# More Programming Practices 2:
# Recursion Problems, Set and Dictionary

[1번] Write a recursive function count_matches(some_list, value) that takes a list and a value and counts the number of elements in the list that are equal to the value.

Example:

```
>>> count_matches([0, 1, 0, 4, 2, 0], 0)
3
>>> count_matches(["a", "b", "c"], 1)
0
>>> count_matches([], "a")
0
```

Some hints: Be sure to review how a recursive function should break a list apart into the first element of the list and the rest of the list.

[2번] Write a recursive function double_each(some_list) that takes a list and returns a new list that has each element in the input list repeated twice.

The original input list must remain unchanged. For example, you may not assign new values to the original list, such some_list[i] = x.

Example:

```
>>> nums = [1, 2, 3]
>>> double_each(nums)
[1, 1, 2, 2, 3, 3]
>>> nums
[1, 2, 3]
>>> double_each([])
[]
```

[3번] Write a recursive function sums_to(nums, k) that takes a list of integers and returns True if the sum of all the elements in the list is equal to k and returns False otherwise.

Example:

```
>>> nums = [1, 2, 3]
>>> sums_to(nums, 6)
True
>>> sums_to(nums, 5)
False
>>> sums_to([], 1)
False
```

Note: You are *not* allowed to use any python sum function in any form, nor sum the list and then at the end check whether it equals k. In addition, you must write sums_to as a single recursive function: That is, you may not use a main function and a recursive helper function, (Obeying this restriction gives you the opportunity to write a very simple function definition.)

Think: What is the base case and for what value of k would sums_to return True? In the recursive call, what input to sums_to would lead towards the base case?

[4번] Write a recursive function is_reverse(string1, string2) that takes two strings and returns True if string1 is the same string as string2 except in the reverse order. It returns False otherwise. To obtain each character in a string, you can use indexing just as you would with a list. You can also get substrings using slicing.

For example,

```
>>> s = "tacos“
>>> s[0]
't'
>>> s[-1]
s‘
```

```
>>> s[1:]
'acos‘
```

```
>>> s[-1]
'taco'
```

```
>>> s[0:-1]
 'taco'
```

Note: You are *not* allowed to compare the lengths of the two input strings with each other. You may only check for empty strings. Again, this restriction leads to a simple function definition. (Remember that you have the Boolean operators and and or available. *Do not use the reverse function in any form. *

Example:

```
>>> is_reverse("abc","cba")
True
>>> is_reverse("abc","cb")
False
```

```
>>>is_reverse("abc","abc“ )
False
```

```
>>> is_reverse("","")
True
```

```
>>>is_reverse("abc","dcba")
False
```

[5번] write a function sort_repeated(L) which returns a sorted list of the repeated elements in the list L

sort_repeated([1,2,3,2,1]) ➔ [1,2]

sort_repeated([1,2,3,2,2,4]) ➔ [2]

sort_repeated(list(range(100))) ➔ [ ]

[6번] Let lst is a list of integers. Write a function make_Dict_number(lst) which returns a dictionary structure consisting of element : frequency and a function most_Frequent(lst) which returns the element having the maximum value among the frequencies.

get() 을 사용하지 않은 version과 get()을 사용한 version을 각각 만드시요

```
print (make_Dict_number([2,5,3,4,6,4,2,4,5]))  # prints { 2: 2,  3;1, 4:3, 5:2, 6:1 }
print (mostFrequent([2,5,3,4,6,4,2,4,5]))  # prints 4
```

[7번] A dictionary is an unordered mutable collection of key/value pairs. It associates a key (for example a word) with a value (for example a definition for that word). Any particular key can appear at most once in a dictionary and keys must be immutable.

While the keys in a dictionary are guaranteed to be unique, the associated values are not. In the file histogram.py,

define a function histogram(d) that takes a dictionary d as input and returns another dictionary representing a histogram of the original dictionary's values.

Specifically, the keys of the new dictionary are the unique values of the input dictionary, and the values of the new dictionary are counts of the number of times each value appears in the original dictionary.

Hint: For any dictionary d, list(d.values()) returns a list consisting of all the values in d.

You may also find it convenient to use the count method for lists:  lst.count(x) returns the number of occurrences of x in lst.

Example usage:

```
>>> letters = {1: "a", 2: "b", 3:"a"}
>>> histogram(letters)
{'b': 1, 'a': 2}
>>> letters = {1: "a", 2: "b", 3:"c"}
>>> histogram(letters)
{'b': 1, 'a': 1, 'c': 1}
>>> letters[4] = "a"
>>> letters[5] = "b"
>>> letters[6] = "a"
>>> histogram(letters)
{'b': 2, 'a': 3, 'c': 1}
```