Gen 1, and eventually Gen 2, based on the premise that short-lived objects die quickly.
    - **GC Roots:** These are the starting points (like static variables or CPU registers) the GC uses to traverse the object graph and determine which objects are still reachable.
    - **Finalization Queue:** A mechanism for tracking objects that require cleanup (via a Finalizer) before their memory is released.

Resource Management:

- **Managed Resources:** Resources like memory and objects that are automatically handled by the Garbage Collector.
- **Unmanaged Resources:** System resources like file handles or database connections that the GC does not clean up automatically.

Transactions in .NET

- **ACID Properties:** The four key properties of a transaction: Atomicity, Consistency, Isolation, and Durability.
    - **Atomicity:** Atomicity guarantees that a series of operations are treated as a single unit; either all of them succeed, or none of them apply.
    - **Consistency:** Consistency ensures that a transaction brings the database from one valid state to another

    - **Isolation:** Isolation ensures that concurrent transactions do not interfere with one another.
    - **Durability:** Durability guarantees that once a transaction is committed, the changes are permanently saved

Structured Exception Handling

- **Keywords:**
    - **Try:** A block of code that is monitored for exceptions.
    - **Catch:** A block of code that executes if a specific exception occurs in the try block.
    - **Finally:** A block that executes regardless of whether an exception occurred, typically used for cleanup.
    - **Throw:** Used to manually raise an exception.

UNIT II

C# Language Basics

- **Data Types:** C# distinguishes between Value types (stored on the stack, e.g., int, bool) and Reference types (stored on the heap, e.g., strings, objects).
- **Variables and Constants:** Variables are storage locations with modifiable values, while constants are immutable values set at compile time.
- **Control Flow Statements:** Directives like `if-else`, `switch`, `for`, and `while` that dictate the order in which statements execute.
- **Arrays:**
    - **Single:** A straightforward list of elements.
    - **Multidimensional:** Rectangular arrays (e.g., matrices).
    - **Jagged:** An "array of arrays" where inner arrays can differ in length.

OOP Concepts in C#

- **Classes and Objects:**
    - **Constructors:** Special methods invoked when an object is created. They can be Static (called once per class), Private (prevent instantiation), or Public.
    - **Destructors:** Methods used to perform final cleanup before an object is reclaimed by garbage collection.
    - **this keyword:** A reference to the current instance of the class.
- **Methods:**
    - **Overloading:** Defining multiple methods with the same name but different parameter lists.
    - **Overriding:** Providing a specific implementation in a derived class for a method defined in the base class using `virtual`, `override`, and `new` keywords.
    - **Extension Methods:** A way to add new methods to existing types without modifying the original source code.
- **Properties:**
    - **Getters and Setters:** Accessors that define how private fields are read (get) or written (set).
    - **Auto-implemented properties:** Simplified syntax where the compiler automatically creates the private backing field.

Collections

- **Non-Generic Collections (System.Collections):** Older collections that store items as `Object` types, requiring casting.
    - **ArrayList:** A dynamically sized array.
    - **Hashtable:** A key-value pair collection using hashing.
    - **Stack:** Last-In, First-Out (LIFO) collection.
    - **Queue:** First-In, First-Out (FIFO) collection.
- **Generic Collections (System.Collections.Generic):** Type-safe collections that perform better because they avoid boxing/unboxing.
    - **List:** A strongly typed dynamic array.
    - **Dictionary<TKey, TValue>:** A strongly typed key-value collection.
    - **HashSet:** A collection of unique elements.

UNIT III

VB.NET Overview

- **Similarities/Differences with C#:** VB.NET and C# have nearly identical capabilities and compile to the same MSIL, differing primarily in syntax (e.g., braces { } vs. keywords like `End If`).

Object Oriented Features (VB Syntax)

- **Keywords:**
    - **Class, Module:** `Class` defines objects; `Module` acts like a static class for global methods.
    - **Inherits, Implements:** `Inherits` is used for class inheritance; `Implements` is used for interfaces.
    - **Overloads, Overrides:** `Overloads` creates multiple methods with the same name; `Overrides` replaces a base class method.
    - **MustInherit (Abstract), NotInheritable (Sealed):** VB.NET specific keywords for Abstract and Sealed classes respectively.

UNIT IV

ASP.NET

- **Architecture:**
    - **Page Life Cycle:** The sequence of events a page goes through: Initialization, Loading controls, Pre-Rendering, and Unloading (cleanup).
    - **Code-Behind vs. Inline Code:** Code-Behind keeps logic in a separate file (`.aspx.cs`), while Inline mixes logic in the `.aspx` file.
- **State Management:**
    - **Client-Side:** Stores data in the browser. Includes `ViewState` (page-level data), `Hidden Fields`, `Cookies` (small text files), and `Query Strings` (URL parameters).
    - **Server-Side:** Stores data on the server. Includes `Session State` (user-specific data) and `Application State` (global data for all users).

UNIT V

ADO.NET Architecture

- **Connected Architecture:** Relying on a continuous open connection (using `DataReader`). It is faster for reading data but holds onto database resources.
- **Disconnected Architecture:** Data is retrieved into a memory cache (`DataSet`), and the connection is closed. Changes are made locally and updated later. This is better for scalability.