1. **OOPs and its principles.**

The four principles of object-oriented programming are encapsulation, abstraction, inheritance, and polymorphism.

* 1. **Abstraction**: *Abstractions is the design time process in which we hide the implementation and show only necessary things. For example, in C# we have a method called as Console.WriteLine which is use to display text on the console. We use this method everywhere but don’t know the internal implementation detail of this method. The part we can see is the method definition, the input parameters as well as return type of the method. This is what is called as abstraction. In this Console.Writeline method Microsoft is exposing only the required details (that is method name, input types and return types) but they are not exposing the implementation details of the method this is what is called as abstraction.*
  2. **Encapsulation:** *Encapsulation is implementation time process. It is a process of binding the data members and member functions into a single unit.*

*Let's assume we have to create a method to insert user’s data and pass it to other developers to use. So first, I need to create a class with a “AddUser” method to insert the data into database with validation.*

*Let there will be three fields: Name, Email and Phone number. So these inputs have to validate first and then insert into db.**For this we have to create two private methods one for validation and another for db insertion. So these methods cannot be accessed from outside of class and whole add user functionality is encapsulated inside a class.*

*Here, we are hiding the procedure of adding data into database from other users, this is Abstraction. And putting all the three methods into one User class and providing other users to use it, that is called Encapsulation.*

*So procedure hiding is Abstraction and putting every necessary thing into one is Encapsulation.*

* 1. **Polymorphism:** *In object-oriented programming world, polymorphism is expressed as 'one interface, multiple functions'. Polymorphism can be static or dynamic. In static polymorphism, the response to a function is determined at the compile time. In dynamic polymorphism, it is decided at run-time.*
* ***Static or Compile Time Polymorphism***

*It is also known as Early Binding. Method overloading is an example of Static Polymorphism. In Overloading, the method / function 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. Overloading is the concept in which method names are the same with a different set of parameters.*

*Here the compiler checks the number of parameters passed and the type of parameter and make the decision of which method to call and it throw an error if no matching method is found.*

* ***Dynamic / Runtime Polymorphism***

*Dynamic / runtime polymorphism is also known as late binding. Here, the method name and the method signature (number of parameters and parameter type must be the same and may have a different implementation). Method overriding is an example of dynamic polymorphism.*

*Method overriding can be done using inheritance. With method overriding it is possible for the base class and derived class to have the same method name and same something. The compiler would not be aware of the method available for overriding the functionality, so the compiler does not throw an error at compile time. The compiler will decide which method to call at runtime and if no method is found then it throws an error.*

* 1. **Inheritance***:*

*Acquiring (taking) the properties of one class into another class is called inheritance. Inheritance provides reusability by allowing us to extend an existing class.*

*The reason behind OOP programming is to promote the reusability of code and to reduce complexity in code and it is possible by using inheritance.*

*Types:*

*Single, Multilevel, Hierarchical (This is the type of inheritance in which there are multiple classes derived from one base class. This type of inheritance is used when there is a requirement of one class feature that is needed in multiple classes).*

1. **OOPs- Association, aggregations and composition.**
2. **OOPS- SOLID principles and implementation with IOC containers.**
3. **OOPs- Static class vs Singleton class. Why should we use singleton over static classes.**
4. **C#- Singleton class with thread safe implementation.**
5. **C#- Generics, delegates, generic delegates and collections.**
6. **C#- Concepts of virtual override and new keyword in inheritance.**
7. **C#- Async and await.**
8. **Design level difference between interface and abstract classes.**
9. **Web API vs WCF.**
10. **SQL- normalization and their types.**
11. **SQL- de-normalization and its benefits.**
12. **SQL- practical implementations of joins, LEFT, INNER, RIGHT, SELF, CROSS and FULL.**
13. **SQL- temp table vs table variable vs common table expressions and their uses.**
14. **SQL-Views and Updatable views.**
15. **SQL- Triggers with types and uses.**
16. **Stored procedures and its practical uses and best practices.**
17. **SQL- Aggregate functions, rank, dense rank and row numbers.**
18. **SQL- Aggregation queries with group by.**
19. **SQL- Get product count using group by.**
20. **SQL- Union vs Union All.**
21. **SQL- Union vs Join.**
22. **SQL- Find intersection of two tables using full join.**
23. **ASPNET- Session vs View state.**
24. **DATAWAREHOUSE- OLTP vs OLAP Servers.**