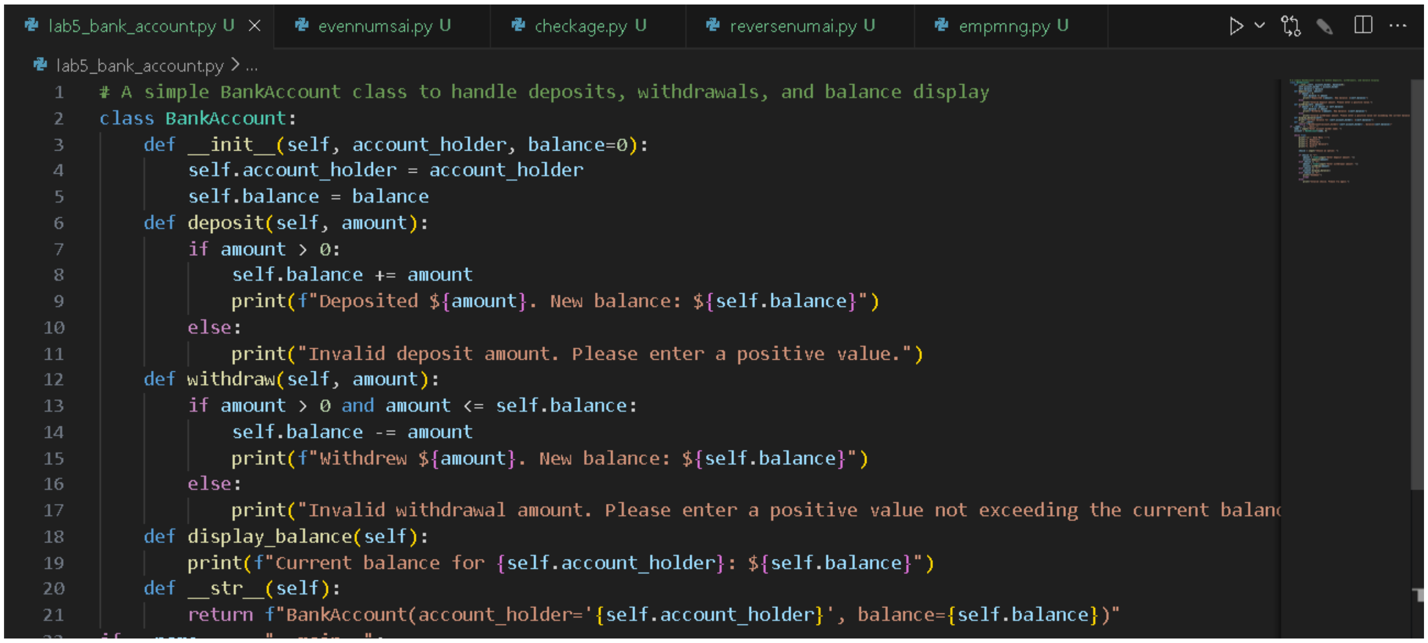
LAB 4.4 ASSIGNMENTS

TASK 1:

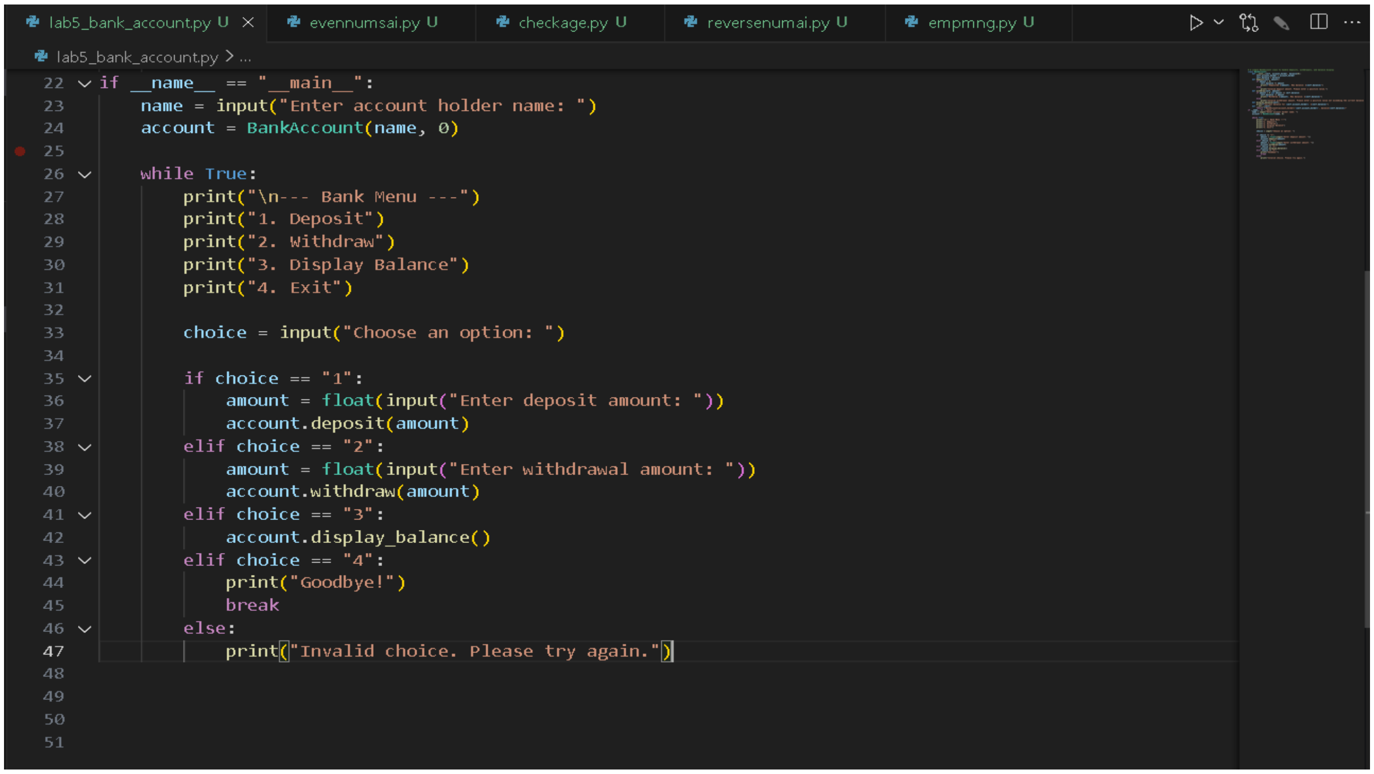
Auto-Complete a Python Class for Bank Account
Write a class definition comment and start the constructor for a class called BankAccount with account_holder and balance attributes. Use GitHub Copilot to auto-complete the rest of the class, including methods to deposit, withdraw, and display balance.

CODE (CLASS):



```python
# A simple BankAccount class to handle deposits, withdrawals, and balance display
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("Invalid deposit amount. Please enter a positive value.")
    def withdraw(self, amount):
        if amount > 0 and amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew ${amount}. New balance: ${self.balance}")
        else:
            print("Invalid withdrawal amount. Please enter a positive value not exceeding the current balanc
    def display_balance(self):
        print(f"Current balance for {self.account_holder}: ${self.balance}")
    def __str__(self):
        return f"BankAccount(account_holder='{self.account_holder}', balance={self.balance})"
```

CODE (INPUTS):



```python
if __name__ == "__main__":
    name = input("Enter account holder name: ")
    account = BankAccount(name, 0)

    while True:
        print("\n--- Bank Menu ---")
        print("1. Deposit")
        print("2. Withdraw")
        print("3. Display Balance")
        print("4. Exit")

        choice = input("Choose an option: ")

        if choice == "1":
            amount = float(input("Enter deposit amount: "))
            account.deposit(amount)
        elif choice == "2":
            amount = float(input("Enter withdrawal amount: "))
            account.withdraw(amount)
        elif choice == "3":
            account.display_balance()
        elif choice == "4":
            print("Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")
```

CODE EXPLANATION:

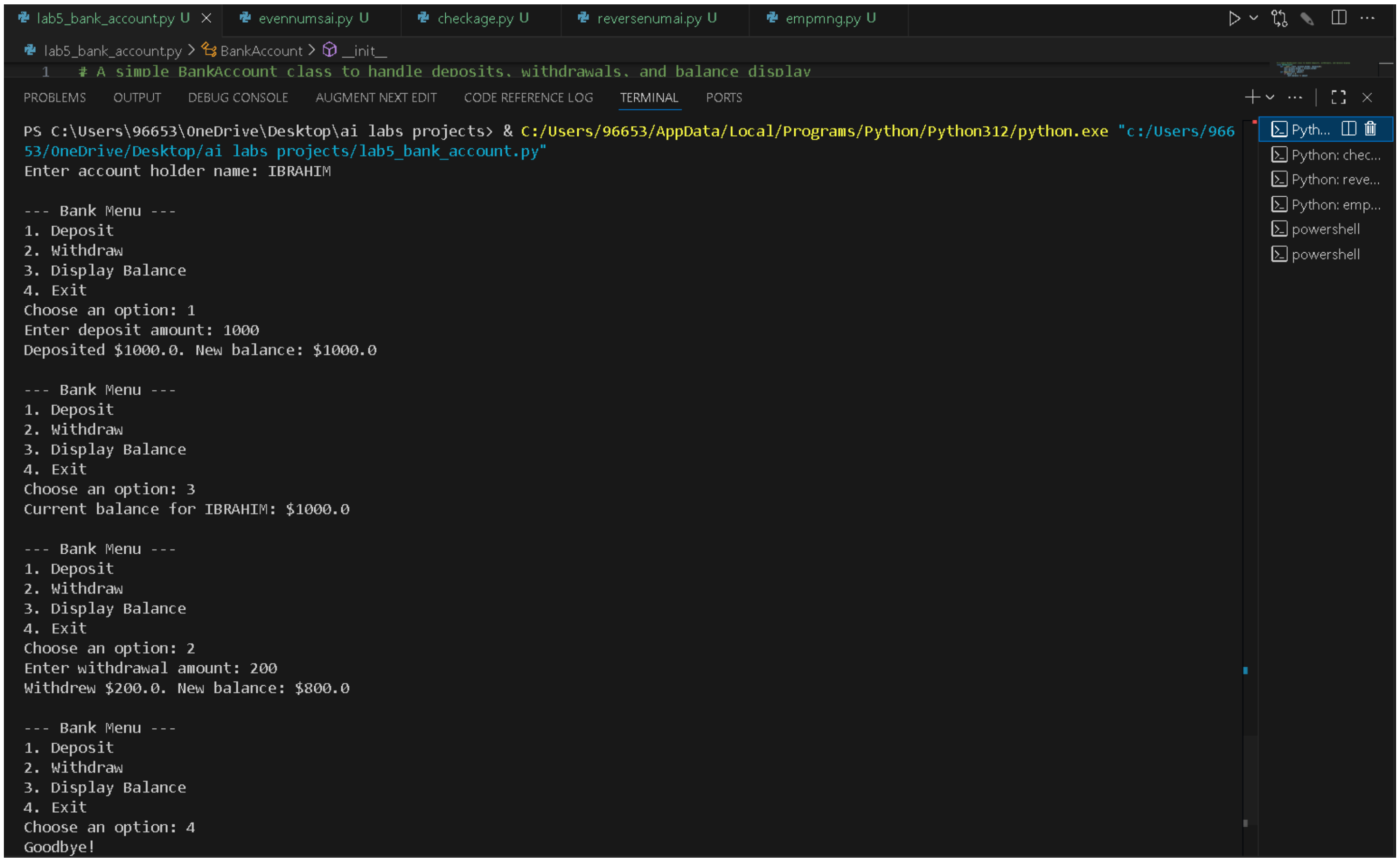
This code defines a simple bank account management system in Python using a class called BankAccount. Here's a breakdown:

- The BankAccount class has methods to deposit money, withdraw money, and display the current balance.
- The constructor (init) initializes the account holder's name and starting balance (default is 0).
- deposit(amount): Adds the specified amount to the balance if it's positive.
- withdraw(amount): Subtracts the specified amount from the balance if it's positive and does not exceed the current balance.
- display_balance(): Prints the current balance and account holder's name.
- str(): Returns a string representation of the account for debugging or display.

The main section:

- Prompts the user to enter their name and creates a BankAccount object.
- Displays a menu in a loop, allowing the user to deposit, withdraw, check balance, or exit.
- Handles invalid input and ensures only valid transactions are processed.
- This script is interactive and runs in the terminal, providing basic banking operations for a single user.

CODE OUTPUTS:



```
# A simple BankAccount class to handle deposits, withdrawals, and balance display
```

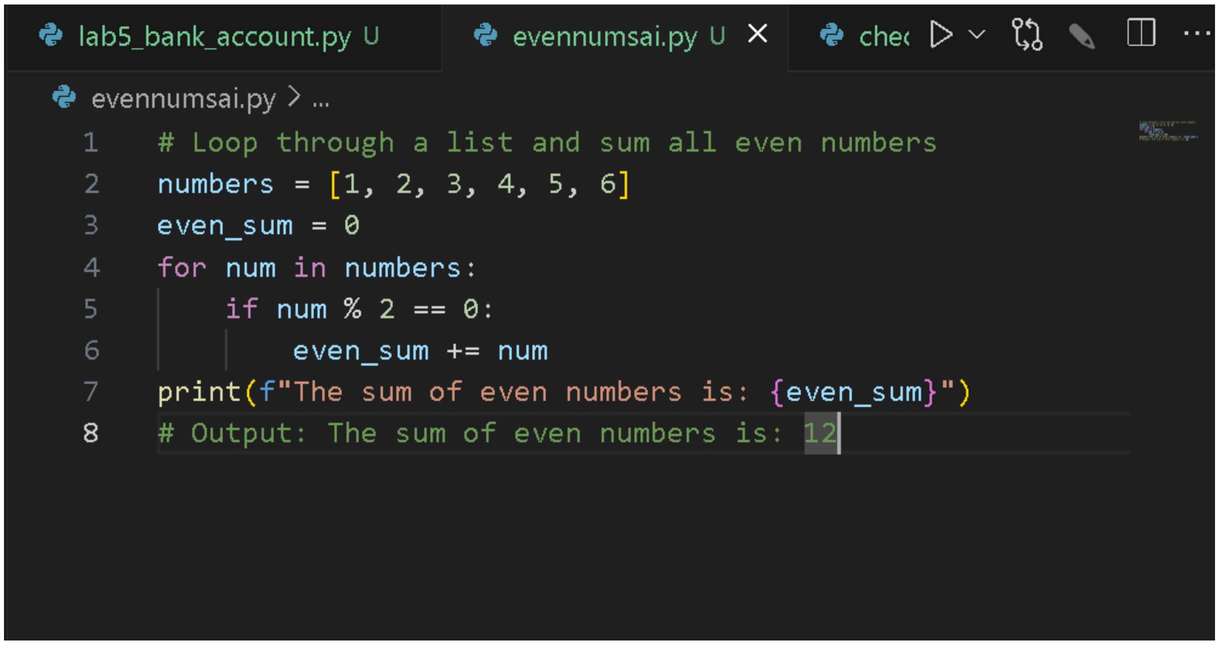
```
PS C:\Users\96653\OneDrive\Desktop\ai labs projects> & C:/Users/96653/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/966
53/OneDrive/Desktop/ai labs projects/lab5_bank_account.py"
Enter account holder name: IBRAHIM

--- Bank Menu ---
1. Deposit
2. Withdraw
3. Display Balance
4. Exit
Choose an option: 1
Enter deposit amount: 1000
Deposited $1000.0. New balance: $1000.0

--- Bank Menu ---
1. Deposit
2. Withdraw
3. Display Balance
4. Exit
Choose an option: 3
Current balance for IBRAHIM: $1000.0

--- Bank Menu ---
1. Deposit
2. Withdraw
3. Display Balance
4. Exit
Choose an option: 2
Enter withdrawal amount: 200
Withdrew $200.0. New balance: $800.0

--- Bank Menu ---
1. Deposit
2. Withdraw
3. Display Balance
4. Exit
Choose an option: 4
Goodbye!
```

TASK 2:

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

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:



```python
# Loop through a list and sum all even numbers
numbers = [1, 2, 3, 4, 5, 6]
even_sum = 0
for num in numbers:
    if num % 2 == 0:
        even_sum += num
print(f"The sum of even numbers is: {even_sum}")
# Output: The sum of even numbers is: 12
```

CODE EXPLANATION:

This code calculates the sum of all even numbers in a given list:

It defines a list numbers = [1, 2, 3, 4, 5, 6].

It initializes even_sum to 0.

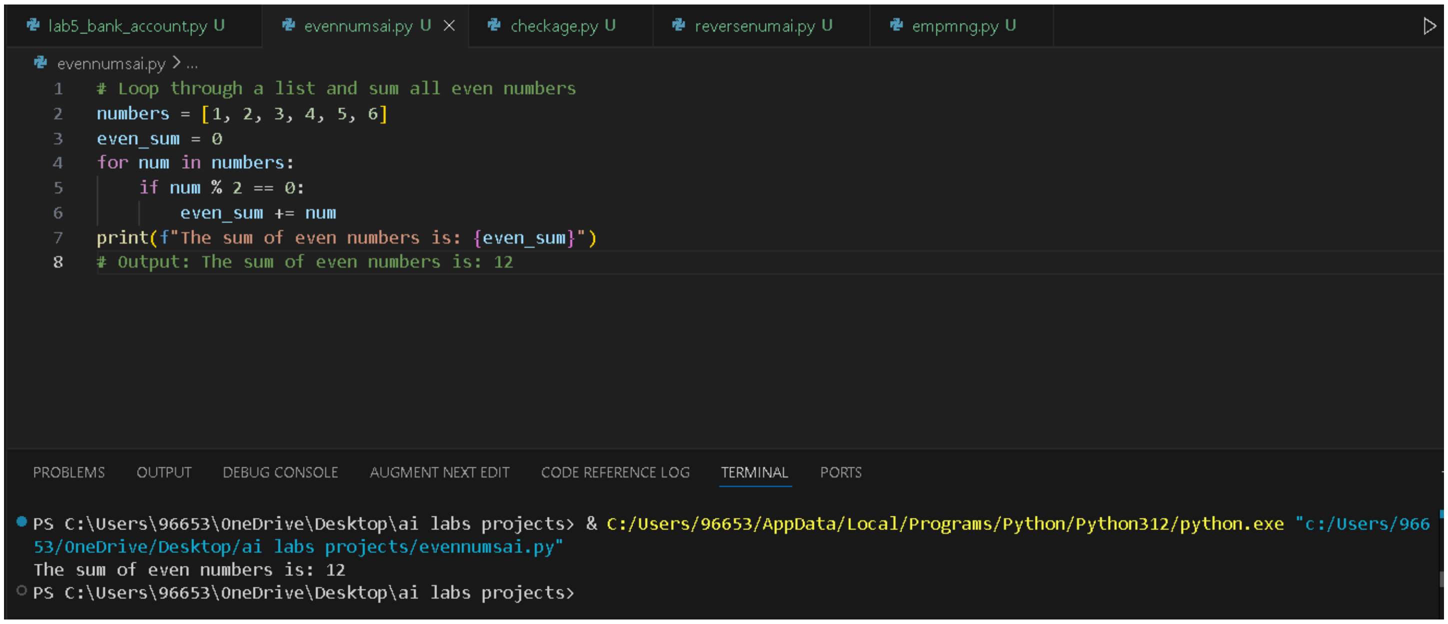
It loops through each number in the list.

If the number is even (num % 2 == 0), it adds it to even_sum.

After the loop, it prints the total sum of even numbers.

For this list, the even numbers are 2, 4, and 6, so the output is: The sum of even numbers is: 12.

CODE OUTPUTS:



```python
# Loop through a list and sum all even numbers
numbers = [1, 2, 3, 4, 5, 6]
even_sum = 0
for num in numbers:
    if num % 2 == 0:
        even_sum += num
print(f"The sum of even numbers is: {even_sum}")
# Output: The sum of even numbers is: 12
```

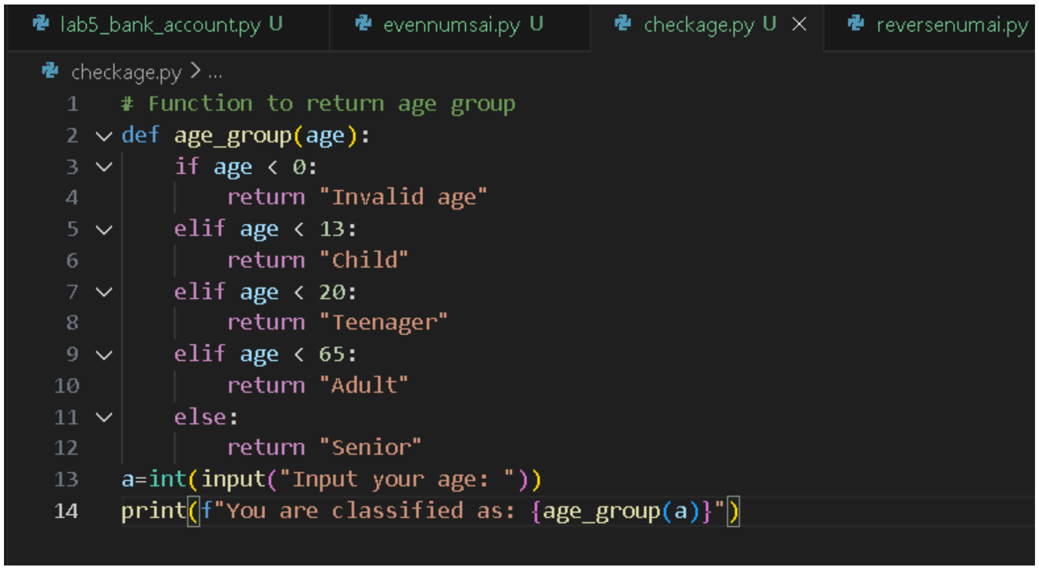
```
PS C:\Users\96653\OneDrive\Desktop\ai labs projects> & C:/Users/96653/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/966
53/OneDrive/Desktop/ai labs projects/evennumsai.py"
The sum of even numbers is: 12
PS C:\Users\96653\OneDrive\Desktop\ai labs projects>
```

TASK 3:

Auto-Complete Conditional Logic to Check Age Group

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:



```python
# Function to return age group
def age_group(age):
    if age < 0:
        return "Invalid age"
    elif age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 65:
        return "Adult"
    else:
        return "Senior"
a=int(input("Input your age: "))
print(f"You are classified as: {age_group(a)}")
```

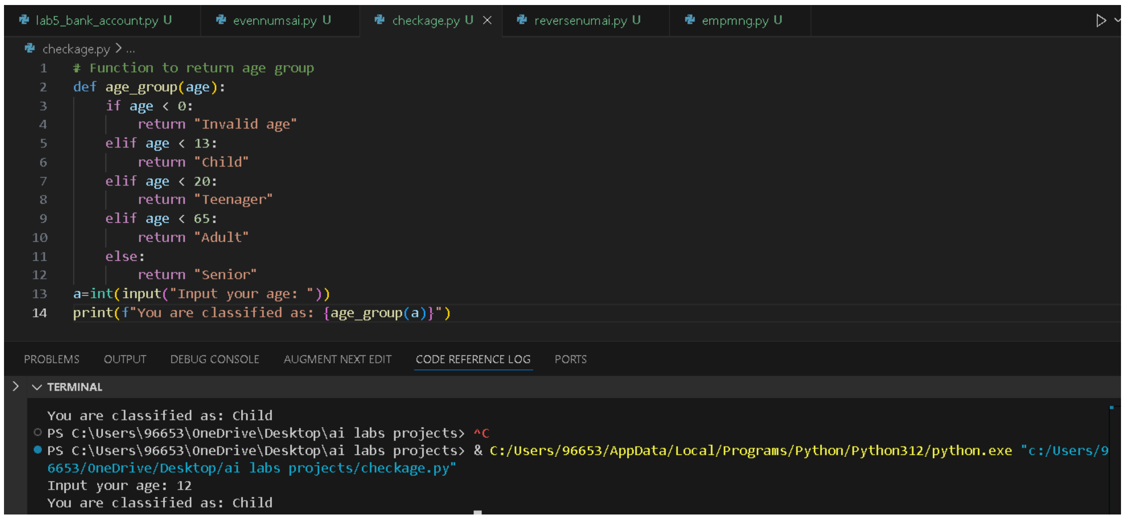
CODE EXPLANATION:

This code classifies a person's age group based on their input:

1. The function age_group(age) checks the value of age and returns a string:
   - If age is less than 0, it returns "Invalid age".
   - If age is less than 13, it returns "Child".
   - If age is less than 20, it returns "Teenager".
   - If age is less than 65, it returns "Adult".
   - Otherwise, it returns "Senior".
2. The user is prompted to input their age, which is converted to an integer.
3. The program prints the classification result using the age_group function.

In summary, it asks for your age and tells you if you are a child, teenager, adult, or senior.

CODE OUTPUTS:



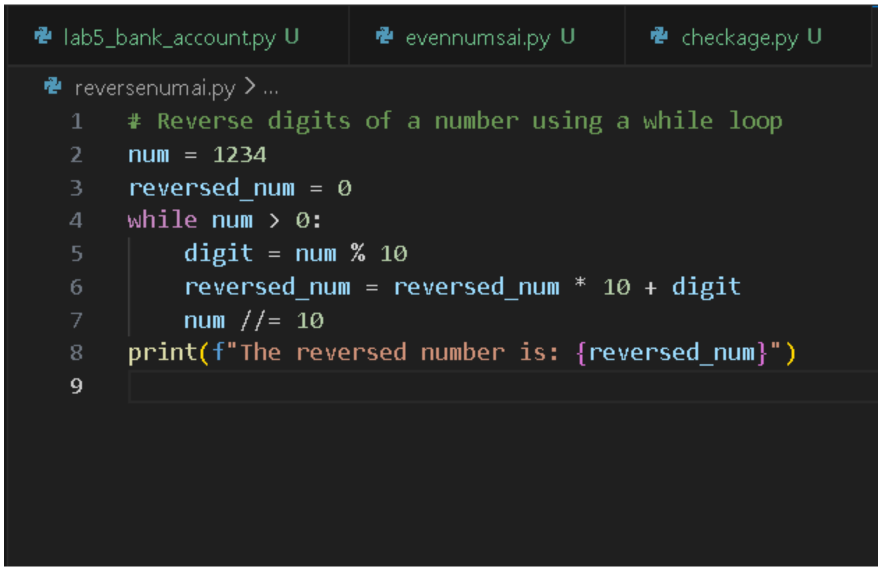
```python
# Function to return age group
def age_group(age):
    if age < 0:
        return "Invalid age"
    elif age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 65:
        return "Adult"
    else:
        return "Senior"
a=int(input("Input your age: "))
print(f"You are classified as: {age_group(a)}")
```

```
You are classified as: Child
PS C:\Users\96653\OneDrive\Desktop\ai labs projects> ^C
PS C:\Users\96653\OneDrive\Desktop\ai labs projects> & C:/Users/96653/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/96653/OneDrive/Desktop/ai labs projects/checkage.py"
Input your age: 12
You are classified as: Child
```

TASK 4:

Auto-Complete a While Loop to Reverse Digits of a Number
Write a comment and start a while loop to reverse the digits of a number. Let Copilot complete the loop logic.

CODE:



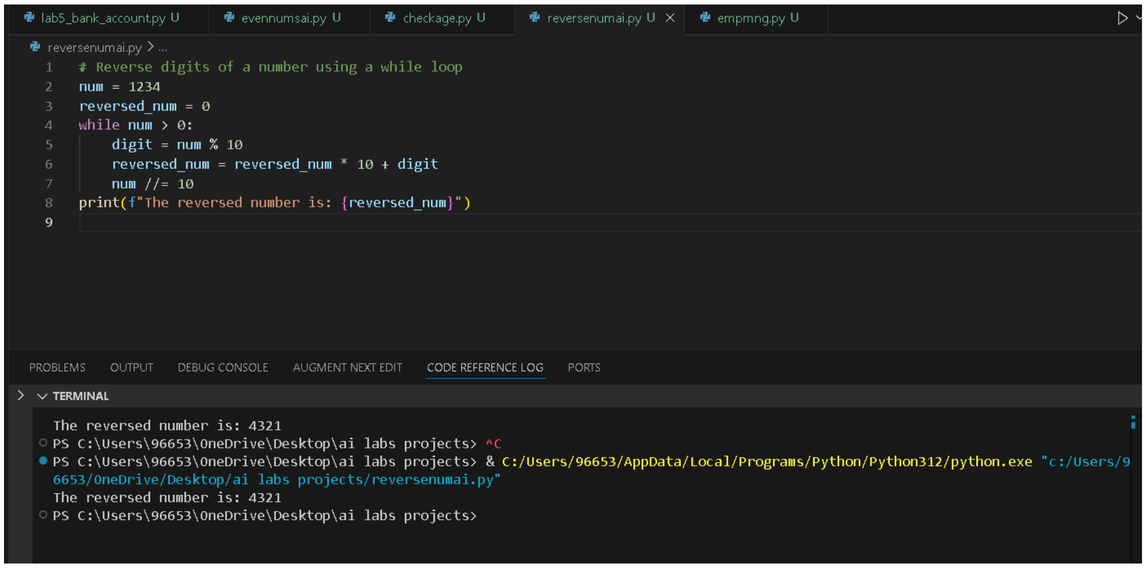
```python
# Reverse digits of a number using a while loop
num = 1234
reversed_num = 0
while num > 0:
    digit = num % 10
    reversed_num = reversed_num * 10 + digit
    num //= 10
print(f"The reversed number is: {reversed_num}")
```

CODE EXPLANATION:

This code reverses the digits of a number using a while loop:

1. It starts with num = 1234 and reversed_num = 0.
2. While num is greater than 0:
    - It gets the last digit of num using num % 10.
    - It adds this digit to reversed_num, shifting previous digits left (reversed_num * 10 + digit).
    - It removes the last digit from num using integer division (num //= 10).
3. After the loop, reversed_num contains the digits of the original number in reverse order.
4. It prints: The reversed number is: 4321.

CODE OUTPUTS:



```python
# Reverse digits of a number using a while loop
num = 1234
reversed_num = 0
while num > 0:
    digit = num % 10
    reversed_num = reversed_num * 10 + digit
    num //= 10
print(f"The reversed number is: {reversed_num}")
```

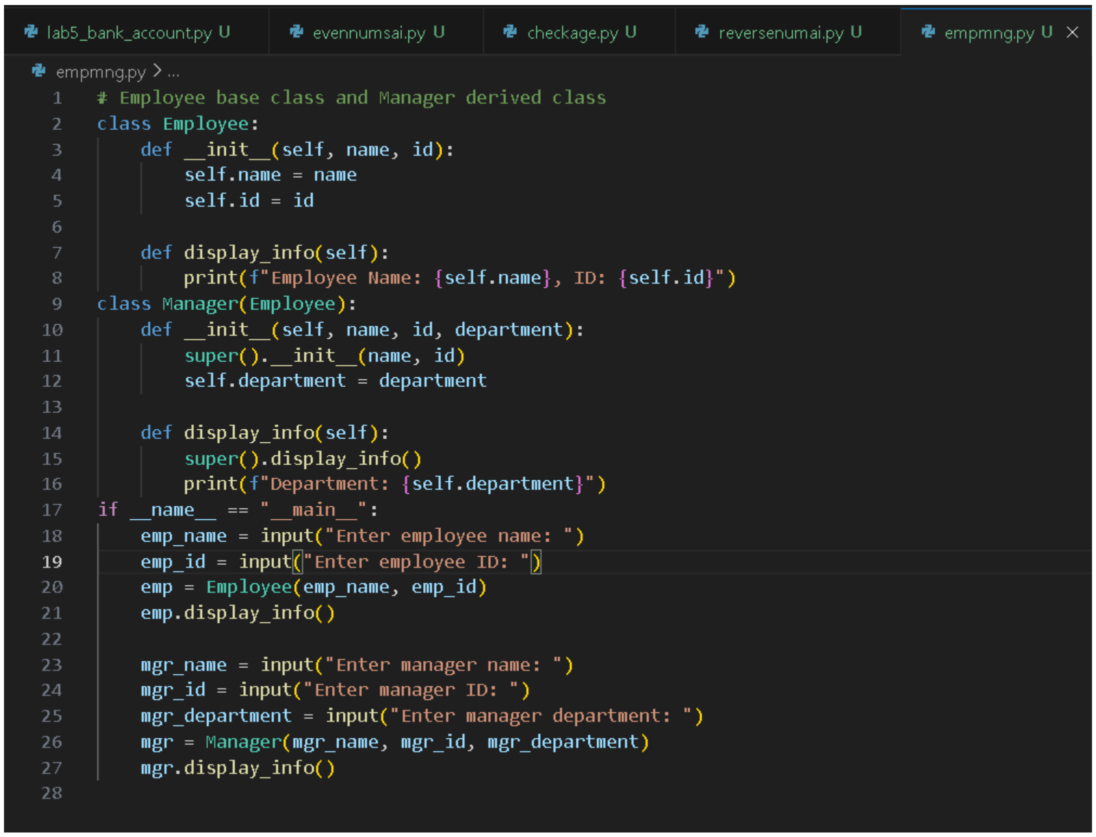
```
The reversed number is: 4321
PS C:\Users\96653\OneDrive\Desktop\ai labs projects> ^C
PS C:\Users\96653\OneDrive\Desktop\ai labs projects> & C:/Users/96653/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/9
6653/OneDrive/Desktop/ai labs projects/reversenumai.py"
The reversed number is: 4321
PS C:\Users\96653\OneDrive\Desktop\ai labs projects>
```

TASK 5:

Auto-Complete Class with Inheritance (Employee → Manager)

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

CODE:



```python
# Employee base class and Manager derived class
class Employee:
    def __init__(self, name, id):
        self.name = name
        self.id = id

    def display_info(self):
        print(f"Employee Name: {self.name}, ID: {self.id}")
class Manager(Employee):
    def __init__(self, name, id, department):
        super().__init__(name, id)
        self.department = department

    def display_info(self):
        super().display_info()
        print(f"Department: {self.department}")
if __name__ == "__main__":
    emp_name = input("Enter employee name: ")
    emp_id = input("Enter employee ID: ")
    emp = Employee(emp_name, emp_id)
    emp.display_info()

    mgr_name = input("Enter manager name: ")
    mgr_id = input("Enter manager ID: ")
    mgr_department = input("Enter manager department: ")
    mgr = Manager(mgr_name, mgr_id, mgr_department)
    mgr.display_info()
```

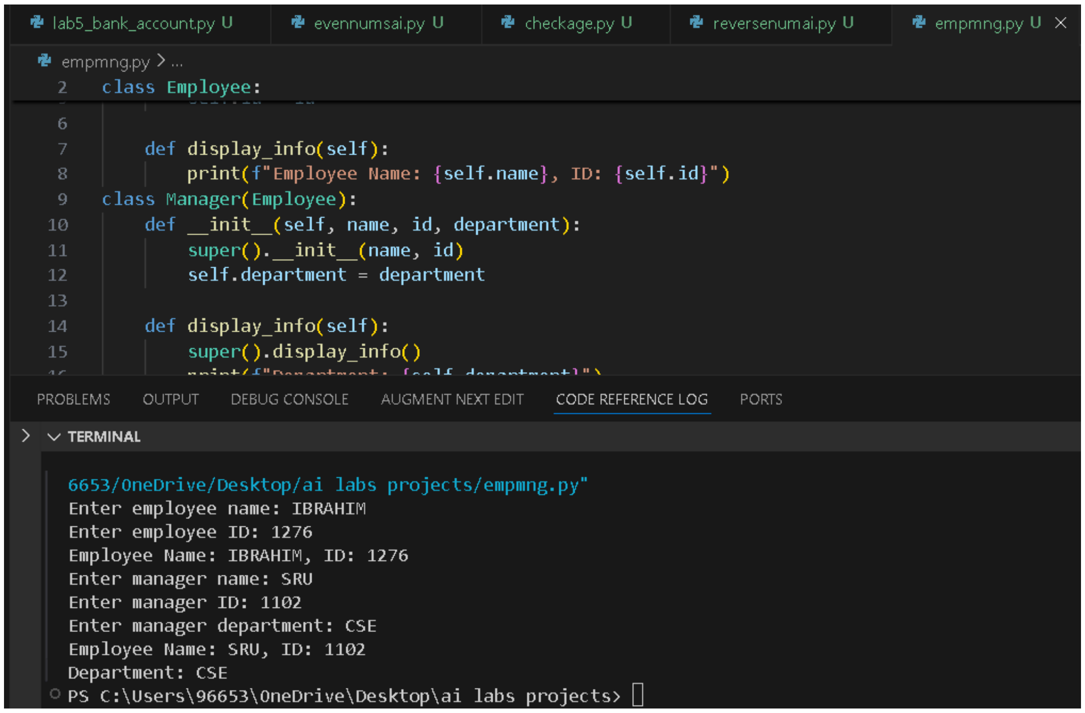
CODE EXPLANATION:

This code demonstrates object-oriented programming with inheritance in Python:

1. It defines a base class Employee with attributes name and id, and a method display_info() to print employee details.
2. It defines a derived class Manager that inherits from Employee, adds a department attribute, and overrides display_info() to also print the department.
3. In the main section, the program:
   - Prompts the user to enter an employee's name and ID, creates an Employee object, and displays its info.
   - Prompts for a manager's name, ID, and department, creates a Manager object, and displays its info.

In summary, the code shows how to use inheritance and method overriding to manage employee and manager information.

CODE OUTPUTS:



```python
class Employee:

    def display_info(self):
        print(f"Employee Name: {self.name}, ID: {self.id}")
class Manager(Employee):
    def __init__(self, name, id, department):
        super().__init__(name, id)
        self.department = department

    def display_info(self):
        super().display_info()
```

```
6653/OneDrive/Desktop/ai labs projects/empmng.py"
Enter employee name: IBRAHIM
Enter employee ID: 1276
Employee Name: IBRAHIM, ID: 1276
Enter manager name: SRU
Enter manager ID: 1102
Enter manager department: CSE
Employee Name: SRU, ID: 1102
Department: CSE
PS C:\Users\96653\OneDrive\Desktop\ai labs projects>
```