Programming with Python 3

NITARP 2013: SHIPs

Babar Ali

Topics

- Problem solving aboard SHIPs
- Scripting & Some useful python commands
- if / then / else blocks
    - Basic concepts
- Loops
    - and indentations.

SHIPs

PROBLEM SOLVING

What to do in case of trouble?

- Try to capture as much of the output and written messages as possible.
    - Take a screen capture, or
    - Cut-and-paste the text, or
    - Re-write the text in your message
- And, include as much information as possible about your machine, your data, your lines of code.

SCRIPT WRITING & SOME USEFUL PYTHON COMMANDS

Scripts

- Series of python commands = the python program.
    - (Usually) One command per script line.
    - But, remember, python allows nesting of commands.
- Only practical way to write complex logical operations on data.
- Saved in ASCII text files so the same commands can be re-used or modified.
    - Use a .py extension for your scripts.

Writing, Editing, Saving, Storing python Scripts Using Spyder

DEMO SCRIPT WRITING IN SPYDER

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

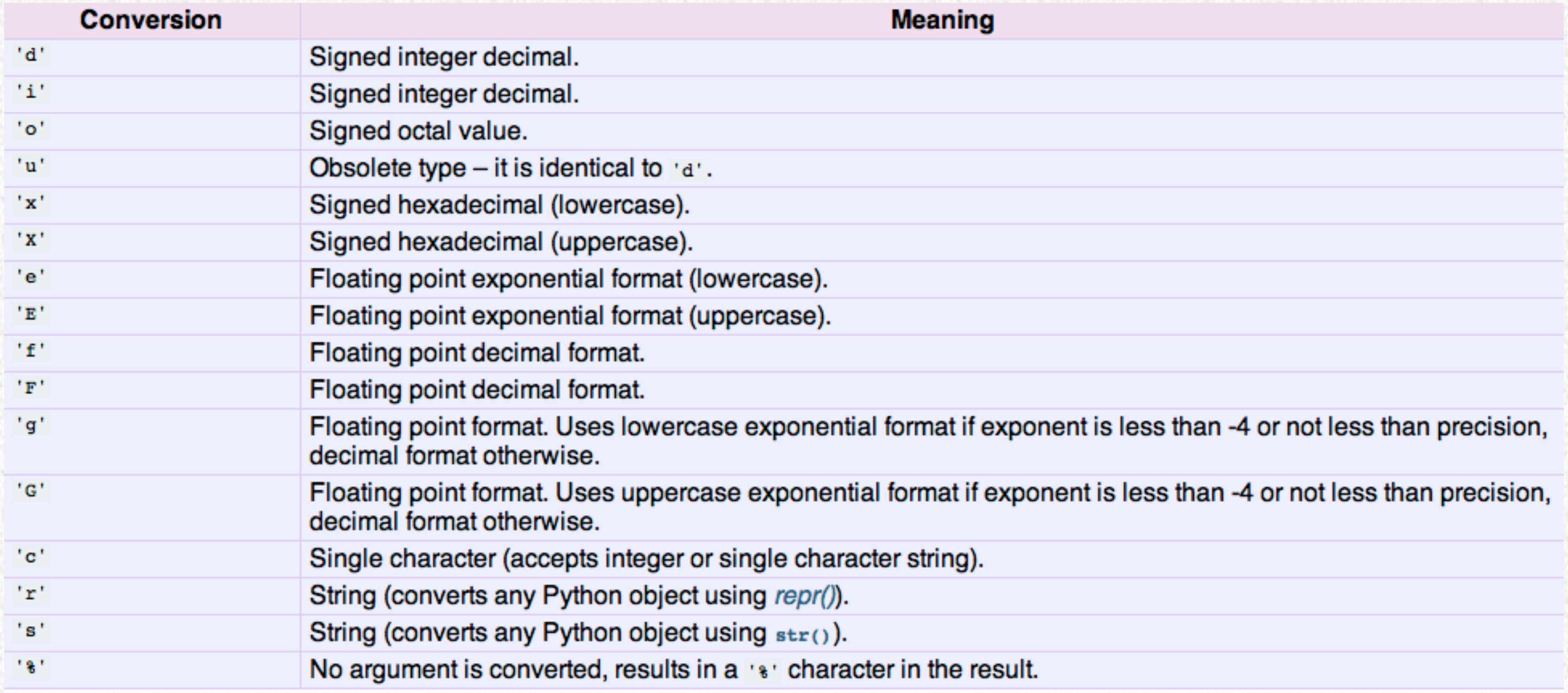
```
%Width.Digits_after_Decimal#
```

# 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 :
- 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-indentation will  to continue
                                           # programming at the previous loop level.
```

Loops arent’ just for numbers.

```
>>> for name in [“Carol”,”Lynn”,”Melissa”,”Peggy”]:  # try it
>>>     print “Hello SHIPs 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)
```