# Week 11: In-class Exercises

**Resource:** http://blue.smu.edu.sg/is111/resource-ice11-2019.zip

1. **[Difficulty: *]** Write a function **has_same_value** that accepts two dictionaries, **d1** and **d2** and a str named **key**. It returns True if the key is found in both dictionaries and the key has the same value. Otherwise, this function returns False. You are given **q1-tester.py**.

2. **[Difficulty: *]** Write the **q2.py** program: This program will keep prompting the user for words until the user enters an empty string. Your program is supposed to note down the frequency of words entered.
   The program then keeps prompting the user for a word and print outs the frequency of the word, until such word is prompted that is not found(not previously entered by the user).

```
D:\IS111\ICE11>python q2.py
Enter the word >apple
Enter the word >apple
Enter the word >orange
Enter the word >pear
Enter the word >orange
Enter the word >pear
Enter the word >pear
Enter the word >

Enter query word>orange
2

Enter query word>pear
3

Enter query word>banana
Not found.
Bye bye
```

3. You are given a file called **q3-tester.py**. Define the following functions in **q3.py**:

   a. **[Difficulty: *] get_common_misspelling_dict()**: The function takes in a filename and returns a dictionary where the key is the misspelled word (e.g. 'thast'), value is the list of corrected word (e.g. ["that", "that's"]). The file **misspellings.txt** contains one word per line. Do remember to remove the spaces from both ends and the trailing end of line character. If the file contains:

   ```
   adressed-> addressed
   adressing-> addressing, dressing
   ```

   **This** method will return the following `dict` object:
   **{**`'adressed': ['addressed'], 'adressing': ['addressing', 'dressing']}`

   **Note**: You are given `misspellings.txt`.

b. **[Difficulty: **] auto_correct_sentence()**: The function takes in a sentence and returns a list of words from the sentence correcting possibly misspelled words, i.e., a list of words that are not found in the English dictionary. For example, auto_correct_sentence('I studdy Infomation Systems at Singapore Managment University') should return 'I study Information Systems at Singapore Management University'. If the first character of the misspelled word is capitalized, you should capitalize the first character of the misspelled word too.

   If there is more than one possible correct words, then randomly pick one of the words. For example, "thast" is a mis-spelling of either "that" or "that's". Therefore, the answer for "Thast a good one" can either be "That a good one" or "That's a good one".

**Reference:** https://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings/For_machines

4. **[Difficulty: **]** You are given a file called **q4-phone-book.txt**. The format is as follows:

```
<name>|<Mobile|Home|Pager|Fax>|<Phone Number>
```

A sample is shown below:

```
George Leung|Mobile|98987676
Eric Wong|Home|67449908
Michelle Lee|Mobile|96667777
Eric Wong|Mobile|91234567
Michelle Lee|Pager|88776655
Michelle Lee|Fax|88776656
```

Implement the following functions in **q4.py**:

a. **get_phone_book()**: The function takes in a filename and returns a dictionary where the key is the name-<Mobile|Home|Pager|Fax>, and the value is the phone number.
b. **get_phone_numbers_for()**: The function takes in a `str` (`name`) and a dictionary with key containing <name>-Mobile|Home|Pager|fax and value that contains contact number. The function must return a dictionary that represents contact numbers of the `name` where the key is a `str` value (Mobile or Home or Pager or Fax) and the value is the phone number.

You can test your code with **q4-tester.py** provided.

5. **[Difficulty: **]** You are given a file called **q5-tester.py**. Write a function called **reverse** in **q5.py** that accepts a dictionary of strings mapped to strings as a parameter and returns a new map that is the reverse of the original. The reverse of a dictionary is a new dictionary that uses the values from the original as its keys and the keys from the original as its values. Since a dictionary's values need not be unique but its keys must be, you will have each value mapped to a List of keys. For example, if the input parameter map consists of the following

| Key(person's name) | Value(Quiz Score) |
|---|---|
| Alfred | 81 |
| Elise | 61 |
| Billy | 41 |
| Daniel | 41 |
| Charlie | 54 |

The return value of the method should be a dictionary consisting of the following:

| Key(Quiz Score) | Value (person's name) |
|---|---|
| 41 | Billy, Daniel |
| 61 | Elise |
| 81 | Alfred |
| 54 | Charlie |

(The order of the keys and values does not matter.) (question adapted from "Building Java Programs", P766)

6. **[Difficulty: ***]** A group of students need to line up for the procession of an event. We only have the information about every pair of students who are next to each other. We use a list of tuples to represent this line-up information: Each element of the list is a tuple that contains the name of a student (say A) and the name of the student behind A.

   For example, ("Alice", "Bob") means Bob is behind Alice. The list [("Alice", "Bob"), ("Bob", "Chris")] means Bob is behind Alice and Chris behind Bob.

   For the student who is at the very end of the line-up, we use an empty string to indicate that there's nobody behind this student. E.g., ("Darren", "") means there is nobody behind Darren, i.e., Darren is at the end of the line-up.

   Given a list of tuples that contains the line-up information of every pair of students who are next to each other, implement a function that returns the students in a list based on their order in the line-up.

   Example 1: If the function is invoked like this:

```
neighbours = [("Chris", "Darren"), ("Alice", "Bob"),
              ("Darren", ""), ("Bob", "Chris")]
print ( get_lineup(neighbours) )
```

the statement generates the following output:

```
["Alice", "Bob", "Chris", "Darren"]
```

Example 1: If the function is invoked like this:

```
neighbours = [("Mary", "Jason"), ("John", "Alan"),
              ("Jason", "George"), ("Alan", "Christie"),
              ("Christie", "Mary"), ("George", "")]
print ( get_lineup(neighbours) )
```

the statement generates the following output:

```
['John', 'Alan', 'Christie', 'Mary', 'Jason', 'George']
```

Note: If the list passed to the function is empty, the function returns an empty list. There is no test case provided for this in **q6-tester.py**.

7. **[Difficulty: ***]** Let us use dictionaries to represent a family tree. The key is the person's name. The value will be a list that contains this person's children. If a person has no children, no key with that specified name will be found in the dictionary.

   Given the following relationship:

   a. Frank has 2 children (Mary & Jane)
   b. Jane has 2 children (Nick & Wendy)
   c. Mary has no child.

```
   Frank
   /   \
Mary  Jane
      /  \
   Nick  Wendy
```

The following family tree can be represented using a dictionary.

```
>>> tree = {}
>>> tree['Jane'] = ['Nick', 'Wendy']
>>> tree['Frank'] = ['Mary', 'Jane']
```

Define a function called `get_family_members()` that takes 2 parameters:

a. `family_tree`: a dictionary that represents a family tree.
b. `name`: the name of the person whom we are interested to locate his family members.

The `head` may have several children, grandchildren, great-grandchildren, etc. The function returns all the members of this person's family (excluding himself/herself) in a list. The members should be ordered by generation. In other words, **a person of the *n*-th generation should always appear after a person of the (*n*-1)-th generation**. Also, people of the same generation should be ordered based on their sequence in the

original list. For example, if Nick and Wendy are of the same generation, and Nick appears before Wendy in the original list, then Nick should appear before Wendy in the returned list.

**Example 1:**

```
family_tree : {}
name : 'Jane'
```
returned value: `[]`

**Example 2:**

```
family_tree : { 'Frank':['Mary','Jane'], 'Jane': ['Nick', 'Wendy'] }
name  : 'Jane'
```
returned value: `['Nick', 'Wendy']`

**Example 3:**

```
family_tree : { 'Frank':['Mary','Jane'], 'Jane': ['Nick'] }
name  : 'Frank'
```
returned value: `['Mary', 'Jane', 'Nick']`

**Example 4:**

```
family_tree  :  {'Alan': ['Bob'],
            'Bob': ['Chris', 'Eric'],
           'Eric': ['Jim'] }
name  : 'Alan'
```
returned value: `['Bob', 'Chris' , 'Eric', 'Jim']`

**Example 5:**

```
family_tree = { 'Alan'  : ['Bob', 'Eric', 'Hannah'],
    'Bob'   : ['Chris', 'Debbie'],
    'Debbie': ['Cindy'],
    'Eric'  : ['Dan', 'Fanny'],
    'Fanny' : ['George']}
name  : 'Alan'
```
returned value: `['Bob', 'Eric', 'Hannah', 'Chris', 'Debbie', 'Dan', 'Fanny',`
`'Cindy', 'George']`

8. A graph is a non-linear data structure. A graph is a network consisting of connecting nodes. A common example will be a social network (e.g. facebook or instagram). On facebook, if A is a friend of B on facebook, then B is a friend of A. If the list of friends is as follows:

```
name    friend
A       B
A       C
B       C
C       D
```

Then it can be represented by the Python's dictionary data structure as follows:

```
graph = {'A': ['B', 'C'],
         'B': ['A', 'C'],
         'C': ['A', 'B', 'D'],
         'D': ['C']}
```

However, on Instagram, if A is the follower of B, B may/may not be the follower of A. If the list of followers is as follows:

```
name    followers
A       B,C
B       C,D
C
D       A,B
```

Then it can be represented by the Python's dictionary data structure as follows:

```
graph = {'A': ['B', 'C'],
         'B': ['C', 'D'],
         'C': [],
         'D': ['A', 'B']}
```

You are given a file called “friends.txt” that stores people who are friends with each other. Each line of the file contains two people’s names separated by a tab. These two people are friends with each other.

A. Write a program that does the following. The program prompts the user for a person’s name. Call this person X. The program then prompts the user for an integer between 1 and 6. Call this number n. The program then displays all people who are within n-degrees of separation from

   **Reference**: https://en.wikipedia.org/wiki/Six_degrees_of_separation

   You can assume that X has at least one friend, i.e., X has appeared in the file “friends.txt.”

   For example, suppose the file contains the following data:

```
A       B
A       C
A       D
B       C
B       E
C       D
C       F
D       G
F       G
```

**Example #1:**

If X is E and n is 1, then the program should display only B because 1-degree separation means direct friends.

**Example #2:**

If X is E and n is 2, then the program should display A, B and C. This is because B is 1-degree separated from E, while A and C are 2-degrees separated from E.

**Example #3:**

If X is E and n is 3, then the program should display A, B, C, D and F. This is because in addition to A, B C, D and F are 3-degrees separated from E, and therefore should be added.

B. Change the program above so that when X and n are given, the program prints out those people who are exactly n-degree separated from X.

C. Write a program that prompts for two names. The program then displays how many degrees away these people are. (You can assume that these two people are definitely connected somehow.)

**Example #1:**

Given the file above, if the two people are A and E, then the program should display 2, because A is connected to B and B is connected to E. (Note that although A could also be connected to E through A via C via B via E, this is a longer path so we do not consider this path.)

**Example #2:**

If the two people are B and G, then the program should display 3.

**Reference**: https://www.python.org/doc/essays/graphs/

## MORE EXERCISES

9. **[Difficulty: *]** Copy the code shown below that shows the price of items in a mart at line 02. The code also shows a list of items along with the quantity bought by 2 customers at lines 17 and 18.

   Complete the function `compute_bill` in the file that takes in a dictionary of items purchased by the customer, and price of items and returns the amount to be paid.

```
01  price_info = {'pencil':0.80, 'pen':1.20, 'eraser':0.50 }
02
03  #complete this function
04  def compute_bill(cart, pricing):
05
06
07
08
09
10
11
12
13
14
15
16  jane_items = {'pen':10, 'eraser':2}
17  eric_items = {'pencil':12, 'eraser':5, 'pen':2}
18
19  print("Jane's bill amount $", compute_bill(jane_items, price_info))
20  print("Eric's bill amount $", compute_bill(eric_items, price_info))
```

   Expected output with the given test data:

```
Jane's bill amount $13.0
Eric's bill amount $14.500000000000002
```

10. **[Difficulty: *]** Write a program to read the file "capitals.txt". Every line in "capitals.txt" consists of 2 parts, country and capital separated by a colon (This question is adapted from the book "Starting out with Python by Tony Gaddis", Page 434, Problem 2)

    Sample of the format in the capitals.txt file is shown below

```
Singapore:Singapore
Thailand:Bangkok
…
```

    The program should quiz the user 10 times by randomly displaying the country's name and asking for the capital. The program should then keep count of the correct responses and display the result.

    **Optional:** Ensure that no two questions quizzed on are the same.

Sample output when the program is run (with the given capitals.txt file):

```
1.What is the capital of Switzerland? Bern
Correct!
2.What is the capital of North Korea? pyongyang
Correct!
3.What is the capital of Pakistan? islamnad
Incorrect

… some questions not shown for brevity

9.What is the capital of Zambia?
Incorrect
10.What is the capital of Portugal? lisbon
Correct!
You got 6 / 10 correct
```

11. **[Difficulty: **]** Write a program to read from the file called "tempest.txt". The program is supposed to read words in the file and print the number of words beginning with every unique alphabet in the .txt file.

**Note:** When you think of designing the solution for the program, think of solving simple problems that could be tested first before you can put them together. For example, you could write a function to read the file and create a word list in one function.
Test this function before you write another function that takes in a list of words and assists to count the number of words starting with every different beginning letter. You may want to use a dictionary to hold these counts!

Expected output when the program is run (with the given tempest.txt file):

```
Words beginning with b : 1
Words beginning with n : 3
Words beginning with a : 12
Words beginning with t : 11
Words beginning with i : 8
Words beginning with f : 1
Words beginning with o : 2
Words beginning with s : 7
Words beginning with g : 1
Words beginning with d : 4

… rest not shown for brevity
```

12. **[Difficulty: **]** Copy the code shown below. Write a function called `reverse_dict` in the file that takes in a dictionary and reverses the dictionary. That is, you are supposed to return a new dictionary that creates keys out of values and values out of keys.

You can assume that the input to the functions consists of a dictionary whose values are `list` objects.

```
01  #Author: your name here
02
03  #write the function
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22  dict1 = reverse_dict( {"a":[1,2,3], "b":[1,2], "c":[3,4], "d":[5,6]} )
23
24  student_subjects  =  reverse_dict({ "Jane":["Economics","Physics","Chemistry"],
25  "Mark":["Literature","Chemistry","Biology"],
26  "Sarah":["Literature","Physics","Chemistry"]} )
27
28  print(dict1)
29  print()
30  print(student_subjects)
```

Expected output when the program is run (with the given two test data):

```
{1: ['a', 'b'], 2: ['a', 'b'], 3: ['a', 'c'], 4: ['c'], 5: ['d'], 6: ['d']}

{'Economics': ['Jane'], 'Physics': ['Jane', 'Sarah'], 'Chemistry': ['Jane', 'Mark',
'Sarah'], 'Literature': ['Mark', 'Sarah'], 'Biology': ['Mark']}
```

13. **[Difficulty: ***]** You are given a file "fifa_winners.txt" that contains information of winners of the FIFA world cup since 1930. Write a program to get the top 3 winners. That is, you are required to print the names of the countries who have won the FIFA world cup the greatest number of times, along with the number of times each country has won.

    Format of the file is shown below:

    ```
    1930	Uruguay
    1934	Italy
    ```

    The first column shows the year, and the second column shows the country that won the world cup in the corresponding year. The two columns are separated by 1 tab (\t) character.

    Expected output when the program is run (with the given file):

```
Brazil 5
Italy 4
West Germany 3
```

## OUT OF SCOPE

14. The special syntax, *args and **kwargs in function definitions is used to pass a variable number of arguments to a function.

    The single asterisk form (*args) is used to pass a *non-keyworded*, variable-length argument list. For example:

```
>>> def do_x(*args):
...     print(type(args))
...
...     for value in args:
...         print(value)
...
>>> do_x(1,2,3,4)
<class 'tuple'>
1
2
3
4
```

    One good example is the print function.

```
>>> print(1)                            # takes in 1 argument
1
>>> print(1,3,'apple', 'orange')       # takes in 4 arguments
1 3 apple orange
```

    This is the method signature for the print function:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

    **Reference :** https://docs.python.org/3/library/functions.html#print

This special syntax can be used, not only in function definitions, but also when calling a function.

```
>>> def do_x(first, second, third):
...     print(first)
...     print(second)
...     print(third)
...
>>> values = [1,2,3]
>>> do_x(values[0], values[1], values[2]) # the usual way of calling it
1
2
3
>>> do_x(*values) # unpack the values in the container before sending in
1
2
3
```

```
>>> do_x(1, *[2,3]) # the total number of values must match
1
2
3

>>> values = [1,2,3,4]
>>> do_x(*values)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: do_x() takes 3 positional arguments but 4 were given
```

The double asterisk form is used to pass a *keyworded*, variable-length argument list instead of a variable-length argument list.

```
>>> def do_x(**kwargs):
...     print(type(kwargs))
...     for key,value in kwargs.items():
...         print(f'{key} = {value}')
...
>>> do_x(apple=3,orange=4,pear=5)
<class 'dict'>
apple = 3
orange = 4
pear = 5
```

Similar, this ** syntax can also be used when calling a function. It will extract the values out from the dictionary whose key (in the dictionary) matches the name of the parameter.

```
>>> def do_x(first,second,third):
...     print(first)
...     print(second)
...     print(third)
...
>>> d = {'first':3, 'second':5, 'third':6}
>>> do_x(**d)
3
5
6
>>> do_x(3,**d)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: do_x() got multiple values for argument 'first'
>>> d = {'second':5, 'third':6}
>>> do_x(3,**d)
3
5
6
```

*Note: There is no exercise for this question. :)*