1. What is Object-Oriented Programming (OOP)?
   - Object-Oriented Programming (OOP) in Python is a programming paradigm centered around the concept of "objects," which are instances of "classes." These objects encapsulate data (attributes) and the methods (functions) that operate on that data. OOP aims to structure code in a way that mirrors real-world entities and their interactions, promoting code reusability, maintainability, and modularity.

2. What is a class in OOP?
   - In Object-Oriented Programming (OOP), a class is a user-defined data type that serves as a blueprint for creating objects. It defines the structure (data members) and behavior (member functions or methods) that objects of that class will have. Think of it as a template that specifies what an object will be like.

3. What is an object in OOP?
   - Object is an instance of class. Object holds attribute and methods. Object is unique and each object has different value.

4. What is the difference between abstraction and encapsulation?
   Abstraction and encapsulation are two fundamental concepts in object-oriented programming, but they serve different purposes.
   - Abstraction is the process of hiding the complex implementation details of a system and showing only the essential features to the user.
   - It helps reduce complexity by allowing the programmer to focus on what an object does rather than how it does it.
   - For example, when you drive a car, you only need to know how to use the steering wheel and pedals, not how the engine works internally.
   - On the other hand, encapsulation is the practice of wrapping data and the methods that operate on that data into a single unit, typically a class.
   - It restricts direct access to some of an object's components, which helps protect the integrity of the data.
   - This is usually done by making variables private and providing public getter and setter methods.
   - While abstraction deals with hiding complexity, encapsulation deals with hiding the internal state and protecting data.

5. What are dunder methods in Python?
   - **Dunder methods** (short for "double underscore") in Python are special methods with names that start and end with double underscores (e.g., __init__, __str__, __len__).
   - They are used to define the behavior of objects for built-in operations like initialization, printing, addition, and more.
   - These methods allow customization of how objects interact with Python's syntax and operators.

6. Explain the concept of inheritance in OOP?
   - Inheritance in oops means child class inherits all the properties of parents class.
   - It promotes code reusability and allows the creation of hierarchical relationships.
   - Example:

```
class Parent_class:
    def method(self):
        Print("Parent Class")
class Child_class(Parent_class):
    def method2(self):
        print("Child class")
ob1=Child_class()
ob1.method() #Parent Class
ob1.method2()  #Child Class
```

7. What is polymorphism in OOP?
   - Polymorphism, derived from the Greek words "poly" (many) and "morph" (forms), refers to the ability of a single entity, such as a function, method, or object, to take on multiple forms or behaviors.

- In object-oriented programming (OOP), polymorphism allows objects of different classes to respond to the same method call in their own specific ways
- There two ways to perform this:
  Method overloading and Method overriding

8. How is encapsulation achieved in Python?
   - Encapsulation in Python is achieved by restricting access to attributes and methods, effectively bundling data and the methods that operate on it within a class.
   - Python uses access modifiers, primarily single and double underscores, to control the visibility of class members.
   - Access modifiers:
     - Public: No prefix underscore. Accessible anywhere within the code.
     - Private: Double underscore prefix. Accessible only within the class.
     - Protected: Single underscore prefix. Accessible within the class or subclass.

9. What is a constructor in Python?
   - A constructor in Python is a special method used to initialize objects of a class.
   - It is automatically called when an object is created.
   - The constructor is defined using the __init__ method within the class.
   - Its primary purpose is to set up the initial state of the object by assigning values to its attributes or performing any necessary setup operations

10. What are class and static methods in Python?
    Class methods and static methods are special types of methods in Python classes that differ in how they are bound to the class and how they receive arguments. \
    Class methods:
    - Are bound to the class and not the instance of the class.
    - Take the class itself as the first argument, conventionally named cls.
    - Can access and modify class-level attributes.
    - Are defined using the @classmethod decorator.

    Static methods:

    - Are not bound to the class or the instance
    - Do not take any special first argument.
    - Cannot access or modify class-level or instance-level attributes
    - Are defined using the @staticmethod decorator

11. What is method overloading in Python?
    - Method overloading in Python refers to the ability to define multiple methods with the same name within a class, but with different parameters.
    - Python does not directly support method overloading in the same way as some other languages like Java or C++.
    - This allows a single method to handle different numbers or types of arguments.

12. What is method overriding in OOP?
    - Method overriding in Python is a feature of object-oriented programming where a subclass provides a specific implementation for a method that is already defined in its superclass.
    - When a method in a subclass has the same name, parameters, and return type as a method in its superclass, the method in the subclass overrides the method in the superclass.
    - This allows objects of the subclass to use the overridden method instead of the superclass's method.

13. What is a property decorator in Python?
    - In Python, a property decorator is a built-in feature that allows methods to be accessed like attributes, providing a way to implement getter, setter, and deleter methods for class attributes.

- It offers controlled access to attributes, encapsulating logic for retrieving, setting, or deleting them.
- This mechanism helps in managing the internal state of objects while presenting a clean and intuitive interface

14. Why is polymorphism important in OOP?

    Polymorphism is important in OOP because it allows different objects to be treated uniformly, enabling flexible and reusable code by letting the same interface work with different data types.

15. What is an abstract class in Python?
    - An **abstract class** in Python is a class that cannot be instantiated on its own and is meant to be a blueprint for other classes.
    - It can define abstract methods that must be implemented by any subclass.
    - Abstract classes are created using the abc module.
    - Example:

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def speak(self):
        pass
class Dog(Animal):
    def speak(self):
        return "Woof!"
# animal = Animal()  # Error: Can't instantiate abstract class
dog = Dog()
print(dog.speak())  # Output: Woof!
```

16. What are the advantages of OOP?
    - Modularity: Code is organized into objects and classes, making it easier to manage and understand.
    - Reusability: Classes and objects can be reused across programs using inheritance.
    - Encapsulation: Protects data by keeping it private and exposing only necessary parts.
    - Polymorphism: Allows one interface to be used for different data types, enabling flexible and scalable code.
    - Inheritance: Promotes code reuse by allowing new classes to inherit properties and methods from existing ones.
    - Maintainability: Easier to update and maintain code due to its organized structure.
    - Abstraction: Hides complex implementation details and shows only the necessary features.

17. What is the difference between a class variable and an instance variable?
    - Class variable: Shared by all instances of a class. Defined inside the class but outside any method.
    - Instance variable: Unique to each object. Defined inside methods using self
    - Example:

```
class MyClass:
    class_var = 10  # Class variable

    def __init__(self, value):
        self.instance_var = value  # Instance variable
```

18. What is multiple inheritance in Python?
    - Multiple inheritance in Python means a class can inherit from more than one parent class.
    - This allows a child class to access attributes and methods from multiple classes
    - Example:

```
class A:
    def method_a(self):
```

```
        print("Method from class A")

class B:
    def method_b(self):
        print("Method from class B")

class C(A, B):
    pass

obj = C()
obj.method_a()  # Output: Method from class A
obj.method_b()  # Output: Method from class B
```

19. Explain the purpose of ‘’ __str__’ and ‘__repr__’ ‘ methods in Python?
    The __str__ and __repr__ methods in Python are special methods used to define how objects are represented as strings. They serve different purposes and are used in different contexts.
    __str__(self):
    - This method returns a user-friendly string representation of the object.
    - It is called by the built-in str() function and implicitly when using the print() function.
    - The goal of __str__ is to provide a human-readable output, focusing on clarity and ease of understanding.
    - If __str__ is not defined, then memory location is printed

    __repr__(self):
    - This method returns a more detailed, unambiguous string representation of the object.
    - It is called by the built-in repr() function and in the interactive interpreter when an object is evaluated.
    - The goal of __repr__ is to provide a string that, ideally, can be used to recreate the object.

20. What is the significance of the ‘super()’ function in Python?
    - The super() function in Python is used to call methods from a parent class within a subclass.
    - This is particularly useful in the context of inheritance, allowing a subclass to extend or override the behavior of its parent class while still leveraging the parent's functionality.
    - It promotes code reusability and maintainability by avoiding the need to explicitly name the parent class

21. What is the significance of the __del__ method in Python?
    - The __del__ method in Python is a destructor. It is called automatically when an object is about to be destroyed, typically when there are no more references to it.
    - Significance:
        - Used to clean up resources (e.g., close files or database connections).
        - Called just before the object is garbage collected.

22. What is the difference between @staticmethod and @classmethod in Python?
    @staticmethod:
    - This decorator defines a method that doesn't receive any implicit argument (neither the instance self nor the class cls).
    - It behaves like a regular function but is logically associated with the class.
    - It can't access or modify the class or instance state directly.

    @classmethod:
    - This decorator defines a method that receives the class itself as the first argument, conventionally named cls.
    - It can access and modify the class state but not the instance state.

23. How does polymorphism work in Python with inheritance?
    - In Python, polymorphism with inheritance allows child classes to override methods from a parent class, and ensures that the correct method is called based on the object's actual type—even when using a reference to the parent class.
    - How it works:
        - A base class defines a method.
        - Derived (child) classes override that method.
        - You can call the method on objects of the child classes using a reference to the base class.

    - Example:

```
class Animal:
    def speak(self):
        return "Some sound"

class Dog(Animal):
    def speak(self):
        return "Woof!"

class Cat(Animal):
    def speak(self):
        return "Meow!"

def make_sound(animal):
    print(animal.speak())

# Polymorphism in action
animals = [Dog(), Cat()]
for a in animals:
    make_sound(a)
```

24. What is method chaining in Python OOP?
    - **Method chaining** in Python OOP is a technique where multiple methods are called **sequentially on the same object** in a single line.
    - This is possible when each method **returns self** (the object itself)
    - Example:

```
class Person:
    def __init__(self, name):
        self.name = name
        self.age = None
        self.city = None

    def set_age(self, age):
        self.age = age
        return self  # Returning self enables chaining

    def set_city(self, city):
        self.city = city
        return self

    def show(self):
        print(f"{self.name}, {self.age}, {self.city}")
        return self

# Method chaining
p = Person("Alice").set_age(30).set_city("New York").show()
```

25. What is the purpose of the __call__ method in Python?
    - The __call__ method in Python allows an object of a class to be called like a function.
    - Purpose:It makes an instance callable, meaning you can use object() syntax as if the object were a function.
    - Example:

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

    def __call__(self, greeting):
        return f"{greeting}, {self.name}!"

greet = Greeter("Alice")
print(greet("Hello"))  # Output: Hello, Alice!
```