TASK-1:

• 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

```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):
        """Displays the student's details."""
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")

    def is_above_average(self, average_marks):
        """Checks if the student's marks are above the given average."""
        return self.marks > average_marks

# Example usage (will be added in a later step)
student1 = Student("Alice", "A101", 95)
student1.display_details()
class_average = 80  # Example average
if student1.is_above_average(class_average):
    print(f"{student1.name} scored above the class average.")
else:
    print(f"{student1.name} did not score above the class average.")
```

```
Name: Alice
Roll Number: A101
Marks: 95
Alice scored above the class average.
```

EXPLANATION:

1. `class Student:` : This line defines a new class named `Student`.
2. `def __init__(self, name, roll_number, marks):` : This is the constructor method. It's called when you create a new `Student` object.
    - `self` : Refers to the instance of the class being created.
    - `name` : An attribute to store the student's name.
    - `roll_number` : An attribute to store the student's roll number.
    - `marks` : An attribute to store the student's marks.
3. `def display_details(self):` : This method prints the student's details (name, roll number, and marks) in a formatted way.
4. `def is_above_average(self, average_marks):` : This method checks if the student's marks are above a given average.
    - It takes `average_marks` as input.
    - It compares `self.marks` (the student's marks) with `average_marks`.
    - It returns `True` if the student's marks are greater than the average, and `False` otherwise.
5. `# Example usage` : The commented-out lines at the end show how to create a `Student` object, call the `display_details` method, set an example `class_average`, and use the `is_above_average` method within an `if` statement to print whether the student scored above the class average.

TASK-2:

• 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.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for number in numbers:
    if number % 2 == 0:
        square = number ** 2
        print(f"The square of {number} is {square}")
```

```
The square of 2 is 4
The square of 4 is 16
The square of 6 is 36
The square of 8 is 64
The square of 10 is 100
```

Explanation:

Certainly! The selected code iterates through a list of numbers and prints the square of only the even numbers.

Here's a breakdown:

1. `numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`: This line initializes a list named `numbers` containing integers from 1 to 10.
2. `for number in numbers:`: This is a `for` loop that iterates through each element in the `numbers` list. In each iteration, the current element is assigned to the variable `number`.
3. `if number % 2 == 0:`: This is an `if` statement that checks if the current `number` is even. The modulo operator (`%`) returns the remainder of a division. If a number divided by 2 has a remainder of 0, it's an even number.
4. `square = number ** 2`: If the `if` condition is true (the number is even), this line calculates the square of the `number` using the exponentiation operator (`** 2`) and assigns the result to the variable `square`.
5. `print(f"The square of {number} is {square}")`: This line prints the output. The `f""` denotes an f-string, which allows you to embed the values of variables directly within the string. It will print a message showing the original even number and its calculated square.

In essence, the code goes through each number, picks out the even ones, and then tells you what their squares are.

TASK-3:

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

balance.

```python
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"Deposit of ${amount} successful. New balance: ${self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if self.balance >= amount:
                self.balance -= amount
                print(f"Withdrawal of ${amount} successful. New balance: ${self.balance}")
            else:
                print("Insufficient balance.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        print(f"Account balance for {self.account_holder}: ${self.balance}")

#Example usage:
account1 = BankAccount("Alice Smith", 1000)
account1.check_balance()
account1.deposit(500)
account1.withdraw(200)
```

```
account1.check_balance()

Account balance for Alice Smith: $1000
Deposit of $500 successful. New balance: $1500
Withdrawal of $200 successful. New balance: $1300
Insufficient balance.
Account balance for Alice Smith: $1300
```

EXPLANATION:

1. `class BankAccount:` : This line defines a new class named `BankAccount`.
2. `def __init__(self, account_holder, balance=0):` : This is the constructor method. It's called when you create a new `BankAccount` object.
    - `self` : Refers to the instance of the class being created.
    - `account_holder` : An attribute to store the name of the account holder.
    - `balance=0` : An attribute to store the account balance, with a default value of 0 if not provided.
3. `def deposit(self, amount):` : This method handles depositing money into the account.
    - It checks if the `amount` is positive.
    - If positive, it adds the `amount` to the `balance` and prints a success message.
    - If not positive, it prints an error message.
4. `def withdraw(self, amount):` : This method handles withdrawing money from the account.
    - It checks if the `amount` is positive.
    - If positive, it checks if the `balance` is sufficient (`self.balance >= amount`).
    - If sufficient, it subtracts the `amount` from the `balance` and prints a success message.
    - If insufficient, it prints an "Insufficient balance" message.
    - If the `amount` is not positive, it prints an error message.
5. `def check_balance(self):` : This method simply prints the current account balance for the `account_holder`.
6. `# Example usage:` : The lines that follow are commented out examples of how to create a `BankAccount` object and use its methods.

TASK-4:

• 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.

```
[8]  students = [
        {"name": "Alice", "score": 85},
        {"name": "Bob", "score": 70},
        {"name": "Charlie", "score": 92},
        {"name": "David", "score": 78},
        {"name": "Eve", "score": 65}
    ]

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

```
Alice
Charlie
David
```

EXPLANATION:

Certainly! The selected code iterates through a list of student dictionaries and prints the names of students whose score is greater than 75 using a `while` loop.

Here's a breakdown:

1. `students = [...]`: This line initializes a list named `students`. Each element in the list is a dictionary, with keys `"name"` and `"score"` representing a student's name and their score.
2. `i = 0`: This line initializes a counter variable `i` to 0. This variable will be used to keep track of the current index while iterating through the list.
3. `while i < len(students):`: This is a `while` loop that continues as long as the value of `i` is less than the total number of elements in the `students` list (`len(students)`).
4. `if students[i]["score"] > 75:`: Inside the loop, this `if` statement checks if the `"score"` value of the dictionary at the current index `i` in the `students` list is greater than 75.
5. `print(students[i]["name"])`: If the `if` condition is true (the student's score is greater than 75), this line prints the `"name"` value of the dictionary at the current index `i`.
6. `i += 1`: This line increments the value of `i` by 1 after each iteration. This is crucial for moving to the next element in the list and eventually terminating the `while` loop.

In essence, the code goes through each student one by one using the index `i`, checks their score, and if the score is above 75, it prints their name.

TASK-5:

• 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.

```python
class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, item):
        """Adds an item to the shopping cart."""
        self.items.append(item)
        print(f"{item['name']} added to the cart.")

    def remove_item(self, item_name):
        """Removes an item from the shopping cart by name."""
        for item in self.items:
            if item['name'] == item_name:
                self.items.remove(item)
                print(f"{item_name} removed from the cart.")
                return
        print(f"{item_name} not found in the cart.")

    def calculate_total(self):
        """Calculates the total bill with conditional discounts."""
        total = 0
        for item in self.items:
            total += item['price']

        # Apply a discount if the total is over a certain amount
        if total > 100:
            discount = total * 0.10  # 10% discount
            total -= discount
            print(f"Applied a 10% discount. Discount amount: ${discount:.2f}")
```

```
        print(f"Total bill: ${total:.2f}")
        return total

# Example Usage (will be added in a later step)
cart = ShoppingCart()
cart.add_item({"name": "Laptop", "price": 1200})
cart.add_item({"name": "Mouse", "price": 25})
cart.add_item({"name": "Keyboard", "price": 75})
cart.calculate_total()
cart.remove_item("Mouse")
cart.calculate_total()
```

```
Laptop added to the cart.
Mouse added to the cart.
Keyboard added to the cart.
Applied a 10% discount. Discount amount: $130.00
Total bill: $1170.00
Mouse removed from the cart.
Applied a 10% discount. Discount amount: $127.50
Total bill: $1147.50
1147.5
```

EXPLANATION:

1. `class ShoppingCart:` : This line defines a new class named `ShoppingCart`.
2. `def __init__(self):` : This is the constructor method. It's called when you create a new `ShoppingCart` object. It initializes an empty list called `self.items` to store the items added to the cart.
3. `def add_item(self, item):` : This method adds an item to the shopping cart.
   - It takes an `item` (expected to be a dictionary with at least a 'name' and 'price' key) as input.
   - It appends the `item` to the `self.items` list.
   - It prints a confirmation message that the item has been added.
4. `def remove_item(self, item_name):` : This method removes an item from the shopping cart by its name.
   - It takes the `item_name` (a string) as input.
   - It iterates through the `self.items` list.
   - If an item's 'name' matches the `item_name`, it removes that item from the list using `self.items.remove(item)`.
   - It prints a confirmation message that the item has been removed and then `return`s to exit the method after removing the first match.
   - If the loop finishes without finding the item, it prints a message indicating that the item was not found.
5. `def calculate_total(self):` : This method calculates the total bill with a conditional discount.
   - It initializes a `total` variable to 0.
   - It iterates through the `self.items` list and adds the 'price' of each item to the `total`.
   - It then checks if the `total` is greater than 100.
   - If the `total` is greater than 100, it calculates a 10% `discount` and subtracts it from the `total`. It also prints the discount amount.
   - Finally, it prints the `Total bill` formatted to two decimal places and returns the calculated `total`.
6. `# Example Usage` : The commented-out lines at the end show how to create a `ShoppingCart` object and use its methods to add items, calculate the total, remove an item, and calculate the total again.