

**Bank account management in Python**

Department of Economy and Business

EMS 331 Data Mining

Fall 2024

Besmira Lushi

Table of Contents

**Requirements3**

**Description4**

Description4

Concepts covered4

Features and tasks5

Functions6

**The code9**

Bank\_account file9

File\_handler file10

Operations file11

Main file15

**Conclusions18**

**Requirements:**

Your project formulation should test your knowledge on the following Python topics:

• - Variables, Comments

• - Arithmetic Operations

• - Strings

• - Slicing

• - If Statements

• - While loops, For loops

• - Lists

• - Tuples

• - Dictionaries

• - Sets

• - Functions, Exceptions

• - Classes

• - Modules

• - Packages

• - Files

• - high-level functions, such as random, zip, map, datetime, lambda, etc.

Project Submission Criteria

* Project Naming: The project should contain a single zipped folder named after the name and surname of the student, and the project’s name. It should contain a short report in Word showing the project’s name, its requirements, the code, input files, test code, and a short presentation.
* Running Code: The code should be fully functional with proper comments.
* Test File: Include a test file in Jupyter Notebook, showing step-by-step all the code ready to be run.
* Input Files: Provide all input files used in the project.
* Charts: Include charts to show the profit of the company and summarize different criteria.
* Presentation: Create a presentation showing the project, its detailed description, programming details mentioning the Python topics used, and a resume of the overall project.

**Concepts covered:** Classes, Functions, Lists, If Statements, File handling, Exceptions.

**Description**: Build a system that tracks bank accounts. Each account is represented by a class with attributes such as balance, account number, and owner. Users can deposit, withdraw, and check balance. All transactions should be logged in a file.

The system will manage bank accounts for users. Each account will have attributes such as:

* **Account Number**
* **Account Owner's Name**
* **Balance**

Users will be able to:

1. **Create new accounts**
2. **Deposit funds**
3. **Withdraw funds**
4. **Check balances**

All transactions will be logged in a file, ensuring persistent storage. Exception handling will ensure robustness against invalid inputs or operations (e.g., withdrawing more than the balance).

**Features**

1. **Account Management**:
   * Create an account.
   * Delete an account.
   * List all accounts.
2. **Transaction Operations**:
   * Deposit funds.
   * Withdraw funds.
   * View account balance.
3. **File Operations**:
   * Save account data and transaction logs to a file.
   * Read account data from a file.
4. **Error Handling**:
   * Handle invalid operations (e.g., overdraft attempts).
   * Manage input errors (e.g., non-numeric amounts).

**Tasks and Subtasks**

1. **Setup and Initialization**
   * Initialize the main script (bank\_account\_management.py).
   * Import required modules (os, random).
2. **Classes**
   * Define a BankAccount class with attributes like account\_number, owner, and balance.
   * Include methods for deposits, withdrawals, and displaying account details.
3. **Data Management**
   * Use a list of BankAccount objects to store all accounts.
   * Implement file operations to read/write account data and transaction logs.
4. **Functions**
   * Define utility functions to:
     + Add an account.
     + Remove an account.
     + Perform transactions (deposit/withdrawal).
     + View account details.
5. **User Interface**
   * Implement a menu-driven interface for interaction.
   * Allow the user to perform operations iteratively until exiting.

**READING AND WRITING FROM A FILE**

import os

from bank\_account import BankAccount

DATA\_FILE = "accounts\_data.txt"

def save\_accounts(accounts):

**with open(DATA\_FILE, 'w') as file:**

for account in accounts:

file.write(f"{account.account\_number},{account.owner},{account.balance}\n")

def load\_accounts():

accounts = []

if os.path.exists(DATA\_FILE):

**with open(DATA\_FILE, 'r') as file:**

for line in file:

account\_number, owner, balance = line.strip().split(',')

accounts.append(BankAccount(account\_number, owner, float(balance)))

return accounts

**ADDING AN ACCOUNT**

def create\_account(accounts):

owner = input("Enter the account owner's name: ")

balance = float(input("Enter the initial deposit amount: "))

account\_number = str(random.randint(100000, 999999)) # Random 6-digit account number

new\_account = BankAccount(account\_number, owner, balance)

**accounts.append(new\_account)**

print(f"Account created successfully! Account Number: {account\_number}")

**VIEW ACCOUNT**

def view\_account\_balance(accounts):

account\_number = input("Enter the account number: ")

for account in accounts:

if account.account\_number == account\_number:

details = account.get\_details()

print(f"Account Number: {details['Account Number']}")

print(f"Owner: {details['Owner']}")

print(f"Balance: {details['Balance']}")

return

print("Account not found.")

**COMPARING Account number**

def delete\_account(accounts):

account\_number = input("Enter the account number to delete: ")

for account in accounts:

if account.account\_number == account\_number:

accounts.remove(account)

print("Account deleted successfully.")

return

print("Account not found.")

**The code:**

bank\_account.py

class BankAccount:

    # Constructor to initialize a new bank account with an account number, owner, and optional initial balance

    def \_\_init\_\_(self, account\_number, owner, balance=0.0):

        self.account\_number = account\_number  # Unique identifier for the bank account

        self.owner = owner  # Name of the account owner

        self.balance = balance  # Initial balance, default is 0.0

    # Method to deposit an amount into the account

    def deposit(self, amount):

        if amount > 0:  # Ensure the deposit amount is positive

            self.balance += amount  # Increase the account balance by the deposit amount

            return f"Deposit successful. New balance: {self.balance}"

        else:

            return "Deposit failed. Amount must be positive."  # Return error message instead of raising an exception

    # Method to withdraw an amount from the account

    def withdraw(self, amount):

        # Check if the withdrawal amount is positive and less than or equal to the current balance

        if amount > 0 and amount <= self.balance:

            self.balance -= amount  # Reduce the account balance by the withdrawal amount

            return f"Withdrawal successful. New balance: {self.balance}"

        elif amount > self.balance:  # If withdrawal amount exceeds the balance

            return "Withdrawal failed. Insufficient funds."  # Return error message instead of raising an exception

        else:

            return "Withdrawal failed. Amount must be positive."  # Return error message instead of raising an exception

    # Method to get the details of the account as a dictionary

    def get\_details(self):

        return {

            "Account Number": self.account\_number,  # Include the account number

            "Owner": self.owner,  # Include the owner's name

            "Balance": self.balance  # Include the current account balance

        }

file\_handler.py

import os  # Import the os module to check if a file exists

from bank\_account import BankAccount  # Import the BankAccount class from the corresponding file

# Name of the file where account data will be stored

DATA\_FILE = "accounts\_data.txt"

# Function to save a list of account objects to a file

def save\_accounts(accounts):

    """

    This function saves the details of all accounts to a file named accounts\_data.txt.

    Each account is written on a separate line in the format:

    account\_number,owner,balance

    """

    with open(DATA\_FILE, 'w') as file:  # Open the file in write mode ('w'), overwriting any existing data

        for account in accounts:  # Iterate through each account in the list

            # Write the account details as a single line, separated by commas

            file.write(f"{account.account\_number},{account.owner},{account.balance}\n")

# Function to load account data from a file

def load\_accounts():

    """

    This function reads account data from the accounts\_data.txt file and recreates a list of BankAccount objects.

    If the file does not exist, it returns an empty list.

    """

    accounts = []  # Initialize an empty list to store account objects

    if os.path.exists(DATA\_FILE):  # Check if the file exists using os.path.exists

        with open(DATA\_FILE, 'r') as file:  # Open the file in read mode ('r')

            for line in file:  # Read each line in the file

                # Split the line by commas to get individual account details

                account\_number, owner, balance = line.strip().split(',')

                # Create a BankAccount object using the extracted details and add it to the list

                accounts.append(BankAccount(account\_number, owner, float(balance)))

    return accounts  # Return the list of loaded account objects

operations.py

# Import the BankAccount class

from bank\_account import BankAccount

import random  # To generate random account numbers

# Function to create a new bank account

def create\_account(accounts):

    # Prompt user for account owner's name and initial balance

    owner = input("Enter the account owner's name: ")

    balance = float(input("Enter the initial deposit amount: "))

    # Generate a random 6-digit account number

    account\_number = str(random.randint(100000, 999999))  # Random 6-digit account number

    # Create a new BankAccount instance

    new\_account = BankAccount(account\_number, owner, balance)

    # Add the new account to the accounts list

    accounts.append(new\_account)

    # Display success message with the new account number

    print(f"Account created successfully! Account Number: {account\_number}")

# Function to delete an existing bank account

def delete\_account(accounts):

    # Prompt user for the account number to delete

    account\_number = input("Enter the account number to delete: ")

    # Loop through the list of accounts and search for the matching account number

    for account in accounts:

        if account.account\_number == account\_number:

            # Remove the account from the list if found

            accounts.remove(account)

            print("Account deleted successfully.")

            return  # Exit function after successful deletion

    # If account not found

    print("Account not found.")

# Function to deposit money into an account

def deposit\_to\_account(accounts):

    # Prompt user for the account number to deposit money into

    account\_number = input("Enter the account number: ")

    # Loop through the list of accounts and find the matching account

    for account in accounts:

        if account.account\_number == account\_number:

            # Prompt user for the deposit amount

            amount = float(input("Enter the deposit amount: "))

            try:

                # Call the deposit method of the BankAccount class

                account.deposit(amount)

                print("Deposit successful.")

            except ValueError as e:

                # Handle any errors (e.g., negative deposit amounts)

                print(e)

            return  # Exit function after successful deposit

    # If account not found

    print("Account not found.")

# Function to withdraw money from an account

def withdraw\_from\_account(accounts):

    # Prompt user for the account number to withdraw money from

    account\_number = input("Enter the account number: ")

    # Loop through the list of accounts and find the matching account

    for account in accounts:

        if account.account\_number == account\_number:

            # Prompt user for the withdrawal amount

            amount = float(input("Enter the withdrawal amount: "))

            try:

                # Call the withdraw method of the BankAccount class

                account.withdraw(amount)

                print("Withdrawal successful.")

            except ValueError as e:

                # Handle any errors (e.g., insufficient funds or invalid withdrawal amount)

                print(e)

            return  # Exit function after successful withdrawal

    # If account not found

    print("Account not found.")

# Function to view the balance and details of an account

def view\_account\_balance(accounts):

    # Prompt user for the account number to view balance

    account\_number = input("Enter the account number: ")

    # Loop through the list of accounts and find the matching account

    for account in accounts:

        if account.account\_number == account\_number:

            # Retrieve the account details (assumed method get\_details)

            details = account.get\_details()

            # Print account details

            print(f"Account Number: {details['Account Number']}")

            print(f"Owner: {details['Owner']}")

            print(f"Balance: {details['Balance']}")

            return  # Exit function after displaying account details

    # If account not found

    print("Account not found.")

# Function to list all existing accounts

def list\_all\_accounts(accounts):

    if accounts:

        # If there are accounts in the list, loop through and print their details

        for account in accounts:

            details = account.get\_details()

            print(f"Account Number: {details['Account Number']}, Owner: {details['Owner']}, Balance: {details['Balance']}")

    else:

        # If there are no accounts

        print("No accounts found.")

main.py

from file\_handler import load\_accounts, save\_accounts  # Import functions to load and save accounts data

from operations import (  # Import all the operations for managing bank accounts

    create\_account, delete\_account, deposit\_to\_account,

    withdraw\_from\_account, view\_account\_balance, list\_all\_accounts

)

def main():

    """

    Main function to manage the bank account system.

    Loads existing accounts from a file, displays a menu for various operations,

    and saves the updated accounts data back to the file upon exiting.

    """

    accounts = load\_accounts()  # Load existing accounts from the data file into a list

    while True:  # Infinite loop to continuously show the menu until the user decides to exit

        # Display menu options

        print("\n=== Bank Account Management System ===")

        print("1. Create Account")

        print("2. Delete Account")

        print("3. Deposit Funds")

        print("4. Withdraw Funds")

        print("5. View Account Balance")

        print("6. List All Accounts")

        print("7. Exit")

        choice = input("Enter your choice: ")  # Prompt the user to select an option

        if choice == '1':  # If the user selects '1', create a new account

            create\_account(accounts)

        elif choice == '2':  # If the user selects '2', delete an existing account

            delete\_account(accounts)

        elif choice == '3':  # If the user selects '3', deposit funds into an account

            deposit\_to\_account(accounts)

        elif choice == '4':  # If the user selects '4', withdraw funds from an account

            withdraw\_from\_account(accounts)

        elif choice == '5':  # If the user selects '5', view the balance of a specific account

            view\_account\_balance(accounts)

        elif choice == '6':  # If the user selects '6', list all accounts in the system

            list\_all\_accounts(accounts)

        elif choice == '7':  # If the user selects '7', exit the program

            save\_accounts(accounts)  # Save the updated accounts data back to the file

            print("Goodbye!")  # Print a goodbye message

            break  # Break the loop and exit the program

        else:  # If the user enters an invalid option

            print("Invalid choice. Please try again.")  # Notify the user and show the menu again

# Entry point for the program

if \_\_name\_\_ == "\_\_main\_\_":

    main()  # Call the main function to start the program

**Conclusions:**

The Bank Management System was created using Python, employing objectoriented programming principles to ensure modularity and clarity.

It features a structured workflow with six primary functionalities: creating accounts, deleting accounts, depositing and withdrawing funds, viewing account balances, and listing all accounts.

The system uses a list to store account objects and relies on method calls within the BankAccount class to perform operations, ensuring encapsulation and clean data handling.

The code includes input validation and error handling to provide a user-friendly experience, and its logic is scalable for future enhancements such as database integration or graphical interfaces.