Advanced Python: Homework set 2

2023/2024

Each problem is worth 2 points. You only have to do three problems.

Problem 1

During general elections to parliament or local government, seats are divided between electoral committees using the d'Hondt method. Program a function that takes as arguments the election result in the form of the number of votes cast for individual committees and the number of seats to be filled; and the returned result is the election result, i.e., the number of seats allocated to individual committees. We assume that

- details of the d'Hondt method are as described in `https://en.wikipedia.org/wiki/D%27Hondt_method`
- the electoral threshold is 5% (we do not include electoral committees of national or ethnic minorities, which are not subject to the electoral threshold).

Problem 2

Program `sudan(n, x, y)` in Python to evaluate the following recursive function:

$$\begin{aligned}
&F_0(x,y) = x + y\\
&F_{n+1}(x,0) = x, x \geq 0\\
&F_{n+1}(x,y+1) = F_n\left(F_{n+1}(x,y), F_{n+1}(x,y) + y + 1\right)
\end{aligned}$$

Since this function grows very quickly, you need to be careful not to test for $n > 2$. To speed up the operation of this function, program your implementation to store already calculated results; this technique is called memoization or memorization.

Experimentally check for which largest arguments it makes sense to call this function in the version without storage and for which in the version with storage. Post the results in a comment in the source file.

Problem 3

Each natural language has its own characteristic letter frequency statistics. Write two functions: one that calculates statistics for at least three languages based on literary works; and the second one, which will indicate the most probable language in which the text was written for the given text. It can be assumed that we only take into account languages that use an alphabet derived from the Latin alphabet. Texts should be saved in files in plain text format.

A good source of texts in various languages is, for example, Project Gutenberg or Wikisource.

Problem 4

Too complicated sentences can be a nuisance to the reader. This is why we will simplify it as follows:

- first, we remove too long words
- and then we remove words randomly if the sentence has too many of them.

Program the appropriate function `simplify_sentence(text, dl_word, word_number)`, where `dl_word` is the maximum allowed word length, `word_number` is the largest number of words that can be included in a sentence.

For example

```
text = "Periclinal division of fusiform initials \
the cambium is characterized by a division wall initiated
in the maximum plane."
simplify_sentence(text, 10, 5)
```

it should return something like this

```
The division cambium initiated wall.
```

Test the operation of your program for some popular literary work legally available on the Internet. In the source file, include the code that downloads such text or post a link to such text in a comment.

Problem 5

Whenever you use compressed text, you can use the same method to identify two parts (character, number), e.g., instead of `'aaaaa'`, you can use `[(5, 'a')]` and write a single letter like a letter. For example, `'suuuuper'` will be compressed to `[(1, 's'), (4, 'u'), (1, 'p'), (1, 'e'), (1, 'r')]`.. Program the `compress(text)` and `decompress(compressed_text)` radios, which receive compressed text (i.e., lists of tuples) and decompressed text, respectively. You can specify that we only compress texts containing letters and punctuation marks.

Test your program on the latest legally available text on the Internet. In your source code, provide a link to this text or write the code that downloads ten texts and is shared by the user.