ASSIGNMENT -6.4

2303a51343

B-10

TASK-1

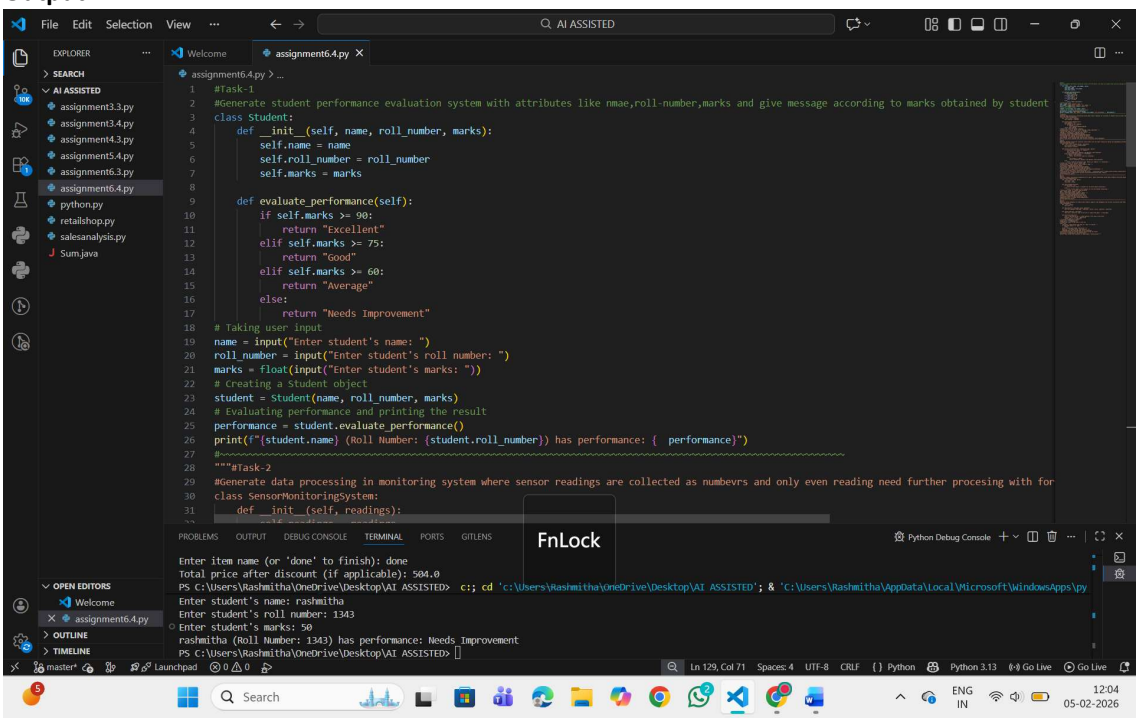
Prompt: Generate student performance evaluation system with attributes like name

Rollno, marks and give message according to marks obtained by 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:



Analysis: This program uses a Student class to store student details and marks.
It evaluates performance using conditional statements based on marks.
The result displays a performance message like Excellent, Good, or Average

TASK-2

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

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

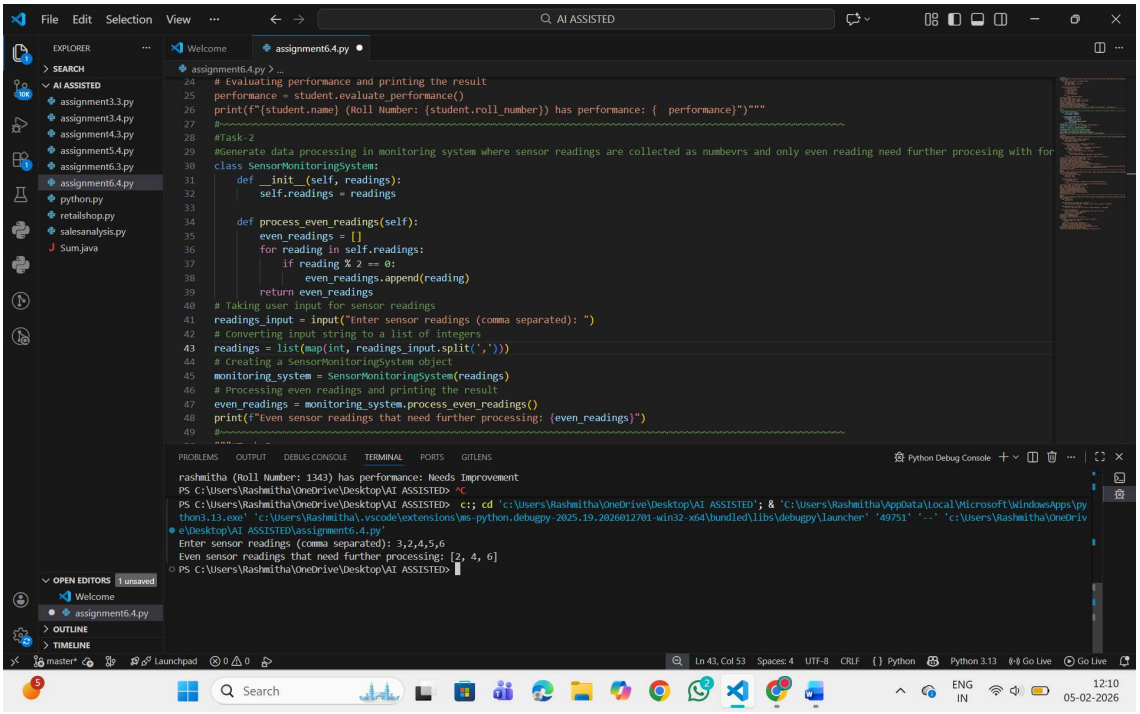
monitoring_system = SensorMonitoringSystem(readings)

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:



Analysis: This program collects sensor readings from the user as a list.
A for loop checks each reading and selects only even numbers.
Even readings are stored and displayed for further processing

TASK-3

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

Code: def __init__(self, account_holder, balance=0):

self.account_holder = account_holder

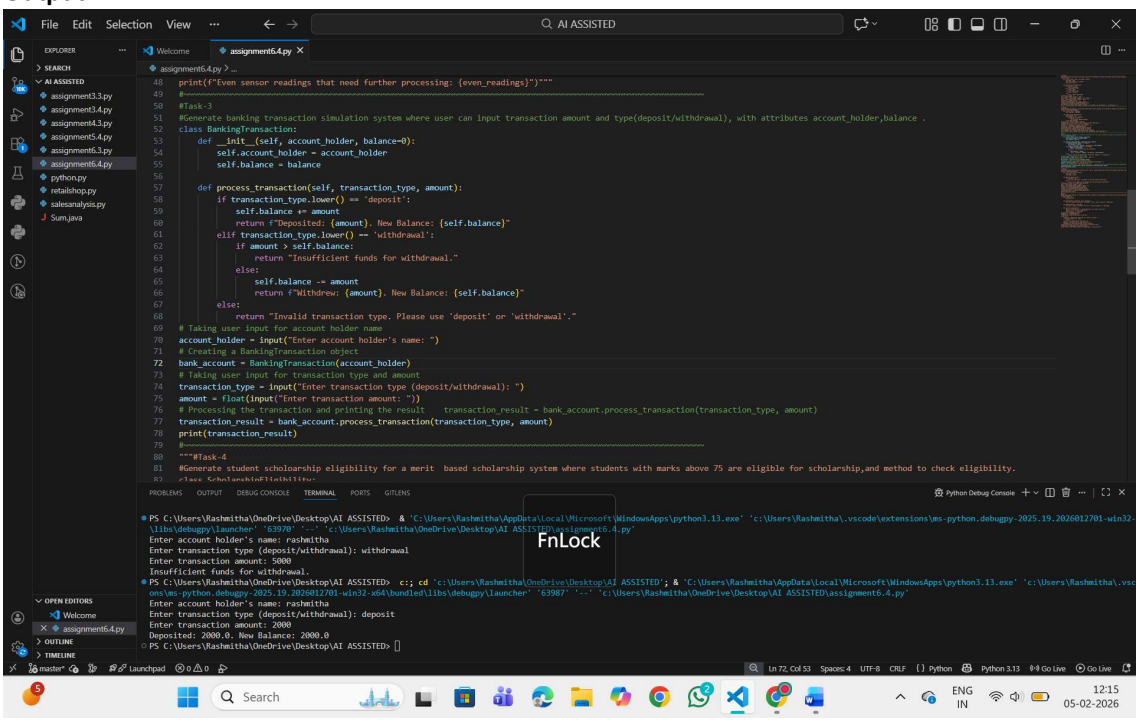
self.balance = balance

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:



Analysis: this system simulates deposit and withdrawal operations using a class.
It updates account balance based on transaction type and amount.
It also prevents withdrawal when funds are insufficient**.**

TASK-4

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

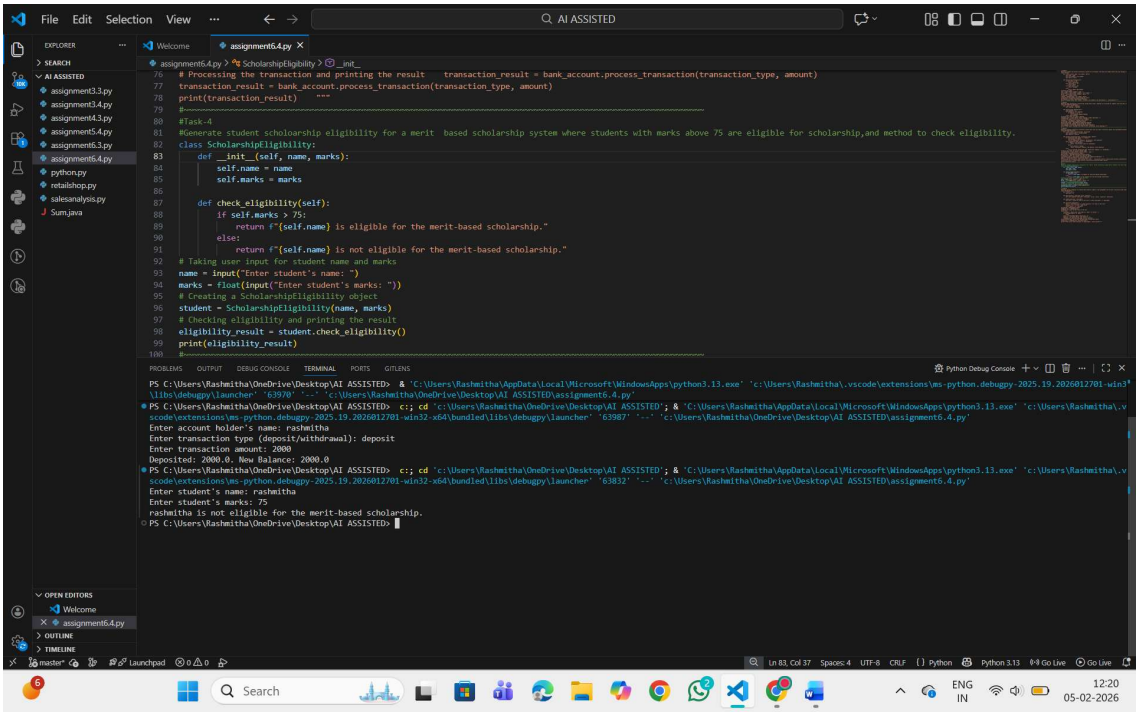
student = ScholarshipEligibility(name, marks)

Checking eligibility and printing the result

eligibility_result = student.check_eligibility()

print(eligibility_result)

Output:



Analysis: This program checks student eligibility for a merit-based scholarship.
Students with marks above 75 are considered eligible.
The result is displayed using a class method

TASK-5

Prompt: Create a Python class ShoppingCart that stores items. Add methods to add items, remove items, calculate total using a loop, and apply discount if total exceeds a limit user input.

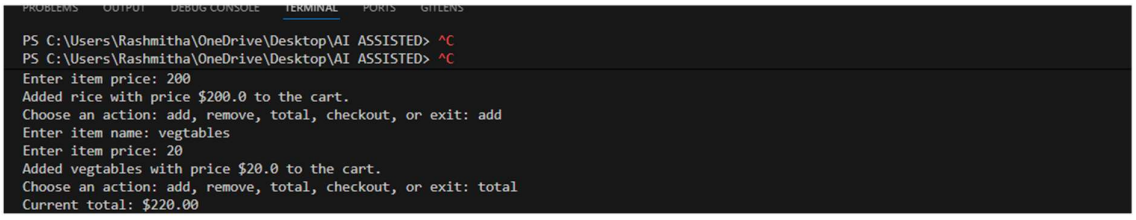
Code: class ShoppingCart:

```
    def __init__(self):
        self.items = []
    def add_item(self, item_name, price):
        """Add an item with its price to the shopping cart."""
        self.items.append({"name": item_name, "price": price})
```

```
        print(f"Added {item_name} with price ${price} to the cart.")

    def remove_item(self, item_name):
        """Remove an item from the shopping cart by name."""
        for item in self.items:
            if item["name"] == item_name:
                self.items.remove(item)
                print(f"Removed {item_name} from the cart.")

            return
        print(f"Item {item_name} not found in the cart.")

    def calculate_total(self):
        """Calculate the total price of items in the cart."""
        total = 0
        for item in self.items:
            total += item["price"]
        return total

    def apply_discount(self, discount_threshold, discount_rate):
        """Apply a discount if the total exceeds a certain threshold."""
        total = self.calculate_total()
        if total > discount_threshold:
            discount = total * discount_rate
            total -= discount
            print(f"Discount of ${discount:.2f} applied.")
        return total
# Demonstration of the ShoppingCart class
cart = ShoppingCart()
while True:
    action = input("Choose an action: add, remove, total, checkout, or exit: ").lower()
```

```
    if action == "add":
        item_name = input("Enter item name: ")
        price = float(input("Enter item price: "))
        cart.add_item(item_name, price)
    elif action == "remove":
        item_name = input("Enter item name to remove: ")
        cart.remove_item(item_name)
    elif action == "total":
        total = cart.calculate_total()
        print(f"Current total: ${total:.2f}")
    elif action == "checkout":
        discount_threshold = float(input("Enter discount threshold: "))
        discount_rate = float(input("Enter discount rate (as a decimal): "))
        final_total = cart.apply_discount(discount_threshold, discount_rate)
        print(f"Final total after discount (if applicable): ${final_total:.2f}")
    elif action == "exit":
        print("Exiting the program.")
        break
    else:
        print("Invalid action. Please choose again.")
```

Output:



Analysis: This program manages a shopping cart with add and remove options.
It calculates the total price using a loop.
A discount is applied if the total exceeds a user-defined limit