Assignment no. 2

Scenario: Automatic analysis of a large number of files.

You are the proud owner of 20 hamsters. Looking at your hamsters, you suspected that the base composition of one of their genes changes over time. Therefore you conducted an experiment over 200 days on your 20 hamsters. Every day you took saliva samples from the hamsters and sent them to a lab to get them sequenced with respect to a specific gene. The lab returns you the DNA sequences in data files ending with the suffix `.raw.seq`. You receive one file per hamster per day, that is $200 \cdot 20 = 4,000$ files in total. The files start with a header that amongst other information includes the ID of the hamster as string and the date, which will just be an integer representing the day, followed by the sequencing data. The sequencing data includes the DNA you are interested in but also other information, such as the quality of the individual DNA base reads.

To verify your suspicion about the change in base composition, you want to plot how the average numbers of the bases (A, C, G, T) changed over the 200 days. Due to reasons you also want to create a histogram of the subsequence *ACC* in the data files.

To keep the program modular, you decide to use functions for different subtasks of the analysis.

Hamsters aside, taking a first look at the data you are working with is typically one of the first steps when confronted with a concrete ML task. Visualizing (properties of) your data (e.g. mean/variance, histograms, clustering, ...) can give you insights that might be useful or even essential for applying further ML methods.

Preparation: Download `hamstergenegen.py` from moodle, create a folder `00`, and run `hamstergenegen.py`. Provide the folder name as first command line argument when running the file. It will create fictional hamster gene data files and write them to the specified folder. You can call it via the command line as

```
python3 hamstergenegen.py 00
```

where `00` is the folder you created earlier. The folder `00` and the file `hamstergenegen.py` should be located in the same directory.

Each file consist of a header followed by 3 columns, where the first column `info` can be ignored. The second column `base` holds the sequenced base. The third column `quality` holds a value representing the quality of the measurement for the single base. Each file holds a consecutive sequence of 700 bases. The sequence starts at the top of the file and continues until the line `% End of data` is reached. There is one base per line in the file.

Hint: It might be helpful to open one of the `.raw.seq` files in a text editor to see what they look like.

Exercise 5 [10 points]

In this exercise you should parse the file content and count the bases and appearances of a certain subsequence of arbitrary length for a single file.

For this you should write a function `count_bases_and_subsequence(data_as_string: str, subsequence: str)` that takes two arguments `data_as_string` and `subsequence`. `data_as_string` is the content of one of the `.raw.seq` data files as a string. `subsequence` is the subsequence that should be counted as a string.

Your function should:

- Count the numbers of the different bases (A, C, G, T) in the `base` column.
- Count the number of the specified subsequence in the `base` column.
- For counting bases and subsequences, your program should:
  - Handle characters in the `base` column and in the specified subsequence in a case-insensitive fashion.
  - Ignore characters in the `base` column that are not valid bases (valid bases are A, C, G, T, a, c, g, t).
  - Ignore empty lines and lines that start with a % character.
  - Ignore lines with a value of less than 0.08 in the `quality` column.
  - Ignore all lines below the line `% End of data`.
  - If an ignored line or ignored character exists within a subsequence, this subsequence should not be counted (e.g. *ACNC* does not count for subsequence *ACC*). You can assume that the specified subsequence is valid (i.e. it is a string containing only valid bases and has a length greater than 0).

The function should return a tuple with 2 elements in this order:

1. The count of the subsequence as integer.
2. The number of bases as dictionary with keys (”a”, ”c”, ”g”, ”t”) and the count of the respective bases as integer values.

For testing your program, you can use the following lines to read the content of a `.raw.seq` file to a string:

```
filename = ”data_00−000.raw.seq” # This is the name or path of the file to read
with open(filename, ’r’) as fh:
    file_content = fh.read()
# At this point file_content will be the file content as string
# Your function should be callable like this:
subsequence_count, base_counts = count_bases_and_subsequence(data_as_string=
  file_content, subsequence=”ATTC”)
```

Hint: If you do not know how to tackle this problem, you can divide the task into subtasks. For example:

1. First write a Python script that loops over all lines of the string (the character `\n` marks the end of a line).
2. Then ignore empty lines and lines that start with a % character.

3. Then try to extract the second column in each iteration (i.e. for each line in the string).
4. Then count the 4 different bases in the second column.
5. Then ignore the bases with quality values of less than 0.08 while counting.
6. Then count a fixed subsequence, e.g. *ACC*.
7. Then count arbitrary subsequences, specified by a variable (which will be replaced by the function argument in the next step).
8. Finally, turn your code into a function and return the 2 objects as specified above.

Exercise 6 [4 points]

In this exercise you should write a function `get_hamsters(folderpath: str)`, which takes a string `folderpath` as argument. `folderpath` is the name or path of the folder where the `.raw.seq` data files are located.

Your function should create a sorted list of all files in the folder `folderpath` that have a filename ending with `.raw.seq`. The search for files should be performed recursively (i.e. also search in sub-folders of `folderpath`). Sorting should be done via `sorted()` to sort the files by their filepath (i.e. the directory name followed by the base filename of the file).

Your function should then return a generator object that returns a tuple for each file. This should be done using `yield`. The returned tuple should contain 2 elements in this order:

1. The base filename of the current file (see `os.path.basename`).
2. The content of the current file as a string.

Make sure to not read all file contents at once but use a loop and yield, which will read them one file at a time.

Example usage:

```
folderpath = ”00”
file_reader = get_hamsters(folderpath = folderpath)
# This should print the length of each file content for all sorted files:
for filename, file_content in file_reader:
    # file_content should be the content of the file as string
    print(f”{filename}: {len(file_content)}”)
```

Hint: Use the `with` statement when reading from the files as shown in the lecture materials.
Hint: Debug your program on a small folder with only a few files first before applying it to the full folder.
Hint: You may again divide the task into subtasks. For example:

1. First write a Python script and not a function.
2. Get a list of all the files in a folder and its sub-folders.
3. Then sort this list (you can apply sorted() directly to the list, since you should sort by filepaths).
4. Then read the content of the first file into a string (i.e. for the first filename in the list).
5. Then write a loop to loop over all filenames in the list.
6. Then read the content for each file at each iteration of the for-loop. For debugging you can print the length of the current file content string at each loop iteration.
7. Then turn your script into a function
8. Then, instead of printing the length of the file content at each iteration, use `yield` to return the base filename and file content string at each iteration.

Exercise 7 [8 points] tba
Exercise 8 [8 points] tba
Exercise 9 [5 bonus points] tba

Submission: electronically via Moodle:

`https://moodle.jku.at/`

Deadline: For deadlines see individual Moodle exercises.
Follow the **instructions for submitting homework** stated on the Moodle page!