**RUHMA ALI**

**BIT-23F-015**

**LAB TASK # 08**

**Task # 1**

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:

"""

A class to represent a car with attributes for make, model, year, and mileage.

"""

def \_\_init\_\_(self, make, model, year, mileage=0):

"""

Constructor to initialize the car's attributes.

Args:

make (str): The make of the car (e.g., "Toyota").

model (str): The model of the car (e.g., "Corolla").

year (int): The manufacturing year of the car.

mileage (int, optional): The initial mileage of the car. Defaults to 0.

"""

self.make = make

self.model = model

self.year = year

self.mileage = mileage

def display\_info(self):

"""

Displays the car's information (make, model, year, mileage).

"""

print(f"Car Information:")

print(f"Make: {self.make}")

print(f"Model: {self.model}")

print(f"Year: {self.year}")

print(f"Mileage: {self.mileage} miles")

def drive(self, miles):

"""

Simulates driving the car by increasing the mileage by the given number of miles.

Args:

miles (int): The number of miles driven.

"""

if miles < 0:

print("Miles driven cannot be negative.")

else:

self.mileage += miles

print(f"Driven {miles} miles. Total mileage is now {self.mileage} miles.")

my\_car = Car("Toyota", "Corolla", 2020)

my\_car.display\_info()

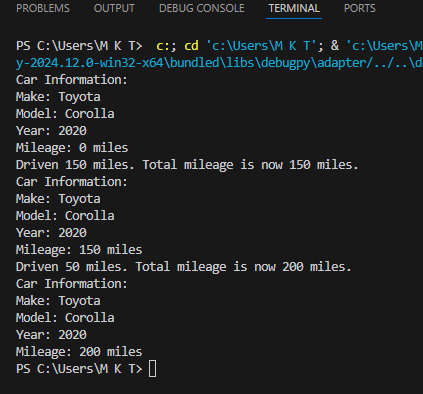
my\_car.drive(150)

my\_car.display\_info()

my\_car.drive(50)

my\_car.display\_info()

**Output:**



**Task # 2**

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:

"""

A class to represent a student with attributes for name, age, and marks.

"""

def \_\_init\_\_(self, name, age):

"""

Constructor to initialize the student's attributes.

Args:

name (str): The name of the student.

age (int): The age of the student.

"""

self.name = name

self.age = age

self.marks = []

def add\_marks(self, marks):

"""

Adds a list of marks to the student's marks list.

Args:

marks (list of int/float): A list of marks to add.

"""

self.marks.extend(marks)

print(f"Marks {marks} have been added to {self.name}'s record.")

def average\_marks(self):

"""

Calculates and returns the average of the student's marks.

Returns:

float: The average of marks. Returns 0 if there are no marks.

"""

if len(self.marks) == 0:

print("No marks available to calculate average.")

return 0

return sum(self.marks) / len(self.marks)

def display\_info(self):

"""

Displays the student's information (name, age, average marks).

"""

average = self.average\_marks()

print(f"Student Information:")

print(f"Name: {self.name}")

print(f"Age: {self.age}")

print(f"Average Marks: {average:.2f}")

# Create a new student instance

student\_1 = Student("Alice", 20)

# Display initial information

student\_1.display\_info()

# Add marks to the student's record

student\_1.add\_marks([85, 90, 78, 92])

# Display updated information with average marks

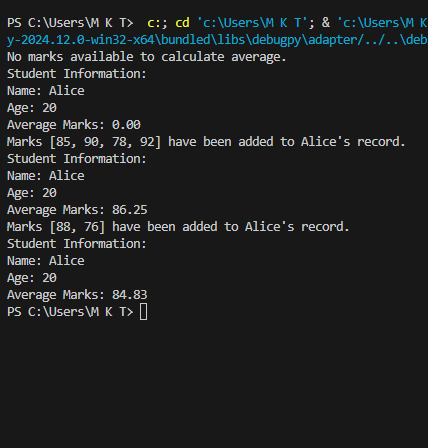
student\_1.display\_info()

# Add more marks

student\_1.add\_marks([88, 76])

student\_1.display\_info()

**Output:**



**Task # 3**

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:

"""

A class to represent a bank account with account holder name and balance.

"""

def \_\_init\_\_(self, account\_holder, initial\_balance=0):

"""

Constructor to initialize the account holder's name and balance.

Args:

account\_holder (str): The name of the account holder.

initial\_balance (float, optional): The initial balance of the account. Defaults to 0.

"""

self.account\_holder = account\_holder

self.balance = initial\_balance

def deposit(self, amount):

"""

Deposits an amount into the bank account.

Args:

amount (float): The amount to deposit.

"""

if amount <= 0:

print("Deposit amount must be positive.")

return

self.balance += amount

print(f"Deposited ${amount:.2f}. New balance is ${self.balance:.2f}.")

def withdraw(self, amount):

"""

Withdraws an amount from the bank account if there are sufficient funds.

Args:

amount (float): The amount to withdraw.

"""

if amount <= 0:

print("Withdrawal amount must be positive.")

return

if amount > self.balance:

print("Insufficient funds! Transaction canceled.")

else:

self.balance -= amount

print(f"Withdrew ${amount:.2f}. New balance is ${self.balance:.2f}.")

def display\_balance(self):

"""

Displays the current balance of the bank account.

"""

print(f"Account Holder: {self.account\_holder}")

print(f"Current Balance: ${self.balance:.2f}")

# Create a bank account instance

account = BankAccount("John Doe", 1000)

# Display initial balance

account.display\_balance()

# Perform a deposit

account.deposit(500)

# Perform a withdrawal

account.withdraw(300)

# Attempt to withdraw more than the balance

account.withdraw(1500)

# Display final balance

account.display\_balance()

**Output:**

