KIT103/KMA155 Programming Assignment 5: Permutations and Combinations

Enter your answers to these questions in the accompanying Python script file `programming5.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 16 October 2019**.

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.

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.

Note: If a question asks you to *calculate* a value then *do not* generate all the alternatives and count them. (Such an answer will receive zero.) Implement the appropriate mathematical expression to calculate the number.

Question 1: Permutations (1 mark)

a. How many strings of length three are possible using characters from 'AUGC', given that 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 phrases can be created from the words 'hop', 'pot', 'top', 'cop', 'lop' and 'mop', using each word at most once? (Most of the phrases 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. With the final episode of *Star Wars* coming to cinemas later this year toy companies are keen to invent new toys for children to collect. They have a series of 16 *droids* (*Star Wars* robots) and sell them in mixed packs of four (no pack contains a duplicate droid). How many distinct packs do they sell? Write Python code to *calculate the number*.
b. Given the droid names defined in the script file, write a set comprehension to *generate all compositions* of packs of four droids.
c. The various *Star Trek* television and film series have introduced a large number of different (humanoid) alien species. One of the common narrative devices is to say that a character's parents are from two different alien species, and hence that character has a blend of characteristics from the two (yes, science fiction can get away with being both horribly racist and ignorant of biology). For example, the popular character Spock is described as having a Human mother and Vulcan father. **Your task:** Assuming characters can only have two parents, and ignoring which parent is which species, how many different combinations of such 'blended' *Star Trek* characters are possible given the subset of *Star Trek* species { 'Andoran', 'Bajoran', 'Cardassian', 'Ferenghi', 'Human', 'Klingon', 'Romulan', 'Vulcan' }? Write Python code to *calculate the number*. Do not directly call `factorial` or `fact` in your solution.
d. Write a set comprehension to *generate those combinations*. (The answer will be a set of 2-tuples, or pairs.)

Question 3: You choose which (2 marks)

The following information is used in all parts of this question. Sally owns 17 DVDs (no Blu-ray discs), which she has systematically relabelled using letters of the alphabet: 'A', 'B', 'C', ..., 'Q' (there is a list of these in the assignment script).

a. Sally's collection fits neatly onto a single shelf. How many different ways can she arrange the DVDs? Write Python code to *calculate the number*.
b. Sally has been invited to a movie night hosted by her friend Larry and would like to take four DVDs from her collection with her (to have alternatives for the party, they won't actually watch them all). What are her options? Write a *set comprehension to generate them all*.
c. With so few DVDs in her collection it's not surprising that Sally will re-watch movies she's seen before, and may even watch the same movie again immediately after finishing it. What are the possible sequences of the last *three* DVDs she's watched? 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: Do words sow rows or swords? (1 mark)

An anagram is a word whose letters, when rearranged, form another word. A 'sub-anagram' is a word that is formed from a subset of letters in the original word (e.g., 'word' and 'do' are sub-anagrams of 'sword'). Sub-anagrams—and anagrams—are common in many word-based puzzles (of which there are many). You don't need to know how to play those puzzles to solve this question though. 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 an anagram solver that allows the user to specify the length *r* of sub-anagrams they're looking for: the first part generates the set of all possible *r*-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. `string_permutations(word, length)`: should generate and **`return`** all *length*-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. You can assume `length` will always be a positive number no larger than `len(word)`.
2. `subanagrams(word, length)`: should **`return`** a set of all the *length*-permutations of the letters in `word` that are valid (that is, that appear in one of the collections of words described below). The input can be assumed to be in lower case and `length` will always be between 2 and the length of `word`. You do not need to include any error handling.

   Not all solutions to this task will receive full marks; an ideal solution will look very short, but still be readable, and will run in reasonable time.

The assignment script file contains code to load collections of words of different lengths (between 2 and 10) into two Python `dictionaries`, 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, `dictionary2-10.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 `string_permutations`, call it with very short words (or smaller values of length), which will allow you to enumerate all the possible permutations yourself. To test `subanagrams` try it on the inputs 'lovers' with lengths 6, 5 and 2 (which produce sets of size 2, 20 and 30 elements, respectively). Other good words you could try include 'alerts', 'heart', 'parsec' and 'spare'. Test initially with length set to the length of the word, then try smaller values (which will produce larger sets of sub-anagrams).

How your assignment is assessed

Your submitted Python script will be assessed initially by a Python program that will execute your code and check the values generated by your answers against the expected results. This produces a tab-delimited text file named AssessmentReport.txt with columns containing the *expected* result, the *actual* result produced by your code, the *score* achieved for that question (and *maximum* score possible), and any *feedback* explaining the score.

Next, a human assessor will inspect your submission and adjust the computer-based assessment accordingly: code that produced the wrong value but is close to correct will have some marks restored;

code that produced the correct answer but did so incorrectly may have marks reduced. The assessor will add further feedback to the assessment report text file as needed.

You can download the Assessment Report from MyLO: go to Assignments then click feedback link in the Evaluation Status column. View it by opening in Excel or Numbers.

Assessment criteria

Submissions that require modification in order to run without error will receive a maximum mark of 3/5.

For Questions 1–3, full marks will be awarded to answers (Python expressions) that:

- produce the correct value;
- use an appropriate approach to producing that value; and
- are written using the technique specified in the question

Partial marks are awarded for answers that meet only some of these criteria. Answers that do not use the technique specified by the question may receive 0 as they do not demonstrate the required skill.

For Question 4: full marks will be awarded to implementations that:

- generate strings that are *r*-permutations of the input word;
- work correctly for any valid inputs;
- run efficiently; and
- are compact but readable.

Partial marks are awarded for answers that meet only some of these criteria. Answers that fail to meet some critical functionality criteria may be awarded zero.

Working independently: This is an individual assignment so your submission should be substantially your own work.