Introduction to .Net Framework.

Q. 1	**What is .Net Platform?**
Ans:	The .NET Platform is used to develop enterprise applications based on industry standards. It was introduced to offer a much more powerful, more flexible, and simpler programming model than COM. It is a fully managed, protected, simplified, feature rich application execution environment.
Q. 2	**Features of .Net**
Ans:	It is OS independent and hardware independent. Extensively uses Industry standards like HTTP, SOAP, XML and XSD. Easily maintainable due to simplified deployment and version management. A platform to build Web services, Multi-Threaded Applications, Windows Services, Rich Internet Applications as well as Mobile Application. Provides seamless integration to a wide variety of languages.
Q. 3	**Which are the major components of .Net Platform?**
Ans:	Common Language Runtime – CLR Framework Class Library – FCL or Base Class Library – BCL
Q. 4	**Roles of CLR**
Ans:	Running of code Memory management Compilation of code (JIT) Provides garbage collection Error handling Code access security for semi-trusted
Q. 5	**Roles of FCL or BCL**
Ans:	Framework class library is object-oriented collection of reusable classes. You use them to develop applications like: 1. System-Side Programing 2. Graphical User Interface (GUI) based Desktop applications 3. Web Applications or Web Sites 4. Web Services and Web API's 5. Distributed Applications
Q. 6	What is MSIL?
Ans:	MSIL stands for Microsoft Intermediate Language. Its an assembly level language, got when we compile source code written in any .net language. Its machine independent. CLR converts MSIL into native code using JIT compiler.
Q. 7	**What is JIT?**
Ans:	JIT stands for Just-In-Time compiler. It converts MSIL code into native code.
Q. 8	**What is Managed Code?**
Ans:	Code which run or executes under the control of the CLR is called managed code. CLR takes care of allocation and deallocation of memory.
Q. 9	**What is Unmanaged Code?**

Ans:	Code which run or executes outside the control of CLR is called unmanaged code. Example code written in legacy languages such as VB or C++ or COM applications.
Q. 10	**What is Assembly?**
Ans:	When we compile the source code, the IL code is stored in an assembly. It's a smallest unit Deployment, Versioning and Execution.
Q. 11	**What is a namespace?**
Ans:	Namespaces are the logical containers for storing related types together. They help in avoiding name clashes. System namespace is the root namespace in FCL|BCL.
Q. 12	**What is Memory Management?**
Ans:	.Net is completely object oriented. So objects have to be created at runtime and memory has to be allocated. When objects are no longer need by the application object's memory must be reclaimed. This process of allocating and deallocating memory is called Memory Management and is done by Garbage Collector.

New Features of .Net Framework 4.6

Q. 1	**What is Roslyn?**
Ans:	The .NET Compiler Platform ("Roslyn") provides open-source C# and Visual Basic compilers with rich code analysis APIs. You can build code analysis tools with the same APIs that Microsoft is using to implement Visual Studio.
Q. 2	**What is Dependency Injection?**
Ans:	Dependency Injection is the process of "injecting" the "dependency" whenever the dependency is requested by a client, where the dependency may be another service which the client may have no means of knowing how to create.
Q. 3	**What does ECDSA mean?**
Ans:	Elliptic Curve Digital Signature Algorithm
Q. 4	**What is Soft keyboard support in WPF?**
Ans:	Soft Keyboard support enables focus tracking in a WPF applications by automatically invoking and dismissing the new Soft Keyboard in Windows 10 when the touch input is received by a control that can take textual input.
Q. 5	**What is UWP?**
Ans:	Windows now offers capabilities to bring existing Windows desktop apps, including WPF and Windows Forms apps, to the Universal Windows Platform (UWP). This technology acts as a bridge by enabling you to gradually migrate your existing code base to UWP, thereby bringing your app to all Windows 10 devices.
Q. 6	**What are the features of .Net Core?**
Ans:	Flexible deployment: Can be included in your app or installed side-by-side user- or machine-wide. Cross-platform: Runs on Windows, macOS and Linux; can be ported to other operating systems. Command-line tools: All product scenarios can be exercised at the command-line. Compatible: .NET Core is compatible with .NET Framework, Xamarin and Mono, via the .NET Standard Open source: The .NET Core platform is open source, using MIT and Apache 2 licenses.
Q. 7	**When to use .Net Core for your server Applications?**

Ans:	You have cross-platform needs. You are targeting microservices. You are using Docker containers. You need high-performance and scalable systems. You need side-by-side .NET versions per application.
Q. 8	**When to use .Net Framework for your server Applications?**
Ans:	Your app currently uses .NET Framework Your app uses third-party .NET libraries or NuGet packages not available for .NET Core. Your app uses .NET technologies that aren't available for .NET Core. Your app uses a platform that doesn't support .NET Core.

Introduction to C#

Q. 1	Why C#?
Ans:	C# is a modern, object-oriented language that enables programmers to quickly build a wide range of applications for the new Microsoft .NET platform, which provides tools and services that fully exploit both computing and communications.
Q. 2	What is Type Safety?
Ans:	Unsafe type conversion is checked automatically by CLR. ValueTypes defaults to Zero and ReferenceTypes defaults to null.
Q. 3	What is Interoperability?
Ans:	It's the ability of the Language to use features from applications or services which were created in a legacy language.
Q. 4	What is a DLL?
Ans:	DLL stands for dynamic Link Libraries. We create libraries or otherwise called components which serves a application. If the same component is required in other application, the library can be reused.

C# 6.0 New Features

Q. 1	What is using static feature?
Ans:	As with the namespaces, we can include a static class in the using statement similar to a namespace. Syntax: using static System.Console;
Q. 2	What is string interpolation?
Ans:	Using string interpolation, you can now directly write your arguments instead of referring them with placeholders inside a string. Syntax: $"I am {name}"
Q. 3	How to use Dictionary Initializer in C# 6.0?

Ans:	Dictionary<string, string=""> dictionaryObj = new Dictionary<string, string="">()
	{
	["Name"] = "Fizzy",
	["Planet"] = "Kepler-452b"
	};

Q. 4	How to use Auto-Property Initializers?
Ans:	public decimal Salary {get; set;} = 10000;

	public string Name {get;} = “Rahul Shah";
Q. 5	What is null conditional operator and null propagation?
Ans:	A new notion of null conditional operator where you can remove declaring a conditional branch to check to see if an instance of an object is null or not with this new ?. ?? null conditional operator syntax. The ?. is used to check if an instance is null or not, if it's not null then execute the code after ?. but if it is not, then execute code after ??. var result = A?.B??C // If A is null B is assigned to result else C will be assigned.
Q. 6	How to use Expression Bodied Function & Property?
Ans:	private static double AddNumbers(double x, double y) => x + y; public string FullName => FirstName + " " + LastName;
Q. 7	How to use Exception Filtering?

Ans:	Exception filtering is nothing but some condition attached to the catch block. Execution of the catch block depends on this condition. Let me give you a simple example.
	Syntax:
	catch (Exception ex)
	{
	if(ex.Message.Equals("400"))
	Write("Bad Request");
	else if (ex.Message.Equals("401"))
	Write("Unauthorized");
	}

Data Types and Arrays

Q. 1	What are ValueTypes and Where is memory of them allocated?
Ans:	Value types are defined in C# by using the struct keyword. Instances of value types are the only kind of instances that can live on the stack.
Q. 2	What are ReferenceTypes and Where is memory of them allocated?
Ans:	Variables of reference types, also referred to as **objects**, store references to the actual data. Reference types are stored in Managed Heap.
Q. 3	What is Boxing and Unboxing?
Ans:	Boxing and unboxing enable value types to be treated as objects.
Q. 4	What is syntax for using Nullable types?
Ans:	Nullable<int> number; or int? number
Q. 5	What are implicit type variable?
Ans:	Type of the local variable being declared is inferred from the expression used to initialize the variable. Syntax: var number = 10;
Q. 6	How to use Single-Dimension array?
Ans:	int[] numbers = new int[5]; or int[] numbers = new int[5] {1,2,3,4,5}; or int[] numbers = {1,2,3,4,5};
Q. 7	How to use Multi-Dimension array?
Ans:	int[,] numbers = new int[2,3]; or

	int[,] numbers = new int[2,3] {{1,2,3},{4,5,6}};
Q. 8	What are jagged arrays?
Ans:	Jagged array is an array whose elements are arrays. The elements of a jagged array can be of different dimensions and sizes. A jagged array is sometimes called an “array of arrays”.
Q. 9	How to use Jagged array?

Ans:	int[][] numbers = new int[2][];
	numbers[0] = new int[3];
	numbers[1] = new int[5]; or
	
	int[][] numbers = new int[2][];
	numbers[0] = new int[3] {1,2,3};
	numbers[1] = new int[5] {4,5,6,7,8};

OOPs in C#

Q. 1	Types of classes?
Ans:	Concrete class Abstract class Static class Sealed class
Q. 2	What is value parameter?
Ans:	The value parameter is the default parameter type in C#. If a parameter does not have any modifier, it is the value parameter by default. When you use the value parameter, the actual value is passed to the function.
Q. 3	What is ref parameter?
Ans:	The ref parameters are input/output parameters. That means, they can be used for passing a value to a function as well as for getting back a value from a function. We create a ref parameter by preceding the parameter data type with a ref modifier. Whenever, a ref parameter is passed, a reference is passed to the function.
Q. 4	What is out parameter?
Ans:	The out parameters are ‘output only’ parameters. That means, they can only return a value from a function. We create an out parameter by preceding the parameter data type with an out modifier. Whenever an out parameter is passed only an unassigned reference is passed to the function.
Q. 5	What is params parameter?
Ans:	The value passed for a params parameter can be either a comma-separated value list or a single dimensional array. The params parameters are ‘input only’. Only one parameter in the parameter list be using params.
Q. 6	Types of Polymorphism
Ans:	Static or Compile Time Polymorphism Dynamic or Runtime Polymorphism
Q. 7	How many static constructors per class?
Ans:	Just one
Q. 8	Types of properties

Ans	ReadWrite – both get and set blocks ReadOnly – only get block WriteOnly – only set block
Q. 9	Types of inheritance
Ans	Single Multi-Level Hierarchy Hybrid Multiple – Not supported in C#
Q. 10	What is a virtual method?
Ans	A virtual method is a method that is declared as virtual in a base class and redefined in one or more derived classes. Each derived class can have its own version of a virtual method.
Q. 11	What is an abstract method?
Ans:	Abstract methods do not have an implementation. Abstract methods must belong to an abstract class. Every concrete derived class must override all the base-class abstract methods and properties using the keyword override.
Q. 12	What is a sealed class?
Ans:	To prevent inheritance, a sealed modifier is used to define a class. A sealed class is the one that cannot be used as a base class. Sealed classes can't be abstract. All structs are implicitly sealed.
Q. 13	Why seal a class?
Ans:	For prevention of unintended derivation For code optimization For resolution of Virtual function calls at compile-time
Q. 14	What is a struct?
Ans:	Struct is a user defined type like class. Can have constructors, data fields, properties, methods and events. Struct is a value type hence memory is allocated on the stack.
Q. 15	What are extension methods?
Ans:	It is a special kind of static method. Allows the addition of methods to an existing class outside the class definition. Without creating a new derived type. Without re-compiling or modifying the original type.

SOLID

Q. 1	What is SOILD?
Ans:	SOLID principles are the design principles that enable us to manage with most of the software design problems. Robert C. Martin compiled these principles in the 1990s. These principles provide us ways to move from tightly coupled code and little encapsulation to the desired results of loosely coupled and encapsulated real needs of a business properly.
Q. 2	What does SOILD stands for?
Ans:	S: Single Responsibility Principle (SRP) O: Open Closed Principle (OCP) L: Liskov Substitution Principle (LSP)

	I: Interface Segregation Principle (ISP) D: Dependency Inversion Principle (DIP)
Q. 3	What is SRP?
Ans:	Robert C Martin expresses the principle as “A Class should have only one reason to change”. Every Module or class should have responsibility over a single part of the functionality provided by the software and that responsibility should be entirely encapsulated by the class
Q. 4	What is OCP?
Ans:	“Software entities should be open for extension, but closed for modification” The design and writing of the code should be done in a way that new functionality should be added with minimum changes in the existing code. The design should be done in way to allow the adding of new functionality as new classes keeping as much as possible existing code unchanged.
Q. 5	What is LSP?
Ans:	Introduced by Barbara Liskov “Objects in a program should be replaceable with instance of their subtypes without altering the correctness of that program” If a program module is using a Base class, then the reference to the Base class can be replaced with a Derived class without affecting the functionality of the program module. We can also state that Derived types must be substitutable for their base types
Q. 6	What is ISP?
Ans:	“Many client-specific interfaces are better than one general purpose interface” We should not enforce clients to implement interfaces that they don’t use , instead of creating one big interface we can break down it to smaller interfaces
Q. 7	What is DIP?
Ans:	One should “depend upon abstractions not concretions” Abstraction should not depend on the details where as the details should depend on abstractions. High level modules should not depend on low level modules.
Q. 8	Why SOLID?
Ans:	If we don’t follow SOLID Principles, then We • End up with tight or strong coupling of the code with many other modules/application • Tight coupling causes time to implement any new requirement, features or any bug fixes and sometimes it creates unknown issues • End up with a code which is not testable • End up with duplication of code • End up creating new bugs by fixing another bug • End up with many unknown issues in the application development cycle

Exception Handling

Q. 1	What is an Exception?
Ans:	An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions during the execution of a program. Exceptions are

	notifications that some error has occurred in the program. When an exception occurs you can ignore the exception or you can write code to deal with the exception, this is known as exception handling.
Q. 2	What is Exception handling?
Ans:	While executing a program if a run-time error occurs, an exception is generated; this is usually referred to as an exception being thrown. An exception is an object that contains information about the runtime error which has occurred. We can use various techniques to act on an exception. In general, this is known as exception handling, and it involves writing code that will execute when an exception is thrown.
Q. 3	Exception handling Overview
Ans:	C# exception handling is managed via four keywords: try, catch, throw, and finally. Try block: Contains program statements you wish to monitor for exceptions. If an exception occurs within the try block, it is thrown. Catch block: Your code catches this exception using catch and handles it in some rational manner. Finally block: Any code that you must execute after you exit a try block is put in a finally block.
Q. 4	How many Catch blocks?
Ans:	Associate more than one catch statement with a try. Each catch must catch a different type of exception. If you wish to use an Exception class in multiple catch statements, it should be the last catch statement.
Q. 5	What is User Defined Exceptions?
Ans:	Although C#'s built-in exceptions handle most common errors, C#'s exception handling mechanism is not limited to them. You can use custom exceptions to handle errors in your own code. As a rule, exceptions you define should be derived from ApplicationException as this is the hierarchy reserved for application-related exceptions.
Q. 6	Which is the base class for all exceptions?
Ans:	System.Exception class is the base class for all the exception
Q. 7	From which class should User Defined Exception class inherit?
Ans:	System.ApplicationException class
Q. 8	All built-In Exception classes inherit from which base class?
Ans:	System.SystemException class

Collections and Generics

Q. 1	What's the need for Collections?
Ans:	Using collections, you can store several items within one object. Collections have methods that you can use to add and remove items. These methods automatically resize the corresponding data structures without requiring additional code.
Q. 2	What are Collections?

Ans:	In C#, collections are: ▪ Groups of objects. ▪ Enumerable data structures that can be accessed using indexes or keys. The .NET Framework: ▪ Has powerful support for collections. ▪ It contains a large number of interfaces and classes that define and implement various types of collections.
Q. 3	Which are the collection interfaces?
Ans:	ICollection, IList, IDictionary, IEnumberable, IComparable, IComparer, etc.
Q. 4	Which are the collection classes?
Ans:	BitArray, Stack, Queue, ArrayList, Hashtable, SortedList, etc..
Q. 5	Which collection follows LIFO principle?
Ans:	Stack Class
Q. 6	Which type of collection make use of Key-Value Pair?
Ans:	Dictionary
Q. 7	What is IEnumerable?
Ans:	An enumerator is an object that provides a forward, read-only cursor for a set of items. The IEnumerable interface has one method called the GetEnumerator() method. Classes implementing this method must return a class that implements the IEnumerator interface.
Q. 8	What is IEnumerator?
Ans:	Defines the notion of a cursor that moves over the elements of a collection. Has three members for moving the cursor and retrieving elements from the collection.
Q. 9	What is a ArrayList?
Ans:	The ArrayList class is a dynamic array of heterogeneous objects. In an array we can store only objects of the same type. However, in an ArrayList we can have different types of objects. These in turn would be stored as object type only. An ArrayList uses its indexes to refer to a particular object stored in its collection.
Q. 10	What is Hashtable?
Ans:	Creates a collection that uses a hash table for storage. Represents a dictionary of associated keys and values, implemented as a hash table. Provides a faster way of storage and retrieval of items of the object type. Provides support for key based searching. The GetHashCode() method of the Hashtable class returns the hash code for an object instance.
Q. 11	What is Generics?
Ans:	Generics permit classes, structs, interfaces, delegates, and methods to be parameterized by the types of data they store and manipulate. C# generics will be immediately familiar to users of generics in Eiffel or Ada or to users of C++ templates; however, they do not suffer many of the complications of the latter. Generics provide a facility for creating types that have type parameters.
Q. 12	What is constraint in generics?
Ans:	Commonly, a generic class will do more than just store data based on a type parameter. Often, the generic class will want to invoke methods on objects whose type is given by a type parameter. To provide stronger compile-time type checking and reduce type casts, C# permits an optional list of. constraints to be supplied for each type parameter.

	A type parameter constraint specifies a requirement that a type must fulfill in order to be used as an argument for that type parameter.Constraints are declared using the word where, followed by the name of a type parameter, followed by a list of class or interface types and optionally the constructor constraint new().
Q. 13	What is a generic method?
Ans:	A generic method has one or more type parameters specified in < and > delimiters after the method name. When calling a generic method, type arguments are given in angle brackets in the method invocation
Q. 14	What is a generic interface?
Ans:	It is often useful to define interfaces either for generic collection classes, or for the generic classes that represent items in the collection. With generic classes it is preferable to use generic interfaces
Q. 15	What are Iterators?
Ans:	An iterator is a method, get accessor or operator that enables you to support foreach iteration in a class or struct without having to implement the entire IEnumerable interface.
Q. 16	What are collection initializers?
Ans:	A collection initializer consists of a sequence of element initializers, enclosed by { and } tokens and separated by commas. Each element initializer specifies an element to be added to the collection object being initialized. To avoid ambiguity with member initializers, element initializers cannot be assignment expressions.

Delegates, Events and Lambda

Q. 1	What is a delegate?
Ans:	A delegate is a reference type that refers to a Shared method of a type or to an instance method of an object. The closest equivalent of a delegate in other languages is a function pointer. Whereas a function pointer can only reference Shared functions, a delegate can reference both Shared and instance methods.
Q. 2	Why delegates?
Ans:	In general, delegates are useful as they: ▪ Support events. ▪ Give your program a way to execute methods at runtime without precise knowledge of which method it is at compile time.
Q. 3	What is a multicast delegate?
Ans:	A delegate which holds reference to multiple methods is called as an multicast delegate.
Q. 4	What is a generic delegate?
Ans:	Like methods, delegates can also be generic. The syntax is similar to that of a generic method, with the type parameter being specified after the delegate's name.
Q. 5	What is an event?
Ans:	An event is a way for a class to provide notifications when something of interest happens.
Q. 6	In Event model who is a Publisher?

Ans:	A publisher is an object that maintains its internal state. When its state changes, it can raise an event to alert other interested objects about the change.
Q. 7	In Event model who is a Subscriber?
Ans:	A subscriber is an object that registers an interest in an event. It is alerted when a publisher raises the event. An event can have zero or more subscribers.
Q. 8	What is an Anonymous method?
Ans:	Anonymous methods allow you to create an instance of a delegate by specifying a block of inline code to be invoked by the delegate rather than specifying an existing method to be invoked by the delegate. This allows you to conserve code because you do not need to create a new method every time a delegate is passed as a parameter to a method call.
Q. 9	What are Lambda in C#?
Ans:	A lambda expression is an anonymous function that can contain expressions and statements and can be used to create delegates or expression tree types. All lambda expressions use the lambda operator =>, which is read as "goes to". The left side of the lambda operator specifies the input parameters (if any) and the right side holds the expression or statement block. The lambda expression x => x * x is read "x goes to x times x."

Garbage Collection

Q. 1	What's is garbage collection?
Ans:	As you have seen, objects are dynamically allocated from a pool of free memory by using the new operator. Of course, memory is not infinite, and the free memory can be exhausted. Thus, it is possible for new to fail because there is insufficient free memory to create the desired object. For this reason, one of the key components of any dynamic allocation scheme is the recovery of free memory from unused objects, making that memory available for subsequent reallocation. In many programming languages, the release of previously allocated memory is handled manually. However, C# uses a different, more trouble-free approach: garbage collection.
Q. 2	What is a Finalizer?
Ans:	Finalize is a protected method in the object class. You can override this method in your class by implementing a destructor.
Q. 3	What is a dispose method?
Ans:	Rather than implementing a destructor, your class can implement IDisposable interface, which has a Dispose method to clean up you resources.

IO and Serialization

Q. 1	What is a file?
Ans:	A file is an ordered and named collection of a sequence of bytes having persistent storage.
Q. 2	What is directory?

Ans:	Directory are nothing but folders in windows OS. Directory provides static methods for creating, moving, and enumerating through directories and subdirectories. The DirectoryInfo class provides instance methods.
Q. 3	What is a FileStream?
Ans:	FileStream supports random access to files through its Seek method. FileStream opens files synchronously by default but supports asynchronous operation as well. File contains static methods, and FileInfo contains instance methods.
Q. 4	What is Serialization?
Ans:	Serialization is the process of taking objects and converting their state information into a form that can be stored or transported. The basic idea of serialization is that an object writes its current state, usually indicated by the value of its member variables, to temporary (either memory or network streams) or persistent storage. Later, the object can be re-created by reading, or deserializing, the object's state from storage. Serialization handles all the details of object pointers and circular object references that are used when you serialize an object.
Q. 5	Why serialization?
Ans:	Serialization is done so that the object can be recreated with its current state at a later point in time or at a different location.
Q. 6	What is a Formatter?
Ans:	A formatter is used to determine the serialization format for objects. All formatters expose an interface called the IFormatter interface.
Q. 7	Types of Formatters used in IO?
Ans:	Binary formatter SOAP formatter
Q. 8	What's is Serializable & NonSerialized Attributes?
Ans:	To make an object available for serialization, you mark each class with the [Serializable] attribute. If you determine that a given class has some member data that should not participate in the serialization scheme, you can mark such fields with the [NonSerialized] attribute.
Q. 9	What are the restrictions on Binary serialization?
Ans:	The class to be serialized must either be marked with the SerializableAttribute attribute, or must implement the ISerializable interface and control its own serialization and deserialization. The binary format produced is specific to the .NET Framework and it cannot be easily used from other systems or platforms. The binary format is not human-readable, which makes it more difficult to work with if the original program that produced the data is not available.
Q. 10	What are the restrictions on SOAP serialization?
Ans:	The class to be serialized must either be marked with the SerializableAttribute attribute or must implement the ISerializable interface and control its own serialization and deserialization. Only understands SOAP. It cannot work with arbitrary XML schemas.
Q. 11	What's the use of IDeserializationCallback interface?
Ans:	The IDeserializationCallback interface specifies that a class is to be informed when deserialization of the whole object graph has been finished.

	To enable your class to initialize a nonserialized member automatically, use the IDeserializationCallback interface and then implement IDeserializationCallback.OnDeserialization.
Q. 12	What are the benefits of SOAP serialization?
Ans:	Class can serialize itself, to be self contained. Produces a fully SOAP-compliant envelope that can be processed by any system or service that understands SOAP. Supports either objects that implement the ISerializable interface to control their own serialization, or objects that are marked with the SerializableAttribute attribute. Can deserialize a SOAP envelope into a compatible set of objects. Can serialize and restore non-public and public members of an object.

Assemblies, Reflection and Attributes

Q. 1	What is an assembly?
Ans:	Assemblies are the building blocks of any .NET application. All functionality of .NET application is exposed via assemblies. Assemblies form a unit of deployment and versioning. Assemblies contain modules which in turn contain various types (classes, structures, enumerations etc.)
Q. 2	What are the Benefits of Assemblies?
Ans:	Assemblies provide the following benefits: Promote code reuse Establish a Type Boundary Are version able and self-describing Entities Enable Side-By-Side Execution
Q. 3	Types of Assemblies
Ans:	Private assembly Shared assembly
Q. 4	What is a Private Assembly
Ans:	Private assemblies are a collection of types that are only used by the application with which it has been deployed. Private assemblies are required to be located within the main directory of the owing application.
Q. 5	What is a Shared Assembly?
Ans:	Shared assemblies can be used by several clients on a single machine Shared assemblies are installed into a machine wide “Global Assembly Cache” (GAC) A shared assembly must be assigned a “shared name” (also known as a “strong name”)
Q. 6	What is a strong name?
Ans:	A strong name contains the following information: ▪ Friendly string name and optional culture information (just like a private assembly) ▪ Version identifier ▪ Public key value ▪ Embedded digital signature
Q. 7	How to create a strong name?

Ans:	To provide a strong name for an assembly, generate the public / private key data using sn.exe utility. This will create a strong name key file that contains data for two distinct but mathematically related keys, the “public” key and the “private” key.
Q. 8	What is metadata?
Ans:	Every Assembly is self describing i.e. it consists of meta data which describes itself. Metadata is defined as “Data about data”. Metadata is generated by compiler and stored in the EXE or DLL. Metadata also contains data about System Level attributes.
Q. 9	What is reflection?
Ans:	Reflection is ability to find information about types contained in an assembly at run time. .NET provides a whole new set of APIs to introspect assemblies and objects. All the APIs related to reflection are located under System.Reflection namespace. .NET reflection is a powerful mechanism which allows you to inspect type information and invoke methods on those types at runtime.
Q. 10	What are Attributes?
Ans:	Attributes concept in .NET is a way to mark or store meta data about the code in assembly. Often it is an instruction meant for the runtime. The Runtime can change its behavior or course of action based on the attribute present.

Parallel and Async programming with C#

Q. 1	What is Optional parameter?
Ans:	Optional parameters allows you to omit arguments in method invocation. They become very handy in cases where parameter list is too long.
Q. 2	What is named arguments?
Ans:	Named arguments is a way to provide an argument using its parameter name, instead of relying on its position in the argument list. Named arguments act as an in-code documentation to help you remember which parameter is which. A Named argument is declared simply by providing the name before argument value An argument with argument-name is a named argument, an argument without an argument-name is a positional argument.
Q. 3	What is dynamic keyword?
Ans:	C# 4.0 supports late-binding using a new keyword called ‘dynamic’. The type ‘dynamic’ can be thought of like a special version of type ‘object’, which signals that the object can be used dynamically. C# provides access to new DLR (Dynamic language runtime) through this new dynamic keyword. When dynamic keyword is encountered in the code, compiler will understand this is a dynamic invocation & not the typical static invocation
Q. 4	What is TPL?
Ans:	The Task Parallel Library (TPL), as its name implies, is based on the concept of the task. Tasks are a new abstraction in .NET 4 to represent units of asynchronous work. Tasks were not available in earlier versions of .NET, and developers would instead use ThreadPool work items for this purpose. The term task parallelism refers to one or more independent tasks running concurrently.
Q. 5	What is task?

Ans:	A task represents an asynchronous operation, and in some ways, it resembles the creation of a ThreadPool work item, but at a higher level of abstraction.
Q. 6	Types of parallelism
Ans:	Data Parallelism Task Parallelism
Q. 7	What is Data Parallelism?
Ans:	When a parallel loop runs, the TPL partitions the data source so that the loop can operate on multiple parts concurrently. Behind the scenes, the Task Scheduler partitions the task based on system resources and workload.
Q. 8	What is Task Parallelism?
Ans:	This breaks down the program into multiple tasks which can be parallelized and are executed on different processors. This is supported by Task Parallel Library (TPL) in .NET and the Task class. Implement the “Task Asynchronous Pattern” TAP
Q. 9	What is PLinq?
Ans:	PLINQ provides a parallel implementation of LINQ and supports all the standard query operators. This is achieved by the System.Linq.ParallelEnumerable class which provides a set of extension methods on IEnumerable interface. Developers can opt-in for parallelism by invoking the AsParallel method of ParallelEnumerable class on the data source.
Q. 10	When to use async?
Ans:	To keep your application responsive while waiting for something that is likely to take a long time to complete. To make your application as scalable as possible (particularly important for services and ASP.NET that have limited threads/resources to service requests)
Q. 11	What is await?
Ans:	Typically, when a developer writes some code to perform an asynchronous operations using threads, then he/she must explicitly write necessary code to perform wait operations to complete a task. In asynchronous methods, every operation which is getting performed is called a Task. The ‘await’ keyword is applied on the task in an asynchronous method and suspends the execution of the method, until the awaited task is not completed.
Q. 12	What are Async rules?
Ans:	The compiler will give you a warning for methods that contain async modifier but no await operator A method that isn’t using the await operator will be run synchronously. Async methods can return any of the following: ▪ **Void** ▪ **Task** ▪ **Task<T>** Several BCL methods now have an Async version for you to use with the await keyword

Encryption and Decryption

Q. 1	What is Encryption?
Ans:	*Encryption* is the process of transforming plain data in a way that makes it harder for an unauthorized person to make sense of it. The encrypted data is called *ciphertext*.
Q. 2	What is decryption?
Ans:	*Decryption* is the reverse process, meaning that having the ciphertext, you must apply a transformation to it to get back the original information.
Q. 3	What is Symmetric Encryption?
Ans:	Symmetric encryption is also known as shared secret encryption, and that is because the encryption of the data is done with an encryption key, a byte array, and the same key is used to decrypt the data.
Q. 4	What is Hashing?
Ans:	*Hashing* is the process of mapping binary data of a variable length to a fixed size binary data.
Q. 5	What is a hash?
Ans:	A hash is a numerical representation of a piece of data and can be thought of as a digital fingerprint.
Q. 6	What is Asymmetric Encryption?
Ans:	Asymmetric encryption uses two mathematically related keys that complement each other, such as whatever is encrypted with one key can be decrypted only with the other key. One key is made public, and is known as the *public key*, by the receiving party, so whoever wants to transmit secured data can encrypt the data.
Q. 7	When to use which encryption?
Ans:	For data privacy, use Aes. For data integrity, use HMACSHA256 or HMACSHA512. For digital signatures, use RSA or ECDsa. For key exchange, use RSA or ECDiffieHellman. For random number generation, use RNGCryptoServiceProvider. For generating a key from a password, use Rfc2898DeriveBytes.
Q. 8	When to use which algorithm?
Ans:	If you need to encrypt data that is used locally, or you have a secure way to distribute the encryption key, use the symmetric encryption If you don't have a secure way to send the encryption key data between parties, then asymmetric encryption is recommended If you need only to ensure integrity of the data, use a hashing algorithm If you need to ensure both integrity and authenticity, choose a MAC algorithm
Q. 9	When to use Symmetric encryption?
Ans:	Based on a common key called shared secret It needs an initialization vector (IV) that doesn't need to be secret but is used to encrypt the first block of data You use it by instantiating a symmetric algorithm object and then calling CreateEncryptor or CreateDecryptor The encryptor/decryptor is then used with either by calling directly the TransformFinalBlock method or by sending it to a CryptoStream
Q. 10	What is key management?

Ans:	Symmetric keys can be exchanged using asymmetric algorithms Asymmetric private keys can be secured either by using certificates or by using Crypto Service Providers containers

Dockers

Q. 1	What is a Docker?
Ans:	The Docker platform is the only container platform to build, secure and manage the widest array of applications from development to production both on premises and in the clouds.
Q. 2	Which are two supported implementations for building server-side containerized Docker applications with .NET?
Ans:	.Net Framework .Net Core
Q. 3	When to choose .NET Core for Docker containers?
Ans:	Developing and deploying cross platform Using containers for new ("green-field") projects Creating and deploying microservices on containers Deploying high density in scalable systems
Q. 4	When to choose .NET Framework for Docker containers?
Ans:	Migrating existing applications directly to a Windows Server container Using third-party .NET libraries or NuGet packages not available for .NET Core Using.NET technologies not available for .NET Core • ASP.NET Web Forms • WCF services • Workflow-related services. Windows Workflow Foundation (WF), Workflow Services (WCF + WF in a single service), and WCF Data Services (formerly known as ADO.NET Data Services) are only available on .NET Framework. Using a platform or API that does not support .NET Core
Q. 5	What is an image?
Ans:	An image is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and config files.
Q. 6	What is a docker container?
Ans:	A container is a runtime instance of an image—what the image becomes in memory when executed. It runs completely isolated from the host environment by default, only accessing host files and ports if configured to do so.
Q. 7	What is Compose in Docker?
Ans:	Compose is used to define applications using multiple Docker containers.
Q. 8	Which are the major component of Docker Engine?
Ans:	A server which is a type of long-running program called a daemon process (the dockerd command).

	A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do. A command line interface (CLI) client (the docker command).
Q. 9	Where to use Dockers?
Ans:	Your developers write code locally and share their work with their colleagues using Docker containers. They use Docker to push their applications into a test environment and execute automated and manual tests. When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation. When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.