(OOPs)Object Oriented Programming Theory Questions

1. What is OOPs? Explain its key features.

- **OOPs stands for Object-Oriented Programming**. OOPs focuses on the concept of objects that contain both data (attributes) and behaviors (methods) and how these objects interact with each other.

- The key features of OOPs are as follows:

1. **Encapsulation**: Encapsulation is the mechanism of hiding the internal details and data of an object and exposing only the necessary interfaces to interact with the object. It helps in achieving data abstraction and improves security and maintainability of the code.

2. **Inheritance**: Inheritance allows the creation of new classes (derived classes) based on existing classes (base classes). The derived classes inherit the properties and behaviors of the base classes, which promotes code reuse and allows the creation of a hierarchical class structure. It enables the implementation of "is-a" relationships between classes.

3. **Polymorphism**: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It provides the ability to use a single interface to represent different types of objects. Polymorphism is achieved through method overriding (runtime polymorphism) and method overloading (compile-time polymorphism), enabling flexibility and extensibility in the code.

4. **Abstraction**: Abstraction is the process of identifying the essential characteristics and behavior of an object and creating a simplified model of the object. It focuses on what an object does rather than how it does it. Abstraction allows the creation of abstract classes and interfaces, providing a blueprint for defining common attributes and methods for a group of objects.

5. **Modularity**: Modularity refers to the concept of breaking down a complex problem into smaller, manageable modules or classes. Each class represents a specific functionality or behavior, making the code easier to understand, maintain, and debug. Modularity enhances code reusability and promotes better organization and collaboration among developers.

2. What is a class, and how is it different from an object?

- In OOPs, a class is a blueprint or template for creating objects. It defines a set of attributes (data members) and behaviors (methods) that an object of that class will have. A class serves as a template for creating multiple objects with similar characteristics and behaviors.

- An object, on the other hand, is an instance of a class. It is a concrete representation of a class that has been created and has its own unique identity. An object has a state (the values of its attributes) and behavior (the methods it can perform).

- In simpler terms, a class is like a recipe for creating objects, while an object is like a dish made using that recipe. A class defines the attributes and behaviors that an object will have, while an object is an actual instance of the class that has its own unique data and can perform the defined behaviors.

- To create an object, you first define a class, and then create instances of that class by calling the class constructor. For example, if you have a class named Person that has attributes like name, age, and gender, and methods like speak() and walk(), you can create an object of the Person class by calling its constructor.
- like so: person1 = Person("sujit", 26, "Male")

- Here, person1 is an object of the Person class with name "sujit", age 26, and gender "Male". You can then use the methods of the Person class to perform actions on this object, like calling person1.speak() to make the object speak.

3. What is the purpose of an init method in a class, and how is it used?

- The __init__ method is a special method in Python classes that is used to initialize the object's attributes when an instance of the class is created. It is also called the constructor method. The purpose of the __init__ method is to define and initialize the instance variables (attributes) of the object.

- The __init__ method takes the self parameter, which refers to the object itself, and any additional parameters required to initialize the object. Within the __init__ method, you can set the values of the object's attributes based on the values of the parameters.

- Here's an example:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

- In this example, the __init__ method initializes the name and age attributes of the Person object with the values passed as parameters. When an instance of the Person class is created, the __init__ method is called automatically and the attributes are initialized.
  like so: person1 = Person("sujit", 26)

- Here, person1 is an object of the Person class with name attribute set to "sujit" and age attribute set to 26.

- The __init__ method is a very important part of defining a class, as it ensures that each instance of the class is properly initialized with the necessary attributes. Without the __init__ method, you would have to manually set each attribute of the object after creating it, which can be tedious and error-prone.

4. What is inheritance, and how is it implemented?

- Inheritance is a fundamental concept in OOPs that allows a class to inherit properties or attributes and methods from another class. The class that is being inherited from is called the superclass or parent class, while the class that inherits from the superclass is called the subclass or child class.

- Inheritance allows for code reuse and reduces redundancy, as the subclass can reuse the code and behavior defined in the superclass, while adding its own unique behavior.

- There are 5 types of inheritance in python.

    1. **Single Inheritance**: Single inheritance involves a class inheriting properties and behaviors from a single base class. The derived class inherits all the members of the base class, and it can add its own unique members. This is the simplest form of inheritance.

    2. **Multiple Inheritance**: Multiple inheritance occurs when a class inherits from more than one base class. In this case, the derived class inherits the properties and behaviors of all the base classes. Multiple inheritance allows for code reuse from multiple sources, but it can lead to complexities and conflicts if not managed carefully.

    3. **Multilevel Inheritance:** Multilevel inheritance involves a class being derived from another derived class. It forms a hierarchical structure where each derived class serves as the base class for the next level. This type of inheritance allows for creating a depth of inheritance hierarchy.

    4. **Hierarchical Inheritance**: Hierarchical inheritance refers to a situation where multiple derived classes inherit from a single base class. It forms a tree-like structure with one base class and multiple derived classes branching out from it. Each derived class inherits the properties and behaviors of the base class while adding its own specific features.

    5. **Hybrid (or Mixed) Inheritance**: Hybrid inheritance is a combination of more than one inheritance. It involves multiple base classes and derived classes forming a complex inheritance structure. This type of inheritance is used when a class needs to inherit from multiple classes with different functionalities.

5. What is polymorphism, and how is it implemented?

- Polymorphism is a fundamental concept in object-oriented programming that allows objects of different classes to be treated as if they were objects of a common parent class. It means that different objects can respond to the same message or method call in different ways, depending on their specific class or type.

- There are two main ways to implement polymorphism in Python:
  1. **Method overriding:** This is when a subclass defines a method with the same name as a method in its parent class. The method in the subclass overrides the method in the parent class, so when the method is called on an object of the subclass, the subclass's implementation is executed instead of the parent class's implementation.

  2. **Method overloading:** This is when a class defines multiple methods with the same name but different parameters. When the method is called, Python determines which version of the method to execute based on the number and types of the arguments passed to it. However, method overloading is not directly supported in Python, unlike some other languages such as Java and C++. One common way to simulate method overloading in Python is to use default arguments or variable-length argument lists.

6. What is encapsulation, and how is it achieved?

- Encapsulation is the practice of hiding the internal details of an object and providing a public interface for interacting with it. It is a fundamental concept in object-oriented programming that helps to promote code modularity, reusability, and maintainability.
- In Python, encapsulation is achieved using access specifier, which restrict the visibility of a class's properties and methods from outside the class. There are four types of access specifier in Python:

  1. **Public Access Specifier:** Public access specifier is a type of access specifier in object-oriented programming that allows properties and methods of a class to be accessible from anywhere, both within the class and outside of it. In Python, all properties and methods of a class that are not prefixed with an underscore (_) are considered public. Using public access specifier, you can define properties and methods of a class that can be accessed and modified by any part of the program. This can be useful in situations where you want to expose the internal details of a class to the outside world or when you want to create a reusable library that can be used by other developers.

2. **Private Access Specifier:** Private access specifier is a type of access Specifier in object-oriented programming that allows properties and methods of a class to be accessible only within the class, and not outside of it. In Python, properties and methods of a class can be made private by prefixing them with two underscores (__). Using private access specifier, you can define properties and methods of a class that can be accessed and modified only by the class itself, and not by any other part of the program. This can be useful in situations where you want to hide the internal details of a class from the outside world or when you want to prevent other developers from modifying the internal state of a class directly.

3. **Protected Access Specifier:** Protected access specifier is a type of access Specifier in object-oriented programming that allows properties and methods of a class to be accessible only within the class and its subclasses, but not outside of them. In Python, properties and methods of a class can be made protected by prefixing them with a single underscore (_). Using protected access specifier, you can define properties and methods of a class that can be accessed and modified only by the class itself and its subclasses, but not by any other part of the program. This can be useful in situations where you want to allow the subclasses to access and modify certain properties and methods of the base class, but not the outside world.

4. **Getters and Setters:** Getters and setters are methods used to access and modify the values of private or protected properties of a class, respectively. They provide an additional layer of encapsulation and control over the access and modification of the class's properties.In Python, getters and setters are usually implemented using properties. A property is a special type of method that looks and behaves like an attribute of a class, but it has additional code that gets executed when the property is accessed or modified. The @property decorator is used to define a getter method, while the @<property_name>.setter decorator is used to define a setter method.

7. What is the difference between a private and a protected attribute or method in a class?

- In object-oriented programming, private and protected are access modifiers that define the visibility of attributes and methods of a class.

Private attribute	Protected attribute
1. Private attributes and methods are accessible only within the class in which they are defined.	1. Protected attributes and methods are accessible within the class in which they are defined and its subclasses.
2. They are denoted by a double underscore prefix (__).	2. They are denoted by a single underscore prefix (_).
3. Private attributes and methods cannot be accessed or modified from outside the class, including its subclasses.	3. Protected attributes and methods can be accessed and modified from within the class and its subclasses, but not from outside the class hierarchy.

- The main difference between private and protected attributes and methods is in their scope of visibility. Private attributes and methods are more restrictive than protected attributes and methods, as they cannot be accessed from outside the class, including its subclasses. Protected attributes and methods provide a limited level of access to the class hierarchy, allowing subclasses to access and modify them, but not the outside world.

8. What is method overriding, and how is it implemented?

- Method overriding is a feature of object-oriented programming that allows a subclass to provide its own implementation of a method that is already defined in its superclass. When a method in a subclass has the same name, same return type, and same parameters as a method in its superclass, the method in the subclass overrides the method in the superclass.

- Method overriding is implemented by defining a method in a subclass with the same signature (name, return type, and parameters) as a method in its superclass. When the method is called on an object of the subclass, the overridden method in the subclass is executed instead of the method in the superclass.

9. How does Python support multiple inheritance, and what are its benefits and drawbacks?

- Python supports multiple inheritance, which means a class can inherit from multiple parent classes. This allows for more flexibility and code reusability, as a subclass can inherit features from multiple parent classes. To implement multiple inheritance in Python, we simply list the parent classes in the class definition, separated by commas.

- Benefits of multiple inheritance:

    1. Code reusability: Multiple inheritance allows us to reuse code from multiple parent classes in a single child class.

    2. Flexibility: Multiple inheritance provides more flexibility in designing class hierarchies as we can inherit from multiple parent classes to create a more specific subclass.

- Drawbacks of multiple inheritance:

    1. Complexity: Multiple inheritance can lead to complex class hierarchies and code, which can be difficult to understand and maintain.

    2. Ambiguity: Multiple inheritance can lead to ambiguity if two or more parent classes have methods with the same name. This can make it difficult to determine which method is being called in the child class.

    3. Diamond problem: The diamond problem is a specific issue that can occur with multiple inheritance when two parent classes have a common ancestor. This can result in method conflicts and can be difficult to resolve. However, Python provides a method resolution order (MRO) to determine the order in which methods are called in the inheritance hierarchy, which helps to resolve the diamond problem.

10. What are abstract classes and methods in Python OOPs, and how are they used?

- Abstract classes and methods are a key feature of object-oriented programming (OOP) that allow us to define a class or method that must be implemented in a subclass. In Python, we can create an abstract class or method using the abc module.

- An abstract class is a class that contains at least one abstract method, which is a method that has no implementation in the abstract class. Abstract classes cannot be instantiated and must be subclassed. The subclass must implement all the abstract methods defined in the abstract class.

11. What is the difference between a static method and a class method, and when would you use each?

- Both static methods and class methods are methods that are defined at the class level rather than the instance level. However, there are some key differences between them.

Static method	Class method
1. A static method is a method that belongs to a class and does not depend on any particular instance of that class.	1. A class method is a method that belongs to a class and operates on the class itself rather than on its instances.
2. It can be called using the class name or an instance of the class.	2. It can be called using the class name or an instance of the class.
3. It cannot access or modify the state of the class or its instances.	3. It can access and modify the state of the class but not the state of its instances.
4. It is defined using the @staticmethod decorator.	4. It is defined using the @classmethod decorator.

- When to use static method:
  1. When a method does not depend on any particular instance of the class and does not modify the state of the class or its instances.
  2. When a method is a utility method that does not use any instance variables or methods.

- When to use class method:
  1. When a method operates on the class itself rather than on its instances and needs to modify the class variables.
  2. When a method needs to access or modify the class-level state.

12.What are global, protected and private attributes?

- In Python, there are three types of attribute access specifiers: global, protected, and private. These access specifiers determine the level of access to class attributes and methods.

    1. **Global attributes:** Global attributes are accessible from anywhere within the code. They are not prefixed with any underscores. For example, a global attribute can be accessed as my_class.global_attribute

    2. **Protected attributes:** Protected attributes are those attributes that are prefixed with a single underscore. They can be accessed within the class and its subclasses but not outside of the class hierarchy. Protected attributes are intended to indicate that they should not be accessed directly from outside the class hierarchy. For example, a protected attribute can be accessed as my_class._protected_attribute.

    3. **Private attributes:** Private attributes are those attributes that are prefixed with double underscores. They can be accessed only within the class where they are defined and not from outside the class hierarchy. Private attributes are intended to indicate that they should not be accessed directly, even within the class hierarchy. For example, a private attribute can be accessed as my_class.__private_attribute.

- In general, it is recommended to use protected and private attributes to prevent accidental modification of class state and to enforce data encapsulation. However, it's worth noting that in Python, there is no strict enforcement of private and protected attributes, and they can still be accessed using name mangling or other means.

13.What is the use of self in Python?

- In Python, self is a reference variable that refers to the instance of the class, which is currently being operated on. It is the first argument to instance methods in a class and is used to access instance variables and methods within the class.

- When a method is called on an instance of a class, Python automatically passes the instance itself as the first argument to the method, which is why self is used as the parameter name. This allows the method to access and modify the instance's state.

14.Are access specifiers used in python?

- Yes, Python supports access specifiers to some extent, although it does not enforce them as strictly as some other programming languages like Java or C++.

- In Python, access specifiers are implemented using naming conventions rather than language keywords. By convention, attributes and methods that are intended to be private are prefixed with a double underscore (__) while those that are intended to be protected are prefixed with a single underscore (_). Public attributes and methods are not prefixed with any underscores.

15.Is it possible to call parent class without its instance creation?

- Yes, it is possible to call a parent class method without creating an instance of the parent class. This can be done using the super() function.

- The super() function returns a temporary object of the superclass, which allows you to call its methods. You can then call the method on this object, passing in the instance of the subclass as the first argument.

16.How is an empty class created in python?

- To create an empty class in Python, you can simply use the class keyword followed by the name of the class and a colon.
- For example:
  ```
  class MyClass:
      pass
  ```
- The pass statement is used as a placeholder to indicate that the class does not have any content yet. It is not strictly necessary, but including it makes the intention of the code clearer.

17.How will you check if a class is a child of another class?

- To check if a class is a child of another class, you can use the issubclass() built-in function in Python. The issubclass() function takes two arguments: the first argument is the class you want to check, and the second argument is the potential parent class.

18.What is docstring in Python?

- A docstring is a documentation string in Python that is used to describe the purpose, usage, and behavior of a function, module, class, or method. It is a string literal that appears as the first statement in a module, function, class, or method definition. The docstring is enclosed in triple quotes (""") and is immediately after the header of the module, function, class, or method.

- Docstrings are important because they can be accessed using the built-in help() function. The help() function displays the docstring for the specified module, function, class, or method.

- Docstrings also help other developers understand how to use your code and can help you remember how your own code works if you come back to it after a long time.

19.Is Python Object-oriented or Functional Programming?

- Python is a multi-paradigm programming language, which means that it supports multiple programming paradigms, including object-oriented programming (OOP) and functional programming (FP). Python allows you to write code in both OOP and FP styles, and you can choose the paradigm that is most appropriate for the problem you are trying to solve

- Python's support for OOP is one of its main strengths. Python has built-in support for classes and objects, which makes it easy to write object-oriented code. You can define classes, create objects, and use inheritance and polymorphism in Python. Python also has a number of built-in classes and modules that you can use to create and work with objects.

- On the other hand, Python also supports functional programming. Python allows you to write code in a functional style by using features like lambda functions, map, reduce, and filter. Python also has built-in support for generators and comprehensions, which are powerful functional programming features.

- So, to sum up, Python is both an object-oriented and a functional programming language. You can use either paradigm, or a combination of both, to write Python code.

20.What does an object() do?

- The object() function in Python returns a new empty object of the built-in class object. This class is the root of the Python class hierarchy and serves as the base class for all built-in classes in Python.

- The object() function is typically used when you want to create a new object without any attributes or methods defined. You can then add attributes and methods to the object using dot notation.

- Note that while the object() function returns a new object, it doesn't create a new class. If you want to create a new class, you'll need to use the class keyword in Python.

21. What is the purpose of the super function in inheritance, and how is it used?

- The super() function in Python is used to call a method in a parent class from a subclass that overrides the method. When a method is overridden in a subclass, the subclass can call the overridden method in the parent class using the super() function.
- The primary purpose of the super() function is to enable cooperative multiple inheritance in Python. When a subclass inherits from multiple parent classes that have the same method name, the super() function ensures that the method is called only once, in the correct order, and with the correct arguments.

22. What is data abstraction ?

- Data abstraction is a programming technique used in object-oriented programming (OOP) that focuses on creating abstract classes or interfaces that expose only the essential features of a class, hiding all the implementation details from the user. It is the process of hiding complex implementation details of a class from the user and providing only the relevant information that is necessary for the user to perform the required operations.

- Abstraction allows you to focus on what an object does instead of how it does it. It simplifies the programming process by providing a clear and concise interface for users to interact with. It helps to reduce complexity and makes the code more understandable and maintainable.

- In Python, data abstraction can be achieved through abstract classes or interfaces, which define a set of methods that must be implemented by any class that inherits from them. By providing a clear set of guidelines for how a class should be implemented, abstract classes or interfaces help to ensure that code is consistent and maintainable across different parts of a program.