EEI3372 - Programing with Python
Mini Project

Name : S.D.DN.Sudasingha

S number : s23010374

Registration number : 623599072

Group Number : G05

Development of a Basic Bank Account Management System in Python

Introduction:

In this report, we will discuss the creation of a command-line application in Python that simulates a basic bank account management system. The application allows users to perform various actions such as creating new accounts, depositing, and withdrawing money, checking account balances, and transferring funds between accounts. The system ensures basic validation for deposit and withdrawal amounts, verifies account existence, and checks for sufficient balance before processing transactions. The account information is stored in memory without the need for persistent storage.

The development process involves using classes, data structures, and functions to handle user input and manage the accounts. The text-based menu provides an interactive platform for users to engage with the system. This report aims to provide insights into the design and implementation of this Python-based bank account management system.

Assumptions:

1. Each account will have a unique account number.
2. When creating a new account, users must provide a positive initial deposit amount.
3. Deposit and withdrawal amounts should be non-negative.
4. The system will ensure that accounts have sufficient balance before allowing withdrawals or transfers.
5. The system will not allow overdrafts. Users can only withdraw up to the available balance.
6. Account numbers will be alphanumeric and case-sensitive.
7. The system will provide basic error messages for invalid inputs and actions, but will not extensively handle all edge cases.

Challenges Faced and Solutions:

1.Challenge: Generating unique account numbers sequentially.

- **Solution:** Implemented a function to generate account numbers using a counter, ensuring uniqueness by incrementing the counter for each new account.

2.Challenge: Ensuring that users provide a positive initial deposit amount.

- **Solution:** Implemented validation checks to reject negative initial deposit amounts.

3.Challenge: Validating deposit and withdrawal amounts to ensure they are non-negative.

- **Solution:** Implemented checks in deposit and withdrawal functions to reject negative amounts.

4.Challenge: Checking if an account has sufficient balance before allowing withdrawals or transfers.

- **Solution:** Implemented checks in withdrawal and transfer functions to verify sufficient balance.

5.Challenge: Creating an intuitive menu and handling user input.

- **Solution:** Designed a simple text-based menu and used input statements to interact with the user.

6.Challenge: Providing meaningful error messages for invalid inputs and actions.

- **Solution:** Incorporated error messages for invalid inputs and actions, guiding users on correct usage.

7.Challenge: Storing account information in memory without using persistent storage.

- **Solution:** Utilized dictionaries to store accounts, with account numbers as keys and account details as values.

Algorithms

1. Create a new account with a unique account number and initial deposit amount:
    - Check if the account_number already exists in the accounts dictionary.
    - If the account_number already exists, print an error message and return.
    - Create a new account object with the account_number and initial_deposit as initial balance.
    - Add the new account to the accounts dictionary.
    - Print a success message confirming the creation of the new account.

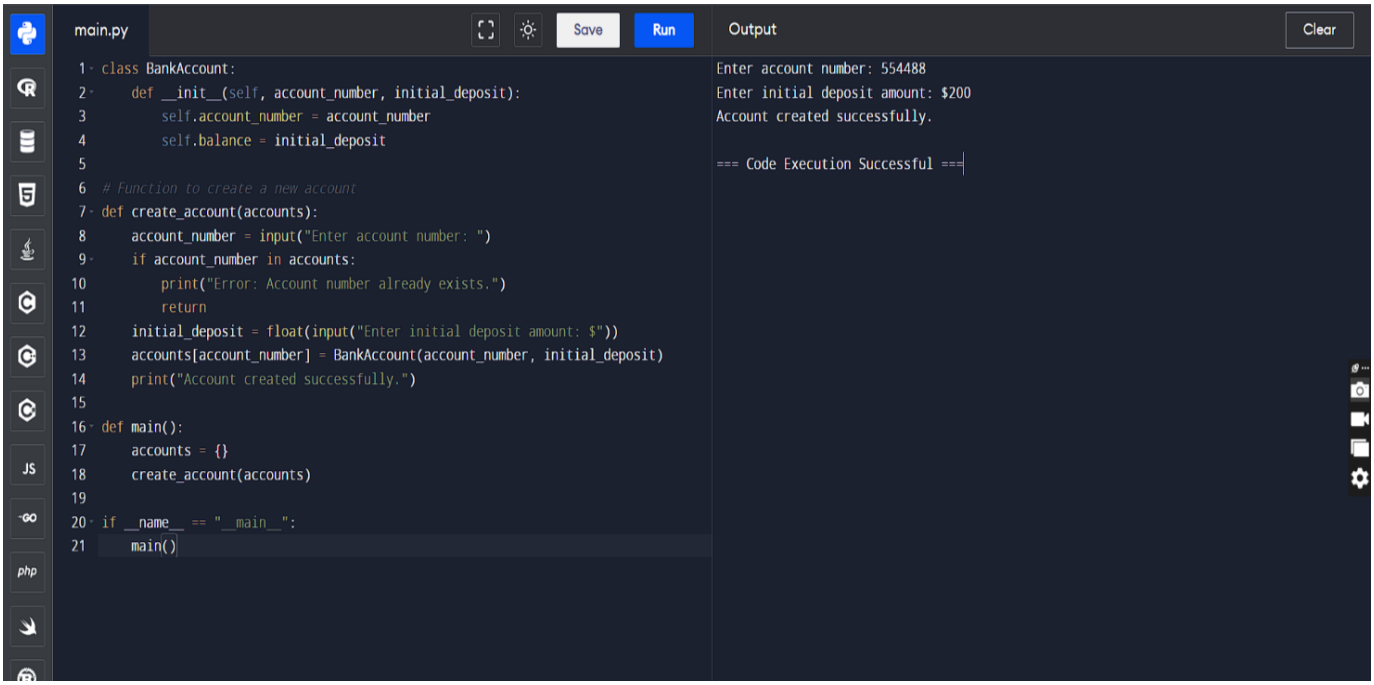
2. Deposit money into an account using the account number:
    - Check if the account_number exists in the accounts dictionary.
    - If the account_number does not exist, print an error message and return.
    - Check if the deposit_amount is greater than 0.
    - If deposit_amount is not greater than 0, print an error message and return.
    - Add the deposit_amount to the balance attribute of the account associated with the account_number.

3. Withdraw money from an account using the account number:
    - Check if the account_number exists in the accounts dictionary.
    - If the account_number does not exist, print an error message and return.
    - Check if the withdrawal_amount is greater than 0 and less than or equal to the available balance.
    - If withdrawal_amount is not greater than 0 or exceeds the available balance, print an error message and return.
    - Subtract the withdrawal_amount from the balance attribute of the account associated with the account_number.

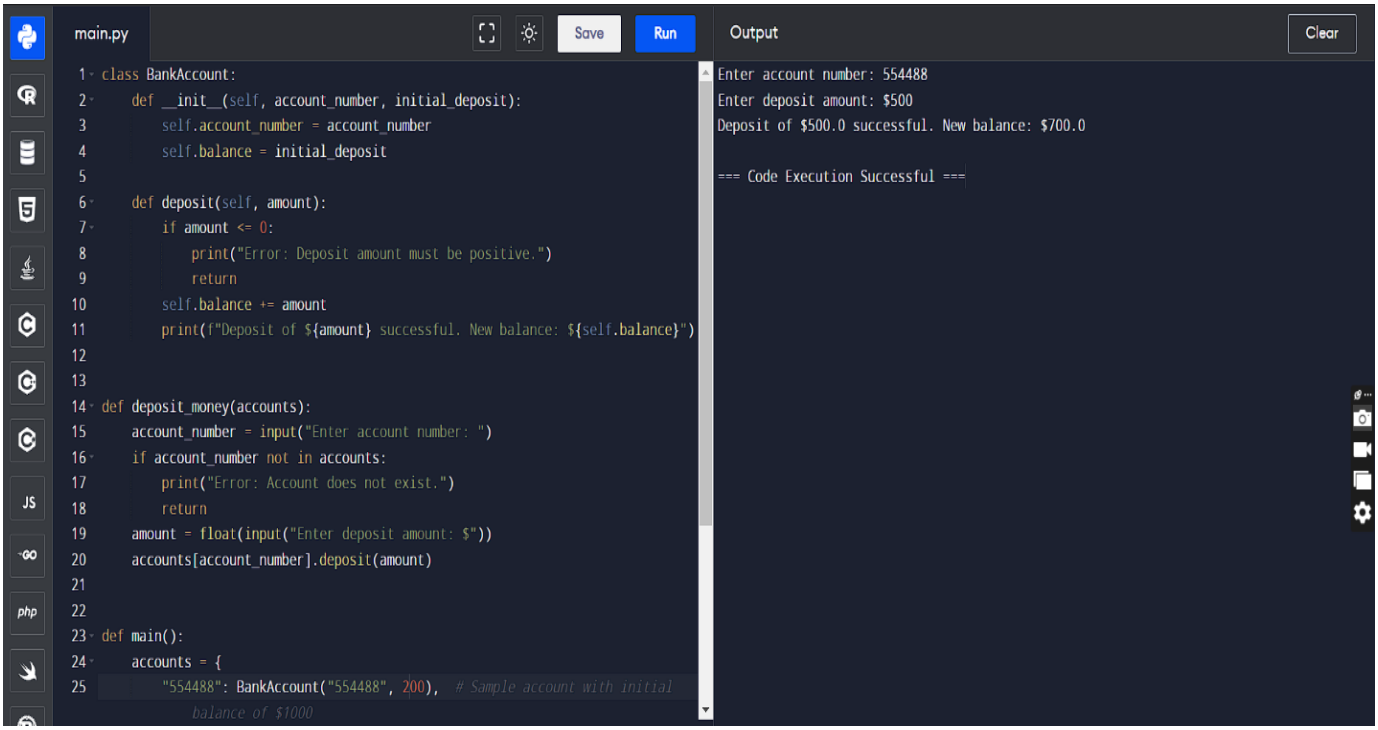
4. Check the account balance using the account number:
    - Check if the account_number exists in the accounts dictionary.
    - If the account_number does not exist, print an error message and return.
    - Retrieve the account associated with the account_number from the accounts dictionary.
    - Print the current balance of the account.

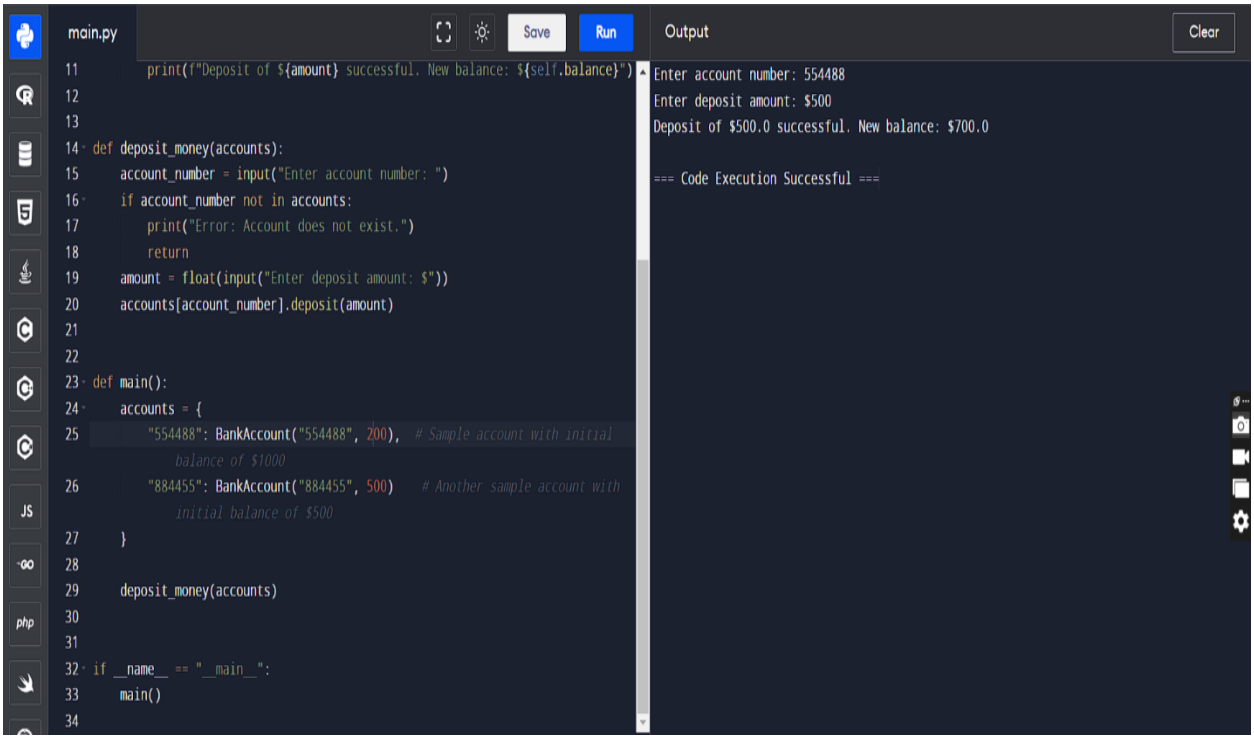
5. Transfer money between two accounts using their account numbers:
    - Check if both source_account_number and destination_account_number exist in the accounts dictionary.
    - If either account number does not exist, print an error message and return.
    - Check if the transfer_amount is greater than 0 and less than or equal to the available balance of the source account.
    - If transfer_amount is not greater than 0 or exceeds the available balance of the source account, print an error message and return.
    - Subtract the transfer_amount from the balance attribute of the source account.
    - Add the transfer_amount to the balance attribute of the destination account.

1. Create a new account with a unique account number and initial deposit amount.

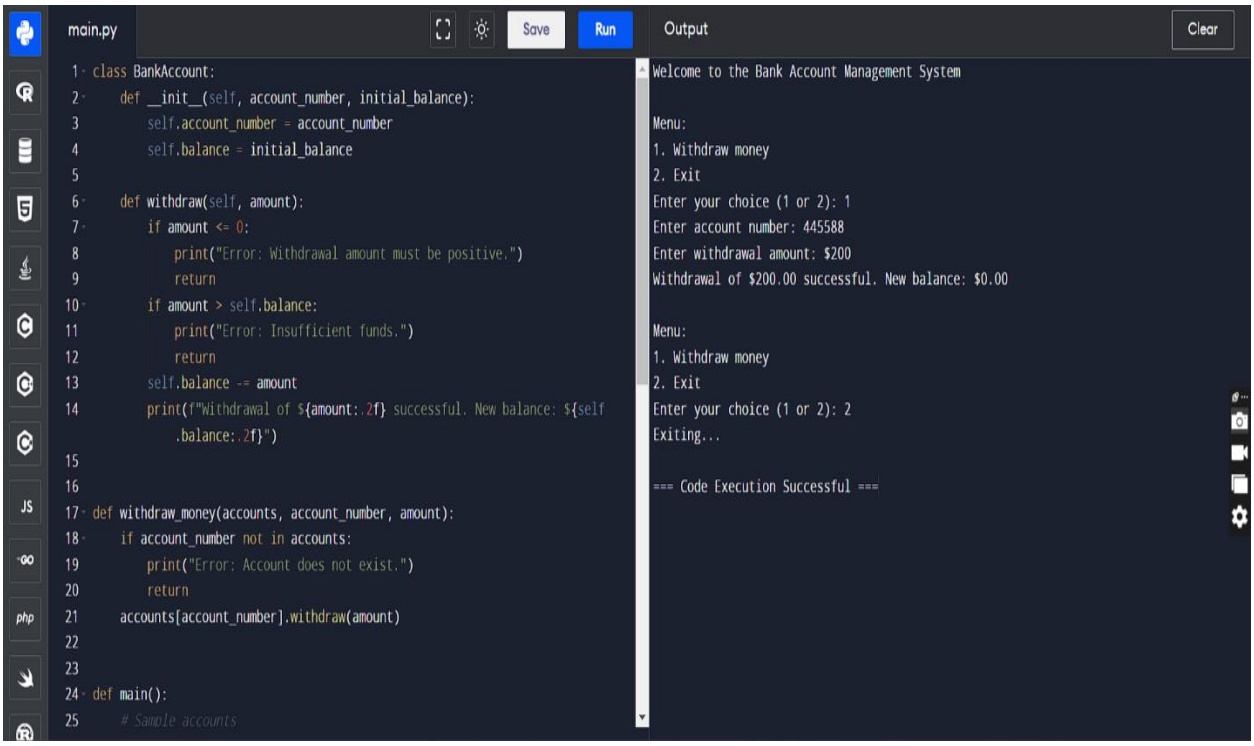


2. Deposit money into an account using the account number.





3. Withdraw money from an account using the account number.



main.py

Save Run Output Clear

```
def main():
    # Sample accounts
    accounts = {
        "445588": BankAccount("123456", 200.00),
        "884455": BankAccount("654321", 500.00)
    }

    print("Welcome to the Bank Account Management System")

    while True:
        print("\nMenu:")
        print("1. Withdraw money")
        print("2. Exit")
        choice = input("Enter your choice (1 or 2): ")

        if choice == "1":
            account_number = input("Enter account number: ")
            amount = float(input("Enter withdrawal amount: $"))
            withdraw_money(accounts, account_number, amount)
        elif choice == "2":
            print("Exiting...")
            break
        else:
            print("Invalid choice. Please enter 1 or 2.")
```

```
Welcome to the Bank Account Management System

Menu:
1. Withdraw money
2. Exit
Enter your choice (1 or 2): 1
Enter account number: 445588
Enter withdrawal amount: $200
Withdrawal of $200.00 successful. New balance: $0.00

Menu:
1. Withdraw money
2. Exit
Enter your choice (1 or 2): 2
Exiting...

=== Code Execution Successful ===
```

4. Check the account balance using the account number.

main.py

Save Run Output Clear

```
class BankAccount:
    def __init__(self, account_number, initial_balance):
        self.account_number = account_number
        self.balance = initial_balance

    def check_balance(self):
        print(f"Account balance for {self.account_number}: ${self.balance:.2f}"
            )


def check_balance(accounts, account_number):
    if account_number not in accounts:
        print("Error: Account does not exist.")
        return
    accounts[account_number].check_balance()


def main():
    # Sample accounts
    accounts = {
        "445588": BankAccount("445588", 2000.00),
        "885544": BankAccount("885544", 500.00)
    }

    print("Welcome to the Bank Account Management System")

```

```
Welcome to the Bank Account Management System

Menu:
1. Check account balance
2. Exit
Enter your choice (1 or 2): 1
Enter account number: 445588
Account balance for 445588: $2000.00

Menu:
1. Check account balance
2. Exit
Enter your choice (1 or 2): 2
Exiting...

=== Code Execution Successful ===
```

main.py

Save Run

```
    accounts = {
        "445588": BankAccount("445588", 2000.00),
        "885544": BankAccount("885544", 500.00)
    }

    print("Welcome to the Bank Account Management System")

    while True:
        print("\nMenu:")
        print("1. Check account balance")
        print("2. Exit")
        choice = input("Enter your choice (1 or 2): ")

        if choice == "1":
            account_number = input("Enter account number: ")
            check_balance(accounts, account_number)
        elif choice == "2":
            print("Exiting...")
            break
        else:
            print("Invalid choice. Please enter 1 or 2.")


if __name__ == "__main__":
    main()

```

Output

Clear

```
Welcome to the Bank Account Management System

Menu:
1. Check account balance
2. Exit
Enter your choice (1 or 2): 1
Enter account number: 445588
Account balance for 445588: $2000.00

Menu:
1. Check account balance
2. Exit
Enter your choice (1 or 2): 2
Exiting...

=== Code Execution Successful ===
```

5. Transfer money between two accounts using their account numbers.

main.py

Save Run

```
class BankAccount:
    def __init__(self, account_number, initial_balance):
        self.account_number = account_number
        self.balance = initial_balance

    def deposit(self, amount):
        if amount <= 0:
            print("Error: Deposit amount must be positive.")
            return
        self.balance += amount
        print(f"Deposit of ${amount:.2f} successful. New balance: ${self
            .balance:.2f}")

    def withdraw(self, amount):
        if amount <= 0:
            print("Error: Withdrawal amount must be positive.")
            return
        if amount > self.balance:
            print("Error: Insufficient funds.")
            return
        self.balance -= amount
        print(f"Withdrawal of ${amount:.2f} successful. New balance: ${self
            .balance:.2f}")

    def transfer(self, recipient_account, amount):
        if amount <= 0:
```

Output

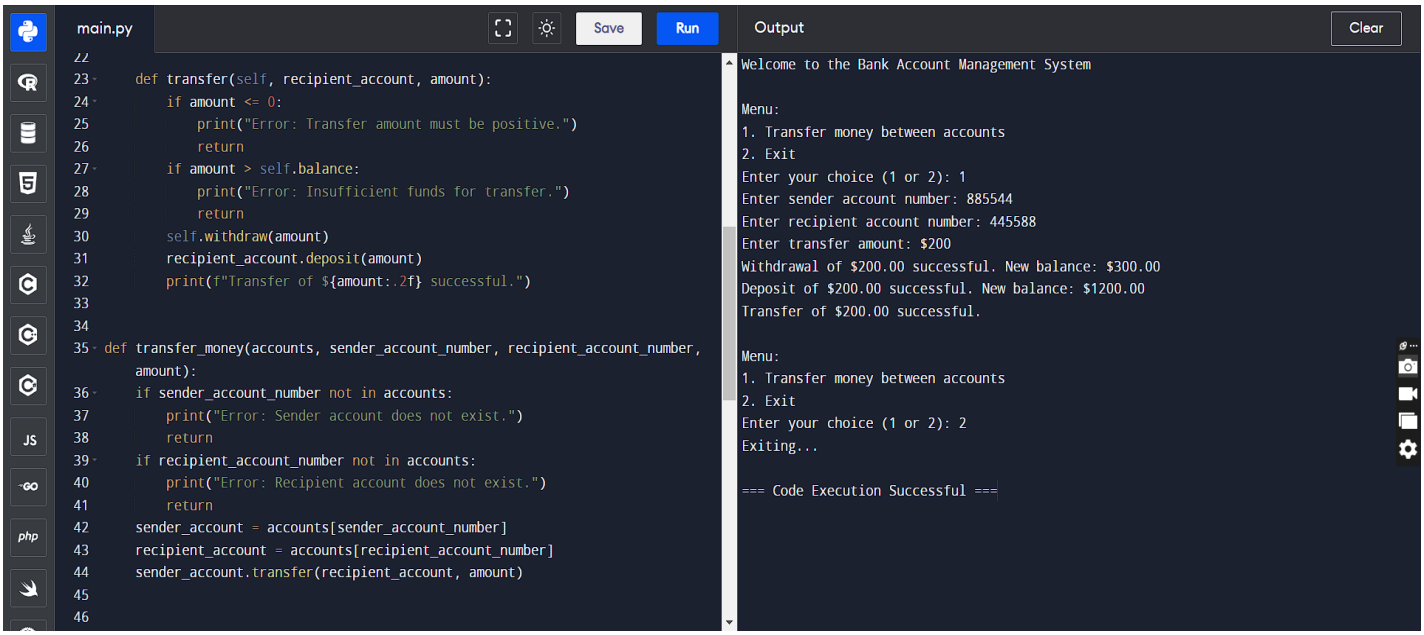
Clear

```
Welcome to the Bank Account Management System

Menu:
1. Transfer money between accounts
2. Exit
Enter your choice (1 or 2): 1
Enter sender account number: 885544
Enter recipient account number: 445588
Enter transfer amount: $200
Withdrawal of $200.00 successful. New balance: $300.00
Deposit of $200.00 successful. New balance: $1200.00
Transfer of $200.00 successful.

Menu:
1. Transfer money between accounts
2. Exit
Enter your choice (1 or 2): 2
Exiting...

=== Code Execution Successful ===
```


main.py
Save
Run
Output
Clear
def transfer(self, recipient_account, amount):
if amount <= 0:
print("Error: Transfer amount must be positive.")
return
if amount > self.balance:
print("Error: Insufficient funds for transfer.")
return
self.withdraw(amount)
recipient_account.deposit(amount)
print(f"Transfer of ${amount:.2f} successful.")
def transfer_money(accounts, sender_account_number, recipient_account_number, amount):
if sender_account_number not in accounts:
print("Error: Sender account does not exist.")
return
if recipient_account_number not in accounts:
print("Error: Recipient account does not exist.")
return
sender_account = accounts[sender_account_number]
recipient_account = accounts[recipient_account_number]
sender_account.transfer(recipient_account, amount)
Welcome to the Bank Account Management System
Menu:
1. Transfer money between accounts
2. Exit
Enter your choice (1 or 2): 1
Enter sender account number: 885544
Enter recipient account number: 445588
Enter transfer amount: $200
Withdrawal of $200.00 successful. New balance: $300.00
Deposit of $200.00 successful. New balance: $1200.00
Transfer of $200.00 successful.
Menu:
1. Transfer money between accounts
2. Exit
Enter your choice (1 or 2): 2
Exiting...
=== Code Execution Successful ===