Introduction to Python
Day Four Exercises

Stephanie Spielman
Email: stephanie.spielman@gmail.com

1 File Input and Output

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. The following exercises make use of files distributed in today's course materials. Be sure that you are in the correct directory when interacting with the files!

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 append-mode, and *append* the sentence: "I just created this upper-case file!" to the bottom of the file. Close the file and open it (separately) to examine the contents.

4. Again, open the file "upper_file1.txt", this time in read-mode. Loop over the file lines *without* using `.read()` or `.readlines()`. Print out lines as you loop.

5. Modify the previous for-loop to only print out lines in "upper_file1.txt" which contain at least (i.e. >=) 5 letter "E"s. This will require an `if` statement as well as the method `.count()`.

6. 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, write a for-loop over the file lines (this does not use `.read()` or `.readlines()`). Print each line that contains more than 25 characters (determine this with `len()`.)

7. Using the zoo-keeper dictionaries from the **Working with dictionaries** exercises (also provided below), create and write to a new file called "animal_food.txt" which contains the sentences you created earlier:
   The lion eats meat
   The gazelle eats grass
   The anteater eats termites
   The alligator eats visitors
   ... etc.
   ```
   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"}
   ```

2 Defining Functions

For this set of exercises, you will re-write some of the course's previous exercises as functions. The code doing the actual computation will remain virtually the same, except it will be written in the context of a function and subsequently called. **After you write each function, run it with 2-3 test cases to confirm that it works as expected!!**

1. In Texas, you can be a member of the elite "top 1%" if you make at least $423,000 per year. Alternatively, in Hawaii, you can be a member once you start making at least $279,000 per year! Finally, if you live in New York, you need to earn at least $506,000 a year to make the cut.

   Write a function to determine if a given salary is a "top-1%" salary in Texas, Hawaii, and/or New York. Your function should take a single argument, the salary, and *print* a sentence indicating in which state(s) this salary is and is not a top-1% salary. Your function should not return a value.

2. Write a function that returns a list of the powers (exponents 0-15, inclusive!!) for a provided number. This function should take 1 argument: the number to raise to powers 0-15.

3. Modify the previous function to take two additional arguments indicating which exponents to calculate. For example, in the previous exercise, you would have provided the additional arguments 0 and 15 (or, 16, depending on how you wrote your loop).

4. Write a function to curve a list of grades, silly-professor style. This function should take *four arguments*:

   - A list of grades to curve
   - The cutoff *above which* grades are reduced
   - The cutoff *below which* grades are raised
   - The scaling value

   The function should return a list of curved grades.

5. Write a function to compute the molecular weight of a protein sequence. This function should take a single argument, a protein-sequence string, and it should return a single value, the molecular weight. Your function should account for the potential presence of ambiguous amino acids (again, compute these weights from average of all weights). Once your function is written, run it on the protein sequence "ABVPOXIRBTQQWS." Use this dictionary in your function:
   ```
   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}
   ```

6. Write a function to open file and return the number of letter "f"'s in the file (both capital and lower-case!).

7. Modify the previous function such that it contains an additional argument indicating which letter/character to count.

3 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".
   - 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.