Programming with Python 3

python for non-programmers

Babar Ali

Topics

- Problems from last week
- Scripting & Some useful python commands
- if / then / else blocks
  - Basic concepts
- Loops
  - and indentations.
- Simple Input/Output
  - (time permitting)

SOME USEFUL PYTHON COMMANDS

printing stuff

- NOTE: The python interpreter does not actually require an explicit print statement and echoes the result anyway.
- However, “print” and others commands provide additional functionality.

```
>>> print 2048+2048  # That’s all to it.
>>> print “2048+2048”  # prints strings.
>>> a=10
>>> print a  # prints the variable ‘a’
```

using sys.stdout.write

```
>>> import sys # Load up the system routines.  Assumes you mean ‘as sys’.
>>> sys.stdout.write(‘some text’)      # prints ‘some text’ to the screen.
>>> sys.stdout.write(‘some text\n’)  # can you spy the difference>
>>> sys.stdout.write(str(a)+’\n’)  # Only works on strings.  Variables must be converted
                                    # to strings.  ‘print’ is usually better.
```

Formatting the printed output

- Many tricks available in python to format output.
- We will use the quotes (“”)syntax.

Basic structure:

```
>>> print “Result = %4i and %3.1f” % (2048+2048,2.2)
```

“” contain unformatted characters and the formatting instructions.

Followed by % character

Followed by what’s to be printed in parenthesis () and separated by comma. The number of formatting instructions must match the number of items (or variables) to be printed, or python gets confused.

```
>>> print “%s” % (“Hello Formatted Printing”)  #  Silly but works
>>> print “%-30s %-30s” (“Hello”,”World”)
```

Formatting instructions

%Width.Digits_after_Decimal#

 # is one of:

i	integer
f	float
s	string

Example	Outcome
%9.2f	Printing takes up 3 characters (width) for a floating point number. If the number does not require all 9 characters, it is right-justified. 2 digit are printed after the decimal point. Ex. 324453.98 or 3.24 or -123.23 Note: The decimal and +/- sign are included in the width.
%f	Don't specify anything fancy. Allow python to use its default printing.
%6i	Integer number to be printed using up 6 character space.
%6.6i	The integer number is preceded by 0s to fill up all 6 character spaces.
%-20s	Print a left-justified string using up 20 characters.

The full scary list

Conversion	Meaning
`'d'`	Signed integer decimal.
`'i'`	Signed integer decimal.
`'o'`	Signed octal value.
`'u'`	Obsolete type – it is identical to `'d'`.
`'x'`	Signed hexadecimal (lowercase).
`'X'`	Signed hexadecimal (uppercase).
`'e'`	Floating point exponential format (lowercase).
`'E'`	Floating point exponential format (uppercase).
`'f'`	Floating point decimal format.
`'F'`	Floating point decimal format.
`'g'`	Floating point format. Uses lowercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.
`'G'`	Floating point format. Uses uppercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.
`'c'`	Single character (accepts integer or single character string).
`'r'`	String (converts any Python object using *repr()*).
`'s'`	String (converts any Python object using `str()`).
`'%'`	No argument is converted, results in a `'%'` character in the result.

range

Range(start, stop, increment)

- Similar to numpy.arange() covered last week, but available in core python.

pass

- Does nothing.
- Good for “filling in” or simply completing a loop (some people’s preference).

CONDITIONING: IF/THEN/ELSE

Controlling the Script

- If / then / else constructs allow different sets of python commands to run depending whether certain conditions (criterion) are met or happen to be.
- It is used, for example, like this:
  - Only compute the logarithm if the value is positive.
  - If the user is Peggy, print the state is Illinois
  - If a 2MASS listed star has at least two matching objects in the IRAS list, take the one closest in brightness to predicted value.

Boolean logic in Python

Operator	Meaning
X == Y	X is equal to Y. Note the double =.
X < Y	X is less than Y
X > Y	X is bigger than Y
X != Y	X is not equal to Y
X <= Y	X is less than or equal to Y
X >= Y	X is greater than or equal to Y

In Python

All the previous examples.

```
>>> if value>0.0: # Basic construct for start.
>>>     logValue = np.alog10(value)        # Note: the subsequent text is indented.
>>> pass # ibidy, ibidy that's all folks.  End of construct.
```

```
>>> if user==“Peggy”: # NOTE, colon : is always at the end of these constructs.
>>>     print “The state is Illinois”
```

```
>>> if nMatched>=2: # NOTE, numpy is imported as np
>>>     deltaMag = np.abs( predicted_K_Mag – k2mass[currentStar] )
>>>     index = np.where( deltaMag==np.min(deltaMag) )
>>>     iras_match[currentStar] = iras_ID[index]
```

What about else?

- If the condition is not met, then you can, optionally, execute a different set of statements.

```
>>> if value>0.0: # Basic construct for start.
>>>     logValue = np.alog10(value)        # Note: the subsequent text is indented.
>>> else:  # if the condition is not met.
>>>     print "ERROR: cannot take log of a negative value." # Indentation still  required
                                                            # for else.
```

LOOPS

What are loops?

- Allow users to repeat calculations.
- **PYTHON**: Indentation is necessary to identify which calculations are part of the loop (are to be repeated).

For loops

```
>>> for i in [0,1,2,3,4]:  # Basic construct for start.
>>>     print i         # Note: the subsequent text is indented.
>>>     sqEye = i*i  # Do something
>>>     print "i is %2i and  i squared is %2i " % (i, sqEye)
>>> print "The loop is finished" # Un-indent to disengage from the loop.
```

- Remember to end the for loop command with colon, :
- Put in as many calculations as desired in the loop.
- In the above example, 3 commands are repeated a total of 5 times. The value of 'i' increments from 0 to 4 sequentially in the 5 loops.

More For Loops

```
>>> for i in range(20):       # Will increment 'i' from 0 to 19.
>>>     for j in range(12):   # Loop within a loop is allowed.
>>>         k = i*j   # Each loop requires its own indentation.
>>>         k = k+1 # loops are frequently used to create counters like this.
>>>     print "The inner loop is finished" # Un-indent to continue
                                          # programming at the previous loop level.
```

Loops arent' just for numbers.

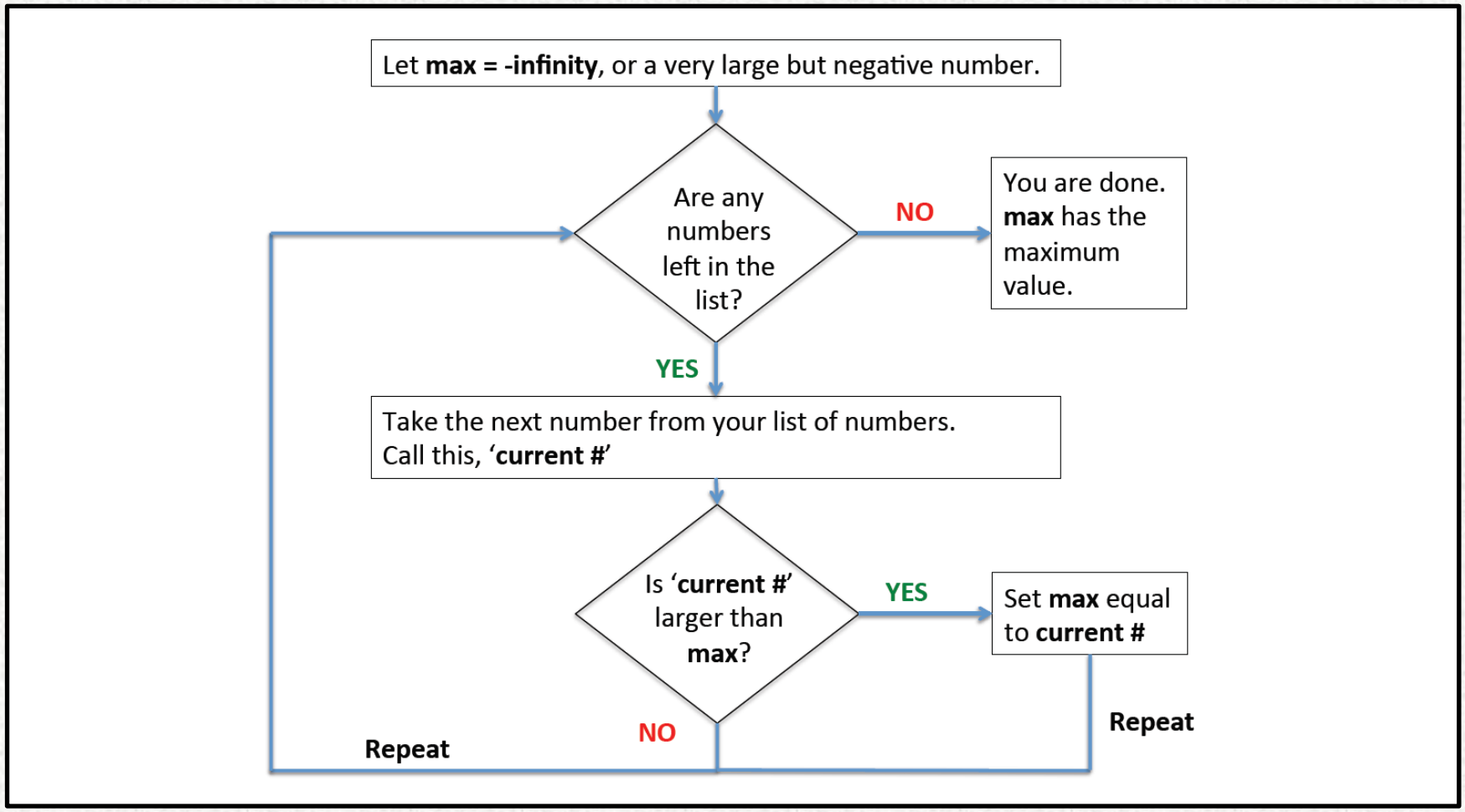
```
>>> for name in ["Carol","Lynn","Melissa","Peggy"]:  # try it
>>>     print "Hello NITARP participant %s" % (name)
 or
>>> for xval in [123.4, 234.4, 124.2]:  # Can even loop on floating point values.
>>>     print xval*xval
```

While Loops

- While a condition is TRUE repeat the calculations.
  - The "For" loops have set number of repetitions, "while" loops continue until the ending condition is met.
  - **WARNING**: if incorrectly programmed, you may never get out of this loop.

```
>>> i = 10.3  # An example of while loop.
>>> target = 100.8
>>> counter = 1
>>> while i<target:
>>>     i = i+2.3 # Note: Indentation still applies.
>>>     counter=counter+1
>>> print "It took %i repetitions to reach %f" % (counter,target)
```

Hands-on: Find the maximum



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.