Decorate, sort, undecorate pattern

In a previous example, we read a file that contained data about Words with Friends games between me and my sister. Suppose we want to read this data again, and this time print out the scores of the games in a certain order. For instance, let's order by the difference in score (Nancy's score minus Michael's score) from largest to smallest.

This is the first three lines of our desired output:

```
Nancy: 474, Michael: 357
Nancy: 440, Michael: 366
Nancy: 361, Michael: 341
... and so on...
```

We can read the data from a file and store nicely formatted results in a list, like this:

```
f = open('wwf_scores.txt')
for i in range(3):  # skip header
    f.readline()
results = [] # a list of nicely formatted results
for line in f:
    data = line.strip().split(',')
    nancy = data[1]
    michael = data[2]
    result = "Nancy:"+nancy + ", Michael"+michael
    results.append(result)
f.close()

print "\n".join(results)
```

But how do we get that list sorted in the way we want it? The **decorate, sort, undecorate pattern** is useful here. It's useful when you have a list of data that you want ordered. It has three steps:

(1) **Decorate** Make a list of pairs where the first item in the pair is the value by which you want to sort and the second item is the data you want sorted.

(2) **Sort** Sort the list.

(3) **Undecorate** Extract the desired data from the list.

Here it is in action.

```
f = open('wwf_scores.txt')
for i in range(3):  # skip header
    f.readline()
results = [] # a list of nicely formatted results
for line in f:
    data = line.strip().split(',')
    nancy = data[1]
    michael = data[2]
    result = "Nancy:"+nancy + ", Michael"+michael
    diff = int(nancy) - int(michael)
    results.append( [diff, result] )  # DECORATE
f.close()
results.sort()      # SORT by diff, smallest to largest
results.reverse()  # now largest to smallest
for i in range(len(results)):
    results[i] = results[i][1] # UNDECORATE: drop diff, keep result
print "\n".join(results)  # print one result per line
```

Exercises

Some solutions are presented in class and also included in the moodle version of this handout. This is a sequence of exercises about anagrams.

1. Suppose there is a file named `'words.txt'` that contains a huge list of English words, one per line. Write a function that reads this file and returns the list of words sorted alphabetically.

Solution:

```
def load_word_list():
    '''
    () -> list of str
    Expects a file called words.txt to be located in same folder.
    Returns a sorted list of words read from words.txt.
    '''
    wordlist = []
    in_file = open('words.txt')
    for line in in_file:
        line = line.strip()
        line_new = line.lower()
        wordlist.append(line_new)
    in_file.close()
    wordlist.sort()
    return wordlist
```

2. Write a function that takes a word (a string) and returns the word's *signature*. The signature is a string containing the letters of word in sorted order.

Solution:

```
def signature(word):
    '''(str) -> str
    Returns signature for word which is a string
    containing the letters of word in sorted order
    >>> signature('god')
    'dgo'
    '''
    L = list(word)
    L.sort()
    return ''.join(L)
```

3. Write a function that takes a word list and returns a list of anagram groups. An anagram group is a list of words that are anagrams of one another. Hint: use a python dictionary to collect words having the same signature.

Solution:

```
def anagram_groups(words):
    '''(list of str) -> list of [list of str]
    Given a list of words, groups together words
    into anagram groups and returns a list of lists
    where each sublist a group of words that are
    anagrams of one another.
    >>> words = ['ate', 'banana', 'dog', 'eat', 'god', 'tea']
    >>> anagram_groups(words)
    [['banana'], ['ate', 'eat', 'tea'], ['dog', 'god']]
    '''
    sig_to_words = {}
    for word in words:
        sig = signature(word)   # see previous exercise
        if sig not in sig_to_words:
            sig_to_words[sig] = []
        sig_to_words[sig].append(word)
    return sig_to_words.values()  # we only want the word groups
```

For the remaining exercises, suppose we have a list called `ana_groups`. It is a list of lists. Each sublist is a group of words that are anagrams. For example:

```
ana_groups = [['ab', 'ba'], ['ate', 'eat', 'tea'], ['east', 'seat'],  ['banana'] ]
```

4. Write a function that takes a list of anagram groups and a number `k` and prints the top *k* largest anagram groups. Size is measured in terms of number of words in the group. Use the DSU pattern!

Solution:

```
def print_largest_anagrams(anagram_lists, k):
    '''(list of [list of str], int) -> NoneType
    Expects a list of anagram groups.  Prints out the
    top k largest anagrams.
    '''

    # DECORATE
    top_k = []
    for group in anagram_lists:
        size = len(group)
        top_k.append( [size, group] ) # group decorated by size of group

    # SORT
    top_k.sort()    # from smallest to largest group
    top_k.reverse() # reverse that


    # UNDECORATE
    for data in top_k[:k]:
        size = data[0]
        group = data[1]  # we only care about the group at this point
        print ','.join(group)


words = [['seat', 'east'], ['ate', 'eat',  'tea'], ['banana']]
print_largest_anagrams(words, 2)
```

5. Write a function that takes a list of anagram groups and a number `k` and prints the top *k* longest anagrams. Length is measured in terms of number of characters in the anagram. Use the DSU pattern!

Solution:

```
def print_longest_anagrams(anagram_lists, k):
    '''(list of [list of str], int) -> NoneType
    Expects a list of anagram groups.  Prints out the
    top k longest anagrams.
    '''
```

```
    # DECORATE
    top_k = []
    for group in anagram_lists:
        if len(group) > 1:      # must have at least two words to be interesting!
            length = len(group)
            top_k.append( [length, group] ) # group decorated by length of words

    # SORT
    top_k.sort()    # from smallest to largest group
    top_k.reverse() # reverse that


    # UNDECORATE
    for data in top_k[:k]:
        length = data[0]
        group = data[1]  # we only care about the group at this point
        print ','.join(group)


words = [['seat', 'east'], ['ate', 'eat',  'tea'], ['banana']]
print_longest_anagrams(words, 2)
```