ASSIGNMENT-6.4

2303a51060

Batch-10

Task-1

Prompt: Generate a student performance evaluation system with attributes like name,roll-number, marks and give a message according to the marks obtained by the student with user input.

Code:

```
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def evaluate_performance(self):
        if self.marks >= 90:
            return "Excellent"
        elif self.marks >= 75:
            return "Good"
        elif self.marks >= 60:
            return "Average"
        else:
            return "Needs Improvement"
# Taking user input
name = input("Enter student's name: ")
roll_number = input("Enter student's roll number: ")
marks = float(input("Enter student's marks: "))
# Creating a Student object
```

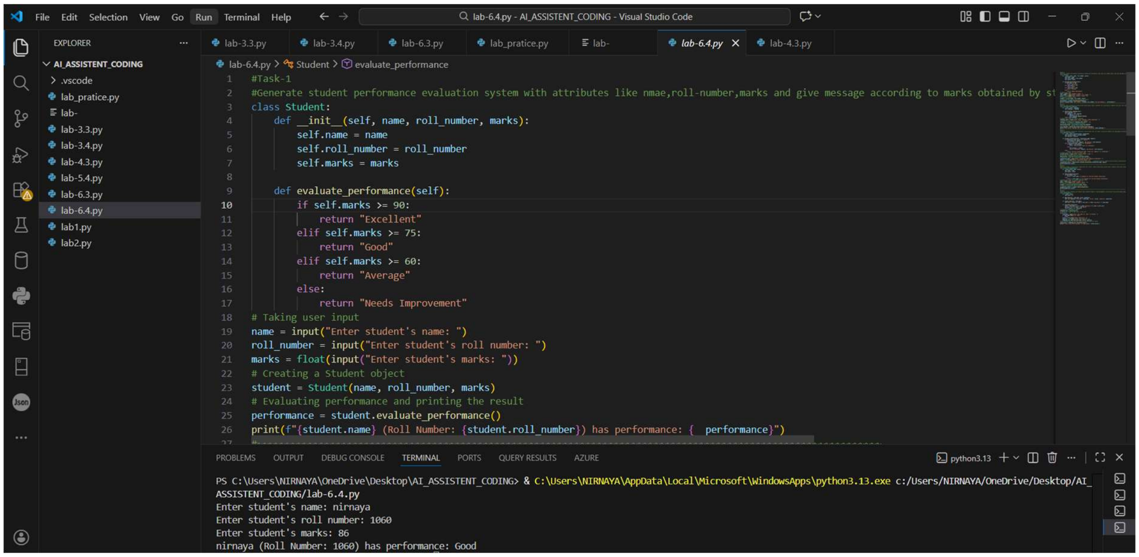
student = Student(name, roll_number, marks)

# Evaluating performance and printing the result

performance = student.evaluate_performance()

print(f"{student.name} (Roll Number: {student.roll_number}) has performance: { performance}")

Output :



```python
#Task-1
#Generate student performance evaluation system with attributes like nmae,roll-number,marks and give message according to marks obtained by st
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def evaluate_performance(self):
        if self.marks >= 90:
            return "Excellent"
        elif self.marks >= 75:
            return "Good"
        elif self.marks >= 60:
            return "Average"
        else:
            return "Needs Improvement"
# Taking user input
name = input("Enter student's name: ")
roll_number = input("Enter student's roll number: ")
marks = float(input("Enter student's marks: "))
# Creating a Student object
student = Student(name, roll_number, marks)
# Evaluating performance and printing the result
performance = student.evaluate_performance()
print(f"{student.name} (Roll Number: {student.roll_number}) has performance: {  performance}")
```

```
PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTENT_CODING> & C:\Users\NIRNAYA\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/NIRNAYA/OneDrive/Desktop/AI_
ASSISTENT_CODING/lab-6.4.py
Enter student's name: nirnaya
Enter student's roll number: 1060
Enter student's marks: 86
nirnaya (Roll Number: 1060) has performance: Good
```

Code Analysis :

- A `Student` class is created with attributes: name, roll number, and marks.
- Constructor (`__init__`) initialises student details when an object is created.
- `evaluate_performance()` method uses conditional statements to classify performance.
- Marks ranges determine messages like Excellent, Good, Average, etc.
- User input is taken to create an object and call the evaluation method.

Task-2

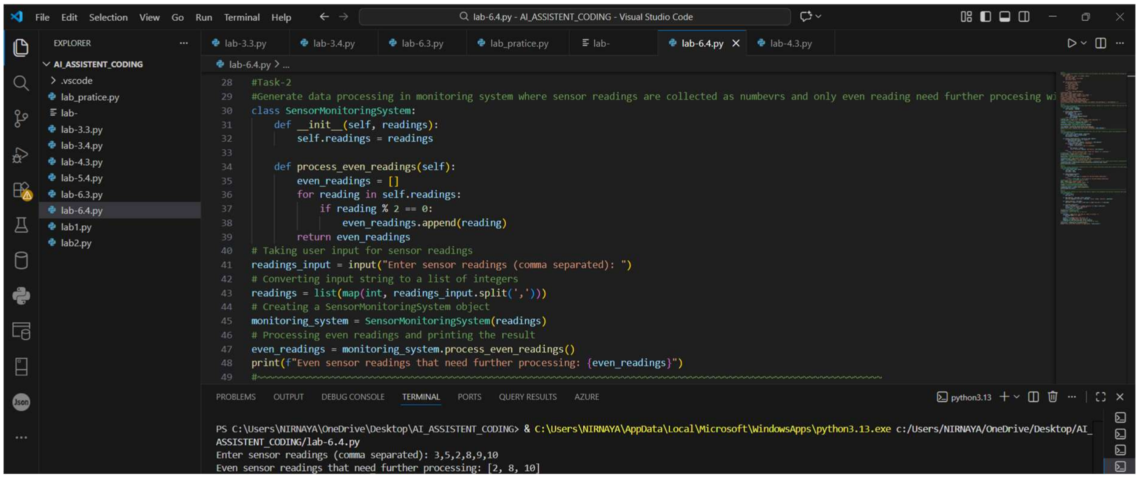
Prompt: Generate data processing in a monitoring system where sensor readings are collected as numbers and only even readings need further processing with a for loop to iterate over a list of integer readings.

Code:

class SensorMonitoringSystem:

```
    def __init__(self, readings):
        self.readings = readings

    def process_even_readings(self):
        even_readings = []
        for reading in self.readings:
            if reading % 2 == 0:
                even_readings.append(reading)
        return even_readings
# Taking user input for sensor readings
readings_input = input("Enter sensor readings (comma separated): ")
# Converting input string to a list of integers
readings = list(map(int, readings_input.split(',')))
# Creating a SensorMonitoringSystem object
monitoring_system = SensorMonitoringSystem(readings)
# Processing even readings and printing the result
even_readings = monitoring_system.process_even_readings()
print(f"Even sensor readings that need further processing: {even_readings}")
```

Output :



```
#Task-2
#Generate data processing in monitoring system where sensor readings are collected as numbevrs and only even reading need further procesing wi
class SensorMonitoringSystem:
    def __init__(self, readings):
        self.readings = readings

    def process_even_readings(self):
        even_readings = []
        for reading in self.readings:
            if reading % 2 == 0:
                even_readings.append(reading)
        return even_readings
# Taking user input for sensor readings
readings_input = input("Enter sensor readings (comma separated): ")
# Converting input string to a list of integers
readings = list(map(int, readings_input.split(',')))
# Creating a SensorMonitoringSystem object
monitoring_system = SensorMonitoringSystem(readings)
# Processing even readings and printing the result
even_readings = monitoring_system.process_even_readings()
print(f"Even sensor readings that need further processing: {even_readings}")
#
```

```
PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTENT_CODING> & C:\Users\NIRNAYA\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/NIRNAYA/OneDrive/Desktop/AI_
ASSISTENT_CODING/lab-6.4.py
Enter sensor readings (comma separated): 3,5,2,8,9,10
Even sensor readings that need further processing: [2, 8, 10]
```

Code Analysis :

- A class `SensorMonitoringSystem` stores sensor readings in a list.
- Constructor receives readings and assigns them to an instance variable.
- `process_even_readings()` iterates through the list using a loop.
- Modulus operator (`% 2`) checks whether readings are even.
- Even readings are collected in a new list and returned for further processing.

Task-3

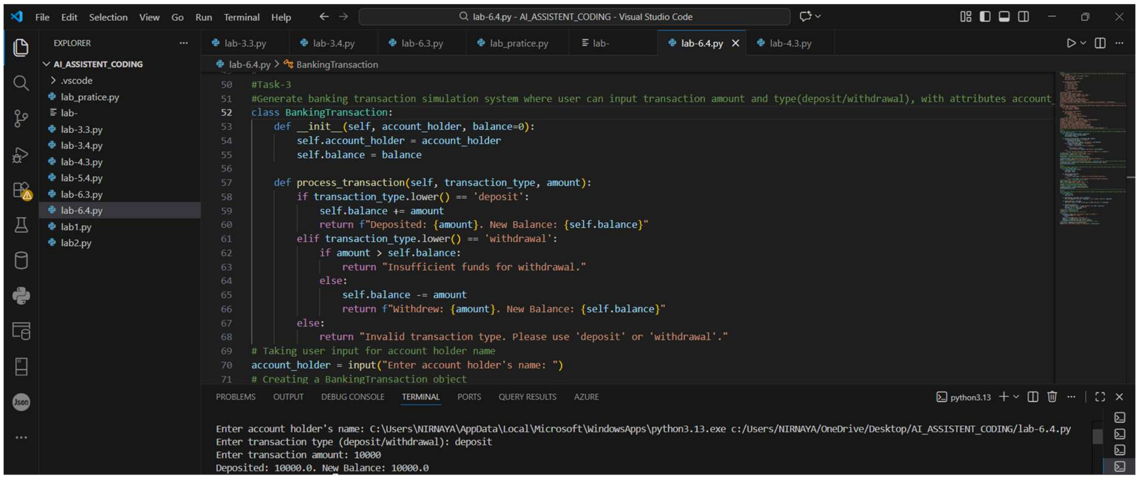
Prompt: Generate a banking transaction simulation system where the user can input transaction amount and type(deposit/withdrawal), with attributes account_holder and balance.

Code:

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

    def process_transaction(self, transaction_type, amount):
        if transaction_type.lower() == 'deposit':
            self.balance += amount
            return f"Deposited: {amount}. New Balance: {self.balance}"
        elif transaction_type.lower() == 'withdrawal':
            if amount > self.balance:
                return "Insufficient funds for withdrawal."
            else:
                self.balance -= amount
                return f"Withdrew: {amount}. New Balance: {self.balance}"
        else:
            return "Invalid transaction type. Please use 'deposit' or 'withdrawal'."
# Taking user input for account holder name
account_holder = input("Enter account holder's name: ")
```

```
# Creating a BankingTransaction object

bank_account = BankingTransaction(account_holder)

# Taking user input for transaction type and amount

transaction_type = input("Enter transaction type (deposit/withdrawal): ")

amount = float(input("Enter transaction amount: "))

# Processing the transaction and printing the result    transaction_result =
bank_account.process_transaction(transaction_type, amount)

transaction_result = bank_account.process_transaction(transaction_type, amount)

print(transaction_result)
```

Output :



```
#Task-3
#Generate banking transaction simulation system where user can input transaction amount and type(deposit/withdrawal), with attributes account_
class BankingTransaction:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def process_transaction(self, transaction_type, amount):
        if transaction_type.lower() == 'deposit':
            self.balance += amount
            return f"Deposited: {amount}. New Balance: {self.balance}"
        elif transaction_type.lower() == 'withdrawal':
            if amount > self.balance:
                return "Insufficient funds for withdrawal."
            else:
                self.balance -= amount
                return f"Withdrew: {amount}. New Balance: {self.balance}"
        else:
            return "Invalid transaction type. Please use 'deposit' or 'withdrawal'."
# Taking user input for account holder name
account_holder = input("Enter account holder's name: ")
# Creating a BankingTransaction object
```

```
Enter account holder's name: C:\Users\NIRNAYA\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/NIRNAYA/OneDrive/Desktop/AI_ASSISTENT_CODING/lab-6.4.py
Enter transaction type (deposit/withdrawal): deposit
Enter transaction amount: 10000
Deposited: 10000.0. New Balance: 10000.0
```

Code Analysis :

- `BankingTransaction` class models a bank account with holder name and balance.
- Constructor initializes account holder and optional starting balance.
- `process_transaction()` method handles deposit and withdrawal operations.
- Conditional logic updates balance and checks insufficient funds.
- Returns transaction result message instead of directly printing.

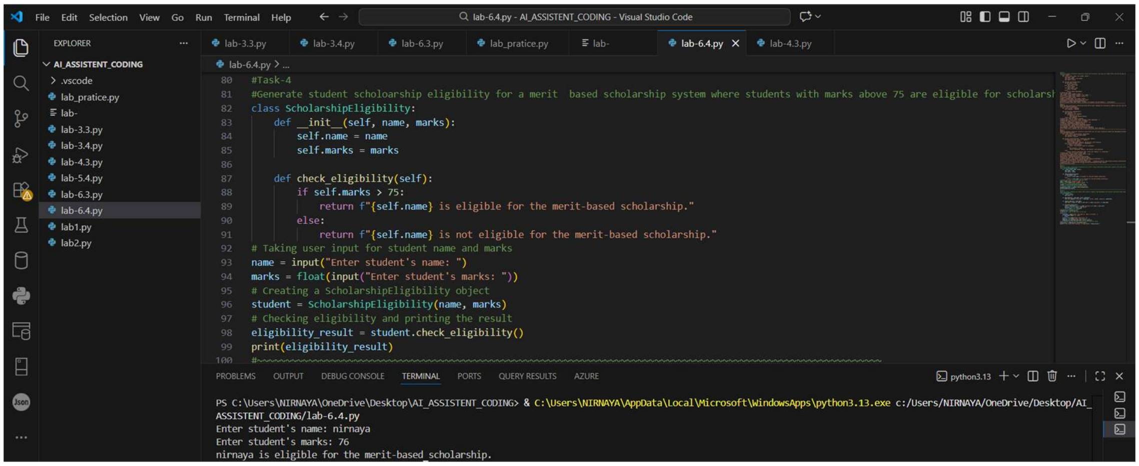
Task-4

Prompt: Generate student scholarship eligibility for a merit based scholarship system where students with marks above 75 are eligible for a scholarship, and a method to check eligibility.

Code:

```
class ScholarshipEligibility:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def check_eligibility(self):
        if self.marks > 75:
            return f"{self.name} is eligible for the merit-based scholarship."
        else:
            return f"{self.name} is not eligible for the merit-based scholarship."
# Taking user input for student name and marks
name = input("Enter student's name: ")
marks = float(input("Enter student's marks: "))
# Creating a ScholarshipEligibility object
student = ScholarshipEligibility(name, marks)
# Checking eligibility and printing the result
eligibility_result = student.check_eligibility()
print(eligibility_result)
```

Output :



File Edit Selection View Go Run Terminal Help
lab-6.4.py - AI_ASSISTENT_CODING - Visual Studio Code
EXPLORER
AI_ASSISTENT_CODING
.vscode
lab_pratice.py
lab-
lab-3.3.py
lab-3.4.py
lab-4.3.py
lab-5.4.py
lab-6.3.py
lab-6.4.py
lab1.py
lab2.py
lab-3.3.py lab-3.4.py lab-6.3.py lab_pratice.py lab- lab-6.4.py lab-4.3.py
lab-6.4.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS AZURE
python3.13

```
#Task-4
#Generate student scholoarship eligibility for a merit  based scholarship system where students with marks above 75 are eligible for scholarsh
class ScholarshipEligibility:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def check_eligibility(self):
        if self.marks > 75:
            return f"{self.name} is eligible for the merit-based scholarship."
        else:
            return f"{self.name} is not eligible for the merit-based scholarship."
# Taking user input for student name and marks
name = input("Enter student's name: ")
marks = float(input("Enter student's marks: "))
# Creating a ScholarshipEligibility object
student = ScholarshipEligibility(name, marks)
# Checking eligibility and printing the result
eligibility_result = student.check_eligibility()
print(eligibility_result)
```

```
PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTENT_CODING> & C:\Users\NIRNAYA\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/NIRNAYA/OneDrive/Desktop/AI_
ASSISTENT_CODING/lab-6.4.py
Enter student's name: nirnaya
Enter student's marks: 76
nirnaya is eligible for the merit-based scholarship.
```

Code Analysis :

- `ScholarshipEligibility` class stores student name and marks.
- Constructor initializes attributes when object is created.
- `check_eligibility()` method checks marks using a condition (`> 75`).
- Returns eligibility message based on performance.
- Demonstrates simple decision-making using class methods.

Task-5

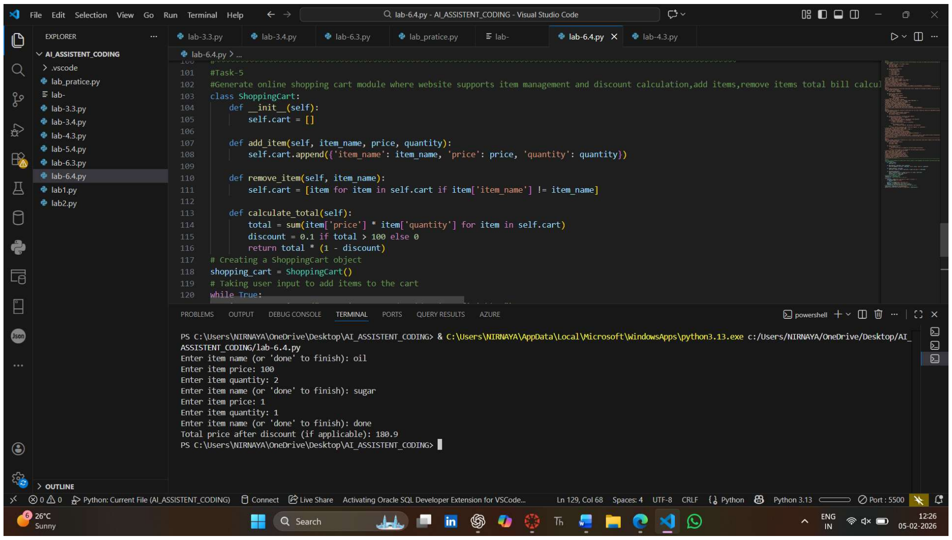
Prompt: Generate an online shopping cart module where the website supports item management and discount calculation, add items, remove items total bill based on the total price with attributes like item_name, price and quantity, and a method to calculate total price after discount with user input.

Code:

```
class ShoppingCart:
    def __init__(self):
        self.cart = []

    def add_item(self, item_name, price, quantity):
        self.cart.append({'item_name': item_name, 'price': price, 'quantity': quantity})

    def remove_item(self, item_name):
        self.cart = [item for item in self.cart if item['item_name'] != item_name]

    def calculate_total(self):
        total = sum(item['price'] * item['quantity'] for item in self.cart)
        discount = 0.1 if total > 100 else 0
        return total * (1 - discount)
# Creating a ShoppingCart object
shopping_cart = ShoppingCart()
# Taking user input to add items to the cart
```

```
while True:
    item_name = input("Enter item name (or 'done' to finish): ")
    if item_name.lower() == 'done':
        break
    price = float(input("Enter item price: "))
    quantity = int(input("Enter item quantity: "))
    shopping_cart.add_item(item_name, price, quantity)
# Calculating total price after discount and printing the result
total_price = shopping_cart.calculate_total()
print(f"Total price after discount (if applicable): {total_price}")
```

Output :



```
#Task-5
#Generate online shopping cart module where website supports item management and discount calculation,add items,remove items total bill calcul
class ShoppingCart:
    def __init__(self):
        self.cart = []

    def add_item(self, item_name, price, quantity):
        self.cart.append({'item_name': item_name, 'price': price, 'quantity': quantity})

    def remove_item(self, item_name):
        self.cart = [item for item in self.cart if item['item_name'] != item_name]

    def calculate_total(self):
        total = sum(item['price'] * item['quantity'] for item in self.cart)
        discount = 0.1 if total > 100 else 0
        return total * (1 - discount)
# Creating a ShoppingCart object
shopping_cart = ShoppingCart()
# Taking user input to add items to the cart
while True:
```

```
PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTENT_CODING> & C:\Users\NIRNAYA\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/NIRNAYA/OneDrive/Desktop/AI_
ASSISTENT_CODING/lab-6.4.py
Enter item name (or 'done' to finish): oil
Enter item price: 100
Enter item quantity: 2
Enter item name (or 'done' to finish): sugar
Enter item price: 1
Enter item quantity: 1
Enter item name (or 'done' to finish): done
Total price after discount (if applicable): 180.9
PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTENT_CODING>
```

Code Analaysis :

- `ShoppingCart` class uses a list to store cart items as dictionaries.
- `add_item()` method adds item name, price, and quantity to cart.
- `remove_item()` removes items using list comprehension.
- `calculate_total()` computes the total price and applies a discount if total > 100.
- Loop allows the user to add multiple items dynamically before final billing.