

SR UNIVERSITY

AI ASSIST CODING

Lab-4.4

ROLL NO:2503A51L13

NAME: B.Hasini

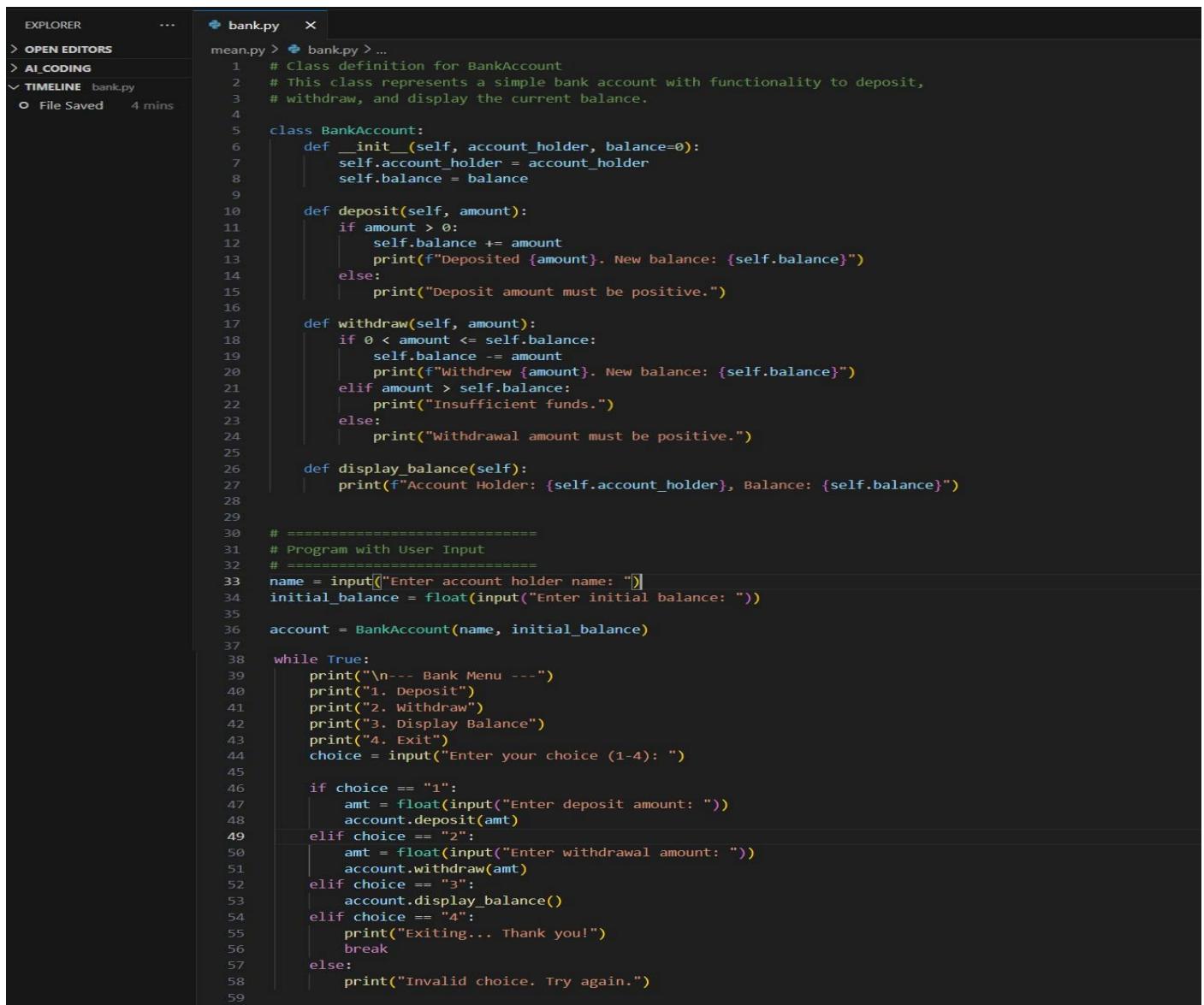
BATCH:19

TASK #1:

Prompt Used:

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 Generated:



The screenshot shows a code editor interface with a dark theme. On the left, there's an Explorer sidebar showing files like 'mean.py' and 'bank.py'. The main area displays the following Python code:

```
EXPLORER      ...
> OPEN EDITORS
> AI CODING
TIMELINE bank.py
File Saved   4 mins

bank.py  x
mean.py > bank.py > ...
1  # Class definition for BankAccount
2  # This class represents a simple bank account with functionality to deposit,
3  # withdraw, and display the current balance.
4
5  class BankAccount:
6      def __init__(self, account_holder, balance=0):
7          self.account_holder = account_holder
8          self.balance = balance
9
10     def deposit(self, amount):
11         if amount > 0:
12             self.balance += amount
13             print(f"Deposited {amount}. New balance: {self.balance}")
14         else:
15             print("Deposit amount must be positive.")
16
17     def withdraw(self, amount):
18         if 0 < amount <= self.balance:
19             self.balance -= amount
20             print(f"Withdrew {amount}. New balance: {self.balance}")
21         elif amount > self.balance:
22             print("Insufficient funds.")
23         else:
24             print("Withdrawal amount must be positive.")
25
26     def display_balance(self):
27         print(f"Account Holder: {self.account_holder}, Balance: {self.balance}")
28
29
30 # =====
31 # Program with User Input
32 # =====
33 name = input("Enter account holder name: ")
34 initial_balance = float(input("Enter initial balance: "))
35
36 account = BankAccount(name, initial_balance)
37
38 while True:
39     print("\n--- Bank Menu ---")
40     print("1. Deposit")
41     print("2. Withdraw")
42     print("3. Display Balance")
43     print("4. Exit")
44     choice = input("Enter your choice (1-4): ")
45
46     if choice == "1":
47         amt = float(input("Enter deposit amount: "))
48         account.deposit(amt)
49     elif choice == "2":
50         amt = float(input("Enter withdrawal amount: "))
51         account.withdraw(amt)
52     elif choice == "3":
53         account.display_balance()
54     elif choice == "4":
55         print("Exiting... Thank you!")
56         break
57     else:
58         print("Invalid choice. Try again.")
```

Output After executing Code:

```
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> & "C:/Users/SANIYA TAHSEEN/AppData/Local/Programs/Python/Python37/python.exe" "c:/Users/SANIYA TAHSEEN/OneDrive/Documents/AI_CODING/mean.py/bank.py"
Enter account holder name: Simra
Enter initial balance: 1000

--- Bank Menu ---
1. Deposit
2. Withdraw
3. Display Balance
4. Exit
Enter your choice (1-4): 1
Enter deposit amount: 500
Deposited 500.0. New balance: 1500.0

--- Bank Menu ---
1. Deposit
2. Withdraw
3. Display Balance
4. Exit
Enter your choice (1-4): 2
Enter withdrawal amount: 100
Withdrew 100.0. New balance: 1400.0

--- Bank Menu ---
1. Deposit
2. Withdraw
3. Display Balance
4. Exit
Enter your choice (1-4): 3
Account Holder: Simra, Balance: 1400.0

--- Bank Menu ---
1. Deposit
2. Withdraw
3. Display Balance
4. Exit
Enter your choice (1-4): 4
Exiting... Thank you!
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING>
```

powershell
 powershell
 Python

Observations:

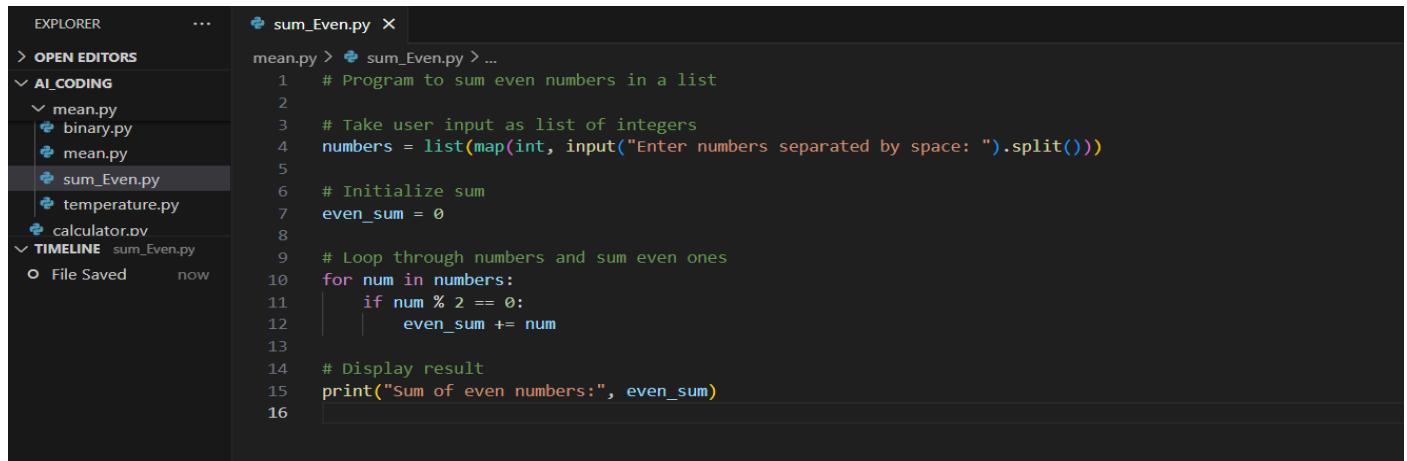
- The BankAccount class is created with account_holder and balance attributes, initialized via the constructor.
- It includes methods to deposit, withdraw, and display balance, each with proper validation.
- A menu-driven loop allows users to perform transactions interactively using input options.

TASK #2:

Prompt Used:

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 Generated:



```
EXPLORER      ...
OPEN EDITORS
AI CODING
mean.py
binary.py
mean.py
sum_Even.py
temperature.py
calculator.py
TIMELINE sum_Even.py
File Saved    now

sum_Even.py ×
mean.py > sum_Even.py > ...
1  # Program to sum even numbers in a list
2
3  # Take user input as list of integers
4  numbers = list(map(int, input("Enter numbers separated by space: ").split()))
5
6  # Initialize sum
7  even_sum = 0
8
9  # Loop through numbers and sum even ones
10 for num in numbers:
11     if num % 2 == 0:
12         even_sum += num
13
14 # Display result
15 print("Sum of even numbers:", even_sum)
16
```

Output After executing Code:

```
Enter numbers separated by space: 10 12 15 17 20
Sum of even numbers: 42
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING>
```

Observations:

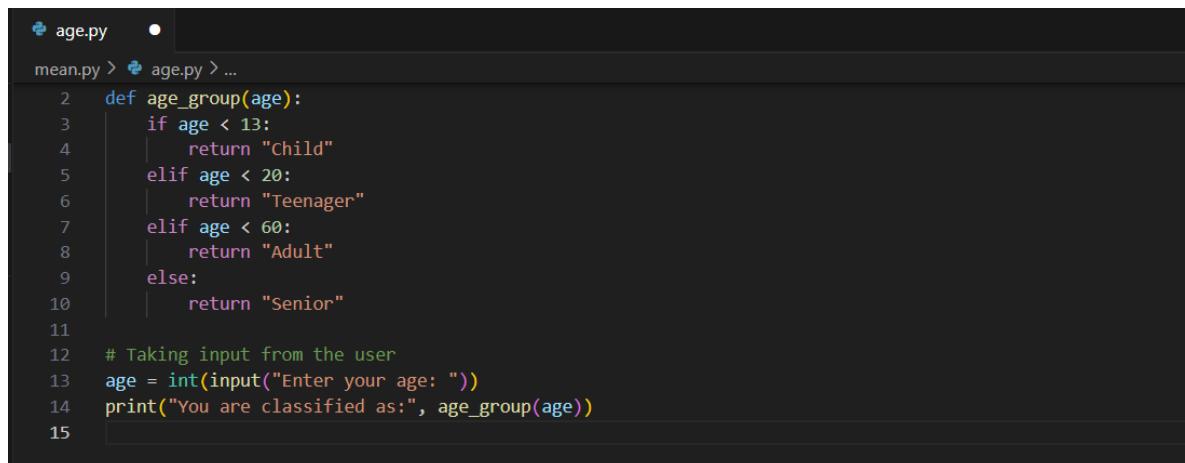
- The program accepts a list of integers from the user using `input().split()` and converts them with `map(int, ...)`.
- An accumulator variable (`even_sum`) is initialized to 0 to store the running total of even numbers.
- Each number is checked with the condition `num % 2 == 0` to determine if it is even.

TASK #3:

Prompt Used:

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 Generated:



```
age.py
mean.py > age.py > ...
1 def age_group(age):
2     if age < 13:
3         return "Child"
4     elif age < 20:
5         return "Teenager"
6     elif age < 60:
7         return "Adult"
8     else:
9         return "Senior"
10
11
12 # Taking input from the user
13 age = int(input("Enter your age: "))
14 print("You are classified as:", age_group(age))
15
```

Output After executing Code:

```
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> mean.py/age.py
Enter your age: 18
You are classified as: Teenager
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING>
```

Observations:

- The function `age_group(age)` uses if-elif-else to categorize the given age.
- Age ranges are checked in order: $< 13 \rightarrow$ Child, $< 20 \rightarrow$ Teenager, $< 60 \rightarrow$ Adult, else \rightarrow Senior.
- The result is returned as a string.

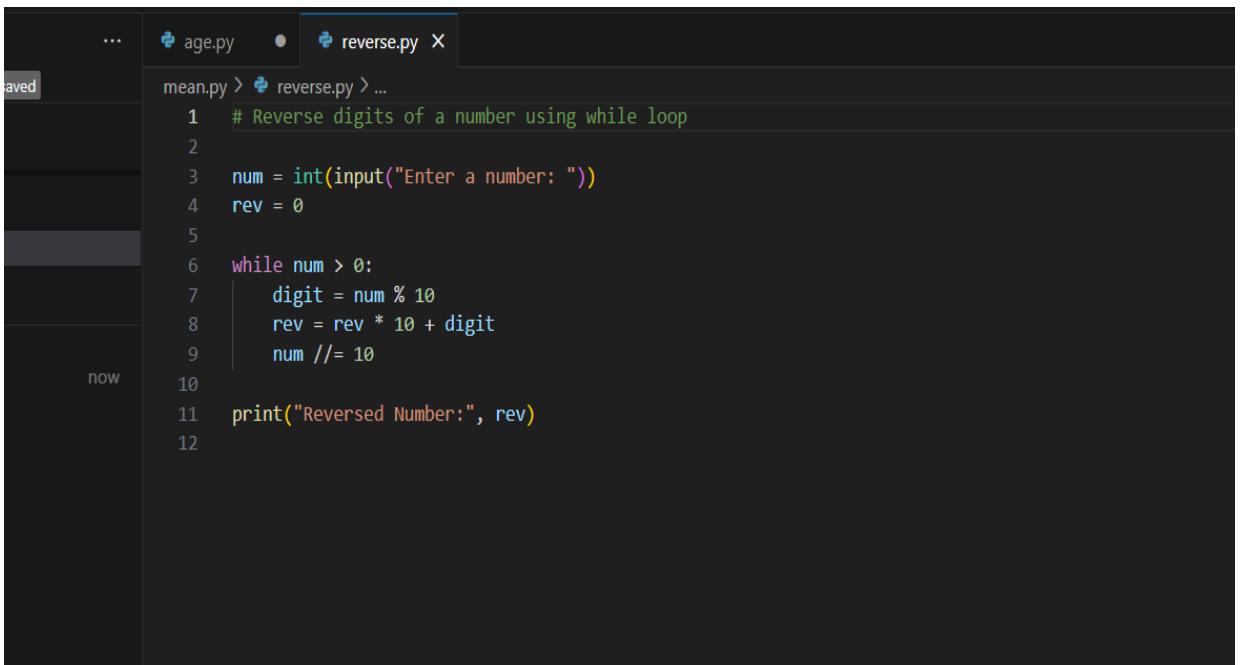
TASK #4:

Prompt Used:

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

Code Generated:

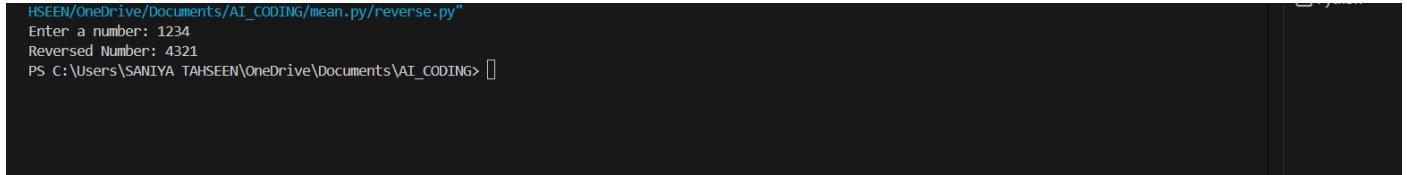
Output
After



The screenshot shows a code editor window with two tabs: 'age.py' and 'reverse.py'. The 'reverse.py' tab is active, showing the following Python code:

```
mean.py > reverse.py > ...
1  # Reverse digits of a number using while loop
2
3  num = int(input("Enter a number: "))
4  rev = 0
5
6  while num > 0:
7      digit = num % 10
8      rev = rev * 10 + digit
9      num //= 10
10
11 print("Reversed Number:", rev)
12
```

executing Code:



The screenshot shows a terminal window with the following output:

```
HSEEN/OneDrive/Documents/AI_CODING/mean.py/reverse.py"
Enter a number: 1234
Reversed Number: 4321
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> []
```

Observations:

- The last digit is obtained using `num % 10` and added to the reversed number.
- The reversed number is built step by step using `rev = rev * 10 + digit`.
- After each step, the last digit is removed from the original number using `num // 10`.

TASK #5:

Prompt Used:

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 Generated:

The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with sections like 'EXPLORER', 'OPEN EDITORS' (with 1 unsaved file), 'AI CODING', 'TIMELINE', and 'File Saved now'. The main area displays a Python script named 'employee.py'. The code defines a base class 'Employee' with an __init__ method that initializes name and salary. It then defines a derived class 'Manager' that inherits from 'Employee' and adds a 'department' attribute. A 'display' method is also defined. Finally, it takes user input for name, salary, and department, creates a Manager object, and calls its display method. The code is color-coded with syntax highlighting.

```
employee.py
mean.py > employee.py > ...
1 # Base class Employee
2 class Employee:
3     def __init__(self, name, salary):
4         # Initializing name and salary attributes
5         self.name = name
6         self.salary = salary
7
8     # Derived class Manager inheriting from Employee
9     class Manager(Employee):
10        def __init__(self, name, salary, department):
11            # Constructor chaining - call parent constructor
12            super().__init__(name, salary)
13            # Adding department attribute for Manager
14            self.department = department
15
16        # Method to display details
17        def display(self):
18            print(f"Name: {self.name}, Salary: {self.salary}, Dept: {self.department}")
19
20    # Taking user input
21    name = input("Enter employee name: ")
22    salary = int(input("Enter salary: "))
23    dept = input("Enter department: ")
24
25    # Creating Manager object
26    m = Manager(name, salary, dept)
27
28    # Displaying details
29    m.display()
```

Output After executing Code:

A terminal window is shown with the command 'python employee.py' run. The user enters 'John' for name, '50000' for salary, and 'IT' for department. The program outputs 'Name: John, Salary: 50000, Dept: IT'.

```
C:\Users\SAHIL TAHSEEN\OneDrive\Documents\AI_CODING> python employee.py
Enter employee name: John
Enter salary: 50000
Enter department: IT
Name: John, Salary: 50000, Dept: IT
PS C:\Users\SAHIL TAHSEEN\OneDrive\Documents\AI_CODING>
```

Observations:

- The Manager class inherits attributes from the Employee class and adds a new attribute department.
- Constructor chaining is achieved using super(), which allows reuse of the parent class constructor.
- Input is taken from the user to dynamically create an object with given values.