Assignment 4 – PYTHON DICTIONARIES

Due: 4:30pm, May 22
Worth: 2% of your final mark
Topic: dictionaries
This assignment is marked out of 20

For assignment 4, a program containing the skeleton and testing code for the 7 Assignment 4 questions has been posted on the CompSci 101 Assignments website. Download this program from the CompSci 101 assignments website:

https://www.cs.auckland.ac.nz/courses/compsci101s1c/assignments/

You must develop the solution to each function in the Python program. Once you are happy that your function executes correctly, submit the whole function to CodeRunner:

https://coderunner2.auckland.ac.nz/moodle/

When you press the submit button in CodeRunner, you will receive immediate feedback telling you if you have passed all the tests for that question. You can submit as many times as you like. You need to submit one function at a time.

Note: A helper function
Several of the testing codes for the functions in this assignment makes use of the **`print_dict_in_key_order(a_dict)`** function which prints dictionary key:value pairs in sorted key order.

```
#---------------------
# A helper function
#---------------------
def print_dict_in_key_order(a_dict):
   all_keys = list(a_dict.keys())
   all_keys.sort()
   for key in all_keys:
      print(key, ":", a_dict[key])
```

Some of the code for some of the Assignment 4 functions may be a little complex. It is a good idea to define and use helper functions of your own to assist in making the code clear and readable. If you do define and use your own helper functions you need to copy and paste the Assignment 4 function as well as any helper functions into CodeRunner when you submit and check your answer in CodeRunner.

#Assignment 4 Questions -

```
#---------------------------------------------------
# 1111111111111111111111111111111111111111111111
# draw_histogram()
#---------------------------------------------------
#---------------------------------------------------
"""
Define the draw_histogram() function which is passed a Python dictionary as
a parameter. The keys of the dictionary are single letters and the
corresponding values are integers, e.g., {'b': 5, 'a': 6, 'c': 3}. For each
key:value pair in the dictionary the function prints a series of stars
followed by a space, followed by the key. The number of stars printed is
given by the value corresponding to the key. The keys are printed in
alphabetical order.  Note that the key is not printed if the corresponding
value is a number less than 1.

For example, the following code:

print("1.")
draw_histogram({'a': 2, 'c': 7, 'b': 5})
print()

print("2.")
draw_histogram({'a': 0, 'c': 5, 'b': 7, 'f': -1})

prints:

1.
** a
***** b
******* c

2.
******* b
***** c
"""
```

```
def draw_histogram(histogram_dict):
   pass
```

```
#-----------------------------------------------------
# 2222222222222222222222222222222222222222222222222
# get_word_len_dict()
#-----------------------------------------------------
"""
Define the get_word_len_dict() function which is passed a string of text as
a parameter.  The function returns a dictionary with keys which are integers
and corresponding values which are lists of unique words.  The list of words
corresponding to a key contains all the unique words from the text that have
a length equal to the key value.  The corresponding lists of unique words
should be in sorted alphabetical order.

Note: the testing code makes use of the print_dict_in_key_order(a_dict)
  which prints the dictionary pairs in sorted key order.

For example, the following code:

text = "May your coffee be strong and your Monday be short"
the_dict = get_word_len_dict(text)
print_dict_in_key_order(the_dict)
print()

text = 'why does someone believe you when you say there are four billion
stars but they have to check when you say the paint is wet'
the_dict = get_word_len_dict(text)
print_dict_in_key_order(the_dict)

prints:

2 : ['be']
3 : ['May', 'and']
4 : ['your']
5 : ['short']
6 : ['Monday', 'coffee', 'strong']

2 : ['is', 'to']
3 : ['are', 'but', 'say', 'the', 'wet', 'why', 'you']
4 : ['does', 'four', 'have', 'they', 'when']
5 : ['check', 'paint', 'stars', 'there']
7 : ['believe', 'billion', 'someone']
"""
```

```
def get_word_len_dict(text):
   return {}
```

```
#---------------------------------------------------
# 3333333333333333333333333333333333333333333333333
# get_text_valuation()
#---------------------------------------------------
#---------------------------------------------------
"""
Define the get_text_valuation() function which is passed two parameters, a
dictionary and a string of text.  The keys of the parameter dictionary are
single letters and the corresponding values are integers (the value of the
key letter), e.g., {'b': 5, 'a': 6, 'c': 3}.  The function returns the total
valuation (an integer) of the string of text where:

• if the letter from the text is a key letter of the dictionary then its
  value is the integer corresponding to the letter in the dictionary,
• any alphabetic characters from the text which are not in the dictionary
  are worth 1,
and,
• all non alphabetic characters are worth 0 (use the isalpha() method to
  check if a character is alphabetic or not).

Notes:
• you will need to change the text to lowercase before you work out the
  total value of the text,
• you can assume that all the keys in the dictionary are lowercase
  characters.

For example, the following code:

letter_value_dict = {"r": 2, "s": 2, "h":4, "t":3, "m": 7, "g":4, "v":8}

letters = "BLAH"
print("1.", letters, "-", get_text_valuation(letter_value_dict, letters))

letters = 'thought provoking'
print("2.", letters, "-", get_text_valuation(letter_value_dict, letters))

letters = "too much month at the end of the money"
print("3.", letters, "-", get_text_valuation(letter_value_dict, letters))

prints:

1. BLAH - 7
2. thought provoking - 40
3. too much month at the end of the money - 70
"""
def get_text_valuation(letter_worth_dict, text):
   return 0
```

```
#-----------------------------------------------------
# 444444444444444444444444444444444444444444444444
# remove_short_synonyms()
#-----------------------------------------------------
"""
Define the remove_short_synonyms() function which is passed a dictionary as
a parameter.  The keys of the parameter dictionary are words and the
corresponding values are lists of synonyms (synonyms are words which have
the same or nearly the same meaning).  The function removes all the synonyms
which have less than 7 characters from each corresponding list of synonyms.
As well, the function sorts each corresponding list of synonyms.
For example, the following code:

synonyms_dict = {'look' : ['gaze', 'see', 'glance', 'watch', 'peruse'],
   'put' : ['place', 'set', 'attach', 'keep', 'save', 'set aside', 'effect',
          'achieve', 'do', 'build'],
   'beautiful' : ['pretty', 'lovely', 'handsome', 'dazzling', 'splendid',
                'magnificent'],
   'slow' : ['unhurried', 'gradual', 'leisurely', 'late', 'behind',
           'tedious', 'slack'],
   'dangerous' : ['perilous', 'hazardous', 'uncertain']
   }
remove_short_synonyms(synonyms_dict)
print("1.")
print_dict_in_key_order(synonyms_dict)
print()

synonyms_dict = {'come' : ['approach', 'advance', 'near', 'arrive', 'reach'],
   'show' : ['display', 'exhibit', 'present', 'note', 'point to', 'indicate',
          'explain', 'reveal', 'prove', 'demonstrate', 'expose'],
   'good' : ['excellent', 'fine', 'superior', 'wonderful', 'grand', 'superb',
                                                              'edifying'],
   'bad' : ['evil', 'immoral', 'wicked', 'rotten', 'contaminated', 'spoiled',
        'defective',  'substandard', 'faulty', 'improper', 'inappropriate']
}
remove_short_synonyms(synonyms_dict)
print("2.")
print_dict_in_key_order(synonyms_dict)


prints:


1.
beautiful : ['dazzling', 'handsome', 'magnificent', 'splendid']
dangerous : ['hazardous', 'perilous', 'uncertain']
look : []
put : ['achieve', 'set aside']
slow : ['gradual', 'leisurely', 'tedious', 'unhurried']

2.
bad : ['contaminated', 'defective', 'immoral', 'improper', 'inappropriate',
'spoiled', 'substandard']
come : ['advance', 'approach']
good : ['edifying', 'excellent', 'superior', 'wonderful']
show : ['demonstrate', 'display', 'exhibit', 'explain', 'indicate', 'point
to', 'present']
"""

def remove_short_synonyms(synonyms_dict):
   pass
```

```
#-------------------------------------------------
# 55555555555555555555555555555555555555555555555
# get_triples_dict()
#-------------------------------------------------
"""
Define the get_triples_dict() function which is passed a string of text as a
parameter.  The function first converts the parameter string to lower case
and then returns a dictionary with keys which are all the unique consecutive
three alphabetic characters from the text, and the corresponding values are
the number of times the three consecutive alphabetic characters appear in
the text.  Use the isalpha() method to check if a character is alphabetic or
not.  The dictionary should only contain entries which occur more than once.
After your dictionary has been created and populated, you need to remove any
key-value pairs which have a corresponding value of 1.  For example, if the
text is "Super, duper" the algorithm proceeds as follows:

Character 's':  String is "s", Dictionary is {}
Character 'u':  String is "su", Dictionary is {}
Character 'p':  String is "sup", change string to "up", {'sup': 1}
Character 'e':  String is "upe", change string to "pe", Dictionary is
                          {'sup': 1, 'upe': 1}
Character 'r':  String is "per", change string to "er", Dictionary is
                          {'sup': 1, 'upe': 1, 'per': 1}
Character ',':  String is "er", Dictionary is {'sup': 1, 'upe': 1, 'per': 1}
Character ' ':  String is "er", Dictionary is {'sup': 1, 'upe': 1, 'per': 1}
Character 'd':  String is "erd", change string to "rd", Dictionary is
                {'sup': 1, 'upe': 1, 'per': 1, 'erd': 1}
Character 'u':  String is "rdu", change string to "du",
                {'sup': 1, 'upe': 1, 'per': 1, 'erd': 1, 'rdu': 1}
Character 'p':  String is "dup", change string to "up", Dictionary is
                {'sup': 1, 'upe': 1, 'per': 1, 'erd': 1, 'rdu': 1, 'dup': 1}
Character 'e':  String is "upe", change string to "pe", Dictionary is
                {'sup': 1, 'upe': 2, 'per': 1, 'erd': 1, 'rdu': 1, 'dup': 1}
Character 'r':  String is "per", change string to "er", Dictionary is
                {'sup': 1, 'upe': 2, 'per': 2, 'erd': 1, 'rdu': 1, 'dup': 1}
Remove all entries with a value of 1: {'upe': 2, 'per': 2}
For example, executing the following code:

print("1.")
print_dict_in_key_order(get_triples_dict('super, duper'))
print("\n2.")
print_dict_in_key_order(get_triples_dict("ABC ABC ABC"))
print("\n3.")
print_dict_in_key_order(get_triples_dict("Sometimes the smallest things make
more room in your heart"))
print("\n4.")
print_dict_in_key_order(get_triples_dict("My favourite painting is the
painting i did of my dog in that painting in my den"))

prints (output is shown here in four separate columsn):
                    1.          2.          3.          4.
                    per - 2     abc - 3     est - 2     ain - 3
                    upe - 2     bca - 2     sma - 2     epa - 2
                                cab - 2                 gin - 2
                                                        ing - 3
                                                        int - 4
                                                        myd - 2
                                                        ngi - 3
                                                        nti - 3
                                                        pai - 3
                                                        tin - 3
"""

def get_triples_dict(sentence):
    return {}
```

```
#--------------------------------------------------
# 6666666666666666666666666666666666666666666666
# contains_keys_and_values()
#--------------------------------------------------
"""
```

Define the contains_keys_and_values() function which is passed two dict objects as parameters, dict1 and dict2. The two parameter dictionaries both have corresponding values which are lists of elements (numbers or strings). The function return True if the following two conditions are met:

- dict1 contains all the keys which are in dict2 (dict1 may also contain extra keys),

and,

- the elements in all the value lists of dict2 are also elements in at least one of the value lists of dict1. **Note:** when testing this part of the condition, the elements can be in any order and in any of the value lists, e.g., if one of the values lists of dict2 is [4, 2] and any one of the value lists of dict1 contains the element 4 and any one of the value lists of dict1 contains the element 2, this part of the condition is satisfied.

The function returns False in all other cases.

For example, the following code:

```
dict1 = {}
dict2 = {}
print("1.", contains_keys_and_values(dict1, dict2))

dict1 = {"a": [4, 3] , "d": [6, 2], "z": [], "t": [2, 23]}
dict2 = {"z": [2, 3, 6, 23], "a": [4]}
print("2.", contains_keys_and_values(dict1, dict2))

dict1 = {"a": [6, 3], "p": []}
dict2 = {"a": [3, 6, 3], "p": [6, 1]}
print("3.", contains_keys_and_values(dict1, dict2))

dict1 = {"a": [6, 3], "p": []}
dict2 = {"a": [3, 6, 3], "p": ['a']}
print("4.", contains_keys_and_values(dict1, dict2))

dict1 = {"a": [6, 3], "p": ['a'], "t": ['s']}
dict2 = {"a": [3, 6, 3], "p": ['a'], "s": ['a']}
print("5.", contains_keys_and_values(dict1, dict2))
```

prints:

```
1. True
2. True
3. False
4. False
5. False
"""
```

```
def contains_keys_and_values(dict1, dict2):
    return False
```

```
#----------------------------------------------------
# 7777777777777777777777777777777777777777777777
# get_previous_words_dict()
#----------------------------------------------------
#----------------------------------------------------
"""
Define the get_previous_words_dict() function which is passed a string of
text as a parameter.  The function returns a dictionary with keys which are
all the unique words from the text, and corresponding values which are lists
of all the unique words in the text which come before the key word.  Note
that in each corresponding list of previous words, the same word should not
appear more than once.

Notes:
• the first word in the sentence will initially have the empty string as its
  previous word,
• you can assume that the text is all in lower case and contains no
  punctuation,
• each list of previous words must be sorted into ascending order,
• the testing code makes use of the print_dict_in_key_order(a_dict) which
  prints the dictionary pairs in sorted key order.

For example, the following code:

text = 'a man we saw saw a saw'
previous_words_dict = get_previous_words_dict(text)
print_dict_in_key_order(previous_words_dict)
print()

text = 'my favourite painting is the painting i did of my dog in that
painting in my den'
previous_words_dict = get_previous_words_dict(text)
print_dict_in_key_order(previous_words_dict)

prints:

a : ['', 'saw']
man : ['a']
saw : ['a', 'saw', 'we']
we : ['man']

den : ['my']
did : ['i']
dog : ['my']
favourite : ['my']
i : ['painting']
in : ['dog', 'painting']
is : ['painting']
my : ['', 'in', 'of']
of : ['did']
painting : ['favourite', 'that', 'the']
that : ['in']
the : ['is']
"""

def get_previous_words_dict(text):
    return {}
```