

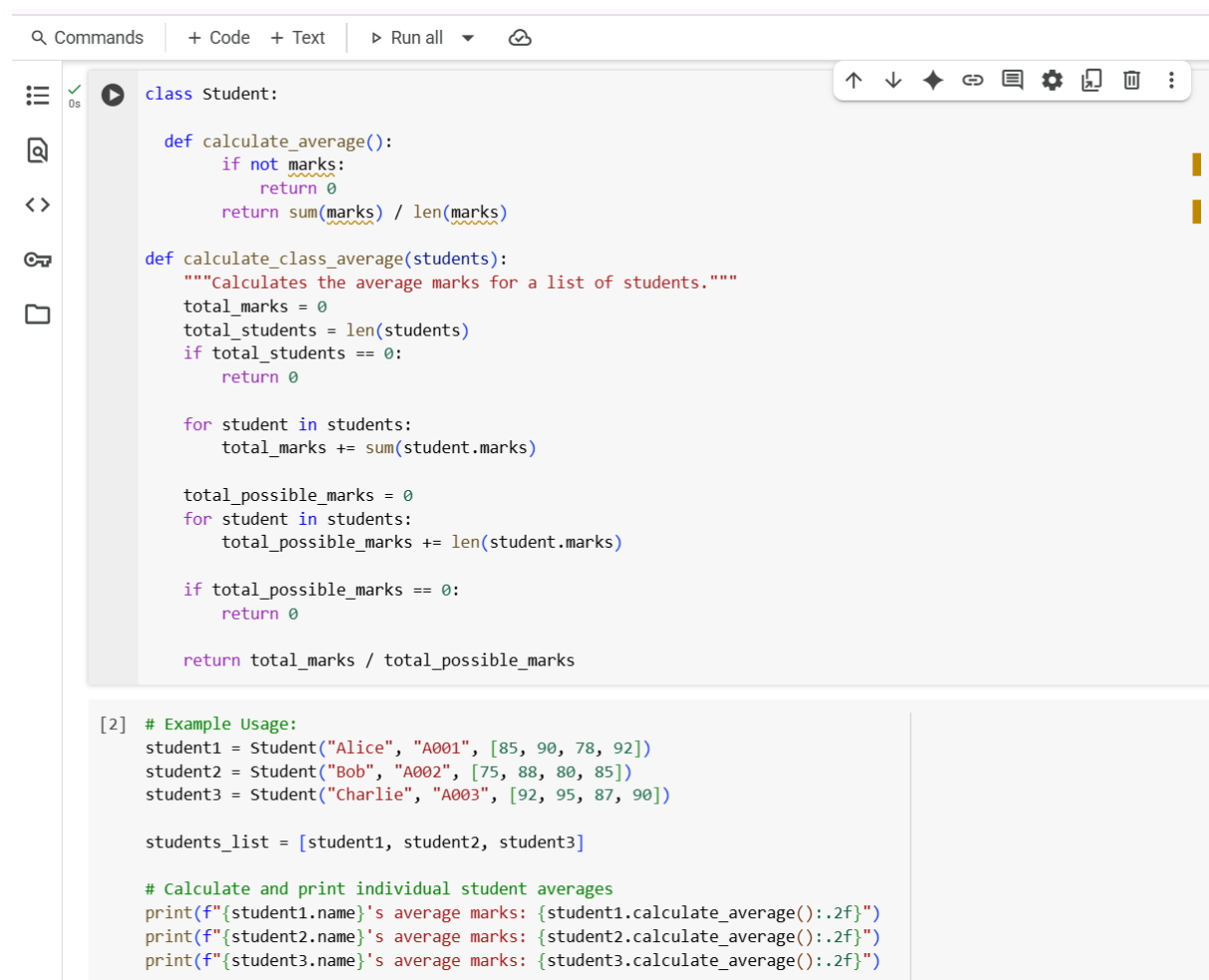
# Ai Assignment: 6.4

## Task Description #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.

## Expected Outcome #1:

- Completed class with Copilot-generated methods like display\_details() and is\_passed(), demonstrating use of if-else conditions.



```
class Student:

    def calculate_average():
        if not marks:
            return 0
        return sum(marks) / len(marks)

    def calculate_class_average(students):
        """Calculates the average marks for a list of students."""
        total_marks = 0
        total_students = len(students)
        if total_students == 0:
            return 0

        for student in students:
            total_marks += sum(student.marks)

        total_possible_marks = 0
        for student in students:
            total_possible_marks += len(student.marks)

        if total_possible_marks == 0:
            return 0

        return total_marks / total_possible_marks

[2] # Example Usage:
student1 = Student("Alice", "A001", [85, 90, 78, 92])
student2 = Student("Bob", "A002", [75, 88, 80, 85])
student3 = Student("Charlie", "A003", [92, 95, 87, 90])

students_list = [student1, student2, student3]

# Calculate and print individual student averages
print(f"{student1.name}'s average marks: {student1.calculate_average():.2f}")
print(f"{student2.name}'s average marks: {student2.calculate_average():.2f}")
print(f"{student3.name}'s average marks: {student3.calculate_average():.2f}")
```

⇒ Alice's average marks: 86.25  
Bob's average marks: 82.00  
Charlie's average marks: 91.00

Class average marks: 86.42

M Please explain this code:

```
<> class Student:
```

◆ This code defines a `Student` class and a function to calculate the class average.

Here's a breakdown:

- `class Student`: This line defines a blueprint for creating student objects.
- `def calculate_average()`: This function within the `Student` class is intended to calculate the average marks for a single student. However, it has an error as it tries to access `marks` directly, which should be `self.marks` within the class method.
- `def calculate_class_average(students)`: This function takes a list of `students` as input and calculates the average marks across all students in the list. It iterates through the students, sums their marks, and then divides by the total number of possible marks (sum of the lengths of each student's marks list).

👍 🗨

## Task Description #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.

## Expected Outcome #2:

- A complete loop generated by Copilot with conditional logic (if `number % 2 == 0`) and appropriate output.

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

for number in numbers:
    if number % 2 == 0:
        # Calculate and print the square of the even number
        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

M Please explain this code:

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

✦ This code iterates through a list of numbers and prints the square of only the even numbers.

Here's a breakdown:

- `numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`: This line creates a list named `numbers` containing integers from 1 to 10.
- `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`.
- `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.
- `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 (`**`) and assigns the result to the variable `square`.
- `print(f"The square of {number} is {square}")`: This line prints the output to the console. It uses an f-string to embed the values of `number` and `square` directly into the output string.



### Task Description #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.

### Expected Outcome #3:

- Functional class with complete method definitions using `if` conditions and `self` attributes. Code should prevent overdrawing.

```
Commands | + Code | + Text | ▶ Run all | 
```

```
class BankAccount:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        """Deposits the specified amount into the account."""
        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):
        """Withdraws the specified amount from the account."""
        if amount > 0:
            if self.balance >= amount:
                self.balance -= amount
                print(f"Withdrew {amount}. New balance: {self.balance}")
            else:
                print("Insufficient balance.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        """Prints the current account balance."""
        print(f"Current balance for {self.account_holder}: {self.balance}")

[10] # Example Usage:
account1 = BankAccount("Ishwarya", 100000)

account1.check_balance()
account1.deposit(50000)
account1.withdraw(20000)
account1.withdraw(15000) # Attempt to withdraw more than balance
account1.check_balance()
```

Current balance for Ishwarya: 100000  
Deposited 50000. New balance: 150000  
Withdrew 20000. New balance: 130000  
Withdrew 15000. New balance: 115000  
Current balance for Ishwarya: 115000

**M** Please explain this code:

```
<> class BankAccount:
```

✦ This code defines a `BankAccount` class, which is a blueprint for creating bank account objects.

Here's a breakdown:

- `class BankAccount:` : This line defines the `BankAccount` class.
- `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.
- `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 `self.balance`.
  - It then prints a confirmation message with the new balance.
  - If the amount is not positive, it prints an error message.
- `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 `self.balance` is sufficient (`>= amount`).
  - If sufficient, it subtracts the `amount` from `self.balance` and prints a confirmation message.
  - If the balance is insufficient, it prints an "Insufficient balance" message.
  - If the withdrawal amount is not positive, it prints an error message.
- `def check_balance(self):` : This method simply prints the current balance for the `account_holder`.

### Task Description #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.

### Expected Outcome #4:

• A complete while loop generated by Copilot with proper condition checks and formatted output.

0s

students = [  
    {"name": "Lucky", "score": 85},  
    {"name": "Gandhi", "score": 70},  
    {"name": "Harsha", "score": 92},  
    {"name": "Ramya", "score": 78},  
    {"name": "ishwarya", "score": 65},  
]

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

Lucky  
Harsha  
Ramya

Gemini X

**M** Please explain this code:

```
<> students = [
```

✦ This code iterates through a list of dictionaries, where each dictionary represents a student with a `name` and a `score`.

Here's a breakdown:

1. **students list**: This list contains dictionaries, with each dictionary representing a student and their score.
2. **i = 0**: Initializes a counter variable `i` to 0. This variable will be used to access elements in the `students` list.
3. **while i < len(students):**: This starts a `while` loop that continues as long as the value of `i` is less than the number of students in the list.
4. **student = students[i]**: In each iteration, this line accesses the dictionary at the index `i` in the `students` list and assigns it to the `student` variable.
5. **if student["score"] > 75:**: This is a conditional statement that checks if the value associated with the key `"score"` in the current `student` dictionary is greater than 75.
6. **print(student["name"])**: If the condition in the `if` statement is true (the student's score is greater than 75), this line prints the value associated with the key `"name"` from the current `student` dictionary.
7. **i += 1**: This line increments the counter `i` by 1 after each iteration. This is crucial for the loop to eventually terminate and move to the next student in the list.

In summary, the code iterates through the list of students and prints the name of any student who has a score greater than 75.



### Task Description #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.

### Expected Outcome #5:

- A fully implemented `ShoppingCart` class with Copilot-generated loops and if-else statements handling item management and discount logic.

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

    def add_item(self, item, price, quantity=1):
        """Adds an item to the shopping cart."""
        self.items.append({"item": item, "price": price, "quantity": quantity})
        print(f"Added {quantity} x {item} to the cart.")

    def remove_item(self, item):
        """Removes an item from the shopping cart."""
        initial_len = len(self.items)
        self.items = [i for i in self.items if i["item"] != item]
        if len(self.items) < initial_len:
            print(f"Removed {item} from the cart.")
        else:
            print(f"{item} not found in the cart.")

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

        # Apply conditional discounts (example)
        if total_bill > 100:
            total_bill *= 0.9 # 10% discount for orders over $100
            print("Applied 10% discount.")
        elif total_bill > 50:
            total_bill *= 0.95 # 5% discount for orders over $50
            print("Applied 5% discount.")

        return total_bill
```

```
# Example Usage (Clothes):
my_cart_clothes = ShoppingCart()

my_cart_clothes.add_item("T-shirt", 20, 3)
my_cart_clothes.add_item("Jeans", 50, 1)
my_cart_clothes.add_item("Dress", 75, 2)
my_cart_clothes.add_item("Socks", 5, 6)

print("\nCurrent items in clothes cart:")
for item in my_cart_clothes.items:
    print(f"- {item['quantity']} x {item['item']} at ${item['price']}")

total_clothes = my_cart_clothes.calculate_total()
print(f"\nTotal bill for clothes: ${total_clothes:.2f}")
```

➡ Added 3 x T-shirt to the cart.  
 Added 1 x Jeans to the cart.  
 Added 2 x Dress to the cart.  
 Added 6 x Socks to the cart.

Current items in clothes cart:  
 - 3 x T-shirt at \$20  
 - 1 x Jeans at \$50  
 - 2 x Dress at \$75  
 - 6 x Socks at \$5  
 Applied 10% discount.

Total bill for clothes: \$261.00

✦ This code defines a Python class called `ShoppingCart` and provides two example usages of it.

Here's a breakdown of the class and its methods:

- **class ShoppingCart:** : This line declares the start of the `ShoppingCart` class definition.
- **\_\_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` which will store the items in the cart.
- **add\_item(self, item, price, quantity=1)** : This method adds an item to the `self.items` list. It takes the `item` name, its `price`, and an optional `quantity` (defaulting to 1) as arguments. It appends a dictionary containing the item details to the `self.items` list and prints a confirmation message.
- **remove\_item(self, item)** : This method removes an item from the `self.items` list based on the item name. It creates a new list containing only the items that do not match the specified item name. It then updates `self.items` with this new list and prints a confirmation or a "not found" message.
- **calculate\_total(self)** : This method calculates the total bill for the items in the cart. It iterates through the `self.items` list, multiplies the price by the quantity for each item, and sums them up. It then applies conditional discounts: a 10% discount if the total is over \$100, and a 5% discount if the total is over \$50. Finally, it returns the calculated total.

The code also includes two example usages:

1. **Electronics Example:** This section creates a `ShoppingCart` instance called `my_cart_electronics`, adds some electronic items (Laptop, Mouse, Keyboard), prints the items, calculates and prints the total, removes the "Mouse", and then recalculates and prints the total after the removal.
2. **Clothes Example:** This section creates another `ShoppingCart` instance called `my_cart_clothes`, adds some clothing items (T-shirt, Jeans, Dress, Socks), prints the items, and calculates and prints the total.

Essentially, this code provides a reusable structure for managing a shopping cart, adding/removing items, and calculating the total with discounts.





