Introduction to Python
Day Four Exercises

Stephanie Spielman
Email: stephanie.spielman@gmail.com

All exercises in this worksheet **must** be completed in Python scripts, and then run from the UNIX command-line with `python name_of_script.py`. Be sure to avoid spaces in file names!!

1 Interacting with files

Be sure to always close the file once you are done with it! If you use `with` control flow, python will close the file automatically.

1. Open the file "file1.txt" in read-mode, and print its contents to screen. For this, you should use the `.read()` method, which saves the contents of the file to a single string. Perform this task twice: once using `open` and `close`, and once using `with` control-flow.

2. Open the file "file1.txt" in read-mode, and save all lines in this file to a list using the `.readlines()` method. Write a new file called "upper_file1.txt" which contains the same contents of "file1.txt" but in upper-case. Try to do this task using a single for-loop, and don't forget that in order to write newlines (the "enter" key) to a file, you must include `"\n"` in the string you are writing to file!

3. Open the file "upper_file1.txt" in read-mode, and *append* the sentence: "I just created this upper-case file!" to the bottom of the file.

4. Open the file "flu_sequences.fasta" in read-mode, and save the contents of a file as a single string using the `.read()` method (be sure to close the file, or use `with` control-flow!). Address the following questions using Python:

   (a) How many *characters* are in this file? Remember that `.read()` saves the entire file as a single string!

   (b) How many *lines* are in this file? (Hint: use the `.split()` method as part of the solution).

   (c) Use a for-loop to print the first 10 lines of the file.

   (d) Open and read the file again, except this time, use the `.readlines()` method instead of `.read()`. Perform the same three tasks. (Hint: to count the characters, loop over the line list and keep a running counter of the length of each line.)

2 Python modules

1. For this question, you will use the `os` module. In the included zip directory, there are 20 text files named file1.txt, file2.txt, file3.txt,..., file20.txt (below, they are referred to as file<1-20.txt> for convenience). For this question, use python to count the number of lines in each file. You will save all file line-counts to a `dictionary`, in which the keys are the file names and the values are the line counts. Once you have created

the dictionary, print it to screen to make sure it looks correct.
Hint: Use the `os` module function `os.listdir` to list all files in the current directory and loop over all of these files in order to count their lines. To ensure that you are only counting lines in these text files, use the `.startswith()` and `.endswith()` method (all files start with "file" and all files end with ".txt").

2. Create a new file called "file_full.txt" which contains all contents from each file<1-20>.txt (in order). Again, use `os.listdir` to loop over these files. Once this file has been created, use the `shutil` module to *move* the file to a different directory on your computer. After this script has run, navigate in your terminal (using the UNIX command `cd`) to this directory to confirm that "file_full.txt" is indeed there.

3. Write a `function` which counts the number of times a given letter occurs in a file. The function should take two arguments: the file name (including its path) and the letter to count. The function should return the number of occurrences for that letter.
Hint: In this function, you should open the file, read it using the `.read()` method, and then use the `.count()` method to count the number of times the letter appears. Perform the following tasks with this function:

   - Run this function on all file<1-20.txt> to count the letter "a". For every file which contains more than 100 a's, use the `shutil` to *make a copy of this file* called "lots_fileX.txt". For example, if file10.txt contains more than 100 a's, you should create a second file called "lots_file10.txt". This will require the `os` and the `shutil` modules.
   - Use the `sys` module (specifically the `sys.argv` variable) to capture `two command-line arguments`: a file on which to run this function and the letter to count. Run the function using the given input parameters, and print a a sentence to the screen summarizing the results.
   - Modify your code from the previous point to use the `os` module function `os.path.exists()`. This function takes a single argument, a path to a file/directory, and returns True or False indicating if the file exists or not. So, before running the function on the command-line provided file, check if the file exists using `os.path.exists()`. If the file exists, then proceed as usual. If the file does not exist, print to screen "Your file does not exist!", and then exit immediately with `sys.exit()`.
   - Now, run the function *from an entirely separate script*. Accomplish this by importing the file with this function into the script you will run. You can run the function on a file of your choice.

Bonus Question!

For this question, you are going to save some useful information to a file. Specifically, you will create a *CSV* (comma-separated values) file. This is basically what Excel files are - files with columns of data. The top row is the header, and each subsequent row contains data. Columns of data are separated by commas in csv files (note that there are other types of *delimited* files, like tab-delimited files).

You will use the data from the file "dopamine_sequences.fasta," which contains DNA sequences of vertebrate dopamine receptors published in Spielman et al. [2015]. For convenience, the separate file "dopamine_sequences_dictionary.txt" contains a Python dictionary of these sequences; the keys are sequences IDs, and the values are the sequences. You can copy and paste the contents of "dopamine_sequences_dictionary.txt" into the Python script you'll write for this question.

Now, create a CSV file with two columns: the sequence ID and the sequence length. Follow this general strategy:

1. Collect all information to write to the output file (this has been done for you in the dictionary!)

2. Open the output file and write a header to the file (remember to add `"\n"` for newlines!)

3. Write each line to the output file. In this case, you will need to loop over the dictionary and write each key:value pair in CSV format with a newline character, e.g. "sequenceID,length\n"

Once you have written the file, open it in a text editor to be sure that it is correct.
Now, modify this code in the following ways:

1. Modify your CSV-writing code such that it uses a function to write the CSV. This function should take three arguments: the header string, the dictionary to write, and the output file name. The function does not need to return anything.

2. Modify your code and function so that you now write a file with three columns: sequence ID, sequence length, and percent of ambiguous nucleotides per DNA sequence. In this sequence, the character "N" is considered ambiguous (unlike A, C, G, T).

References

SJ Spielman, K Kumar, and CO Wilke. Comprehensive, structurally-informed alignment and phylogeny of vertebrate biogenic amine receptors. *PeerJ*, 3:e773, 2015. doi: 10.7717/peerj.773.