Homework 2

Total Points: 100 (Correctness and Style)

Due: Tuesday, January 23nd, 11:59 pm

Starter Files

Download hw02.zip. Inside the archive, you will find starter files for the questions of this homework. You cannot import anything to solve the problems.

IMPORTANT: Coding and Docstring Style

This is a reminder that your code style will be graded. Here are a few useful links:

Style Guide on Course Website

Style Guide Examples

Submission

By the end of this homework, you should have submitted the homework via Gradescope. You may submit more than once before the deadline; only the final submission will be graded. Refer to Lab00 directions for submission.

Testing

At any point of the homework, use the following command to test your work:

```
>>> python3 -m doctest hw02.py
```

General Notes and Requirements

1. **DO NOT IMPORT ANY PACKAGES.**
2. For all questions, only basic loops (for/while) and/or conditional statements (if/elif/else), NO list comprehension (will be covered in future lectures). Other built-in methods are allowed.
3. Please add your own doctests (**at least three**) as the given doctests are not sufficient. **You will be graded on the doctests**.
4. For file handling questions, you should create new files in the files folder for your own doctests, and submit them along with your code file and given files. These files can be created manually using a notepad. You don't need to use code to generate them.

5. In this assignment you can assume that the input given to you is valid and does not have any errors. **No input validation required for HW02.**
6. Read the **Submission** section at the end closely before you submit to Gradescope.

Required Questions

Question 1

Since you are a new citizen in `Dictionary Land`, you need to learn the rules of this land. For now, you just walk around and record everything you see into two lists: *keys* and *values*. Write a function that takes two lists (of the same size) that can contain data of many types: in particular you can expect:

- Integers, floats
- Strings (including empty)
- Lists, Tuples (including empty)
- Dictionaries (including empty)
- Boolean

Your function should return a list of tuples, where each tuple contains two elements: first elements from the lists, then second elements from the lists and so on, in the same order as the inputs. **BUT**: you can only keep the items from the `key` list if the type is the **valid** type for a dictionary key. If it is not the case, then the tuple contains a string "not valid".
Return an empty list if the lists are empty.

Note:

1. To create a tuple, you could use the same approach as you use for lists but replace the square brackets [] with parentheses (). For example, `var = (1, 2)`. There is no append for tuples, but you could use it in the same way (i.e. `var[1]`) to access an element from a tuple.

For example:

Input	Output	Explanation
keys = ["fun", ["not so much"]] values = [("learning",), 6]	[('fun', ('learning',)), ('not valid',)]	The first two items from both lists are "fun" and ("'learning",). Since strings can be dictionary keys (they are immutable), we create a tuple with these items and append it to the list.

		The second pair [“no so much”] and 6 can’t be added, since lists are mutable and can’t be used as a dictionary’s key.

```
def valid_pairs(keys, values):
    """
    >>> keys = ["fun", ["not so much"]]
    >>> values = [("learning",), 6]
    >>> valid_pairs(keys, values)
    [('fun', ('learning',)), ('not valid',)]

    >>> keys = [1, "fun", [2], (1,), {}]
    >>> values = [1, {}, (1,), "island", [2]]
    >>> valid_pairs(keys, values)
    [(1, 1), ('fun', {}), ('not valid',), ((1,), 'island'), ('not
valid',)]

    >>> keys =[]
    >>> values =[]
    >>> valid_pairs(keys, values)
    []
    """
```

Question 2

Write a function that takes a list of tuples (the output from the previous question, i.e. a list of (<key>, <value> or ‘not valid’) tuples) and converts it to a dictionary, where <key> is a key in the output dictionary and values are a list of values associated with that key. Return an empty dictionary if no tuples are provided.

Notes:

1. You should skip the tuple if it is in the ‘not valid’ form from Question 1.

```
def tup_to_dict(tuples):
    """
    >>> lst = [(1, 1), ('fun', {}), ('not valid',), ((1,), 'island'),\
    ('not valid',)]
    >>> tup_to_dict(lst)
    {1: [1], 'fun': [{}], (1,): ['island']}

    >>> lst = [("not valid",), ("tree", "green"), ("tree", "blue"),\
    (1, ["key"])]
    >>> tup_to_dict(lst)
```

```
    {'tree': ['green', 'blue'], 1: [['key']]}
    """
```

Question 3

After walking around the `Dictionary Island`, you realize that they have very different food items: Ziblo, Crinx, Yumzo, Fluzu,... You do not have much money left so you wanted to find out the average price of these foods.

Write a function that takes two parameters: a list of foods (as strings) and a list of lists, where each inner list represents a price for each item from the first list. The latter looks like this:

`[[<price for food1>, <price for food2>], [<price for food1>, <price for food2>], …]`

Your function should return a dictionary where the keys are the food items from the first list and the values are the average amount of money for this food.

`{‘food1’: <average price for food1>, ‘food2’: <average price for food2>}`

Assumptions:

- The food item list is never empty.
- If the input list of prices is empty, then the dictionary values should be 0 for all food items.
- There are always 2 elements in the inner lists (if not empty) and food item list.
- Round your answer to two decimal places.
- Elements in the food item list will not be the same.

Hints:

- Create a dictionary with food items as keys and initialize their values as 0.
- Use a loop that iterates through the second list. Since the length of the inner lists is fixed, you can use the form:

```
for item1, item2 in items:
```

```
def foods(items, prices):
    """
    >>> items = ["Ziblo","Crinx"]
```

```
    >>> prices = [[1, 2], [3, 4], [5, 6]]
    >>> foods(items, prices)
    {'Ziblo': 3.0, 'Crinx': 4.0}

    >>> items = ["Yumzo","Fluzu"]
    >>> prices = [[2, 3], [3, 4], [4, 5], [5, 6.2]]
    >>> foods(items, prices)
    {'Yumzo': 3.5, 'Fluzu': 4.55}
    """
```

Question 4

In order to cook a meal you need to estimate the total price of the items you already have. Write a function that takes two parameters: a dictionary (similar to the output from Question 3), where the keys are strings (food items) and values are non-negative numbers. The second parameter is the list of ingredients for the recipe. Your function should return a total price of the items that are in the given dictionary as well as in the list of ingredients.

```
def total_cost(items_you_have, ingredients):
    """
    >>> items = {'Yumzo': 3.5, 'Fluzu': 4.55}
    >>> ingredients = ['Yumzo', 'Fluzu']
    >>> total_cost(items, ingredients)
    8.05

    >>> items = {'Yumzo': 3.5, 'Fluzu': 4.55}
    >>> ingredients = ['Ziblo','Yumzo', 'Fluzu']
    >>> total_cost(items, ingredients)
    8.05

    >>> items = {'Yumzo': 3.5, 'Fluzu': 4.55, 'Crinx': 5.34}
    >>> ingredients = ['Ziblo','Yumzo']
    >>> total_cost(items, ingredients)
    3.5
    """
```

Question 5

You realized that you do not have enough ingredients, so you went shopping. You went to different stores and purchased various items. Then you realized that there are many repetitive items, so you need to be more organized.

Write a function that takes in a dictionary, where each key is the name of the store and each value is a list of foods purchased from that store. The format looks like this:

```
{‘shop1’: [<list of purchased items>], ‘shop2’:[<list of purchased items>], ...}
```

Your function returns a list of unique foods from the dictionary lists, sorted alphabetically. If there is no item (either because the input dictionary is empty or all value lists in the dictionary are empty), return an empty list.

Hint:

1. You may use the built-in `sorted()` function to sort the list.
2. Nested loops might be needed. The link for the video is here for your convenience.

```
def shopping(grocery_bag):
    """
    >>> bag = {'shop1': ['Munchi', 'Chirp'], 'shop2': ['Blipz',
'Sprova']}
    >>> shopping(bag)
    ['Blipz', 'Chirp', 'Munchi', 'Sprova']
    >>> bag = {'shop1': ['Munchi', 'Chirp'], 'shop2': ['Munchi',
'Sprova']}
    >>> shopping(bag)
    ['Chirp', 'Munchi', 'Sprova']
    >>> bag = {'shop1': ['Munchi', 'Chirp', 'Blipz'], \
    'shop2': ['Munchi', 'Sprova', 'Sprova'], \
    'shop3': ['Blipz', 'Sprova', 'Eggs']}
    >>> shopping(bag)
    ['Blipz', 'Chirp', 'Eggs', 'Munchi', 'Sprova']
    """
```

Question 6

After spending all your money, you start thinking about getting a job in Pythonia. You asked around and were told that there is an opened position in the nearby district called "Filemystica", where you are required to to work with files. You gladly took the position but some training is required first. Questions below involve reading and writing to files. Make sure that the folder called `files` is in the same directory as `hw02.py`

For all the questions below you will solve the same problem using a different approach.

Write a function that takes a `filepath` to an **non-empty** file and returns the number of lines in the file. Assume that the file always exists.

Hint: In some parts below, the built-in function `str.split()` might be useful. When using, note what the argument to this function means.

Question 6.1:

Use the `for` keyword to implement the function.

```
def count_lines_1(filepath):
    """
    >>> count_lines_1('files/test1.txt')
    6
    >>> count_lines_1('files/test2.txt')
    24
    """
```

Question 6.2:

Use the `read()` function to implement the function.

```
def count_lines_2(filepath):
    """
    >>> count_lines_2('files/test1.txt')
    6
    >>> count_lines_2('files/test2.txt')
    24
    """
```

Question 6.3:

Use the `readlines()` function to implement the function.

```
def count_lines_3(filepath):
    """
    >>> count_lines_3('files/test1.txt')
    6
    >>> count_lines_3('files/test2.txt')
    24
    """
```

Question 7

You were given access to the files that have information about Pythonia citizens! It would be great to make a few friends, so you decided to use your knowledge of file manipulation to extract all email addresses.

Given a `filepath` to a .txt file containing the first name, last name, number of days at Pythonia, email address and expert level of Python language on each line (always separated by commas).
Write a function that returns a list of emails in the same order as in the input file. Return an empty list if the file is empty. Assume there are no empty lines if the file is not empty and the file always exists.

Hints:

1. Note that when reading files, an entire line is represented as a long string.
2. To get rid of extra spaces or newline characters at the end of a string, consider using `str.strip()`
3. Now with a string of comma-separated elements, you could split it into a list and retrieve the information needed (do not forget about `str.split()`)
4. List indices that are not -1, 0, 1 are also considered magic numbers.

The input file format for each line is:
`f_name,l_name,days,email,expert_level`

Example of an input file: (See the first doctest for output)

```
milo,frost,555,milo@pythonia.py,2
ben,chen,376,ben@discussion.py,5
Bryce,hackel,398,bryce@gradescope.py,5
```

```
def emails(filepath):
    """
    >>> emails('files/file1.txt')
```

```
    ['milo@pythonia.py', 'ben@discussion.py', 'bryce@gradescope.py']
    >>> emails('files/file2.txt')
    ['milo@pythonia.py', 'ben@discussion.py', 'bryce@gradescope.py', \
'dave@gmail.com']
    >>> emails('files/file3_empty.txt')
    []
    """
```

Question 8

Many customers do not know how to use files properly, so your first job is to fix given strings. Write a function that takes a string, converts it to a valid file path, adds a .txt extension and writes the number of digits and non-digits in a given string.

Notes:

- The string will have a valid form to become a filename, except possible characters at the end. You need to remove them.
- You only need to add `files/` to the file path.
- To get rid of extra spaces or newline characters at the end of a string, consider using `str.strip()`

Remark:

The `with open...print...` in doctests opens the file you write in and checks if the content is correct. It does not hint towards the question itself.

Example: `filepath_to_fix = 'list123'`
In the given string, red characters are digits and blue characters are not, so output file should contain:

```
3
4
```

The file should be found at: **files/**list123**.txt**

```
def counter(filepath_to_fix):
    """
    >>> counter('list123')
    >>> with open('files/list123.txt', 'r') as outfile1:
    ...     print(outfile1.read().strip())
    3
    4
```

```
>>> counter('one1two2three3')
>>> with open('files/one1two2three3.txt', 'r') as outfile2:
...     print(outfile2.read().strip())
3
11
>>> counter('tricky ')
>>> with open('files/tricky.txt', 'r') as outfile2:
...     print(outfile3.read().strip())
0
7
"""
```

Question 9

Now it's time to combine reading and writing to files together in the same function.

You found such a useful set of files: how many people crossed different bridges in order to come to Pythonia! Unfortunately, the files were not organized in a clean way: each file contains the name of the bridge and a number of people crossed it during different periods of time, separated by a colon.

Example:

Input File `files/bridges1.txt`:

```
bridge1:5
bridge2:3
bridge3:1
bridge1:4
```

Given a `filepath` to a `.txt` file containing the information described above, write a function that combines totals the number of people crossed same bridges and writes this information using the same format into a file `files/combined_bridges.txt`

Output File `files/combined_bridges.txt`:

```
bridge1:9
bridge2:3
bridge3:1
```

Notes:

1. Assume on each line there is a valid integer and files will not be empty.
2. Assume the file always exists.

3. You will always write to the `files/combined_bridges.txt` file, and you don't need to remove the last new line in this output file.

Hints:

1. When reading in a line, what is the data type?
2. To get rid of extra spaces or newline characters at the end of a string, consider using `str.strip()` first before processing your number.
3. Think what data structure might be used to keep the information before writing it to a file

```
def combine_bridges(bridges):
    """
    >>> combine_bridges('files/bridges1.txt')
    >>> with open('files/combined_bridges.txt', 'r') as f:
    ...     print(f.read().strip())
    bridge1:9
    bridge2:3
    bridge3:1

    >>> combine_bridges('files/bridges2.txt')
    >>> with open('files/combined_bridges.txt', 'r') as f:
    ...     print(f.read().strip())
    bridge1:5
    bridge2:0

    >>> combine_bridges('files/bridges3.txt')
    >>> with open('files/combined_bridges.txt', 'r') as f:
    ...     print(f.read().strip())
    bridge2:3
    bridge3:1
    bridge1:5
    """
```

Submission

When submitting to Gradescope, you can include the files folder by compressing it with your hw02.py file, creating a .zip file you can submit that zip file to Gradescope. If you have your hw02.py and files folder inside of another folder, make sure that you select hw02.py and the files folder directly to compress it, rather than compressing the encompassing folder, as it will cause issues with your submission.

You can do this by going into File Explorer or Finder and selecting **both** the files folder and the hw02.py file. Right-click and then select either

Windows: Send to -> Compressed (zipped) folder
Mac: Compress 2 items

Your .zip file should have the following files if you open it:

```
files/
--- age1.txt
--- age2.txt
--- ... (other files)
hw02.py
```

Please submit the homework via Gradescope. You may submit more than once before the deadline, and only the final submission will be graded. Please refer to the Gradescope FAQ if you have any issues with your submission.