Assignment 4 – PYTHON DICTIONARIES

Due: 4:30pm, February 12
Worth: 3% of your final mark
Topic: dictionaries
This assignment is marked out of 30

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/compsci101ssc/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: For question 6 of the assignment you need to download the 'ShortNamesAndNums.txt' file which is available from the Assignment website (see above). Make sure that this file is inside the same folder as your A4_Testing.py program.

Note: Two helper functions
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.

The testing code for function 3 in this assignment makes use of the **`remove_less_than_2(a_dict)`** function which removes any key:value pairs which have a corresponding value of 1 from the dictionary.

```
#---------------------
# Two helper functions
#---------------------
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])

def remove_less_than_2(a_dict):
   all_keys = list(a_dict.keys())
   for key in all_keys:
      if a_dict[key] == 1:
         del a_dict[key]
```

Assignment 4 Questions

```
#---------------------------------------------------
# 1111111111111111111111111111111111111111111111111
# 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 the key, followed by ":
", followed by a series of stars. 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': 0})

prints the following:

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

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

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

```
#---------------------------------------------------
# 2222222222222222222222222222222222222222222222222
# test_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
```

```
#---------------------------------------------------
# 3333333333333333333333333333333333333333333333333
# get_last_three_letters_dict(sentence)
#---------------------------------------------------
"""
Define the get_last_three_letters_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 object which has:

    • keys which are the last three letters of any words in the
      parameter string of text with length greater than 2,
    and,
    • corresponding values which are the number of words in the
      parameter string of text which end with these last three letters.

Notes:
• the testing code removes any pairs in the resulting dictionary where the
  last three letters occur just the once in the parameter string of text.
• 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 code:

text = ' "west best worst first tapping snapping in a pest the straining
singing forest living'
a_dict = get_last_three_letters_dict(text)
remove_less_than_2(a_dict)
print("1.")
print_dict_in_key_order(a_dict)
print()

text = 'Kublai Khan does not necessarily believe everything Marco Polo says
when he describes the cities visited on his expeditions but the emperor of
the Tartars does continue listening to the young Venetian with greater
attention and curiosity than he shows any other messenger or explorer of
his'
a_dict = get_last_three_letters_dict(text)
remove_less_than_2(a_dict)
print("2.")
print_dict_in_key_order(a_dict)

prints:

1.
est : 4
ing : 5
rst : 2

2.
han : 2
his : 2
ing : 2
oes : 2
the : 4
"""

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

```
#-----------------------------------------------------
# 4444444444444444444444444444444444444444444444444
# 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 keyletter 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
```

```
#-----------------------------------------------------
# 5555555555555555555555555555555555555555555555555
# 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 {}
```

```
#---------------------------------------------------
# 6666666666666666666666666666666666666666666666666
# get_names_num_tuple_dict()
#---------------------------------------------------
"""
Define the get_names_num_tuple_dict() function which is passed a filename as
a parameter.  The file contains lines of text where each line is made up of
a name followed by a series of numbers.  An example input file is shown
below ("ShortNamesAndNums.txt"):

   Bella 5 2 6 2 2 30 4 8 9 2
   Gill 2 9 7 54 67
   Jin 3 26 51 3 344 23
   Elmo 4 3 8 6 8
   Ali 6 7 6 5 12 31 61 9

The get_names_num_tuple_dict() function returns a dictionary with the names
as the keys and corresponding values which are tuples of numbers.  The first
number after the name controls how many numbers are in the corresponding
tuple (the numbers included in the tuple are taken from the left to right).
For example, the line of text "Ali 6 7 6 5 12 31 61 9" has 6 as the first
number after the name and this line of text becomes the dictionary entry
with the keyword "Ali" and the corresponding value is a tuple made up of the
next six integers "Ali": (7, 6, 5, 12, 31, 61).  You can assume that there
are always enough numbers in each line of the input file.

Notes:
• the testing code makes use of the print_dict_in_key_order(a_dict) which
  prints the dictionary pairs in sorted key order.
• Make sure you have downloaded the 'ShortNamesAndNums.txt' file which is
  available from the Assignment website.



For example, the following code:

names_and_nums_dict = get_names_num_tuple_dict("ShortNamesAndNums.txt")
print_dict_in_key_order(names_and_nums_dict)

prints:

Ali : (7, 6, 5, 12, 31, 61)
Bella : (2, 6, 2, 2, 30)
Elmo : (3, 8, 6, 8)
Gill : (9, 7)
Jin : (26, 51, 3)
"""

def get_names_num_tuple_dict(filename):
   return {}
```

```
#---------------------------------------------------
# 7777777777777777777777777777777777777777777777777
# 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, the 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
• the testing code makes use of the print_dict_in_key_order(a_dict) which
  prints the dictionary pairs in sorted key order.
• each list of previous words must be sorted into ascending order.


For example, the following code:

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

sentence = 'my favourite painting is the painting i did of my dog in that
painting in my den'
previous_words_dict = get_previous_words_dict(sentence)
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 {}
```