PYTHONS OOPS QUESTION

Q.1/ What is Object-Oriented Programming (OOP) ?
ANS:- object-oriented programm is a special style of programming that organize code using objects instead of functions. Objects combine data and behavior into a single unit . This makes programs more modular and easier to manage.

Q.2/ What is a class in OOP ?
ANS:- A class is a blueprint or a template for creating an object . It defines variables and functions that describe the behavior of an object.

Q.3/ What is an object in OOP ?
ANS:- An object is a self-contained unit in oop class . An object has its own state and behavior . and its behaviors defined by class .

Q.4/ What is the difference between abstraction and encapsulation ?
ANS:- The main differnce between Abstraction and encapsulatio is that Abstraction can only show important details and hides the backgroud work but encapsulation keeps the data and methods together in a class and restricting direct access.

Q.5/ What are dunder methods in Python ?
ANS:- Dunder methods are special predefined method in python .They allow objects to behave like built-in types and support operator overloading.

Q.6/ Explain the concept of inheritance in OOP ?
ANS:- Inheritance allows a child class to acquire properties and methods from parent class .It helps in code reusability because we don’t need to write the same code again .

Q.7/What is polymorphism in OOP ?
ANS:-Polymorphism allows objects of different classes to be treated as objects of a common superclass. This means a single function, method, or operator can work with different types of objects, and the behavior will vary depending on the specific object being used.

Q.8/ How is encapsulation achieved in Python ?
ANS:- Encapsulation in python can simpely be achived by hiding data inside a class and giving controlled access. its keeps the data safe and prevents accidental changes from outside the class.

Q.9/ What is a constructor in Python ?
ANS:- A constructor is a special method in python . It is automatically called when an object is created from a class and Every class can have only one constructor, but it can take different arguments.

Q.10/ What are class and static methods in Python ?
ANS:- A class method is defined with @classmethod and takes cls as the first parameter. It can access or modify class variables. A static method is defined with @staticmethod and does not take self or cls. It behaves like a normal function inside a class.

Q.11/ What is method overloading in Python ?
ANS:- Method overloading is when we have the multiple values with same name but different parameters In many languages, but Python does not support overloading method directly .In Python we achieve overloading using default arguments or kwywords .

Q.12/ What is method overriding in OOP ?
ANS:- Method overriding happens when a child class defines a method with the same name as a parent class method.In this case, the child class method replaces the parent method.

Q.13/ What is a property decorator in Python ?
ANS:- property decorator is used to define getter methods that can be accessed like attributes without parentheses .It allows controlled access to private attributes .

Q.14/ Why is polymorphism important in OOP ?
ANS:- Polymorphism is really important in oop because its Makes code more flexible,Increases reusability and Supports abstraction by letting us write general code for different object types.

Q.15/ What is an abstract class in Python ?
ANS:- An abstract class is a blueprint for other classes, containing one or more abstract methods. It defines abstract methods that must be

implemented by child classes.

Q.16/ What are the advantages of OOP ?
ANS:- The advantage of oop is that its make the code reusable and it seceure data handeling in classes. oops makes the code easy to update .

Q.17/ What is the difference between a class variable and an instance variable ?
ANS:- The main difference between a class variable and an instance variable that A class variable is shared by all instance of a class and An instance variable is a variable that is unique to each instance (object) of a class .

Q.18/ What is multiple inheritance in Python ?
ANS:- If A class can inherit from more than one parent class is called inheritance in python .

Q.19/ Explain the purpose of ‘’**str**’ and ‘**repr**’ ‘ methods in Python ?
ANS:- super() is used to call parent class methods. Especially useful in inheritance to avoid rewriting code.

Q.20/ What is the significance of the ‘super()’ function in Python ?
ANS:- super() is used to call parent class methods.Especially useful in inheritance to avoid rewriting code .

Q.21/ What is the significance of the **del** method in Python ?
ANS:- The **del** method in Python is a special method, known as a destructor, that is called when an object is about to be garbage collected or destroyed. It is primarily used to perform cleanup operations before an object is removed from memory, such as closing file handles .

Q.22/ What is the difference between @staticmethod and @classmethod in Python ?
ANS :- The main difference between staticmethod and classmethod is that staticmethod Does not take self or cls as the first parameter.Cannot access instance attributes or class attributes.Works like a normal function, just grouped inside a class but in classmethod Takes cls as the first parameter Can access and modify class variables.Used when you want to work with class-level data.

Q.23/ How does polymorphism work in Python with inheritance ?
ANS:- Polymorphism in Python, particularly with inheritance, allows objects of different classes to be treated as objects of a common superclass.

Q.24/ What is method chaining in Python OOP ?
ANS:- Method chaining in Python OOP is when methods return self, allowing multiple method calls on the same object in a single statement.

Q.25/ What is the purpose of the **call** method in Python ?
ANS:- The **call** method makes a class instance callable like a function, allowing you to use obj() syntax.

**PRACTICE QUESTION*

Q.1/ Create a parent class Animal with a method speak() that prints a generic message. Create a child class Dog that overrides the speak() method to print "Bark!"?

```
class Animal:
  def speak(self):
    print("please dont hurt us")

class Dog(Animal):
  def speak(self):
    print("Bark!")
a = Animal()
a.speak()
d = Dog()
d.speak()
```

```
please dont hurt us
Bark!
```

Q.2/ Write a program to create an abstract class Shape with a method area(). Derive classes Circle and Rectangle from it and implement the area() method in both ?

```
from abc import ABC, abstractmethod
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * (self.radius ** 2)
class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
c = Circle(5)
r = Rectangle(4, 6)
print("Circle Area:", c.area())
print("Rectangle Area:", r.area())
```

```
Circle Area: 78.5
```

Q.3/ Implement a multi-level inheritance scenario where a class Vehicle has an attribute type. Derive a class Car and further derive a class ElectricCar that adds a battery attribute ?

```
class Vehicle:
    def __init__(self, vehicle_type):
        self.type = vehicle_type
class Car(Vehicle):
    def __init__(self, vehicle_type, brand):
        super().__init__(vehicle_type)
        self.brand = brand
class ElectricCar(Car):
    def __init__(self, vehicle_type, brand, battery_capacity):
        super().__init__(vehicle_type, brand)
        self.battery = battery_capacity
v = Vehicle("General Vehicle")
print("Vehicle Type:", v.type)
c = Car("Car", "Toyota")
print("Car Type:", c.type, "| Brand:", c.brand)
e = ElectricCar("Car", "Tesla", "85 kWh")
print("Electric Car Type:", e.type, "| Brand:", e.brand, "| Battery:", e.battery)
```

```
This is a General Vehicle
This is a Toyota Car
This is a Tesla Car with 85 kWh battery
```

Q.4/ Demonstrate polymorphism by creating a base class Bird with a method fly(). Create two derived classes Sparrow and Penguin that override the fly() method ?

```
class Bird:
    def fly(self):
        print("Birds can usually fly")
class Sparrow(Bird):
    def fly(self):
        print("Sparrow flies high in the sky")
class Penguin(Bird):
    def fly(self):
        print("Penguins cannot fly, they swim instead")
birds = [Sparrow(), Penguin()]
for b in birds:
    b.fly()
```

```
Sparrow flies high in the sky
```

Q.5/ Write a program to demonstrate encapsulation by creating a class BankAccount with private attributes balance and methods to deposit, withdraw, and check balance ?

```
class BankAccount:
    def __init__(self, balance=0):
        self.__balance = balance
    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Deposited: {amount}")
        else:
            print("Invalid deposit amount!")
    def withdraw(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount
            print(f"Withdrawn: {amount}")
        else:
            print("Insufficient balance or invalid amount!")
    def check_balance(self):
        print(f"Current Balance: {self.__balance}")
account = BankAccount(1000)
account.deposit(500)
account.withdraw(300)
account.check_balance()
```

```
Deposited: 500
Withdrawn: 300
Current Balance: 1200
```

Q.6/ Demonstrate runtime polymorphism using a method play() in a base class Instrument. Derive classes Guitar and Piano that implement their own version of play() ?

```
class Instrument :
       def play(self):
        print(" i creat music")
class Guitar(Instrument):
    def play(self):
        print(" guiter : i make background music")
class  Piano(Instrument):
    def play(self):
        print("Piano : i make movies more dramatic")
music=[Guitar(),Piano()]
for instrument in music:
    instrument.play()
```

```
 guiter : i make background music
Piano : i make movies more dramatic
```

Q.7/ Create a class MathOperations with a class method add_numbers() to add two numbers and a static method subtract_numbers() to subtract two numbers ?

```
class MathOperations:
    @classmethod
    def add_numbers(self, a, b):
        return a + b
    @staticmethod
    def subtract_numbers(a, b):
        return a - b
print("Addition:", MathOperations.add_numbers(10, 5))
print("Subtraction:", MathOperations.subtract_numbers(10, 5))
```

```
Addition: 15
Subtraction: 5
```

Q.8/ Implement a class Person with a class method to count the total number of persons created ?

```
class Person:
    count = 0
    def __init__(self, name):
        self.name = name
        Person.count += 1
    @classmethod
    def total_persons(cls):
        return cls.count
p1 = Person("Soumya")
p2 = Person("Rahul")
p3 = Person("Ananya")
print("Total persons created:", Person.total_persons())
```

```
Total persons created: 3
```

Q.9/ Write a class Fraction with attributes numerator and denominator. Override the str method to display the fraction as "numerator/denominator" ?

```
class Fraction:
    def __init__(self, numerator, denominator):
        self.numerator = numerator
        self.denominator = denominator
    def __str__(self):
        return f"{self.numerator}/{self.denominator}"
f1 = Fraction(3, 4)
f2 = Fraction(5, 7)
print(f1)
print(f2)
```

```
3/4
5/7
```

Q.10/ Demonstrate operator overloading by creating a class Vector and overriding the add method to add two vectors ?

```
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"Vector({self.x}, {self.y})"
v1 = Vector(2, 4)
v2 = Vector(5, 7)
v3 = v1 + v2
print("v1 =", v1)
print("v2 =", v2)
print("v1 + v2 =", v3)
```

```
v1 = Vector(2, 4)
v2 = Vector(5, 7)
v1 + v2 = Vector(7, 11)
```

Q.11/ Create a class Person with attributes name and age. Add a method greet() that prints "Hello, my name is {name} and I am {age} years old ?

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def greet(self):
        return f"Hello, my name is {self.name} and I am {self.age} years old."
p1 = Person("Soumya", 20)
print(p1.greet())
```

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

Q.12/ Implement a class Student with attributes name and grades. Create a method average_grade() to compute the average of the grades ?

```
class Student:
    def __init__(self, name, grades):
        self.name = name
        self.grades = grades
    def average_grade(self):
        if len(self.grades) == 0:
            return 0
        return sum(self.grades) / len(self.grades)
s1 = Student("Soumya", [85, 90, 78, 92])
s2 = Student("Rahul", [70, 75, 80])
print(f"{s1.name}'s average grade:", s1.average_grade())
print(f"{s2.name}'s average grade:", s2.average_grade())
```

```
Soumya's average grade: 86.25
Rahul's average grade: 75.0
```

Q.13/ Create a class Rectangle with methods set_dimensions() to set the dimensions and area() to calculate the area ?

```
class Rectangle:
    def __init__(self):
        self.length = 0
        self.width = 0
    def set_dimensions(self, length, width):
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
l = int(input("Enter the length of rectangle: "))
w = int(input("Enter the width of rectangle: "))
rect = Rectangle()
```

```
rect.set_dimensions(l, w)
print("Area of rectangle:", rect.area())
```

Q.14/Create a class Employee with a method calculate_salary() that computes the salary based on hours worked and hourly rate. Create a derived class Manager that adds a bonus to the salary ?

```
from abc import ABC, abstractmethod
class Employee(ABC):
    def __init__(self, name, hours_worked, hourly_rate):
        self.name = name
        self.hours_worked = hours_worked
        self.hourly_rate = hourly_rate
    @abstractmethod
    def calculate_salary(self):
        pass
class Manager(Employee):
    def __init__(self, name, hours_worked, hourly_rate, bonus):
        super().__init__(name, hours_worked, hourly_rate)
        self.bonus = bonus

    def calculate_salary(self):
        base_salary = self.hours_worked * self.hourly_rate
        return base_salary + self.bonus
m = Manager("Soumya", 160, 100, 5000)
print(f"{m.name}'s Salary: {m.calculate_salary()}")
```

```
Soumya's Salary: 21000
```

Q.15/ Create a class Product with attributes name, price, and quantity. Implement a method total_price() that calculates the total price of the 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
p1 = Product("Laptop", 50000, 2)
p2 = Product("Headphones", 1500, 3)

print(f"{p1.name} total price: ₹{p1.total_price()}")
print(f"{p2.name} total price: ₹{p2.total_price()}")
```

```
Laptop total price: ₹100000
Headphones total price: ₹4500
```

Q.16/ Create a class Animal with an abstract method sound(). Create two derived classes Cow and Sheep that implement the sound() method ?

```
from abc import ABC, abstractmethod
class Animal(ABC):
    @abstractmethod
    def sound(self):
        pass
class Cow(Animal):
    def sound(self):
        return "Moo Moo"
class Sheep(Animal):
    def sound(self):
        return "Baa Baa"
cow = Cow()
sheep = Sheep()
print("Cow Sound:", cow.sound())
print("Sheep Sound:", sheep.sound())
```

```
Cow Sound: Moo Moo
Sheep Sound: Baa Baa
```

Q.17/ Create a class Book with attributes title, author, and year_published. Add a method get_book_info() that returns a formatted string with the book's details ?

```
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"Title: {self.title}\nAuthor: {self.author}\nYear Published: {self.year_published}"
b1 = Book("The Alchemist", "Paulo Coelho", 1988)
b2 = Book("1984", "George Orwell", 1949)
print(b1.get_book_info())
print("--------------")
print(b2.get_book_info())
```

```
Title: The Alchemist
Author: Paulo Coelho
Year Published: 1988
--------------
Title: 1984
Author: George Orwell
Year Published: 1949
```

Q.18/ Create a class House with attributes address and price. Create a derived class Mansion that adds an attribute number_of_rooms ?

```
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
m1 = Mansion("Park Street, Kolkata", 50000000, 12)
print("Address:", m1.address)
print("Price:", m1.price)
print("Number of Rooms:", m1.number_of_rooms)
```

```
Address: Park Street, Kolkata
Price: 50000000
Number of Rooms: 12
```