C# Interview Questions

1. What is polymorphism ? give two **example** with code

   One object has many forms or has one name with multiple functionalities. "Poly" means many and "morph" means forms. Polymorphism provides the ability to a class to have multiple implementations with the same name. It is one of the core principles of Object Oriented Programming after encapsulation and inheritance.
   There are two types of polymorphism in C#:
   1. Static / Compile Time Polymorphism.
   2. Dynamic / Runtime Polymorphism. Static or Compile Time Polymorphism : It is also known as Early Binding. Method overloading is an example of Static Polymorphism. In overloading, the method has the same name but different signatures. It is also known as Compile Time Polymorphism because the decision of which method is to be called is made at compile time.

```
public int Add(int a, int b)
{
    int sum = a + b;
    return sum;
}

// adding three integer values.
public int Add(int a, int b, int c)
{
    int sum = a + b + c;
    return sum;
}
```

https://www.c-sharpcorner.com/UploadFile/ff2f08/understanding-polymorphism-in-C-Sharp/#:~:text=Method%20overloading%20is%20an%20example,same%20name%20but%20different%20signatures.&text=Here%20C%23%20compiler%20checks%20the,no%20matching%20method%20is%20found.

2. What is system.object ?
   It is the base class from which everything is derived
   ultimate base class of all .NET classes; it is the root of the type hierarchy

3. What is value type and reference type ?
https://www.tutorialsteacher.com/csharp/csharp-value-type-and-reference-type

4. what is managed code ? advantages of managed code ? what is CLR ?
Skip
5. What is garbage collection ?
Skip

6. Different versions of .net framework ?

It includes a large class library called Framework Class Library and provides language interoperability across several programming languages.

https://docs.microsoft.com/en-us/dotnet/framework/migration-guide/versions-and-dependencies

.NET is a developer platform made up of tools, programming languages, and libraries for building many different types of applications.

There are various implementations of .NET. Each implementation allows .NET code to execute in different places—Linux, macOS, Windows, iOS, Android, and many more.

1. .NET Framework is the original implementation of .NET. It supports running websites, services, desktop apps, and more on Windows.
2. .NET Core is a cross-platform implementation for running websites, services, and console apps on Windows, Linux, and macOS. .NET Core is open source on GitHub.
3.

7.What is type safety?

Type safety prevents assigning a type to another type when are not compatible.

```
1. int number = 40;
2. number = "String"; // Type safe language
```

This is clearly an error.

The advantages of type safety are pretty straightforward. At compile time, we get an error when a type instance is being assigned to an incompatible type; hence preventing an error at runtime. So at compilation time itself, developers come to know such errors and code will be modified to correct the mistake. So developers get more confidence in their code. Run time type safety ensures, we don't get strange memory exceptions and inconsistent behavior in the application.

http://rangahc.blogspot.com/2013/04/what-is-type-safety-in-net.html

8. What is metadata ? what is CTS ? what is Common language specification ?

Skip

9. What is .net standard ?

http://www.codedigest.com/quick-start/9/what-is-netstandard

https://github.com/dotnet/standard

10. What is the difference between "continue" and "break" statements in C#?

- Using break statement, you can jump out of a loop
- using continue statement, you can jump over one iteration and then resume your loop execution

```
{
    for (int i = 1; i <= 5; i++)
    {
        // Program comes out of loop when i becomes multiple of 3.
        if ((i % 3) == 0)
            break;
        else
            Console.WriteLine(i);
    }

    for (int i = 1; i <= 5; i++)
    {
        // The loop prints all values except those that are multiple of 3.
        if ((i % 3) == 0)
            continue;
        Console.WriteLine(i);
    }
}
```

11. What is generics? Open and close type ?
In C#, generic means not specific to a particular data type.
generics allow you to write a class or method that can work with any data type.

Advantages of Generics
Generics increase the reusability of the code. You don't need to write code to handle different data types.
Generics are type-safe. You get compile-time errors if you try to use a different data type than the one specified in the definition.
Generic has a performance advantage because it removes the possibilities of boxing and unboxing.

Example: Define Generic Class

```
class DataStore<T>
{
    public T Data { get; set; }
}
```

Instantiating Generic Class

```
DataStore<string> store = new DataStore<string>();
```

An "open generic type" is just a generic type that doesn't yet have its type specified
It becomes "closed" once a concrete type has been assigned

T, List<T>, and Dictionary<string,T>, and Dictionary<T,U> are all open types (T and U are type arguments) whereas

List<int> and Dictionary<string,int> are closed types.

```
4  public class Program
5  {
6      public static void Main()
7      {
8          Calculator<int> calc = new Calculator<int>();
9          int result = calc.Add(1, 2);
10         Console.WriteLine(result);
11     }
12 }
13
14 public class Calculator<T>
15 {
16     public T Add(T a, T b)
17     {
18         //return a+b won't work as the operator overload is not allowed
19         //dynamic is used to avoid type checking in compile time
20         dynamic dynamic1=a;
21         dynamic dynamic2=a;
22         return dynamic1+dynamic2;
23     }
24 }
```

C# 4.0 (.NET 4.5) introduced a new type called dynamic that avoids compile-time type checking. A dynamic type escapes type checking at compile-time; instead, it resolves type at run time.
https://dotnetfiddle.net/WKJ8Fb

12. How does garbage collection work ?
Skipped

13. Can you return multiple values from a function in C#?

https://www.c-sharpcorner.com/UploadFile/9b86d4/how-to-return-multiple-values-from-a-function-in-C-Sharp/#:~:text=We%20can%20return%20multiple%20values,Returning%20an%20Array
We can return multiple values from a function using the following 3 approaches:

- Reference parameters
- Output parameters
- Returning an Array
- Returning an object of class/struct type
- Returning a Tuple

- Reference parameters

Reference parameters also known as “ref” parameters
they are passed as reference to the function and after the function executes, the updated value of the passed reference variable is returned back to the calling method. It is important to note that reference parameters must be defined, initialized and assigned before they are passed to function else you may encounter a compile time error.

```
        int max = 10;
        int result=m1.MultipleReturns(30, 20, ref max);
        Console.WriteLine("result="+result+"max="+max);
    }
}
public class Maths
{
    public int MultipleReturns(int a, int b, ref int max)
        {
            if (a < b)
            {
                max = a;
                return a+b;
            }
            else
            {
                max = b;
                return a-b;
            }
        }
}
```

https://dotnetfiddle.net/QZQ5Qr

- Output Parameters

  Output parameters also known as “out” parameters, they are passed to the function as parameters and the calling method expects some values to be passed back in the parameter from the function. it is mandatory to assign values to out parameters in the function body.

  https://dotnetfiddle.net/OTHr21

- Returning an Array

  https://dotnetfiddle.net/0442eu
  Arrays can be used when you need to return single type of data

```
        result=m1.MultipleReturns(a, b);
        Console.WriteLine("max=" + result[0]+" result="+result[1]);
    }
}
public class Maths
{
    public int[] MultipleReturns(int a, int b)
    {
        int max,result;
        if (a < b)
        {
            max = a;
            result=a + b;
        }
        else
        {
            max = b;
            result=a - b;
        }
        int[] returnvalue=new int[]{result,max};
        return returnvalue;
```

```
    {
        Maths m1 = new Maths();
        int max, a = 30, b = 20;
        int result = m1.MultipleReturns(a, b, out max);
        Console.WriteLine("result=" + result + "max=" + max);
    }
}
public class Maths
{
    public int MultipleReturns(int a, int b, out int max)
    {
        if (a < b)
        {
            max = a;
            return a + b;
        }
        else
        {
            max = b;
            return a - b;
        }
    }
```

- Returning an object of Class/Struct Type
  https://dotnetfiddle.net/XtHU0v
- Returning a Tuple (skipped)

14. What is Exception Handling ? use of try,catch and finally. Can we add multiple catch statements? Sample code to write exception handling in C#

Exception handling in C# is a mechanism to detect and handle run-time errors in code.

C# exception handling is built upon four keywords: try, catch, finally, and throw.

- try – Contains a block of code for which an exception will be checked.
- catch – It is a program that catches an exception with the help of an exception handler.
- finally – It is a block of code written to execute regardless whether an exception is caught or not.
- Throw – Throws an exception when a problem occurs.

```
try
{
  int[] myNumbers = {1, 2, 3};
  Console.WriteLine(myNumbers[10]);
}
catch (Exception e)
{
  Console.WriteLine("Something went wrong.");
}
finally
{
  Console.WriteLine("The 'try catch' is finished.");
}
```

Yes, we can add multiple catch statements.
A parameterless catch block and a catch block with the Exception parameter are not allowed in the same try-catch statements, because they both do the same thing.

Also, parameterless catch block catch{ } or general catch block catch(Exception e){ } must be the last block.

```
try
{
    //code that may raise an exception
}
catch
{
    // this catch block must be last block
}
catch (NullReferenceException nullEx)
{
    Console.WriteLine(nullEx.Message);
}
catch (InvalidCastException inEx)
{
    Console.WriteLine(inEx.Message);
}
```

User-defined exception classes are derived from the Exception class

```
public class TempIsZeroException: Exception
{
   public TempIsZeroException(string message): base(message)
   {
   }
}
```

The throw statement allows you to create a custom error.

throw new *ArithmeticException("Access denied - You must be at least 18 years old.");*

15. What is Boxing and Unboxing? Example code

In C#, all types, predefined and user-defined, reference types and value types, inherit directly or indirectly from Object. So basically it is the base class for all the data types in C#.

When a variable of a value type is converted to object, it's called **boxing**. Boxing is implicit.

When a variable of type object is converted to a value type, it's called **unboxing**. Unboxing is explicit.

```
public class Program
{
    public static void Main()
    {
        int num=10;
        object o1=num; // Boxing

        object o2=1;
        int num2=(int)o2; //Unboxing
    }
}
```

https://dotnetfiddle.net/6muaNq

16. What is namespace in C# ? **example** code of nesting namespaces

Namespaces are used in C# to organize and provide a level of separation of codes. They can be considered as a container which consists of other namespaces, classes, etc. We can define a namespace in C# using the namespace keyword

A namespace inside a namespace is called a nested namespace in C#. This is mainly done to properly structure your code.

```
namespace N1     // N1
{
    class C1      // N1.C1
    {
        class C2   // N1.C1.C2
        {
        }
    }
    namespace N2  // N1.N2
    {
        class C2   // N1.N2.C2
        {
        }
    }
}
```

17 Inheritance ? **example** code.

Inheritance is a mechanism by which a new class is derived from an existing one. The new class, which is being created, is called a derived class and the class from which it is derived is called a base class. . This provides an opportunity to reuse the code functionality and speeds up implementation time. reduces code redundancy, supports code extensibility by overriding the base class functionality within derived classes

- Inheritance Eg with new keyword:

The new keyword is used to hide the method in derived class

https://dotnetfiddle.net/7PtaoR

- Using Virtual and Override in Inheritance

https://dotnetfiddle.net/buYe3M

Virtual and override are required when object created is of type base class.Otherwise the method in base class will be accessed instead of method in the derived class

BaseClass objectName=new DerivedClass();

- Using base keyword to access the members in base class from derived class

https://dotnetfiddle.net/hxJpkF

19. Write a generic function ? **example** code

A Generic method is a method that has Type Parameters as arguments or return and that can pass the actual data type later on.

```
public class Students
{
    public int Marks<T>(T num1, T num2)
    {
        dynamic mark1=num1;
        dynamic mark2=num2;
        return mark1+mark2;
    }
}
```

https://dotnetfiddle.net/rAtbFi

20. Write a generic class ? **example** code

A generic class is one which is defied with Type Parameters in angle brackets after the class name. The actual data types are specified when the class is instantiated.

https://dotnetfiddle.net/JwKtVt

```
public class Example<T>
{
    T number;
    public T Add(T number)
    {
        dynamic mynumber = number;
        return mynumber * 2;
    }
}
```

21. How to write a function which accepts another user defined class object as argument ? **example** code

```
public class Person
{
    public string name;
    public int age;
}

public class Operations
{
    public void Change(Person P1)
    {
        P1.name="Aneesh";
        P1.age=30;
    }
}
```

https://dotnetfiddle.net/1wQ2lk

22. Write a function to return a custom class object as return value ? **example** code

```
public class Rectangle
{
    public int width;
    public int length;
    public Rectangle Enlarge(int factor)
    {
        length = length * factor;
        width = width * factor;
        return this;
    }
}
```

https://dotnetfiddle.net/6lOE4o

23. Method Overloading using base class, object and passing derived object ? **example** code

If you pass a Derived object to a method, the overload with the most-derived arguments that match the object passed will be executed.

```
37 public class Methods
38 {
39     public void Print(object o)
40     {
41         Console.WriteLine(o.ToString());
42     }
43
44     public void Print(FirstClass f1)
45     {
46         Console.WriteLine(f1);
47     }
48
49     public void Print(SecondClass f2)
50     {
51         Console.WriteLine(f2);
52     }
53 }
```

https://dotnetfiddle.net/RtMr2x

http://www.csharp411.com/c-overloaded-methods-with-inherited-arguments/#comments

24. Different access modifiers available in C# ? what is internal ? what's the use of it ?

Access modifiers specify the accessibility of an object and all of its members in the C# project.

C# provides four types of access modifiers: private, public, protected, internal, and two combinations: protected-internal and private-protected.

Private Access Modifier : Objects that implement **private** access modifier are accessible only inside a class or a structure. As a result, we can't access them outside the class they are created:

Public Access Modifier: Objects that implement **public** access modifiers are accessible from everywhere in our project.

Protected Access Modifier: The **protected** keyword implies that the object is accessible inside the class and in all classes that derive from that class

Internal : The internal keyword implies that the access is limited to only the current Assembly, that is object or members declared as internal are accessible anywhere inside the same namespace. It is the ***default access modifier in C#***.

Internal keyword enables a group of components to cooperate in a private manner without being exposed to the rest of the application code.

Protected Internal Access Modifier: The **protected internal** access modifier is a combination of protected and internal. As a result, we can access the protected internal member only in the same assembly or in a derived class in other assemblies

Private Protected Access Modifier: The **private protected** access modifier is a combination of the private and protected keywords. We can access members inside the containing class or in a class that derives from a containing class, but only in the same assembly(project).

25. What are initializers in C# ? **example** code

In object initializer, you can initialize the value to the fields or properties of a class at the time of creating an object without calling a constructor.

```
        Author mahesh = new Author()
        {Name = "Mahesh Chand", Book = "LINQ Programming", publisher = "APress", Year = 2013, Price = 49.95};
    }
}

public class Author
{
    public string Name{get;set;}
    public string Book{get;set;}
    public string publisher{get;set;}
    public Int16 Year{get;set;}
    public double Price{get;set;}
}
```

https://dotnetfiddle.net/OJCnha

26. Explain Anonymous type in C# ? **example** code

In C#, an anonymous type is a type (class) without any name that can contain public read-only properties only. It cannot contain other members, such as fields, methods, events, etc.

You create an anonymous type using the new operator with an object initializer syntax. The implicitly typed variable- var is used to hold the reference of anonymous types.

```
public static void Main()
{
    //Object Initializer
    Author mahesh = new Author()
    {Name = "Mahesh Chand", Book = "LINQ Programming", publisher = "APress", Year = 2013, Price = 49.95};
    //Anonymous type
    var satheesh=new{Name = "Mahesh Chand", Book = "LINQ Programming", publisher = "APress", Year = 2013, Price = 49.95};
}
```

https://dotnetfiddle.net/xLf3Iy

27. How encapsulation is implemented in C#? Show **example** code

Encapsulation is defined 'as the process of enclosing one or more items within a physical or logical package' that prevents the data from being accessed by the code outside the shield.

Encapsulation can be achieved by declaring all the variables in the class as private and using C# Properties in the class to set and get the values of variables.

```
public class Author
{
    private string name;
    public string Name
    {
        get
        {
            return name;
        }
        set
        {
            name=value;
        }
    }
}
```

https://dotnetfiddle.net/nRWOc4

28. What is delegates in C# and uses of delegates? **Example** code

A delegate is a reference type variable that can hold a reference to the methods. Delegates in C# are similar to the function pointer in C/C++.

There are three steps involved while working with delegates:

1. Declare a delegate
2. Set a target method
3. Invoke a delegate

Uses of delegates:

- Provides a good way to encapsulate the methods.

- Delegates are mainly used in implementing the call-back methods and events
- Delegates can also be used in “anonymous methods” invocation.

```
public delegate int AddDel(int num1, int num2);     // 1. Declaring a delegate with same signature as the method
public class Calc
{
    public int Add(int n1, int n2)
    {
        return n1+n2;
    }
}
public class Program
{
    public static void Main()
    {
        Calc calc1=new Calc();
        AddDel myAddDel=new AddDel(calc1.Add); //Creating instance of the delegate and passing the method
        int result=myAddDel(1,2);  //Calling the delegate by passing the arguments
        Console.WriteLine(result);
    }
```

https://dotnetfiddle.net/j007iK

29. What is the difference between constants and readonly? When to use each ?

Difference between const and readonly

- const fields has to be initialized while declaration only, while readonly fields can be initialized at declaration or in the constructor.
- const variables can be declared in methods ,while readonly fields cannot be declared in methods.
- const fields cannot be used with static modifier, while readonly fields can be used with static modifier.
- A const field is a compile-time constant, the readonly field can be used for run time constants.

```
public class Test
{
    const double Pi = 3.14;     // Constants have to be initialized while declaring
    readonly string Name;       //Readonly variables doesn't have to be initialized while declaring
    static readonly int Age=30; //Readonly variables can be created with static keyword.
    public Test()
    {
        Name="Aneesh";          //Readonly variales needs to be initialized before the constructor exits

    }
    public void Print()
    {
        const int number=10;   //Constants can be initialzed inside Methods, Readonly fields cannot be initialized inside methods
        Console.WriteLine(number);
    }
}
```

If we are confident that the value of the variable won't ever change use const
If we have a constant that may change (e.g. w.r.t. precision).. or when in doubt, use a readonly.

30. What is the difference between Interface and Abstract Class? When to use abstract over interface

Data abstraction is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either abstract classes or interfaces

Abstract Class	Interface

It can contain both declaration and definition	It contains only a declaration part.
It contain constructor.	It does not contain constructor.
It can contain static members.	It does not contain static members.
It can contain different types of access modifiers like public, private, protected etc.	It only contains public access modifier because everything in the interface is public.
The performance of an abstract class is fast.	The performance of interface is slow because it requires time to search the actual method in the corresponding class.
Abstract class can contain methods,	Interface can only contain methods .

fields, constants, etc.	
It can be fully, partially or not implemented.	It should be fully implemented.
Multiple inheritance is not achieved by abstract class.	Multiple inheritance is achieved by interface.

- Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the derived class (inherited from).
- An abstract class is a good choice if you have plans for future expansion
- You should use an interface if you want a contract on some behavior or functionality

31.What is the difference between overloading and overriding? **Example** code

https://dotnetfiddle.net/4D1Yg3

```
    }
    public int Add(int num1, int num2) //Creating overload for Add method.
    {
        return num1+num2;
    }
    public int Add(int num1, int num2, int num3)//Creating overload for Add method.
    {
        return num1+num2+num3;
    }
}
public class Second:First
{
    public override void Print() //Overridden method in derived class
    {
        Console.WriteLine("Derived");
    }
}
```

Method Overloading	**Method Overriding**
Creating more than one method or function having same name but different	Creating a method in the derived class with the same signature as a method in

signatures or the parameters in the same class is called method overloading.	the base class is called as method overriding
It is called the compile time polymorphism	It is called runtime polymorphism
Method overloading is also called early binding.	Method overriding is also called late binding.
Method overloading is possible in single class only	Method overriding needs hierachy level of the classes i.e. one parent class and other child class.

32. What is the difference between ref and out keywords? **Example** code

Question 13

33. What is the difference between String and string in C#?

1. String is a class in .Net library while the string is an alias or keyword for 'System.String' Class.
2. You can't use String Class without importing System namespace, but string keyword can be used.
3. After Compilation of C# program string keywords is converted to System.String

34. What do you understand by regular expressions in C#? Write a program that searches a string using regular expressions.

Skip

35. What is an Indexer in C#, and how do you create one? **Example** code

An indexer allows an object of a class to be indexed like an array. When you define an indexer for a class, this class behaves similar to a virtual array.

Steps to create an indexer

1. Declare a **private** array of type that we will be passed to the class { private string[] arr = new string[10]; }
2. Now create a property of **this[int i]** with the type of the array

```
private string this[int i]
{
    get
    {
        return arr[i];
    }

    set
    {
        arr[i] = value;
    }
}
```

3. Create an object for the class and and the value can be passed and accessed from the object like **objectName[index]**

https://dotnetfiddle.net/tpTRqb

36.What are Nullable types? **Example** code

Nullable type allow null values to the assigned to variables

The main use of nullable type is in database applications to store null values.

sNullable type is also useful to represent undefined value.

https://dotnetfiddle.net/Qscxjy

```
using System;

public class Program
{
    public static void Main()
    {
        Nullable<int> num1 = null; // Two ways to declare Nu
        int? num2 = null; // Second method
        int? num3 = 23; // A nullable type initialized with
        Console.WriteLine(num2.GetValueOrDefault()); // GetV
        Console.WriteLine(num3.GetValueOrDefault());
    }
}
```

37. What is an extension method ? **example** code

Extension methods are static methods in a static class that allows us to add additional methods to an existing type without deriving, compiling or modifying the original type.

They can be used in a program by adding the namespace under which the extension method was written.

The first parameter of the extension method should start with **this** keyword.

```
namespace myExtension
{
    public static class Compare      //Should be a static class
    {
        public static bool IsGreater(this int i, int value) // Should be a static method, first pararmeter start with this
        {
            if (i > value)
                return true;
            else
                return false;
        }
    }
}
```

https://dotnetfiddle.net/BQumEk

XXX 38. What is lambda expression ? **example** code

A lambda expression is an anonymous function that can contain expressions and statements. They make the code more readable and compact.

47. Static constructor ? **example** code

Static Constructor is useful to perform a particular action only once throughout the application. If we declare a constructor as static, then it will be invoked only once irrespective of the number of class instances