Lab:6

Name:B.YASHWANTH KUMAR

Roll Number: 2503A51L42

Course Code: 24CS002PC215

Course Title: AI Assisted Coding

Assignment Number: 6

Academic Year: 2025-2026

TASK1:

Start a Python class named Student with attributes name, roll_number, and marks. Prompt GitHub Copilot to complete methods for displaying details and checking if marks are above average.

PROMPT:

Start a Python class named Student with attributes name, roll_number, and marks. Prompt GitHub Copilot to complete methods for displaying details and checking if marks are above average.

CODE:

```python
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=50):
        """Check if marks are above the given average (default 50)."""
        return self.marks > average

if __name__ == "__main__":
    student1 = Student("Ramsai","L53", 100)
    student1.display_details()
    print("Above average:", student1.is_above_average())
```

OUTPUT:

```
PS C:\Users\ramsa> & C:/Users/ramsa/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/ramsa/OneDrive/Documents/ai assistant lab/ailab6.1.py"
Name: Ramsai
Roll Number: L53
Marks: 100
Above average: True
```

OBSERVATION:

1. The Student class allows easy creation of student objects with name, roll number, and marks.

2. The display_details() method prints all student information in a readable format.

3. The is_above_average() method checks if the student's marks are above the default average (50), making it flexible for other averages too.

4. The output clearly shows the student's details and whether their marks are above average.

5. The code is easy to understand and modify for more features, such as adding more attributes or methods.

6. Using classes makes the code organized and reusable for multiple students.

TASK2: Write the first two lines of a for loop to iterate through a list of numbers. Use a comment ,prompt to let Copilot suggest how to calculate and print the square of even numbers only.

PROMPT: Write the first two lines of a for loop to iterate through a list of numbers. Use a comment ,prompt to let Copilot suggest how to calculate and print the square of even numbers only.

CODE:

```
numbers = [1, 2, 3, 4, 5, 6]
for number in numbers:
    # Calculate and print the square of even numbers only
    if number % 2 == 0:
        print(f"{number} squared is {number ** 2}")
```

OUTPUT:

```
b6.2.py"
2 squared is 4
4 squared is 16
6 squared is 36
```

OBSERVATION:

1. The code is well-structured and easy to follow, making it simple to understand the logic.

2. The use of loops and conditionals helps automate repetitive tasks and decision-making.

3. The output is clear and provides direct feedback based on the input and logic implemented.

4. The program can be easily modified to handle more cases or different conditions.

TASK3: Create a class called BankAccount with attributes account_holder and balance. Use Copilot to complete methods for deposit(), withdraw(), and check for insufficient balance.

PROMPT: Create a class called BankAccount with attributes account_holder and balance. Use Copilot to complete methods for deposit(), withdraw(), and check for insufficient balance.

CODE:

```
portfolio.html   ailab3.py   ailab3.4.py 1 ●   ailab3.5.py   ailab6.1.py   ailab6.2.py   ailab6.3.py ×   Untitled-1   ai_lab2.py   ai_lab3.py
C: > Users > ramsa > OneDrive > Documents > ai assistant lab > ailab6.3.py > ...
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 > self.balance:
            print("Insufficient balance. Withdrawal denied.")
        elif amount > 0:
            self.balance -= amount
            print(f"Withdrew {amount}. New balance: {self.balance}")
        else:
            print("Withdrawal amount must be positive.")

# Example usage:
account = BankAccount("John Doe", 100)
account.deposit(50)
account.withdraw(30)
```

OUTPUT:

```
PS C:\Users\ramsa> & C:/Users/ramsa/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/ramsa/OneDrive/Documents/ai assistant lab/ailab6.3.py"
Deposited 50. New balance: 150
Withdrew 30. New balance: 120
```

OBSERVATION:

1. The BankAccount class uses attributes to store account holder information and balance, making account management organized.

2. The deposit() method correctly increases the balance and provides feedback for invalid amounts.

3. The withdraw() method checks for sufficient balance before allowing withdrawal, preventing overdrawing and teaching good error handling.

4. The output messages for deposits and withdrawals are clear, helping users understand each transaction's result.

5. The class structure allows easy creation and management of multiple accounts, demonstrating object-oriented programming principles.

6. Example usage shows how to interact with the class and verify its behavior, making the code practical and easy to test.

TASK4: Define a list of student dictionaries with keys name and score. Ask Copilot to write a while loop to print the names of students who scored more than 75.

PROMPT: Define a list of student dictionaries with keys name and score. Ask Copilot to write a while loop to print the names of students who scored more than 75.

CODE:

```
rtfolio.html   ailab3.py   ailab3.4.py 1   ailab3.5.py   ailab6.1.py   ailab6.2.py
C: > Users > ramsa > OneDrive > Documents > ai assistant lab > ailab6.4.py > ...
students = [
    {"name": "Alice", "score": 82},
    {"name": "Bob", "score": 74},
    {"name": "Charlie", "score": 90},
    {"name": "Diana", "score": 65},
    {"name": "Eve", "score": 78}
]

# Print names of students who scored more than 75 using a while loop
index = 0
while index < len(students):
    if students[index]["score"] > 75:
        print(f"{students[index]['name']} scored {students[index]['score']}")
    index += 1
```

OUTPUT:

```
.4.py"
Alice scored 82
Charlie scored 90
Eve scored 78
PS C:\Users\ramsa> & C:/Users/ramsa/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/ramsa/OneDrive/Documents/ai assistant lab/ailab6
```

OBSERVATION:

1. The code uses a list of dictionaries to store student names and scores, making data management simple and organized.

2. The while loop iterates through each student and checks if their score is above 75, demonstrating practical use of loops and conditionals.

3. The output clearly lists only those students who meet the score criteria, making results easy to understand.

4. The code can be easily modified to change the score threshold or add more students.

5. This approach helps students learn how to filter data and display relevant information efficiently.

TASK5: Begin writing a class ShoppingCart with an empty items list. Prompt Copilot to generate methods to add_item, remove_item, and use a loop to calculate the total bill using conditional discounts.

PROMPT: Begin writing a class ShoppingCart with an empty items list. Prompt Copilot to generate methods to add_item, remove_item, and use a loop to calculate the total bill using conditional discounts.

CODE:

```python
class ShoppingCart:
    def __init__(self):
        self.items = []  # Each item is a dict: {"name": str, "price": float}

    def add_item(self, name, price):
        self.items.append({"name": name, "price": price})

    def remove_item(self, name):
        self.items = [item for item in self.items if item["name"] != name]

    def total_bill(self):
        total = 0
        for item in self.items:
            total += item["price"]
        # Apply discount: 10% off if total > 100, else no discount
        if total > 100:
            print("Discount applied: 10% off")
            total *= 0.9
        print(f"Total bill: {total:.2f}")
        return total

if __name__ == "__main__":
    cart = ShoppingCart()
    cart.add_item("Shoes", 60)
    cart.add_item("Bag", 50)
    cart.add_item("Pen", 5)
    cart.remove_item("Pen")
    cart.total_bill()
```

OUTPUT:

```
.5.py"
Discount applied: 10% off
Total bill: 99.00
PS C:\Users\ramsa> & C:/Users/ramsa/AppData/Local/Microsoft/WindowsApps/python3.11.exe
```

OBSERVATION:

1. The ShoppingCart class uses a list to manage items, making it easy to add and remove products.

2. The add_item and remove_item methods allow flexible item management and demonstrate good use of class methods.

3. The total_bill method uses a loop to calculate the total and applies a conditional discount, showing practical use of if-else logic.

4. The output clearly displays the total bill and any discount applied, making it easy to understand the result of each operation.

5. The code is organized and can be easily extended to support more features, such as multiple discounts or item quantities.

6. This example helps students learn how to combine data structures, loops, and conditionals in a real-world scenario.