```
#Name  -  Shivam kashyap
#Course Name - Data analtyics
#Assignment Name - OOPS assignment
#Github link - https://github.com/mrkashyap222/shivam-kashyap

#Thoery Questions

#1. What is Object-Oriented Programming (OOP)?
# Answer- OOP is a way of programming where we create objects that
contain both data (variables) and actions (methods)

#2. What is a class in OOP?
# Answer- A class is like a blueprint used to create objects

#3. What is an object in OOP?
# Answer- An object is something created from a class. Example: Car is
a class, a red car is an object

#4. What is the difference between abstraction and encapsulation?
#Answer- Abstraction hides details, shows only what is needed.
Encapsulation binds data and methods together

#5. What are dunder methods in Python?
#answer- Dunder methods are special methods with double underscores
like __init__, __str__

#6. Explain the concept of inheritance in OOP
#Answer- Inheritance means a class can use features of another class

#7. What is polymorphism in OOP?
#answer- Polymorphism means one function can have different forms.
Example: + can add numbers or join strings

#8. How is encapsulation achieved in Python?
#Answer- Encapsulation is done using classes and making variables
private

#9. What is a constructor in Python?
#answer- A constructor is a special method (__init__) that runs when
an object is created

#10. What are class and static methods in Python?
#Answer- Class method uses @classmethod and works with class. Static
method uses @staticmethod and works without class or object

#11. What is method overloading in Python
#answer- Python does not support true overloading, but we can use
default arguments to act like overloading

#12. What is method overriding in OOP
#answer- When a child class gives its own version of a method from the
parent class
```

```python
#13. What is a property decorator in Python
#Answer- @property is used to make a method act like a variable

#14. Why is polymorphism important in OOP
#Answer- It makes code flexible and reusable

#15. What is an abstract class in Python
#answer- A class that cannot be used directly, only used for other
classes to inherit

#16. What are the advantages of OOP
#answer- Code reuse, easy to understand, easy to maintain

#17. What is the difference between a class variable and an instance
variable
#answer- Class variable is shared by all objects. Instance variable is
different for each object

#18. What is multiple inheritance in Python
#Answer- When a class can inherit features from more than one class

#19. Explain the purpose of '__str__' and '__repr__' methods in Python
#answer- __str__ gives a user-friendly string, __repr__ gives a
developer-friendly string

#20. What is the significance of the 'super()' function in Python
#answer- super() is used to call parent class methods in child class

#21. What is the significance of the '__del__' method in Python
#answer- __del__ is a destructor, it runs when an object is deleted

#22. What is the difference between @staticmethod and @classmethod in
Python
#answer- @staticmethod does not need class or object, @classmethod
needs the class but not the object

#23. How does polymorphism work in Python with inheritance
#Answer- It allows child classes to have different implementations of
parent class methods

#24. What is method chaining in Python OOP?
#Answer- When methods return the object itself so we can call multiple
methods one after another

#Practical Questions

#1. Parent and Child class with method overriding

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

```python
class Dog(Animal):
    def speak(self):
        print("Bark")

d = Dog()
d.speak()
```

```
Bark
```

```python
#2. Abstract class Shape

from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, r):
        self.r = r
    def area(self):
        return 3.14 * self.r * self.r

class Rectangle(Shape):
    def __init__(self, l, w):
        self.l = l
        self.w = w
    def area(self):
        return self.l * self.w

c = Circle(5)
r = Rectangle(4, 6)
print(c.area())
print(r.area())
```

```
78.5
24
```

```python
#3. Multi-level inheritance

class Vehicle:
    def __init__(self, type):
        self.type = type

class Car(Vehicle):
    def __init__(self, type, brand):
        super().__init__(type)
        self.brand = brand

class ElectricCar(Car):
```

```
    def __init__(self, type, brand, battery):
        super().__init__(type, brand)
        self.battery = battery

e = ElectricCar("Car", "Tesla", "100 kWh")
print(e.type, e.brand, e.battery)
```

```
Car Tesla 100 kWh
```

```
#4. Polymorphism with Bird

class Bird:
    def fly(self):
        print("Bird is flying")

class Sparrow(Bird):
    def fly(self):
        print("Sparrow flies fast")

class Penguin(Bird):
    def fly(self):
        print("Penguins cannot fly")

b = [Bird(), Sparrow(), Penguin()]
for x in b:
    x.fly()
```

```
Bird is flying
Sparrow flies fast
Penguins cannot fly
```

```
#5. Encapsulation with BankAccount
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount

    def check_balance(self):
        return self.__balance

acc = BankAccount(1000)
acc.deposit(500)
acc.withdraw(200)
print(acc.check_balance())
```

```
1300
```

```python
#6. Runtime Polymorphism (Instrument)
class Instrument:
    def play(self):
        print("Playing instrument")

class Guitar(Instrument):
    def play(self):
        print("Playing guitar")

class Piano(Instrument):
    def play(self):
        print("Playing piano")

instruments = [Guitar(), Piano()]
for i in instruments:
    i.play()
```

```
Playing guitar
Playing piano
```

```python
#7. Class and Static methods
class MathOperations:
    @classmethod
    def add_numbers(cls, a, b):
        return a + b

    @staticmethod
    def subtract_numbers(a, b):
        return a - b

print(MathOperations.add_numbers(10, 5))
print(MathOperations.subtract_numbers(10, 5))
```

```
15
5
```

```python
#8. Count total persons created

class Person:
    count = 0
    def __init__(self):
        Person.count += 1

p1 = Person()
p2 = Person()
print(Person.count)
```

```
2
```

```python
#9. Fraction class

class Fraction:
    def __init__(self, n, d):
        self.n = n
        self.d = d

    def __str__(self):
        return f"{self.n}/{self.d}"

f = Fraction(3, 4)
print(f)
```

```
3/4
```

```python
#10. Operator overloading with Vector

class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)

    def __str__(self):
        return f"({self.x}, {self.y})"

v1 = Vector(2, 3)
v2 = Vector(4, 5)
print(v1 + v2)
```

```
(6, 8)
```

```python
#11. Class Person with greet()

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

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age}
years old.")

p = Person("Ajay", 20)
p.greet()
```

```
Hello, my name is Ajay and I am 20 years old.
```

```python
#12. Class Student with average grade
```

```python
class Student:
    def __init__(self, name, grades):
        self.name = name
        self.grades = grades

    def average_grade(self):
        return sum(self.grades) / len(self.grades)

s = Student("Ravi", [80, 90, 70])
print(s.average_grade())
```

```
80.0
```

```python
#13. Rectangle with set_dimensions() and area()

class Rectangle:
    def set_dimensions(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

r = Rectangle()
r.set_dimensions(5, 4)
print(r.area())
```

```
20
```

```python
#14. Employee and Manager (Inheritance)

class Employee:
    def __init__(self, hours, rate):
        self.hours = hours
        self.rate = rate

    def calculate_salary(self):
        return self.hours * self.rate

class Manager(Employee):
    def __init__(self, hours, rate, bonus):
        super().__init__(hours, rate)
        self.bonus = bonus

    def calculate_salary(self):
        return super().calculate_salary() + self.bonus

e = Employee(40, 20)
m = Manager(40, 20, 500)
```

```
print(e.calculate_salary())
print(m.calculate_salary())
```

```
800
1300
```

```
#15. Class Product

class Product:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

    def total_price(self):
        return self.price * self.quantity

# Example:
p = Product("Laptop", 1000, 3)
print(p.total_price())
```

```
3000
```

```
#16. Abstract Class Animal and Derived Classes

from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def sound(self):
        pass

class Cow(Animal):
    def sound(self):
        return "Moo"

class Sheep(Animal):
    def sound(self):
        return "Baa"

(cow.sound())
```

```
{"type":"string"}
```

```
(sheep.sound())
```

```
{"type":"string"}
```

```
#17. Class Book

class Book:
    def __init__(self, title, author, year_published):
```

```
        self.title = title
        self.author = author
        self.year_published = year_published

    def get_book_info(self):
        return f"{self.title} by {self.author}, published in
{self.year_published}"
b = Book("1984", "George Orwell", 1949)

print(b.get_book_info())
```

```
1984 by George Orwell, published in 1949
```

```
#18. Class House and Derived Class Mansion


class House:
    def __init__(self, address, price):
        self.address = address
        self.price = price

class Mansion(House):
    def __init__(self, address, price, number_of_rooms):
        super().__init__(address, price)
        self.number_of_rooms = number_of_rooms
m = Mansion("123 Palm Street", 5000000, 15)

print(m.address, m.price, m.number_of_rooms)
```

```
123 Palm Street 5000000 15
```