Bonus Assignment Week 3

Important!

The idea is to solve the assignment with data structures that were presented to you in class up to this point. Therefore, it is not aloowed to use sets for this assignment.

Assignment 3.1

Write a function called has_duplicates that takes a list as input and returns True if there is any element that appears more than once, and False otherwise. It should not modify the original list.

In [2]:

```
def has_duplicates(list1):
    i=0
    for element in list1:
        i += list1.count(element)
    if i > len(list1):
        return True
    else:
        return False

#testing:
has_duplicates([0,1,2,2])
```

Out[2]:

```
True
```

Assignment 3.2

Use a dictionary to write a faster, simpler version of has_duplicates().

In [30]:

```
def has_duplicates2(list1):
    dict1 = dict.fromkeys(list1)     # Elements of list are converted into keys of 
dictionary;
                                     # duplicate elements in list will appear only 
nce in dict
                                     # because keys in dictionary are unique.
    if len(dict1) == len(list1):     # If there are no duplicates, length of dict a
nd list will be
                                     # the same.
        return False
    else:
        return True

#testing:
has_duplicates2([0,1,2,2])
```

Out[30]:

```
True
```

Assignment 3.3

Assignment [illegible]

Write a program that reads a word list from a file (*words.txt*) and prints all the sets of words that are anagrams. An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Here is an example of what the output might look like:

['deltas', 'desalt', 'lasted', 'salted', 'slated', 'staled']

['retainers', 'ternaries']

['generating', 'greatening']

['resmelts', 'smelters', 'termless']

Hint: you might want to build a dictionary that maps from a collection of letters to a list of words that can be spelled with those letters. How can you represent the collection of letters in a way that can be used as a key?

In [4]:

```
def signature(s):
    sig = list(s)                          # A word (string) is converted into a signat
re (list).
    sig.sort()                             # The letters in the list are sorted alphabe
ically.
    sig = ''.join(sig)                     # The list is converted back into a string;
                                           # and letters in this string are separated b
whitespace.
    return sig


def anagrams(filename):
    d = {}
    for line in open(filename):
        word = line.strip().lower()        # Opening the text file.
        i = signature(word)                # Each word is converted into its signature
a string).

        if i not in d:                         # If its signature is not in dictionary,
a word will be
            d[i] = [word]                  # added as value in dictionary, and its sign
ture becomes
                                           # the orresponding key.

        else:                              # If its signature is already in dictionary,
a word
            d[i].append(word)              # becomes the next value for the same key (T
is key will
                                           # have several values).

    for j in d.values():                   # If a key has several values (length of val
e > 1),
        if len(j) > 1:                     # that means it is signature of anagrams.
            print (j)


#testing:
anagrams('words.txt')
```

```
['aba', 'aab']
['acaba', 'abaca']
```