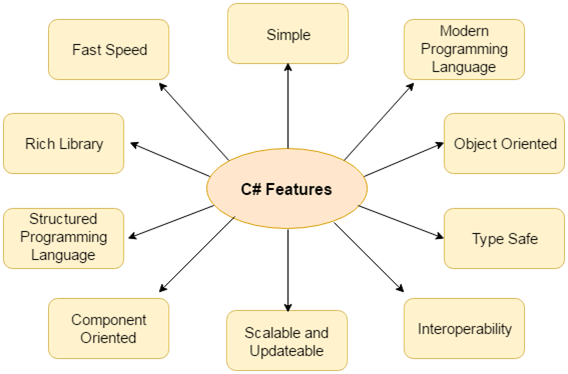
# **C#**

## Why to choose?

* **Comparatively easier:** Starting with C# is termed comparatively easier than other programming languages
* **Wider use of development:** Using C#, you are prone to create web applications or gaming apps. C# has some fantastic features like automatic garbage collection, interfaces, etc. which help build better applications.
* **Larger target audience**:



## **Garbage Collection:**

* Process of freeing up memory which is captured by unwanted objects.
* When you create objects, automatically memory is allocated in heap memory. after some time it becomes a waste.

### **Garbage collection happens in three cases:**

1. If the occupied memory by the objects exceeds the pre-set threshold value.
2. If the garbage collection method is called.
3. If your system has low physical memory.

## **Types of classes:**

* **Static:** does not allow inheritance.
* **Partial:** allows its members to partially divide or share source (.cs) files
* **Abstract:** classes that cannot be instantiated where you cannot create objects
* **Sealed:** class can’t be inherited.



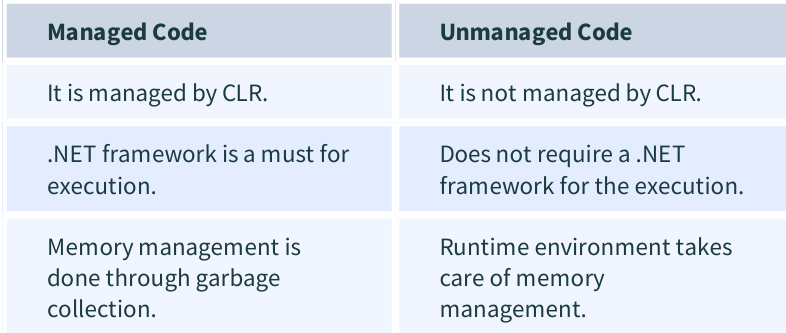
## **Managed code:**

Managed code is written in high-level languages like C#, Visual Basic, and F#, and is compiled into Intermediate Language (IL) code. The CLR then compiles and executes the IL code into machine code.

* Advantages: Improved code security, automatic garbage collection, dynamic type checking, and reference checking

## **Unmanaged code:**

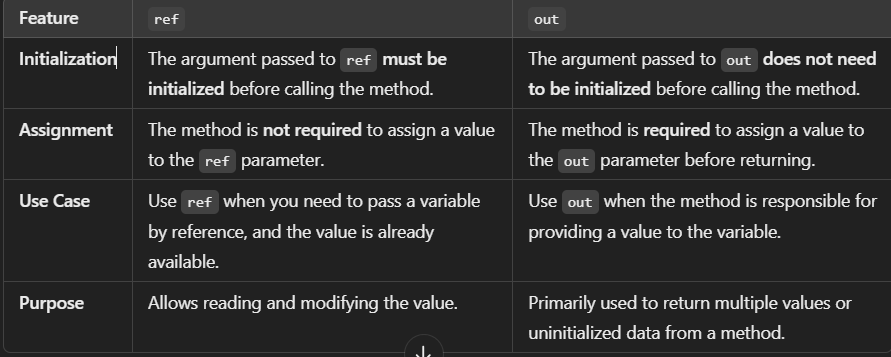
Runs directly on the hardware, giving the developer more control but requiring manual memory and pointers management.



## **Ref vs out**

**ref** is used when a method should both read and modify the value of a variable, and the variable is initialized before being passed.

**out** is used when a method should output a value or multiple values, and the variable does not need to be initialized before being passed but **must** be assigned within the method.



## **Extension methods:**

**Extension methods** in C# allow you to add new methods to existing types (like classes, structs, or interfaces) without modifying the original type or creating a new derived type. This is especially useful when you want to extend the functionality of a class that you don't have control over (e.g., built-in classes like string, int, etc.).

**Key Points:**

* Extension methods are **static** methods.
* They are defined in **static** classes.
* The first parameter of an extension method specifies the type it extends, and it is prefixed with the this keyword.
* They allow you to call the method as if it were part of the existing type.

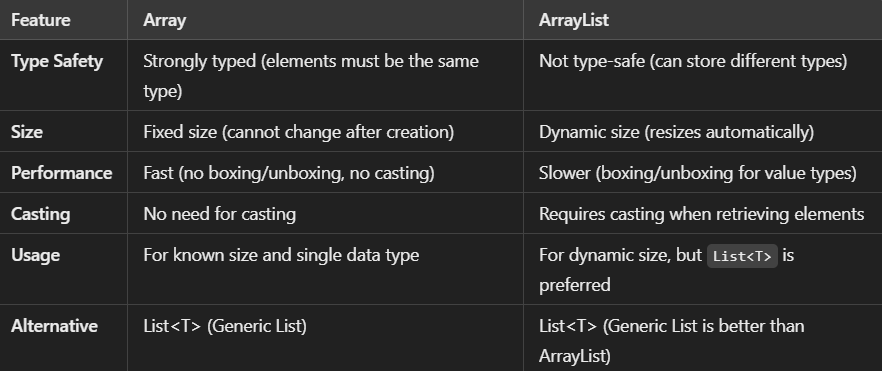
## **Generics:**

**Generics** in C# allow you to define classes, methods, interfaces, or delegates with placeholders for the types of objects they store or manipulate. This makes your code more flexible, reusable, and type-safe by enabling it to work with different data types without sacrificing type safety.

**Why Use Generics?**

1. **Type Safety**: You can ensure that only a specific type of object is used in your class or method.
2. **Code Reusability**: You can write a single class or method that works with different types.
3. **Performance**: Reduces the need for boxing/unboxing when using value types (like int, float).

## **Array vs ArrayList:**



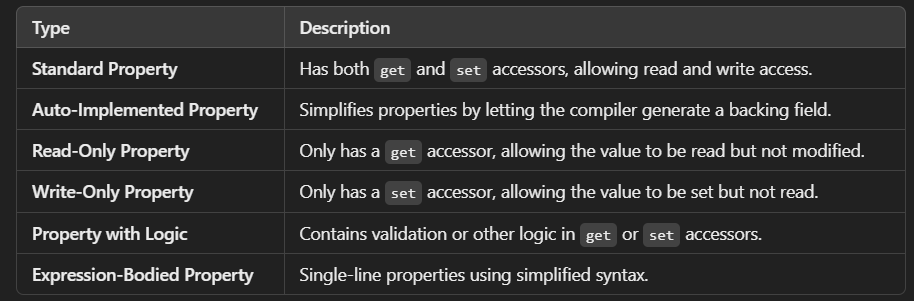
## **Properties:**

Properties in C# are public members of a class where they provide the ability to access private members of a class

**Properties** in C# are special class members that provide a flexible way to expose private fields while controlling how the data is accessed and modified. Properties encapsulate the fields of a class and allow you to validate the data before setting or getting its value. They are often used to create a layer of abstraction for class fields.

**Key Components of a Property:**

1. **Get Accessor (get)**: Used to **retrieve** the value of a property.
2. **Set Accessor (set)**: Used to **assign** a value to the property.



## **Arrays:**

When a group of similar elements is clubbed together under one name, they are called arrays.

* A few pointers for arrays in C#:
* The memory allocation is DYNAMIC.
* Arrays in C# are treated as objects.
* The length of the array is easy to find by detecting the number of members in the array.
* The members in the array are ordered and begin with the index value=0. The array types are reference types derived from the base array type

## **Indexers:**

**When to Use Indexers:**

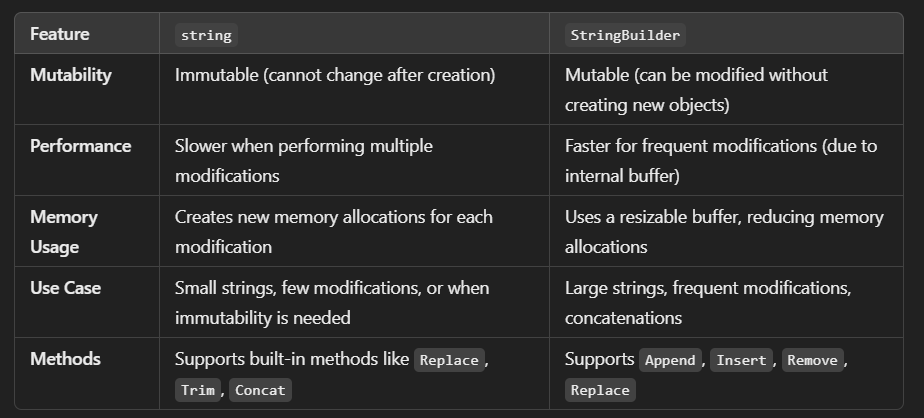
* When your class represents a collection or array-like structure and you want to provide intuitive access to the elements.
* When you want to allow multiple elements to be accessed using indexes, just like with arrays or lists.

**Summary:**

* **Indexers** provide a way to access class instances like arrays, making them ideal for classes that represent collections.
* They are defined using the this keyword and allow both get and set accessors.
* You can define multi-parameter indexers to support multi-dimensional array access.

In simple words, indexers give your class array-like behavior!

## String vs StringBuilder:



**When to Use:**

* **Use string**:
  + When the value is mostly static or doesn't require frequent modifications.
  + For smaller or single-instance string manipulations.
* **Use StringBuilder**:
  + When you need to perform frequent modifications like appending or replacing.
  + When concatenating strings inside loops or handling large dynamic content.

**A delegate in C# is an object that holds the reference to a method. It is like a function pointer in C++.**

# **.NET FRAMEWORK**

The .NET Framework is a software development platform that provides a common platform to run applications built using different programming languages like C#, VB.NET, and Visual Basic. It is a virtual machine that allows the development of various types of applications, including desktop, web, mobile, and services, that can run on the Windows platform. The .NET Framework consists of several key components, such as the Common Language Runtime (CLR), Common Type System (CTS), Base Class Library (BCL), and Framework Class Library (FCL), which work together to support the development and execution of .NET applications.

**Key Points**

**Overview of the .NET Framework**

* The .NET Framework is a virtual machine that provides a common platform to run applications built using different programming languages, such as C#, VB.NET, and Visual Basic.3
* The .NET Framework is used to create various types of applications, including form-based, console-based, mobile, and web-based applications or services, available in the Microsoft environment.4
* The .NET Framework is an object-oriented platform, similar to the Java language, but it is not platform-independent and can only run on the Windows platform.
* The main objective of the .NET Framework is to develop applications that can run on the Windows platform.
* The current version of the .NET Framework is 4.8.9

**Key Components of the .NET Framework**

1. **Common Language Runtime (CLR)**:
   * The CLR is a virtual component of the .NET Framework that executes programs written in different languages, such as C# and Visual Basic.
   * The CLR is the engine that executes all .NET programs and provides automatic services for these applications, such as security checking, memory management, and optimization.
   * The CLR converts the source code into byte code, known as CIL (Common Intermediate Language) or MSIL (Microsoft Intermediate Language), and then uses a JIT (Just-In-Time) compiler at runtime to convert the CIL/MSIL code into machine or native code.
2. **Common Type System (CTS)**: defines how types are declared, used, and managed in the runtime
   * The CTS specifies a standard for representing the types of data and values that can be defined and managed in computer memory at runtime.13
   * The CTS ensures that programming data defined in various languages can interact with each other and share information.1415
3. **Base Class Library (BCL)**:
   * The BCL is a rich collection of libraries, features, and functions that help to implement many programming languages in the .NET Framework, such as C#, F#, and Visual C++.16
   * The BCL includes the Common Language Specification (CLS), which defines a set of rules and regulations that every language in the .NET Framework must follow to ensure cross-language integration and interoperability.
4. **Microsoft .NET Assemblies**:
   * A .NET assembly is the main building block of the .NET Framework, containing a logical compiled code in the Common Language Infrastructure (CLI) and used for deployment, security, and versioning.2021
   * .NET assemblies are defined in two parts: DLL (Dynamic Link Library) and library (EXE) assemblies.22
   * When a .NET program is compiled, it generates metadata with Microsoft Intermediate Language, which is stored in a file called an Assembly.23
5. **Framework Class Library (FCL)**:
   * The FCL provides various pre built system functionalities in the .NET Framework, including classes, interfaces, and data types, to create multiple functions and different types of applications, such as desktop, web, and mobile applications.
   * The FCL can be considered the base on which various applications, controls, and components are built in the .NET Framework.

## **MSIL:**

MSIL is the Microso Intermediate Language, which provides instructions for calling methods, memory handling, storing and initializing values, exception handling, and so on.

## **Value type vs Reference type**

|  |  |
| --- | --- |
| **Value type** | **Reference type** |
| Holds the actual data within memory | Contains a pointer which consists of the address of another memory location |
| Stores its contents on stack memory | Heap memory |
| Assigning value to another variable will copy value directly | Doesn’t copy the value, instead creates a second copy of the reference |
| Eg. Predefined data types, structures, enums | Eg. Objects, Arrays, Indexers, Interfaces |

## **Delegate:**

A delegate is a .NET object which defines a method signature and it can pass a function as a parameter.

a **delegate** is like a **pointer to a method**. It allows you to **pass methods as parameters** to other methods. A delegate defines a method signature (return type and parameters), and any method with a matching signature can be assigned to the delegate.

**Why Use Delegates?**

1. **Flexibility**: You can change what method is executed at runtime.
2. **Callbacks**: When one method completes, it can call another method via a delegate (useful in asynchronous programming).
3. **Events**: Delegates are key to implementing events in C#.

|  |
| --- |
| // 1. Declare a delegate (it defines the method signature)  public delegate void Activity();  // 2. Define methods that match the delegate signature  public void GoShopping()  {  Console.WriteLine("Going shopping!");  }  public void WatchMovie()  {  Console.WriteLine("Watching a movie!");  }  // 3. Use the delegate to call methods  public void PlanDay()  {  // Declare a delegate instance  Activity myActivity;  // Assign method to delegate  myActivity = GoShopping;  myActivity(); // Output: Going shopping!  // Reassign method to delegate  myActivity = WatchMovie;  myActivity(); // Output: Watching a movie!  } |

## **IIS:**

Webserver

IIS can also act as a load balancer to distribute incoming HTTP requests to different application servers to allow high reliability and scalability.

It can also act as a reverse proxy, i.e. accept a client’s request, forward it to an application server, and return the client’s response. A reverse proxy improves the security, reliability, and performance of your application

# **MVC:**

## **Model:**

They hold data and its related Business logic. It handles the object storage and retrieval from the databases for an application. For example: A Controller object will retrieve the employee information from the database. It manipulates employee data and sends back to the database or uses it to render the same data.

## **View:**

View handles the UI part of an application. They get the information from the models for their display. For example, any employee view will include many components like text boxes, dropdowns, etc

The HomeController is the default controller for the application.

An index file is the default for the home controller in the view folder.

## **Controller:**

They handle the user interactions, figure out the responses for the user input and also render the final output. For instance, the Employee controller will handle all the interactions and inputs from the Employee View and update the database using the Employee Model.

# **Controller:**

* Controller is a class that handles user requests. It retrieves data from the Model and renders the view as the response.
* Controller processes incoming requests, handles user input and interactions and executes appropriate business logic.
* The ControllerBase class is a base class for all controller classes.

## **Tasks of Controller:**

* It locates for the appropriate action method to call and validate.
* It gets the values to use as the action method's arguments.
* It handles all errors that might occur during the execution of the action.
* It uses the WebFormViewEngine class for rendering ASP.NET pages.

## **ASP.NET Controller Actions and Parameters**

The controller defines action methods that are used to handle user requests and render view as the response. A controller can have any number of actions.

The MVC application uses the routing rules that are defined in the **Global.asax.cs** file. This file is used to parse the URL and determine the path of the controller.

## **ActionResult Return Type**

The ActionResult class is the base class for all action results. Action methods return an instance of this class.

|  |  |  |
| --- | --- | --- |
| **Action Result** | **Helper Method** | **Description** |
| ViewResult | View | It is used to render a view as a Web page. |
| PartialViewResult | PartialView | It is used to render a partial view. |
| RedirectResult | Redirect | It is used to redirect to another action method by using its URL. |
| RedirectToRouteResult | RedirectToAction RedirectToRoute | It is used to redirect to another action method. |
| ContentResult | Content | It is used to return a user-defined content type. |
| JsonResult | Json | It is used to return a serialized JSON object. |
| JavaScriptResult | JavaScript | It is used to return a script that can be executed on the client. |
| FileResult | File | It is used to return binary output to write to the response. |
| EmptyResult | (None) | It represents a return value that is used if the action method must return a null result. |

## **Action Method Parameters :-**

* Action parameters are the variables that are used to retrieve user requested values from the URL.
* The parameters are retrieved from the request's data collection. It includes name/value pairs for form data, query string value etc. the controller class locates for the parameters values based on the RouteData instance. If the value is present, it is passed to the parameter.

**URL:** **localhost:port-no/MusicStore/ShowMusic?MusicTitle=Classic**

|  |
| --- |
| public string ShowMusic(string MusicTitle)  {  return "You selected " + MusicTitle + " Music";  } |

## **MVC Action Selectors:**