KIT103/KMA155 Programming Assignment 5: Permutations and Combinations

Enter your answers to these questions in the accompanying Python script file `kit103_assign5.py`. Questions 1–3 require only short pieces of code to calculate or generate the answers, so replace the entries that say `None` with your answers to each part. Question 4 has you completing the implementation of two functions. Question 4 is also a little more challenging than the other parts, but is weighted the same as Questions 1 and 2 so you can demonstrate your understanding of the more fundamental skills (and get rewarded for it) and then boost your mark further by stretching yourself. Include your name and student ID in the places indicated in the header comment.

Submit your completed script file to the *Programming Assignment 5 (Counting)* assignment folder on MyLO by **1500 (3pm) Wednesday 18 October 2017**.

For questions 1–3, a helper function, `fact(n)`, has been included that provides a simpler interface to SciPy's `factorial` function. Note that the `comb` function will produce real numbers as output instead of integers. The assessment scheme will accept either, as long as the value is correct.

Test your solutions thoroughly. Your submission is expected to run without failure (even if it doesn't produce the correct answer for each question). If we have to correct your submission in order for it to run then the maximum total mark you can receive will be 3/5 (1.5/2.5 if in KMA155).

KMA155 students will be assessed primarily on Questions 1a–b, 2a–d and 3a–b (which are also marked with an asterisk (*)), but are encouraged to attempt all questions if they wish.

Hints for checking answers in which you generate values:

- While developing your code, **store the result** in a variable so that Spyder doesn't immediately try to display it (since it may be large).
- **Use the `len()` function** to check that your result set is the expected size.
- If your result, stored in set `s`, is (necessarily) large, then use code like `sorted(s)[:20]` to look at only the first 20 elements of `s`. The use of `sorted()` will often make it easier for you to confirm that it is generating the right entries.
- If your script file takes more than ~3 seconds to run then you've probably implemented one of the set comprehensions incorrectly.

Question 1: Permutations (1 mark)

a. (*) How many strings of length three are possible using characters from 'AUGC' (characters may be reused)? Write Python code to *calculate the number*.
b. (*) Write a set comprehension to *generate those strings*.
c. How many three-word sentences can be created from the words 'cat', 'sat', 'hat', 'mat', and 'pat', using each word at most once? (Most of the sentences won't make sense.) Write Python code to *calculate the number*.
d. Write a set comprehension to *generate tuples* (not single strings) of the different arrangements described in part (c).

Question 2: Combinations (1 mark, *)

You should use the definitions of toy droid names and Star Trek species provided in the assignment script in your answers.

a. (*) How many different combinations of two distinct colours are possible given the set of colour names { 'red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet' }? Write Python code to *calculate the number*. Do not directly call `factorial` or `fact` in your solution.
b. (*) Given the list of colour names defined in the script file, write a set comprehension to *generate those combinations*. (The answer will be a set of 2-tuples, or pairs.)
c. (*) How many different groups of three students can be formed from a class of 12? (Assume that only a single group is to be formed.) Write Python code to *calculate the number*.
d. (*) Assuming the students are named 'Student *k*' for $k \in \{1, 2, ..., 12\}$, write a set comprehension to *generate the possible compositions* of a single group.

Question 3: You choose which (2 marks)

The following information is used in all parts of this question. Jodie has 18 books in her collection at home. They all have short titles: 'A', 'B', 'C', ..., 'R' (there is a list of these in the assignment script).

a. (*) Jodie is going on holiday, but can only take five books with her. What are her options? Write a *set comprehension to generate them all*.
b. (*) All of Jodie's books fit on a single shelf. How many different ways can she arrange them? Write Python code to *calculate the number*.
c. Jodie sometimes likes to reread books, and may even reread a book she has just finished. What are the possible sequences of book titles of the last *three* books she has read? Write a set comprehension to *generate the alternatives*.

   > **Do not** attempt to display your answer. Check it in other ways.

d. Write Python code to calculate *how many* such sequences there are. (Do not use the `len` function on your previous answer; doing so will receive 0 marks.)

Question 4: Lovers of anagrams know *words* are mightier than the *sword* (1 mark)

An anagram is a word whose letters, when rearranged, form another word. They are a common feature of cryptic crosswords, but you do not need to know how to solve a cryptic crossword to solve this question. An anagram solver finds the alternative arrangements of a word's letters that represent valid words. **Your task** is to write the two components of a basic anagram solver: the first part generates the set of all possible permutations of the letters in a given word; the second part restricts this set to contain only those permutations that are valid words.

There are two stub functions for you to complete:

1. `word_perms(word)`: should generate and **`return`** all permutations of the letters in the string represented by `word` (each permutation will be a single string, *not* a tuple). It will probably be one line of code.
2. `anagrams(word)`: should **`return`** a set of all the permutations of the letters in `word` that are valid (that is, that appear in one of the collections of words described below), including the original word. The input can be assumed to be in lower case.
   Not all solutions to this task will receive full marks; an ideal solution will look very short, but still be readable.

The assignment script file contains code to load collections of words of different lengths (between 2 and 10) into two Python `dict`ionaries, one that holds the words in sorted lists and another that holds them in sets. To use the *list* of words of length *n*, use `word_lists[n]`, whereas to use the *set* of words of length *n*, use `word_sets[n]` (you will probably not have a variable called `n` but an expression instead). You do not necessarily have to use both of these. Select whichever you believe is best-suited to the task.

In order for these words to be loaded in your program you must download and unzip the file containing the words, `words.2-10.txt`, that accompanies this document on MyLO. *Do not submit this file with your assignment; we already have a copy.*

Warning: The list of words has only been filtered to remove entries that do not contain ordinary letters 'a' to 'z', so offensive words will be present. The list also contains many proper nouns and acronyms, so you might not recognise every output from your anagram solver.

Testing your anagram solver

To test `word_perms`, call it with very short words for which you can enumerate all the possible permutations. To test `anagrams` try it on the word 'lovers'. Other good words you could try include 'alerts', 'heart', 'parsec' and 'spare'.
