Batch: A3 **Experiment Number:2**

Roll Number:16010423099 **Name: Suryanshu Banerjee**

Aim of the Experiment: Write a program to create StudentInfo class .Calculate the percentage scored by the student

---

Program/ Steps:

1. For given program find output

Sr.No	Program	Output

1	class Rectangle:
	    def __init__(self, length, width):
	        self.length = length
	        self.width = width
	
	    def area(self):
	        return self.length * self.width
	
	    def perimeter(self):
	        return 2 * self.length + 2 * self.width
	
	class Square(Rectangle):
	    def __init__(self, length):
	        super().__init__(length, length)
	
	square = Square(4)
	print(square.area())
		16

2	class Person:
	  def __init__(self, fname, lname):
	    self.firstname = fname
	    self.lastname = lname
	
	  def printname(self):
	    print(self.firstname, self.lastname)
	
	class Student(Person):
	  def __init__(self, fname, lname, year):
	    super().__init__(fname, lname)
	    self.graduationyear = year
	
	x = Student("Wilbert", "Galitz", 2018)
	print(x.graduationyear)
		2018

3	class Bank:
	    def getroi(self):
	        return 10
	
	class SBI:
	        def getroi(self):
	            return 7
	
	class ICICI:
	   def getroi(self):
	        return 8
	
	b1=Bank()
	b2=SBI()
	b3=ICICI()
	print("Bank rate of interest:",b1.getroi())
	print("SBI rate of interest:",b2.getroi())
	print("ICICI rate of interest:",b3.getroi())
		Bank rate of interest: 10
		SBI rate of interest: 7
		ICICI rate of interest: 8

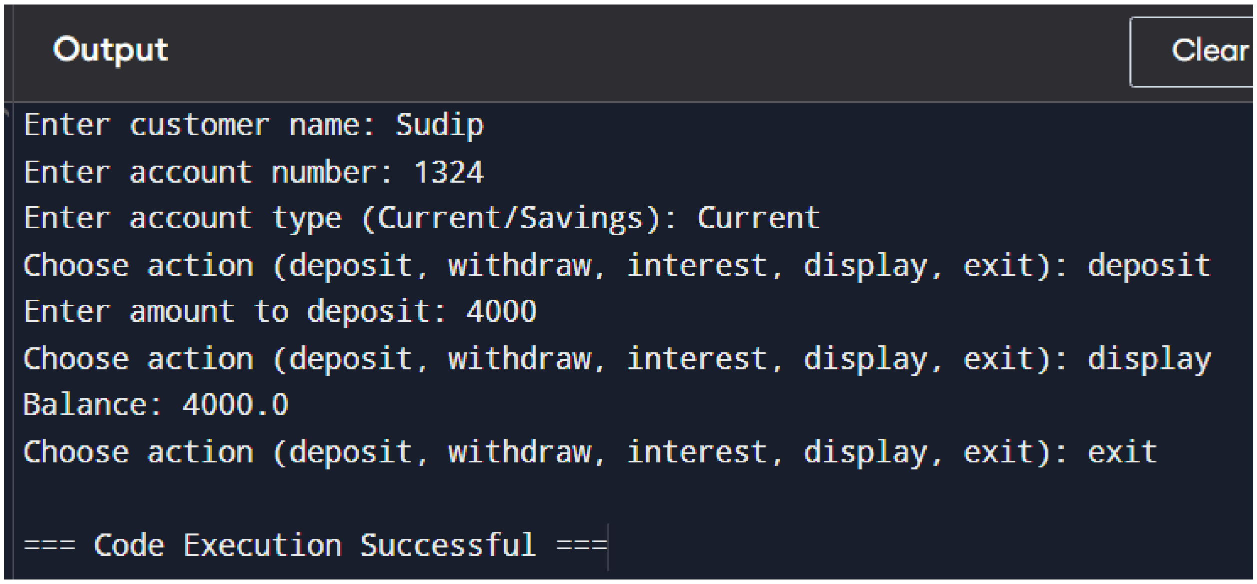
2. Create a class account that stores customer name, account number and type of account. From this derive the classes cur_acct and sav_acct to make them more specific to their requirements. Include necessary member functions in order to achieve the following tasks:
- Accept deposit from a customer and update the balance.
- Display the balance.
- Compute and deposit interest.
- Permit withdrawal and update the balance.
- Check for the minimum balance, impose penalty, necessary and update the balance.

```
class Account:
    def __init__(self, name, acc_number, acc_type):
        self.name = name
        self.acc_number = acc_number
        self.acc_type = acc_type
        self.balance = 0

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

    def display_balance(self):
        print(f"Balance: {self.balance}")

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
        else:
            print("Insufficient funds")

class CurAcct(Account):
    pass

class SavAcct(Account):
    def deposit_interest(self, rate):
        self.balance += self.balance * (rate / 100)

    def withdraw(self, amount):
        if self.balance - amount < 1000:
            print("Minimum balance required")
        else:
            self.balance -= amount

name = input("Enter customer name: ")
acc_number = input("Enter account number: ")
acc_type = input("Enter account type (Current/Savings): ")

if acc_type.lower() == "savings":
    account = SavAcct(name, acc_number, acc_type)
else:
    account = CurAcct(name, acc_number, acc_type)

while True:
    action = input("Choose action (deposit, withdraw, interest, display, exit): ").lower()
    if action == "deposit":
        amount = float(input("Enter amount to deposit: "))
        account.deposit(amount)
    elif action == "withdraw":
        amount = float(input("Enter amount to withdraw: "))
        account.withdraw(amount)
    elif action == "interest" and isinstance(account, SavAcct):
        rate = float(input("Enter interest rate: "))
        account.deposit_interest(rate)
    elif action == "display":
        account.display_balance()
    elif action == "exit":
        break
```

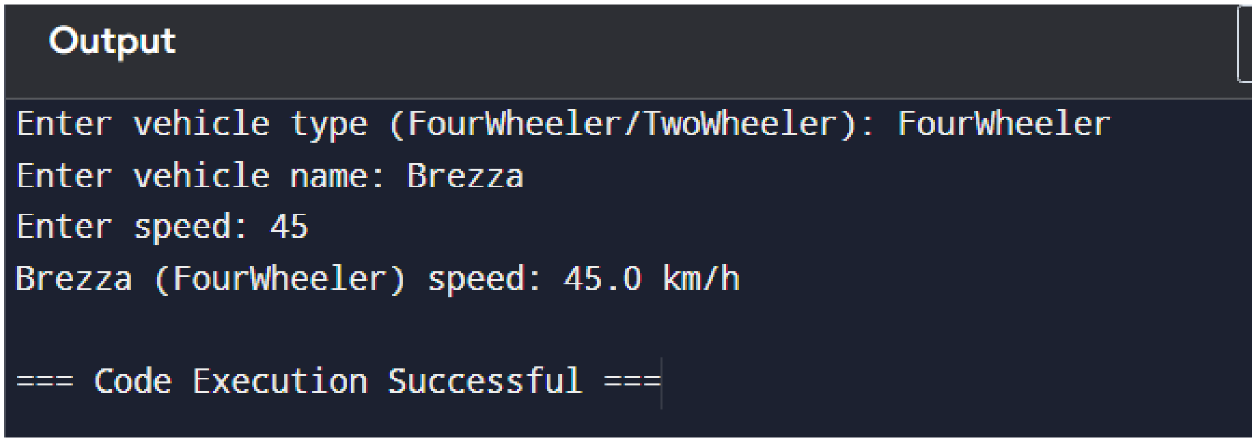
3. Write a program that defines an abstract class called Vehicle containing an abstract method speed (). Derive from it two classes - FourWheeler and TwoWheeler. Create objects of derived classes and call the speed () method using these objects, passing to it the name of vehicle and speed of vehicle. In the speed () method print the vehicle name and the speed of vehicle to which speed () belongs.

```
from abc import ABC, abstractmethod

class Vehicle(ABC):
    @abstractmethod
    def speed(self, name, speed):
        pass

class FourWheeler(Vehicle):
    def speed(self, name, speed):
        print(f"{name} (FourWheeler) speed: {speed} km/h")

class TwoWheeler(Vehicle):
    def speed(self, name, speed):
        print(f"{name} (TwoWheeler) speed: {speed} km/h")

vehicle_type = input("Enter vehicle type (FourWheeler/TwoWheeler): ")
name = input("Enter vehicle name: ")
speed = float(input("Enter speed: "))

if vehicle_type.lower() == "fourwheeler":
    vehicle = FourWheeler()
elif vehicle_type.lower() == "twowheeler":
    vehicle = TwoWheeler()

vehicle.speed(name, speed)
```

Output/Result:



```
Enter customer name: Sudip
Enter account number: 1324
Enter account type (Current/Savings): Current
Choose action (deposit, withdraw, interest, display, exit): deposit
Enter amount to deposit: 4000
Choose action (deposit, withdraw, interest, display, exit): display
Balance: 4000.0
Choose action (deposit, withdraw, interest, display, exit): exit

=== Code Execution Successful ===
```



```
Enter vehicle type (FourWheeler/TwoWheeler): FourWheeler
Enter vehicle name: Brezza
Enter speed: 45
Brezza (FourWheeler) speed: 45.0 km/h

=== Code Execution Successful ===
```

Post Lab Question-Answers:

Explain *isinstance()* and *issubclass()* functions with example

isinstance(object, classinfo): Checks if an object is an instance of a specified class or tuple of classes. Example:

```
class Animal: pass
class Dog(Animal): pass
dog = Dog()
print(isinstance(dog, Dog))  # True
```

issubclass(object, classinfo): Checks if a class is a subclass of another class or tuple of classes. Example:

```
print(issubclass(Dog, Animal))  # True
```

Explain difference between inheritance and abstract class with example

Inheritance allows a subclass to inherit properties and methods from a parent class, such as a "Car" class deriving from a "Vehicle." In contrast, an abstract class cannot be instantiated and often includes abstract methods that must be implemented by subclasses. For example, a "Shape" abstract class might define an area method that subclasses like "Circle" need to implement. This ensures inheritance promotes reuse while abstract classes enforce a specific structure for derived classes.

Outcomes:

CO2: Implement object oriented programming principles such as classes and objects, Inheritance and polymorphism to model real-world entities and relationships.

Conclusion (based on the Results and outcomes achieved):

Successfully applied object oriented programming and executed the program.

References:

Books/ Journals/ Websites referred:

1. Reema Thareja, *Python Programming: Using Problem Solving Approach*, Oxford University Press, First Edition 2017, India
2. Sheetal Taneja and Naveen Kumar, *Python Programming: A modular Approach*, Pearson India, Second Edition 2018,India