1. Assembly - > single unit of deployment -> exe (executable)or dll (libraries) file
2. How to access assemble ->
3. if we add reference to the assembly then solution exp -> reference,
4. or in bin folder ProjectName/bin/Debug/ or ProjectName/bin/Release/
5. The Global Assembly Cache is a central repository for shared .NET assemblies. You can use tools like the **gacutil** command or navi gate to C:\Windows\Assembly to view the assemblies stored in the GAC
6. If you'd like to dynamically examine assemblies in your code, you can use the System.Reflection namespace
7. app configuration -> what will be there in app config file :
8. It is typically an XML file, such as App.config for Windows applications or Web.config for web applications.
9. The file is located in the root directory of the project, and when the application is built, it gets renamed to YourAppName.exe.config in the output directory

 <appSettings>: For key-value pairs.

 <connectionStrings>: For database-related configuration.

 <system.web>: For web application settings

1. How to access app config file: using configurationManager class
2. Reflection : accessing the assembly during runtime ->
3. Why string are immutable :
4. memory optimization
5. security
6. consistency
7. thread safety
8. enum : An **enum** (short for **enumeration**) in C# is a special data type that allows you to define a group of named constants
9. By default, the constants in an enum are assigned integer values starting from 0
10. Enums give meaningful names to a set of values, making your code more readable and type-safe.
11. You can explicitly assign values to the enum members.
12. Struct/ classes

| **Aspect** | **Structure** | **Class** |
| --- | --- | --- |
| **Type** | Value type | Reference type |
| **Memory Location** | Stack | Heap |
| **Inheritance** | Not supported | Supported |
| **Default Constructor** | No parameterless constructors | Can define custom ones |
| **Performance** | Faster for small objects | Slower for small objects |

1. Attributes: provide a way to add metadata to your code elements, such as classes, methods, properties, and assemblies. They offer a powerful mechanism to associate declarative information, which can later be used by the compiler, tools, or runtime for various purposes.
2. [Obsolete] - Marks a code element as obsolete or outdated.
3. [Serializable] -Indicates that a class can be serialized.
4. [DllImport] -Used for calling unmanaged code from a DLL (e.g., invoking native code).
5. Aspect Convert.ToString() .ToString()Null

Handling Returns "" for null Throws NullReferenceException Type Conversion Works on various types Depends on object's implementation

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | 1. **Aspect** | **Session State** | **Application State** | | --- | --- | --- | | **Scope** | Specific to each user session | Shared across all users | | **Lifetime** | Active only during the user's session | Persists for the lifetime of the app | | **Storage** | Server-side, per session | Server-side, globally | | **Use Case** | User-specific data (e.g., shopping cart) | Global data (e.g., application settings) | |

1. **ViewData:** A dictionary-type object that allows data transfer between the Controller and View. It is short-lived, meaning it only exists during the current request and view data requires explicit type casting.

**ViewBag:** Similar to ViewData but uses dynamic properties. It's easier to work with but also exists only during the current request.

**TempData:** Used to store data that needs to persist across multiple requests, often for redirection.

**Model Binding:** The most structured way to pass data, where the Controller sends a strongly typed model to the View.

1. **MVC Lifecycle Workflow**

**User Request**: The user sends an HTTP request to the application.

**Routing**: Maps the URL to the appropriate controller and action method.

**Controller**: Executes the logic in the specified action method.

**Action Method**: Interacts with the model (if needed) and prepares data for the view.

**View Engine**: Renders the view and creates the final HTML response.

**Response**: The generated HTML is sent back to the user's browser.

1. **Filter Order of Execution**: Filters execute in the following order:

**Authorization Filters** → **Action Filters** → **Result Filters** → **Exception Filters**.

Authentication Filters ->Run first.Verify if the user is authenticated.

Interface: IAuthenticationFilter

Authorization Filters->Check if the user has permission to access a resource.

Examples: [Authorize], [RequireHttps]

Interface: IAuthorizationFilter

Action Filters->Run before and after an action method executes.

Great for logging, input validation, or modifying parameters.

Interface: IActionFilter

Result Filters->Run before and after the result (like a view) is processed.

Useful for modifying the response or adding headers.

Interface: IResultFilter

Exception Filters->Handle unhandled exceptions thrown during action or result execution.

Example: [HandleError]

Interface: IExceptionFilter

1. **When to Use Static Methods?**

When the method does not depend on instance-level data or behavior.

To provide utility functions, such as string manipulation or calculations.

For global application functionality (e.g., logging frameworks, console.writeline. math).

To create shared state or behaviors across the application.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | 1. **Aspect** | **var** | **dynamic** | | --- | --- | --- | | **Type Checking** | Done at compile-time | Done at runtime | | **Type Safety** | Statically typed; type is fixed after assignment | Dynamically typed; type can change | | **Initialization** | Required at the time of declaration | Not required at declaration | | **Usage** | For strongly-typed variables with evident types | For working with unknown or changing types | | **Compile-Time Errors** | Detected if type mismatches occur | Errors occur only at runtime | | **Flexibility** | Less flexible, but safer | More flexible, but less safe |  1. **When to Use var vs dynamic**   Use var:   * + When you know the type of the variable at compile-time.   + For code that is safer, easier to debug, and with minimal runtime errors.   + Example: LINQ queries, anonymous types, or collections.   Use dynamic:   * + When the type is not known until runtime (e.g., JSON deserialization, Reflection, or COM Interop).   + When working with APIs that produce dynamic objects. |

1. How to handle error in applications :

Exception handling, try catch, custom error page, logging error, using throw to rethrow exception, exception filter

1. How to handle Garbage collector:

Short lived obj(0), Destructor, using keyword(DISPOSE method), IDisposable.

Aspect Variable Property

Access Directly accessible Accessed via get and set methods

Validation No validation or logic Allows validation and logic

Encapsulation Does not enforce encapsulation Promotes encapsulation

Visibility Often private Can be public, private, or proected

Usage Temporary storage or internal state Controlled access to fields

1. When to Use:

Variables: Use for internal, temporary storage or when validation isn't required.

Properties: Use when data needs to be exposed publicly or requires validation, computation, or logic during access.

1. Types of classes:
2. standard class : blue print of an obj
3. Abstract class : meant to serve as a base class for other call no instantiation, used in scenarios requiring polymorphism.
4. Sealed class : prevent inheritance -> It is often used when you want to prevent further extension of a class, ensuring that its functionality remains unchanged. This can be useful in scenarios where a class contains critical or secure logic, and you want to avoid accidental or intentional modification.-> Financial Transactions
5. Static class: only for static members no instantiation.eg: math,console
6. Partial class : organize code
7. Nested class: class defined within another class
8. Generic class: class that wrk with any data type, by using type parameter.
9. **What is Application Builder in .NET Core?**

IApplicationBuilder is an interface provided by ASP.NET Core that allows you to configure **middleware** components for processing incoming HTTP requests and outgoing responses in a sequential pipeline. Middleware is responsible for tasks such as routing, authentication, exception handling, and request/response modifications

| 1. **Data Structure** | **Key Features** | **Use Case** |
| --- | --- | --- |
| **Array** | Fixed size, fast access by index | Simple collections with a known size |
| **List<T>** | Dynamic size, indexed access | Collections with frequent additions/removals |
| **Dictionary<TKey, TValue>** | Fast key-based lookups | Key-value mapping |
| **Stack<T>** | LIFO structure | Undo/redo, function call tracking |
| **Queue<T>** | FIFO structure | Task scheduling, order-based processing |
| **HashSet<T>** | Unique elements | Fast membership testing |
| **LinkedList<T>** | Dynamic structure with nodes | Frequent insertions/removals at arbitrary positions |
| **SortedList<TKey, TValue>** | Key-value pairs sorted by key | Sorted lookups |

Best Practices

Choose arrays or List<T> for sequential data.

Use Dictionary<TKey, TValue> or HashSet<T> for key-based lookups or ensuring uniqueness.

Use Stack<T> or Queue<T> for LIFO/FIFO operations.

Optimize memory usage and performance by selecting the most appropriate structure for your use case.

1. different types of polymorphism, (compile time and run time) & real time examples:

Complie time real world exp : cacl program -> add 2 integerm double folat digits.  
run time -> animal sound.

1. What is View State?

View State is a client-side state management technique used in ASP.NET to preserve the values of page controls and variables during postbacks. ASP.NET is stateless, which means every request to the server is treated as a new request. View State solves this problem by retaining control values across postbacks without relying on server memory.

Initial State:

When the page is loaded for the first time, txtName is empty.

User Interaction:

The user enters "John" into the text box and clicks the "Submit" button.

Postback:

The value "John" is stored in the \_\_VIEWSTATE hidden field, sent to the server, and re-applied to the txtName control.

Even after a postback, the txtName control retains the value "John" because of View State.

1. Client side state management:

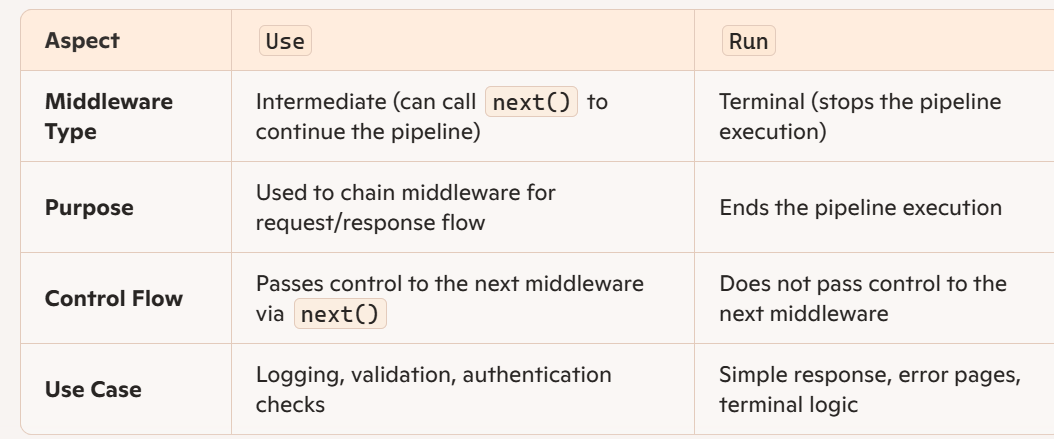
View state, control state, hidden fields,cookies, query string  
server side state management:

Application state, session state, cache, database

1. Configure method : Middleware components like UseRouting, UseEndpoints, and UseAuthentication are added in this method.

| **Middleware** | **What It Does** | **When It Runs** |
| --- | --- | --- |
| UseRouting() | Matches the URL to a route | Before authorization |
| UseEndpoints() | Executes the matched route's code | After authorization |

configure service method: uses methods like AddSingleton, AddScoped, or AddTransient for dependency registration.

1. 
2. **Attribute Routing: routeConfig.cs file**

Introduced in ASP.NET MVC 5, this allows routing to be defined directly on controller actions:

csharp

[Route("products/details/{id}")]

public ActionResult Details(int id)

{

// Logic here

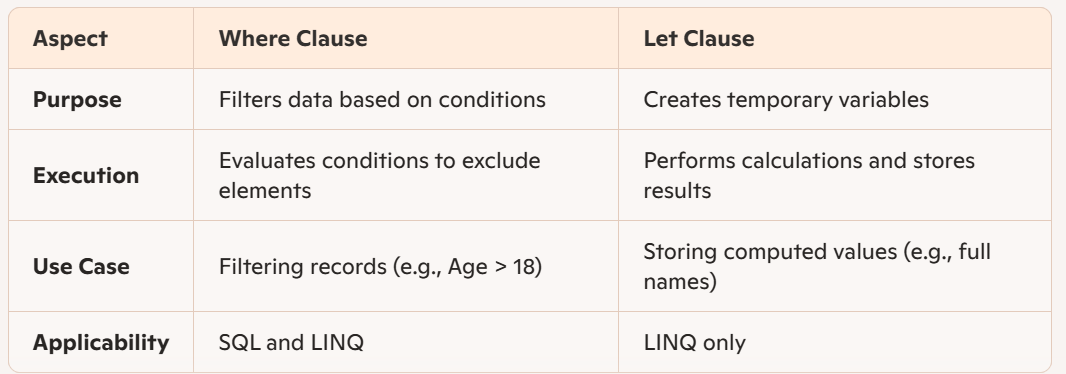
return View();

}

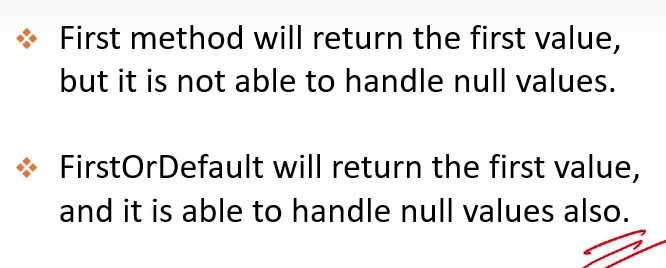
1. **WEB API ROUTING ::**

Routes are defined in the WebApiConfig.cs file:

Core web API : program.cs

1. 
2.  **LINQ** -> language intergreated query -> provides a declarative way to query data, making code easier to read and understand.

 **Lambda Expressions** ; is used to simplify the syntax of anonymous method -> means method without name

1. First method or first0rDefault  
   
2. An API (Application Programming Interface) serves as a bridge between different software systems, allowing them to communicate and interact. APIs are essential in modern software development, as they enable developers to build scalable, modular, and integrated applications
3. A screenshot of a computer

   AI-generated content may be incorrect.
4. **1xx (Informational)**

100 Continue: Indicates the client should continue with its request.

101 Switching Protocols: Server is switching protocols as requested by the client.

**2xx (Success)**

200 OK: The request was successful, and the server returned the requested data.

201 Created: A new resource was successfully created.

204 No Content: The request was successful, but there is no content to return.

**3xx (Redirection)**

301 Moved Permanently: The resource has been permanently moved to a new URL.

302 Found: The resource has been temporarily moved to a different URL.

304 Not Modified: Indicates that the resource has not changed, so the client can use its cached version.

**4xx (Client Errors)**

400 Bad Request: The server could not understand the request due to invalid syntax.

401 Unauthorized: Authentication is required to access the resource.

403 Forbidden: The client does not have permission to access the resource.

404 Not Found: The requested resource could not be found.

405 Method Not Allowed: The HTTP method used is not supported by the server for the requested resource.

**5xx (Server Errors)**

500 Internal Server Error: The server encountered an unexpected condition.

502 Bad Gateway: The server received an invalid response from an upstream server.

503 Service Unavailable: The server is temporarily unable to handle the request (e.g., maintenance or overload).

504 Gateway Timeout: The server did not receive a timely response from an upstream server.

1. **Dynamic Insertions**: Use Add() to insert elements into collections such as List, Dictionary, Queue, etc.

**Custom Logic**: Define and call Add() in your custom classes when specific logic is needed for adding items.

**Real-Time Data Handling**: Use it in scenarios where items are added dynamically during execution.

1. A screenshot of a computer

   AI-generated content may be incorrect.
2. Versioning:

API versioning, software versioning, document versioning.

1. A CTE (Common Table Expression) is a temporary, named result set in SQL that simplifies complex queries by breaking them into manageable parts.
2. A screenshot of a computer

   AI-generated content may be incorrect.
3. IWebHost : is an interface that powers the ASP.NET Core application's hosting and web server integration. It ensures the application is properly configured, initialized, and executed. For modern applications, transitioning to the Generic Host model (IHost) is recommended for greater flexibility
4.  **Use** ==:

When comparing value types for equality (e.g., integers, floats).

When checking if two references point to the same instance.

1. **Use** .Equals():

 By default:

* **Reference types**: Compares whether the objects refer to the **same memory location** (similar to == for objects).
* **Value types**: Compares the **actual values**.

 When overridden:

* **Reference types**: Can compare object data (instead of memory references).
* Example:
* When comparing the contents of objects for equality.

In custom classes, override .Equals() to implement comparison logic specific to the class.

1. A screenshot of a computer

   AI-generated content may be incorrect.
2. lllA screenshot of a computer

   AI-generated content may be incorrect.
3. A screenshot of a computer

   AI-generated content may be incorrect.
4. **Use JOINS** when retrieving data from multiple related tables for efficiency. ✔ **Use subqueries** only when absolutely necessary to fetch independent values. ✔ **Use Stored Procedures** for reusable business logic, not just SELECT statements
5. **When to use Async/await and parallel**

If your task involves waiting for external resources (like API calls or file operations), use async/await. If your task involves performing multiple calculations in parallel, use Parallel.ForEach or Task.WhenAll.

1. A screenshot of a computer

   AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Web API action results:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Web API life cycle

1.Routing

The incoming HTTP request is matched to a controller and action method based on routing rules.

Routes are defined in Program.cs or Startup.cs.

2.Controller Selection

The routing system selects the appropriate API controller based on the request URL.

Example: /api/products maps to ProductsController.

3.Model Binding

Converts HTTP request data (JSON, XML, Form Data) into .NET objects.

4.Action Execution

The selected controller method is executed.

Business logic, database operations, or external API calls take place.

5.Result Formatting

The action result is converted into a response format (JSON, XML).

6.Response Sent to Client

The final formatted response is sent to the requester.

Status codes like 200 OK, 404 Not Found, 500 Internal Server Error are included

1. A screenshot of a computer program

   AI-generated content may be incorrect.
2. MVC -> bin/debug or release folder -> solution exp = references

MVC Core -> bin/debug or release folder -> solution exp = dependencies

1. Routing in ASP.NET MVC (.NET Framework)

Routing in ASP.NET MVC is configured in the RouteConfig.cs file inside App\_Start.

Routing in ASP.NET Core MVC (.NET Core)

In ASP.NET Core MVC, routing is defined in Program.cs or Startup.cs.

Attribute Routing (Both MVC and MVC Core)

Instead of defining routes globally, you can specify routes directly on controllers using [Route] attributes.

1. Configure, confiruration service => MVC core in startup.cs file in MVC we don’t have such things.
2. Transaction in sql:

A transaction in SQL is a sequence of one or more operations (like queries or updates) that are executed as a single unit. The goal of a transaction is to ensure data integrity and consistency by following the ACID properties: Atomicity, Consistency, Isolation, and Durability.

1. Stored procedure:

A Stored Procedure is a precompiled collection of SQL statements and optional control-of-flow logic stored in a database. Instead of writing the same query repeatedly, you can execute the stored procedure to perform operations like data manipulation, validation, and complex calculations.

1. Triggers in sql:

A trigger in SQL is a special kind of stored procedure that automatically executes when a specified event occurs on a table or view. Triggers are often used to maintain the integrity of the database, enforce business rules, or automatically perform auditing and logging tasks.

Types of Triggers

DML Triggers (Data Manipulation Language) => Triggered by INSERT, UPDATE, or DELETE statements. Example: Automatically log changes to a table after an update.

DDL Triggers (Data Definition Language) =>Triggered by CREATE, ALTER, or DROP operations. Example: Track schema changes or prevent dropping a table.

Logon Triggers => Triggered by user logins to the database. Example: Restrict users from logging in during specific hours.

1. Store the value temporary in SQL :

**Temporary tables** offer session-level persistence for data. ✅ **CTEs and derived tables** are lightweight and optimized for queries. ✅ **Table variables** provide high-performance batch or procedure-level storage.

1. A correlated subquery is a type of subquery where the inner query depends on values from the outer query for its execution. Unlike a regular subquery, which is executed once, a correlated subquery is executed repeatedly, once for each row processed by the outer query.
2. Types of Functions in SQL

System Functions:

Predefined functions provided by SQL databases.Includes:

String Functions: LEN(), SUBSTRING(), CHARINDEX().

Aggregate Functions: SUM(), AVG(), MAX(), COUNT().

Date Functions: GETDATE(), DATEADD(), DATEDIFF().

User-Defined Functions (UDFs):Custom functions created by users.Can be:

Scalar Functions: Returns a single value.

Table-Valued Functions: Returns a table.

Inline Table-Valued Functions: A simpler version of a table-valued function.

1. Parse and Try Parse:

If the number is in string format you have 2 options Parse and try Parse

Prase : will throw an exception error if it cannot parse the value where as try parse will true an Boolean value.

1. Constructor:

It is a special method that is used to initialized an objects.

Advantage: it is called when an object is created. It can be used to set initial values for fields.

1. Data Definition Language (DDL)

Create, update, drop, truncate

Data Manipulation Language (DML)

Insert, update, delete

Data Query Language (DQL)

Select

Data Control Language (DCL)

Grant, revoke

Transaction Control Language (TCL)

Commit, rollback, savepoint.

1. **(Constant)**

**Value is set at compile time** and **cannot change** afterward.

Must be initialized **when declared**.

**Implicitly static**—shared across all instances.

Can only be **primitive types**, string, or null.

**readonly**

Value is set at **runtime**, either at declaration or in the **constructor**.

Can be used with **reference types** and **complex objects**.

Can be **instance-level** or **static**.

Useful when the value is known only at runtime.

1. **when to Use a Cursor**

Performing **row-by-row operations** that can't be done with a single SQL statement

Applying **conditional logic** to each row

Working with **hierarchical or recursive data**

Performing **complex transformations** or **custom updates**

1. strongly typed view ?

A strongly typed view in ASP.NET MVC is a Razor view that’s tightly bound to a specific model class. This means the view knows exactly what type of data it’s working with, giving you IntelliSense, compile-time checking, and cleaner, more maintainable code.

1. **JWT (JSON Web Token) Authentication**

JWT is a **stateless authentication** method where the server issues a digitally signed token after verifying a user's credentials.

1. **What is a Delegate?**

A **delegate** is like a *type-safe function pointer*. It holds a reference to a method with a specific signature and return type.

**What is an Event?**

An **event** is a **wrapper around a delegate** that follows the *publisher-subscriber* pattern. It lets one part of your app notify others when something happens—like a button click or a file download completing.

1. User defined data types:

Struct, union, enum, typedef

1. Static, this, base, var, async, virtual, override, new
2. Static files in core located in wwwroot we need to serve it like (useStaticFiles => program.cs or startup.cs file)

In ASP .net > wwwroot it will automatically server

1. Dependency injection : instead of directy implementing the class we can create by an constructor, addscope -> for per HTTP request, addtransiant-> each and every instance, singleton-> only once
2. ONE TO MANY replationship - > in model we need to give the value in collection and add it to dbmodel.
3. The **Repository Pattern** is a design pattern used in C# (and other languages) to **abstract the data access layer** from the business logic layer.
4. SOLID principle:

=>Single Responsibility Principle (SRP)

A class should have only one reason to change.

=>Open/Closed Principle (OCP)

Software should be open for extension, but closed for modification.

=>Liskov Substitution Principle (LSP)

Subtypes must be substitutable for their base types.

If class B is a subclass of class A, it should behave like A without breaking anything.

=>Interface Segregation Principle (ISP)

Clients should not be forced to depend on methods they do not use.

=>Dependency Inversion Principle (DIP)

High-level modules should not depend on low-level modules. Both should depend on abstractions.

1. Guideline for RESTful and rest API  
   1.stateless -> no session state and application state.

2.seperation of client and server.

3. uniform interface

4. cacheable

5. layered system -> like MVC

1. Design Pattern :**Singleton**: Ensures a class has only one instance.

**Factory Method**: Creates objects without specifying the exact class.

| 1. **Type** | **Purpose** | **Common Patterns** |
| --- | --- | --- |
| **Creational** | Object creation | Singleton, Factory, Builder, Prototype |
| **Structural** | Object composition | Adapter, Decorator, Facade, Composite |
| **Behavioral** | Object interaction | Observer, Strategy, Command, State |
| 1. **public class Calculator**   **{**  **// Static method**  **public static int Add(int a, int b)**  **{**  **return a + b;**  **}**  **// Non-static (instance) method**  **public int Add(int a, int b, int c)**  **{**  **return a + b + c;**  **}**  **}** |  |  |

class Program

{

static void Main()

{

// Call static method

int sum1 = Calculator.Add(2, 3); // Output: 5

// Call instance method

Calculator calc = new Calculator();

int sum2 = calc.Add(2, 3, 4); // Output: 9

Console.WriteLine($"Static Add: {sum1}");

Console.WriteLine($"Instance Add: {sum2}");

}

}  
if the method signature is different we can use with without creating obj. if the signature is same we will get compie time error.

1. Nunit : create the main project -> create the nunit project testing by seect create, new, Nunitproject -> add reference -> write the testcase-> run

| **Attribute** | **Purpose** | **Runs When?** |
| --- | --- | --- |
| [TestFixture] | Marks a class as a test container | Once per test class |
| [SetUp] | Initializes test context | Before each test method |
| [Test] | Marks a method as a test | For each test run |
| [TestCase] | Runs a test method with parameters | Once per test case |

1. void Method(int a, int b)

int Method(int a, int b)  
this will have a compile time error like we can’t have same parameter and method name with different return type.  
valid method overloading :

void Method(int a) { }

int Method(int a, int b) { return a + b; }

double Method(double a, double b) { return a + b; }

1. routing types in .net:

| **Routing Type** | | **Where Defined** | | | **Use Case** | |
| --- | --- | --- | --- | --- | --- | --- |
| Convention-Based | | Startup.cs / Program.cs | | | MVC apps with standard patterns | |
| Attribute-Based | | On controllers/actions | | | APIs, custom route control | |
| Endpoint Routing | | Middleware pipeline | | | Minimal APIs, Razor Pages | |
| Custom Routing | | Custom logic/middleware | | | Advanced scenarios | |
| 1. **Feature** | **DELETE** | | **TRUNCATE** | **DROP** | |
| Type | DML | | DDL | DDL | |
| Removes | Specific rows | | All rows | Entire table | |
| Keeps table structure | ✅ Yes | | ✅ Yes | ❌ No | |
| WHERE clause | ✅ Yes | | ❌ No | ❌ No | |
| Rollback possible | ✅ Yes | | ❌ No (mostly) | ❌ No | |
| Fires triggers | ✅ Yes | | ❌ No | ❌ No | |
| Speed | Slow | | Faster | Fastest | |

| 1. **Feature** | | **Purpose** | **Example** | |
| --- | --- | --- | --- | --- |
| RouterModule | | Registers and configures routes | RouterModule.forRoot(routes) | |
| routerLink | | Binds a URL to a clickable element | <a routerLink="/home">Home</a> | |
| <router-outlet> | | Placeholder for routed components | <router-outlet></router-outlet> | |
| Lazy Loading | Loads modules only when needed | | |
| Child Routes | Organizes nested views cleanly | | |
| Route Guards | Protects routes from unauthorized access | | |
|  |  | | |

1. **Two-way data binding** in Angular allows synchronization between the **component class** and the **template view**. When the user updates the input field, the component property updates automatically—and vice versa.
2. **When 2 components are there in angular how to invoke:**

| **Relationship** | **Technique** |
| --- | --- |
| Parent → Child | @Input() |
| Child → Parent | @Output() + EventEmitter |
| Sibling ↔ Sibling | Shared service with RxJS |
| Direct Method Call | @ViewChild() |

1. Linq query :  
   var result = from student in students

join hobby in hobbies

on student.Id equals hobby.StudentId into studentHobbies

select new

{

StudentName = student.Name,

HobbyCount = studentHobbies.Count()

};  
-------------------------------------------------------------------------------------------------------

1.Select Specific Fields From a list of products, select only the product names and prices.

var result = products.Select(p => new

{

p.Name,

p.Price

});

2.Sort by Name Sort a list of students alphabetically by their first name.

var result = from stu in students

orderby stu.FirstName descending

select stu;

3.Count Items Count how many orders have a total amount greater than $100.

var count = (from odr in orders

where odr.Amount > 100

select odr).Count();

4.Group by Category Group a list of books by their genre and count how many books are in each genre.

var result = from bk in books

group bk by bk.Genre into genreGroup

select new

{

Genre = genreGroup.Key,

BookCount = genreGroup.Count()

};

5.Find Maximum Value Find the highest salary among all employees.

var maxSalary = employees.Max(emp => emp.Salary);

6.Join Two Lists Join a list of customers with their orders and display customer name with order total.  
var result = from cust in customers

join ord in orders on cust.CustomerId equals ord.CustomerId

select new

{

CustomerName = cust.Name,

OrderTotal = ord.Total

};

7.Check for Existence Check if any product in the list is out of stock (quantity = 0).

var result = from prod in products

where prod.Quantity == 0

select prod;

bool anyOutOfStock = result.Any();

8.Get First Match Find the first student whose grade is "A".

var student = students.FirstOrDefault(s => s.Grade == "A");

9.Distinct Values Get a list of unique cities from a list of customer addresses.

var result = customers.Select(s => s.City).Distinct();

Programs :

1. In SQL Server, you can concatenate the Firstname and Lastname into a FullName using the + operator or the CONCAT function

SELECT Firstname + ' ' + Lastname AS FullName

FROM Employees;

--------------------

SELECT CONCAT(Firstname, ' ', Lastname) AS FullName

FROM Employees;

1. SELECT

CASE

WHEN Gender = 'Male' THEN 'Mr. ' + Firstname + ' ' + Lastname

WHEN Gender = 'Female' THEN 'Mrs. ' + Firstname + ' ' + Lastname

ELSE Firstname + ' ' + Lastname

END AS FullName

FROM Employees;

1. select unique records where rank is greater than 3 and group by course id

SELECT CourseID, COUNT(\*) AS RecordCount

FROM Students

WHERE Rank > 3

GROUP BY CourseID;

1. example with having and group by

SELECT CourseID, COUNT(StudentID) AS TotalStudents

FROM Enrollments

WHERE EnrollmentDate >= '2023-01-01' -- Filter rows before grouping

GROUP BY CourseID

HAVING COUNT(StudentID) > 10; -- Filter grouped results

select \* FROM( SELECT enpname, salary, DENSE\_RANK() OVER(ORDER BY salary desc) as r from employee)

as k where r =2;

1. select top 1 salary from (select distinct top 5 salary from employee order by salary desc)

result order by salary;

1. select Max(salary) from employee

where salary <( select max(salary) from employee);

string input = " hi team"

string[] word = input.Split(' ');

string reverse = " ";

for(int i= word.Length-1; i>=0;i--)

{

reverse += word[i] + ' ';

}

Console.WriteLine(reverse.Substring(0, reverse.Length-1));

Console.ReadKey();

output : team hi

--------------------------------------------------------------------------------------------------

string input = " bhavani is one team";

string reverse = " ";

for (int i = input.Length - 1; i >= 0; i--)

{

reverse += input[i];

}

Console.WriteLine(reverse);

Console.ReadKey();

output: maet eno si inavahb

-----------------------------------------------------------------------------------------------------

string input = "bhavani is perparing";

string reverse = string.Empty;

string[] everyword = input.Split(' ');

for (int i = 0; i <= everyword.Length - 1; i++)

{

string workword = everyword[i];

for (int j = workword.Length - 1; j >= 0; j--)

{

reverse += workword[j];

}

reverse += ' ';

}

Console.WriteLine(reverse);

Console.ReadKey();

output inavahb si gniraprep

-------------------------------------------------------------------------------------------------------

string input = "madam is not ";

string reverse = string.Empty;

for (int i =input.Length-1;i>=0;i--)

{

reverse += input[i];

}

if (reverse == input)

{

Console.WriteLine("plain");

}

else{

Console.WriteLine("not plain");

}

Console.ReadKey();

Input : abc // output ; a ab abc b bc c

Console.WriteLine("enter the string");

string input = Console.ReadLine();

for(int i=0;i< input.Length; i++)

{

StringBuilder stbuilt = new StringBuilder();

for (int j= i; j< input.Length; j++)

{

stbuilt.Append(input[j]);

Console.WriteLine(stbuilt+ " ");

}

}

Console.ReadKey();

Occurrence of a char in given stringl:

using System;

using System.Collections.Generic;

using System.Text;

namespace ConsoleApp19

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Enter the senyence");

string message = Console.ReadLine();

// int count = 0;

Dictionary<char, int> dic = new Dictionary<char, int>();

foreach(char ch in message)

{

if (ch == ' ') // Skip spaces

continue;

if (dic.ContainsKey(ch))

{

dic[ch] = dic[ch] + 1;

}

else

{

dic.Add(ch, 1);

}

}

foreach(char item in dic.Keys)

{

Console.WriteLine(item + ":" + dic[item]);

}

Console.ReadKey();

}

}

}

-A/--------------------------------------------------------------------------

Remove duplicate words:

string input = "this is is bad one for one the one";

HashSet<string> seenword = new HashSet<string>();

string[] words = input.Split(" ");

string result = "";

foreach(string word in words)

{

if(!seenword.Contains(word.ToLower()))

{

seenword.Add(word.ToLower());

result += word + " ";

}

}

Console.WriteLine(result);

Console.ReadKey();

Occurrence of a char:

Console.WriteLine("enter");

string message = Console.ReadLine();

char keywrd = 'a';

int count = 0;

foreach(char ch in message)

{

if (ch == 'a')

{

count++;

}

}

Console.WriteLine(count);

Console.ReadKey();

Input: at that space at.

Output: 4

Reverse a string and case change

Console.WriteLine("enter");

string message = Console.ReadLine();

string rev = "";

for (int i=message.Length-1;i>=0;i--)

{

char ch = message[i];

// Swap case using built-in functions

rev += char.IsUpper(ch) ? char.ToLower(ch) : char.ToUpper(ch);

}

Console.WriteLine(rev);

Console.ReadKey();

Input: ObnIaR

Output: rAiNBo

Occurrence of a char followed by ‘a’

Console.WriteLine("Enter a string:");

string input = Console.ReadLine();

Console.WriteLine("Enter the character to search:");

char target = Console.ReadLine()[0];

int count = 0;

for (int i = 0; i < input.Length - 1; i++)

{

if (input[i] == target && input[i + 1] == 'a')

{

count++;

}

}

Console.WriteLine($"Occurrences of '{target}' followed by 'a': {count}");

Console.ReadKey();

occ

input: bhavani vani

output : 2

DS – bubble sort :  
-------------------------  
 class Program

{

static void Main(string[] args)

{

// op 2,3,4,5,7;

int[] arr = { 5, 4, 7, 3, 2 };

for(int i =0; i<arr.Length-1; i++)

{

for(int j=0;j< arr.Length-i-1; j++)

{

if(arr[j] > arr[j+1])

{

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

Console.WriteLine("Sorted Array: " + string.Join(", ", arr));

//Console.WriteLine("array" + arr);  
 Console.ReadKey();

Binary search:

static void Main(string[] args)

{

// op 2,3,4,5,7;

int[] arr = { 5, 4, 7, 3, 2 ,8};

for (int i = 0; i < arr.Length - 1; i++)

{

for (int j = 0; j < arr.Length - i - 1; j++)

{

if (arr[j] > arr[j + 1])

{

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

Console.WriteLine("Sorted Array: " + string.Join(", ", arr));

int left = 0, right = arr.Length - 1;

bool found = false;

int target = 4;

while (left <= right)

{

int mid = (left + right) / 2;

if (arr[mid] == target)

{

Console.WriteLine($"Target {target} found at index {mid}");

found = true;

break;

}

else if (arr[mid] < target)

left = mid + 1;

else

right = mid - 1;

}

if (!found)

Console.WriteLine($"Target {target} not found in array.");

Console.ReadKey();

------------------------------------------------------------------------------------------------------------

Fact

static void Main(string[] args)

{

Console.WriteLine(Factorial(5));

int Factorial(int n)

{

if (n == 1 || n == 0)

return 1;

return (n \* Factorial(n - 1));

}

Console.ReadKey();

Febo:

static void Main(string[] args)

{

int n = 8, a = 0, b = 1;

Console.WriteLine($"{a}, {b}");

for(int i=0;i<=n;i++)

{

int c = a + b;

Console.Write(c + " ");

a = b;

b = c;

}

Console.ReadKey();

Prime:

int n = 17;

bool isPrime = n > 1;

for (int i = 2; i < Math.Sqrt(n); i++)

{

if (n % i == 0)

{

isPrime = false;

break;

}

}

Console.WriteLine(isPrime ? "Prime" : "Not Prime");

Console.ReadKey();