Programming with Python 4

NITARP 2013: SHIPs

Babar Ali

Topics

- Input from text files
- Output to text files and screen.
- Try, except blocks and error handling
- Functions & Libraries

INPUT

Files on disk

- We will focus on ASCII (as opposed to binary) files here.
- Libraries (such as numpy) add significantly easier to use functions to handle files than core python.
    - Try to use them.
- The following lines of code show one way to read the file. We will expand on each command.

Basic code:

```
>>> file = "/Users/babar/file.txt"   # Tell python which file.
>>> ou = open(file,"r")              # Have python "open" the file for access.
>>> line=ou.readline()               # Read the first line from our file.
>>> ou.close()                       # Close the file.  No more access.
>>> print line                       # Print what we just read.
```

A detailed look

```
>>> file = "/Users/babar/file.txt"
```

- The string variable 'file' is used to define the full directory path and name of the ASCII text file.
- If you don't tell python / spyder the full pathname, it will use the current working directory.
- Since, you may not always remember or wish to use a directory different than the current working directory, it is best
- Always define the full path name so there is no ambiguity.
- If you plan to use the same directory for access to many files and for output, also define the directory in a separate variable.

```
>>> dirname = "/Users/babar/"
>>> file1 = dirname+"Jphot.txt"
>>> file2 = dirname+"Kphot.txt"
```

A detailed look (cont.)

```
>>> ou = open(file,”r”)
```

- This line calls a core python function ‘open’ and gives it two arguments:
  - “file” is the variable that is pointing to the file we wish to use.
  - “r” tells python to **r**ead from the file.
- You can specify the file name directly instead of first defining it as a string.
  - But, take care to put quotes around the name.
- Here is what else you can tell python to do with files instead of “r”ead:
  - “w” tells python you wish to **w**rite to a file. If the file already exists, it will be deleted first and recreated. So, be careful. Its an easy mistake to wipe out existing file.
  - “a” tells python to **a**ppend to an existing file.
- Once opened, the file will stay opened until closed.
- The variable ‘ou’ contains the link to the opened file. You need it to access the linked file.

What to do with 'line'?

- Recall: we stored the first line in a variable called 'line'.
- This is a string variable and can be manipulated just like any other string variables.
- For file I/O in particular, the following provide some useful functions:

```
>>> line.strip()       # Remove the trailing '\n' end of line character as well as spaces.
>>> line.split()       # Split the contents of the line using space ' ' as the delimiter.
>>> line.split(',')    # Same as above, except use comma ',' as delimiter.
```

Example:

```
>>> line = "100  34.2345  -5.2344 1"      # This is our line.
>>> line.split()       # produces the following output
['100', '34.2345', '-5.2344', '1']
>>> # A string vector with four values – the original 4 numbers in the line separated by space ' '
```

For more help

- http://python4astronomers.github.io/files/asciifiles.html

A more complete example

- Read and parse a text file containing 5 columns and two header line.

```
>>> # Our input file looks like this:
>>> # ID,    ra,          dec,        flux,        comments
>>> #     ,   (deg),       (deg),     (mJy),
>>> # 1 , 35.2345  , -5.1234  ,  100.0 ,    There is a bright filament nearby.
>>> # 2 , 35.5436  , -5.5567  ,  120.0 ,
>>> # 3 , 35.8934  , -5.9832  ,  150.0 ,    Part of a binary pair.
>>> # 4 , 36.52  , -6.1654  ,  102.0 ,      Value is from Scott.
>>> iu=open(myFile,"r")
>>> header1=ou.readline()
>>> header2=ou.readline()
>>> id = []
>>> ra = []
>>> dec = []
>>> flux = []
>>> comment = []
>>> line=ou.readline()
```

Example continued.

```
>>> while line!="":
>>>     words = line.split(',')
>>>     id.append( int(words[0]) )
>>>     ra.append( float(words[1]) )
>>>     dec.append( float(words[2]) )
>>>     flux.append( float(words[3]) )
>>>     comment.append( words[4] )
>>>     line=iu.readline()
>>> iu.close()
```

Using numpy

- numpy provides two functions to read ASCII files. We will use genfromtxt
- Functions automatically perform a number of the steps.
- This makes programming simpler to understand and less prone to errors.

Our complete example, now in numpy:

```
>>> import numpy as np
>>> data = np.genfromtxt(myFile, dtype=('i','f','f','f','S20'),\
                         names="id,ra,dec,flux,comment", \
                         skip_header=3, usecols=(0,1,2,3,4),delimiter=",")
```

A closer look at numpy genfromtxt

- *myFile* is the name of the file to read from.
- *dtype* specifies the type of data. One per column, in parenthesis and quotes, as shown.
- *names* tells python what name to use for each column.
    - NOTE: Not related to names in file itself.
- *skip_header =* tells how many header lines to skip.
- *usecols* says read data from these columns.
- *delimiter=* tells what separates data columns

The output

- data contains data on all columns and accessed by the assigned name.
- data["id"] # All elements of id
- data["ra"][0] # The first element of ra
- data["dec"][0:2] # The first & 2nd element
- data["flux"][:] # All elements of flux
- data["comments"]

The output (cont.)

- If names was omitted during the call, the default name of the columns is used.
  - ➢ The default name is 'f#', where # stands for 0, 1, 2, 3 ....
- The extracted columns in the output are numpy arrays of the type specified in dtype.
  - For example, data["id"] is a numpy integer array.

What else can you do with genfromtxt?

Option	
comment="#"	Treat the row/line as a comment If the line begins with whatever character is specified in quotes. In this case, '#'.
autostrip=0 or autostrip=1	Whether to strip white spaces from the variables. 1=yes, 0=no.
skip_footer=	The number of lines to skip at the end of the file.

OUTPUT

Output is just as easy

We have already seen

print

Which prints to the screen.

Now we will use it to print to a file.

Basic code:

```
>>> file = “/Users/babar/output.txt”      # Tell python which file.
>>> ou = open(file,”w”)                  # Have python “open” the file for access.
>>> print >>ou, "Hello File"             # Read the first line from our file.
>>> ou.close()                           # Close the file.  No more access.
```

A detailed look

```
>>> file = "/Users/babar/output.txt"
```

- As for input, the string variable 'file' is used to define the full directory path and name of the ASCII text file.
- If you don't tell python / spyder the full pathname, it will use the current working directory.
- Since, you may not always remember or wish to use a directory different than the current working directory, it is best
- Always define the full path name so there is no ambiguity.

A detailed look (cont.)

```
>>> ou = open(file,"w")
```

- This line calls a core python function 'open' and gives it two arguments:
  - "file" is the variable that is pointing to the file we wish to use.
  - "w" tells python to **w**rite to the file.
- You can specify the file name directly instead of first defining it as a string.
  - But, take care to put quotes around the name.
- You can also use:
  - "a" to **a**ppend to an existing file.
- Once opened, the file will stay opened until closed.
- The variable 'ou' contains the link to the opened file. You need it to access the linked file.

Writing to the file

```
>>> print >>ou, "Hello File"
```

- Use the normal python print statement to write to the file.
- The **>>ou** construct added to the print simply tells it to redirect the result of the printing to **ou**.
- All formatting rules for printing apply here as well.

Full example

```
>>> # Using data read earlier in variable data with numpy's genfromtxt
>>> ou=open(myOutputFile,"w")
>>> print >>ou, "This is a header line"
>>> print >>ou, "This is another header line"
>>> nlines = len( data["id"] )
>>> for i in range(nlines):
...         print >>ou, "%4i,  %8.3f,   %8.3f,   %8.3f,   %s" % ( data["id"][i], data["ra"][i], \
...                                     data["dec"][i], data["flux"][i], data["comment"][i] )
>>> ou.close()
```

Using numpy:

```
>>> import numpy as np
>>> np.savetxt(myOutputFile,data,fmt="%4i %8.3f %8.3f %8.3f %s",delimiter=",",\
...                 header="This is first line\nThis is 2nd line", comments="#")
```

CATCHING ERRORS

Try and if fail then do something else

```
>>> # python allows a way to catch errors and/or problems:
>>> try:
...       command1   # Run some commands.
...       command2
>>> except Exception, e:
...       print "Your commands did not work.  Here is why, maybe"
...       print e.message
```

The ***try, except*** syntax allows you to test the execution of any command. If any of the commands in the ***try:*** block generates a python error, then the statements under the ***except:*** block are executed.

Generally, you put "safe" print statements in the except block to let you figure out what may have happened, but there are always exceptions.
YES: if you have an error in the except block, you may crash the program.

FUNCTIONS

The basic building blocks of code

- Functions:
  - Execute a set of instructions when called.
  - Tidy up code by modularizing it.
  - Should always be used when the same algorithms (with slight parameter differences) are repeated.
- A library is a set of themed functions. E.g. numpy for numerics-related python functions

def:ining Functions.

```
>>> # Creating a function is easy.
>>> def myFunction(input,optionA=1,optionB=2):
...     command1   # Run some commands.
...     command2
...     return result
```

The ***def*** statement tell python we are defining a function.
The name of the function follows ***def*** call.
Mandatory input parameters are specified without the = syntax.
Optional parameters use the = syntax, where the value to the right of = is the default value, if no other value is specified.
Commands are indented.
The ***return*** statement returns the result. Please return only one variable or value. We will encounter how to return multiple values later.

A simple example.

```
>>> # A function to perform .
>>> def myMathTool(a,b,operation="+"):
...     if operation=="+":
...         result = a+b
...     elif operation=="-":
...         result = a-b
...     elif operation=="*":
...         result = a*b
...     elif operation=="/":
...         if b!=0.:
...             result = a-b
...         else:
...             print "I will not divide by 0.  You should know better"
...             print "returned value is 0"
...             result = 0
...     else:
...         print "Operation not understood "+operation
...         print "returned value is 0"
...         result = 0
...     return result
```

Executing functions

- In spyder:
  - Read or edit your function in the editor,
  - Then choose 'Run' to define it.
- Command line python
  - execfile("/path/fileWithFunction.py") defines the function.
- simply call the function by its name and the parameters, if needed.

```
>>> # Creating a function is easy.
>>> myMathTool(10.,20.,operation="/")
>>> c = myMathTool(10.,201.,operation="*")
```