Introduction to software

John Kitchin

2012-08-28 Tue

Contents

If you need a total introduction to python, you might start here: http://www.greenteapress.com/thinkpython/thinkpython.pdf

1 Python

Download the Enthought python distribution (http://www.enthought.com/repo/.epd_academic_installe
This distribution bundles a lot of additional python modules such as numpy and scipy that we will be using later.

1.1 Simple calculations

Python is a lot like Matlab for simple math. A good overview of the basic operators can be found at http://www.tutorialspoint.com/python/python_basic_operators.htm

Here are some simple examples

```
print 2+3
print 4-6
print 2*8
print 4.0 / 6.0
```

```
5
-2
16
0.666666666667
```

Note some tricky issues with division. Python distinguishes between integer division and float division. In the first line we have integer division, where the remainder is discarded and an integer is returned. If any number is a float (indicated by a decimal or because it is converted to a float) then a float is returned.

```
print 2/3
print 2./3.
print 2/3.
print 2/float(3)  # the float function casts the integer to a float
```

```
0
0.666666666667
0.666666666667
0.666666666667
```

```
print 2*3
print 2*3.0
```

```
6
6.0
```

We can also do powers with **

```
print 2**3
print 2**0.5
print 2^4     # Binary XOR operator!
```

```
8
1.41421356237
6
```

The modulus operator (%)divides the left hand operand by the right hand operand and returns the remainder.

```
print 5 % 4
print 5. % 4.
```

```
1
1.0
```

1.2 Formatted printing

http://docs.python.org/library/stdtypes.html#string-formatting-operations

We will usually want to print more than a number, e.g. some descriptive text and the number. We also will want to format numbers so we do not see 9 decimal places all the time. We use string formatting for that. Here are some typical examples.

In a string we can specify where to put numbers with positional arguments like {0}. That says take the first argument (python starts counting at zero) and put it in place of {0}.

```
a = 4.5 + 3
print 'The answer is {0}'.format(a)
```

```
The answer is 7.5
```

We can have more than one number to format like this.

```
a = 5**3
b = 23
print 'a = {1} and b = {0}'.format(b,a)
```

```
a = 125 and b = 23
```

Alternatively, we can use named arguments to specify the values. It is your choice which one to do. Named arguments require more typing, but are easier to understand.

```
a = 5**3
b = 23
print 'a = {ans0} and b = {ans1}'.format(ans0=a,
                                          ans1=b)
```

```
a = 125 and b = 23
```

To do formatting, we need additional syntax. We use {i:format} to specify how the value should be formatted. Here we show how to specify only three decimal places on a results. See this link for a lot more details of formatting strings.

```
a = 2./3.
print 'a = {0}'.format(a)
print 'a = {0:1.3f}'.format(a)
```

```
a = 0.666666666667
a = 0.667
```

1.3 Data types

Numeric types http://docs.python.org/library/stdtypes.html#numeric-types-int-float-long-complex

strings http://docs.python.org/library/stdtypes.html#string-methods

1.3.1 lists/tuples

Lists and tuples are similar in that they are both sets of data. A list is delimited by [] (square brackets) and a tuple is delimited by () (parentheses). The difference between them is a list can be changed after it is created (it is mutable), but a tuple cannot (it is immutable).

```
# short list example
a = [1, 2, 3, 4] # a list
print a
print len(a)
print a[0] # first element
print a[-1] # last element
print a[3]  # also last element
print 2*a   # surprise!!!
```

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

We can create a list with the `range` command:

```
a = range(4)
print a

b = range(4,10)
print b

print a + b  # surprise again!!!
```

```
[0, 1, 2, 3]
[4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Note that algebraic/math operations are not defined for lists the way they are for Matlab. We have to use `numpy.array` for that, which we will see later.

```
# short list example
a = [1, 2, 3, 4] # a list
print a
a[1] = 56 # change the value of 2nd element
print a
```

```
[1, 2, 3, 4]
[1, 56, 3, 4]
```

Tuples are like lists e

```
a = (1,2,3,4)
print len(a)
print a[0]
print a[-1]
# a[1] = 56 this is not allowed!
```

```
4
1
4
```

1.3.2 dictionaries

http://docs.python.org/library/stdtypes.html#mapping-types-dict

Dictionaries provide labeled access to data. A dictionary is defined by {key:value} (curly brackets). Almost anything can be a key, except a list.

```
d = {'key1':23,
     'key2':'test',
     5:[2,3],
     (3,4):'tuple value'}

print d['key1']
print d['key2']
print d[5]
print d[(3,4)]

defaultvalue = None
print d.get('invalidkey', defaultvalue) # default value for nonexistant keys
```

```
23
test
[2, 3]
tuple value
None
```

1.4 Conditional statements

conditional operators http://docs.python.org/library/stdtypes.html#comparisons

Python has the standard conditional operators for testing if a quantity is equal to (==), less than (¡), greater than (¿), lessthan or equal to (¡=) greater than or equal to (¿=) and not equal (!=). These generally work on numbers and strings.

```python
print 4 == 4.
print 'a' != 'A'
print 4 > 3
print 4 <= 3
print 'a' < 'b' # hmmm....
```

```
True
True
True
False
True
```

We use these conditional operators to determine whether conditional statements should be run or not.

```python
a = 4
b = 5

if a < b:
    print 'a is less than b'
```

```
a is less than b
```

In this next example we use an `else` statement. Note the logic is not complete, if a=b in this case, we would get the statement “a is less than b” printed.

```python
a = 14
b = 5

if a > b:
    print 'a is greater than b'
else:
    print 'a is less than b'
```

```
a is greater than b
```

Here is a more complete logic that uses `elif` to add an additional logic clause.

```python
a = 4
b = 4
if a > b:
    print 'a is greater than b'
```

```
elif a == b:
    print 'a is equal to b'
else:
    print 'a is less than b'
```

```
a is equal to b
```

Finally, to illustrate that the first conditional statement that evaluates to True is evaluated, consider this example:

```
a = 4
b = 4
if a > b:
    print 'a is greater than b'
elif a >= b:
    print 'a is greater than or equal to b'
elif a == b:
    print 'a is equal to b'
elif a <= b:
    print 'a is less than or equal to b'
else:
    print 'a is less than b'
```

```
a is greater than or equal to b
```

1.5 Loops

http://docs.python.org/tutorial/datastructures.html#looping-techniques for while/break/continue enumerate, zip

```
for i in [0,1,2,3]:
    print i


for i in range(4):
    print i
```

```
0
1
2
3
0
1
2
3
```

1.6 functions

http://docs.python.org/tutorial/controlflow.html#defining-functions

We can define functions with the `def` statement, and specify what they `return`

```
def myfunc(x):
    return x*x
print myfunc(3)
print myfunc(x=3)
```

```
9
9
```

1.7 Modules

http://docs.python.org/tutorial/modules.html

The default Python environment has minimal functionality. We can `import` additional functionality from modules. The full standard library is documented at http://docs.python.org/library/. It is not likely you will use everything there, but it is helpful to be familiar with what is available so you do not reinvent solutions.

We import modules, and then we can access functions inthe module with the `.` operator.

```
# list contents of current directory
import os
for item in os.listdir('.'):
    print item
```

```
L01-intro-molecular-simulations.pptx
L01-intro-to-dft.pdf
L02-intro-software.html
L02-intro-software.org
L02-intro-software.pdf
L02-intro-software.tex
L02-plot1.png
```

You can import exactly what you need also with the `from/import` syntax

```
# list contents of current directory
from os import listdir
for item in listdir('.'):
    print item
```

```
L01-intro-molecular-simulations.pptx
L01-intro-to-dft.pdf
L02-intro-software.html
L02-intro-software.org
L02-intro-software.pdf
L02-intro-software.tex
L02-plot1.png
```

Finally, you can change the name of a module. This may be done for readability, or to shorten the amount of typing.

```python
# list contents of current directory
import os as operating_system
for item in operating_system.listdir('.'):
    print item
```

```
L01-intro-molecular-simulations.pptx
L01-intro-to-dft.pdf
L02-intro-software.html
L02-intro-software.org
L02-intro-software.pdf
L02-intro-software.tex
L02-plot1.png
```

1.7.1 Some common standard modules

http://docs.python.org/tutorial/stdlib.html os, sys, glob, re

1.8 Error handling

http://docs.python.org/tutorial/errors.html

Errors happen, and when they do they usually kill your script. Sometimes that is not desirable, and it is nice to catch errors, handle them, and keep on going. When errors occur in python, an Exception is raised. We can use `try/except` code blocks to try some code, and then respond to any exceptions that occur.

```python
try:
    1/0
except ZeroDivisionError, e:
    print e
    print 'an error was found'
```

```
integer division or modulo by zero
an error was found
```

1.9 Scientific and numerical python

1.9.1 numpy

http://docs.scipy.org/doc/numpy/reference/

```
import numpy as np
a = np.array([1,2,3,4])

print a*a          # element-wise operation

print np.dot(a,a)  # linear-algebra dot product
```

```
[ 1  4  9 16]
30
```

Numpy defines lots of functions that operate element-wise on arrays.

```
import numpy as np
a = np.array([1, 2, 3, 4])
print a**2
print np.sin(a)
print np.exp(a)
print np.sqrt(a)
```

```
[ 1  4  9 16]
[ 0.84147098  0.90929743  0.14112001 -0.7568025 ]
[  2.71828183   7.3890561   20.08553692  54.59815003]
[ 1.          1.41421356  1.73205081  2.        ]
```

```
import numpy as np
a = np.array([1, 2, 3, 4])
print a.min(), a.max()
print a.sum()  # sum of elements
print a.mean() # average
print a.std()  # standard deviation
```

```
1 4
10
2.5
1.11803398875
```

- Linear algebra

  `numpy.linalg` provides a lot of the linear algebra functionality we need. See http://docs.scipy.org/doc/numpy/reference/routines.linalg.html for details of all the things that are possible. For example, given these linear equations:

  x + y = 3 x - y = 1

  we can represent these equations in matrix form $Ax = b$ where

  $A = [11$
  $1 - 1]$ and $b = [3$
  $1]$. Finally we solve them as:

```
import numpy as np
import numpy.linalg as la

A = np.array([[1, 1],[1, -1]])
b = np.array([3, 1])
print la.solve(A, b)
```

```
[ 2.  1.]
```

  You might be familiar with the following solution:

  $x = A^-1b$

  We can also compute that:

```
import numpy as np
import numpy.linalg as la

A = np.array([[1, 1],[1, -1]])
b = np.array([3, 1])

print np.dot(la.inv(A), b)
```

```
[ 2.  1.]
```

  Finally, we can do linear least squares easily. Suppose we have these three equations, and two unknowns:

  x + y = 3 x - y = 1 x - y = 0.9

```python
import numpy as np
import numpy.linalg as la

A = np.array([[1, 1],
              [1, -1],
              [1, -1]])
b = np.array([3, 1, 0.9])

[x, residuals, rank, s] = la.lstsq(A,b)
print x
```

```
[ 1.975  1.025]
```

- Polynomials
  numpy can do polynomials too. We express polynomials by the coefficients in front of the powers of x, e.g. $4x^2 + 2x - 1 = 0$ is represented by [4, 2, -1].

```python
import numpy as np
p = [4, 2, -1]
print np.roots(p)
```

```
[-0.80901699  0.30901699]
```

```python
import numpy as np
p = [4, 2, -1]
print np.polyder(p) # coefficients of the derivative
print np.polyint(p)
```

```
[8 2]
[ 1.33333333  1.          -1.           0.        ]
```

We can also readily evaluate polynomials at specific points:

```python
import numpy as np
p = [4,2,-1]
print np.polyval(p,[0, 1, 2])
```

```
[-1  5 19]
```

Polynomials are very convenient functions to fit to data. the `numpy.polyfit` command does this, and returns the coefficients.

```
import numpy as np
x = [0, 2, 3, 4]
y = [1, 5, 7, 9]
p = np.polyfit(x, y, 1)
print 'slope = {0}\nintercept = {1}'.format(*p)
print p
```

```
slope = 2.0
intercept = 1.0
[ 2.  1.]
```

1.9.2 scipy

http://docs.scipy.org/doc/scipy/reference/

`scipy` provides all the functionality we need for integration, optimization, interpolation, statistics, and File I/O. Here is a typical usage for solving the equation $x^2 = 2$ for x. We have to define a function that is $f(x) = 0$, and then use the `scipy.optimize.fsolve` function to solve it with an initial guess.

```
from scipy.optimize import fsolve

def f(x):
    y = 2 - x**2
    return y

x0 = 1.4
x = fsolve(f, x0)
print x
print type(x)
```

```
[ 1.41421356]
<type 'numpy.ndarray'>
```

- Confidence intervals

```
import numpy as np
from scipy.stats.distributions import  t

n = 10 #number of measurements
dof = n - 1 #degrees of freedom
avg_x = 16.1 #average measurement
std_x = 0.01 #standard deviation of measurements

```

```python
#Find 95% prediction interval for next measurement

alpha = 1.0 - 0.95

pred_interval = t.ppf(1-alpha/2.,dof)*std_x*np.sqrt(1.+1./n)

s = ['We are 95%% confident the next measurement',
       ' will be between %1.3f and %1.3f']
print ''.join(s) % (avg_x - pred_interval, avg_x + pred_interval)
```

```
We are 95% confident the next measurement will be between 16.076 and 16.124
```

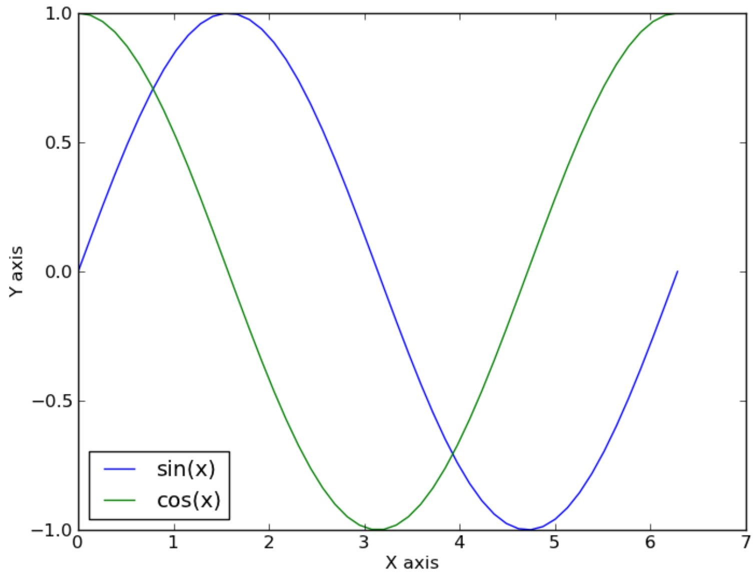
1.10 Plotting in python

http://matplotlib.sourceforge.net/ `matplotlib` is the prime plotting module for python. The syntax is similar to Matlab. The best way to learn matplotlib is to visit the gallery (http://matplotlib.sourceforge.net/gallery.html) and look for examples that do what you want. Here is a simple example.

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0,2*np.pi)
y = np.sin(x)

plt.plot(x,y)
plt.plot(x,np.cos(x))
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend(['sin(x)', 'cos(x)'], loc='best')
plt.savefig('L02-plot1.png')
plt.show()
```



2 git

for windows, you need to install Git for windows from http://code.google.com/p/msysgit/downloads/list

Then, go to http://github.com and register for an account. Make sure to follow the instructions at https://help.github.com/articles/generating-ssh-keys for setting up your ssh keys.

2.1 Getting the initial copy of the notes

Run this command in git-bash to get the initial copy of the repository.

```
git clone git://github.com/jkitchin/dft-course.git
```

This creates a new directory called dft-course in the directory you ran the command in. Later, you can update the repository with this command (you run this insie the repository):

```
git pull
```

3 emacs

A recent pre-compiled version of emacs for windows is available at http://ftp.gnu.org/gnu/emacs/windov
24.1-bin-i386.zip

you unzip this file where you want it, and run $ROOT/emacs-24.1/bin/runemacs.exe where $ROOT is where you unzipped the file.