**EN 811 300 Fundamentals of Computer Programming**

**Final Examination**

**Faculty of Engineering, Khon Kaen University**

**Academic Year 2562 Semester 1**

**24 November 2019, 1:00pm – 4:00pm**

**Instructions:**

1. There are 10 questions. Full scores require every problem solved.
2. This is a closed book exam.

**\* Dictionary is allowed. No other reading materials are allowed.**

1. **Network communication is allowed only for submission of the answers to the designated system.  
   \* Personal communication, social media, file sharing, or internet searching is NOT allowed.**
2. Name the file as follows:

\* Name your submission program by the corresponding problem: **Px.py**

For example, P1.py for problem 1. P2.py for problem 2, and so on.

1. Put verify.txt with the other answer files. The verify.txt has the content as handed out by the exam staff.
2. Write a main program under

if \_\_name\_\_ == '\_\_main\_\_':

1. Submit the program through the designated system.

====================================================

The designated system is **autolab.en.kku.ac.th**

====================================================

**P1.** Write a function named firstLetter to take a string argument and return the first letter of the argument.

**Example 1**, when the function is invoked:

=====================================================

r = firstLetter('This is a test.')

print(r)

=====================================================

we will see:

=====================================================

T

=====================================================

**Example 2**, when the function is invoked:

=====================================================

r = firstLetter('is this too easy?')

print(r)

=====================================================

we will see:

=====================================================

i

=====================================================

**P2.** Write a function named swop to take 3 arguments: listA (as a list), listB (as a list), and swop\_id (as an integer). The function swops items of listA and listB at the swop\_id and returns both swopped lists. The original lists must be preserved. (They must not be changed. See Example 3. Partial credit if this requirement is not met.) Assume that the swop\_id is valid for both lists. (You don’t have to worry about it.)

***Hint****: <list>.copy can be handy. Also, recall that return A, B gives both out (as we have seen in the practice session).*

**Example 1**, when the function is invoked:

=====================================================

list1 = [1, 2, 3, 4, 5, 6]

list2 = [-10, -30, -50, -60, -70]

r1, r2 = swop(list1, list2, 4)

print(r1)

print(r2)

=====================================================

we will see:

=====================================================

[1, 2, 3, 4, -70, 6]

[-10, -30, -50, -60, 5]

=====================================================

**Example 2**, when the function is invoked:

=====================================================

list1 = [1, 2, 3, 4, 5, 6]

list2 = ['a', 'b', 'c', 'd', 'e', 'f']

r1, r2 = swop(list1, list2, 2)

print(r1)

print(r2)

=====================================================

we will see:

=====================================================

[1, 2, 'c', 4, 5, 6]

['a', 'b', 3, 'd', 'e', 'f']

=====================================================

**Example 3**, when the function is invoked:

=====================================================

list1 = ["Be", "in", "the", "present", "."]

list2 = ['a', 'b', 'c', 'd', 'e', 'f']

r1, r2 = swop(list1, list2, 1)

print(r1)

print(r2)

print(list1)

print(list2)

=====================================================

we will see:

=====================================================

['Be', 'b', 'the', 'present', '.']

['a', 'in', 'c', 'd', 'e', 'f']

['Be', 'in', 'the', 'present', '.']

['a', 'b', 'c', 'd', 'e', 'f']

=====================================================

Note the original lists are preserved. (They are not changed.)

**P3.** Write a function named clear\_dict to take an argument (as a dictionary). The function goes through each key-value pair and changes each value to 0. Then it returns the cleared dictionary. The original dictionary must be preserved. (They must not be changed. See Example 2. *Hint: <dict>.copy may be handy.* Partial credit if this requirement is not met.)

**Example** 1, when the function is invoked:

=====================================================

count = {'FCP': 78, 'CEW': 81, 'ANN': 38,

'PHD': 2, 'python book': 3, 'PRML': 1}

clean\_count = clear\_dict(count)

print(clean\_count)

=====================================================

we will see:

=====================================================

{'PHD': 0, 'python book': 0, 'ANN': 0, 'PRML': 0, 'CEW': 0, 'FCP': 0}

=====================================================

Note the order may appear differently.

**Example** 2, when the function is invoked:

=====================================================

count = {'class': 3, 'other': 1, 'home': 5,

'grad': 2, 'book': 2}

clean\_count = clear\_dict(count)

print(count)

print(clean\_count)

=====================================================

we will see:

=====================================================

{'grad': 2, 'home': 5, 'book': 2, 'other': 1, 'class': 3}

{'other': 0, 'home': 0, 'book': 0, 'grad': 0, 'class': 0}

=====================================================

The original dictionary is preserved. It is not changed after calling the function. Note the order may appear differently.

**P4.** Write a function named count\_io. The function takes arguments: fname (as a string). The fname will be a file name, whose content is an integer. The function reads the content of the file as an integer, increments its value by 1, and writes back to the file. The function returns nothing.

**Example**, given the file mycount.txt content as:

|  |
| --- |
| 3 |

after the function is run:

=====================================================

count\_io('mycount.txt ')

=====================================================

the content of mycount.txt will be changed to:

|  |
| --- |
| 4 |

**P5.** Bigram is a pair of adjacent words. For example, a sentence

*“Clear writing is the logical arrangement of thought”*

has ("Clear", "writing"), ("writing", "is"), ("is", "the"), ("the", "logical"), ..., ("of", "thought") as its bigrams. Note a bigram would not count across sentences. For example, two consecutive sentences

*“Yet the truely unique feature of our language is not its ability to transmit information about men and lions”*

*“Rather it's the ability to transmit information about things that do not exist at all”*

has ("Yet", "the"), ..., ("and", "lions") and ("Rather", "it's"), ..., ("at", "all"). It does NOT have ("lions", "Rather"), because this crosses the sentences.

Write a function named bigram\_count to count the occurrences of every bigram in the file. The function takes a string argument passage\_file and returns a dict containing bigrams and corresponding counts. Use a tuple of two words for a dictionary key.

*Hint a bigram tuple is created with ('word1', 'word2') where 'word1' and 'word2' are two words in a bigram.*

**Example**, the file kubo\_rmPunct.txt has a content as:

|  |
| --- |
| What are they singing  It's beautiful  Many say it's the song about what happen when we die  How we just don't disappear  We don't disappear  Umm so what'll happen to us  Like Kubo's papers we shift  We transform so we can continue our stories the other place  The end of one's story is merely the beginning of another |

Note: the file has been cleaned up for all punctuations. So you don’t have to worry about them. Each line corresponds to each sentence. There are 9 lines, so there are 9 sentences.

Therefore, when the function is invoked:

=====================================================

r = bigram('kubo\_rmPunct.txt')

print(r)

=====================================================

we will see:

=====================================================

{('Many', 'say'): 1, ('can', 'continue'): 1, ('the', 'song'): 1, ('so', 'we'): 1, ("what'll", 'happen'): 1, ("It's", 'beautiful'): 1, ('the', 'other'): 1, ("don't", 'disappear'): 2, ('to', 'us'): 1, ('say', "it's"): 1, ('just', "don't"): 1, ('stories', 'the'): 1, ('papers', 'we'): 1, ('The', 'end'): 1, ('happen', 'when'): 1, ('We', 'transform'): 1, ('what', 'happen'): 1, ('so', "what'll"): 1, ('We', "don't"): 1, ('about', 'what'): 1, ('is', 'merely'): 1, ('we', 'can'): 1, ('Umm', 'so'): 1, ("one's", 'story'): 1, ('merely', 'the'): 1, ('transform', 'so'): 1, ('of', "one's"): 1, ('we', 'shift'): 1, ('the', 'beginning'): 1, ('story', 'is'): 1, ('they', 'singing'): 1, ('Like', "Kubo's"): 1, ('we', 'die'): 1, ('our', 'stories'): 1, ('when', 'we'): 1, ('song', 'about'): 1, ('we', 'just'): 1, ('of', 'another'): 1, ('other', 'place'): 1, ("Kubo's", 'papers'): 1, ('continue', 'our'): 1, ("it's", 'the'): 1, ('happen', 'to'): 1, ('beginning', 'of'): 1, ('What', 'are'): 1, ('end', 'of'): 1, ('How', 'we'): 1, ('are', 'they'): 1}

=====================================================

Notice that only bigram ("don't", 'disappear'), which is highlighted, appears twice. All other bigrams appear once.

*Hint: (1) break a text down to sentences; (2) break each sentence down to words (<str>.split may be handy); then each sentence has a list of words; (3) for each list, iterate through to build a key of 2 words, e.g., k = (word[i], word[i+1]); and (4) count appearance into a dict (as we have practiced in the lab session).*