CSCI 1133, Spring 2023
Homework 10
Due: 11:55 pm, Tuesday April 4th, 2023

Due Date: Submit your solutions to Gradescope by 11:55 pm, Tuesday, April 4th. Do not upload anything to Canvas and PLEASE be sure to use proper naming conventions for the file, classes, and functions. We will NOT change anything to run it using our scripts.

Late Policy: Submissions will be accepted up to 24 hours late (11:55 pm on Wednesday) for a 1 point penalty, up to 48 hours late (11:55 pm on Thursday) for a 2 point penalty, or up to 72 hours late (11:55 pm on Friday) for a 3 point penalty. Submissions past 11:55 pm on Friday will not be accepted.

Unlike the computer lab exercises, this is not a collaborative assignment. See the syllabus and read section "Academic Dishonesty" for information concerning cheating. Always feel free to ask the instructor or the TAs if you are unsure of something. They will be more than glad to answer any questions that you have. We want you to be successful and learn so give us the chance to help you.

Purpose: The purpose of this assignment is to introduce user-defined functions and documentation, essential concepts for building larger programs in a reasonable and organized manner.

Instructions: This assignment consists of 3 problems, worth a total of 30 points. You must create a Python file called `hw10.py`, and submit it to Gradescope. If you have questions, ask!

Because your homework file is submitted and tested electronically, the following are very important:

- Submit the correct file, hw10.py, through Gradescope by the due date.
- Your program should run without errors when we execute it using Python 3.

Submit your work to Gradescope under the Homework 10 assignment. Make sure that your file is called hw10.py: if you don't match the file name exactly then the testing script won't work and you'll lose points.

The following will result in a score reduction equal to a percentage of the total possible points:

- Incorrectly named/submitted source file or functions (20%)
- Failure to execute due to syntax errors (30%)
- Bad coding style (missing documentation, meaningless variable names, etc.) (up to 30%)

Some of the points in this assignment are given based on the Gradescope autograder, so doing something that breaks the gradescope tests (naming your file incorrectly, naming your function incorrectly, having syntax errors, or using the input() function somewhere the autograder doesn't expect) are likely to lose you all of those points.

Documentation:
Use the following template for EVERY function that you write. The docstring below should be placed inside of the function, just after the function signature.

```
'''
Purpose:
Parameter(s):
Return Value:
'''
```

Download the hw10files.zip folder from Canvas, extract the sample files. You'll want to create your hw10.py file within the extracted hw10 folder.

Problem A. (10 *points*) **Find the Min**

Write a function `find_the_mins(dictionary)` that takes in a single argument, a dictionary. The keys in this dictionary will be strings, and the values will be lists of numeric values. The function should return a new dictionary with all the same keys as the argument dictionary. The values are the minimum value in each list associated with the key in the original input dictionary. If a value in the input dictionary is an empty list, the key should not be included in the output dictionary.

Hint:

- Using the built-in function `min()` is allowed.

Constraints:

- The function should not mutate the input dictionary in any way.

Examples :

```
>>> find_the_mins({"One": [1, 2, 3],
                   "Two": [4, 5, 6],
                   "Three": [44, 41, 41]})

{'One': 1, 'Two': 4, 'Three': 41}


>>> find_the_mins({"X": [-1, 3, -33, 100],
                  "Y": [], "Z": [1, 1, 0],
                  "W": [3]})
```

```
{'X': -33, 'Z': 0, 'W': 3}


>>> find_the_mins({
           "In": [12, 100, 72, 74, 11, 89],
           "Out": [14, 100],
           "Failed": [-1, 0, 0, 1],
           "Queued": [3],})

{'In': 11, 'Out': 14, 'Failed': -1, 'Queued': 3}
```

Problem B. (10 *points*) **Contacts**

Write a function `find_the_contact(directory, name, field)` that takes in three arguments, a directory which is a dictionary, a name which is a string, and a field which is a string. In the directory, keys will be names (strings), and the values will be nested dictionaries. An example of directory follows

{'Lee': {'Phone': '643-756-5612', 'Email': 'example@umn.edu', 'Username': 'Lee'},
'Katie': {'Email': 'test_email@gmail.com', 'Username': 'Kat'},
'Amanda': {'Phone': '234-462-4513', 'Email': 'no_email@yahoo.com', 'Username': 'Ama'},
'Nat': {'Phone': 612-379-5234: '', 'Username': 'Nat'}}

The `name` value is a string which may be a key in the directory. The `field` is the key for the information you want to return for that name.

If a given name and field is in the directory, you should return the corresponding value. If a given name is in the directory, but does not have a key for the correct field, you should return `None`.

If a given name is not in the directory, you should add it to the dictionary with a nested dictionary value. This nested dictionary should have one key `'Username'` and the value should be the first three characters of `Name`. In addition, if the field is not `Username`, you should return `None`; if the field is `Username`, you should return the created `Username`.

Examples :

```
>>> a_dict = {'Lee': {'Phone': '643-756-5612', 'Email':
'example@umn.edu', 'Username': 'Lee'},
'Katie': {'Email': 'test_email@gmail.com', 'Username': 'Kat'},
'Amanda': {'Phone': '234-462-4513', 'Email':'no_email@yahoo.com',
'Username': 'Ama'},
'Nat': {'Phone':'612-379-5234', 'Username': 'Nat'}}
>>> print(find_the_contact(a_dict, "Lee", "Phone"))
643-756-5612
>>> print(find_the_contact(a_dict, "Nat", "Email"))
None
>>> print(find_the_contact(a_dict, "Abi", "Email"))
None
```

```
>>> a_dict
{'Lee': {'Phone': '643-756-5612', 'Email': 'example@umn.edu',
'Username': 'Lee'}, 'Katie': {'Email': 'test_email@gmail.com',
'Username': 'Kat'}, 'Amanda': {'Phone': '234-462-4513', 'Email':
'no_email@yahoo.com', 'Username': 'Ama'}, 'Nat':
{'Phone':’612-379-5234’, 'Username': 'Nat'}, 'Abi': {'Username':
'Abi'}}
>>> print(find_the_contact(a_dict, "Abby", "Username"))
Abb
>>> a_dict
{'Lee': {'Phone': '643-756-5612', 'Email': 'example@umn.edu',
'Username': 'Lee'}, 'Katie': {'Email': 'test_email@gmail.com',
'Username': 'Kat'}, 'Amanda': {'Phone': '234-462-4513', 'Email':
'no_email@yahoo.com', 'Username': 'Ama'}, 'Nat':
{'Phone':’612-379-5234’, 'Username': 'Nat'}, 'Abi': {'Username':
'Abi'}, 'Abby': {'Username': 'Abb'}}
```

Problem C. (10 *points*) **Shopping List**

A surgery checklist, developed by the World Health Organization and adopted by more than 20 countries, was declared by the independent to be "the biggest clinical invention in thirty years."[1]

You use a similar checklist for your weekly shopping, putting all the items you need to purchase on your list. Your checklist is stored in a CSV file. Your checklist file uses the following format:

- There are three columns.
- The first column is the store name where you want to buy the item, as your list is cumulative for all stores you need to shop at.
- The second column is the name of the item you wish to purchase.
- The third column is a positive integer, greater than or equal to zero, representing the total amount of those items you want to buy.

For example:

```
Target,Sheets,1
Target,Bananas,4
Target,Pen,11
Target,water,0
```

Write a function `create_lists(file_name)` that takes in a string file_name that is the path of your checklist. The function should return a dictionary in which each key is a name present in the column corresponding to the stores you want to shop at, and the value is a nested dictionary. In the nested dictionary, each key is the name of an item you wish to purchase at that store, and the value is an integer sum of the total number of that item you wish to buy.

An item may appear multiple times in a file for the same store. If this happens, the integer you return should be the sum of all the values for that item at that given store.

- Do not change the checklist file

[1] Read more about this checklist in Atul Gawande's "The Checklist Manifesto: How to Get Things Right"

Examples (remember, the order of keys in a dictionary doesn't matter)**:**

```
>>> create_lists('small.csv')
{'Target': {' Bananas': 4, ' Pen': 11, ' Sheets': 1, ' water': 0}}

>>> create_lists('medium.csv')
{"Dan's": {'Apple': 6,
           'Candy': 10,
           'Cereal': 7,
           'Cheese': 2,
           'Chips': 1,
           'Rice': 11,
           'Water': 12},
 'Islands': {'Candy': 10,
           'Cheese': 1,
           'Donuts': 6,
           'Eggs': 8,
           'Sandwich': 4},
 'Target': {'Bananas': 0,
           'Bread': 0,
           'Hair brush': 9,
           'Milk': 9,
           'Pen': 6,
           'Soda': 7,
           'Tea': 3,
           'Tuna': 11}}
```

```
>>> create_lists('large.csv')
{'7-11': {'Batteries': 2,
          'Cheese': 17,
          'Frozen Peas': 6,
          'Hair Brush': 3,
          'Hand Sanitizer': 6,
          'Insta Pot': 8,
          'Tomato Sauce': 11,
          'Toothpaste': 1,
          'Tuna': 1,
          'Water': 6},
 'Corner Store': {'Canned Soups': 3,
                  'Chewing Gum': 8,
                  'Chips': 15,
                  'Legumes': 6,
                  'Pen': 5},
 "Dan's": {'Apple': 9,
           'Bananas': 10,
           'Bread': 12,
           'Candy': 5,
           'Donuts': 5,
           'Maple Syrup': 8,
           'Rice': 2},
 'Islands': {'Eggs': 0, 'Meatballs': 13, 'Sandwich': 6, 'Tea': 12},
 "Nate's": {'Candy': 12,
            'Cheese': 0,
            'Dog Food': 12,
            'Folding Table': 7,
            'Honey': 2,
            'Soda': 13,
            'Vinegar': 7},
 'Target': {'Cereal': 5,
            'Dish Soap': 11,
            'Milk': 12,
            'Nuts': 2,
            'Sweater Box': 16}}
```