Class Project, Spring 2021
Due: May 21, 11:59PM (Mountain Time)

The task of this class project is to write a Python program to manage customer accounts for a bank. A real bank account management system is complicated, and here we implement a simplified version. In this simple, small bank account management system, please define a class of bank account objects. Each bank account has the following attributes:

- Account number. This is a unique integer between 100 and 9999 (including 100 and 9999).
- Account owner's name. This is a string, not necessarily unique.
- Account owner's age. This is a positive integer.
- Account owner's residency (in which state the owner lives). This is a string, such as Colorado, Utah, California, and so on.
- Current balance. This is a non-negative floating-point number.
- The date when the account is opened. This is a string in the format of $month/day/year$.

In the class definition, your program needs to define a method (i.e., $\_\_init\_\_$) with proper parameters to initialize the above attributes when a new bank account object is created. In addition, the class needs to contain (at least) three additional methods, i.e., $deposit$, $withdraw$, and $transfer$, for account managements. The class can also contain other methods to access and change attribute values.

Please download the file $account\_data.csv$ from Canvas. This file contains account information, such as account number, owner's name, age, residency, current balance and the date of account opening. This file should be used to initialize the bank account database when the system starts running. This bank account management system needs to provide the following operations for the user:

- User login. This requires the user to provide a username and a password during runtime. Here assume all of us use the same username (i.e., $BIOS6642$) and password (i.e., $python$). The system will successfully start running only if correct username and password are provided by the user. The system allows the user to re-enter username and password, if they are not correct. However, if the user provides 3 successive incorrect inputs for username and/or password, the system (i.e., your program) automatically stops running, and you have to re-start the system to enter username and password. After successful user login, the system should automatically initialize the customer account database by using the file $account\_data.csv$, and then the system will ask the user to provide inputs for account management (see below). Once user login is successful, the system will keep running unless the user explicitly conducts the quit operation to quit the system/program.

- Display all the accounts (i.e., the account attributes) in a tabular format on the screen. The system needs to provide the following four options of how to display the account information, which is sorted by (1) account number (by default), (2) account owner's name, (3) current balance, and (4) the date when the account is opened. Your program needs to implement and provide these four display options

to the user.

- Print account statistics on the screen. This operation displays (1) total balance, average balance and median balance of all accounts; (2) total balance, average balance and median balance of accounts for each following age group: (a) under 35, (b) 35 - 44, (c) 45 - 54, (d) 55 - 64, (e) 65 - 74, and (f) 75 and older; (3) total balance, average balance and median balance of accounts for each state/residency in the database. You might want to use *statistics.mean* and *statistics.median* (or *statistics.median_high*) methods in the *statistics* module at the Python Standard Library to calculate the average/mean and median values. In addition, please create a grouped bar plot for average balance and median balance of accounts by age group, and create another grouped bar plot for average balance and median balance of accounts by residency.

- Display a subset of accounts on the screen, given a certain criterion. Your program needs to implement the following options: (1) display the accounts with the owners having ages between *min_age* and *max_age*, which are positive integers given by the user during runtime; (2) display the accounts with the owners having a residency, *residency*, which is a string provided by the user during runtime; (3) display the accounts with opening dates between *min_date* and *max_date*, which are strings provided by the user during runtime.

- Query and display a specific account given (1) an account number or (2) an account owner's name. If the given account number or owner's name does not exist in the current account database, the system should print a proper message and ask the user to re-enter another account number or owner's name. If multiple accounts are found given an account owner's name (note that the owner's name is not unique), please display them all on the screen.

- Open a new account. The new account owner's name, age, residency and current balance are provided by the user during runtime. The account number should be randomly generated between 100 and 9999. Please note that each account number is unique and it must be different from other account numbers in the current account database. The date of opening the account is the day when the system runs (i.e., current date. You can use the method *datetime.date.today*() in the *datetime* module at the Python Standard Library to get the current date). If the account is successfully created, the system automatically updates the account database, i.e., add this newly opened account to the current database, and also updates the file *account_data.csv* on your hard drive.

- Delete an account given an account number, which is provided by the user during runtime. If the given account number exists in the current account database, the system deletes the corresponding account from the database and also from the file *account_data.csv* on your hard drive. If the given account number is not found in the current database, the system should print a proper message and ask the user to re-enter another account number.

– Conduct transactions for a specific account. In this operation, the system provides the following options to perform transactions for a given account, which (i.e., account number) is provided by the user during runtime:
  - Deposit money into the given account. The amount of money (i.e., a floating-point number) is provided by the user during runtime. If the deposit is successfully performed, the system automatically updates the account database and also the file *account_data.csv* on your hard drive.

  - Withdraw money from the given account. The amount of money (i.e., a floating-point number) is provided by the user during runtime. In order to obtain a successful withdrawal, the current balance (before the withdrawal) of the account must be not lower than the amount that the user intends to withdraw. If the withdrawal is successfully performed, the system automatically updates the account database and also the file *account_data.csv* on your hard drive.

  - Transfer money from the given account (sender) to another account (receiver). The receiver account number and the amount of money (i.e., a floating-point number) to transfer are provided by the user during runtime. In order to obtain a successful transfer, the current balance (before the transfer) of the sender account must be not lower than the amount that the user intends to transfer. If the transfer is successfully performed, the system automatically updates the account database and also the file *account_data.csv* on your hard drive. Note that both sender and receiver accounts need to be updated.

  - Display current balance for the given account on the screen.

  - Account settings such as changing owner's name, age and/or residency for the given account. If this change is successfully performed, the system automatically updates the account database and also the file *account_data.csv* on your hard drive. The account number, current balance and the date of account opening cannot be changed via this option.

  - Display recent transactions for the given account on the screen. Here the recent transactions are the last few (e.g, 5) transactions of money deposit, withdrawal, and transfer as well as the activities in account settings, such as changing owner's name, age and/or residency. Please note that these transactions include the events/activities of money deposit, withdrawal, transfer, and owner's name/age/residency changes that occurred during the last running of the bank account management system. This means that these transactions should be recorded in a file (e.g., a .txt file) for persistent storage on your hard drive such that they would not be deleted when you quit the system. When you re-run the bank account management system and want to display the previous transactions for a specific account on the screen, you would need to read this file to get the information of the events/activities.

– Provide an option to quit the system.

For the operations or options above, please properly handle exceptions that might occur. For example, if an operation asks the user to enter a number 2 but receives an English word *two*, the system should be able to handle a potential exception of $ValueError$.

Please test each operation/option above, take a screenshot of the output, and write a report to clearly show the output of each operation/option (e.g., compile the screenshots and clearly show the output of each operation/option in the report). Please submit your source codes (.py or .ipynb file) and your report (a PDF or Microsoft word file) to Canvas. The source codes need to be properly documented or commented such that they are readable.

You might want to use the starter code provided on Canvas: *class_project.ipynb* or *class_project.py*. The *class_project.ipynb* is for Jupyter Notebook and *class_project.py* is for script mode. Please choose the one that works best for you.