Muhammad Sheheryar

BIT-23F-042

SECTION: “B”

ARTIFICIAL INTELLIGENCE

LAB 8

ASSIGNMENT

MISS: AQSA

Task 01:

Q: Write a Python class named Car that represents a car. The class should have the following attributes:

- make: the car's make (e.g., "Toyota")
- model: the car's model (e.g., "Corolla")
- year: the car's manufacturing year (e.g., 2020)
- mileage: the number of miles driven by the car.

The class should have the following methods:

- __init__(self): Constructor to initialize the car's attributes.
- display_info(): Displays the car's information (make, model, year, mileage).
- drive(miles): Increases the mileage by the specified number of miles

CODE:

```
class Car:
    def __init__(self, make, model, year, mileage=0):
        self.make = make
        self.model = model
        self.year = year
        self.mileage = mileage

    def display_info(self):
        print(f"Car Info:\nMake: {self.make}\nModel: {self.model}\nYear: {self.year}\nMileage: {self.mileage} miles")

    def drive(self, miles):
        self.mileage += miles

# Create a Car object with predefined details
car = Car("Toyota", "Corolla", 2020, 15000)
```

```
# Display initial car information
car.display_info()

# Ask the user for miles to drive
miles_to_drive = int(input("Enter miles to drive: "))

# Drive the car and update mileage
car.drive(miles_to_drive)

# Display the updated car information
car.display_info()
```

OUTPUT:

```
Car Info:
Make: Toyota
Model: Corolla
Year: 2020
Mileage: 15000 miles
Enter miles to drive: 160
Car Info:
Make: Toyota
Model: Corolla
Year: 2020
Mileage: 15160 miles
```

Task 02:

Q: Write a Python class named Student that represents a student. The class should have the following attributes:

- name: the student's name.
- age: the student's age.
- marks: a list of the student's marks.

The class should have the following methods:

- __init__(self): Constructor to initialize the student's attributes.
- add_marks(self, marks): Adds a list of marks to the student's marks list.
- average_marks(self): Calculates and returns the average of the student's marks.
- display_info(self): Displays the student's information (name, age, average marks)

CODE:

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.marks = []

    def add_marks(self, marks):
        self.marks.extend(marks)

    def average_marks(self):
        if self.marks:
            return sum(self.marks) / len(self.marks)
        return 0

    def display_info(self):
        print(f"Name: {self.name}, Age: {self.age}, Average Marks: {self.average_marks():.2f}")
```

```
# Example usage
name = input("Enter student's name: ")
age = int(input("Enter student's age: "))

student = Student(name, age)

marks = list(map(int, input("Enter the student's marks (separated by spaces): ").split()))
student.add_marks(marks)

student.display_info()
```

OUTPUT:

```
Enter student's name: ALI
Enter student's age: 20
Enter the student's marks (separated by spaces): 30 30 40 60 60 70 80
Name: ALI, Age: 20, Average Marks: 52.86
```

Task 03:

Q: Write a Python class named BankAccount that represents a bank account. The class should have the following attributes:

- account_holder: the name of the account holder.
- balance: the balance of the account.

The class should have the following methods:

- __init__(self): Constructor to initialize the account holder's name and balance.
- deposit(self, amount): Deposits an amount into the account.
- withdraw(self, amount): Withdraws an amount from the account if there are sufficient funds.
- display_balance(self): Displays the current balance of the account

CODE:

```
class BankAccount:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited ${amount}. New balance: ${self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0 and amount <= self.balance:
            self.balance -= amount
```

```
            print(f"Withdrew ${amount}. New balance: ${self.balance}")
        else:
            print("Invalid withdrawal amount or insufficient funds.")

    def display_info(self):
        print(f"Account Holder: {self.account_holder}")
        print(f"Current Balance: ${self.balance}")

# Example usage
account = BankAccount("John Doe", 1000)

# Display initial account info
account.display_info()

# Ask user for deposit
deposit_amount = float(input("Enter amount to deposit: "))
account.deposit(deposit_amount)

# Ask user for withdrawal
withdraw_amount = float(input("Enter amount to withdraw: "))
account.withdraw(withdraw_amount)

# Display updated account info
account.display_info()
```

OUTPUT:

Account Holder: John Doe

Current Balance: $1000

Enter amount to deposit: 1500

Deposited $1500.0. New balance: $2500.0

Enter amount to withdraw: 500

Withdrew $500.0. New balance: $2000.0

Account Holder: John Doe

Current Balance: $2000.0