python™

BANK ACCOUNT MANAGEMENT SYSTEM

INTRODUCTION

- **This project is a Bank Account Management System built using Python.**
- **It demonstrates how to create, manage, and operate multiple bank accounts.**

- **Features include:**

1. Opening new accounts
2. Deposits & withdrawals
3. Fund transfers
4. Transaction history
5. Account summary with statistics
6. Data persistence and report generation

LIBRARIES

Python Libraries & Their Purpose

Library	Purpose
pickle	Save & load account data (persistence)
random	Generate unique account numbers
os	Check file existence for data storage
numpy	Perform summary statistics (sum, mean)
datetime	Record timestamps for transactions

In [4]:

```python
import pickle
import random
import os
import numpy as np
from datetime import datetime

# Global variables
ACCOUNTS_FILE = "bank_data.pkl"

class BankAccount:
    def __init__(self, account_holder, account_type, initial_balance=0, password
        self.account_holder = account_holder
        self.account_number = self.generate_account_number()
        self.account_type = account_type
        self.balance = initial_balance
        self.password = password
        self.transactions = []
        self.creation_date = datetime.now()

    def generate_account_number(self):
        return f"{random.randint(10000000, 99999999)}"

    def deposit(self, amount):
        if amount <= 0:
            return False, "Deposit amount must be positive"

        self.balance += amount
        transaction = {
            'date': datetime.now(),
            'type': 'deposit',
            'amount': amount,
            'balance_after': self.balance
        }
        self.transactions.append(transaction)
        return True, f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}

    def withdraw(self, amount):
        if amount <= 0:
            return False, "Withdrawal amount must be positive"

        if amount > self.balance:
            return False, "Insufficient funds"

        self.balance -= amount
        transaction = {
            'date': datetime.now(),
            'type': 'withdrawal',
            'amount': amount,
            'balance_after': self.balance
        }
        self.transactions.append(transaction)
        return True, f"Withdrew ${amount:.2f}. New balance: ${self.balance:.2f}"

    def transfer(self, amount, target_account):
        """Transfer without double-logging. Only create transfer_in/out entries.
        if amount <= 0:
            return False, "Transfer amount must be positive"
        if amount > self.balance:
```

```
            return False, "Insufficient funds"

        # Adjust balances
        self.balance -= amount
        target_account.balance += amount

        # Log transfer out (source)
        transfer_out = {
            'date': datetime.now(),
            'type': 'transfer_out',
            'amount': amount,
            'target_account': target_account.account_number,
            'balance_after': self.balance
        }
        self.transactions.append(transfer_out)

        # Log transfer in (target)
        transfer_in = {
            'date': datetime.now(),
            'type': 'transfer_in',
            'amount': amount,
            'source_account': self.account_number,
            'balance_after': target_account.balance
        }
        target_account.transactions.append(transfer_in)

        return True, f"Transferred ${amount:.2f} to account {target_account.accc

    def get_transaction_history(self):
        return self.transactions

    def get_summary_statistics(self):
        if not self.transactions:
            return {
                'total_deposits': 0,
                'total_withdrawals': 0,
                'average_transaction': 0,
                'transaction_count': 0
            }

        amounts = [t['amount'] for t in self.transactions]
        # deposits = explicit deposits + transfer_in
        deposits = [t['amount'] for t in self.transactions if t['type'] in ['dep
        # withdrawals = explicit withdrawals + transfer_out
        withdrawals = [t['amount'] for t in self.transactions if t['type'] in ['

        return {
            'total_deposits': float(np.sum(deposits)) if deposits else 0.0,
            'total_withdrawals': float(np.sum(withdrawals)) if withdrawals else
            'average_transaction': float(np.mean(amounts)) if amounts else 0.0,
            'transaction_count': len(self.transactions)
        }

    def to_dict(self):
        return {
            'account_holder': self.account_holder,
            'account_number': self.account_number,
            'account_type': self.account_type,
            'balance': self.balance,
            'password': self.password,
```

```
            'transactions': self.transactions,
            'creation_date': self.creation_date
        }

    @classmethod
    def from_dict(cls, data):
        account = cls(
            data.get('account_holder', 'Unknown'),
            data.get('account_type', 'savings'),
            data.get('balance', 0),
            data.get('password', None)
        )
        account.account_number = data.get('account_number', account.generate_acc
        account.transactions = data.get('transactions', [])
        account.creation_date = data.get('creation_date', datetime.now())
        return account

class BankSystem:
    def __init__(self):
        self.accounts = self.load_data()

    def load_data(self):
        if os.path.exists(ACCOUNTS_FILE):
            try:
                with open(ACCOUNTS_FILE, 'rb') as f:
                    accounts_data = pickle.load(f)
                    if isinstance(accounts_data, dict):
                        return {acc_num: BankAccount.from_dict(acc_data) for acc
                    else:
                        # Backward compatibility if a list of BankAccount object
                        return {acc.account_number: acc for acc in accounts_data
            except (EOFError, pickle.UnpicklingError) as e:
                print(f"Error loading data: {e}. Starting with empty accounts.")
                return {}
        return {}

    def save_data(self):
        accounts_data = {acc_num: account.to_dict() for acc_num, account in self
        with open(ACCOUNTS_FILE, 'wb') as f:
            pickle.dump(accounts_data, f)

    def create_account(self, account_holder, account_type, initial_balance=0, pa
        account = BankAccount(account_holder, account_type, initial_balance, pas
        self.accounts[account.account_number] = account
        self.save_data()
        return account

    def get_account(self, account_number):
        return self.accounts.get(account_number)

    def authenticate_account(self, account_number, password):
        account = self.get_account(account_number)
        if account and account.password == password:
            return account
        return None

    def transfer_funds(self, source_account_number, target_account_number, amoun
        source_account = self.authenticate_account(source_account_number, passwo
        if not source_account:
            return False, "Authentication failed or account not found"
```

```
        target_account = self.get_account(target_account_number)
        if not target_account:
            return False, "Target account not found"

        success, message = source_account.transfer(amount, target_account)
        if success:
            self.save_data()
        return success, message

def display_menu():
    print("="*50)
    print("BANK ACCOUNT MANAGEMENT SYSTEM".center(50))
    print("="*50)
    print("1. Open New Account")
    print("2. View Account Details")
    print("3. Deposit")
    print("4. Withdraw")
    print("5. Transfer")
    print("6. View Transaction History")
    print("7. Account Summary")
    print("8. Exit")
    print("="*50)
    return input("Please select an option (1-8): ")

def save_to_readable_format(bank_system):
    with open("bank_accounts_report.txt", "w") as f:
        f.write("BANK ACCOUNTS REPORT\n")
        f.write("=" * 50 + "\n")
        f.write(f"Generated on: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n
        f.write("=" * 50 + "\n\n")

        for account_number, account in bank_system.accounts.items():
            f.write(f"Account Holder: {account.account_holder}\n")
            f.write(f"Account Number: {account.account_number}\n")
            f.write(f"Account Type: {account.account_type}\n")
            f.write(f"Balance: ${account.balance:.2f}\n")
            f.write(f"Created: {account.creation_date.strftime('%Y-%m-%d %H:%M:%

            # Transaction history
            f.write("\nTransaction History:\n")
            f.write("-" * 50 + "\n")
            if account.transactions:
                for i, transaction in enumerate(account.transactions, 1):
                    date_obj = transaction['date']
                    # Ensure we can handle either datetime or string (defensive)
                    if isinstance(date_obj, datetime):
                        date_str = date_obj.strftime('%Y-%m-%d %H:%M')
                    else:
                        date_str = str(date_obj)
                    amt = transaction['amount']
                    bal = transaction.get('balance_after', account.balance)
                    ttype = transaction['type']
                    f.write(f"{i}. {date_str} - {ttype}: ${amt:.2f} - Balance: $
            else:
                f.write("No transactions yet.\n")

            # Account statistics
            stats = account.get_summary_statistics()
            f.write("\nAccount Statistics:\n")
```

```
            f.write("-" * 50 + "\n")
            f.write(f"Total Deposits: ${stats['total_deposits']:.2f}\n")
            f.write(f"Total Withdrawals: ${stats['total_withdrawals']:.2f}\n")
            f.write(f"Average Transaction: ${stats['average_transaction']:.2f}\n
            f.write(f"Transaction Count: {stats['transaction_count']}\n")

            f.write("\n" + "=" * 50 + "\n\n")

        f.write("Thank you for trusting our bank!\n")

def main():
    bank_system = BankSystem()

    while True:
        choice = display_menu()

        if choice == '1':
            print("\n--- Open New Account ---")
            name = input("Enter account holder's name: ").strip()
            account_type = input("Enter account type (savings/current): ").strip
            if account_type not in {"savings", "current"}:
                print("Invalid account type. Please enter 'savings' or 'current'
                continue

            try:
                initial_deposit = float(input("Enter initial deposit amount: "))
                if initial_deposit < 0:
                    print("Initial deposit cannot be negative.")
                    continue
            except ValueError:
                print("Invalid amount. Please enter a numeric value.")
                continue

            password = input("Set your account password: ")
            confirm_password = input("Confirm your password: ")

            if password != confirm_password:
                print("Passwords do not match. Please try again.")
                continue

            account = bank_system.create_account(name, account_type, initial_dep
            print(f"\nAccount created successfully!")
            print(f"Your account number is: {account.account_number}")
            print("Please remember your account number and password for future l

        elif choice == '2':
            print("\n--- View Account Details ---")
            account_number = input("Enter your account number: ")
            password = input("Enter your password: ")

            account = bank_system.authenticate_account(account_number, password)
            if account:
                print(f"\nAccount Holder: {account.account_holder}")
                print(f"Account Number: {account.account_number}")
                print(f"Account Type: {account.account_type}")
                print(f"Current Balance: ${account.balance:.2f}")
                print(f"Account Created: {account.creation_date.strftime('%Y-%m-
            else:
                print("Invalid account number or password.")
```

```
        elif choice == '3':
            print("\n--- Deposit ---")
            account_number = input("Enter your account number: ")
            password = input("Enter your password: ")

            account = bank_system.authenticate_account(account_number, password)
            if account:
                try:
                    amount = float(input("Enter amount to deposit: "))
                    success, message = account.deposit(amount)
                    print(message)
                    if success:
                        bank_system.save_data()
                except ValueError:
                    print("Invalid amount. Please enter a numeric value.")
            else:
                print("Invalid account number or password.")

        elif choice == '4':
            print("\n--- Withdraw ---")
            account_number = input("Enter your account number: ")
            password = input("Enter your password: ")

            account = bank_system.authenticate_account(account_number, password)
            if account:
                try:
                    amount = float(input("Enter amount to withdraw: "))
                    success, message = account.withdraw(amount)
                    print(message)
                    if success:
                        bank_system.save_data()
                except ValueError:
                    print("Invalid amount. Please enter a numeric value.")
            else:
                print("Invalid account number or password.")

        elif choice == '5':
            print("\n--- Transfer ---")
            account_number = input("Enter your account number: ")
            password = input("Enter your password: ")

            account = bank_system.authenticate_account(account_number, password)
            if account:
                try:
                    target_account_number = input("Enter target account number:
                    amount = float(input("Enter amount to transfer: "))

                    success, message = bank_system.transfer_funds(
                        account_number,
                        target_account_number,
                        amount,
                        password
                    )
                    print(message)
                except ValueError:
                    print("Invalid amount. Please enter a numeric value.")
            else:
                print("Invalid account number or password.")

        elif choice == '6':
```

```
            print("\n--- View Transaction History ---")
            account_number = input("Enter your account number: ")
            password = input("Enter your password: ")

            account = bank_system.authenticate_account(account_number, password)
            if account:
                transactions = account.get_transaction_history()
                if not transactions:
                    print("No transactions found.")
                else:
                    print("\nTransaction History:")
                    print("-" * 70)
                    for i, transaction in enumerate(transactions, 1):
                        date_obj = transaction['date']
                        date_str = date_obj.strftime('%Y-%m-%d %H:%M') if isinst
                        ttype = transaction['type']
                        amt = transaction['amount']
                        bal = transaction.get('balance_after', account.balance)
                        print(f"{i}. {date_str} - {ttype}: ${amt:.2f} - Balance:
            else:
                print("Invalid account number or password.")

        elif choice == '7':
            print("\n--- Account Summary ---")
            account_number = input("Enter your account number: ")
            password = input("Enter your password: ")

            account = bank_system.authenticate_account(account_number, password)
            if account:
                stats = account.get_summary_statistics()
                print(f"\nTotal Deposits: ${stats['total_deposits']:.2f}")
                print(f"Total Withdrawals: ${stats['total_withdrawals']:.2f}")
                print(f"Average Transaction Amount: ${stats['average_transaction
                print(f"Total Transactions: {stats['transaction_count']}")
            else:
                print("Invalid account number or password.")

        elif choice == '8':
            print("\nSaving account data and generating report...")
            bank_system.save_data()
            save_to_readable_format(bank_system)
            print("Report saved to 'bank_accounts_report.txt'")
            print("\nThank you for using our banking system. Goodbye!")
            print("\nThank you for trusting our bank!")
            break

        else:
            print("Invalid option. Please try again.")

if __name__ == "__main__":
    main()
```

```
==================================================
          BANK ACCOUNT MANAGEMENT SYSTEM
==================================================
1. Open New Account
2. View Account Details
3. Deposit
4. Withdraw
5. Transfer
6. View Transaction History
7. Account Summary
8. Exit
==================================================
--- Open New Account ---
Account created successfully!
Your account number is: 60833367
Please remember your account number and password for future logins.
==================================================
          BANK ACCOUNT MANAGEMENT SYSTEM
==================================================
1. Open New Account
2. View Account Details
3. Deposit
4. Withdraw
5. Transfer
6. View Transaction History
7. Account Summary
8. Exit
==================================================
--- Deposit ---
Deposited $50000.00. New balance: $70000.00
==================================================
          BANK ACCOUNT MANAGEMENT SYSTEM
==================================================
1. Open New Account
2. View Account Details
3. Deposit
4. Withdraw
5. Transfer
6. View Transaction History
7. Account Summary
8. Exit
==================================================
--- Withdraw ---
Withdrew $10000.00. New balance: $60000.00
==================================================
          BANK ACCOUNT MANAGEMENT SYSTEM
==================================================
1. Open New Account
2. View Account Details
3. Deposit
4. Withdraw
5. Transfer
6. View Transaction History
7. Account Summary
8. Exit
==================================================
--- Transfer ---
```

```
Target account not found
==================================================
          BANK ACCOUNT MANAGEMENT SYSTEM
==================================================
1. Open New Account
2. View Account Details
3. Deposit
4. Withdraw
5. Transfer
6. View Transaction History
7. Account Summary
8. Exit
==================================================
--- View Transaction History ---
Transaction History:
----------------------------------------------------------------------
1. 2025-08-24 13:47 - deposit: $50000.00 - Balance: $70000.00
2. 2025-08-24 13:47 - withdrawal: $10000.00 - Balance: $60000.00
==================================================
          BANK ACCOUNT MANAGEMENT SYSTEM
==================================================
1. Open New Account
2. View Account Details
3. Deposit
4. Withdraw
5. Transfer
6. View Transaction History
7. Account Summary
8. Exit
==================================================
--- Account Summary ---
Total Deposits: $50000.00
Total Withdrawals: $10000.00
Average Transaction Amount: $30000.00
Total Transactions: 2
==================================================
          BANK ACCOUNT MANAGEMENT SYSTEM
==================================================
1. Open New Account
2. View Account Details
3. Deposit
4. Withdraw
5. Transfer
6. View Transaction History
7. Account Summary
8. Exit
==================================================
Saving account data and generating report...
Report saved to 'bank_accounts_report.txt'

Thank you for using our banking system. Goodbye!

Thank you for trusting our bank!
```


Thank
You