

NAME: KANDIBANDA SRI SAI HARSHA KISHORE

ROLL.NO: 2403A52133.

BATCH:6.

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.

```

roll_number = input("Enter roll number: ")
marks = int(input("Enter marks: ")) # Convert marks to integer

# Create a Student object
student1 = Student(name, roll_number, marks)

# Display student details
student1.display_details()

# Check if the student passed
if student1.is_passed():
    print(f"{student1.name} has passed.")
else:
    print(f"{student1.name} has failed.")

# You can add more students and calculate average marks if needed
# For example, to calculate average of multiple students, you would store them in a list
# and iterate through the list to sum up the marks and divide by the number of students.

```

```

Enter student name: manideep
Enter roll number: 2146
Enter marks: 99
Name: manideep
Roll Number: 2146
Marks: 99
manideep has passed.

```

[] Start coding or [generate](#) with AI.

✓
16s



```

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_passed(self):
        # Assuming passing marks is 40
        return self.marks >= 40

# Get user input for student details
name = input("Enter student name: ")
roll_number = input("Enter roll number: ")
marks = int(input("Enter marks: ")) # Convert marks to integer

# Create a Student object
student1 = Student(name, roll_number, marks)

# Display student details
student1.display_details()

```

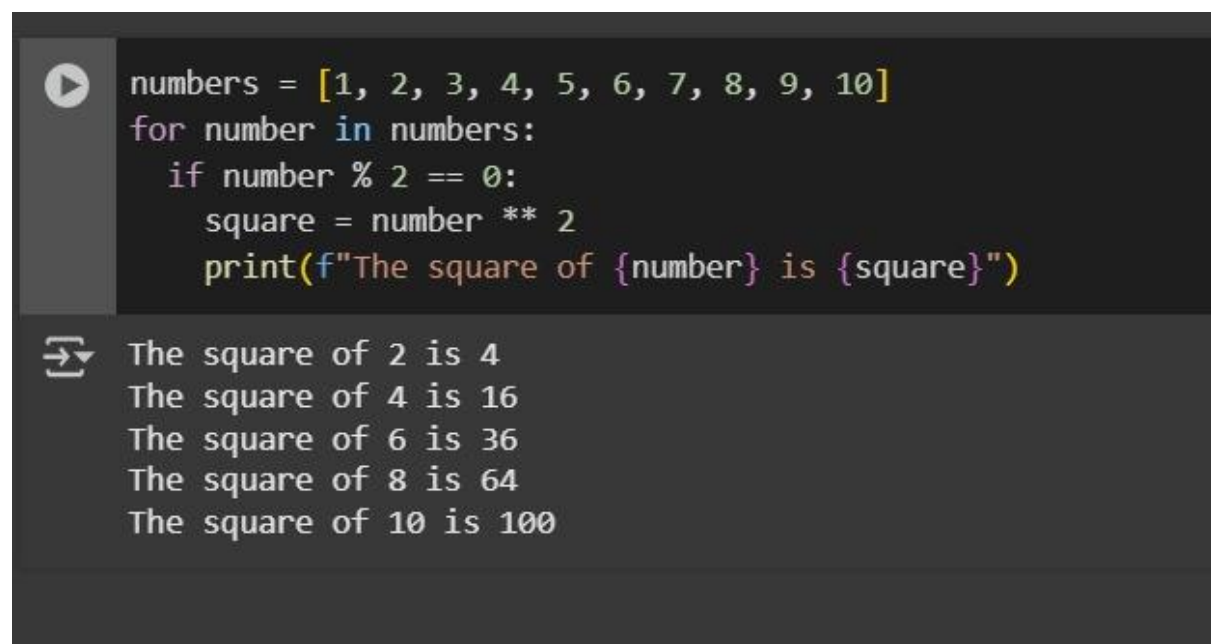
- ❑ **A blueprint is created for students** using a class called "Student". This blueprint allows storing a student's name, roll number, and marks.
- ❑ When a student is created, their name, roll number, and marks are saved using this blueprint.
- ❑ There is a function inside the blueprint that shows the details of the student, like printing their name, roll number, and marks.
- ❑ Another function checks whether the student has passed or failed. It does this by comparing their marks with the passing score. The passing score is set to forty. If the marks are forty or more, the student has passed. Otherwise, the student has failed.
- ❑ The program asks the user to enter the student's name, roll number, and marks through the keyboard.
- ❑ Once the details are entered, a new student is created using the entered data.
- ❑ The program then shows the entered student's details on the screen.
- ❑ After that, it checks if the student passed or failed and displays the result.
- ❑ At the end, there's a suggestion written in a comment explaining that more students can be added. If that's done, you can store them in a list and then calculate the average marks by adding all the marks and dividing by the number of students.

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.



```
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

- The code starts with a list of numbers from 1 to 10.

- It then goes through each number in this list one by one.
- For every number, it checks if it's an even number (meaning it can be divided by 2 with no remainder).
- If the number is even, it calculates the result of multiplying that number by itself (this is called squaring the number).
- Finally, for each even number it found, it tells you the original number and the result of squaring it.

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.

[] Start coding or [generate](#) with AI.

```
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 > 0:
            if self.check_insufficient_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_insufficient_balance(self, amount):
        return self.balance >= amount
```

```
def withdraw(self, amount):
    if amount > 0:
        if self.check_insufficient_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_insufficient_balance(self, amount):
    return self.balance >= amount
```

You can create an instance of the `BankAccount` class and use its methods like this:

```
[ ] # Example usage:
account1 = BankAccount("Alice")
account1.deposit(100)
account1.withdraw(50)
account1.withdraw(60) # This will result in an insufficient balance message
```

- BankAccount is like a blueprint for creating these virtual accounts.

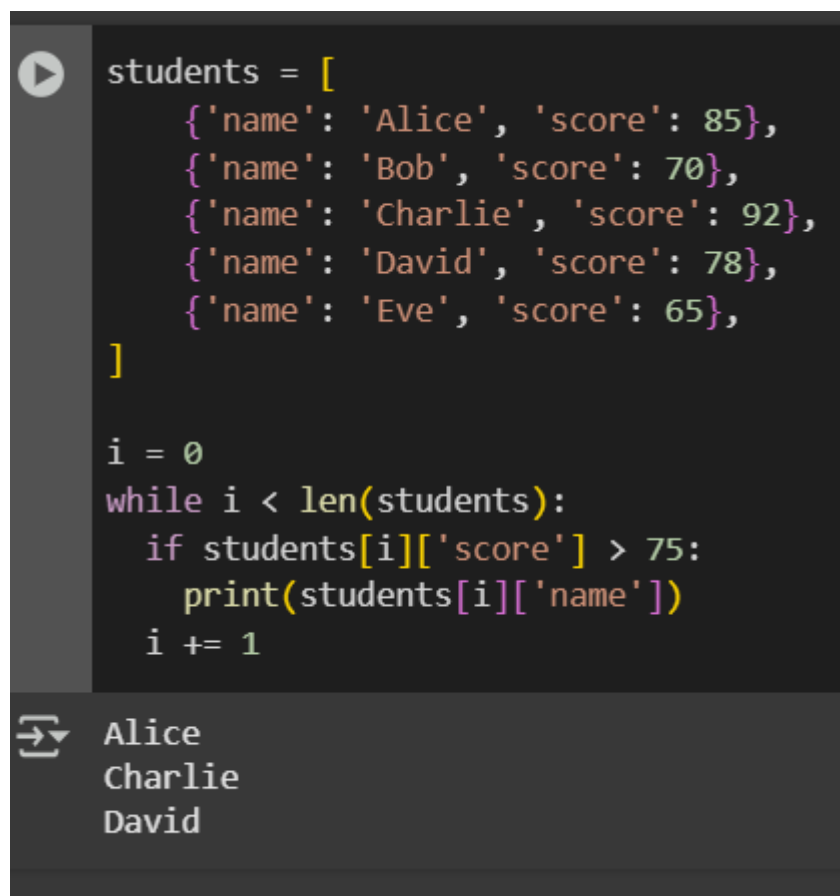
- When you create an account, you give it an `account_holder` name (like your name) and it starts with a balance (usually zero at first).
- The deposit part is like putting money *into* your account. You tell it how much amount to add, and if it's a valid amount (more than zero), it increases your balance.
- The withdraw part is like taking money *out* of your account. You tell it how much amount to take. It first checks if you have enough money (`check_insufficient_balance`). If you do, it reduces your balance. If not, it tells you there's not enough money.
- `check_insufficient_balance` is just a little helper that looks at your balance and the amount you want to take out to see if you have enough.

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.



```

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

Imagine you have a list of students, and for each student, you know their name and their score on a test.

This code goes through each student one by one. It checks if a student's score is higher than 75. If it is, the code says the student's name out loud. It keeps doing this until it has checked every student on the list.

So, in short, the code is just finding and telling you the names of the students who did really well on the test (scored more 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):
        self.items.append({'item': item, 'price': price, 'quantity': quantity})

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

    def calculate_total(self):
        total = 0
        for item in self.items:
            item_total = item['price'] * item['quantity']
            # Apply conditional discounts (example)
            if item['quantity'] > 5:
                item_total *= 0.9 # 10% discount for quantity > 5
            total += item_total
        return total

# Example usage:
cart = ShoppingCart()
cart.add_item("Apple", 0.5, 10)
cart.add_item("Banana", 0.3, 3)
cart.add_item("Orange", 0.75, 6)

print("Items in cart:", cart.items)
print("Total bill:", cart.calculate_total())

```

```

cart = ShoppingCart()
cart.add_item("Apple", 0.5, 10)
cart.add_item("Banana", 0.3, 3)
cart.add_item("Orange", 0.75, 6)

print("Items in cart:", cart.items)
print("Total bill:", cart.calculate_total())

cart.remove_item("Banana")
print("\nItems in cart after removing Banana:", cart.items)
print("Total bill after removing Banana:", cart.calculate_total())

```

Items in cart: [{'item': 'Apple', 'price': 0.5, 'quantity': 10}, {'item': 'Banana', 'price': 0.3, 'quantity': 3}, {'item': 'Orange', 'price': 0.75, 'quantity': 6}]
Total bill: 9.45

Items in cart after removing Banana: [{'item': 'Apple', 'price': 0.5, 'quantity': 10}, {'item': 'Orange', 'price': 0.75, 'quantity': 6}]
Total bill after removing Banana: 8.55

- **ShoppingCart:** This is like a digital basket for your shopping.
- **__init__:** When you get a new basket, it starts empty. This part sets up the empty basket.
- **add_item:** This is how you put things into your basket. You tell it the name of the item, how much it costs, and how many you want.
- **remove_item:** This is how you take something out of your basket. You just tell it the name of the item you want to remove.
- **calculate_total:** This figures out how much everything in your basket costs.
 - It goes through each item.

- It multiplies the price by how many you have of that item to get the cost for that item.
- If you have more than 5 of an item, you get a 10% discount on just that item.
- It adds up the cost of all the items to get the final total.