Week 07 Dictionary

Part 1 Dictionary

In this exercise, please try to come out with the answer without using IDLE/Python first. Then type in the expressions into IDLE to verify your answers. The objective is for you to understand why and how they work. If there is an error, specify the error and the cause of error.

Expressions	**Output**

`a = (("A",2), ("B",3), (1,4))`
`dict_a = dict(a)`
`print(dict_a)`
`b = [[1,"A"], [(2,3),4]]`
`dict_b = dict(b)`
`print(dict_b)`	

`a = (("A",2), ("B",3), (1,4))`
`dict_a = dict(a)`
`print(dict_a)`
`print(dict_a[2])`
`b = [[1,"A"], [(2,3),4]]`
`dict_b = dict(b)`
`print(dict_b)`
`print(dict_b[(2,3)])`	

`for key in dict_b.keys():`
`  print(key)`
`for val in dict_b.values():`
`  print(val)`	

`for k,v in dict_b.items():`
`  print(k, v)`
`del dict_b[(2, 3)]`
`print(dict_b)`
`del dict_b[2]`
`print(dict_b)`	

`print(tuple(dict_a.keys()))`
`print(list(dict_a.values()))`
`dict_c = {1: {2: 3}, 4: 5}`
`print(dict_c)`	

`dict_d = dict_c.copy()`
`dict_d[4] = 9`
`dict_d[1][2] = 9`
 	
`print(dict_c)`
`del dict_c`
`print(dict_c)`	

Part 2 Anagram

An anagram is a word or a phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once. For example, "nag a ram" is an anagram for "anagram" and "eleven plus two" is an anagram for "twelve plus one". (This is a bit different from our assignment that we ignore the spaces here.)

Write a function `is_anagram(word1, word2)` that returns `True` if the two words are anagram of one another, otherwise returns `False` *by using dictionary*.

Part 3 T9

Old mobile phone only has numerical keypads where every letter is associated with a number as shown below. Number 0 is designated as the spacebar. Since writing is tedious in old mobile phone, T9 system is created as the predictive text technology for mobile phone. It works as follows:

- Every word is composed of letters.
- Every letter can be mapped to a digit.
- Given a series of digit from keypresses, we can find the expected word.

We can represent the keypad in two different ways:

1. A list such that each element is a string of characters associated with the number which is the element's index.
   This gives us: `[" ", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"]`
2. A dictionary where they keys are the characters and the values are the associated number.
   This gives us: `{"a": 2, "b": 2, ..., "z": 9, " ", 0}`

Suppose there are other alphabets and other symbols for larger numbers that are not restricted to just 10 digit symbols and 26 alphabets.

A. Write a function `to_dict(keyL)` which take in the keys as the list representation and returns the dictionary representation.
B. Write a function `to_list(keyD)` which take in the keys as the dictionary representation and returns the list representation.

From this point onwards, you may assume that both `keyL` and `keyD` are already initialized.

C. Write a function `to_nums(word)` that takes in an input string and returns an integer representing the numbers to be pressed to input the string using T9.
   For instance, `to_nums("i luv u")` returns `4058808`.
D. Write a function `to_letters(num)` that takes in a number and returns a list of all combinations of letters that can be represented by the numbers on a keypad. The combinations of letters may appear in any order.