# Python Banking System

A comprehensive console-based banking application with persistent data storage using Python collections and file handling.

## Features

### Core Banking Operations

- **Account Management**: Create accounts with unique 6-digit account numbers
- **User Authentication**: Secure login with password protection
- **Fund Operations**: Deposit, withdraw, and transfer money between accounts
- **Interest Calculation**: Monthly interest calculation for savings accounts
- **Transaction History**: Complete transaction tracking and history viewing
- **Account Modification**: Update personal information and account details

### Data Persistence

- **Automatic Data Saving**: All data saved to bank_data.txt on exit
- **Data Loading**: Previous session data automatically loaded on startup
- **Human-Readable Format**: Dictionary-based storage for easy understanding

### Security & Validation

- **Password Protection**: Minimum 6-character password requirement
- **Input Validation**: Comprehensive error handling for all user inputs
- **Fund Validation**: Insufficient funds checking for withdrawals/transfers
- **Account Validation**: Duplicate account prevention and account existence verification

## Workflow

### Main Menu Operations:

Main Menu
├── 1. Create New Account
├── 2. Login to Existing Account
└── 3. Exit → Save Data → bank_data.txt

### Account Creation Flow:

**Create Account**

├── **Enter Name**

├── **Generate Unique Account Number**

├── **Set Initial Deposit (min $500)**

├── **Select Account Type (Savings/Current)**

├── **Create Password (min 6 chars)**

└── **Save Account Data**

### Login & Account Operations:

**Login**

**├── Enter Account Number**

**├── Enter Password**

**└── Account Menu**

**├── 1. Deposit Money**

**├── 2. Withdraw Money**

**├── 3. Transfer Money**

**├── 4. View Transaction History**

**├── 5. Calculate Interest**

**├── 6. Update Account Information**

**└── 7. Logout**

## Financial Operations

**Deposit**

**Deposit Flow**

**├── Enter Amount**

**├── Validate Positive Amount**

**├── Update Account Balance**

**├── Record Transaction**

**└── Display New Balance**

**Withdrawal**

**Withdrawal Flow**

**├── Enter Amount**

**├── Validate Positive Amount**

**├── Check Sufficient Funds**

**├── Update Account Balance**

**├── Record Transaction**

**└── Display New Balance**

**Transfer**

**Transfer Flow**

**├── Enter Recipient Account**

**├── Validate Account Exists**

**├── Prevent Self-Transfer**

**├── Enter Amount**

**├── Check Sufficient Funds**

**├── Update Both Balances**

**├── Record Two Transactions**

**├── Transfer Out (Sender)**

**├── Transfer In (Recipient)**

**└── Display Both New Balances**

## 🛡️ Security Features

### Input Validation

- **Account Numbers**: 6-digit random generation with uniqueness check
- **Passwords**: Minimum 6 characters with validation
- **Amounts**: Numeric validation with positive value enforcement
- **Account Types**: Restricted to 'savings' or 'current' options

### Business Logic Validation

- **Minimum Deposit**: $500 required for new accounts
- **Sufficient Funds**: Balance checking before withdrawals/transfers
- **Self-Transfer Prevention**: Cannot transfer to own account
- **Account Existence**: Verification before login/transfers

### Error Handling

- **Keyboard Interrupt**: Graceful exit with data saving
- **Value Errors**: Handling of invalid numeric inputs
- **File I/O Errors**: Robust data loading/saving error handling
- **General Exceptions**: Comprehensive error catching and user feedback

## 💾 Data Persistence

### File Storage

- **Location**: bank_data.txt in same directory
- **Format**: Human-readable dictionary structure
- **Timing**: Automatic save on normal/abnormal exit
- **Loading**: Automatic load on application startup

### Data Serialization

- **DateTime Handling**: ISO format conversion for storage
- **Dictionary Preservation**: Maintains exact data structure
- **Cross-Session**: Data persists between application runs

## 🎯 User Experience

### Navigation

- **Exit Anytime**: Type 'exit' at any prompt to return to previous menu
- **Clear Instructions**: Step-by-step guidance for all operations
- **Visual Feedback**: Emojis and clear success/error messages

- **Menu Organization**: Logical grouping of related functions

### Information Display

- **Account Summary**: Current balance and account details
- **Transaction History**: Chronological listing with details
- **Interest Calculator**: Real-time interest estimation
- **Confirmation Messages**: Clear feedback for all operations

## 🚀 Getting Started

### Prerequisites

- Python 3.x installed
- No external libraries required

### Installation

**git clone <repository-url>**

**cd python-banking-system**

**python banking_system.py**

### Usage

1. Run the application
2. Choose to create a new account or login to existing one
3. Perform banking operations
4. Exit to automatically save data
5. Restart to continue from where you left off

## 🛠️ Technical Implementation

### Core Components

- **Single Class Architecture**: BankingSystem handles all operations
- **Collection-Based Storage**: Dictionaries and lists for data management
- **File I/O Operations**: ast.literal_eval() for safe data parsing
- **Datetime Handling**: Proper timestamp management for transactions

### Key Methods

- main_menu(): Central navigation hub
- create_account(): Account registration process
- login(): User authentication system
- deposit()/withdraw()/transfer(): Financial operations
- save_data()/load_data(): Persistence management
- generate_account_number(): Unique ID generation

This banking system demonstrates fundamental programming concepts including data structures, file handling, error management, and user interface design while providing a complete banking simulation experience.