KIT103/KMA155 Practical 03: Set combinations & bitsets

This week we'll partition a set of words, generate novel colour name combinations and explore how bitsets can be used to represent sets.

Preparation

In the Labs section of MyLO download the file **bitsets.py**. It contains some helper functions for checking your answers in a task below. Also download the file **tlw.txt**, which is a collection of lower case three-letter words taken from the dictionary of words found in all Linux installations (this will save you creating a list of words to partition).

Open bitsets.py in Spyder and run it. This isn't absolutely necessary until later, but it also sets the current directory (folder) to the location of both bitsets.py and tlw.txt, making the next task much easier. (Otherwise you'll need to change the directory manually.)

1 Partitions and products

1.1 How many words in A–K?

If IPython's current directory is the location of `tlw.txt` then you can use the following to read all its entries into a set:

```
words = { line.rstrip() for line in open('tlw.txt') }
```

There are shorter ways of loading the lines from a file, but this removes the trailing new line characters from each line. You can now use `len(words)` to see how many words you just loaded.

Task: Try the following. *What do you think the sets `ak` and `lz` represent? How big are they?*

```
ak = { w for w in words if w < 'l' }
lz = { w for w in words if w >= 'l' }
```

Have we partitioned the set of words? How would you check?

1.2 Colour mixes

Consider the set `colours = { 'red', 'green', 'blue' }` and imagine we'd like to generate unusual colour names from combinations of these three colours (the names we create may not—probably won't!—correspond to actual colours).

Task: Write a set comprehension that generates *pairs* of colours from this set. That is, `colours` × `colours`. *Before you execute it, how many elements do you expect this set of pairs to have? After you execute it: were you right?*

Task: Refine the set comprehension so that it eliminates (or rather, does not generate) pairs made up of only one colour, like (red, red). *How many elements do you expect this set to have? Is the result still a Cartesian product?*

1.3 (optional) Deck of cards

Task: A standard deck of playing cards contains four suits (diamonds, hearts, clubs and spades), each with 13 ranks (Ace, King, Queen, Jack, 10, 9, ..., 2). Write a set comprehension to generate a deck of cards, with rank followed by suit. The high ranks, plus Ace, should be strings (feel free to abbreviate their names), while the numerical ranks should be integers.

2 Compact representations (bitsets)

Define a universe $\mathcal{U}$ (or just `u` in Python) to be the set `{ 'cat', 'dog', 'bird', 'elk', 'ant' }`.

Also define these two subsets:

```
a = { 'bird', 'dog' }
b = { 'cat', 'bird', 'elk' }
```

In the following, assume that the elements of the set `u` will be sorted alphabetically to determine the location of each element's bit in a bitset representation.

Task: Write the bitset representations of `a` and `b` as a 5-bit long binary literals (a binary literal starts with `0b` and is followed by the bits that make up the number). Store these in variables for use in the next task.

> **Tip:** You can use the `set2bits` function to verify that you have done the conversion correctly. (Note that the bitsets' values will be shown as decimal numbers, not binary, when you compare your answers with the output of `set2bits`. We considered including a function to display them as binary strings, but this would make the final question in the first assignment too easy!)

Task: Perform the following set operations on the bitsets you just defined. Write down what you expect the answer to be (in its binary representation) before you write the code. Check your answers using the `bits2set` function.

1. a $\cup$ b
2. a $\cap$ b
3. a $\setminus$ b
4. a $\oplus$ b

3 (optional) Challenge exercise

If you have time and interest, write a (recursive) function that generates *n*-tuples from a given set. For instance, given a set A and value of n = 5 it would generate $A \times A \times A \times A \times A$. Note that there already exists a quite flexible function for this purpose in Python, but this isn't present in many other languages.

Write another recursive function that accepts a list of sets and generates the Cartesian product of those sets. You may find information on default argument values useful, so that the first call to your recursive method doesn't need to include the index of the set you are currently working on.

Notes

product()

The Python Standard Library includes a flexible function for generating powers of sets (the Cartesian product of a set with itself two or more times) or the more general Cartesian product of different sets: `product`. It is part of the `itertools` module:
https://docs.python.org/3/library/itertools.html#itertools.product

In the future, if you're coding in Python and need to enumerate an entire Cartesian product, then use the `product` function. If you need to only enumerate a part of the (potentially large) Cartesian product, then you will need to use the skills developed above.

Getting and setting the current working directory

To read from or write to a file you can either give the full path to its location or change the *current working directory*. To find out what the current working directory is, use the following:

```
import os
os.getcwd()
```

To change the current directory use the following:

```
os.chdir(r'C:\path\to\your\directory')
```

The r in front of the string means you can use Windows-style directory separators without having to double them up, as normally a \ inside a string indicates an escape sequence. For instance, `'\n'` is a string that contains only the new line character, while `'\t'` is a string containing just the tab character.