Lab 4: AI-Based Code Auto-Completion – Classes, Loops, and Conditionals in Python using GitHub Copilot

Course Code: 24CSBTB41
Course Title: Assistant Coding
Assignment No: 4.4
Roll Number: 2503A51L36

Task 1: Auto-Complete a Python Class for Bank Account

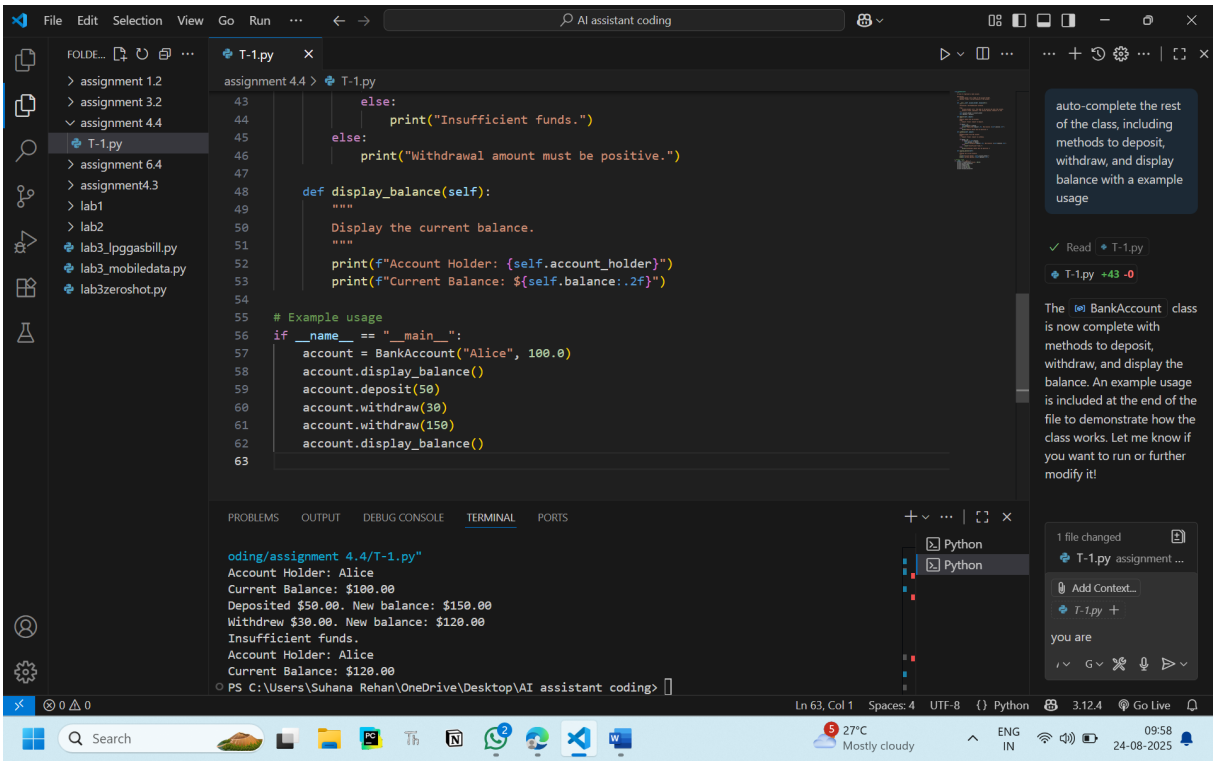
Prompt: Write a class definition comment and start the constructor for a class called BankAccount with account_holder and balance attributes. Use GitHub Copilot to auto-complete the rest of the class, including methods to deposit, withdraw, and display balance.

Code:

```
class BankAccount:
    """
    A class to represent a bank account.

    Attributes:
        account_holder (str): Name of the account holder.
        balance (float): Current balance of the account.
    """

    def __init__(self, account_holder, balance=0.0):
        """
        Initialize a new BankAccount instance.

        Args:
            account_holder (str): The name of the person who owns the account.
            balance (float, optional): The initial balance. Defaults to 0.0.
        """
```

```
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        """
        Deposit money into the account.
        Args:
            amount (float): Amount to deposit.
        """
        if amount > 0:
            self.balance += amount
            print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        """
        Withdraw money from the account.
        Args:
            amount (float): Amount to withdraw.
        """
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew ${amount:.2f}. New balance: ${self.balance:.2f}")
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def display_balance(self):
        """
        Display the current balance.
        """
        print(f"Account Holder: {self.account_holder}")
        print(f"Current Balance: ${self.balance:.2f}")

# Example usage
if __name__ == "__main__":
    account = BankAccount("Alice", 100.0)
    account.display_balance()
    account.deposit(50)
    account.withdraw(30)
    account.withdraw(150)
    account.display_balance()
```

OBSERVATION: The copilot completed the program when I just wrote class definition comment and constructor for a class called BankAccount with account_holder and balance attributes



Expected Output:

Account Holder: Alice
Current Balance: $100.00
Deposited $50.00. New balance: $150.00
Withdrew $30.00. New balance: $120.00
Insufficient funds.
Account Holder: Alice
Current Balance: $120.00

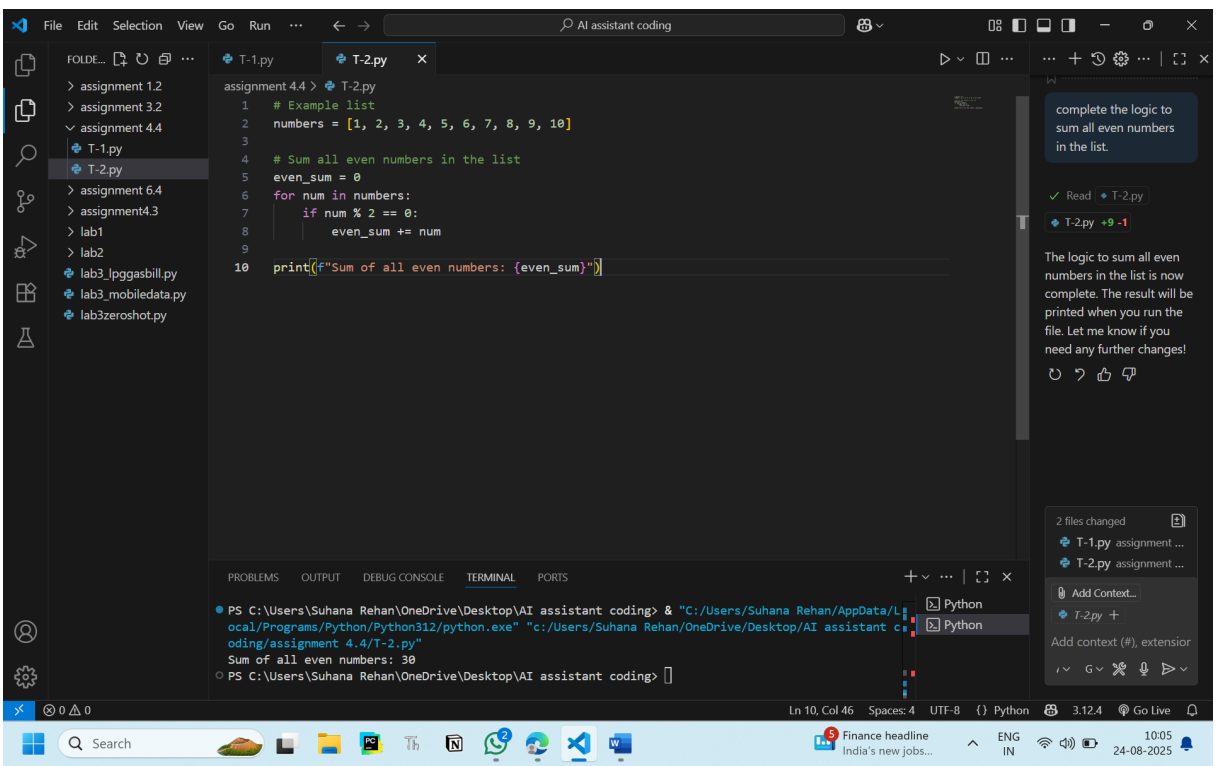
Task 2: Auto-Complete a For Loop to Sum Even Numbers in a List

Prompt: Write a comment and the initial line of a loop to iterate over a list. Allow GitHub Copilot to complete the logic to **sum all even numbers** in the list.

Code:

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

# Sum all even numbers in the list
even_sum = 0
for num in numbers:
    if num % 2 == 0:
        even_sum += num

print(f"Sum of all even numbers: {even_sum}")
```

OBSERVATION: GitHub Copilot to completed the logic to **sum all even numbers** in the list.



Expected Output:

Sum of all even numbers: 30

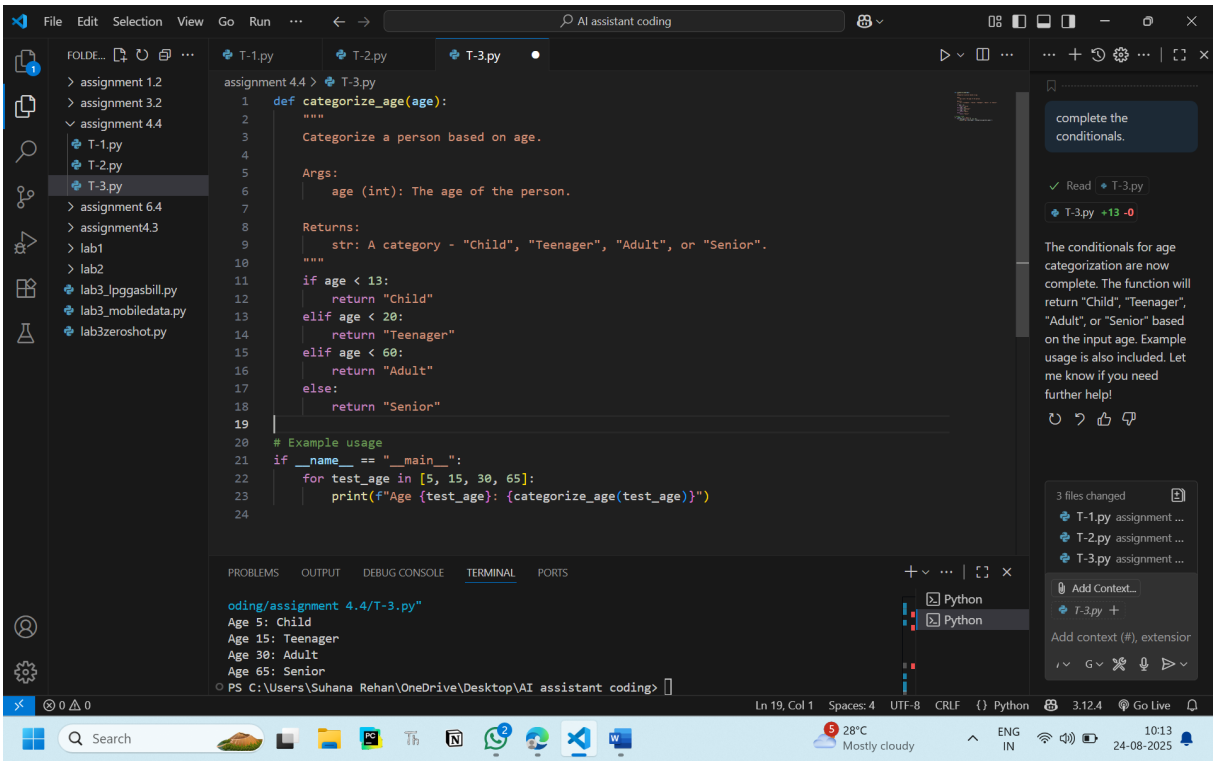
Task 3: Auto-Complete Conditional Logic to Check Age Group

Prompt: Start a function that takes age as input and returns whether the person is a **child**, **teenager**, **adult**, or **senior** using if-elif-else. Use Copilot to complete the conditionals.

Code:

```
def categorize_age(age):
    """
    Categorize a person based on age.

    Args:
        age (int): The age of the person.

    Returns:
        str: A category - "Child", "Teenager", "Adult", or "Senior".
    """
    if age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 60:
        return "Adult"
    else:
        return "Senior"

# Example usage
if __name__ == "__main__":
    for test_age in [5, 15, 30, 65]:
        print(f"Age {test_age}: {categorize_age(test_age)}")
```

OBSERVATION: I started a function that takes age as input and returns whether the person is a **child**, **teenager**, **adult**, or **senior** using if-elif-else and used Copilot to complete the conditionals. And the copilot completed the the conditionals with the clear logic



Expected Output:

Age 5: Child
Age 15: Teenager
Age 30: Adult
Age 65: Senior

Task 4: : Auto-Complete a While Loop to Reverse Digits of a Number

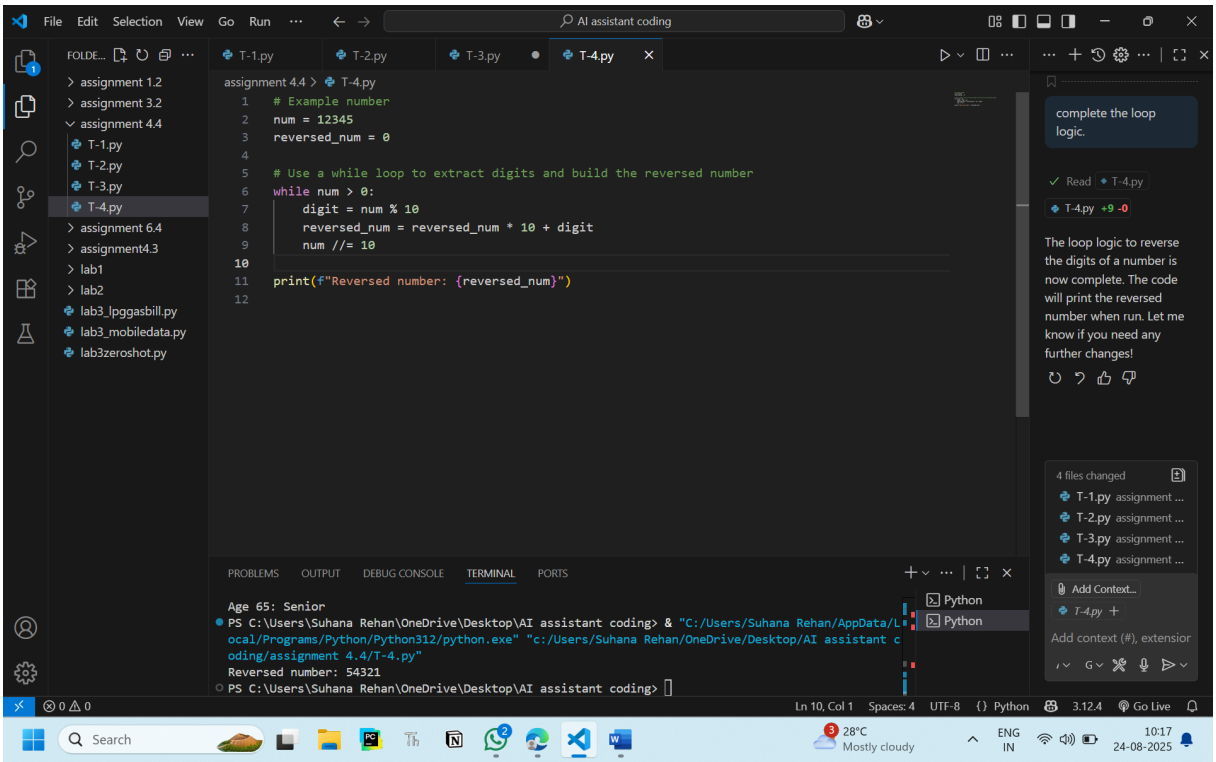
Prompt: Write a comment and start a while loop to reverse the digits of a number. Let Copilot complete the loop logic.

Code:

```
# Example number
num = 12345
reversed_num = 0

# Use a while loop to extract digits and build the reversed number
while num > 0:
    digit = num % 10
    reversed_num = reversed_num * 10 + digit
```

```
    num //= 10

print(f"Reversed number: {reversed_num}")
```

OBSERVATION: AI completed the logic when I just wrote the while loop



Expected Output:

Reversed number: 54321

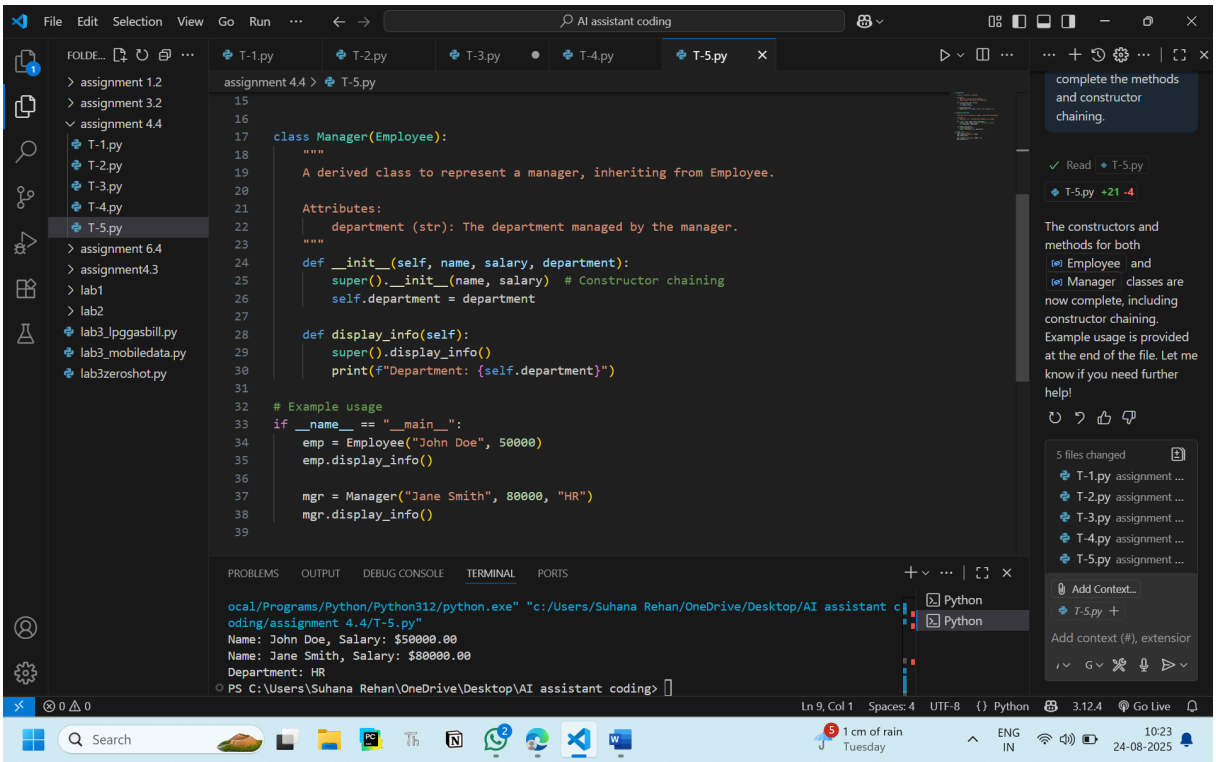
Task 5: Auto-Complete Class with Inheritance (Employee → Manager)

Prompt: Begin a class Employee with attributes name and salary. Then, start a derived class Manager that inherits from Employee and adds a department. Let GitHub Copilot complete the methods and constructor chaining.

Code:

```
class Employee:
    """
    A class to represent an employee.

    Attributes:
        name (str): The name of the employee.
        salary (float): The salary of the employee.
    """
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def display_info(self):
        print(f"Name: {self.name}, Salary: ${self.salary:.2f}")

class Manager(Employee):
    """
    A derived class to represent a manager, inheriting from Employee.

    Attributes:
        department (str): The department managed by the manager.
    """
    def __init__(self, name, salary, department):
        super().__init__(name, salary)  # Constructor chaining
        self.department = department

    def display_info(self):
        super().display_info()
        print(f"Department: {self.department}")

# Example usage
if __name__ == "__main__":
    emp = Employee("John Doe", 50000)
    emp.display_info()

    mgr = Manager("Jane Smith", 80000, "HR")
    mgr.display_info()
```

OBSERVATION: AI completed the the methods and constructor chaining.



Expected Output:

Name: John Doe, Salary: $50000.00

Name: Jane Smith, Salary: $80000.00

Department: HR