COMP 361/651 Data Analytics Techniques Assignment 3

10 points possible

1 (6.5 pts.)**.** Write a Python script that reads in names and corresponding bank balances (integers, without a '$' or commas) and makes a dictionary of the associations. You can prompt the user first for how many name-income pairs (s)he wants to enter. Use **`int(input())`** for input. After constructing the initial balances, repeatedly do the following:

> Prompt for a transaction, using a code number, for each bank customer: a deposit (code 1), a withdrawal (code 2), or no transaction (code 0). Update the dictionary of bank balances and output the updated customer-balance associations. Make a dictionary of the transactions where the keys are the same as for the balance dictionary (names of customers); positive values represent deposits, negative values represent withdrawals, and a zero represents no transaction.

Maintain a list of the transaction dictionaries (one for each iteration). When the loop exits, output the current balances and the list of transactions in a well-formatted table. This could be implemented with recursion as well as with a loop.

Example run

```
Number of accounts: 3
Name: Ed
Initial balance: 3.50
Name: Al
Initial balance: 6.50
Name: Sue
Initial balance: 7.75

Processing Ed
Transaction type:
0. Nothing
1. Deposit
2. Withdraw
: 2
Amount: 4.00

Processing Sue
Transaction type:
0. Nothing
1. Deposit
2. Withdraw
: 1
Amount: 2.00

Processing Al
Transaction type:
0. Nothing
1. Deposit
2. Withdraw
: 2
Amount: 8.00

Current Balances
        Ed         Sue          Al
    -$0.50      $9.75     -$1.50
0 to exit, any other key to continue: 1

Processing Ed
Transaction type:
0. Nothing
1. Deposit
```

```
2. Withdraw
: 1
Amount: 3.50

Processing Sue
Transaction type:
0. Nothing
1. Deposit
2. Withdraw
: 0

Processing Al
Transaction type:
0. Nothing
1. Deposit
2. Withdraw
: 1
Amount: 1.00

Current Balances
        Ed         Sue          Al
     $3.00      $9.75     -$0.50
0 to exit, any other key to continue: 1

Processing Ed
Transaction type:
0. Nothing
1. Deposit
2. Withdraw
: 0

Processing Sue
Transaction type:
0. Nothing
1. Deposit
2. Withdraw
: 0

Processing Al
Transaction type:
0. Nothing
1. Deposit
2. Withdraw
: 1
Amount: 3.25

Current Balances
        Ed         Sue          Al
     $3.00      $9.75      $2.75
0 to exit, any other key to continue: 0

Record of Transactions
Round            Ed         Sue          Al
-----------------------------------------
   1          -$4.00      $2.00      -$8.00
   2           $3.50      $0.00       $1.00
   3           $0.00      $0.00       $3.25
>>>
```

Use type hinting in annotations so that Mypy may catch infelicities. Use the special types defined in the **`typing`** module and include an assertion specifying that the integer code for the transaction type is between 0 and 2 (inclusive).

2 (3.5 pts.). We cover a couple of preliminaries before presenting the problem.

Using **with** for Opening Files

It is recommended to open files within the scope of a **with** statement. When execution leaves normally the scope of a **with** or an exception is raised within this scope, the open file is automatically closed. (The lecture slides thus don't introduce **with** until exceptions are presented.) The following illustrates the syntax of **with** (note the **as** keyword, followed by the identifier to which the file pointer is bound).

```
with open('example_in.txt', 'r') as infile:
    for line in infile:
        print(int(line) + 3)
```

This assumes that file **example_in.txt** has one integer per line. It inputs, in sequence, each line and prints the sum of the integer on that line plus 3. Note that what is read is a string and must explicitly be converted to an integer. As an example, suppose that the contents of **example_in.txt** are as follows.

```
3
4
5
```

Then the output produced by the above code is

```
6
7
8
```

Within a **with** statement, we may open more than one file; we must separate **open(...) as *name*** clauses with commas. As an example, the following does what the last code snippet did but writes the results to file **example_out.txt**.

```
with open('example_in.txt', 'r') as infile, \
     open('example_out.txt', 'w') as outfile:
    for line in infile:
        outfile.write(str(int(line) + 3) + '\n')
```

Note that, as the **with** statement does not fit comfortable on one line, we escape (using '****') the new line to carry the statement over to the next line. Also, since the **write()** method requires a string argument, we convert the sum to a string for writing. The contents of **example_out.txt** are exactly what is printed by the previously shown code snippet.

Converting a String of Integer Representations to a List of Integers

It is common to have lines in a data file that consist of integers separated by whitespace (e.g., sequences of one or more spaces or tabs). We generally want to convert such a string to a list of integers. (Similar comments apply when a line consists of floats.) The string method **split()** does exactly this. For example,

```
>>> "23 145 2  43".split()
['23', '145', '2', '43']
```

Where **strg** is any string, **strg.split()** returns the list of substrings of **strg** delimited by whitespace. Provide a string argument to **split()** to specify a different delimiter. For example, the following delimits with commas. (Note that the string on which **split()** is invoked is usually the value of a variable.)

```
>>> "23,14,52,43".split(',')
['23', '14', '52', '43']
```

We'll go into further detail on how to split strings (including more sophisticated uses of **`split()`**) when we cover regular expressions.

At this point, we have converted our string of integer representations into a list of integer strings. It remains to convert these strings into real integers. Recall that **`int()`** converts its argument to an integer if it can. Thus, **`int('37')`** evaluates to the integer 37 while **`int('1a')`** raises an exception. To convert a list of integer strings to the list of the integers thereby represented, we use list comprehension. For example,

```
>>> [int(x) for x in ['23', '14', '52', '43']]
[23, 14, 52, 43]
```

So, to convert a string of integer representations separated by whitespace to a list of the integers thereby represented, first split the string on the whitespaces, then use list comprehension to apply **`int()`** to all the resulting string representations.

The Assigned Problem

Write a function **`file_sums()`** that has two parameters, say, **`infile`** and **`outfile`**, that are bound to file names. Each line in **`infile`** has one or more integers separated by whitespace. This function writes as a line to **`outfile`** the sum of the integers on the corresponding line of **`infile`**. Open both files in a **`with`** statement. Use a list comprehension to convert the split line into a list of integers. Note that function **`sum()`**, passed a list of numbers, returns the sum of those numbers. Recall that file method **`write()`** takes a single, string argument; that argument must end with a new line (use **`'\n'`**) for what is written to be a complete line (so that what is written next is on the following line).

For example, if the contents of the input file **`data.txt`** (available from the assignment page, and which you should use as a test file) are

```
3 12 72
32 54
29
78 31 28 9
7 83
```

then the contents of the output file become

```
87
86
29
146
90
```

For both problems, include multi-line docstrings for all except very simple functions (for which you use one-liners).