Name: Shailaja Pipalatkar **PRN:** 21070521072

Department: CSE **Division:** B

Symbiosis Institute of Technology
Department of Applied Science

॥वसुधैव कुटुम्बकम्॥

Generative AI
CA-2

Computer Science and Engineering
Batch 2021-2025

Semester - VII

Q:1 Generate a model in Python for representation of a bank account of type savings and balance along with transactions of deposit and withdrawals and currently create a program to generate 100 accounts with Random balance and transactions for no, of months and no. of transactions with a seed value of amount. Print all 100 accounts with the last balance and organize them by lowest to highest balance.

Ans:

The objective of this task was to develop a Python model that simulates a simple banking system. Specifically, we created 100 bank accounts with random initial balances and performed random transactions over a set period. After performing the transactions, the accounts were sorted based on their final balances from lowest to highest.

1. Defining a BankAccount class:

We defined a BankAccount class, which contains attributes for account ID, balance, and a transaction history list. This class encapsulates the behavior of an individual account, including deposits and withdrawals. When a new account is initialized, it starts with a randomly assigned balance.

2. Transaction Simulation:

For each account, we randomly selected a number of months (between 1 and 12) and simulated a series of transactions (deposits or withdrawals) for each month. Each transaction affected the balance, and all transactions were recorded in a list for future reference.

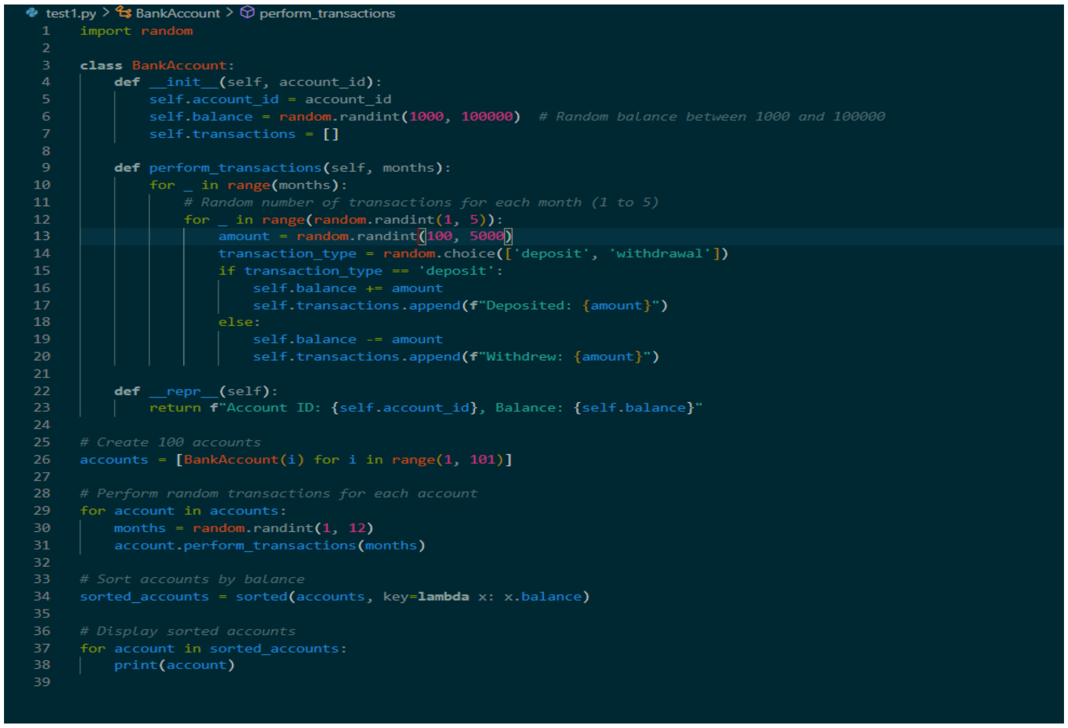
3. Account Creation:

The code created 100 accounts with unique IDs. For each account, we performed the random transactions. The number and type of transactions varied for each account, simulating real-world scenarios.

4. Sorting and Display:

After the transactions were completed, the accounts were sorted based on their final balance, from lowest to highest. This sorting was achieved using Python's sorted() function, which allowed efficient ordering of accounts based on their balance. Finally, the program printed each account's ID along with its final balance.

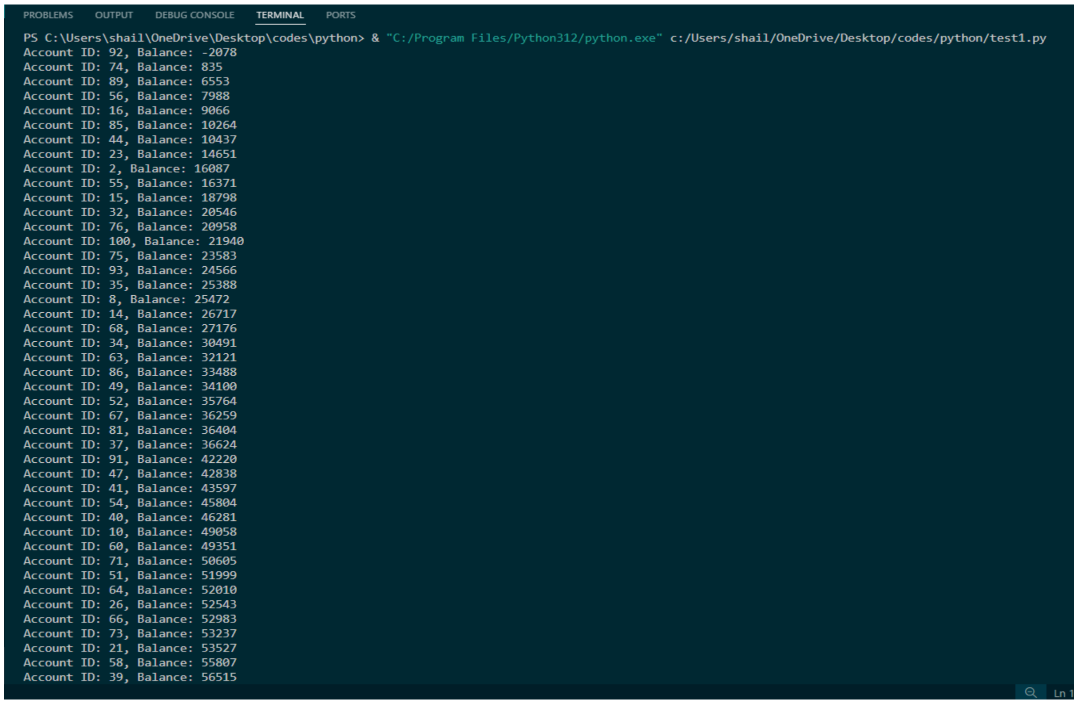
Python Code :



test1.py > BankAccount > perform_transactions

```python
import random

class BankAccount:
    def __init__(self, account_id):
        self.account_id = account_id
        self.balance = random.randint(1000, 100000)  # Random balance between 1000 and 100000
        self.transactions = []

    def perform_transactions(self, months):
        for _ in range(months):
            # Random number of transactions for each month (1 to 5)
            for _ in range(random.randint(1, 5)):
                amount = random.randint(100, 5000)
                transaction_type = random.choice(['deposit', 'withdrawal'])
                if transaction_type == 'deposit':
                    self.balance += amount
                    self.transactions.append(f"Deposited: {amount}")
                else:
                    self.balance -= amount
                    self.transactions.append(f"Withdrew: {amount}")

    def __repr__(self):
        return f"Account ID: {self.account_id}, Balance: {self.balance}"

# Create 100 accounts
accounts = [BankAccount(i) for i in range(1, 101)]

# Perform random transactions for each account
for account in accounts:
    months = random.randint(1, 12)
    account.perform_transactions(months)

# Sort accounts by balance
sorted_accounts = sorted(accounts, key=lambda x: x.balance)

# Display sorted accounts
for account in sorted_accounts:
    print(account)

```

Output:



```
PS C:\Users\shail\OneDrive\Desktop\codes\python> & "C:/Program Files/Python312/python.exe" c:/Users/shail/OneDrive/Desktop/codes/python/test1.py
Account ID: 92, Balance: -2078
Account ID: 74, Balance: 835
Account ID: 89, Balance: 6553
Account ID: 56, Balance: 7988
Account ID: 16, Balance: 9066
Account ID: 85, Balance: 10264
Account ID: 44, Balance: 10437
Account ID: 23, Balance: 14651
Account ID: 2, Balance: 16087
Account ID: 55, Balance: 16371
Account ID: 15, Balance: 18798
Account ID: 32, Balance: 20546
Account ID: 76, Balance: 20958
Account ID: 100, Balance: 21940
Account ID: 75, Balance: 23583
Account ID: 93, Balance: 24566
Account ID: 35, Balance: 25388
Account ID: 8, Balance: 25472
Account ID: 14, Balance: 26717
Account ID: 68, Balance: 27176
Account ID: 34, Balance: 30491
Account ID: 63, Balance: 32121
Account ID: 86, Balance: 33488
Account ID: 49, Balance: 34100
Account ID: 52, Balance: 35764
Account ID: 67, Balance: 36259
Account ID: 81, Balance: 36404
Account ID: 37, Balance: 36624
Account ID: 91, Balance: 42220
Account ID: 47, Balance: 42838
Account ID: 41, Balance: 43597
Account ID: 54, Balance: 45804
Account ID: 40, Balance: 46281
Account ID: 10, Balance: 49058
Account ID: 60, Balance: 49351
Account ID: 71, Balance: 50605
Account ID: 51, Balance: 51999
Account ID: 64, Balance: 52010
Account ID: 26, Balance: 52543
Account ID: 66, Balance: 52983
Account ID: 73, Balance: 53237
Account ID: 21, Balance: 53527
Account ID: 58, Balance: 55807
Account ID: 39, Balance: 56515
```

Q:6 Q Generate a model to represent a mathematical equation, write a program to parse the equation, and ask for input for each parameter.

Ans:

The goal of this task was to create a Python program that can parse and evaluate a user-defined mathematical equation containing variables. The program takes user input for the equation and substitutes the variable with a value provided by the user. It then calculates and displays the result.

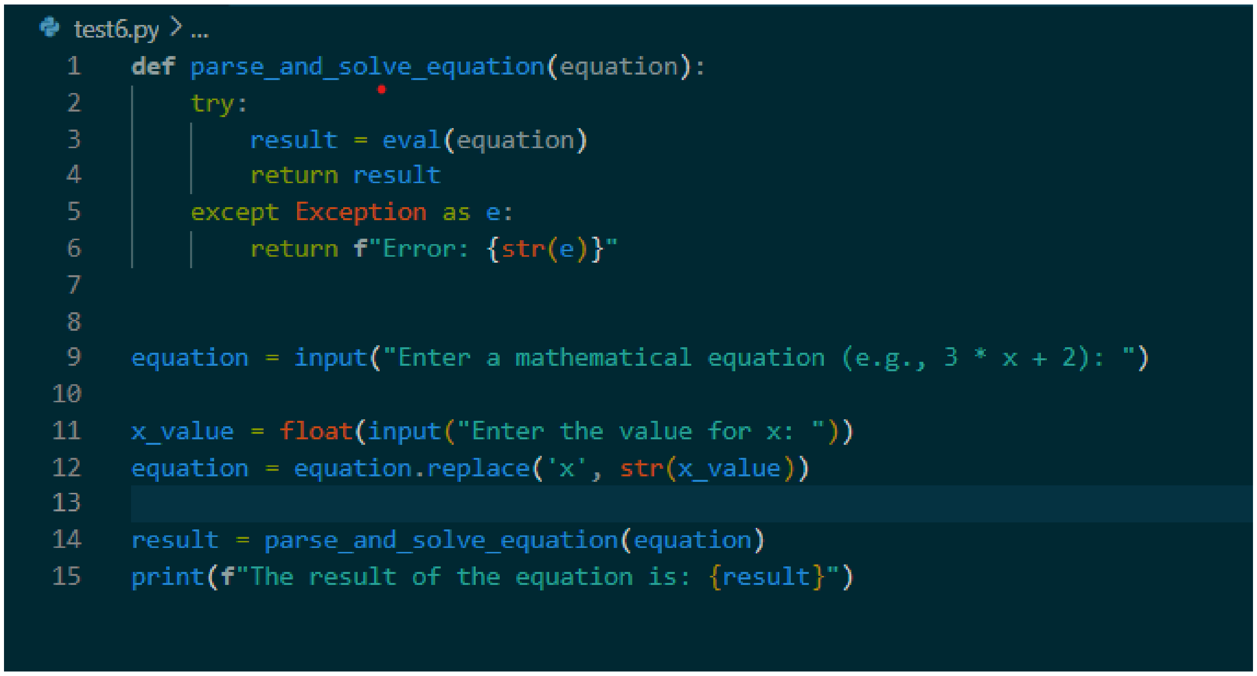
1. Equation Input:
The program begins by prompting the user to enter a mathematical equation. This equation can include mathematical operations and a variable, typically represented as x. The program stores the equation as a string, allowing dynamic handling of mathematical expressions.

2. Variable Substitution:
The next step involves asking the user for the value of the variable x. Once the user inputs this value, the program replaces the variable in the equation with the provided number. This substitution ensures that the equation is now fully numeric and ready for evaluation.

3. Equation Evaluation:
To evaluate the equation, the program uses Python's built-in eval() function. This function takes a string representing a mathematical expression and computes its value. For instance, if the input equation is 3 * x + 2 and the user provides x = 4, the program replaces x with 4 and computes the result as 14.

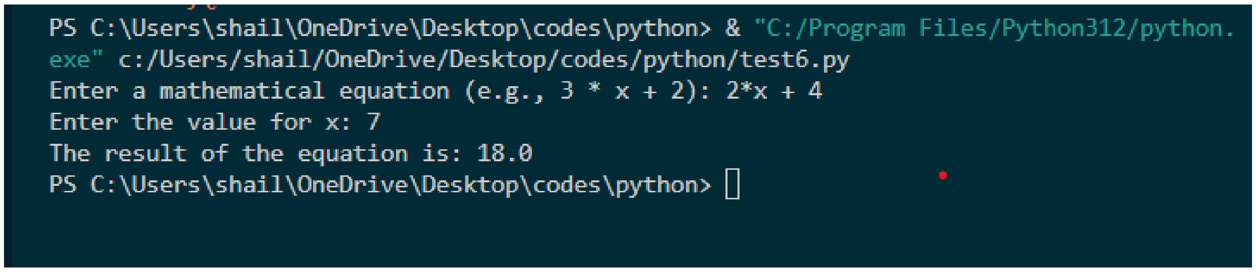
4. Error Handling:
The program includes basic error handling to catch any potential issues during evaluation, such as syntax errors or invalid input. If the equation cannot be evaluated, the program returns an appropriate error message to the user.

Python Code:



```
test6.py > ...
def parse_and_solve_equation(equation):
    try:
        result = eval(equation)
        return result
    except Exception as e:
        return f"Error: {str(e)}"


equation = input("Enter a mathematical equation (e.g., 3 * x + 2): ")

x_value = float(input("Enter the value for x: "))
equation = equation.replace('x', str(x_value))

result = parse_and_solve_equation(equation)
print(f"The result of the equation is: {result}")
```

Output:



```
PS C:\Users\shail\OneDrive\Desktop\codes\python> & "C:/Program Files/Python312/python.
exe" c:/Users/shail/OneDrive/Desktop/codes/python/test6.py
Enter a mathematical equation (e.g., 3 * x + 2): 2*x + 4
Enter the value for x: 7
The result of the equation is: 18.0
PS C:\Users\shail\OneDrive\Desktop\codes\python> 
```