CSC 401 ASSIGNMENT FOUR

Due Date: Tuesday, Aug 18th by 11:58 PM

The purpose of this assignment is to assess your understanding of

- Dictionary, tuple, set data types
- Read csv file type (do not import csv library)
- Encapsulation in functions
- Namespaces: global vs local
- Exception handling
- Program modularity

SUBMISSION

- Include your full name as a comment on the first line of your Python programs.
- Save each program to a separate file labeled as YourName_hw4_1.py and YourName_hw4_2.py.
- Upload each file to Submissions folder in D2L.
- Late assignments are not accepted. To earn partial credit you must submit the work you completed by the deadline.

Note: You may not use Python statements, functions, data type, etc. that where not discussed in the reading assignment or the lecture notes/videos for this week or previous weeks. This is a class for students who have not programmed before and I expect everyone to code on the same level. If you have a better way of writing the code, then upload two versions: one that codes according to the specifications and the other that demonstrates advanced programming techniques.

I encourage you to

- Use computational thinking to solve the problems. These are straight-forward solutions, but developing a good habit of analyzing the problem and describing the steps will serve you well as the problems get more complex

- Test your code thoroughly. You are **not** required to include your test cases as part of your programs. But, **I do require that you code according to the problem specifications.**

PROBLEM 1 (25 PTS) DICTIONARY, SET, PUNCTUATION

At the end of books, there usually is an index that lists the pages where a certain word appears. In this problem, you will create an index for a text but, instead of the page number, you will use the line

numbers. You are to write a Python function **index (fname, letter)** that takes as input the name of the text file and the first letter of the words for which you are to create the line number index. For each word that begins with 'letter', you are to print the corresponding line numbers. You should

1. Open and read the text file only **once**

2. Remove all punctuation from the text except the apostrophe (single quote).
   Use p = punctuation.replace(" ' ", " ") to replace the apostrophe in p with an empty string, and use p in the maketrans statement

3. Create a dictionary of the text in which the key-value pairs are described as follows:

   a. Key: The keys are the individual words found in the file and stored in the dictionary in uppercase.

   b. Values: The values are a list that contains the line number in the file where the word (key) is found. For each word, a line number should be listed only once (you may convert list to set to remove duplicates) In the output, use commas to separate the line numbers, but they are not printed as lists (no brackets). The words in the output should be in alphabetical order, i.e. A -Z

4. For each word that begins with the specified letter, find all the line numbers in which the letter appears.

5. Print the total number of words that begin with the specified letter. If there are no words in the text for a specific letter, print the message 'There are no words that begin with " x " ' where x is the specified letter.

6. Printed output should be formatted as shown in the sample case.

7. This program gives you the opportunity to use statements and concepts we have discussed so far in the course plus exception handling. All **variables must be local** and passed into and from functions**.**

   Sample case: The images below show partial data output

```
>>> index('Pride_and_Prejudice.txt', 'k')
Word            Line Nbr.
KEENER          2159
KEENEST         2665, 2939, 983
KEEP            261, 647, 4231, 1417, 3465, 653, 919, 667, 669, 1825, 2977, 2473, 311, 3641, 957, 205, 2893, 2523, 485, 3685
103, 2027, 3567, 247, 3579
KEEPING         1553, 3023, 3933, 1463

KNOWLEDGE       385, 2435, 1925, 2067, 2461, 4259, 1839, 1841, 2233, 313, 3257, 2239, 3263, 575, 1983, 2243, 89, 4075, 2931,
1535
KNOWN           129, 2817, 3073, 3971, 2823, 3207, 2697, 3083, 1935, 3985, 4241, 1683, 21, 281, 1049, 287, 2591, 1441, 2849,
1443, 2213, 3365, 2729, 3373, 2095, 2863, 3001, 2235, 2875, 2125, 2511, 3535, 2257, 1491, 2259, 2519, 1753, 1627, 2523, 2527
2273, 2529, 2787, 1125, 2789, 1639, 2663, 2791, 4069, 1899, 4077, 3825, 1017
KNOWS           1155, 3491, 2663, 1737, 2923, 3023, 2195, 3513, 2719
KYMPTON         3439

There are 34 words that begin with " K "
```

PROBLEM TWO USER-DEFINED FUNCTIONS (60 PTS.)

You are to simulate a very simple ATM machine that has a full keyboard. An account owner (user) should be able to enter their pin number and select from a menu of transactions: Deposit, Withdraw, Balance, and Quit. You are to assume that the user has only one account on which these transactions can be performed. This account is associated with the user's pin number.

The ATM should properly and regularly communicate with the user. The ATM should get information on current account from a file (**accounts.csv).** This file is posted with the assignment. Each line of the file contains a 4-digit code (pin), first name, last name, and the account balance. Partial contents of accounts.csv file follows:

```
9967,Harry,Potter,10.05
4466,Kay,Winthrop,356
2813,Mickey,Mouse,125
2245,Donald,Duck,1235456.5
2315,Daffy,Duck,234
1234,Hermoine,Grainger,23000
6345,Darth,Vader,98745.35
```

Note: whenever I use **'Code the exception'**, this means you need to include code for a 'try/except' block to catch the exception caused by a run-time error.

You are to use the following **main()** function code in your program. This function controls the flow of the program. You **may not** change the names of the functions or the arguments that are passed or modify main() in any way. You must use only local variables (i.e. global variables are not allowed)

```
def main():
    dict = startUp('accounts.csv') #dict dictionary name in main()
    if dict == None:
        return
    pin, name = verifyPin(dict)
    if pin != None and name != None:
        while True:
            print()
            choice = menu(name)
            if choice in 'Dd':
                deposit(pin, dict)
            elif choice in 'Ww':
                withdraw(pin, dict)
            elif choice in 'Bb':
                b = balance(pin, dict)
                msg = 'Your current balance is ${:,.2f}'
                print (msg.format(b))
            elif choice in 'Qq':
                fname, lname = quit(pin, dict)
                if fname == None and lname == None:
                    pass
                else:
                    str = '\n{} {}, thanks for using the ABC Bank'
                    print(str.format(fname, lname))
                    break
    return
```

STEPS

1. (10 pts.) Write a function **startUp(fname)** that takes as input a filename that contains the account owner (user) information and current account balance.

    - Initialize the dictionary in the function. All variables in this program **must be local** variables.

    - If the filename is invalid, **Code an exception** ('Cannot get to the file') and return None.

    - If the filename is valid, read the file and store each line in the **dictionary.**

        i. The **key is** the 4-digit code (**pin),** and the **value** is a **list** with the user first name, last name and balance. **balance** is a **float** data type. Do an explicit conversion to float on that list item.

        ii. The function **returns the dictionary** if the file was successfully read and the dictionary successfully filled with the file data; if the file as not successfully opened/read, **return None.**

2. (5 pts.) Write a function **verifyPin()** that takes as input the dictionary.

    - Prompt the user to enter a pin number.

- If the pin is valid, that is, it is 4-digits and is in the dictionary, **return pin and name** where pin is the dictionary key and name is the user's first name. If the pin is invalid, print ('Incorrect pin') and **return (None, None)**.

3. (5 pts.) Write a function **menu(name)** that takes as input the user's first name, displays the user's name and the menu options:
   D: Deposit
   W: Withdraw
   B: Balance
   Q: Quit
   If the user enters a value that is not D, W, B, Q, then print the message 'Valid choices are D, W, B, Q, try again' and display the menu options again. When a valid number is entered, the function should **return t**he number of the chosen option.

4. (10 pts.) Write a function **verifyAmount()** that takes no input.

   - Inside a loop, prompt the user for an amount to be deposited or withdrawn; at this point it does not matter which it is. The amount must be converted to float. If the amount is negative, print ' Negative amount. Please try again'.

   - **Code an exception** (' Invalid amount. Use digits only.). An interruption would be caused if the user enters a string instead of a numeric value, or hits the enter key without entering anything.

   - Stay in the loop until a valid number is entered and **return amount**.

5. (5 pts.) Write a function **deposit(pin, d)** that takes as input the user's pin number and the dictionary. The function calls getAmount(), calculates the new balance and updates the dictionary. The function does not return a value, but it should have a return statement.

6. (10 pts.) Write a function **withdraw(pin, d)** that takes as input the user's pin number and the dictionary. The function calls **getAmount().** If the amount to be withdrawn is greater than the balance print ('Insufficient funds to complete the transaction'). The user should be prompted to enter a new amount, until the amount entered is less than the balance or 0. The function uses the balance in the dictionary to calculate the new balance. The balance is updated in the dictionary. This function does not return a value, but it should have a return statement.

7. (5 pts.) Write a function **balance(pin, d)** that takes as input the user's pin number and dictionary and **return**s the user's balance from the dictionary.

8. (5 pts.) Write a function **quit(pin, d)** that takes as input the user's pin number and dictionary. The user is prompted as to whether or not she wishes to leave the transaction. If she does, then the function returns the user's first and last name and the message 'thanks for using ABC Bank.'. If she does want to leave the application, then **return(None, None)**.

IF YOU HAVE ANY QUESTIONS REGARDING THIS ASSIGNMENT, PLEASE PORT THEM TO THE ASSIGNMENT FOUR DISCUSSION FORUM

Assignment Four Grading Rubric

Learning outcomes:

- Create dictionary
- Initialize and update dictionary
- Create user-defined functions
- Pass data to functions and from functions
- Understand and use namespaces and variable scope
- Code using program modularity

Problem Number	Proficient 25 – 22	Nearing Proficiency 21 - 18	Needs Improvement 17 – 0
One	Correctly defined user-defined function index() with arguments fname and letter Correctly initializes the dictionary Correctly reads the file and saves the requested data to the dictionary. Correctly removes the punctuation from the dictionary key (except for the apostrophe) Correctly saves dictionary key in all uppercase Dictionary values do not contain duplicate values Dictionary is printed as described and shown in the example. Printed dictionary is sorted Correct calculation and printing of number of lines in the file. Correct calculation and printing	Adequately defined user-defined function index() with only one argument Dictionary is not initialized in the function Correctly reads the file but data is not save in the dictionary Removed some of the punctuation from the dictionary key Dictionary key is not all in uppercase Dictionary values contain duplicate values Dictionary is printed but not as shown in the example Printed dictionary is not sorted Incorrect calculation of the number of lines in the file Incorrect calculation and	Minimally or no definition of user-defined function index() with no arguments Dictionary is not initialized Incorrectly reads the file Punctuation is not removed from the dictionary key Dictionary key is not in uppercase Dictionary values not stored as a list Dictionary is not printed

	of number of words that begin with the requested letter.	printing of number of words that begin with the requested letter	
For all problems	User-defined functions do not use any data types, statement, methods or operators that have not been presented in week 4 or prior week's lectures or reading assignments. Complete thorough testing User- defined functions have no syntax errors User-defined functions execute with no run time errors All problem specifications are correctly coded	User-defined functions use a data type, statement, method or operator that have not been presented in week 4 or prior week's lectures or reading assignments. Only tested with given data or partially tested User-defined functions have one or two syntax errors. User-defined functions do not execute because of a run time error Some deviations from specifications	User-defined functions use more than one data type, statement, method or operator that have not been presented in week 4 or prior week's lectures or reading assignments. Minimal or no testing User-defined functions have more than two syntax errors User-defined functions do not execute because of run-time error Hardly follows the specifications

Problem Two

Step Numbers	Proficient 5 - 4	Nearing Proficiency 3	Needs Improvement 2 – 0
	Shows a comprehensive understanding of encapsulation in user-defined functions, namespace local variables, exception handling and data validation.	Shows an adequate understanding of encapsulation in user-defined functions, namespace local variables, exception handling and data validation.	Shows a minimal or no understanding of encapsulation in user-defined functions, namespace local variables, exception handling and data validation.
2, 3, 5, 7, 8	Correctly defines user-defined functions (UDF) as described All UDFs accept the correct number of inputs and variable names All UDFs have a return statement and return either None or the appropriate value(s) verifyPin() correctly performs all requested validation menu() correctly accepts only the valid choice entries deposit() updates dictionary with correct deposit amount	Adequately defines user-defined functions (UDF) as described Most UDFs accept the correct number of inputs and variable names Most UDFs have a return statement and return either None or the appropriate value(s) verifyPin() correctly performs most requested validation menu() correct accepts most of the valid choice entries deposit() updates dictionary	User-defined functions (UDF) have minimal or no correct code Minimal or no UDFs accept the correct number of inputs or variable names Minimal or no UDF's have a return statement verifyPin() incorrectly performs or does not perform requested validation menu() does not check for valid entries deposit() does not update

	balance() retrieves correct balance amount from the dictionary quit() requires input from user regarding if she really want to leave the application.	with incorrect amount balance() retrieves incorrect balance amount from the dictionary quit() requires input from user; correctly responds she wants to stay in the application, but not if she wants to leave or vice versa	balance in the dictionary balance() does not retrieve balance from the dictionary quit() does not get input from user and leaves the application
Problem Numbers	**Proficient 10 – 9**	**Nearing Proficiency 8 - 7**	**Needs Improvement 6 – 0**
1, 4, 6	Correctly defines user-defined functions (UDF) as described All UDFs accept the correct number of inputs and variable names All UDFs have a return statement and return either None or the appropriate value(s) startUp() correctly codes exception processing; initializes dictionary as local variable; stores data from file in dictionary withdraw() calls appropriate function to get withdrawal amount; updates dictionary with validated amount; checks amount for insufficient funds. verifyAmount() does not take arguments; request amount from user; processes exception for non-numeric data; validates for appropriate numeric amount value;	Adequately defines user-defined functions (UDF) as described Most UDFs accept the correct number of inputs and variable names Most UDFs have a return statement and return either None or the appropriate value(s) startup() codes most of the exception processing correctly; initializes dictionary as a global variable; stores most data from file in dictionary withdraw() does not call function to get withdrawal amount; updates dictionary with incorrect amount; allows a negative balance verifyAmount() does not take arguments, requests amount from user; does not code exception for non-numeric data; validates for appropriate numeric amount value.	User-defined functions (UDF) have minimal or no correct code Minimal or no UDFs accept the correct number of inputs or variable names Minimal or no UDF's have a return statement startup() incorrectly or does not perform exception processing; initializes dictionary as a global variable; stores no data in the dictionary withdraw() does not call function to get withdrawal amount; does not update the dictionary; allows a negative balance verifyAmount() allows user to enter amount only once; does not code exception for non-numeric data; does not validate for appropriate numeric amount value.
For all problems	User-defined functions do not use any data types, statement, methods or operators that have	User-defined functions use a data type, statement, method or operator that have not been	User-defined functions use more than one data type, statement, method or

	not been presented in week 4 or prior week's lectures or reading assignments. Complete thorough testing User- defined functions have no syntax errors User-defined functions execute with no run time error All problem specifications are correctly coded Uses all local variables	presented in week 4 or prior week's lectures or reading assignments. Only tested with given data or partially tested User-defined functions have one or two syntax errors. User-defined functions do not execute because of a run time error Some deviations from specifications Uses some local and some global variables	operator that have 4 or prior week's lectures or reading assignments. Minimal or no testing User-defined functions have more than two syntax errors User-defined functions do not execute because of run-time errors Hardly follows the specifications Uses all global variables