

2303A51563

batch=10

ask 1: Student Performance Evaluation System

Scenario

You are building a simple academic management module for a university

system where student performance needs to be evaluated automatically.

Task Description

Create the skeleton of a Python class named `Student` with the attributes:

- `name`
- `roll_number`
- `marks`

Write only the class definition and attribute initialization.

Then, using GitHub Copilot, prompt the tool to complete:

- 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

Use comments or partial method names to guide Copilot for code

completion.

Expected Outcome

- A completed `Student` class with Copilot-generated methods
- Proper use of:

- o self attributes
- o Conditional statements (if-else)
 - Sample output showing student details and performance status

```

class Student:
    def __init__(self, name, roll_number, marks):
        # Attribute initialization
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    # Copilot Prompt: Complete a method to display student details
    def display_details(self):
        print("Student Details:")
        print("Name:", self.name)
        print("Roll Number:", self.roll_number)
        print("Marks:", self.marks)

    # Copilot Prompt: Complete a method to check performance above class average
    def check_performance(self, class_average):
        # If marks are greater than class average return message
        if self.marks > class_average:
            return "Performance Status: Above Class Average ✓"
        else:
            return "Performance Status: Below Class Average ✗"

# -----
# Example Usage (Sample Output)
# -----


# Creating student objects
student1 = Student("Akshaya", 101, 85)

# Class average marks
average_marks = 75

# Display student details
student1.display_details()

```

```
# Check performance
result = student1.check_performance(average_marks)
print(result)
```

The screenshot shows the OnlineGDB interface with a Python script named `main.py`. The code defines a `Student` class with `__init__`, `display_details`, and `check_performance` methods. It prints student details and checks if marks are above or below average. The output window shows the student details and the performance status as "Above Class Average" with a green checkmark.

```
1 # Student Performance Evaluation System
2
3 class Student:
4     def __init__(self, name, roll_number, marks):
5         # Attribute initialization
6         self.name = name
7         self.roll_number = roll_number
8         self.marks = marks
9
10    # Copilot Prompt: Complete a method to display student details
11    def display_details(self):
12        print("Student Details:")
13        print("Name:", self.name)
14        print("Roll Number:", self.roll_number)
15        print("Marks:", self.marks)
16
17    # Copilot Prompt: Complete a method to check performance above class average
18    def check_performance(self, class_average):
19        # If marks are greater than class average return message
20        if self.marks > class_average:
21            return "Performance Status: Above Class Average ✅"
22        else:
23            return "Performance Status: Below Class Average ❌"
24
25
26 # -----
27 # Example Usage (Sample Output)
28 #
29
30 # Creating student objects
31 student1 = Student("Akshaya", 101, 85)
32
33 # Class average marks
34 average_marks = 75
35
36 # Display student details
37 student1.display_details()
38
39 # Check performance
40 result = student1.check_performance(average_marks)
41 print(result)
```

Output:

```
Student Details:
Name: Akshaya
Roll Number: 101
Marks: 85
Performance Status: Above Class Average ✅
...Program finished with exit code 0
Press ENTER to exit console.
```

The screenshot shows the OnlineGDB interface with the same Python script `main.py`, but with a different implementation of the `check_performance` method. It still prints student details and shows the performance status as "Above Class Average" with a green checkmark.

```
15    # If marks are greater than class average return message
16    if self.marks > class_average:
17        return "Performance Status: Above Class Average ✅"
18    else:
19        return "Performance Status: Below Class Average ❌"
20
21
22 # -----
23 # Example Usage (Sample Output)
24 #
25
26 # Creating student objects
27 student1 = Student("Akshaya", 101, 85)
28
29 # Class average marks
30 average_marks = 75
31
32 # Display student details
33 student1.display_details()
34
35 # Check performance
36 result = student1.check_performance(average_marks)
37 print(result)
```

Output:

```
Student Details:
Name: Akshaya
Roll Number: 101
Marks: 85
Performance Status: Above Class Average ✅
...Program finished with exit code 0
Press ENTER to exit console.
```

Task 2: Data Processing in a Monitoring System

Scenario

You are working on a basic data monitoring script where sensor readings are collected as numbers. Only even readings need further processing.

Task Description

Write the initial part of a for loop to iterate over a list of integers representing sensor readings.

Add a comment prompt instructing GitHub Copilot to:

- Identify even numbers
- Calculate their square
- Print the result in a readable format

Allow Copilot to complete the remaining loop logic.

Expected Outcome

- A complete for loop generated by Copilot
- Use of:
 - Modulus operator to identify even numbers
 - Conditional statements
- Correct and formatted output for valid inputs

Data Processing in a Monitoring System

CODE

```
# List of sensor readings (integers)
sensor_readings = [12, 7, 9, 20, 15, 8, 3, 14]
```

```
# Iterate through each reading
for reading in sensor_readings:
```

```
# Copilot Prompt:
# 1. Check if the reading is an even number using modulus operator (%)
# 2. If it is even, calculate its square
# 3. Print the reading and its squared value in a readable format
```

```
if reading % 2 == 0:
    squared_value = reading ** 2
```

```

print(f"Even Reading: {reading} -> Square: {squared_value}")

```

```

1 # Data Processing in a Monitoring System
2
3 # List of sensor readings (integers)
4 sensor_readings = [12, 7, 9, 20, 15, 8, 3, 14]
5
6 # Iterate through each reading
7 for reading in sensor_readings:
8
9     # Copilot Prompt:
10    # 1. Check if the reading is an even number using modulus operator (%)
11    # 2. If it is even, calculate its square
12    # 3. Print the reading and its squared value in a readable format
13
14    if reading % 2 == 0:
15        squared_value = reading ** 2
16        print(f"Even Reading: {reading} -> Square: {squared_value}")
17

```

```

Even Reading: 12 -> Square: 144
Even Reading: 20 -> Square: 400
Even Reading: 8 -> Square: 64
Even Reading: 14 -> Square: 196
...Program finished with exit code 0
Press ENTER to exit console.

```

task 3: Banking Transaction Simulation

Scenario

You are developing a basic banking module that handles deposits and withdrawals for customers.

Task Description

Create the structure of a Python class named BankAccount with attributes:

- account_holder
- balance

Use GitHub Copilot to complete methods for:

- Depositing money
- Withdrawing money

Code

```
# Banking Transaction Simulation
```

```

class BankAccount:
    def __init__(self, account_holder, balance):
        # Initialize account holder name and starting balance
        self.account_holder = account_holder
        self.balance = balance

    # Copilot Prompt: Complete method to deposit money
    def deposit(self, amount):

```

```
# Add the deposit amount to balance
# Print a confirmation message with updated balance
pass

# Copilot Prompt: Complete method to withdraw money
def withdraw(self, amount):
    # Check if sufficient balance is available
    # If yes, subtract amount and print success message
    # Otherwise, print insufficient balance message
    pass

# Example Usage
account1 = BankAccount("Akshaya", 5000)

account1.deposit(2000)
account1.withdraw(3000)
account1.withdraw(6000)
```

The screenshot shows the OnlineGDB interface with a Python script named 'main.py' open. The code defines a 'BankAccount' class with methods for deposit and withdraw. It includes a copilot prompt for completing the deposit method. The script then creates an account and demonstrates its usage.

```
1 # Banking Transaction Simulation
2
3 class BankAccount:
4     def __init__(self, account_holder, balance):
5         # Initialize account holder name and starting balance
6         self.account_holder = account_holder
7         self.balance = balance
8
9     # Copilot Prompt: Complete method to deposit money
10    def deposit(self, amount):
11        # Add the deposit amount to balance
12        # Print a confirmation message with updated balance
13        pass
14
15    # Copilot Prompt: Complete method to withdraw money
16    def withdraw(self, amount):
17        # Check if sufficient balance is available
18        # If yes, subtract amount and print success message
19        # Otherwise, print insufficient balance message
20        pass
21
22
23 # Example Usage
24 account1 = BankAccount("Akshaya", 5000)
25
26 account1.deposit(2000)
27 account1.withdraw(3000)
28 account1.withdraw(6000)
```

...Program finished with exit code 0
Press ENTER to exit console.

The screenshot shows the OnlineGDB interface with the same Python script 'main.py'. This time, the deposit method has been completed by the user, and the withdraw method is now highlighted with a copilot prompt. The script runs successfully, demonstrating the updated balance after deposits and withdrawals.

```
1 self.account_holder = account_holder
2 self.balance = balance
3
4 # Copilot Prompt: Complete method to deposit money
5 def deposit(self, amount):
6     # Add the deposit amount to balance
7     # Print a confirmation message with updated balance
8     pass
9
10 # Copilot Prompt: Complete method to withdraw money
11 def withdraw(self, amount):
12     # Check if sufficient balance is available
13     # If yes, subtract amount and print success message
14     # Otherwise, print insufficient balance message
15     pass
16
17
18 # Example Usage
19 account1 = BankAccount("Akshaya", 5000)
20
21 account1.deposit(2000)
22 account1.withdraw(3000)
23 account1.withdraw(6000)
```

...Program finished with exit code 0
Press ENTER to exit console.

Task 4: Student Scholarship Eligibility Check

Scenario

A university wants to identify students eligible for a merit-based scholarship based on their scores.

Task Description

Define a list of dictionaries where each dictionary represents a student with:

- name
- score

Write the initialization and list structure yourself.

Then, prompt GitHub Copilot to generate a while loop that:

- Iterates through the list
- Prints the names of students who scored more than 75

Use comments to guide Copilot's code completion.

Expected Outcome

- A complete while loop generated by Copilot
- Correct index handling and condition checks
- Cleanly formatted output listing eligible students

Code

```
# Student Scholarship Eligibility Check
```

```
# List of students (each student is represented as a dictionary)
```

```
students = [  
    {"name": "Akshaya", "score": 82},  
    {"name": "Ravi", "score": 68},  
    {"name": "Sita", "score": 91},  
    {"name": "Kiran", "score": 74},  
    {"name": "Meena", "score": 88}  
]
```

```
# Copilot Prompt:
```

```
# Write a while loop that:
```

```
# 1. Iterates through the students list using an index  
# 2. Checks if the student's score is greater than 75  
# 3. Prints the student's name if eligible for scholarship
```

```
index = 0
```

```
while index < len(students):  
    if students[index]["score"] > 75:  
        print(f"Scholarship Eligible: {students[index]['name']}")  
    index += 1
```

The screenshot shows the OnlineGDB interface. The left sidebar includes a user profile (2303A51563), navigation links (Create New Project, My Projects, Classroom, Learn Programming, Programming Questions, Upgrade, Logout), and footer links (About, FAQ, Blog, Terms of Use, Contact Us, GDB Tutorial, Credits, Privacy, Copyright). The main area displays Python code for a `BankAccount` class:

```
1 # Banking Transaction Simulation
2
3 class BankAccount:
4     def __init__(self, account_holder, balance):
5         # Initialize account holder name and starting balance
6         self.account_holder = account_holder
7         self.balance = balance
8
9     # Copilot Prompt: Complete method to deposit money
10    def deposit(self, amount):
11        # Add the deposit amount to balance
12        # Print a confirmation message with updated balance
13        pass
14
15    # Copilot Prompt: Complete method to withdraw money
16    def withdraw(self, amount):
17        # Check if sufficient balance is available
18        # If yes, subtract amount and print success message
19        # Otherwise, print insufficient balance message
20        pass
21
22
23 # Example Usage
24 account1 = BankAccount("Akshaya", 5000)
```

The code is run and the output shows:

```
...Program finished with exit code 0
Press ENTER to exit console.
```

The screenshot shows the OnlineGDB interface. The left sidebar includes a user profile (2303A51563), navigation links (Create New Project, My Projects, Classroom, Learn Programming, Programming Questions, Upgrade, Logout), and footer links (About, FAQ, Blog, Terms of Use, Contact Us, GDB Tutorial, Credits, Privacy, Copyright). The main area displays Python code for a `BankAccount` class:

```
1     self.account_holder = account_holder
2     self.balance = balance
3
4     # Copilot Prompt: Complete method to deposit money
5     def deposit(self, amount):
6         # Add the deposit amount to balance
7         # Print a confirmation message with updated balance
8         pass
9
10    # Copilot Prompt: Complete method to withdraw money
11    def withdraw(self, amount):
12        # Check if sufficient balance is available
13        # If yes, subtract amount and print success message
14        # Otherwise, print insufficient balance message
15        pass
16
17
18 # Example Usage
19 account1 = BankAccount("Akshaya", 5000)
20
21 account1.deposit(2000)
22 account1.withdraw(3000)
23 account1.withdraw(6000)
24
```

The code is run and the output shows:

```
...Program finished with exit code 0
Press ENTER to exit console.
```

ask 5: Online Shopping Cart Module

Scenario

You are designing a simplified shopping cart system for an e-commerce website that supports item management and discount calculation.

Task Description

Begin writing a Python class named ShoppingCart with:

- An empty list to store items (each item may include name, price, quantity)

Use GitHub Copilot to 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)

Use meaningful comments and method names to guide Copilot.

Expected Outcome

- A fully implemented ShoppingCart class
- Copilot-generated loops and conditional logic
- Correct handling of item addition, removal, and discount calculation
- Sample input/output demonstrating cart functionality

Code

```
# Online Shopping Cart Module
```

```
class ShoppingCart:  
    def __init__(self):  
        # Initialize an empty list to store cart items  
        # Each item will have: name, price, quantity  
        self.items = []  
  
        # Copilot Prompt: Complete method to add an item to the cart  
    def add_item(self, name, price, quantity):  
        # Add a dictionary item with name, price, quantity into self.items  
        # Print confirmation message  
        pass  
  
        # Copilot Prompt: Complete method to remove an item from the cart  
    def remove_item(self, name):  
        # Search item by name and remove it from self.items  
        # Print message if item removed or not found  
        pass  
  
        # Copilot Prompt: Complete method to calculate total bill  
    def calculate_total(self):  
        # Use a loop to calculate total = sum(price * quantity)  
        # Apply discount condition:  
        # If total exceeds 5000, give 10% discount  
        # Return final amount
```

```
pass

# -----
# Example Usage
# -----


cart = ShoppingCart()

cart.add_item("Laptop Bag", 1500, 2)
cart.add_item("Wireless Mouse", 800, 1)
cart.add_item("Keyboard", 1200, 1)

cart.remove_item("Wireless Mouse")

final_bill = cart.calculate_total()
print("Final Bill Amount: ₹", final_bill)
```

OnlineGDB
online compiler and debugger for c/c++

Welcome, 2303A51563 ▲

Create New Project
My Projects
Classroom new
Learn Programming
Programming Questions
Upgrade
Logout ▾

```
main.py
1 # Online Shopping Cart Module
2
3 class ShoppingCart:
4     def __init__(self):
5         # Initialize an empty list to store cart items
6         # Each item will have: name, price, quantity
7         self.items = []
8
9     # Copilot Prompt: Complete method to add an item to the cart
10    def add_item(self, name, price, quantity):
11        # Add a dictionary item with name, price, quantity into self.items
12        # Print confirmation message
13        pass
14
15    # Copilot Prompt: Complete method to remove an item from the cart
16    def remove_item(self, name):
17        # Search item by name and remove it from self.items
18        # Print message if item removed or not found
19        pass
20
21    # Copilot Prompt: Complete method to calculate total bill
22    def calculate_total(self):
23        # Use a loop to calculate total = sum(price * quantity)
24        # Apply discount condition
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
```

Final Bill Amount: ₹ None

...Program finished with exit code 0
Press ENTER to exit console.■

OnlineGDB
online compiler and debugger for c/c++

Welcome, 2303A51563 ▲

Create New Project
My Projects
Classroom new
Learn Programming
Programming Questions
Upgrade
Logout ▾

```
main.py
1 # COPilot Prompt: Complete method to calculate total bill
2 def calculate_total(self):
3     # Use a loop to calculate total = sum(price * quantity)
4     # Apply discount condition:
5     # If total exceeds 5000, give 10% discount
6     # Return final amount
7     pass
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
```

Final Bill Amount: ₹ None

...Program finished with exit code 0
Press ENTER to exit console.■