CS-UY 1114: Lab 10

Dictionaries

You must get checked out by your lab CA prior to leaving early. **If you leave without being checked out, you will receive 0 credits for the lab.**

Restrictions

The Python structures that you use in this lab should be restricted to those you have **learned in lecture so far (topics learned in Rephactor may only be used if they have also been taught in lecture)**. Please check with your teaching assistants in case you are unsure whether something is or is not allowed!

Create a new python file for each of the following problems.

Your files should be named *lab[num]_q[num].py* similar to homework naming conventions.

Problem 1: *Dictionary Warmup*

Write the outputs of the following code snippets by hand.

```
def f1(my_dict):
    temp = 0
    for val in my_dict.values():
        temp = temp + val
    return temp
```

```
def f2(my_dict):
    temp = ""
    for key in my_dict:
        if temp < key :
            temp = key
    return temp
```

```
def f3(my_dict, k, v):
    if k in my_dict:
        my_dict[k] = v
```

```
def main ():
    a_dict = {"Kimberly" : 13, "Amalie" : 27, "Mariam" : 19,
"Oleg" : 23}
    print(f1(a_dict)) # line 1
    print(f2(a_dict)) # line 2
    f3(a_dict, "Mariam", 30)
    print(a_dict) # line 3
```

(a) What is the output produced by the line `print(f1(a_dict))` (line 1)?

(b) What is the output produced by the line `print(f2(a_dict))` (line 2)?

(c) What is the output produced by the line `print(a_dict)` (line 3)?

Problem 2: *Dictionary Fun*

Given the dictionary:

```
my_dict = {"a": 15 , "c": 35 , "b": 20}
```

Write Python code to do the following:

(a) Print all the keys

(b) Print all the values

(c) Print all the key, value pairs

(d) Print all the key, value pairs in order by key

Hints:

- Use the list method sort(). If given a list of tuples, the method will sort on the first items in the tuple.

Problem 3: *Vowel Tracking*

Define the following functions in order to properly track the number of vowels in a file. Define the function `read_file()` , which accepts a `filename` and returns the contents of a file. In this specific implementation, we ask that you complete all steps regarding file I/O **within a try except block**. This includes opening the file, retrieving the data from the file, closing the file, and returning the contents (or returning an empty dictionary if the file can't be found). For best practice, **don't** do all these steps within the try part of the try/except block (what other component of the block can we put the extraction method into?).

Additionally, define `count_vowels()`, which will accept a `filename` string and count the total number of each vowel in the file. This information will be stored into a dictionary. If a certain vowel does not exist in the file, it should not appear in the final dictionary. Finally, return this dictionary. `read_file()` should be called within `count_vowels()`.

```
def count_vowels(filename):
    """
    Counts total number of each vowel in the given file.

    :param filename: str of the file to read from
    :return: dict of format {"vowel": int}
    """
```

If the function is ran on `sample_text.txt` whose content looks like this:

```
Desperate, they asked for the help
of a fairy godmother
who had them lock the princess
away in a tower,
there to await the kiss...
of the handsome Prince Charming.
```

The expected output should be:

```
{'E': 18, 'A': 12, 'O': 10, 'I': 7}
```

Problem 4: Sentence Cleaner

In this question, you must define a function `sentence_cleaner()`. This function will accept a string, remove all duplicate words within a string, and return the result. In addition, the words in the returned string should be in the same order that was given by the input string. Think about how we can keep track of duplicates, and how to keep the order of the words **HINT:** (Shallow/Deep copy?).

An example input can look like this:

```
'I love cats, I love fish, and I love birds, but I love dogs more than all of them'
```

When passed as an argument into the function, the output looks like this:

```
I love cats, fish, and birds, but dogs more than all of them
```

Problem 4: *Making a Merger*

You work at a large tech corporation called SkywardSolutions that just acquired an up-and-coming startup called formul-AI-c. Due to this acquisition, you are responsible for merging your company's employee database with your acquisition's employee database. Each company has stored their employee data in a Python dictionary, where the key is the employee ID and the value is a list containing information in the order name, position, and department such as the following:

```
example_database = {100: [“John”, “Developer”, “IT”]}
```

Create a function called `merge_databases()` that accepts two databases (dictionaries) and returns a new dictionary of the two successfully merged together. In merging the two databases, it is very important to make sure that no two employees have the same employee ID. In the event that two employees have the same employee ID, prioritize the SkywardSolutions employee's ID and give the formul-AI-c employee a new ID that numerically follows the highest current employee ID. Also, print the name of the employee who was given a new employee ID as well as previous and new employee ID.

For example, using the following main…

```
def main():
```

```
    skyward_database = {100: [“John”, “Developer”, “IT”]}
    formulaic_database = {100: [“Gerald”, “Social Media
Manager”, “Marketing”]}
    merged_database = merge_databases(skyward_database,
formulaic_database)
```

The expected output would be:

```
Gerald had their employee ID changed from 100 to 101
```

And `merged_database` would be equal to the following:

```
{100: [“John”, “Developer”, “IT”], 101: [“Gerald”, “Social
Media Manager”, “Marketing”]}
```

You can use the following main definition to test your code:

```
def main():
    skyward_database = {100: [“Alice”, “Manager”, “HR”], 101:
[“Arnold”, “Team Lead”, “IT”], 102: [“Chris”, “Developer”, “IT”]}
    formulaic_database = {101: [“Lucas”, “Developer”, “IT”], 102:
[“Anna”, “Intern”, “Marketing”], 103: [“Muhammad”, “Research
Analyst”, “Marketing”]}
    merged_database = merge_databases(skyward_database,
formulaic_database)
```

Expected output:

```
Lucas had their employee ID changed from 101 to 103
Anna had their employee ID changed from 102 to 104
Muhammad had their employee ID changed from 103 to 105
```

Assumptions:

- You may assume that each company's database will be valid meaning that there will be no conflicting employee IDs before merging.
- You may assume that employee IDs will be sequential (i.e., having three employees with the IDs 100, 101, and 110 is not valid)