

## ASSIGNMENT-6.4

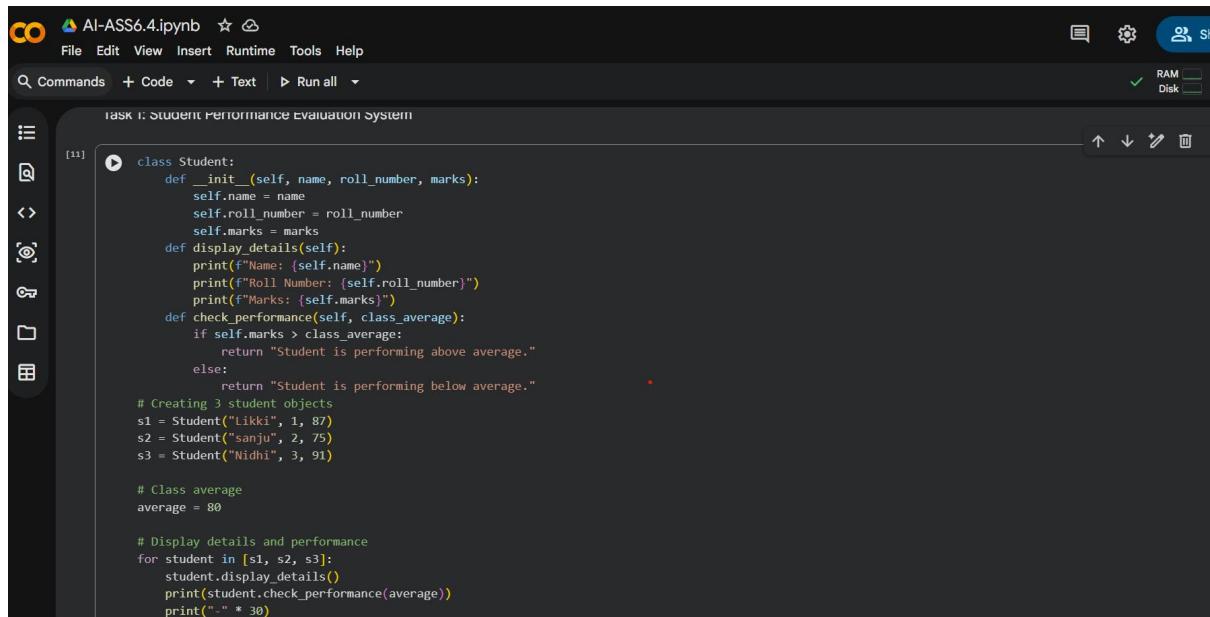
Name:-P.Likitha

HT.NO:-2303A52393

Batch:-43

### Task 1 – Student Performance Evaluation.

Write a method that displays all the student details using the class attributes. The method should print the name, roll number, and marks in a clear format. Then create another method that compares the student's marks with the class average. Use an if-else condition to decide whether the student is above or below average. Return a meaningful message based on this comparison.

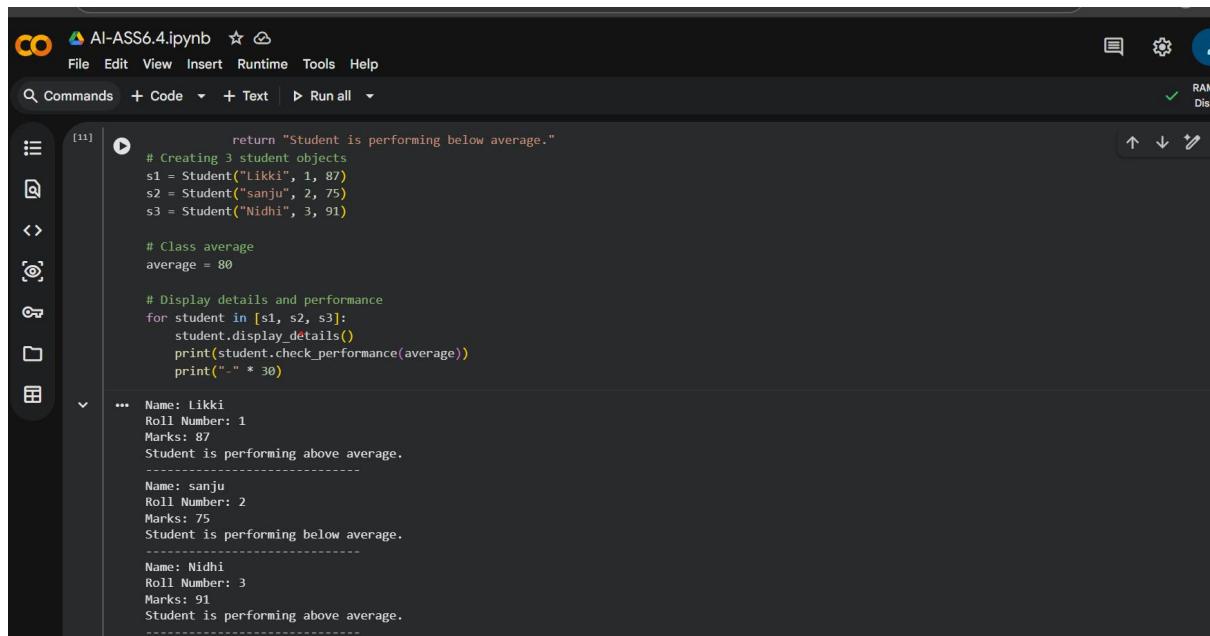


```
File Edit View Insert Runtime Tools Help
File Commands + Code + Text Run all
RAM Disk
[1]: class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks
    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")
    def check_performance(self, class_average):
        if self.marks > class_average:
            return "Student is performing above average."
        else:
            return "Student is performing below average."
# Creating 3 student objects
s1 = Student("Likki", 1, 87)
s2 = Student("sanju", 2, 75)
s3 = Student("Nidhi", 3, 91)

# Class average
average = 80

# Display details and performance
for student in [s1, s2, s3]:
    student.display_details()
    print(student.check_performance(average))
print("-" * 30)
```

OUTPUT:-



```
File Edit View Insert Runtime Tools Help
File Commands + Code + Text Run all
RAM Disk
[1]: ...
      return "Student is performing below average."
# Creating 3 student objects
s1 = Student("Likki", 1, 87)
s2 = Student("sanju", 2, 75)
s3 = Student("Nidhi", 3, 91)

# Class average
average = 80

# Display details and performance
for student in [s1, s2, s3]:
    student.display_details()
    print(student.check_performance(average))
print("-" * 30)
...
... Name: Likki
... Roll Number: 1
... Marks: 87
... Student is performing above average.
-----
... Name: sanju
... Roll Number: 2
... Marks: 75
... Student is performing below average.
-----
... Name: Nidhi
... Roll Number: 3
... Marks: 91
... Student is performing above average.
-----
```

### Task 2 – Data Processing in Monitoring System.

Inside the loop, add logic to identify whether the current number is even by using the modulus operator. If the number is even, calculate its square value. Print both the number and its square in a readable format. Make sure odd numbers are ignored. Use proper conditional statements to control the flow.

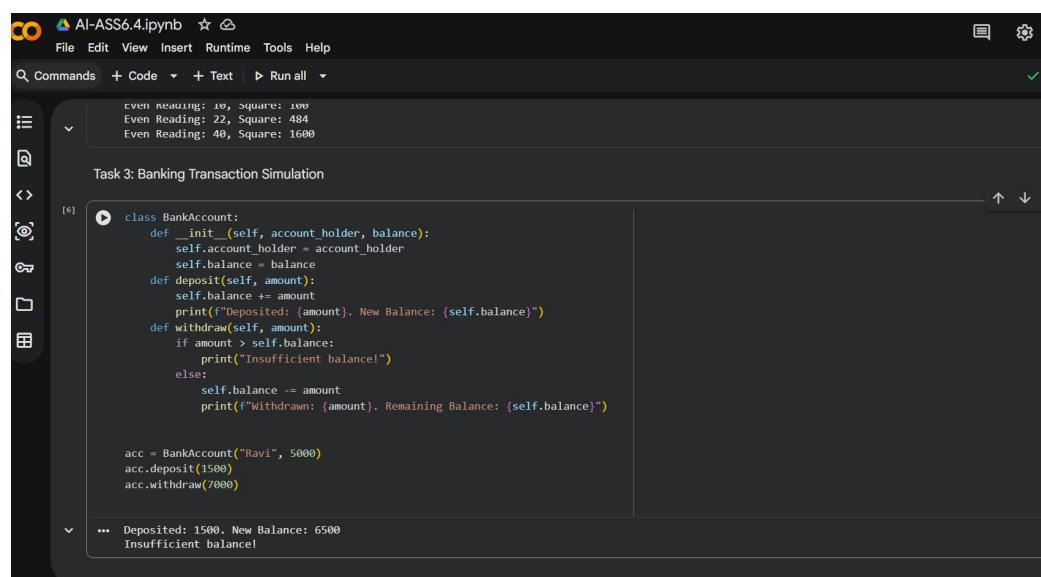
### Task 2: Data Processing in a Monitoring System

```
[1] 0s
      readings = [10, 15, 22, 33, 40]
      for value in readings:
          if value % 2 == 0:
              square = value ** 2
              print(f"Even Reading: {value}, Square: {square}")

      Even Reading: 10, Square: 100
      Even Reading: 22, Square: 484
      Even Reading: 40, Square: 1600
```

### Task 3 – Banking Transaction Simulation.

Create a deposit method that increases the account balance by the given amount. Also write a withdraw method that deducts money from the balance. Use an if-else condition to prevent withdrawal if the balance is insufficient. Display user-friendly messages after each transaction. Access all data using the class attributes with self.



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** AI-ASS6.4.ipynb, File, Edit, View, Insert, Runtime, Tools, Help.
- Toolbar:** Commands, + Code, + Text, Run all.
- Output Cell:** Shows the results of the previous code execution:

```
even reading: 10, square: 100
Even Reading: 22, Square: 484
Even Reading: 40, Square: 1600
```
- Code Cell:** Contains the Python code for the BankAccount class and its methods deposit and withdraw.

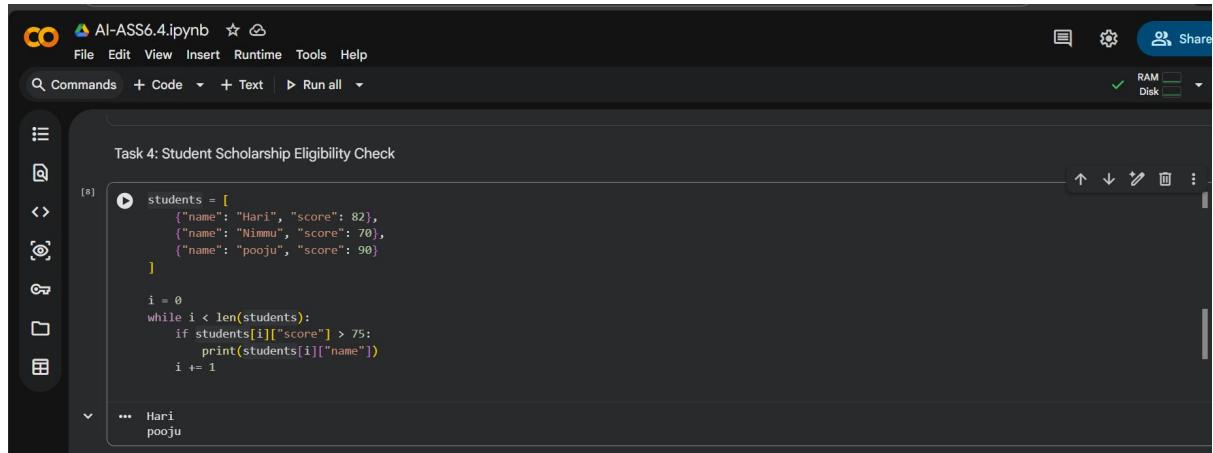
```
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance
    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited: {amount}. New Balance: {self.balance}")
    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient balance!")
        else:
            self.balance -= amount
            print(f"Withdrawn: {amount}. Remaining Balance: {self.balance}")

acc = BankAccount("Ravi", 5000)
acc.deposit(1500)
acc.withdraw(7000)
```
- Output of the code cell:** Shows the results of the new code execution:

```
... Deposited: 1500. New Balance: 6500
Insufficient balance!
```

#### Task 4 – Scholarship Eligibility Check.

Write a while loop that iterates through the list of student dictionaries using an index. Access each student's score and check if it is greater than 75. If the condition is satisfied, print the student's name as eligible for the scholarship. Ensure proper index increment and loop control. Format the output neatly.



The screenshot shows a Jupyter Notebook interface with the title "Task 4: Student Scholarship Eligibility Check". The code cell contains the following Python code:

```
students = [
    {"name": "Hari", "score": 82},
    {"name": "Nimmi", "score": 70},
    {"name": "pooju", "score": 90}
]

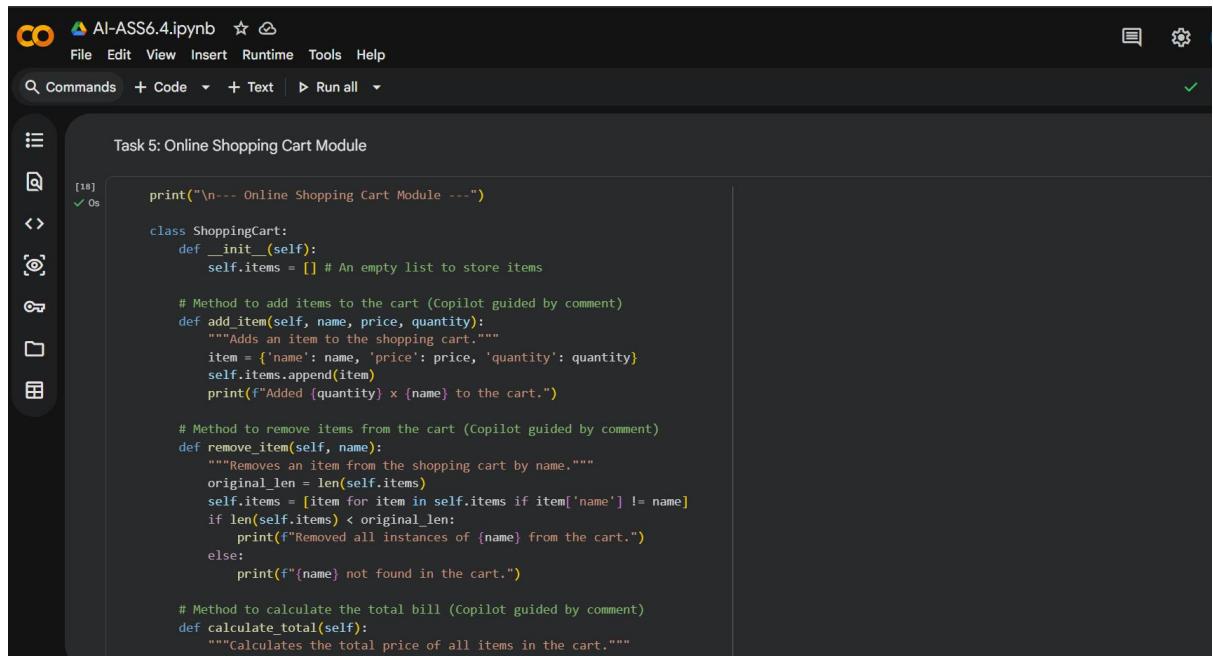
i = 0
while i < len(students):
    if students[i]["score"] > 75:
        print(students[i]["name"])
    i += 1
```

The output cell shows the results of the execution:

```
... Hari
pooju
```

#### Task 5 – Online Shopping Cart Module.

Develop methods to add items to the cart and remove items from it. Use a loop to calculate the total bill by multiplying price and quantity of each item. Add a condition to apply a discount if the total exceeds a specific limit. Return the final bill amount after discount. Make sure to use class attributes and conditional logic correctly.



The screenshot shows a Jupyter Notebook interface with the title "Task 5: Online Shopping Cart Module". The code cell contains the following Python code:

```
print("\n--- Online Shopping Cart Module ---")

class ShoppingCart:
    def __init__(self):
        self.items = [] # An empty list to store items

    # Method to add items to the cart (Copilot guided by comment)
    def add_item(self, name, price, quantity):
        """Adds an item to the shopping cart."""
        item = {'name': name, 'price': price, 'quantity': quantity}
        self.items.append(item)
        print(f"Added {quantity} x {name} to the cart.")

    # Method to remove items from the cart (Copilot guided by comment)
    def remove_item(self, name):
        """Removes an item from the shopping cart by name."""
        original_len = len(self.items)
        self.items = [item for item in self.items if item['name'] != name]
        if len(self.items) < original_len:
            print(f"Removed all instances of {name} from the cart.")
        else:
            print(f"{name} not found in the cart.")

    # Method to calculate the total bill (Copilot guided by comment)
    def calculate_total(self):
        """Calculates the total price of all items in the cart."""
        total_bill = sum(item['price'] * item['quantity'] for item in self.items)
        return total_bill
```

AI-ASS6.4.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

```
[18] ⏪ return total

# Method to apply conditional discounts (Copilot guided by comment)
def apply_discount(self, min_total_for_discount, discount_percentage):
    """Applies a discount if the total bill exceeds a certain amount."""
    current_total = self.calculate_total()
    if current_total >= min_total_for_discount:
        discount_amount = current_total * (discount_percentage / 100)
        discounted_total = current_total - discount_amount
        print(f"Discount of {discount_percentage}% applied! Saved ${discount_amount:.2f}.")
        return discounted_total
    else:
        print(f"No discount applied. Total ${current_total:.2f} is below minimum for discount ${min_total_for_discount:.2f}.")
        return current_total

def display_cart(self):
    """Displays all items currently in the cart."""
    if not self.items:
        print("Your cart is empty.")
        return
    print("\n--- Your Shopping Cart")
    for item in self.items:
        print(f" {item['name']} (Price: ${item['price']:.2f}, Quantity: {item['quantity']}) = ${item['price'] * item['quantity']:.2f}")
    print(f"Subtotal: ${self.calculate_total():.2f}")

...
--- Online Shopping Cart Module ---
```

AI-ASS6.4.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

```
[19] ✓ 0s ⏪ # --- Sample Usage ---
my_cart = ShoppingCart()

# Add items
my_cart.add_item("Laptop", 1200, 1)
my_cart.add_item("Mouse", 25, 2)
my_cart.add_item("Keyboard", 75, 1)
my_cart.display_cart()

# Calculate total before discount
initial_total = my_cart.calculate_total()
print(f"Cart total before discount: ${initial_total:.2f}")

# Try to apply discount (e.g., 10% off if total > $1000)
discounted_total = my_cart.apply_discount(1000, 10)
print(f"Cart total after discount: ${discounted_total:.2f}")

# Add more items to reach discount threshold if not met
my_cart.add_item("Monitor", 300, 1)
my_cart.display_cart()

discounted_total = my_cart.apply_discount(1000, 10)
print(f"Cart total after re-applying discount: ${discounted_total:.2f}")

# Remove an item
my_cart.remove_item("Mouse")
my_cart.display_cart()
```

AI-ASS6.4.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

```
[19] ✓ 0s ⏪ # Remove an item
my_cart.remove_item("Mouse")
my_cart.display_cart()

# Calculate final total after removal
final_total = my_cart.calculate_total()
print(f"Final cart total: ${final_total:.2f}")

# Attempt to remove a non-existent item
my_cart.remove_item("Webcam")
```

## Output:-

The screenshot shows a Jupyter Notebook interface with a dark theme. The code cell contains the command `my_cart.remove_item("Webcam")`. The output pane displays the execution results, which include three distinct shopping cart sessions. Each session starts with the message "... Added 1 x Laptop to the cart. Added 2 x Mouse to the cart. Added 1 x Keyboard to the cart.". The first session shows a subtotal of \$1325.00, a discount of 10% applied (saving \$132.50), and a final total of \$1192.50 after adding a monitor. The second session shows a subtotal of \$1625.00, a discount of 10% applied (saving \$162.50), and a final total of \$1462.50 after removing all instances of Mouse from the cart. The third session shows a subtotal of \$1575.00, a final cart total of \$1575.00, and a message stating "Webcam not found in the cart."

```
my_cart.remove_item("Webcam")
...
... Added 1 x Laptop to the cart.
Added 2 x Mouse to the cart.
Added 1 x Keyboard to the cart.

--- Your Shopping Cart ---
Laptop (Price: $1200.00, Quantity: 1) = $1200.00
Mouse (Price: $25.00, Quantity: 2) = $50.00
Keyboard (Price: $75.00, Quantity: 1) = $75.00
-----
Subtotal: $1325.00
Cart total before discount: $1325.00
Discount of 10% applied! Saved $132.50.
Cart total after discount: $1192.50
Added 1 x Monitor to the cart.

--- Your Shopping Cart ---
Laptop (Price: $1200.00, Quantity: 1) = $1200.00
Mouse (Price: $25.00, Quantity: 2) = $50.00
Keyboard (Price: $75.00, Quantity: 1) = $75.00
Monitor (Price: $300.00, Quantity: 1) = $300.00
-----
Subtotal: $1625.00
Discount of 10% applied! Saved $162.50.
Cart total after re-applying discount: $1462.50
Removed all instances of Mouse from the cart.

--- Your Shopping Cart ---
Laptop (Price: $1200.00, Quantity: 1) = $1200.00
Keyboard (Price: $75.00, Quantity: 1) = $75.00
Monitor (Price: $300.00, Quantity: 1) = $300.00
-----
Subtotal: $1575.00
Final cart total: $1575.00
Webcam not found in the cart.
```