**SR UNIVERSITY**

**AI ASSIST CODING**

**Lab-6.4**: *AI-Based Code Completion – Classes, Loops, and Conditionals*

**ROLL NO:**2503A51L23

**NAME**:MOHAMMAD NASRIN

**BATCH:19**

**Lab Objectives:**

* To explore AI-powered auto-completion features for core Python constructs.

* To analyze how AI suggests logic for class definitions, loops, and conditionals.

* To evaluate the completeness and correctness of code generated by AI assistants. **Lab Outcomes (LOs):**

After completing this lab, students will be able to:

* Use AI tools to generate and complete class definitions and methods.

* Understand and assess AI-suggested loops for iterative tasks.

* Generate conditional statements through prompt-driven suggestions.

* Critically evaluate AI-assisted code for correctness and clarity **TASK #1:**

**Prompt Used:**

* 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.

**Code:**

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}, Roll No: {self.roll\_number}, Marks: {self.marks}") def is\_above\_average(self, average=50): if self.marks > average:

print(f"{self.name} has marks above average.")

else:

print(f"{self.name} does not have marks above average.") if \_\_name\_\_ == "\_\_main\_\_":

name = input("Enter student name: ") roll\_number = int(input("Enter roll number: ")) marks = float(input("Enter marks: ")) student = Student(name, roll\_number, marks) print("\n--- Student Details ---") student.display\_details() avg = float(input("Enter average marks to compare: ")) student.is\_above\_average(avg) class Student: def \_\_init\_\_(self, name, roll\_number, marks):

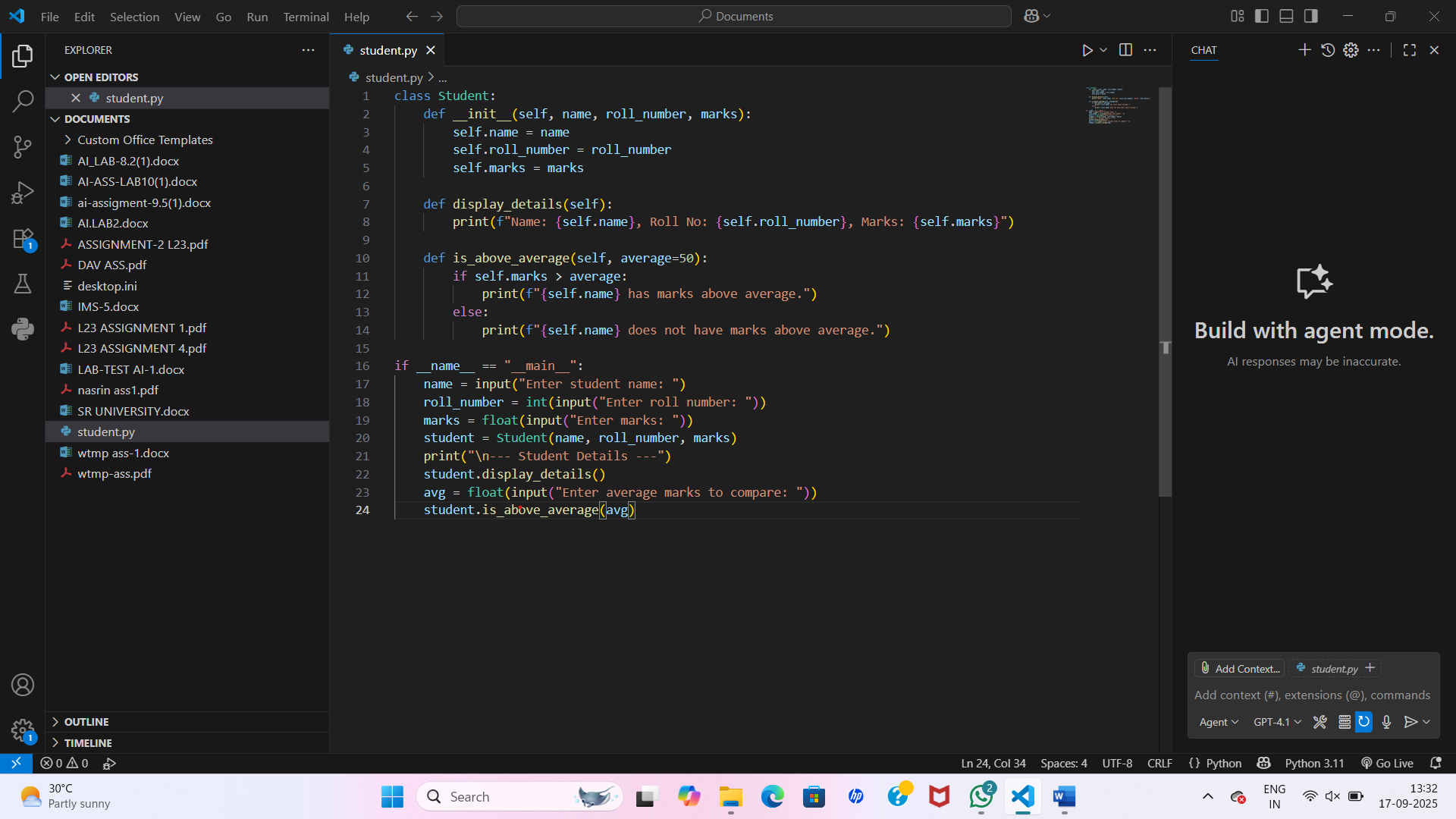
self.name = name self.roll\_number = roll\_number self.marks = marks def display\_details(self): print(f"Name: {self.name}, Roll No: {self.roll\_number}, Marks: {self.marks}")

def is\_above\_average(self, average=50): if self.marks > average: print(f"{self.name} has marks above average.")

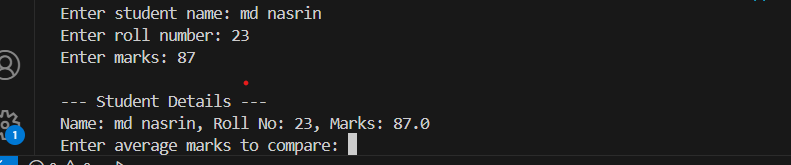
else:

print(f"{self.name} does not have marks above average.") if \_\_name\_\_ == "\_\_main\_\_": name = input("Enter student name: ") roll\_number = int(input("Enter roll number: ")) marks = float(input("Enter marks: ")) student = Student(name, roll\_number, marks) print("\n--- Student Details ---") student.display\_details() avg = float(input("Enter average marks to compare: ")) student.is\_above\_average(avg)

**code generated:**

****

**Output After executing Code:**



**Observations:**

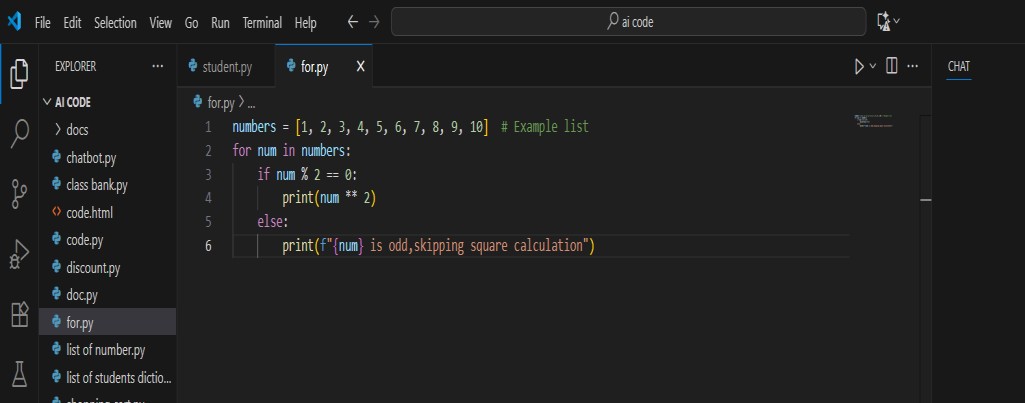
* A Student class is created with attributes **name**, **roll number**, and **marks**.
* The display\_details() method neatly prints the student’s details.
* The is\_above\_average() method compares the student’s marks with a given average and prints the result.
* User input is taken for **name, roll number, marks, and average** at runtime, making the program interactive.

**TASK #2:**

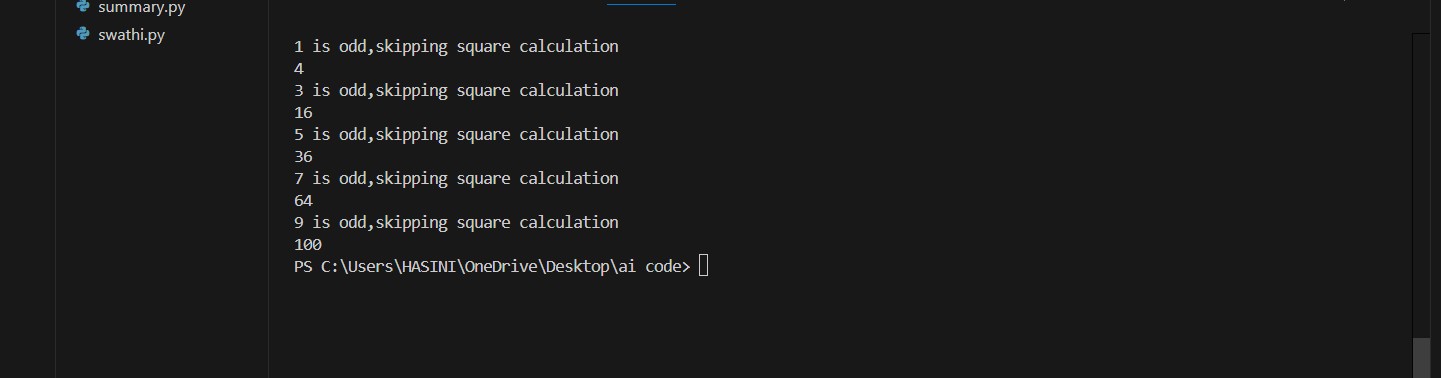
**Prompt Used:**

Write the first two lines of a for loop to iterate through a list o f numbers. Suggest how to calculate and print the square of even numbers only.

**Code Generated:**



**Output After executing Code:**



**Observations:**

The function Iterates through numbers.

We have to give the Condition if num % 2 == 0 checks even numbers.

It results in Prints their square using num \*\* 2.

**TASK#3:**

**Prompt Used:**

•Create a class called Bank Account with attributes accountholder and balance . Complete methods for deposit() ,withdraw() ,and check for insufficient balance.

**Code:**

class BankAccount: def \_\_init\_\_(self, account\_holder,

balance=0, overdraft\_limit=0):

self.account\_holder = account\_holder self.balance = balance self.overdraft\_limit = overdraft\_limit 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:

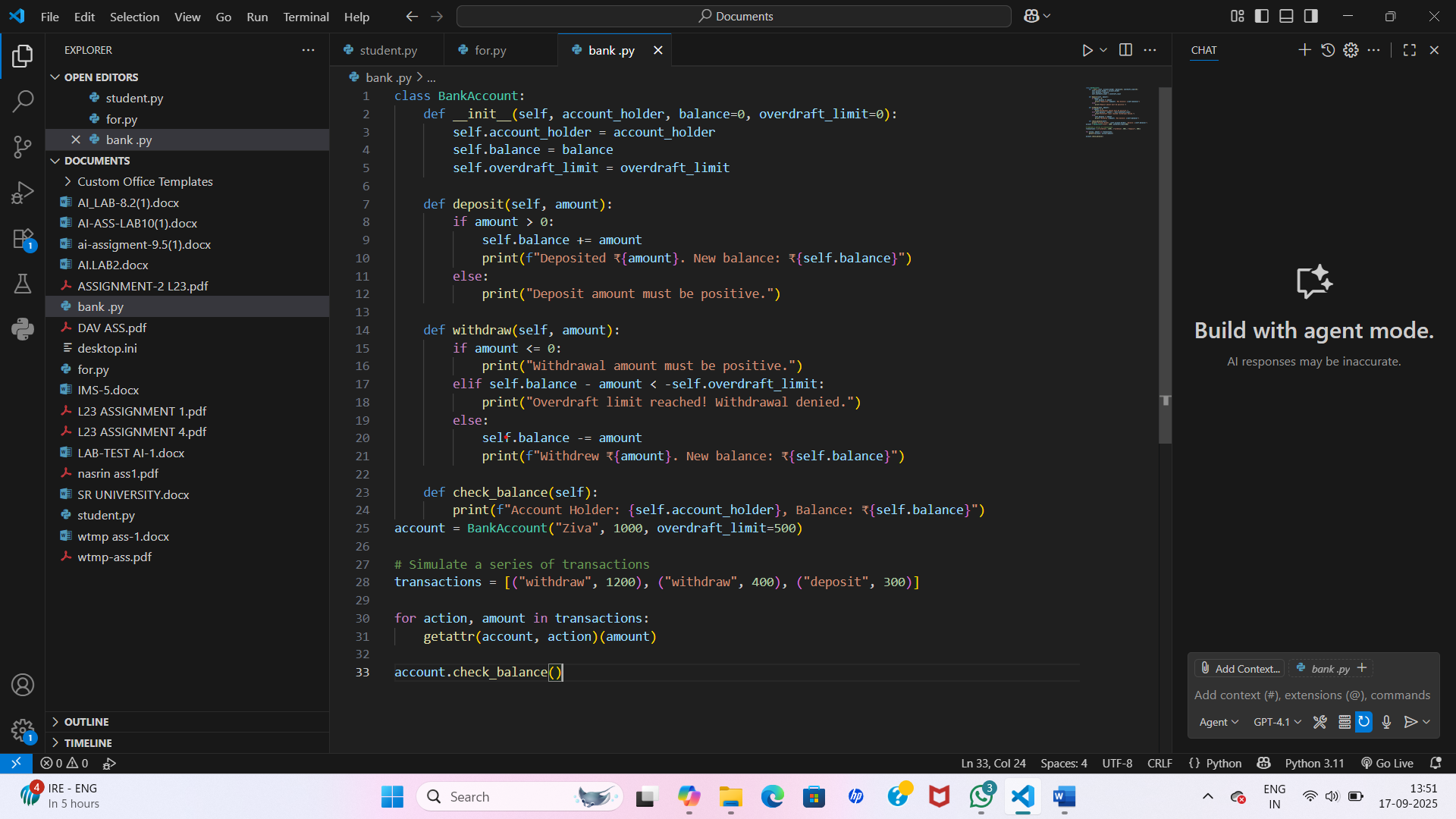
print("Withdrawal amount must be positive.") elif self.balance - amount < -self.overdraft\_limit:

print("Overdraft limit reached! Withdrawal denied.")

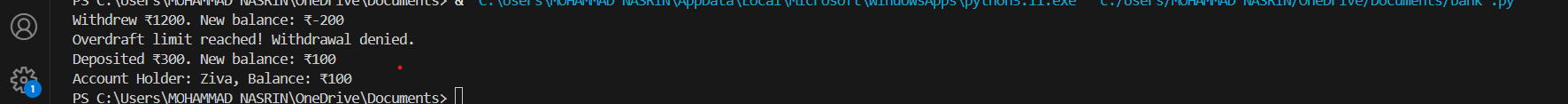
else:

self.balance -= amount print(f"Withdrew {amount}.

New balance: {self.balance}") def check\_balance(self): print(f"Account Holder: {self.account\_holder}, Balance: {self.balance}") account = BankAccount("ziva", 1000, overdraft\_limit=500) for action, amount in [("withdraw", 1200), ("withdraw", 400), ("deposit", 300)]: getattr(account, action)(amount) account.check\_balance()

**Code Generated:** 

**Output After executing Code:**



**Observation:**

We used function deposit(): increases balance. we can able to use the function withdraw(): prevents overdrawing using if conditions . its results in check\_balance(): shows current balance**.**

**TASK#4:**

**Prompt Used:**

Define a list of student dictionaries with keys name and score. Write a while loop to print the names of students who scored more than 75.

**Code:**

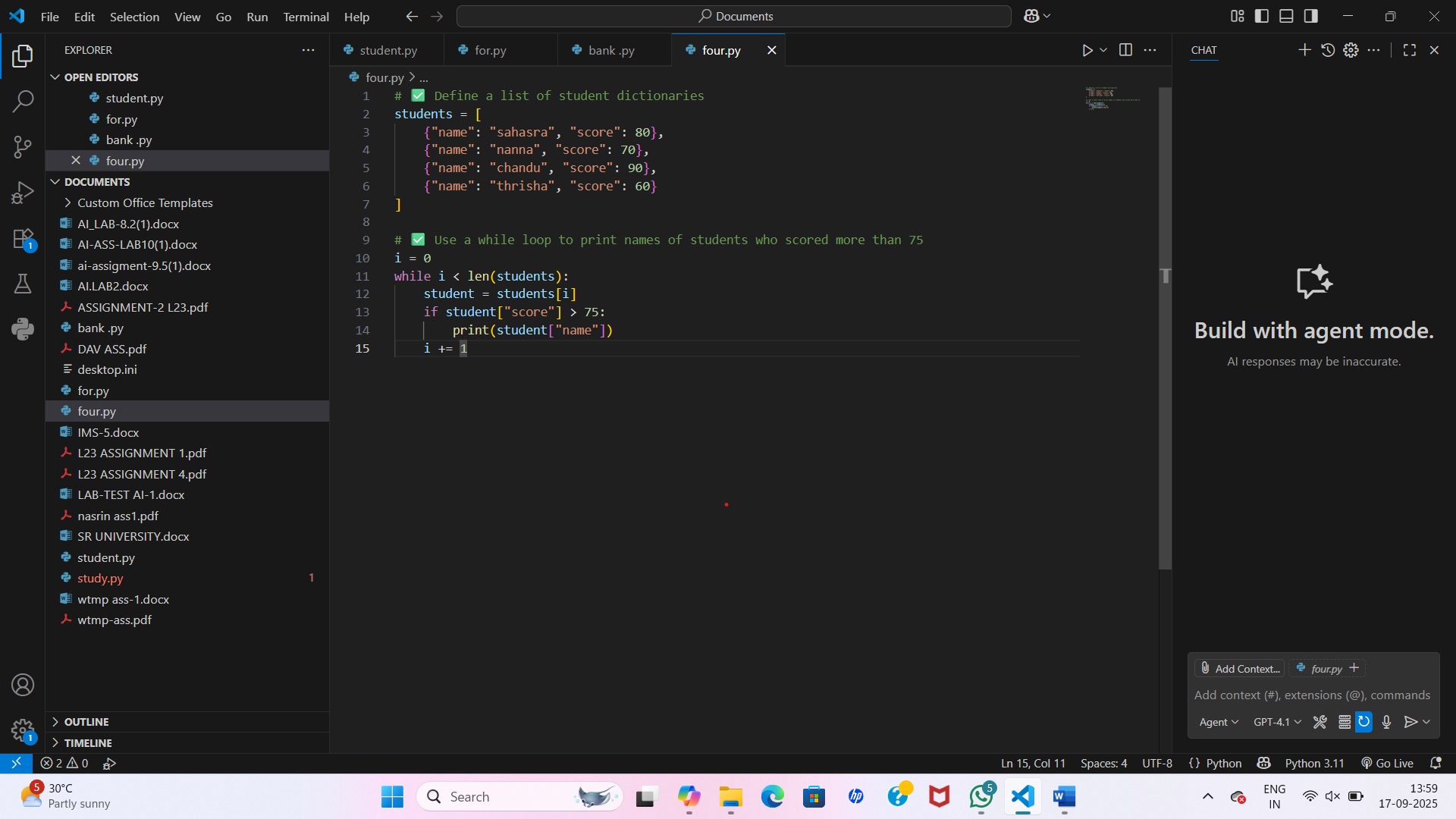
students = [("sahasra", 80), ("nanna", 70), ("chandu", 90), ("thrisha", 60)]

i = 0 while i < len(students):

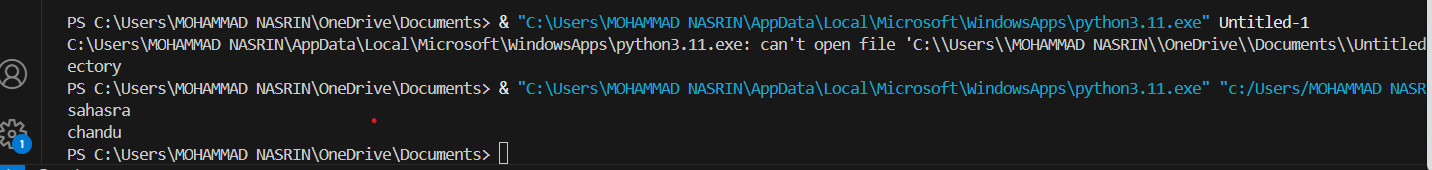
name, score = students[i] if score > 75:

print(name) i += 1

**Code Generated:**



**Outpit after executing code:**

****

**Observations:**

* **We uses while loop with counter i.**
* **The loop checks if score >75**
* **It will prints qualifying students**

**PROMPT:**

Begin writing a class Shopping Cart 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.

**Code:**

lass ShoppingCart: def \_\_init\_\_(self): self.items = [] def add\_item(self, name, price):

self.items.append((name, price)) print(f"added {name} to the cart") def

remove\_item(self, name):

initial\_len = len(self.items) self.items = [item for item in self.items if item[0] != name] if len(self.items) < initial\_len:

print(f"removed shoes from the cart{name}")

else:

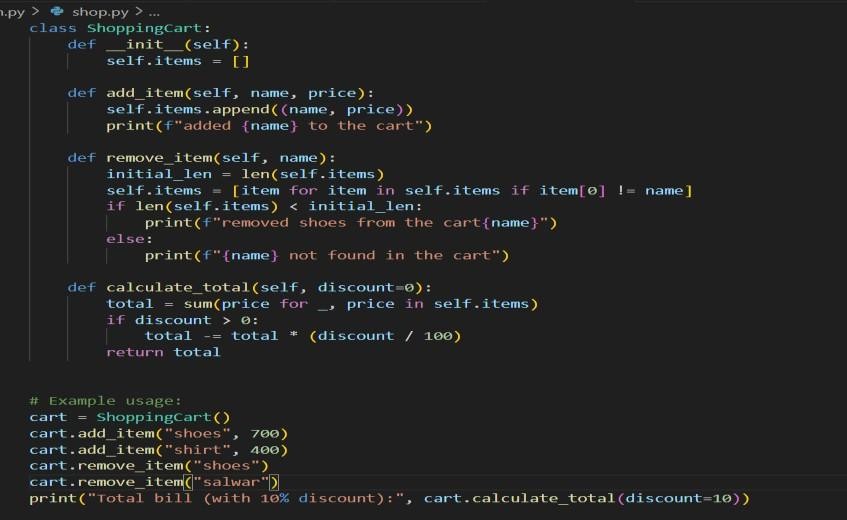
print(f"{name} not found in the cart") def calculate\_total(self, discount=0):

total = sum(price for \_, price in self.items) if discount > 0: total -= total \*

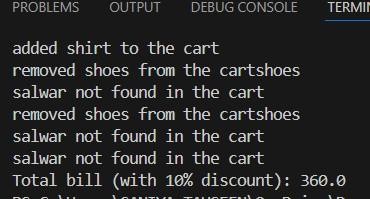
(discount / 100) return total

# Example usage: cart = ShoppingCart() cart.add\_item("shoes", 700) cart.add\_item("shirt", 400) cart.remove\_item("shoes") cart.remove\_item("salwar") print("Total bill (with 10% discount):", cart.calculate\_total(discount=10))

**Code Generated:**



**Output After executing Code:**



**Observations:**

If we want to add item use function-add\_item(): adds item to cart.

If we want to remove item use function remove\_item(): removes by name.

If we want to calculate the total use function calculate\_total(): loops through cart, applies discounts with if-elif.