This lesson is being piloted (Beta version)

‹ Python Scripting for Computational Molecular Science (../) (../01-introduction/index.html) › (../03-multipl

File Parsing

Overview

Teaching: 20 min
Exercises: 25 min
Questions

- How do I sort through all the information in a text file and extract particular pieces of information?

Objectives

- Open a file and read in its contents line by line.
- Search for a particular string in a file.
- Manipulate strings and change data types.
- Print to a new file.

Working with files

One of the most common tasks in research is analyzing data. Many computational chemistry programs output text files that include a large amount of information including text and data that you need to analyze. Often, you need to sort through the output file and identify particular pieces of information that are most important to you. In general, this is called file parsing.

Working with file paths - the `os.path` module

For this section, we will be working with the file `ethanol.out` in the `outfiles` directory.

To see this, go to a new cell and type `ls .` `ls` stands for 'list', and will list all of the contents of the current directory. **This command is not a Python command, but will work in the Jupyter notebook**. To see everything in the `data` directory, type

```
ls data
```

You should see something like

Output

```
distance_data_headers.csv   outfiles                 sapt.out                water.xyz
```

Here, `distance_data_headers.csv` , `sapt.out` , and `water.xyz` are all files, while `outfiles` is another directory.

In order to parse a file, you must tell Python the location of the file, or the "file path". For example, you can see what folder your Jupyter notebook is in by typing `pwd` into a cell in your notebook and evaluating it. `pwd` stands for 'print working directory', and can also be used in your terminal to see what directory you're in.

After evaluating the cell with `pwd` , you should see an output similar to the following if you are on Mac or Linux:

Output

```
'/Users/YOUR_USER_NAME/cms-workshop'
```

or similar to this if you are on Windows

Output

```
'C:\Users\YOU_USER_NAME\Desktop\cms-workshop'
```

Notice that the file paths are different for these two systems. The Windows system uses a backslash ('\'), while Mac and Linux use a forward slash ('/') for filepaths.

When we write a script, we want it to be usable on any operating system, thus we will use a python module called `os.path` that will allow us to define file paths in a general way.

In order to get the path to the `ethanol.out` file in a general way, type

Python

```
import os

ethanol_file = os.path.join('data', 'outfiles', 'ethanol.out')
print(ethanol_file)
```

You should see something similar to the following

Output

```
data/outfiles/ethanol.out
```

Here, we have specified that our filepath contains the 'data' and 'outfiles' directory, and the `os.path` module has made this into a filepath that is usable by our system. If you are on Windows, you will instead see that a backslash is used.

📌 Absolute and relative paths

File paths can be *absolute*, or *relative*.

A relative file path gives the location relative to the directory we are in. Thus, if we are in the `cms-workshop` directory, the relative filepath for the `ethanol.out` file would be `data/ethanol.out`

An absolute filepath gives the complete path to a file. This could file path could be used from anywhere on a computer, and would access the same file. For example, the absolute filepath to the `ethanol.out` file on a Mac might be `Users/YOUR_USER_NAME/Desktop/cms-workshop/data/ethanol.out`. You can get the absolute path of a file using `os.path.abspath(path)`, where path is the relative path of the file.

📌 Python `pathlib`

We are working with the `os.path` module here, and this is how you will see people handle file paths in most Python code. However, as of Python 3.6, there is also a `pathlib` module in the Python standard library that can be used to represent and manipulate filepaths. `os.path` works with filepaths as strings, while in the `pathlib` module, paths are objects. A good overview of the pathlib module can be found here (https://realpython.com/python-pathlib/).

Reading a file

In Python, there are many ways in python to read in information from a text file. The best method to use depends on the type of data and the type of analysis you are performing. If you have a file with lots of different types of information, text and numbers, with different types of formatting, the most generic way to read in information is the `readlines()` function. Before you can read in a file, you have to open the file using the file path we defined above. This will create a file object, or filehandle. The file we will be analyzing in this example is a PSI4 (http://www.psicode.org) output file for a SCF/cc-pVDZ energy calculation for an ethanol molecule.

Python

```
outfile = open(ethanol_file,"r")
data = outfile.readlines()
```

This code opens a file for reading and assigns it to the filehandle `outfile`. The `r` argument in the function stands for `read`. Other arguments might be `w` for `write` if we want to write new information to the file, or `a` for append if we want to add new information at the end of the file.

In the next line, we use the `readlines` function to get our file as a list of strings. Notice the dot notation introduced last lesson; readlines acts on the file object given right before the dot. The function creates a list called data where each element of the list is a string that is one line of the file. This is always how the `readlines()` function works.

📌 `readlines` function behavior

Note that the `readlines` function can only be used on a file object one time. If you forget to set `outfile.readlines()` equal to a variable, you must `open` the file again in order to get the contents of the file.

After you open and read information from a file object, you should always close the file.

Python

```
outfile.close()
```

An alternative way to open a file.

Alternatively, you can open a file using `context-manager`. In this case, the context manager will automatically handle closing of the file. To use a context manager to open and close the file, you use the word `with`, and put everything you want to be done while the file is open in an indented block.

Python

```
with open(ethanol_file,"r") as outfile:
    data = outfile.readlines()
```

This is often the preferred way to deal with files because you do not have to remember to close the file.

Check Your Understanding

Check that your file was read in correctly by determining how many lines are in the file.

Answer

Searching for a pattern in your file

The file we opened is an output file which calculates the energy (and a lot of other stuff!) for an ethanol molecule. As stated previously, the `readlines()` function put the file contents into a list where each element is a line of the file. You may remember from lesson 1 that a `for` loops can be used to execute the same code repeatedly. As we learned in the previous lesson, we can use a for loop to iterate through elements in a list.

Let's take a look at what's in the file.

Python

```
for line in data:
    print(line)
```

This will print exactly what is in the file.

If you look through the output, you will see that the critical line says "Final Energy". We want to search through this file and find that line, and print only that line. We can do this using an `if` statement.

Returning to our file example,

Python

```
for line in data:
    if 'Final Energy' in line:
        energy_line = line
        print(energy_line)
```

Output

```
@DF-RHF Final Energy:  -154.09130176573018
```

Remember that `readlines()` saves each line of the file as a string, so `energy_line` is a string that contains the whole line. For our analysis, if we are most interested in the energy, we need to split up the line so we can save just the number as a different variable name. To do this, we use a new function called `split`. The `split` function takes a string and divides it into its components using a *delimiter*.

The *delimiter* is specified as an argument to the function (put in the parenthesis `()`). If you do not specify a delimiter, a space is used by default. Let's try this out.

Python

```
energy_line.split()
```

```
['@DF-RHF', 'Final', 'Energy:', '-154.09130176573018']
```

Or, we can use the colon (':') as the delimiter.

Python

```
energy_line.split(':')
```

```
['  @DF-RHF Final Energy', '  -154.09130176573018\n']
```

When we use ':' as the delimiter, a list with two elements is returned. It is split where a colon was found.

We can save the output of this function to a variable as a new list. In the example below, we take the line we found in the `for` loop and split it up into its individual words.

Python

```
words = energy_line.split()
print(words)
```

Output

```
['@DF-RHF', 'Final', 'Energy:', '-154.09130176573018']
```

From this `print` statement, we now see that we have a list called words, where we have split `energy_line`. The energy is actually the fourth element of this list, so we can now save it as a new variable.

Python

```
energy = words[3]
print(energy)
```

📌 Python negative indexing

We also recogize that "energy" is the last element of the list. Therefore, an alternative way to assign `energy` is:

Python

```
energy = words[-1]
print(energy)
```

In the example above, the index value of `-1` gives the last element, and `-2` would give the second last element of a list, and so on. An excelent tutorial on Python list accessed by index can be found here (https://realpython.com/python-lists-tuples/#list-elements-can-be-accessed-by-index)

Output

```
-154.09130176573018
```

If we now try to do a math operation on energy, we get an error message? Why do you think that is?

Python

```
energy + 50
```

Error

```
TypeError                                 Traceback (most recent call last)
<ipython-input-52-7bda8dd3f95d> in <module>()
----> 1 energy + 50

TypeError: must be str, not int
```

Even though `energy` looks like a number to us, it is really a string, so we can not add an integer to it. We need to change the data type of energy to a float. This is called *casting*.

Python

```
energy = float(energy)
```

Now it will work. If we thought ahead we could have changed the data type when we assigned the variable originally.

Python

```
energy = float(words[3])
```

Exercise on File Parsing (should we move this to the end?)

Use the provided sapt.out file. In this output file, the program calculates the interaction energy for an ethene-ethyne complex. The output reports four interaction energy components: electrostatics, induction, exchange, and dispersion. Parse each of these energies, in kcal/mole, from the output file. (Hint: study the file in a text editor to help you decide what to search for.) Calculate the total interaction energy by adding the four components together. Your code's output should look something like this:

```
Electrostatics : -2.25850118 kcal/mol
Exchange : 2.27730198 kcal/mol
Induction : -0.5216933 kcal/mol
Dispersion : -0.9446677 kcal/mol
Total Energy : 1.4475602000000003 kcal/mol
```

Solution

Searching for a particular line number in your file

There is a lot of other information in the output file we might be interested in. For example, We might want to pull out the initial coordinates for the molecule. If we look through the file in a text editor, we notice that the coordinates begin with a line that says

Center X Y Z Mass

and then the coordinates begin on the next line. In this case, we don't want to pull something out of this line, as we did in our previous example, but we want to know which line of the file this is so that we can then pull the coordinates from the next few lines.

When you use a for loop, it is easy to have python keep up with the line numbers using the `enumerate` command. The general syntax is

Python

```
for linenum, line in enumerate(list_name):
    do things in the loop
```

In this notation, there are now *two* variables you can use in your loop commands, `linenum` (which can be named something else) will keep up with what iteration you are on in the loop, in this case what line you are on in the file. The variable `line` (which could be named something else) functions exactly as it did before, holding the actual information from the list. Finally, instead of just giving the list name you use `enumerate(list_name)`.

`Enumerate` with index other than 0:

`enumerate(list_name)` will start with 0-index so the first line will be label as '0', to change this behavior, use `start` variable in enumerate. For example, to start with index of "1" you can do: ```python for linenum, line in enumerate(data, start=1): # do something with 'linenum' and 'line'

This block of code searches our file for the line that contains "Center" and reports the line number.

Python

```
for linenum, line in enumerate(data):
    if 'Center' in line:
        print(linenum)
        print(line)
```

Output

```
77
       Center              X                 Y                 Z              Mass
```

Now we know that this is line 77 in our file (remember that you start counting at zero!).

Check Your Understanding

What would be printed if you entered the following:

Python

```
print(data[77])
print(data[78])
print(data[79])
print(data[80])
print(data[81])
```

Answer

A final note about regular expressions

Sometimes you will need to match something more complex than just a particular word or phrase in your output file. Sometimes you will need to match a particular word, but only if it is found at the beginning of a line. Or perhaps you will need to match a particular pattern of data, like a capital letter followed by a number, but you won't know the exact letter and number you are looking for. These types of matching situations are handled with something called *regular expressions* which is accessed through the python module `re`. While using regular expressions (regex) is outside the scope of this tutorial, they are very useful and you might want to learn more about them in the future. A tutorial can be found at Automate the Boring Stuff with Python (https://automatetheboringstuff.com/2e/chapter7/) book. A great test site for regex is here (https://regex101.com/)

Key Points

- You should use the os.path module to work with file paths.
- One of the most flexible ways to read in the lines of a file is the `readlines()` function.
- An `if` statement can be used to find a particular string within a file.
- The split() function can be used to seperate the elements of a string.
- You will often need to recast data into a different data type when it was read in as a string.

‹ (../01-introduction/index.html)

› (../03-multipl