SR UNIVERSITY

AI ASSIST CODING

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

ROLL NO:2503A51L13

NAME:BEGALA HASINI

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:

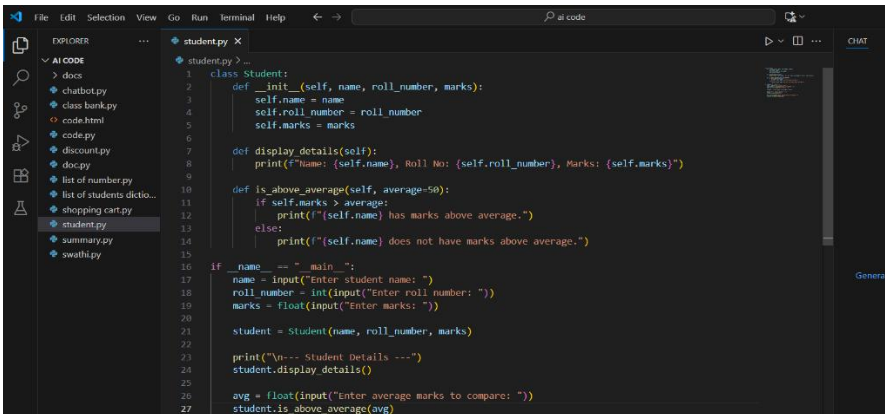
• Start a Python class named Student with attributes name, roll number, and marks, Prompt GitHub Copilot to complete methods for displaying details and checking if marks are above average.

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:
```

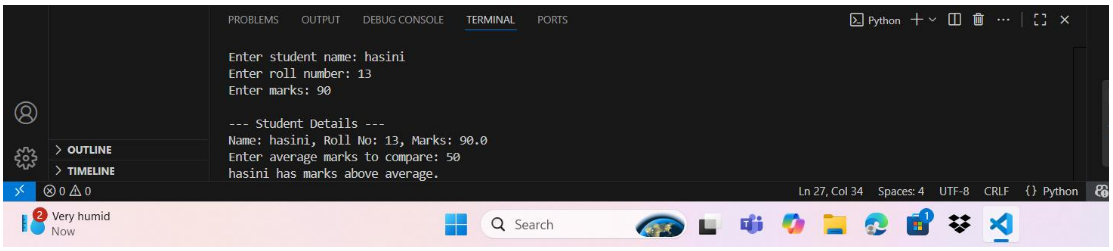
```
      print(f"{self.name} has marks above average.")

    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:



```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):
        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)
```

Output After executing Code:



```
Enter student name: hasini
Enter roll number: 13
Enter marks: 90

--- Student Details ---
Name: hasini, Roll No: 13, Marks: 90.0
Enter average marks to compare: 50
hasini has marks above average.
```

Observations:

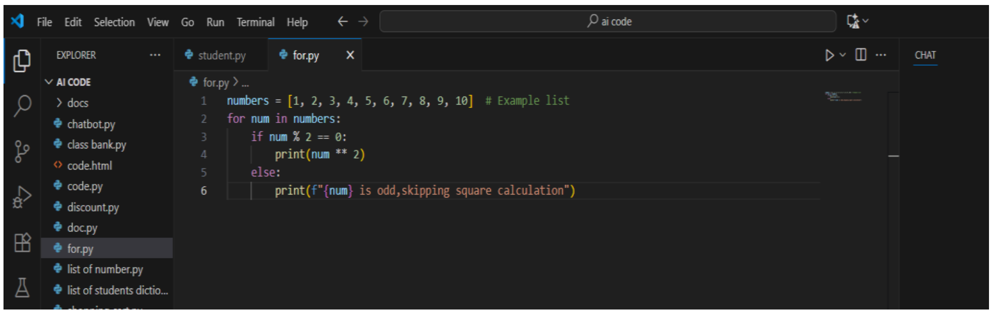
- A Student class is created with attributes **name**, **roll number**, and **marks**.
- The display_details() method neatly prints the student's details.
- The is_above_average() method compares the student's marks with a given average and prints the result.
- User input is taken for **name, roll number, marks, and average** at runtime, making the program interactive.

TASK #2:

Prompt Used:

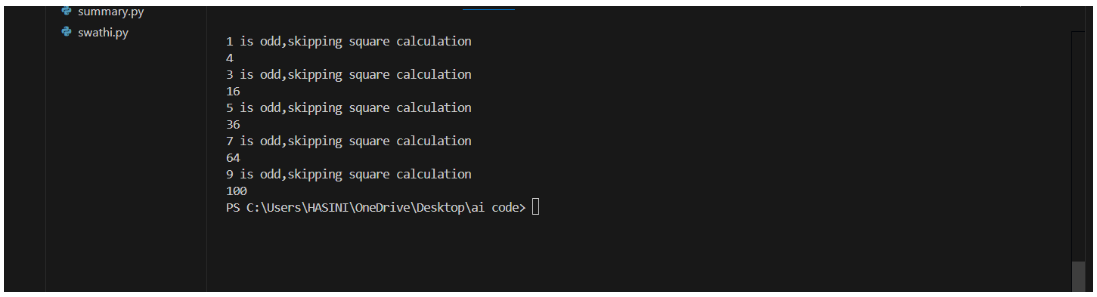
- Write the first two lines of a for loop to iterate through a list o f numbers. Suggest how to calculate and print the square of even numbers only.

Code Generated:



```python
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  # Example list
for num in numbers:
    if num % 2 == 0:
        print(num ** 2)
    else:
        print(f"{num} is odd,skipping square calculation")
```

Output After executing Code:



```
1 is odd,skipping square calculation
4
3 is odd,skipping square calculation
16
5 is odd,skipping square calculation
36
7 is odd,skipping square calculation
64
9 is odd,skipping square calculation
100
PS C:\Users\HASINI\OneDrive\Desktop\ai 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:

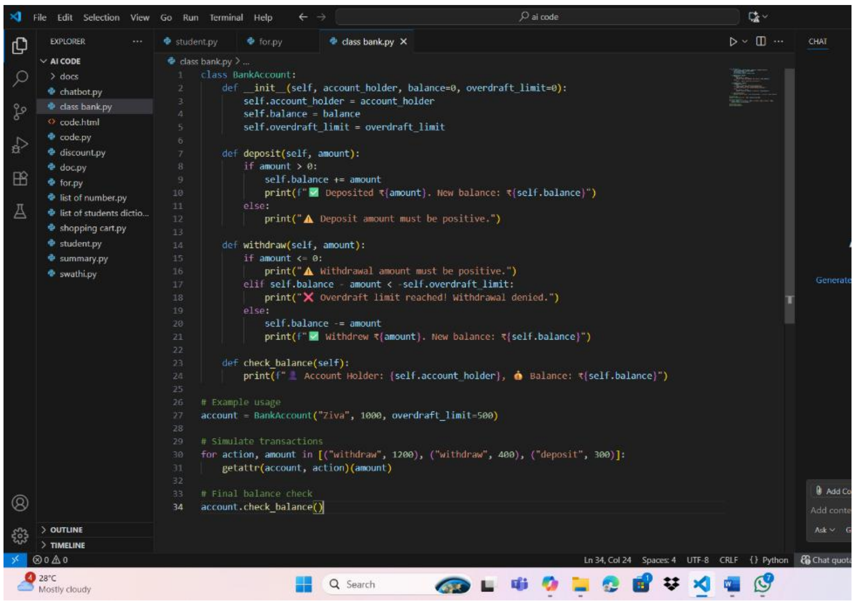
Prompt Used:

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

Code:

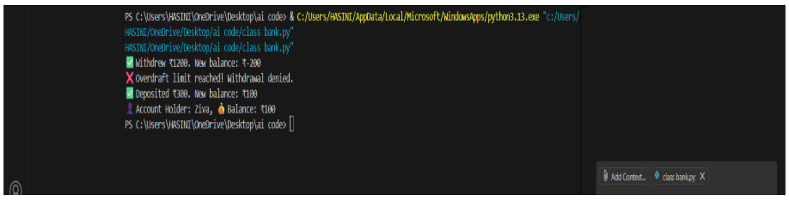
```
class BankAccount:     def __init__(self, account_holder,

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:



```python
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}")

# Example usage
account = BankAccount("Ziva", 1000, overdraft_limit=500)

# Simulate transactions
for action, amount in [("withdraw", 1200), ("withdraw", 400), ("deposit", 300)]:
    getattr(account, action)(amount)

# Final balance check
account.check_balance()
```

Output After executing Code:



```
PS C:\Users\HASINI\OneDrive\Desktop\ai code> & C:/Users/HASINI/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/
HASINI/OneDrive/Desktop/ai code/class bank.py"
HASINI/OneDrive/Desktop/ai code/class bank.py"
✅Withdrew ₹1200. New balance: ₹-200
❌Overdraft limit reached! Withdrawal denied.
✅Deposited ₹300. New balance: ₹100
👤Account Holder: Ziva, 💰Balance: ₹100
PS C:\Users\HASINI\OneDrive\Desktop\ai 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 = [("Pari", 80), ("Sam", 70), ("Katrina", 90), ("David", 60)]

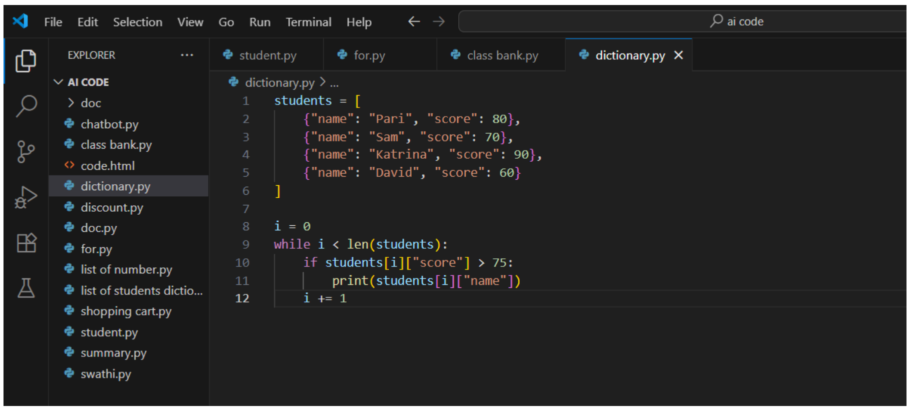
i = 0 while i < len(students):

name, score =

students[i] if score > 75:

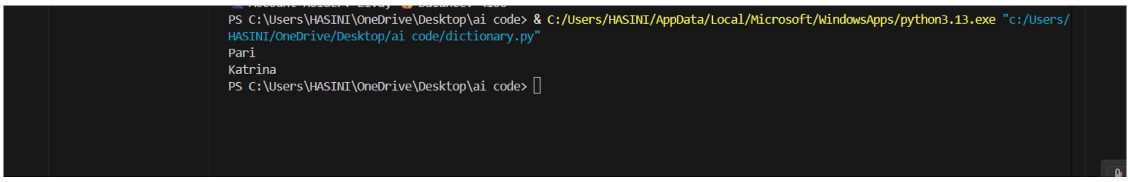
print(name) i += 1

Code Generated:



```python
students = [
    {"name": "Pari", "score": 80},
    {"name": "Sam", "score": 70},
    {"name": "Katrina", "score": 90},
    {"name": "David", "score": 60}
]

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

Output After executing Code:



```
PS C:\Users\HASINI\OneDrive\Desktop\ai code> & C:/Users/HASINI/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/
HASINI/OneDrive/Desktop/ai code/dictionary.py"
Pari
Katrina
PS C:\Users\HASINI\OneDrive\Desktop\ai code> 
```

Observations:

- We Uses while loop with counter i.
- The loop Checks if score > 75.
-

It will Prints qualifying students.

TASK#5:

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

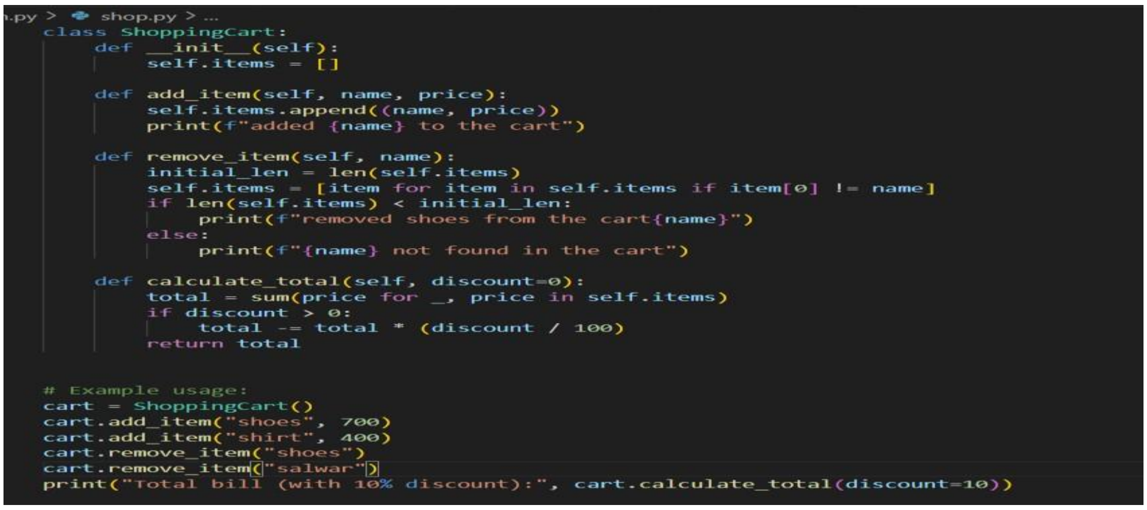
# Example usage: cart = ShoppingCart() cart.add_item("shoes", 700)

cart.add_item("shirt", 400) cart.remove_item("shoes")

cart.remove_item("salwar") print("Total bill (with 10% discount):",

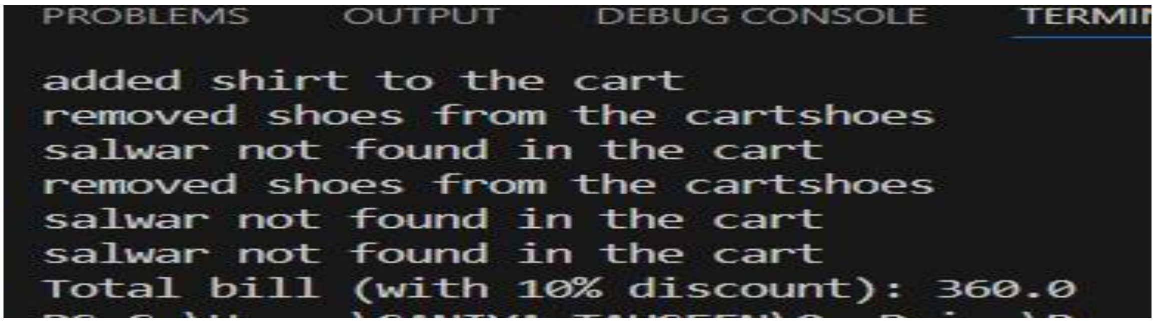
cart.calculate_total(discount=10))

Code Generated:



```
.py > shop.py > ...
class 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))
```

Output After executing Code:



```
added shirt to the cart
removed shoes from the cartshoes
salwar not found in the cart
removed shoes from the cartshoes
salwar not found in the cart
salwar not found in the cart
Total bill (with 10% discount): 360.0
```

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.