Exercises

Set 7

DM857 Introduction to Programming
DS830 Introduction to Programming

1 Collections

1. Write a function called `tuple_avgs` to calculate the average value of the numbers in a given tuple of tuples
2. Write a function called `find_dups` that takes a list of integers as its input argument and returns a set of those integers occurring two or more times in the list.
3. Write a function to convert a given list of tuples to a list of lists.
4. Two words are anagrams if you can transform one into the other by swapping some letters; for example, "lasted" and "salted".
   (a) Write a function `keyword(word : str) -> str` that returns a unique key-value for each word, the key must be unique for all the words in an anagram set.
   Hint use the `sorted` built-in function.
   (b) Write a function `insert_anagrams(words : list, anagram_dict: dict) -> None` that includes new words in the list of their anagrams
      i. Define the correct type of dictionary and use it to store the results of your function.
      ii. Evaluate the complexity of your code.
   (c) Write an helper function `return_anagrams(word : str) -> list` that returns the list of anagrams of the words in your dictionary.

For each of the above, you must have both docstring and doctest implemented.
Exercise 4 is a prerequisite to solve Section 4.

2 Working with text files

This section contains some warm-up exercises for Section 3, 4 and 5.

1. Write the following functions for collecting basic statistics about a text file.
   (a) `lc(reader : TextIO) -> int` that reads an open file until the end and returns the number of lines in it.
   (b) `wc(reader : TextIO) -> int` that reads an open file until the end and returns the number of words in it.
   (c) `cc(reader : TextIO) -> int` that reads an open file until the end and returns the number of characters in it.
2. Write a function `copy_lowercase(reader:TextIO, writer:TextIO)` that copies one file into another while converting its content to lowercase.

3. Write a function `erase_comments(reader:TextIO, writer:TextIO)` that copies one file into another while removing any line that starts with `'#'`.

Test your solutions with some text file (be careful when writing) or using `StringIO` from module `io` (to avoid overwriting some important file).

3 Working with CSV files

Comma Separated Values (csv) refers to a common text representation of tabular data where each row is represented as a line of text where the values are listed separated by a comma (or some other predefined separator). Optionally, the first line is used to encode the header. Below is an example of a small dataset (the population of the major Danish cities for 2015,2016 and 2017) in csv format.

```
City,2017,2016,2015
Copenhagen,1295686,1280371,1263698
Aarhus,269022,264716,261570
Odense,176683,175245,173814
Aalborg,113417,112194,110495
```

1. Write a function

   ```
   read_csv(reader:TextIO, skip_header:bool = True,
            separator:str = ',') -> List[List[str]]
   ```

   that reads a file in csv format (optionally skipping the first row) and returns a table represented as a nested list (a list of rows). For instance, calling `read_csv` on the data above must return the following list.

   ```
   [['Copenhagen', '1295686', '1280371', '1263698'],
    ['Aarhus', '269022', '264716', '261570'],
    ['Odense', '176683', '175245', '173814'],
    ['Aalborg', '113417', '112194', '110495']]
   ```

2. Write a function

   ```
   write_csv(writer:TextIO, data:List[list],
             header:Optional[List[str]] = None, separator:str = ',')
   ```

   that writes tabular data represented as a nested list `data` in csv format.

3. Write a function

   ```
   read_csv_cols(reader:TextIO, cols:List[int], skip_header:bool = True,
                 separator:str=',') -> List[List[str]]
   ```

   that reads from a csv files only the values in columns listed in `cols` (without first creating the full table like `read_csv`). For instance, calling `read_csv_cols` on the data above and `cols=[0,-1]` must return the following list.

   ```
   [['Copenhagen', '1263698'],
    ['Aarhus', '261570'],
    ['Odense', '173814'],
    ['Aalborg', '110495']]
   ```

4. Download the dataset for the COVID vaccination campaign in Denmark from itsLearning (or follow `https://bit.ly/3xdn2sx` for an up to date version by Our World in Data). This dataset records (among other things) the number of vaccinated people (column, `people_vaccinated`) per day of the campaign (column, `date`).

   (a) Using python and your solutions to the previous exercises, copy the columns `data` and `people_vaccinated` of this dataset into a new file.

   (b) Module `matplotlib.pyplot` offers a simple yet powerful way for creating plots and charts on your screen. The snippet below illustrates how to create and display a simple line plot using this module.

   ```
   import matplotlib.pyplot as plt
   ys = [ 1,  2,  0,  1 ]
   plt.plot(ys)  # plot ys as a line
   plt.show()    # raise the chart window and wait until it is closed
   ```

   Modify this snipped to plot the number of vaccinated people over the last 30 days in the dataset.

4 Metathesis Pair

Two words form a "metathesis pair" if you can transform one into the other by swapping exactly two letters; for example, "converse" and "conserve".

1. Download the text file "anagrams.txt" from itslearning and write a function that reads the file entries skipping comments and blank entries.

2. Write a program that finds all of the metathesis pairs in the dataset.
   Store the results in a dictionary.
   Hint: don't test all pairs of words, and don't test all possible swaps.

3. Evaluate the complexity of your code, as function of the number of words.

4. Evaluate the complexity of your code, as function of the length of the words.

5 Caesar Ciphers

Caesar ciphers form an encryption technique in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on.

```
Plaintext:  THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD
```

The method is named after Julius Caesar, who used it in his private correspondence.

The encryption can also be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, $A \mapsto 0$, $B \mapsto 1$, $\dots$, $Z \mapsto 25$. Encryption and decription of a letter c by a shift key can be described mathematically as

$$E(c, key) = (c + key) \mod 26 \qquad \text{and} \qquad D(c, key) = (c - key) \mod 26$$

Letter	Frequency
E	16.09%
R	7.63%
N	7.32%

(a)

Letter	Frequency
E	12.60%
T	9.37%
A	8.34%

(b)

Table 1: Most frequent letters in Danish (left) and English (right).

5.1 Ciphering and deciphering

In Python, we can convert characters into their numerical representations and back using the build-in functions `ord(c : str) -> int` and `chr(c : int) -> str`. Capital letters of the English alphabet are mapped to the interval 65–90 respecting alphabetical order, so `ord('A')` returns `65`, `ord('B')` `66` and so on; lowercase letters are mapped to the interval 97–122.[1]

1. Write a function `cipher_text(text:str, key:int) -> str` that takes a string of text and returns the string obtained ciphering it with the Caesar cipher for the given. The function must preserve case (capital letters are replaced with capital letters). Characters that are not letters are not replaced. Then write a function `decipher_text(text:str, key:int)->str` for deciphering a string ciphered with the Caesar cipher and the given key.

2. Write a function `cipher_file(input_file:TextIO, output_file:TextIO, key:int)` and a function `decipher_file(input_file:TextIO, output_file:TextIO, key:int)` for ciphering and deciphering a file.

5.2 Breaking the cipher

Caesar ciphers (and shift ciphers in general) are easily broken using some basic information and a technique called frequency analysis (besides brute-forcing the 25 possible shifts for the English alphabet). This is because Caesar cipher preserve the frequency of letters: if A is the most frequent letter in the plain text and ciphering shifts right by 3 then D will be the most frequent letter. For texts of even few sentences of natural language, letter frequency tends to fall under typical values. For English and Danish, the most frequent letter is likely E. Thus, if the plain text is a text in English or Danish and we observe that the most frequent letter is say K we can guess that the shift is such to replace E with K.

1. Write a function `letter_frequency(reader : TextIO) -> Dict[str,int]` that returns a dictionary containing for each letter of the English alphabet its frequency in the given source. The function must be case insensitive (so `'A'` and `'a'` both increment the counter for the letter "a") and handle large files (read the file in chunks or by line).

2. Module `matplotlib.pyplot` offers a simple yet powerful way for creating plots and charts on your screen. The snippet below illustrates how to create and display a simple bar chart using this module.

```
import matplotlib.pyplot as plt
xs = ['a','b','c','d']
```

[1] This mapping is defined by the American Standard Code for Information Interchange (ASCII), the de facto standard for character encoding since the '70s which later was extended into Unicode.

```
ys = [ 1,  3,  1,  2 ]
plt.bar(xs,ys)    # bar chart for ys against xs
plt.show()        # raise the chart window and wait until it is closed
```

Starting from this snippet, write a function `plot_frequencies(fr : Dict[str,int])` that shows letter frequencies as a bar chart. (Alternative: a version that also sorts bards by height.)

3. Write a function `max_vals(Dict[str,int],k=1) -> List[str]` that takes a dictionary and returns a list of the `k` keys with the largest values in decreasing order (so the key with the highest value is first).

4. Write a program that performs this attack automatically on a given text file.