AI Assisted Coding Lab-6.4

2303A51693

Task 1: Student Performance Evaluation System

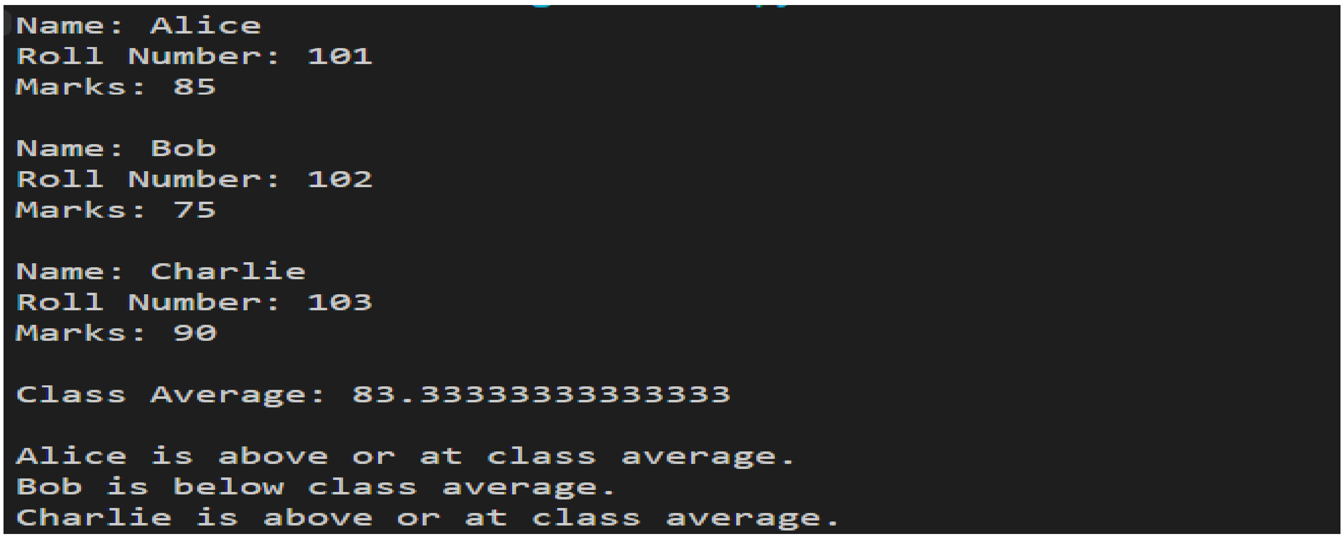
Prompt:

Create the skeleton of a Python class named Student with the attributes: name , roll_number , marks. And A method to display student details, A method that checks whether the student's marks are above the class average and returns an appropriate message

Code:

```python
#Task-01
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")

    def check_above_average(self, class_average):
        if self.marks >= class_average:
            print(f"{self.name} is above or at class 
average.")
        else:
            print(f"{self.name} is below class average.")

# Example usage
students = [
    Student("Alice", 101, 85),
    Student("Bob", 102, 75),
    Student("Charlie", 103, 90)
]

# Display all students
for student in students:
    student.display_details()
    print()

# Calculate class average
class_average = sum(s.marks for s in students) / len(students)
print(f"Class Average: {class_average}\n")

# Check each student against average
for student in students:
    student.check_above_average(class_average)
```

Output:



```
Name:  Alice
Roll Number:  101
Marks:  85

Name:  Bob
Roll Number:  102
Marks:  75

Name:  Charlie
Roll Number:  103
Marks:  90

Class Average:  83.33333333333333

Alice is above or at class average.
Bob is below class average.
Charlie is above or at class average.
```

Task 2: Data Processing in a Monitoring System

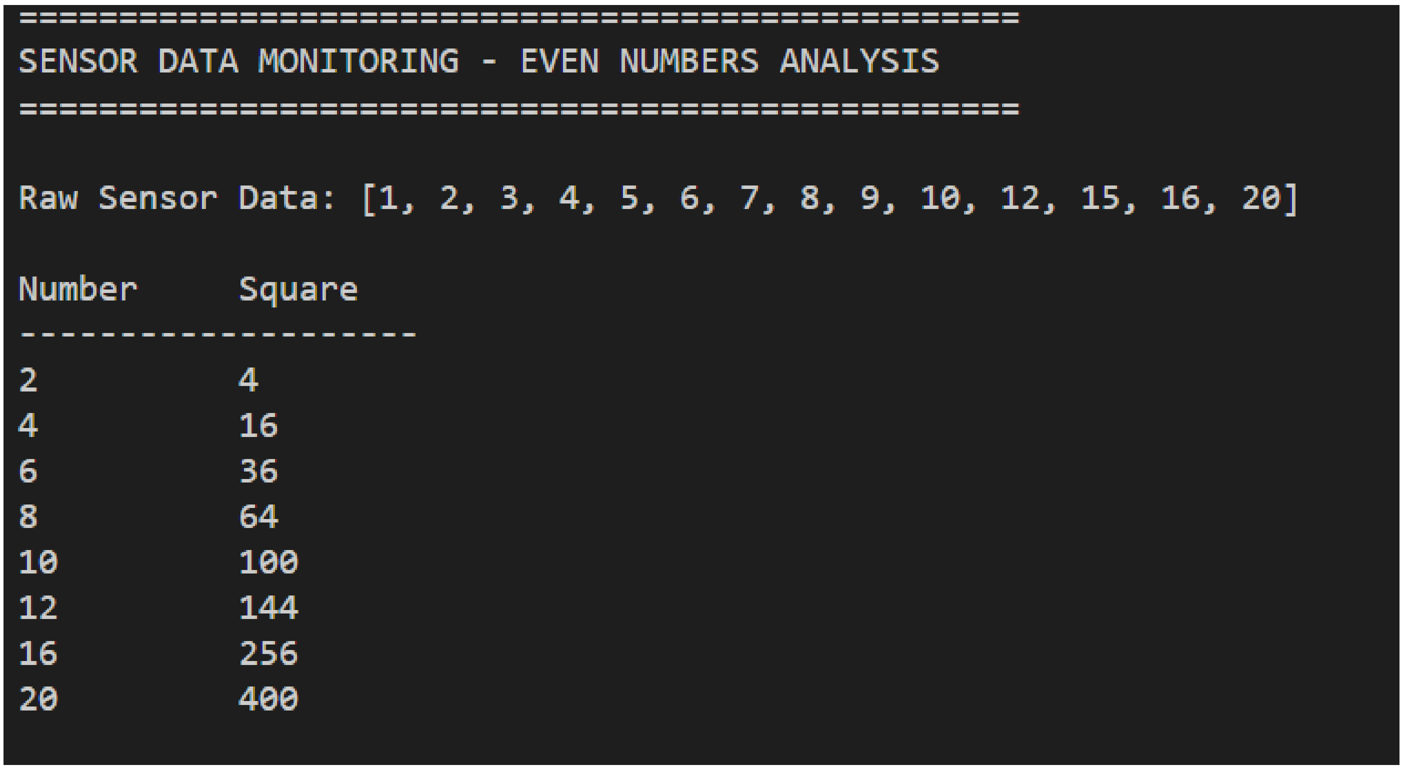
Prompt:

Write the initial part of a for loop to iterate over a list of integers representing sensor readings. And Identify even numbers, Calculate their square, Print the result in a readable format

Code:

```python
#TASK 2
# Data monitoring script - sensor reading even numbers
sensor_data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 16, 20]

print("=" * 50)
print("SENSOR DATA MONITORING - EVEN NUMBERS ANALYSIS")
print("=" * 50)
print(f"\nRaw Sensor Data: {sensor_data}\n")

print(f"{'Number':<10} {'Square':<10}")
print("-" * 20)

for number in sensor_data:
    if number % 2 == 0:
        square = number ** 2
        print(f"{number:<10} {square:<10}")

print("\n" + "=" * 50)
```

Output:



```
=====================================================
SENSOR DATA MONITORING - EVEN NUMBERS ANALYSIS
=====================================================

Raw Sensor Data: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 16, 20]

Number     Square
--------------------
2          4
4          16
6          36
8          64
10         100
12         144
16         256
20         400
```

Task 3: Banking Transaction Simulation

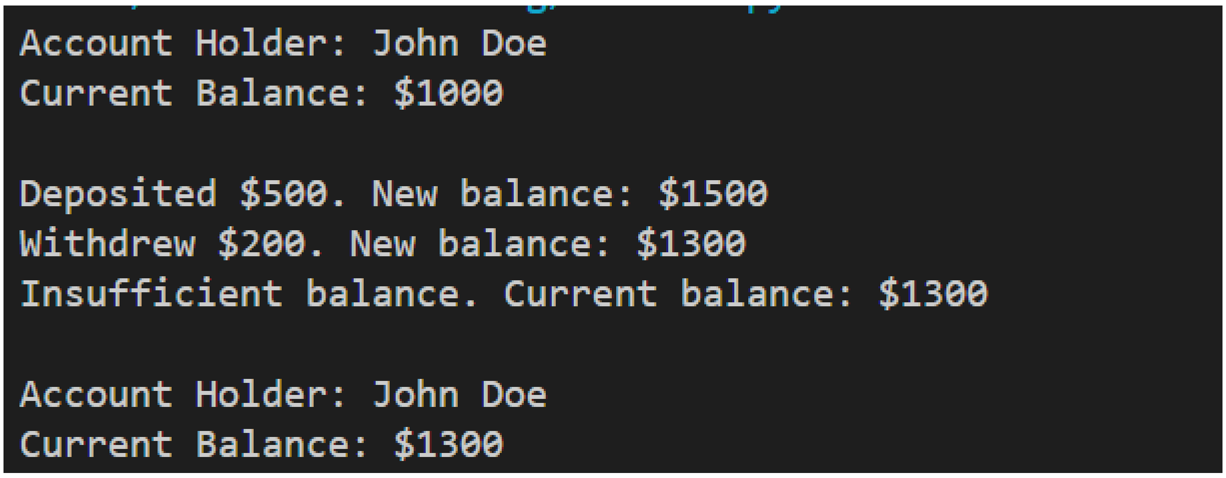
Prompt:

Create the structure of a Python class named BankAccount with attributes: account_holder, balance and methods for Depositing money, Withdrawing money, Preventing withdrawals when the balance is insufficient

Code:

```
#TASK3
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:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew ${amount}. New balance: 
${self.balance}")
            else:
                print(f"Insufficient balance. Current balance: ${self.balance}")
        else:
            print("Withdrawal amount must be positive.")

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


# Example usage
account1 = BankAccount("John Doe", 1000)
account1.display_balance()
print()

account1.deposit(500)
account1.withdraw(200)
account1.withdraw(2000)
print()
account1.display_balance()
```

Output:



```
Account Holder: John Doe
Current Balance: $1000

Deposited $500. New balance: $1500
Withdrew $200. New balance: $1300
Insufficient balance. Current balance: $1300

Account Holder: John Doe
Current Balance: $1300
```

Task 4: Student Scholarship Eligibility Check

Prompt:

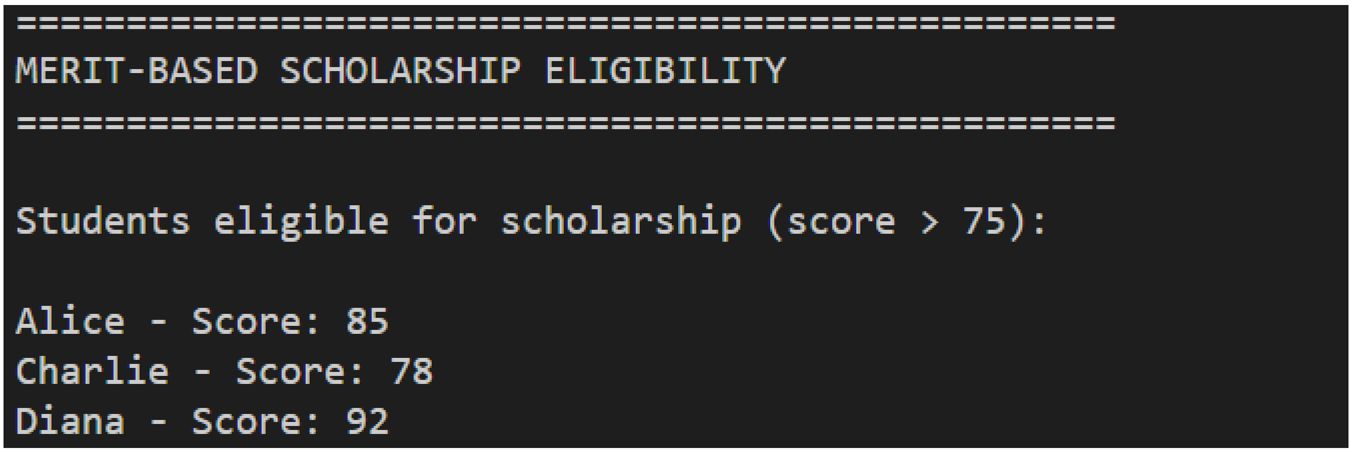
Define a list of dictionaries where each dictionary represents a student with: name, score. And generate a while loop that: Iterates through the list and Prints the names of students who scored more than 75

Code:

```
# Task 4: Merit-based Scholarship Eligibility
students_data = [
    {"name": "Alice", "score": 85},
    {"name": "Bob", "score": 65},
    {"name": "Charlie", "score": 78},
    {"name": "Diana", "score": 92},
    {"name": "Eve", "score": 72}
]
```

```
print("=" * 50)
print("MERIT-BASED SCHOLARSHIP ELIGIBILITY")
print("=" * 50)
print("\nStudents eligible for scholarship (score > 75):\n")

index = 0
while index < len(students_data):
    student = students_data[index]
    if student["score"] > 75:
        print(f"{student['name']} - Score: 
{student['score']}")
    index += 1
```

Output:



```
==================================================
MERIT-BASED SCHOLARSHIP ELIGIBILITY
==================================================

Students eligible for scholarship (score > 75):

Alice - Score: 85
Charlie - Score: 78
Diana - Score: 92
```

Task 5: Online Shopping Cart Module

Prompt:

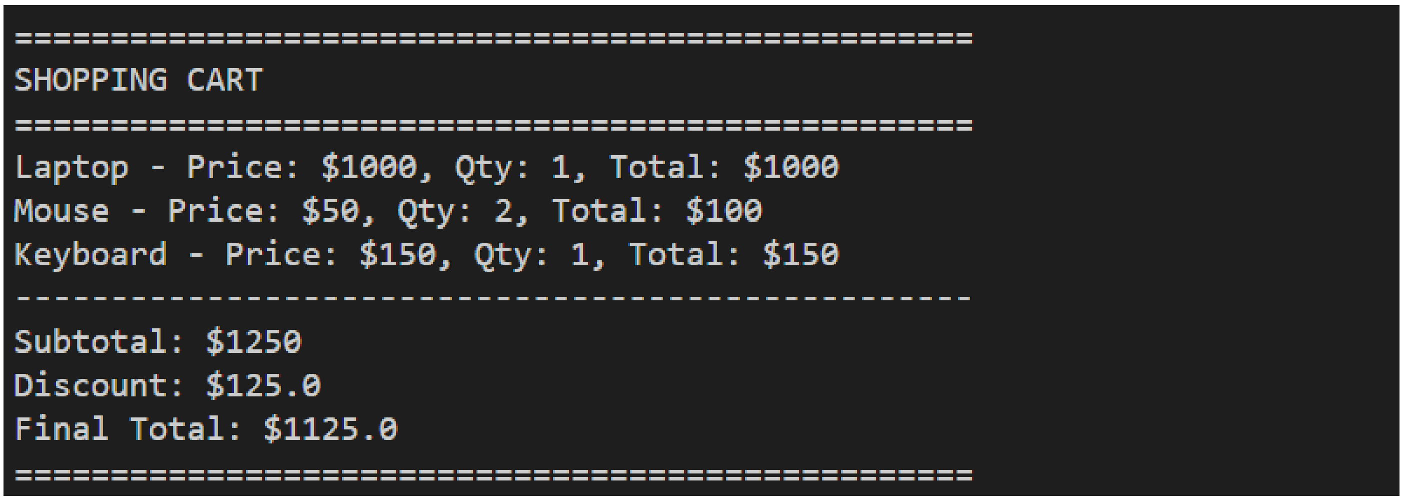
write a Python class named ShoppingCart with: An empty list to store items (each item may include name, price, quantity). And generate methods that: Add items to the cart, Remove items from the cart, Calculate the total bill using a loop, Apply conditional discounts (e.g., discount if total exceeds a certain amount)

Code:

```
# Shopping Cart for E-commerce Website
class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, name, price, quantity):
        self.items.append({"name": name, "price": price, 
"quantity": quantity})
        print(f"Added {quantity} x {name} to cart.")

    def remove_item(self, name):
        self.items = [item for item in self.items if 
item["name"] != name]
        print(f"Removed {name} from cart.")
```

```python
    def calculate_total(self):
        total = 0
        for item in self.items:
            total += item["price"] * item["quantity"]
        return total

    def apply_discount(self):
        total = self.calculate_total()
        discount = 0

        if total > 5000:
            discount = total * 0.20
        elif total > 3000:
            discount = total * 0.15
        elif total > 1000:
            discount = total * 0.10

        final_total = total - discount
        return total, discount, final_total

    def display_cart(self):
        print("\n" + "=" * 50)
        print("SHOPPING CART")
        print("=" * 50)

        for item in self.items:
            item_total = item["price"] * item["quantity"]
            print(f"{item['name']} - Price: ${item['price']},
Qty: {item['quantity']}, Total: ${item_total}")

        total, discount, final_total = self.apply_discount()
        print("-" * 50)
        print(f"Subtotal: ${total}")
        print(f"Discount: ${discount}")
        print(f"Final Total: ${final_total}")
        print("=" * 50 + "\n")


# Example usage
cart = ShoppingCart()
cart.add_item("Laptop", 1000, 1)
cart.add_item("Mouse", 50, 2)
cart.add_item("Keyboard", 150, 1)
cart.display_cart()
```

Output:



```
==================================================
SHOPPING CART
==================================================
Laptop - Price: $1000, Qty: 1, Total: $1000
Mouse - Price: $50, Qty: 2, Total: $100
Keyboard - Price: $150, Qty: 1, Total: $150
--------------------------------------------------
Subtotal: $1250
Discount: $125.0
Final Total: $1125.0
==================================================
```