LAB ASSIGNMENT 6.4

NAME: KOLIPAKA SANJANA

ROLL.NO:2303A52139

BATCH:41

TASK1: Student Performance Evaluation System

CODE:

```python
#create a student class and write a method display to display the student details that prints name,roll number and m
#write a method check_performance that calculates the average marks of the class and compares the student's marks
#Take 5 students data with average_marks and returns a message using if-else statements
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

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

    def check_performance(self, average_marks):
        if self.marks > average_marks:
            return f"{self.name} is above average."
        elif self.marks < average_marks:
            return f"{self.name} is below average."
        else:
            return f"{self.name} is at average."
# Taking data for 5 students
students = [
    Student("Alice", 1, 85),
    Student("Bob", 2, 70),
    Student("Charlie", 3, 90),
    Student("David", 4, 60),
    Student("Eve", 5, 75)
]
# Calculating average marks
total_marks = sum(student.marks for student in students)
average_marks = total_marks / len(students)
# Displaying student details and performance
for student in students:
    student.display()
    print(student.check_performance(average_marks))
```

Sanjana Kolipaka (sanjanakolipaka1112@gmail.com) is signed in

OUTPUT:

```
PS D:\AI assistance coding>  & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\.vs
code\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51142' '--' 'D:\AI assistance co
ding\2303a52139(Assignment6.3).py'
Name: Alice, Roll Number: 1, Marks: 85
Alice is above average.
Name: Bob, Roll Number: 2, Marks: 70
Bob is below average.
Name: Charlie, Roll Number: 3, Marks: 90
Charlie is above average.
Name: David, Roll Number: 4, Marks: 60
David is below average.
Name: Eve, Roll Number: 5, Marks: 75
Eve is below average.
```

JUSTIFICATION:

This task introduces object-oriented programming (OOP) fundamentals using a real academic scenario.

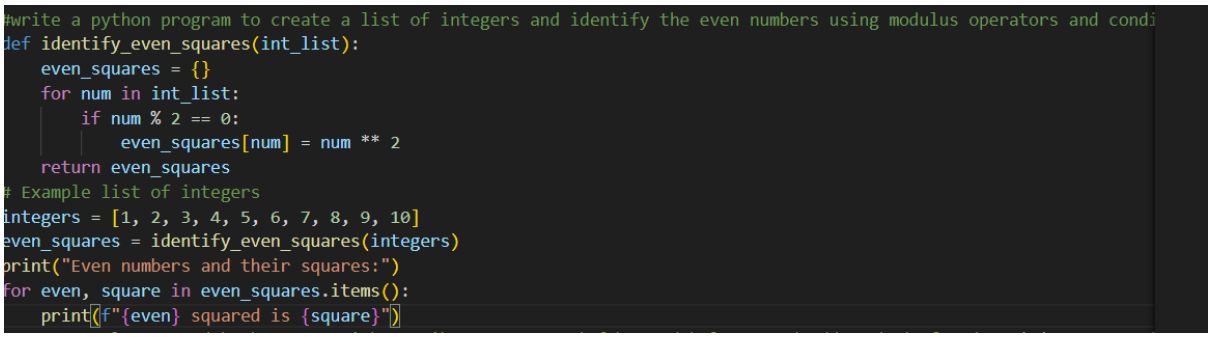
It helps students understand:

- How to define a class structure in Python.
- How to initialize attributes using self.
- How to use conditional logic to compare student marks with the class average.
- How GitHub Copilot can assist in completing repetitive logic using comments as prompts.

By combining manual class creation with Copilot completion, the task trains students to effectively use AI-assisted coding while still understanding core Python concepts.

TASK2: Data Processing in a Monitoring System

CODE:



```
#write a python program to create a list of integers and identify the even numbers using modulus operators and condi
def identify_even_squares(int_list):
    even_squares = {}
    for num in int_list:
        if num % 2 == 0:
            even_squares[num] = num ** 2
    return even_squares
# Example list of integers
integers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_squares = identify_even_squares(integers)
print("Even numbers and their squares:")
for even, square in even_squares.items():
    print(f"{even} squared is {square}")
```

OUTPUT:

```
1.exe' 'c:\Users\sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63885' '--' 'D
:\AI assistance coding\2303a52139(Assignment6.3).py'
Even numbers and their squares:
2 squared is 4
4 squared is 16
6 squared is 36
8 squared is 64
10 squared is 100
```

Justification

This task focuses on looping and condition checking in a practical data-processing context.

It allows learners to practice:

- Iterating through a list using a for loop.
- Using the modulus operator (%) to detect even numbers.
- Applying conditions to filter data.
- Performing calculations (square of even numbers).
- Printing formatted outputs.

Using Copilot with comments teaches how developers can speed up logic creation while maintaining control over program flow.

TASK3:

CODE:

```python
#create a class named bankaccount with attributes account_holder and balance and add methods for depositing money,withdrawing money and preventing withdrawal when the ba
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 for withdrawal.")
        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", 1000)
account.deposit(500)
account.withdraw(200)
account.withdraw(2000)
account.deposit(-100)
account.withdraw(-50)
```

OUTPUT:

```
-2025.18.0-win32-x64\bundled\libs\debugpy\laun
Deposited: 500. New balance: 1500
Withdrew: 200. New balance: 1300
Insufficient balance for withdrawal.
Deposit amount must be positive.
Withdrawal amount must be positive.
```

Justification

This task simulates a real-world banking system to reinforce OOP with state management.

It strengthens:

- Understanding of class attributes and methods.
- Handling deposits and withdrawals logically.
- Using if-else conditions to prevent invalid operations.
- Writing user-friendly error handling.

It also demonstrates how Copilot can generate business logic safely when guided with meaningful method names and comments.

TASK4:

CODE:

```python
#create a list of dictionaries where each dictionary represents a student with name,score and generate a wwhile loop to iterate through the list and print name of studen
students = [
    {"name": "Alice", "score": 85},
    {"name": "Bob", "score": 65},
    {"name": "Charlie", "score": 90},
    {"name": "David", "score": 55},
    {"name": "Eve", "score": 78}

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

OUTPUT:

```
Alice
Charlie
Eve
```

Justification

This task focuses on data structures and control flow using while loops.

It enables learners to:

- Work with lists of dictionaries.
- Use indexing inside a while loop.
- Apply conditions to filter records.
- Print selected outputs cleanly.

The use of Copilot here highlights how AI can assist in building loop logic while the user controls the dataset and decision rules.

TASK5:

CODE:

```python
#create a class named shoppingcart with an empty list to store items(each  item may include name,price,quantity).Add methods to add items to the cart,remove items from th
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} of {name} at ${price} each to the cart.")

    def remove_item(self, name):
        for item in self.items:
            if item["name"] == name:
                self.items.remove(item)
                print(f"Removed {name} from the cart.")
                return
        print(f"{name} not found in the cart.")

    def calculate_total(self):
        total = sum(item["price"] * item["quantity"] for item in self.items)
        if total > 100:
            discount = total * 0.1  # 10% discount
            total -= discount
            print(f"Applied a discount of ${discount:.2f}.")
        print(f"Total price of items in the cart: ${total:.2f}")
        return total
# Example usage
cart = ShoppingCart()
cart.add_item("Laptop", 800, 1)
cart.add_item("Mouse", 20, 2)
cart.add_item("Keyboard", 50, 1)
cart.calculate_total()
cart.remove_item("Mouse")
cart.calculate_total()
```

OUTPUT:

```
Added 1 of Laptop at $800 each to the cart.
Added 2 of Mouse at $20 each to the cart.
Added 1 of Laptop at $800 each to the cart.
Added 2 of Mouse at $20 each to the cart.
Added 2 of Mouse at $20 each to the cart.
Added 1 of Keyboard at $50 each to the cart.
Applied a discount of $89.00.
Total price of items in the cart: $801.00
Removed Mouse from the cart.
Applied a discount of $89.00.
Total price of items in the cart: $801.00
Removed Mouse from the cart.
Removed Mouse from the cart.
Applied a discount of $85.00.
Total price of items in the cart: $765.00
PS D:\AI assistance coding>
```

Justification

This task integrates multiple programming concepts into a realistic e-commerce system.

It develops skills in:

- Managing collections using lists.
- Designing modular classes.
- Writing loops for calculations.
- Applying conditional discounts.
- Handling add/remove operations logically.

This task also shows how Copilot supports building full modules with minimal code, encouraging efficient development while preserving correctness.