Merging Files

Lesson 4 (modified from Haddock and Dunn Chapter 11)

In this lesson we will go over combining pieces of data from different files into one master data file. Comments in blue will indicate code written in `nano` and comments in brown will indicate commands used in the terminal (bold indicates commands you need to type in to the terminal).

In previous lessons, you have learned how to combine files end to end using the `append` command. In many cases, you will want to add additional columns to data, which cannot be done with `append`. For example, you have one file with two columns: Species Name and Mass. In another file you have three more columns you want to add with additional measurements taken for the species indicated in the first file. We will create a list of identical file types and select the desired data and combine them into a master file. First, it is important that you know the structure of the data files that you want to piece together.

If you haven't already, download the example files and scripts from the Practical Computing for Biologists website: http://practicalcomputing.org/downloads
Click on 'Examples' to download the files.

Examining files

Move into the spectra directory and look at the contents. This directory has four files containing emission spectra for different color LEDs. View one of the files using the `less` command.

```
host:~ name$ cd ~/pcfb/examples/spectra
host:spectra name$ ls
LEDBlue.txt    LEDGreen.txt   LEDRed.txt     LEDYellow.txt
host:spectra name$ less LEDGreen.txt
x       Green
350.12  9
350.48  9
350.85  12
…
850.36  9
850.67  14
850.97  10
```

You can also look at the beginning or end of each file using the `head` or `tail` command, respectively.

```
host:spectra name$ head LED*
==> LEDBlue.txt <==
x    BlueLED
350.12    4
350.48    8
…
352.67    8
353.03    10

==> LEDGreen.txt <==
x    Green
350.12    9
350.48    9
…
352.67    11
353.03    8

==> LEDRed.txt <==
x    Red
350.12    13
350.48    11
…
352.67    13
353.03    12

==> LEDYellow.txt <==
x    Yellow
350.12    8
350.48    7
…
352.67    14
353.03    13
```

In these files, the X columns are identical. The emission spectra for each color were measured under the same levels (e.g. with the same spectrometer).

Creating files with `nano`

For this lesson, we will create our script within the terminal. There are multiple ways to create a file in the terminal. This example will use `nano`. To create your file, move to your scripts directory and use the command `nano` followed by the name of your file. This will open up a blank file for you to write your script in. For now, just save the script by pressing ^o, which will prompt you to change the name of your file or save it as the same name. Then press ^x to get out of the file. Alternatively, you can just press ^x and you will be asked if you want to save the changes by typing Y for yes or N for no.

```
host:scripts name$ nano mergedfiles.py     ← Remember to save as .py
```

Make the script executable with `chmod`.

```
host:spectra name$ chmod u+x ~/scripts/mergedfiles.py
```

Note: We saved the script in the scripts directory and moved back to the spectra directory. You can change the file mode in either directory. If you are in the scripts directory, you do not have to specify the path to the script.

Command-line arguments in Python

The command `sys.argv` allows us to use command-line arguments within a Python script by including the `sys` module. Excluding the shebang line, this will be the first line of the script. We can make a simple `for` loop with `sys.argv`.

```
 #! /usr/bin/env python

import sys
for Object in sys.argv:
     print (Object)
```

Run the script in the terminal and you will see that it prints an output even though we haven't given it any arguments. `sys.argv` has a zeroth element, which is the file path for the script. Remember that the first item in a list is not actually item 1 but item 0. So, the first argument you call for `sys.argv` is actually the second item within the list (even though it is in place 1). Add arguments this time when you run the script to see what the output will look like.

```
host:spectra name$ mergedfiles.py one two "three and four"
/Users/name/scripts/mergedfiles.py
one
two
three and four
```

If you need to include a space within an argument, place that argument within quotes or each word will be interpreted as an argument. With the example above, if we had not included quotes around `"three and four"` each word (three, and, four) would have been read as a separate argument.

With this script, you can list the names of the text files of interest by including the file names as an argument using regular expressions.

```
host:spectra name$ mergedfiles.py LED*.txt
/Users/name/scripts/mergedfiles.py
LEDBlue.txt
LEDGreen.txt
LEDRed.txt
LEDYellow.txt
```

Note: You can get this same output, excluding the script path, by using the `echo` *command.*

Creating a file list

In this script, we will add a usage statement so everyone who uses this script will know its purpose. We will print the usage statement each time the list does not include additional arguments. This will also be an obvious way to know if `sys.argv` is detecting arguments or not. The next part simply uses a loop to list the files.

```
#! /usr/bin/env python

Usage= """
mergedfiles.py - version 1.0
Practice script for merging files.
Include columns of data from one file to a master data file.
Print all to screen.
Usage: mergedfiles.py > alldata.txt
"""

import sys

if len(sys.argv)<2:     ←If only one argument (file path), only print the usage statement
        print (Usage)
else:                   ←If not, save the rest of the arguments to FileList and print those
names
        FileList= sys.argv[1:]
        for InFileName in FileList:
                print (InFileName)
```

Run the script with no arguments.

```
host:spectra name$ mergedfiles.py

mergedfiles.py - version 1.0
Practice script for merging files.
Include columns of data from one file to a master data file.
Print all to screen.
```

```
Usage: mergedfiles.py > alldata.txt
```

Just as expected, the usage statement was printed because there was only a single argument (the file path). Now include an argument of the LED files within the spectra directory.

```
host:spectra name$ mergedfiles.py LED*.txt
LEDBlue.txt
LEDGreen.txt
LEDRed.txt
LEDYellow.txt
```

This time the first (or zeroth) argument was not included because the script says to print out the list of arguments from `sys.argv` starting from place 1 (the second argument). All arguments that are given to `sys.argv` are strings. If you want something to be a number or floating point you need to convert the string using `int()` or `float()`.

Commenting with `sys.stderr.write()`

When redirecting output to a new file (with the > symbol), including print statements (for troubleshooting or whatever use) can conflict with data analysis in the future. You may need to go back in the file and remove all the additional statements. The statement `sys.stderr.write()` allows you to write statements/comments throughout your script without having to save these comments in the new output file. Switch the print statements in the `mergedfiles.py` script with the `sys.stderr.write()` statement.

```
#! /usr/bin/env python

#Usage statement omitted

import sys

if len(sys.argv)<2:
        sys.stderr.write(Usage)
else:
        FileList= sys.argv[1:]
        for InFileName in FileList:
                sys.stderr.write(InFileName)
```

As mentioned before, the `sys.argv` command interprets arguments as strings unless told otherwise. Run the script and you will see how the output has changed. If you want to change the structure of the output you will need to include those specifications within the `sys.stderr.write()` statement.

```
host:spectra name$ mergedfiles.py LED*.txt
LEDBlue.txtLEDGreen.txtLEDRed.txtLEDYellow.txt
```

Note: You can add a tab (`\t`) or end of line character (`\n`) to the `sys.stderr.write()` statement if you don't want the output to print in a single line.

Looping through the file list

With the loop we have created in `mergedfiles.py` we can now add code that will manipulate these files (open, transforms, combine, etc.). In this example, our objective is to have one file with the X column (the emission spectra level) and 4 columns containing the results from each of the LED colors (blue, green, red, and yellow). One file we don't need to manipulate, for example, the file LEDBlue.txt. It already contains two of the 5 columns of data desired for the final dataset. We then need to select the column of interest from the additional three files and add those columns to the final dataset. Again, it is very important to know the structure of your data files before you merge them. The columns of data must correspond line by line. To create our final dataset, we will create a loop within the loop we have already written.

```
#! /usr/bin/env python

#Usage statement omitted

import sys

if len(sys.argv)<2:
        sys.stderr.write(Usage)
else:
        FileList= sys.argv[1:]
        FileNum = 0
        MasterList = []
        for InFileName in FileList:
                InFile = open(InFileName, 'r')
                LineNumber = 0
                RecordNum = 0

                for Line in InFile:
                        if LineNumber >0:
                                Line=Line.strip('\n')
                                if FileNum == 0:
                                        MasterList.append(Line)
                                else:
                                        ElementList = /
Line.split('\t')
                                        MasterList[RecordNum] /
+= ( '\t' + ElementList[1] )
```

```
                                        RecordNum += 1
                LineNumber += 1
        InFile.close()
        FileNum += 1
```

Note: Remember the / symbol indicates a full line of code that was too long to fit on the screen and is now in two parts. The second part does not need to be aligned under the correct loop because Python knows it is just a continuation of the previous line.

Here, we have created an empty list called MasterList that we are adding our data of interest into. We want the first two columns from the first data file so we include a variable called FileNum that will allow us to collect the first two columns of data for the first file only, or when FileNum is equal to 0. The first row is skipped and each row of data is then appended to MasterList creating a list of strings. The subsequent data files are split so each row in put into the variable ElementList. This allows us to select the second item in the list (in position 1) and ignore the other item in the list. If you want to see if the structure of the data is correct you can add a `print(MasterList)` statement to the end of the script. The output should look something like this:

```
(base) host:spectra name$ mergedfiles.py LED*.txt
['350.12\t4\t9\t13\t8', '350.48\t8\t9\t11\t7',
'350.85\t3\t12\t12\t4', '351.21\t11\t12\t10\t5',
...
'850.06\t10\t7\t28\t12', '850.36\t6\t9\t26\t5',
'850.67\t7\t14\t25\t9', '850.97\t10\t10\t25\t10']
```

To print the output in a more readable manner, add a loop that prints out each item in MasterList.

```
#! /usr/bin/env python

#Usage statement omitted

import sys

if len(sys.argv)<2:
        sys.stderr.write(Usage)
else:
        FileList= sys.argv[1:]
        FileNum = 0
        MasterList = []
        for InFileName in FileList:
                InFile = open(InFileName, 'r')
                LineNumber = 0
                RecordNum = 0
```

```
                for Line in InFile:
                        if LineNumber >0:
                                Line=Line.strip('\n')
                                if FileNum == 0:
                                        MasterList.append(Line)
                                else:
                                        ElementList = /
Line.split('\t')
                                        MasterList[RecordNum] /
+= ( '\t' + ElementList[1] )
                                        RecordNum += 1
                        LineNumber += 1
                InFile.close()
                FileNum += 1
          for Item in MasterList:
               print (Item)
```

Output:

```
(base) host:spectra name$ mergedfiles.py LED*.txt
350.12    4    9    13   8
350.48    8    9    11   7
…
850.67    7    14   25   9
850.97    10   10   25   10
```

Most data files need a header to specify what data is in each column. We will do this by adding the file name to a Header variable. This variable needs to be included within the loop that reads in the files.

```
#! /usr/bin/env python

import sys

if len(sys.argv)<2:
        sys.stderr.write(Usage)
else:
        FileList= sys.argv[1:]
        Header = "Emission_Spectra_Level" ← Naming X column
        FileNum = 0
        MasterList = []
        for InFileName in FileList:
                InFile = open(InFileName, 'r')
                                        ⊖ Adding file names to Header and removing .txt
```

```
                Header += "\t" + InFileName.strip(".txt")
                LineNumber = 0
                RecordNum = 0

                for Line in InFile:
                        if LineNumber >0:
                                Line=Line.strip('\n')
                                if FileNum == 0:
                                        MasterList.append(Line)
                                else:
                                        ElementList = \
Line.split('\t')
                                        MasterList[RecordNum] \
+= ( '\t' + ElementList[1] )
                                        RecordNum += 1
                        LineNumber += 1
                InFile.close()
                FileNum += 1
        print (Header)
        for Item in MasterList:
                print (Item)
```

Output:

```
(base) host:spectra name$ mergedfiles.py LED*.txt
Emission_Spectra_Level   LEDBlue   LEDGreen  LEDRed    LEDYellow
350.12    4    9    13   8
350.48    8    9    11   7
…
850.67    7    14   25   9
850.97    10   10   25   10
```

Avoiding future issues

Creating your script to work with different file formats and structures will save you time in the future. For example, if you were to use the `mergedfiles.py` script with data files that had 10 header lines, the first few lines of the new file will not be the data you were hoping for. To escape issues like these, we can include a new variable, LinesToSkip, and merely change this variable to be specific to the new data files you wish to merge. Additionally, using this method will avoid the script becoming confused because the same value (in our case the LineNumber value we assigned was 0) is used elsewhere within the program.

```
#! /usr/bin/env python

import sys
```

```
if len(sys.argv)<2:
        sys.stderr.write(Usage)
else:
        FileList= sys.argv[1:]
Header = "Emission_Spectra_Level"
        LinesToSkip = 1          ← Change according to file structure
        FileNum = 0
        MasterList = []
        for InFileName in FileList:
                InFile = open(InFileName, 'r')
                Header += "\t" + InFileName.strip(".txt")
                LineNumber = 0
                RecordNum = 0

                for Line in InFile:          ← Replaced the 0
                        if LineNumber > (LinesToSkip - 1):
                                Line=Line.strip('\n')
                                if FileNum == 0:
                                        MasterList.append(Line)
                                else:
                                        ElementList = \
Line.split('\t')
                                        MasterList[RecordNum] \
+= ( '\t' + ElementList[1] )
                                        RecordNum += 1
                        LineNumber += 1
                InFile.close()
                FileNum += 1
        print (Header)
        for Item in MasterList:
                print (Item)  ← Included to make sure the assumed # of files were merged
        sys.stderr.write("Converted %d files(s)" % FileNum)
```

If this script does not work with other data files, you may need to change the `.strip()` method depending on how the data is structured. You may also need to change how you read in the file. Including `'rU'` in place of `'r'` when opening the file will allow the program to work with Unix and Windows line endings.

You can save the output to a new file using the > symbol.

```
(base) host:spectra name$ mergedfiles.py LED*.txt > alldata.txt
```

References

Haddock S, Dunn C (2010) Practical computing for biologists. Sunderland (Massachusetts): Sinauer Associates.