FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERIG

Department of Computer Engineering

Experiment 4 - Python Programs to create class, object and methods

1. Course Details:

Academic Year	2023 - 24	Estimated Time	Experiment No. 4 – 02 Hours
Course & Semester	S.E. (COMP) – Sem. IV	Subject Name	Python Programming Lab
Module No.	01	Chapter Title	Python Basics
Experiment Type	Software Performance	Subject Code	CSL405

Name of Student	PRAKASH P. BISWAS	**Roll No.**	9947
Date of Performance.:	16-02-2024	**Date of Submission. :**	29-02-2024
CO Mapping	**CSL405.1: Apply basic concepts of python like control statements, in-built data structures, functions and Object Oriented Paradigms.**		

Timeline (2)	Preparedness (2)	Effort (2)	Result (2)	Post Lab (2)	Total (10)

2. Aim & Objective of Experiment

To implement following programs in Python.

Objective of experiment 4 is to understand the basic concepts of Class and Objects in Python Programming. Students will be able to demonstrate how to create Class and Objects, class and Instance Variables, Class properties in Python.

Pre-Requisite: Any programming language like C, C++

Tools: Python IDE

Python Lab 3 (class, object and methods, Inheritance)

Q.1 Create a Student class and initialize it with name and roll number. Make methods to :

1. Display - It should display all informations of the student.
2. setAge - It should assign age to student
3. setMarks - It should assign marks to the student.

Q. 2 Create a Time class and initialize it with hours and minutes.

1. Make a method addTime which should take two time object and add them. E.g.- (2 hour and 50 min)+(1 hr and 20 min) is (4 hr and 10 min)
2. Make a method displayTime which should print the time.
3. Make a method DisplayMinute which should display the total minutes in the Time. E.g.- (1 hr 2 min) should display 62 minute.

Q.3 Online Shopping Cart System:

Construct a system for managing shopping carts in an online store. Begin with a base class CartItem with attributes like product_id, quantity, and price. Derive classes PromotionalItem and RegularItem from CartItem. Further, derive classes ElectronicItem and ClothingItem representing different product categories. Use Multiple Inheritance to handle both promotional status and product type.

Q. 4 Transportation Company Management

Design a system for managing vehicles in a transportation company. Implement a base class Vehicle with attributes such as make, model, and year. Derive classes Car and Truck from Vehicle, representing different types of vehicles.

Further, derive classes ElectricCar from Car and ElectricTruck from Truck to represent electric vehicles. Implement methods specific to electric vehicles such as charge_battery().

Demonstrate the usage of these classes by creating instances of each subclass and invoking their methods.

Post Lab:

1. Consider the following code:

```
class Clock:

        def __init__(self, time):

                self.time = time

        def print_time(self):

                time = ’6:30’print self.time

clock = Clock(’5:30’)clock.print_time()
```

(a) What does the code print out? If you aren’t sure, create a Python file and run it.

(b) Is that what you expected? Why?

2. Consider the following code:

```
class Clock:
    def __init__(self, time):
        self.time = time
    def print_time(self, time):
        print time
clock = Clock('5:30')
clock.print_time('10:30')
```

(a) What does the code print out? If you aren't sure, create a Python file and run it.

(b) What does this tell you about giving parameters the same name as object attributes?

3. Consider the following code:

```
class Clock:
    def __init__(self, time):
    self.time = time
    def print_time(self):
    print self.time
boston_clock = Clock('5:30')
paris_clock = boston_clock
paris_clock.time = '10:30'
boston_clock.print_time()
```

(a) What does the code print out? If you aren't sure, create a Python file and run it.

(b) Why does it print what it does? (Are boston clock and paris clock different objects? Why or why not?)

In [6]:
```python
class Student:
    def __init__(self, name, roll_no):  #constructor
        self.name = name
        self.roll_no = roll_no


    def display(self):
        print("Student Information:")
        print("Name:", self.name)
        print("Roll Number:", self.roll_no)
        print("Age:", self.age)
        print("Marks:", self.marks)

    def set_age(self, age):
        self.age = age

    def set_marks(self, marks):
        self.marks = marks


student1 = Student("Prakash", "9947")


student1.set_age(20)
student1.set_marks(85)

student1.display()

```

```
Student Information:
Name: Prakash
Roll Number: 9947
Age: 20
Marks: 85
```

In [7]:

```python

class Time:
    def __init__(self, hrs, mins):
        self.hrs= hrs
        self.mins= mins

    def addTime(self, other):
        total_hrs= self.hrs + other.hrs
        total_mins= self.mins + other.mins

        if total_mins >=60:
            total_hrs += total_mins //60    # divide by 60 kiya s
            total_mins %=60   #to fid the left over min

            return Time(total_hrs, total_mins)

    def displayTime(self):
        print(f"Time: {self.hrs} hours and {self.mins} minutes")

    def DisplayMinute(self):
        total_mins = self.hrs * 60 + self.mins
        print(f"Total minutes: {total_mins}")


time1 = Time(10, 50)
time2 = Time(11, 20)

result_time = time1.addTime(time2)

time1.displayTime()
time2.displayTime()
result_time.displayTime()

result_time.DisplayMinute()
```

```
Time: 10 hours and 50 minutes
Time: 11 hours and 20 minutes
Time: 22 hours and 10 minutes
Total minutes: 1330
```

In [8]:
```
class CartItem:
    def __init__(self, prod_id, qty, price):
        self.prod_id = prod_id
        self.qty = qty
        self.price = price


class PromotionalItem(CartItem):
    def __init__(self, prod_id, qty, price, discount_percent):
        super().__init__(prod_id, qty, price)
        self.discount_percent = discount_percent

    def calc_discounted_price(self):
        discount_amount = (self.discount_percent / 100) * self.pr
        discounted_price = self.price - discount_amount
        return discounted_price


class RegularItem(CartItem):
    pass


class ElectronicItem(CartItem):
    def __init__(self, prod_id, qty, price, brand):
        super().__init__(prod_id, qty, price)
        self.brand = brand


class ClothingItem(CartItem):
    def __init__(self, prod_id, qty, price, size):
        super().__init__(prod_id, qty, price)
        self.size = size



regular_item = RegularItem("P123", 2, 25000)

promotional_item = PromotionalItem("P456", 1, 40000, 10)

electronic_item = ElectronicItem("E789", 1, 10000, "XYZ Electron


clothing_item = ClothingItem("C101", 3, 3010, "Large")


print("Regular Item:")
print(f"Product ID: {regular_item.prod_id}")
print(f"Quantity: {regular_item.qty}")
print(f"Price: Rs{regular_item.price}\n")

print("Promotional Item:")
print(f"Product ID: {promotional_item.prod_id}")
print(f"Quantity: {promotional_item.qty}")
print(f"Original Price: Rs{promotional_item.price}")
print(f"Discounted Price: Rs{promotional_item.calc_discounted_pr

print("Electronic Item:")
print(f"Product ID: {electronic_item.prod_id}")
print(f"Quantity: {electronic_item.qty}")
```

```python
print(f"Price: Rs{electronic_item.price}")
print(f"Brand: {electronic_item.brand}\n")

print("Clothing Item:")
print(f"Product ID: {clothing_item.prod_id}")
print(f"Quantity: {clothing_item.qty}")
print(f"Price: Rs{clothing_item.price}")
print(f"Size: {clothing_item.size}")

```

```
Regular Item:
Product ID: P123
Quantity: 2
Price: Rs25000

Promotional Item:
Product ID: P456
Quantity: 1
Original Price: Rs40000
Discounted Price: Rs36000.0

Electronic Item:
Product ID: E789
Quantity: 1
Price: Rs10000
Brand: XYZ Electronics

Clothing Item:
Product ID: C101
Quantity: 3
Price: Rs3010
Size: Large
```

In [12]:
```python
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def display_info(self):
        print(f"{self.year} {self.make} {self.model}")


class Car(Vehicle):
    def __init__(self, make, model, year, doors):
        super().__init__(make, model, year)
        self.doors = doors

    def display_info(self):
        super().display_info()
        print(f"Doors: {self.doors}")


class Truck(Vehicle):
    def __init__(self, make, model, year, payload_capacity):
        super().__init__(make, model, year)
        self.payload_capacity = payload_capacity

    def display_info(self):
        super().display_info()
        print(f"Payload Capacity: {self.payload_capacity} lbs")


class ElectricCar(Car):
    def __init__(self, make, model, year, doors, battery_capacity
        super().__init__(make, model, year, doors)
        self.battery_capacity = battery_capacity

    def charge_battery(self):
        print(f"Charging the battery of the {self.year} {self.mak


class ElectricTruck(Truck):
    def __init__(self, make, model, year, payload_capacity, batte
        super().__init__(make, model, year, payload_capacity)
        self.battery_capacity = battery_capacity

    def charge_battery(self):
        print(f"Charging the battery of the {self.year} {self.mak


car1 = Car("Tata ", "Sumo", 2022, 4)
truck1 = Truck("Ashok", "Lendar", 2022, 2000)
electric_car1 = ElectricCar("Tesla", "Model 3", 2022, 4, 75)
electric_truck1 = ElectricTruck("Mahindra", "Thar", 2022, 3500, 

car1.display_info()
print("\n")
truck1.display_info()
print("\n")
electric_car1.display_info()
electric_car1.charge_battery()
```

```
print("\n")
electric_truck1.display_info()
electric_truck1.charge_battery()

```

```
2022 Tata  Sumo
Doors: 4


2022 Ashok Lendar
Payload Capacity: 2000 lbs


2022 Tesla Model 3
Doors: 4
Charging the battery of the 2022 Tesla Model 3


2022 Mahindra Thar
Payload Capacity: 3500 lbs
Charging the battery of the 2022 Mahindra Thar
```

```
In [ ]:
```

POSTLAB

1) (a) The code initializes a clock object with the time '5:30' and then calls the print_time method. Inside the print_time method, there is a local variable time set to '6:30', but it is not used. Instead, it prints the instance variable self.time, which is set to '5:30' during the object initialization.
when the code is executed, it will print '5:30'.

(b) yes, this is expected because the print_time method prints the value of the self.time instance variable, which was set to '5:30' when the clock object was created. The local variable time = '6:30' inside the method does not affect the printing of self.time.

2) (a) The code initializes a clock object with the time '5:30' and then calls the print_time method with the argument '10:30'. Inside the print_time method, there is a parameter time, and it prints the value of this parameter.
when the code is executed, it will print '10:30'.

(b) This example highlights the concept of shadowing. Shadowing occurs when a parameter has the same name as an attribute of the class. In this case, the parameter time in the print_time method shadows the instance variable self.time. when you use print(time) inside the method, it refers to the local parameter time instead of the instance variable.

It is generally a good practice to avoid shadowing to prevent confusion and make the code more readable. If you need to differentiate between the instance variable and the parameter, you can use a different name for the parameter or use self.time to explicitly reference the instance variable.

3) (a) The code creates a Clock object named boston_clock with the time '5:30'. Then, it creates another reference paris_clock pointing to the same object as boston_clock. The time of paris_clock is then set to '10:30'. Finally, the print_time method of boston_clock is called.
When the code is executed, it will print '10:30'.

(b) The reason it prints '10:30' is because paris_clock is not a separate object; it is just another reference to the same object as boston_clock. When you modify the time attribute through paris_clock, it affects the underlying object, and any reference to that object (including boston_clock) will reflect the change.

In Python, assignments like paris_clock = boston_clock do not create a new object but rather create another reference to the same object. Therefore, both boston_clock and paris_clock refer to the same Clock object, and changes made through one reference are visible through the other.