SIKSHA 'O' ANUSANDHAN
DEEMED TO BE UNIVERSITY

Admission Batch: **Session:**

Laboratory Record

Python for Computer Science and Data Science 2 (CSE 3652)

Submitted by

Name: _____

Registration No.:_____

Branch:_____

Semester: _____ Section:_____

Department of Computer Science & Engineering

Faculty of Engineering & Technology (ITER)

Jagamohan Nagar, Jagamara, Bhubaneswar, Odisha - 751030

INDEX

Sl. No	Name of Program	Page No	Remarks

ASSIGNMENT-1

Q1) What is the significance of classes in Python programming, and how do they contribute to object oriented programming?

Ans) Classes in Python are fundamental to object-oriented programming (OOP) as they provide a structured way to model real-world entities. They serve as blueprints for creating objects, encapsulating data (attributes) and behaviour (methods) into a single unit.

Contribution to Object-Oriented Programming (OOP)

- Encapsulation – Bundles data and methods together, restricting direct access to some variables and ensuring data integrity.
- Inheritance – Allows one class (child) to inherit attributes and methods from another class (parent), promoting code reusability and reducing redundancy.
- Polymorphism – Enables different classes to have methods with the same name but different implementations, improving flexibility and scalability.
- Abstraction – Hides complex implementation details and exposes only necessary functionalities to the user, enhancing simplicity and security.

Q2) Create a custom Python class for managing a bank account with basic functionalities like deposit and withdrawal?

Ans)

Q2.py

```python
class BankAccount:  1 usage
    def __init__(self, account_holder, balance=0.0):
        self.account_holder = account_holder
        self.balance = balance
    def deposit(self, amount):  1 usage
        if amount > 0:
            self.balance += amount
            print(f"Deposited Amount: ${amount:.2f}")
        else:
            print("The Deposited Amount is Invalid")
    def withdraw(self, amount):  1 usage
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrawn: ${amount:.2f}. New Balance: ${self.balance:.2f}")
            else:
                print("Insufficient balance.")
        else:
            print("Withdrawal amount must be positive.")
    def get_balance(self):  1 usage
        return self.balance
    def __str__(self):
        return f"BankAccount({self.account_holder}, Balance: ${self.balance:.2f})"
account = BankAccount( account_holder: "Ritikh",  balance: 1000)
print(account)
account.deposit(100)
account.withdraw(50)
print(f"Final Balance: ${account.get_balance():.2f}")
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q2.py
BankAccount(Ritikh, Balance: $1000.00)
Deposited Amount: $100.00
Withdrawn: $50.00. New Balance: $1050.00
Final Balance: $1050.00

Process finished with exit code 0
```

Q3) Create a Book class that contains multiple Chapters, where each Chapter has a title and page count. Write code to initialize a Book object with three chapters and display the total page count of the book

Ans)

```
"""A.Ritikh CSE-51"""
class Chapter:  3 usages
    def __init__(self, title, page_count):
        self.title = title
        self.page_count = page_count
class Book:  1 usage
    def __init__(self, title):
        self.title = title
        self.chapters = []
    def add_chapter(self, chapter):  3 usages
        self.chapters.append(chapter)
    def total_pages(self):  1 usage
        return sum(chapter.page_count for chapter in self.chapters)
my_book = Book("Python Programming")
my_book.add_chapter(Chapter( title: "Introduction to Python",  page_count: 20))
my_book.add_chapter(Chapter( title: "Object-Oriented Programming",  page_count: 35))
my_book.add_chapter(Chapter( title: "Advanced Topics",  page_count: 50))
print(f"Total pages in '{my_book.title}': {my_book.total_pages()}")
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q3.py
Total pages in 'Python Programming': 105

Process finished with exit code 0
```

Q4) How does Python enforce access control to class attributes, and what is the difference between public, protected, and private attributes?

Ans) Python controls access to class attributes using naming conventions: public, protected, and private.

1. Public Attributes

- Naming: `attribute` (no underscore)
- Access: Can be used anywhere
- Enforcement: No restriction

2. Protected Attributes

- Naming: `_attribute` (single underscore)
- Access: Meant for internal use but still accessible
- Enforcement: Just a convention

3. Private Attributes

- Naming: `__attribute` (double underscore)
- Access: Cannot be accessed directly but possible using `_ClassName__attribute`

- Enforcement: Partial restriction

DIFFERENCE

TYPE	NAMING	Can Access Outside ?	Enforced	Purpose
Public	attribute	Yes	No	Free to Use
Protected	_attribute	Yes	No	Internal Use
Private	__attribute	No	Somewhat	Prevent accidental access

Q5) Write a Python program using a Time class to input a given time in 24-hour format and convert it to a 12-hour format with AM/PM. The program should also validate time strings to ensure they are in the correct HH:MM:SS format. Implement a method to check if the time is valid and return an appropriate message.

Ans)

```
class Time:  1 usage
    def __init__(self, time_str):
        self.time_str = time_str
    def is_valid_time(self):  1 usage
        try:
            hh, mm, ss = map(int, self.time_str.split(":"))
            return 0 <= hh < 24 and 0 <= mm < 60 and 0 <= ss < 60
        except ValueError:
            return False
    def convert_to_12_hour(self):  1 usage
        if not self.is_valid_time():
            return "Invalid time format. Use HH:MM:SS."
        hh, mm, ss = map(int, self.time_str.split(":"))
        period = "AM" if hh < 12 else "PM"
        hh = hh % 12 or 12
        return f"{hh:02}:{mm:02}:{ss:02} {period}"
time_input = input("Enter time (HH:MM:SS): ")
time_obj = Time(time_input)
print(time_obj.convert_to_12_hour())
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q5.py
Enter time (HH:MM:SS): 23:21:21
11:21:21 PM
```

Q6) **Write a Python program that uses private attributes for creating a BankAccount class. Implement methods to deposit, withdraw, and display the balance, ensuring direct access to the balance attribute is restricted. Explain why using private attributes can help improve data security and prevent accidental modifications.**

Ans) Private attributes (like __balance) prevent direct modification from outside the class, reducing the risk of accidental or malicious changes. This ensures controlled access via methods, improving data security and integrity.

```
class BankAccount:  1 usage
    def __init__(self, balance=0):
        self.__balance = balance

    def deposit(self, amount):  1 usage
        if amount > 0:
            self.__balance += amount

    def withdraw(self, amount):  1 usage
        if 0 < amount <= self.__balance:
            self.__balance -= amount

    def get_balance(self):  1 usage
        return self.__balance

account = BankAccount(1000)
account.deposit(500)
account.withdraw(200)
print("Balance:", account.get_balance())
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q6.py
Balance: 1300

Process finished with exit code 0
```

Q7) Write a Python program to simulate a card game using object-oriented principles. The program should include a Card class to represent individual playing cards, a Deck class to represent a deck of cards, and a Player class to represent players receiving cards. Implement a shuffle method in the Deck class to shuffle the cards and a deal method to distribute cards to players. Display each player's hand after dealing.

Ans)

```
import random
class Card:  1 usage
    def __init__(self, suit, rank):
        self.suit = suit
        self.rank = rank
    def __str__(self):
        return f"{self.rank} of {self.suit}"
class Deck:  1 usage
    def __init__(self):
        suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
        ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']
        self.cards = [Card(suit, rank) for suit in suits for rank in ranks]
    def shuffle(self):  1 usage
        random.shuffle(self.cards)
    def deal(self, num_players, cards_per_player):  1 usage
        return [[self.cards.pop() for _ in range(cards_per_player)] for _ in range(num_players)]
class Player:  1 usage
    def __init__(self, name, hand):
        self.name = name
        self.hand = hand
    def show_hand(self):  1 usage
        return f"{self.name}'s hand: " + ", ".join(str(card) for card in self.hand)
deck = Deck()
deck.shuffle()
players = [Player( name: f"Player {i+1}", hand) for i, hand in enumerate(deck.deal( num_players: 4,  cards_per_player: 5))]
for player in players:
    print(player.show_hand())
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q7.py
Player 1's hand: 8 of Hearts, 5 of Spades, 7 of Spades, 9 of Diamonds, 4 of Spades
Player 2's hand: 9 of Clubs, Q of Hearts, 4 of Diamonds, 2 of Spades, 2 of Diamonds
Player 3's hand: J of Spades, K of Diamonds, 4 of Clubs, Q of Spades, A of Spades
Player 4's hand: 6 of Spades, A of Clubs, J of Diamonds, 3 of Spades, 10 of Spades

Process finished with exit code 0
```

Q8) Write a Python program that defines a base class Vehicle with attributes make and model, and a method display info(). Create a subclass Car that inherits from Vehicle and adds an additional at tribute num doors. Instantiate both Vehicle and Car objects, call their display info() methods, and explain how the subclass inherits and extends the functionality of the base class.

Ans)

```python
"""A.Ritikh Cse-Sec 51
Red No : 2241018124"""
class Vehicle:  2 usages
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def display_info(self):  1 usage
        return f"Vehicle: {self.make} {self.model}"

class Car(Vehicle):  1 usage
    def __init__(self, make, model, num_doors):
        super().__init__(make, model)
        self.num_doors = num_doors

    def display_info(self):  1 usage
        return f"Car: {self.make} {self.model}, Doors: {self.num_doors}"

vehicle = Vehicle( make: "Toyota",  model: "Corolla")
car = Car( make: "Honda",  model: "Civic",  num_doors: 4)

print(vehicle.display_info())
print(car.display_info())
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q8.py
Vehicle: Toyota Corolla
Car: Honda Civic, Doors: 4

Process finished with exit code 0
```

Q9) Write a Python program demonstrating polymorphism by creating a base class Shape with a method area(), and two subclasses Circle and Rectangle that override the area() method. Instantiate objects of both subclasses and call the area() method. Explain how polymorphism simplifies working with different shapes in an inheritance hierarchy.

Ans)

```python
import math

class Shape:  2 usages
    def area(self):
        pass

class Circle(Shape):  1 usage
    def __init__(self, radius):
        self.radius = radius

    def area(self):  1 usage
        return math.pi * self.radius ** 2

class Rectangle(Shape):  1 usage
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):  1 usage
        return self.width * self.height

shapes = [Circle(5), Rectangle( width: 4,  height: 6)]

for shape in shapes:
    print(f"Area: {shape.area():.2f}")
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q9.py
Area: 78.54
Area: 24.00

Process finished with exit code 0
```

Q10) Implement the CommissionEmployee class with __init__ , earnings, and __repr__ methods. Include properties for personal details and sales data. Create a test script to instantiate the object, display earnings, modify sales data, and handle data validation errors for negative values.

Ans)

```python
"""A.Ritikh Cse-Sec 51
Red No : 2241018124"""

class CommissionEmployee:  1 usage
    def __init__(self, name, sales, commission_rate):
        self.name = name
        self.sales = sales if sales >= 0 else 0
        self.commission_rate = commission_rate
    def earnings(self):  1 usage
        return self.sales * self.commission_rate
    def update_sales(self, new_sales):  1 usage
        if new_sales < 0:
            raise ValueError("Sales cannot be negative!")
        self.sales = new_sales
    def __repr__(self):
        return f"CommissionEmployee({self.name}, Sales: {self.sales}, Earnings: {self.earnings():.2f})"
emp = CommissionEmployee( name: "A.Ritikh",  sales: 5000,  commission_rate: 0.1)
print(emp)
emp.update_sales(7000)
print(emp)
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q10.py
CommissionEmployee(A.Ritikh, Sales: 5000, Earnings: 500.00)
CommissionEmployee(A.Ritikh, Sales: 7000, Earnings: 700.00)

Process finished with exit code 0
```

Q11) What is duck typing in Python? Write a Python program demonstrating duck typing by creating a function describe() that accepts any object with a speak() method. Implement two classes, Dog and Robot, each with a speak() method. Pass instances of both classes to the describe() function and explain how duck typing allows the function to work without checking the object's type.

Ans) Duck typing means "If it looks like a duck and quacks like a duck, it's a duck."
The function describe() works with any object that has a speak() method, regardless of class type.

```
"""A.Ritikh Cse-Sec 51
Red No : 2241018124"""

class Dog:  1 usage
    def speak(self):  1 usage (1 dynamic)
        return "Woof!"

class Robot:  1 usage
    def speak(self):  1 usage (1 dynamic)
        return "Beep boop!"

def describe(entity):  2 usages
    print(entity.speak())

describe(Dog())
describe(Robot())
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q11.py
Woof!
Beep boop!

Process finished with exit code 0
```

Q12) WAP to overload the +operator to perform addition of two complex numbers using a custom Complex class?

Ans)

```
"""A.Ritikh Cse-Sec 51
Red No : 2241018124"""

class Complex:  3 usages
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __add__(self, other):
        return Complex(self.real + other.real, self.imag + other.imag)

    def __str__(self):
        return f"{self.real} + {self.imag}i"

c1 = Complex( real: 3,  imag: 4)
c2 = Complex( real: 1,  imag: 2)
print(c1 + c2)
```

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q12.py
4 + 6i

Process finished with exit code 0
```

13. WAP to create a custom exception class in Python that displays the balance and withdrawal amount when an error occurs due to insufficient funds?

Ans)

```
"""A.Ritikh Cse-Sec 51
Red No : 2241018124"""

class InsufficientFundsError(Exception):  2 usages
    def __init__(self, balance, amount):
        super().__init__(f"Insufficient funds! Balance: {balance}, Withdrawal Amount: {amount}")

class BankAccount:  1 usage
    def __init__(self, balance):
        self.balance = balance

    def withdraw(self, amount):  1 usage
        if amount > self.balance:
            raise InsufficientFundsError(self.balance, amount)
        self.balance -= amount
        return self.balance

account = BankAccount(1000)
try:
    account.withdraw(1500)
except InsufficientFundsError as e:
    print(e)
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q13.py
Insufficient funds! Balance: 1000, Withdrawal Amount: 1500

Process finished with exit code 0
```

14. Write a Python program using the Card data class to simulate dealing 5 cards to a player from a shuffled deck of standard playing cards. The program should print the player's hand and the number of remaining cards in the deck after the deal.

Ans)

```
"""A.Ritikh Cse-Sec 51
Red No : 2241018124"""

import random
from dataclasses import dataclass

@dataclass  1 usage
class Card:
    suit: str
    rank: str

class Deck:  1 usage
    def __init__(self):
        suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
        ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']
        self.cards = [Card(suit, rank) for suit in suits for rank in ranks]
        random.shuffle(self.cards)

    def deal(self, num):  1 usage
        return [self.cards.pop() for _ in range(num)]

deck = Deck()
hand = deck.deal(5)
print("Player's Hand:", hand)
print("Cards Left in Deck:", len(deck.cards))
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q14.py
Player's Hand: [Card(suit='Hearts', rank='K'), Card(suit='Spades', rank='9'), Card(suit='Hearts', rank='3'), Card(suit='Hearts', rank='6'), Card(suit='Diamonds', rank='K')]
Cards Left in Deck: 47

Process finished with exit code 0
```

15. How do Python data classes provide advantages over named tuples in terms of flexibility and functionality? Give an example using python code.

Ans)Advantages of Data Classes Over Named Tuples:

Mutability – Data classes allow modification of attributes, while named tuples are immutable.
Methods – Data classes can define methods, making them more functional.
Type Hints & Default Values – Data classes support default values and type hints easily.
Readability – dataclass automatically generates __init__, __repr__, and other useful methods.

```
"""A.Ritikh Cse-Sec 51
Red No : 2241018124"""

from dataclasses import dataclass
@dataclass  1 usage
class Car:
    make: str
    model: str
    year: int
    def display(self):  1 usage
        return f"{self.year} {self.make} {self.model}"
from collections import namedtuple
CarTuple = namedtuple( typename: "CarTuple",  field_names: ["make", "model", "year"])
car1 = Car("Toyota", "Corolla", 2020)
car2 = CarTuple("Honda", "Civic", 2019)
print(car1.display())
print(car2.year)
car1.year = 2021
# car2.year = 2021
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q15.py
2020 Toyota Corolla
2019

Process finished with exit code 0
```

16. Write a Python program that demonstrates unit testing directly within a function's docstring using the doctest module. Create a function add(a, b) that returns the sum of two numbers and includes multiple test cases in its docstring. Implement a way to automatically run the tests when the script is executed.

Ans)

```
"""A.Ritikh Cse-Sec 51
Red No : 2241018124"""

def add(a, b):  3 usages
    """
    Returns the sum of two numbers.

    >>> add(2, 3)
    5
    >>> add(-1, 1)
    0
    >>> add(0, 0)
    0
    """
    return a + b

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Output:

```
C:\Users\ASUS\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\PythonProject\Q16.py

Process finished with exit code 0
```

Q17) Scope Resolution: object's namespace → class namespace → global namespace → built-in names pace.

```
species = "Global Species"

class Animal:
    species = "Class Species"

    def __init__(self, species):
        self.species = species

    def display_species(self):
        print("Instance species:", self.species)
        print("Class species:", Animal.species)
        print("Global species:", globals()['species'])

a = Animal("Instance Species")
a.display_species()
```

What will be the output when the above program is executed? Explain the scope resolution process step by step

Ans) Instance species: Instance Species
Class species: Class Species
Global species: Global Species

Scope Resolution Process:

1. Instance Namespace (self.species)
   - When self.species is accessed, it first looks in the instance namespace.
   - self.species = "Instance Species", so "Instance Species" is printed.
2. Class Namespace (Animal.species)
   - If not found in the instance, Python looks in the class namespace.
   - Animal.species = "Class Species", so "Class Species" is printed.
3. Global Namespace (globals()['species'])
   - If the attribute isn't found in the instance or class, Python checks the global namespace.
   - species = "Global Species" in the global scope, so "Global Species" is printed.
4. Built-in Namespace (Not Used Here)
   - If Python doesn't find the name in instance → class → global scopes, it checks built-in functions/constants.

18. Write a Python program using a lambda function to convert temperatures from Celsius to Kelvin, store the data in a tabular format using pandas, and visualize the data using a plot

Ans)

```
"""A.Ritikh Cse-Sec 51
Red No : 2241018124"""

import pandas as pd
import matplotlib.pyplot as plt
celsius_to_kelvin = lambda c: c + 273.15
celsius_values = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
kelvin_values = list(map(celsius_to_kelvin, celsius_values))
df = pd.DataFrame({"Celsius": celsius_values, "Kelvin": kelvin_values})
print(df)
plt.plot( *args: df["Celsius"], df["Kelvin"], marker="o", linestyle="-", color="b")
plt.xlabel("Celsius (°C)")
plt.ylabel("Kelvin (K)")
plt.title("Celsius to Kelvin Conversion")
plt.grid(True)
plt.show()
```

Output: