Introduction to Python, 2018

Day 3 Exercises

Part I: Practice our skills

1. A professor has decided to curve grades in a very special way:
   - Grades above 95 are reduced by 10%
   - Grades between 75-95 (inclusive) remain the same
   - Grades below 75 are raised by 10%.

   You have been tasked with crunching the numbers! Perform the following tasks:

   - Create a list of new grades that reflects these rules from the following original grades:
     `grades = [45, 94, 25, 68, 88, 95, 72, 79, 91, 82, 53, 66, 58]`
   - The professor has changed his mind: he now wants to use a scaling factor of 0.078325 (instead of 0.1), because why not! Recompute the grades from part 1 using this new scaling (Hint: no hard-coding!)
   - The *nested* list below contains three sets of grades for silly professor's three classes:
     ```
     all_grades = [[45, 94, 25, 68, 88, 95, 72, 79, 91, 82, 53, 66, 58],
                   [23, 46, 17, 67, 55, 42, 31, 73],
                   [91, 83, 79, 76, 82, 91, 95, 77, 82, 77]]
     ```
     Create a new nested list with the curved grades for each of these groups.
   - Now, imagine that those three sets of grades correspond, in order, to the classes indicated in this list:
     `class_names = ["Psychology 101", "Sociology 101", "Political Science 101"]`
     Create a *dictionary* representing the *curved* grades for each of these classes. Your final dictionary should have the class name as keys, and each list of curved grades as values.

2. For this set of questions, you will calculate the molecular weight of a protein sequence, using this dictionary:

```
amino_weights = {"A":89.09, "R":174.20, "N":132.12,
                 "D":133.10, "C":121.15, "Q":146.15,
                 "E":147.13, "G":75.07, "H":155.16,
                 "I":131.17, "L":131.17, "K":146.19,
                 "M":149.21, "F":165.19, "P":115.13,
                 "S":105.09, "T":119.12, "W":204.23,
                 "Y":181.19, "V":117.15}
```

   - Calculate the molecular weight of this sequence: `"GAHYADPLVKMPWRTHC"`
   - Now, calculate the molecular weight of this sequence *which contains ambiguities*: `"KLSJXXFOWXNNCPRHGGYA"`. Assume that the molecular weight of an ambiguous amino acid is the average weight of all amino acids.

3. For this question, you will tabulate the number of each nucleotide in a DNA sequence.

   - Create a dictionary which contains key:value pairs as nucleotide:count for this sequence: `"ACATAGACCAGAGACT"`. Use the `.count()` method by looping over a list of nucleotides (`nucs = ["A", "C", "G", "T"]`) to solve this question.
   - Now, create a similar dictionary for this DNA sequence which contains *ambiguities*: `"AGCTANTAGNNNNNAGGATCCNNAANNNNCATAGC"`. This time, use a for-loop over the sequence itself to "build up" a dictionary of counts for those characters which appear in the sequence.

Part II: Functions

1. Write a function to compute the GC content of a DNA sequence. The function should accept a single argument, the DNA sequence, and return the GC percentage. Test your function with the nucleotide sequence "AGCTATAGCATAGC".
2. Write a function that calculates the percentage of a given nucleotide from a DNA sequence. The function should accept two arguments: the nucleotide of interest and the DNA sequence. It should return the nucleotide percentage. Test your function with the nucleotide sequence "AGCTATAGCATAGC".
3. Write a function that calculates the percentage each nucleotide in a given DNA sequence. of a given nucleotide from a DNA sequence. The function should accept a single argument, the DNA sequence, and return *a dictionary* containing `key:value` pairs of `nucleotide:percentage`. You can assume that the provided sequence contains only A, C, G, T. Test your function with the nucleotide sequence `"AGCTATAGCATAGC"`.
4. Write a function to guess whether a provided sequence is DNA or protein. For this task, assume that any sequence comprised of $\geq 50\%$ A, C, G, T is a DNA sequence. Test your function with the following two sequences:

   - `"AGCTATGCATACGAGCATAGC"`

- `"AGIILLCPKLKKQWTATWCAGCATADSARCVLMKGC"`

5. Modify the previous function to *ignore all ambiguities in calculations*. Use this list of ambiguous characters for this task: `ambig = ["B", "J", "N", "O", "X", "Z"]`. Test your function with the following sequence:`"APAPPPKKLRATNNYPOPPBXXXXXNTYGCTATLMQASDFTDTCATAGC"`

Part III: File Input/Output

Files used in these exercises can be downloaded from the course website. Be sure to write your scripts in the same directory as these files!

1. Open the file `file1.txt` in read-mode, and print its contents to screen. 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 newly created file `upper_file1.txt` in read-mode. Loop over the file lines *without* using `.read()` or `.readlines()`, and print out lines as you loop.
4. Modify the previous for-loop to only print out lines in `upper_file1.txt` which contain at least (i.e. >=) 5 letter `E`'s.
5. You should notice 20 files named `file1.txt, file2.txt, ..., file20.txt`. Write a for-loop to open each of these files (Hint: use the `range()` function to loop over file names). For each file, print each line that contains more than 25 characters.
6. Convert our zoo-keeper dictionaries into a *comma-separated file* with the header `animal,vore,food`, and rows should contain corresponding information, i.e. `lion,carnivore,meat`. Perform this task with a *single* for-loop.

```
category = {"lion": "carnivore",
            "gazelle": "herbivore",
            "anteater": "insectivore",
            "alligator": "homovore",
            "hedgehog": "insectivore",
            "cow": "herbivore",
            "tiger": "carnivore",
            "orangutan": "frugivore"}

 feed = {"carnivore": "meat",
         "herbivore": "grass",
         "frugivore": "mangos",
         "homovore": "visitors",
         "insectivore": "termites"}
```

7. Create a second zoo-keeper file by *converting* the CSV into a tab-separated file. Perform this task by reading in the CSV, *replacing* commas with tabs, where tabs can be created as the string `"\t"`. For this, use the string method `.replace()`. This method takes two arguments: the pattern to replace, and the replacement. For example, the following snippet will replace all commas with tabs in a string called `mystring`:

```
mystring2 = mystring.replace(",", "\t")
```