CS-UY 1114: Lab 11

Dictionary and List Comprehension

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: *So You Think You Can Comprehension?*

Use Python's list and/or dictionary comprehension syntax to generate the following lists and dictionaries:

```
[1, 2, 4, 8, 16, 32, 64, 128]
```

```
[0, 2, 4, 6, 8]
```

```
{0: 0,
 4: 16,
 8: 64,
 12: 144}
```

```
{4: [4, 8, 12, 16],
 5: [5, 10, 15, 20],
 6: [6, 12, 18, 24],
 7: [7, 14, 21, 28],
 8: [8, 16, 24, 32]}
```

Problem 2: *Fill In The Blank*

Finish the following list comprehension syntax. The result is a list of characters of the input repeated twice. Do not use any arithmetic operators or additional libraries. Your answer must use my_str and length.

```
my_str = input("Enter a string:")
length = len(my_str)
print([_____________________________________])
```

```
my_str = "Python" →
["P","y","t","h","o","n","P","y","t","h","o","n"]
```

```
my_str = "Java" →  ["J","a","v","a","J","a","v","a"]
```

Problem 3: *Vowel Tracking Revamped*

You need to track the vowels in a file again but this time, modify your solution from `lab10q3` to count the number of vowels present through dictionary comprehension. Define the function `count_vowels()` below. `count_vowels()` will accept a `filename` string and count the total number of each vowel in the file. This information will be stored into a dictionary. In this version, the dictionary will ***always*** have `"A"`, `"E"`, `"I"`, `"O"`, `"U"` as keys.

```
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:

```
{'A': 12, 'E': 18, 'I': 7, 'O': 10, 'U': 0}
```

Hints:

- Recall the `str.count()` method. How can this function help you in your dictionary comprehension?

Problem 4: *Grocery List*

Preparing for grocery shopping can be done in two steps: taking inventory of what you currently have, then using that information to determine how many of each item you need to buy. Your program should help complete this task in two functions, each function reflecting one step of this two-step process.

Part 1: *Creating an Inventory of Our Fridge Items*

In a function called `def create_grocery_inventory()`, continuously prompt the user for input of an item name and the current quantity they own of this item. These values should be separated by a comma. You may assume the user will provide a valid input in the format of `item_name,item_amount`.

Then, this function should store this information inside of a dictionary where the keys are the `item_name` and the values are the `item_amount`. Finally, return this dictionary.

The definition of the function is as follows:

```
def create_grocery_inventory():
    """
    Prompts for user input to populate a grocery list

    :return: a dictionary of item keys with their quantities as
values
    """
    # Code here
```

Note: The user may input the same `item_name` more than once. There should only be one dictionary entry per item!

Part 2: *Creating a Grocery List*

In a function called `def create_grocery_list(fridge_inventory)`, given a dictionary `fridge_inventory`, which reflects the current quantities of items in a fridge, continuously prompt the user for an item giving an item name and the quantity desired. These values should be separated by a comma. You may assume the user will provide a valid input in the format of `item_name,item_desired_amount`.

Then, based on the information in `fridge_inventory` and user input, you must create an appropriate entry in a new dictionary where keys are the `item_name` and values are the `amount_to_buy` after your calculations. For example, if we already own 5 apples and want to have at least 3 apples, we do not need to buy any more apples. Finally, return this dictionary.

The definition of the function is as follows:

```
def create_grocery_shopping_list(fridge_inventory):
    """
    Prompts for user input to populate a grocery shopping list
given the
    current inventory of items

    :param fridge_inventory: dictionary of str key (item name)
and int quantity
    :return: dictionary of item keys with their quantities to
buy as values
    """
    # Code here
```

Constraint:

> You may assume the user will only input items that already exist in the fridge. The user will additionally only input an item once.

You may use the following main definition to test your code.

```
def main():
    fridge_inventory = create_grocery_inventory()
    print()
    grocery_list = create_grocery_shopping_list(fridge_inventory)
    print()

    print("Your shopping list, based off of what you have in your
fridge, should be:")
    print(grocery_list)
```

The following is an example of a possible output:

```
Please enter the item and quantity you own separated by a comma or
DONE when complete: apple,3
Please enter the item and quantity you own separated by a comma or
DONE when complete: banana,2
Please enter the item and quantity you own separated by a comma or
DONE when complete: coconut,1
Please enter the item and quantity you own separated by a comma or
DONE when complete: banana,3
Please enter the item and quantity you own separated by a comma or
DONE when complete: DONE

Please enter the item and quantity you desire separated by a comma or
DONE when complete: apple,4
Please enter the item and quantity you desire separated by a comma or
DONE when complete: banana,6
Please enter the item and quantity you desire separated by a comma or
DONE when complete: coconut,1
Please enter the item and quantity you desire separated by a comma or
DONE when complete: DONE

Your shopping list, based off of what you have in your fridge, should
be:
{'apple': 1, 'banana': 1, 'coconut': 0}
```

Problem 5: *Contact List*

In this problem, you will write a program which uses a dictionary to store phone numbers of contacts. You will name your dictionary contacts. It will store the names as keys and phone numbers as values. Your program should have the following functions:

Part A: *Adding Contacts*

Write a function `add_entry(contacts, name, number)` that adds an entry to the dictionary contacts.

```
def add_entry(contacts, name, number):
    """
    Adds a new entry to a dictionary of contacts

    :param contacts: dictionary of contacts
    :param name: str of contact name
    :param number: str of contact's phone number
    """
    # Code here
```

The function takes a dictionary, contacts, a name as a string and a number as a string. It adds an entry to contacts with the name as the key and the number as the value. The item should only be added to the dictionary if the following is true:

- An entry in contacts with the key name does not exist already.
- Number is a valid phone number. A valid phone number is a string with 10 digits. For example, `2014567890` is a valid phone number. But `201a45b789` or `20145678` are both invalid.
- Your function should display an error message if the entry cannot be added. Test your function with valid and invalid numbers and duplicate names to ensure that it works correctly.

Part B: *Searching for a Contact*

Write a function `lookup(contacts, name)`. This function should return the phone number associated with the name in contacts. If the input name is not found in contacts, return an error message that indicates so. Test your function with valid and invalid inputs to ensure that it works correctly.

```
def lookup(contacts, name):
    """
    Returns phone number for a contact name if contact exists
in contacts
```

```
    :param contactS: dictionary of contacts
    :param name: str of contact name
    :return: str of contact's number if contact in contacts,
otherwise error
    """
    # Code here
```

Part C: *Deleting a Contact*

Write a function `delete_entry(contacts, name)` that deletes the entry with the key name from contacts. If the input name is not found in contacts, print an error message that indicates so. Test your function with valid and invalid inputs to ensure that it works correctly.

```
def delete_entry(contacts, name):
    """
    Deletes a contact from a contacts dictionary

    :param contacts: dictionary of contacts
    :param name: str of contact name
    """
    # Code here
```

Part D: *Printing all Contacts*

Write a function `print_all(contacts)` that prints all the entries in contacts. Your function should print both the name and the phone number.

```
def print_all(contacts):
    """
    Displays all contacts in contacts
```

```
    :param contacts: dictionary of contacts
    """
    # Code here
```

For example, the output of contacts could be:

```
Ricky      1234567890
Kim        2345678901
Violet     3456789012
```

Use the tab character `"\t"` for spacing between the name and phone number. Test your function to ensure that it works properly.

Part E: *Main Function*

Write a `main()` function to test your program. Your main function should create an empty dictionary, contacts. Then it should prompt the user to input an option. The options are:

- Q: Quit the program
- A: Add an entry to contacts
- L: Look up a phone number
- P: Print all entries in contacts
- D: Delete an entry from contacts

Your program should continue to prompt the user for an option until the user enters `Q`. If the user enters `A`, your program should prompt the user for a name and a phone number and add the contact. If the user enters `L`, your program should prompt the user for a name and lookup the contact. If the user enters `P`, your program should print all the entries in contacts. If the user enters `D`, your program should prompt the user for a name and delete the contact. If the user enters an invalid option you should print an error message.

Here is an example of the output of your program:

```
Please enter an option: A
Please enter a name: Ricky
Please enter a phone number: 1234567890
Please enter an option: A
Please enter a name: Brian
Please enter a phone number: 2345678901
Please enter an option: A
Please enter a name: Elsa
Please enter a phone number: 3456789012
Please enter an option: L
Please enter a name: Ricky
1234567890
Please enter an option: L
Please enter a name: Tinos
Tinos is not in the contact list
Please enter an option: P
Ricky:      1234567890
Brian:        2345678901
Elsa:     3456789012
Please enter an option: D
Please enter a name: Ricky
Please enter an option: P
Brian:        2345678901
Elsa:     3456789012
Please enter an option: Q
```