Reading Columns of Data into Lists

Leigh J. Little SUNY Brockport December 6, 2021

Contents

1 Introduction

Reading data from files into Python (or any other language) is an important component to programming. It is very often the case that data needs to be analyzed using a different environment than the one in which it was generated. For example, you may have performed the calculations in Python and but need to use MATLAB's volumetric viewing capabilities. Also, a given situation may require reading in and processing data from thousands of files. Being able to automate this process is critical.

The easiest situation is when you have control over how the data is saved and used. You can write functions to handle this and you can customize the data output/input to make the process as easy as possible.

This is not always feasible. You sometimes need to modify your data to conform to a specific format required for the application you intend to use for post-processing. Or you may need to write a function to read in data that always is saved in a particular, very complex template. You can also encounter very strange cases; for example, you only need lines 52 and 52 from a file.

This document will discuss how to read a data file consisting of a rectangular block of numbers. In particular, the process of extracting a particular column of numbers into lists or `numpy` arrays is covered.

2 The `with open` Loop

One attractive aspect of Python is that it makes it much easier to work with reading in data than other languages. This is always a challenging topic because data files can be in different (spoken) languages or use other symbols for representing numbers (for example, in Europe a number like 1,200 is often written as 1.200.

2.1 Example 1

Download the file `sample.dat` from the course web page. The first step in reading a data file is to open it and see what it looks like. In this case, the file contains some lines of text. Type the code below into a script file and run it.

```
filename = 'sample.dat'
with open(filename) as fnum:   # Open filename with handle fnum
     for currline in fnum:     # Read the lines in fnum one at a time until the
                               #   end of the file is reached
                               #   currline is the loop index variable and contains
                               #   each line in the file as a string variable
          print(currline)
```

You should see the output

```
This is a test

of the

basic with open

environment.
```

This code reads the file line by line into a variable called `currline`. This is a string variable. The variable `fnum` is called a file handle. Everything you want to do to with the original `sample.dat` file is done through the `fnum` handle.

What makes the `with open` loop great to work with is that the loop will automatically exit when the end of the file is reached. The file will automatically close when the script terminates.

2.2 The `.split()` Method

It is rare that will you will just read in an entire line as a string and leave it that way. Normally the string needs to be broken up into pieces in some way. The `.split()` method is used to break up the string into individual portions. For example, modify the script to

```
filename = 'sample.dat'
with open(filename) as fnum:
     for currline in fnum:
         v1 = currline.split()
         print(v1)
```

You should see the output

```
['This', 'is', 'a', 'test']
['of', 'the', 'basic']
['with', 'open', 'reading']
['environment', 'in', 'Python.']
```

The `.split()` method takes a string and puts the 'words' in the string into individual list elements. The method uses the spaces in the string to determine where one word begins and ends. We call the spaces *delimiters*. A delimiter is a character that is used to indicate separate entities in a string. The most common delimiters are spaces, commas and colons.

In the event that a file uses something other than a space as the delimiter, you can send in an optional argument to the `.split()` method.

```
currline.split(',')   # uses the comma as the delimiter
currline.split(':')   # uses the colon as the delimiter
currline.split(', ')  # uses the combination of comma and space as the delimiter
```

2.3 The `.strip()` Method

Sometimes the `.split()` method alone will not sufficiently isolate the portions of a data file that you would like. The `.strip()` method is used to remove leading and trailing characters from a string. The default character is the space, but you can specify what character you want as an optional input, for example

```
.strip(',')   # removes leading/trailing commas from the string
.strip(':')   # removes leading/trailing colons from the string
.strip('a')   # removes leading/trailing letter a's from the string
```

Note that this will not affect delimiters that appear in the middle of a string.

3 Reading Columns of Numbers into `numpy` Arrays

The easiest situation for reading data from a file is when it fits the data files we have examined to this point in the courses. These are files where the data consists of columns of numbers and each column has the same number of rows (*i.e.*, there are no missing entries in any of the columns). We will develop a process for reading a particular column of such a data file.

Consider the file `data1.dat`. This file contains 5 columns of integers. We would like to read in the second column into a `numpy` array. Because of the regular and predictable structure of the array, this can easily be done.

Start with the code below:

```
filename = 'data1.dat'
with open(filename) as fnum:
     for currline in fnum:
         v1 = currline.split()
         print(v1)
```

You should see the output

```
['1', '2', '3', '4', '5']
['5', '4', '3', '2', '1']
['1', '2', '3', '4', '5']
['5', '4', '3', '2', '1']
['1', '2', '3', '4', '5']
['5', '4', '3', '2', '1']
['1', '2', '3', '4', '5']
['5', '4', '3', '2', '1']
```

Note how each line is a list that contains each value in the line as a string. To complete the process, we need to extract the second value from each of these strings and `append` it to a list.

Change the code to

```
colnum = 2
L2 = []
filename = 'data1.dat'
with open(filename) as fnum:
     for currline in fnum:
         v1 = currline.split()
         L2.append(int(v1[colnum-1]))
print('L2 = ',L2)
```

Note that we need `colnum-1` of `v1` because the numbering of elements starts at 0. The output of the code is simply the second column of the file in the list `L2`.

```
L2 =  [2, 4, 2, 4, 2, 4, 2, 4]
```

We can leave the list like this, but in most cases if we are going to do computing with this, it would be better to convert it to a `numpy` array. To do this, import the `numpy` library into the code and add the line below to the end of the file:

```
L2 =  np.array(L2)
```

You should see that `L2` is now a `numpy` array.

3.1 Reading More Columns

This process can be extended to handle as many columns as you wish (but also, it doesn't work too well if there are many columns and you need all of them). Suppose you wanted columns 2 and 5. Then the code below will work:

```
import numpy as np
filename = 'data1.dat'
c1 = 2
c2 = 5
L2 = []
L5 = []
with open(filename) as fnum:
     for currline in fnum:
         v1 = currline.split()
         L2.append(int(v1[c1-1]))
         L5.append(int(v1[c2-1]))
L2 = np.array(L2)
L5 = np.array(L5)
```

3.2 A More Complex Example

Suppose instead you have the file `data2.dat`. This is the same file as `data1.dat` except there are both commas and spaces between the numbers. Notice that if we do

```
filename = 'data2.dat'
with open(filename) as fnum:
     for currline in fnum:
         v1 = currline.split()
         print(v1)
```

this will not give the correct results. This is because the default delimiter is the space. As a result, the commas are part of the partitioned strings.

```
['1,', '2,', '3,', '4,', '5']
['5,', '4,', '3,', '2,', '1']
['1,', '2,', '3,', '4,', '5']
['5,', '4,', '3,', '2,', '1']
['1,', '2,', '3,', '4,', '5']
['5,', '4,', '3,', '2,', '1']
['1,', '2,', '3,', '4,', '5']
['5,', '4,', '3,', '2,', '1']
```

Changing the delimiter to as string and a space also does not work

```
filename = 'data2.dat'
with open(filename) as fnum:
     for currline in fnum:
         v1 = currline.split(', ')
         print(v1)
```

This gives the output

```
['1', '2', '3', '4', '5\n']
['5', '4', '3', '2', '1\n']
['1', '2', '3', '4', '5\n']
['5', '4', '3', '2', '1\n']
['1', '2', '3', '4', '5\n']
['5', '4', '3', '2', '1\n']
['1', '2', '3', '4', '5\n']
['5', '4', '3', '2', '1']
```

In this case, the `\n` newline (or return) character becomes part of the string. This happens because there are no extra spaces at the ends of the lines in the file.

In order to get the values into arrays, we need to use a combination of the `.strip()` and `.split()` methods. We will split based on the spaces then strip off the commas and new lines.

```
filename = 'data2.dat'
with open(filename) as fnum:
     for currline in fnum:
         v1 = currline.split(' ')
         for i in range(len(v1)):
             v1[i] = v1[i].strip('\n').strip(',')
         print(v1)
```

Note that we need two loops. We can't apply the `.strip` methods to `v1` as a whole because `v1` is not a string. It is a list of strings, so we need to loop over each individual element of `v1` to strip the excess characters. This gives the output that we need.

```
['1', '2', '3', '4', '5']
['5', '4', '3', '2', '1']
['1', '2', '3', '4', '5']
['5', '4', '3', '2', '1']
['1', '2', '3', '4', '5']
['5', '4', '3', '2', '1']
['1', '2', '3', '4', '5']
['5', '4', '3', '2', '1']
```

Now the process can continue as in the previous example.

```
import numpy as np
filename = 'data2.dat'
colnum = 2
L2 = []
with open(filename) as fnum:
     for currline in fnum:
         v1 = currline.split(' ')
         for i in range(len(v1)):
             v1[i] = v1[i].strip('\n').strip(',')
         L2.append(int(v1[colnum-1]))
L2 = np.array(L2)
print('L2 = ',L2)
```

This gives the same `L2` array as before.