KIT103/KMA155 Practical 11: More Permutations & Combinations

This week you'll predict, count and generate a combination of different collections.

Preparation

Download the text file `br31.txt` from the *Labs/Scripts and additional files* area on MyLO. You will use it in the second task.

Next, start Spyder and save the starting .py file so you have somewhere to save your work from today. Add the following to your file so that the counting functions from SciPy and generating functions from the `itertools` module are available (you probably won't use all of the imported functions below).

```
from itertools import combinations, permutations, product
from scipy.misc import comb, factorial
```

1 Permutations with repeated elements

Last week you counted and generated strings in which characters could appear any number of times. Such strings represent *permutations with replacement* because each time a character is selected to be added to the permutation it is immediately returned to the set of characters that can be used later. Permutations with *repeated elements* are permutations in which the same value can be used only a fixed number of times.

The `permutations` function from `itertools` guarantees that its results are unique (it won't generate the same permutation twice) *but* it identifies items by their position in the given collection of items, so while it can be used to generate permutations with repeated elements by having the same value appear more than once in the items list, it will also repeat some permutations in its output. The task will be to deal with this by collecting generated permutations in a set.

Task: A librarian has 10 books to be sorted on a shelf: 1 copy of book 'A', 3 copies of book 'B', 4 copies of book 'C' and and 2 copies of book 'D'. As far as the librarian and users of the library are concerned, each copy of a particular book is equivalent to any other copy. Without applying any external notion of sorting order, what are the possible permutations of the books?

1. How many permutations are possible? Write down the expression on paper.
2. Write some code to evaluate that expression.
3. Define a string containing the book 'titles', including the repeated elements: `'ABBBCCCCDD'`, then write a *list* comprehension to generate all possible permutations. Make sure you assign its result to a variable. **Call `len()` on that list to see how many results `permutations` produced. Is it what you expected?**
4. Modify your code to be a *set* comprehension and test the size of the generated set. **Was the time to evaluate the set comprehension any different to the list comprehension?**

2 C(31, 2) flavours

Task: Baskin-Robbins reputedly sell 31 flavours of ice cream, although the true number is higher than that. Assuming that they did only have 31 flavours on offer, how many different two-scoop ice creams are possible?

1. What is *n*, what is *r* in this case? What's the expression that will calculate the number of alternatives?
2. Use Python to evaluate that expression.
3. Use the following code to load the list of the 31 original Baskin-Robbins flavours.

   ```
   flavours = [ l.rstrip() for l in open('br31.txt') ]
   ```

4. Then, using the lecture notes as a guide, write a set comprehension that uses `combinations` to generate all the alternatives. (Remember to assign the result to a variable.) **Check the size is what you predicted.**

Task: Baskin-Robbins' double-scoop ice creams place one scoop on top of another, so you have to eat most or all of the scoop on top before getting to the second one. Assuming the order in which you get to each flavour matters to you, how many alternative options are there?

1. What's the expression that will give the number of alternatives?
2. Write some code to evaluate that expression.
3. Write a set comprehension to generate all the alternatives. **Is it larger or smaller than before? By how much is it larger or smaller?**

3 (optional) Challenge exercise: Recursively generating combinations

If you'd like a challenge, try implementing the general recursive approach to generating combinations given in the lecture. Note that, as with last week's challenge exercise, if you try to implement this then you should use `result.add( set(p) )` to take a copy of the combination you have constructed, since the same set object is modified as your recursive function proceeds.

4 Still have more time? Work on your final assignment

If you get through the above quickly then use the remaining time to work on the fifth programming assignment. You can ask your tutor questions. While they won't be able to give you the answers to the assignment questions they should be able to nudge you in the right direction.

Useful references

Python's `combinations` function
    https://docs.python.org/3/library/itertools.html#itertools.combinations