This lesson is being piloted (Beta version)

‹ (../02-file_parsing/index.html) Python Scripting for Computational Molecular Science (../) (../04-tabular ›

Processing Multiple Files and Writing Files

Overview

Teaching: 25 min
Exercises: 10 min
Questions

- How do I analyze multiple files at once?

Objectives

- Import a python library.
- Use python library funtions.
- Process multiple files using a `for` loop.
- Print output to a new text file.

Processing multiple files

In our previous lesson, we parsed values from output files. While you might have seen the utility of doing such a thing, you might have also wondered why we didn't just search the file and cut and paste the values we wanted into a spreadsheet. If you only have 1 or 2 files, this might be a very reasonable thing to do. But what if you had 100 files to analyze? What if you had 1000? In such a case the cutting and pasting method would be very tedious and time consuming.

One of the real powers of writing a program to analyze your data is that you can just as easily analyze 100 files as 1 file. In this example, we are going to parse the output files for a whole series of aliphatic alcohol compounds and parse the energy value for each one. The output files are all saved in a folder called outfiles that you should have downloaded in the setup for this lesson. Make sure the folder is in the same directory as the directory where you are writing and executing your code.

To analyze multiple files, we will need to import a python **library**. A **library** is a set of modules which contain functions. The functions within a library or module are usually related to one another. Using libraries and in Python reduces the amount of code you have to write. In the last lesson, we imported `os.path`, which was a module that handled filepaths for us.

In this lesson, we will be using the `glob` library, which will help us read in multiple files from our computer. Within a library there are modules and functions which do a specific computational task. Usually a function has some type of input and gives a particular output. To use a function that is in a library, you often use the dot notation introduced in the previous lesson. In general

Python

```
import library_name
output = library_name.funtion_name(input)
```

We are going to import two libraries. One is the `os` library which controls functions related to the operating system of your computer. We used this library in the last lesson to handle filepaths. The other is the `glob` library which contains functions to help us analyze multiple files. If we are going to analyze multiple files, we first need to specify where those files are located.

Exercise

How would you use the `os.path` module to point to the directory where your outfiles are?

Solution

In order to get all of the files which match a specific pattern, we will use the wildcard character `*`.

Python

```
file_location = os.path.join('data', 'outfiles', '*.out')
print(file_location)
```

Output

```
data/outfiles/*.out
```

This specifies that we want to look for all the files in a directory called `data/outfiles` that end in ".out". The * is the wildcard character which matches any character.

Next we are going to use a function called `glob` in the library called `glob`. It is a little confusing since the function and the library have the same name, but we will see other examples where this is not the case later. The output of the function `glob` is a list of all the filenames that fit the pattern specified in the input. The input is the file location.

Python

```
import glob
filenames = glob.glob(file_location)
print(filenames)
```

Output

```
['data/outfiles/butanol.out', 'data/outfiles/decanol.out', 'data/outfiles/ethanol.out', 'data/outfiles/heptanol.ou
t', 'data/outfiles/hexanol.out', 'data/outfiles/methanol.out', 'data/outfiles/nonanol.out', 'data/outfiles/octanol.
out', 'data/outfiles/pentanol.out', 'data/outfiles/propanol.out']
```

This will give us a list of all the files which end in `*.out` in the `outfiles` directory. Now if we want to parse every file we just read in, we will use a `for` loop to go through each file.

Python

```
for f in filenames:
    outfile = open(f,'r')
    data = outfile.readlines()
    outfile.close()
    for line in data:
        if 'Final Energy' in line:
            energy_line = line
            words = energy_line.split()
            energy = float(words[3])
            print(energy)
```

Output

```
-232.1655798347283
-466.3836241400086
-154.09130176573018
-349.27397687072676
-310.2385332251633
-115.04800861868374
-427.3465180082815
-388.3110864554743
-271.20138119895074
-193.12836249728798
```

Notice that in this code we actually used two `for` loops, one nested inside the other. The outer `for` loop counts over the filenames we read in earlier. The inner `for` loop counts over the line in each file, just as we did in our previous file parsing lesson.

The output our code currently generates is not that useful. It doesn't show us which file each energy value came from.

We want to print the name of the molecule with the energy. We can use `os.path.basename`, which is another function in `os.path` to get just the name of the file.

Python

```
first_file = filenames[0]
print(first_file)

file_name = os.path.basename(first_file)
print(file_name)
```

Output

```
data/outfiles/propanol.out
propanol.out
```

Exercise

How would you extract the molecule name from the example above?

Solution

Using the solution above, we can modify our loop so that it prints the file name along with each energy value.

Python

```
for f in filenames:
    # Get the molecule name
    file_name = os.path.basename(f)
    split_filname = file_name.split('.')
    molecule_name = split_filename[0]

    # Read the data
    outfile = open(f,'r')
    data = outfile.readlines()
    outfile.close()

    # Loop through the data
    for line in data:
        if 'Final Energy' in line:
            energy_line = line
            words = energy_line.split()
            energy = float(words[3])
            print(molecule_name, energy)
```

Output

```
propanol -193.12836249728798
pentanol -271.20138119895074
decanol -466.3836241400086
methanol -115.04800861868374
octanol -388.3110864554743
ethanol -154.09130176573018
hexanol -310.2385332251633
heptanol -349.27397687072676
butanol -232.1655798347283
nonanol -427.3465180082815
```

Printing to a File

Finally, it might be useful to print our results in a new file, such that we could share our results with colleagues or or e-mail them to our advisor. Much like when we read in a file, the first step to writing output to a file is opening that file for writing. In general to open a file for writing you use the syntax

Python

```
filehandle = open('file_name.txt', 'w+')
```

The `w` means open the file for writing. If you use `w+` that means open the file for writing and if the file does not exist, create it. You can also use `a` for append to an existing file or `a+`. The difference between `w+` and `a+` is that `w+` will overwrite the file if it already exists, whereas `a+` will keep what is already there and just add additional text to the file.

Python can only write strings to files. Our current print statement is not a string; it prints two python variables. To convert what we have now to a string, you place a capital **F** in front of the line you want to print and enclose it in single quotes. Each python variable is placed in braces. Then you can either print the line (as we have done before) or you can use the `filehandle.write()` command to print it to a file.

To make the printing neater, we will separate the file name from the energy using a tab. To insert a tab, we use the special character `\t`.

Python

```
datafile = open('energies.txt','w+')  #This opens the file for writing
for f in filenames:
    # Get the molecule name
    file_name = os.path.basename(f)
    split_filename = file_name.split('.')
    molecule_name = split_filename[0]

    # Read the data
    outfile = open(f,'r')
    data = outfile.readlines()
    outfile.close()

    # Loop through the data
    for line in data:
        if 'Final Energy' in line:
            energy_line = line
            words = energy_line.split()
            energy = float(words[3])
            datafile.write(F'{molecule_name} \t {energy} \n')
datafile.close()
```

After you run this command, look in the directory where you ran your code and find the “energies.txt” file. Open it in a text editor and look at the file.

In the file writing line, notice the `\n` at the end of the line. This is the newline character. Without it, the text in our file would just be all smushed together on one line. Also, the `filehandle.close()` command is very important. Think about a computer as someone who has a very good memory, but is very slow at writing. Therefore, when you tell the computer to write a line, it remembers what you want it to write, but it doesn’t actually write the new file until you tell it you are finished. The `datafile.close()` command tells the computer you are finished giving it lines to write and that it should go ahead and write the file now. If you are trying to write a file and the file keeps coming up empty, it is probably because you forgot to close the file.

A final note about string formatting

The F’string’ notation that you can use with the print or the write command lets you format strings in many ways. You could include other words or whole sentences. For example, we could change the file writing line to

Python

```
datafile.write(F'For the file {molecule_name} the energy is {energy} in kcal/mole.')
```

where anything in the braces is a python variable and it will print the value of that variable.

Project Assignment

This is a project assignment which you can complete to test your skills. This project should be used when this material is used in a long workshop, or if you are working through this material independently.

File parsing homework

In the lesson materials, there is a file called `03_Prod.mdout`. This is a file output by the Amber molecular dynamics simulation program.

If you open the file and look at it, you will see sections which look like this:

```
 NSTEP =      100   TIME(PS) =      20.200  TEMP(K) =   296.43  PRESS =  -300.8
 Etot   =     -4585.1049  EKtot   =      1129.2368  EPtot      =     -5714.3417
 BOND   =         1.5160  ANGLE   =         6.9846  DIHED      =        11.7108
 1-4 NB =         4.3175  1-4 EEL =        49.9911  VDWAALS    =       882.6741
 EELEC  =     -6671.5358  EHBOND  =         0.0000  RESTRAINT  =         0.0000
 EKCMT  =       583.7810  VIRIAL  =       786.8823  VOLUME     =     31270.0410
                                                    Density    =         0.6104
 Ewald error estimate:   0.3214E-03
 ------------------------------------------------------------------------------
```

Your assignment is to parse this file, and write a new file containing a list of the > total energies. Name your file `Etot.txt`. When you open it, it should look like this:

```
-4585.1049
-4573.5326
-4548.1223
-4525.341
-4542.8995
-4550.9376
-4543.8652
-4570.4109
-4550.4225
-4585.2078
...
```

`...` indicates that you will have many more rows. We've only shown the first 10 here.

If you are unsure of where to start with this assignment, check the hint section!

Hints

Solution

Key Points

- Use the glob function in the python library glob to find all the files you want to analyze.
- You can have multiple `for` loops nested inside each other.
- Python can only print strings to files.
- Don't forget to close files so python will actually write them.

‹ (../02-file_parsing/index.html)

› (../04-tabular

Copyright © 2018–2020 The Molecular Sciences Software Institute (http://molssi.org)

Edit on GitHub (https://github.com/MolSSI-Education/python_scripting_cms/edit/gh-pages/_episodes/03-multiple_files.md) / Contributing (https://github.com/MolSSI-Education/python_scripting_cms/blob/gh-pages/CONTRIBUTING.md) / Source (https://github.com/MolSSI-Education/python_scripting_cms/) / Cite (https://github.com/MolSSI-Education/python_scripting_cms/blob/gh-pages/CITATION) / Contact (mailto:education@molssi.org)

Using The Carpentries style (https://github.com/carpentries/styles/) version 9.5.0 (https://github.com/carpentries/styles/releases/tag/v9.5.0).