

## ASSIGNMENT – 6.4

2303A51525

Batch-10

Task-1

Prompt: generate a Python class Student with name, roll\_number, and marks. Add a method to display details and another method to check if marks are above the class average using if-else. take user input

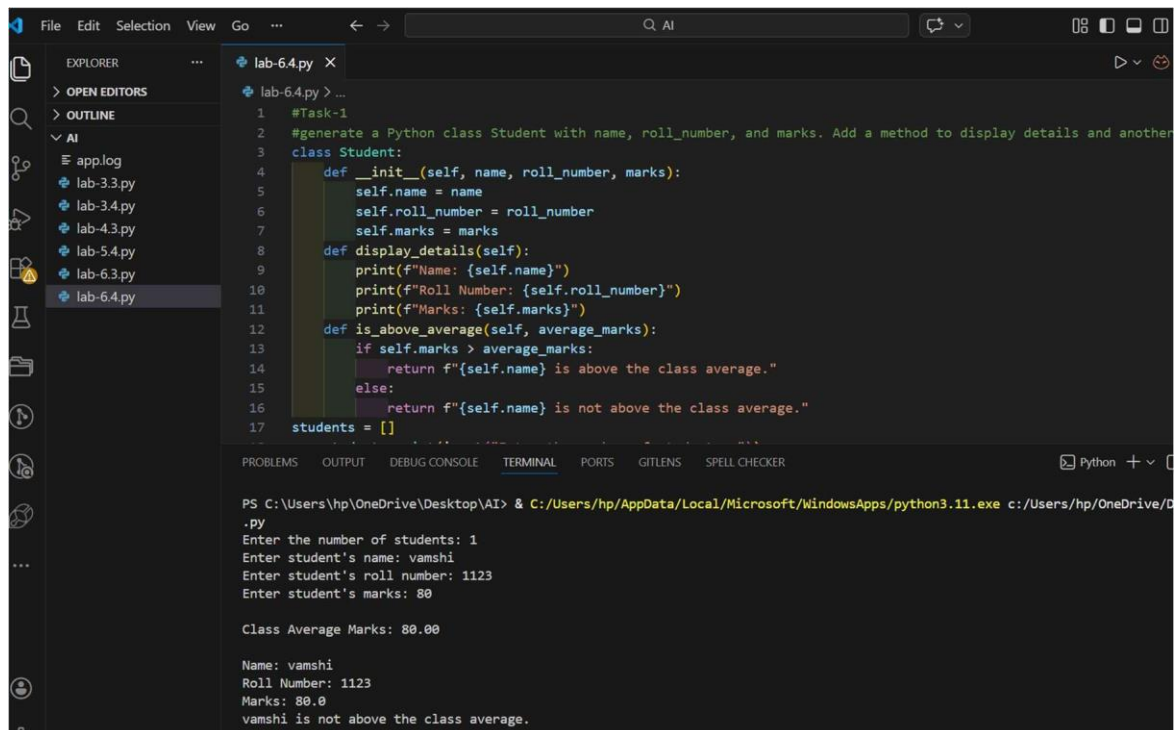
Code :

```
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 is_above_average(self, average_marks):
        if self.marks > average_marks:
            return f'{self.name} is above the class average.'
        else:
            return f'{self.name} is not above the class average.'

students = []
num_students = int(input("Enter the number of students: "))
for _ in range(num_students):
    name = input("Enter student's name: ")
    roll_number = input("Enter student's roll number: ")
    marks = float(input("Enter student's marks: "))
    students.append(Student(name, roll_number, marks))

total_marks = sum(student.marks for student in students)
average_marks = total_marks / num_students
print(f'\nClass Average Marks: {average_marks:.2f}\n')

for student in students:
    student.display_details()
print(student.is_above_average(average_marks))
print() Output :
```



```
1 #Task-1
2 #generate a Python class Student with name, roll_number, and marks. Add a method to display details and another
3 class Student:
4     def __init__(self, name, roll_number, marks):
5         self.name = name
6         self.roll_number = roll_number
7         self.marks = marks
8     def display_details(self):
9         print(f"Name: {self.name}")
10        print(f"Roll Number: {self.roll_number}")
11        print(f"Marks: {self.marks}")
12    def is_above_average(self, average_marks):
13        if self.marks > average_marks:
14            return f"{self.name} is above the class average."
15        else:
16            return f"{self.name} is not above the class average."
17
18 students = []
19
20 PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/D
21 .py
22 Enter the number of students: 1
23 Enter student's name: vamshi
24 Enter student's roll number: 1123
25 Enter student's marks: 80
26
27 Class Average Marks: 80.00
28
29 Name: vamshi
30 Roll Number: 1123
31 Marks: 80.0
32 vamshi is not above the class average.
```

### Code Analysis :

- Defines a Student class with name, roll number, and marks as attributes.
- Uses methods to display student details and compare marks with class average.
- Takes user input to create multiple student objects and stores them in a list.
- Calculates class average using total marks and number of students.
- Uses if-else to check whether a student is above average

### Task-2

Prompt: Write a for loop to iterate through sensor readings, identify even numbers using modulus operator, calculate their square, and print the result clearly and user input.

### Code :

```
sensor_readings = list(map(int, input("Enter sensor readings separated by spaces: ").split()))
for reading in sensor_readings:
    if reading % 2 == 0:
        square = reading ** 2
        print(f'Sensor Reading: {reading}, Square: {square}')
```

### Output :

```

27 for student in students:
28     student.display_details()
29     print(student.is_above_average(average_marks))
30     print()
31 """
32 #Task-2
33 # Write a for loop to iterate through sensor readings, identify even numbers using modulus operator, calculate their
34 sensor_readings = list(map(int, input("Enter sensor readings separated by spaces: ").split()))
35 for reading in sensor_readings:
36     if reading % 2 == 0:
37         square = reading ** 2
38         print(f"Sensor Reading: {reading}, Square: {square}")
39
40
41 #Task-3
42 #generate a Python class BankAccount with account_holder and balance. Add methods to deposit money, withdraw money,
43 """class BankAccount:
44     def __init__(self, account_holder, initial_balance=0):

```

```

PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop
.py
Enter sensor readings separated by spaces: 1 2 3 4 5 6 7 8
Sensor Reading: 2, Square: 4
Sensor Reading: 4, Square: 16
Sensor Reading: 6, Square: 36
Sensor Reading: 8, Square: 64

```

### Code Analysis :

- Takes multiple sensor readings as user input in a list format.
- Uses a for loop to iterate through each sensor reading.
- Applies the modulus operator (%) to identify even numbers.
- Calculates the square of even readings using the exponent operator.
- Prints the reading and its square in a clear format.

### Task-3

Prompt: generate a Python class Bank Account with account\_holder and balance. Add methods to deposit money, withdraw money, and prevent withdrawals when balance is insufficient using if-else user input.

Code :

```

class BankAccount:
    def __init__(self,
account_holder, initial_balance=0):
        self.account_holder =
account_holder
        self.balance =
initial_balance
    def deposit(self,
amount):
        if amount > 0:
            self.balance += amount
        print(f"Deposited: ${amount}")

```

```

        else:
            print("Deposit amount must be positive.")
def withdraw(self, amount):    if 0 < amount <=
self.balance:
    self.balance -= amount
print(f'Withdrew: ${amount}')
    else:
        print("Insufficient balance or invalid withdrawal amount.")
def check_balance(self):
    print(f'Current balance: ${self.balance}')
account_holder = input("Enter account holder's name: ")
initial_balance = float(input("Enter initial balance: "))
account = BankAccount(account_holder, initial_balance)
while True:
    action = input("Choose an action: deposit, withdraw, check balance, or exit: ").lower()
    if action == "deposit":
        amount = float(input("Enter amount to deposit: "))
        account.deposit(amount)    elif action == "withdraw":
        amount = float(input("Enter amount to withdraw: "))
        account.withdraw(amount)
        elif action == "check balance":
        account.check_balance()    elif
        action == "exit":
            print("Exiting the program.")    break
    else:    print("Invalid action. Please choose
again.") Output :

```

The screenshot shows a VS Code editor with a file named 'lab-6.4.py'. The code defines a 'BankAccount' class with methods for withdrawing, checking balance, depositing, and a main loop for user interaction. The terminal output shows the program running successfully, with user input for account holder name, initial balance, and actions like deposit and withdrawal.

```

43 class BankAccount:
44     def withdraw(self, amount):
45         if 0 < amount <= self.balance:
46             self.balance -= amount
47             print(f"Withdraw: ${amount}")
48         else:
49             print("Insufficient balance or invalid withdrawal amount.")
50     def check_balance(self):
51         print(f"Current balance: ${self.balance}")
52
53 account_holder = input("Enter account holder's name: ")
54 initial_balance = float(input("Enter initial balance: "))
55 account = BankAccount(account_holder, initial_balance)
56 while True:
57     action = input("Choose an action: deposit, withdraw, check balance, or exit: ").lower()
58     if action == "deposit":
59         amount = float(input("Enter amount to deposit: "))
60         account.deposit(amount)
61     elif action == "withdraw":
62         amount = float(input("Enter amount to withdraw: "))
63         account.withdraw(amount)
64     elif action == "check balance":
65         account.check_balance()
66     elif action == "exit":
67         break

```

Terminal Output:

```

PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive
.py
Enter account holder's name: vamshi
Enter initial balance: 100
Choose an action: deposit, withdraw, check balance, or exit: deposit
Enter amount to deposit: 100
Deposited: $100.0
Choose an action: deposit, withdraw, check balance, or exit: check balance
Current balance: $200.0
Choose an action: deposit, withdraw, check balance, or exit: withdraw
Enter amount to withdraw: 100
Withdraw: $100.0

```

## Code Analysis :

- Creates a BankAccount class with account holder name and balance.
- Includes methods for deposit, withdrawal, and balance checking.
- Uses if-else to prevent withdrawals with insufficient balance.
- Accepts user input through a menu-driven while loop.
- Ensures valid banking operations and safe program exit.

## Task-4

Prompt: **generate a code Using a list of student dictionaries with name and score, write a class ScholarshipEligibility:**

```

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.'"

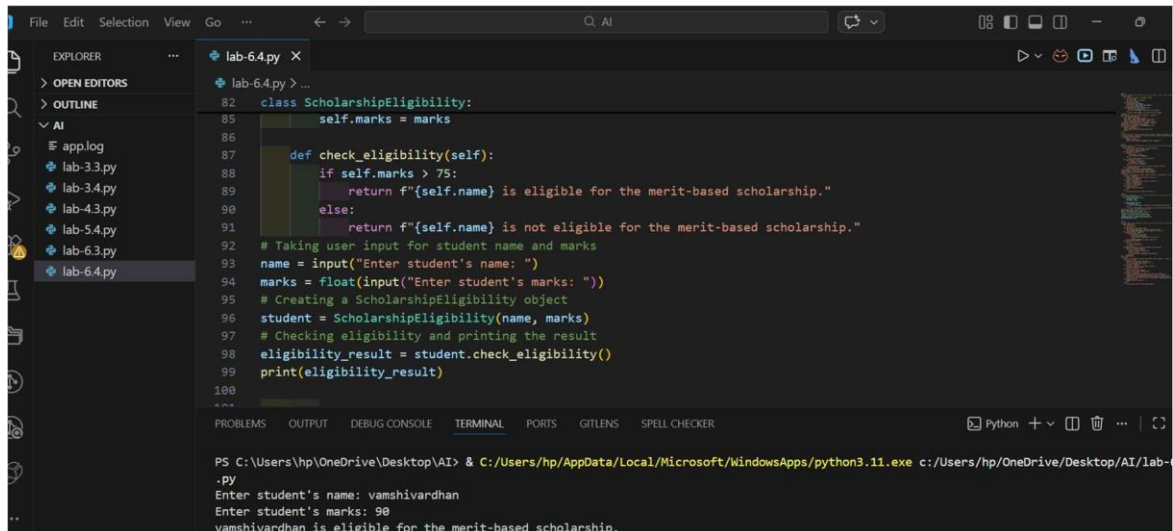
# Taking user input for student name and
marks name = input("Enter student's name: ")
marks = float(input("Enter student's marks: "))

# Creating a ScholarshipEligibility object
student = ScholarshipEligibility(name, marks)

# Checking eligibility and printing the result
eligibility_result = student.check_eligibility()
print(eligibility_result)

```

Output :



The screenshot shows a code editor with a file explorer on the left and a terminal at the bottom. The code in the editor is as follows:

```

82 class ScholarshipEligibility:
83     self.name = name
84     self.marks = marks
85
86     def check_eligibility(self):
87         if self.marks > 75:
88             return f'{self.name} is eligible for the merit-based scholarship.'"
89         else:
90             return f'{self.name} is not eligible for the merit-based scholarship.'"
91
92 # Taking user input for student name and marks
93 name = input("Enter student's name: ")
94 marks = float(input("Enter student's marks: "))
95 # Creating a ScholarshipEligibility object
96 student = ScholarshipEligibility(name, marks)
97 # Checking eligibility and printing the result
98 eligibility_result = student.check_eligibility()
99 print(eligibility_result)
100

```

The terminal output shows the execution of the program:

```

PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-6.4.py
Enter student's name: vamshivardhan
Enter student's marks: 90
vamshivardhan is eligible for the merit-based scholarship.

```

Code Analysis :

- Defines a class to store student name and marks.
- Takes user input for student details.
- Uses an if-else condition to check marks greater than 75.
- Determines eligibility for a merit-based scholarship.
- Displays the eligibility result clearly.

Task 5

Prompt: #Create a Python class Shopping Cart 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. class ShoppingCart: def \_\_init\_\_(self): self.items = [] def add\_item(self, item\_name, price):

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):

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):

total = 0 for item in

self.items: total +=

item["price"]

```

        return total
    def
    apply_discount(self,
    discount_threshold,
    discount_rate):
        total =
    self.calculate_total()
        if
    total >
    discount_threshold:
            discount = total *
    discount_rate
            total -= discount
    print(f"Discount      of
    ${discount:.2f} applied.")
        return total
    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)

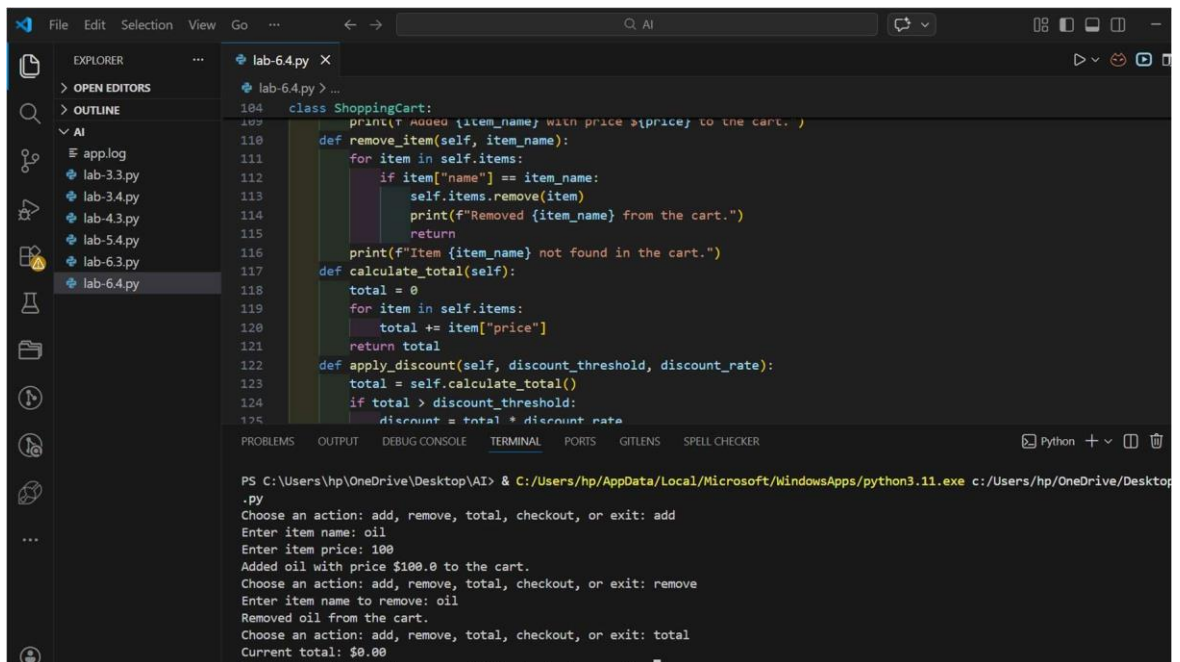
    elif action == "total":    total = cart.calculate_total()    print(f"Current total:
${total:.2f}")    elif action == "checkout":

        discount_threshold =
float(input("Enter discount
threshold: "))
discount_rate =
float(input("Enter discount
rate (as a decimal): "))
final_total =
cart.apply_discount(discou
nt_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 :



The screenshot shows a Visual Studio Code editor with a Python file named `lab-6.4.py`. The code defines a `ShoppingCart` class with the following methods:

- `add_item(self, item_name, price)`: Adds an item to the cart. It prints a confirmation message and increments the total price.
- `remove_item(self, item_name)`: Removes an item from the cart. It checks if the item exists and prints a confirmation message.
- `calculate_total(self)`: Calculates the total price of items in the cart.
- `apply_discount(self, discount_threshold, discount_rate)`: Applies a discount to the total price if it exceeds a specified threshold.

The terminal output shows the program's execution:

```
PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/.../lab-6.4.py
Choose an action: add, remove, total, checkout, or exit: add
Enter item name: oil
Enter item price: 100
Added oil with price $100.0 to the cart.
Choose an action: add, remove, total, checkout, or exit: remove
Enter item name to remove: oil
Removed oil from the cart.
Choose an action: add, remove, total, checkout, or exit: total
Current total: $0.00
```

### Code Analysis :

- Implements a `ShoppingCart` class to store items in a list.
- Provides methods to add and remove items from the cart.
- Uses a loop to calculate the total price of items.
- Applies a discount when total exceeds a user-defined threshold.
- Uses a while loop for continuous user interaction.