

# Introduction to software

John Kitchen

2012-08-28 Tue

## Contents

<b>1</b>	<b>Python</b>	<b>2</b>
1.1	Simple calculations . . . . .	2
1.2	Formatted printing . . . . .	3
1.3	Data types . . . . .	4
1.3.1	lists/tuples . . . . .	4
1.3.2	dictionaries . . . . .	6
1.4	Conditional statements . . . . .	6
1.5	Loops . . . . .	8
1.6	functions . . . . .	9
1.7	Modules . . . . .	9
1.7.1	Some common standard modules . . . . .	10
1.8	Error handling . . . . .	11
1.9	Scientific and numerical python . . . . .	11
1.9.1	numpy . . . . .	11
1.9.2	scipy . . . . .	12
1.10	Plotting in python . . . . .	12
<b>2</b>	<b>git</b>	<b>13</b>
<b>3</b>	<b>emacs</b>	<b>13</b>

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\\_install](http://www.enthought.com/repo/.epd_academic_install) )  
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](http://www.tutorialspoint.com/python/python_basic_operators.htm)

Here are some simple examples

---

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

---

```
5
-2
16
0.6666666666667
```

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.

---

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

---

```
0
0.6666666666667
0.6666666666667
0.6666666666667
```

---

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

---

```
6
6.0
```

We can also do powers with `**`

---

```
1 print 2**3
2 print 2**0.5
3 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.

---

```
1 print 5 % 4
2 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}`.

---

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

---

```
The answer is 7.5
```

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

---

```
1 a = 5**3
2 b = 23
3 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.

---

```
1 a = 5**3
2 b = 23
3 print 'a = {ans0} and b = {ans1}'.format(ans0=a,
4                                         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.

---

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

---

a = 0.6666666666667  
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).

---

```

1  # short list example
2  a = [1, 2, 3, 4] # a list
3  print a
4  print len(a)
5  print a[0] # first element
6  print a[-1] # last element
7  print a[3] # also last element
8  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:

---

```

1  a = range(4)
2  print a
3
4  b = range(4,10)
5  print b
6
7  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.

---

```

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

```

---

```

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

```

Tuples are like lists e

---

```
1 a = (1,2,3,4)
2 print len(a)
3 print a[0]
4 print a[-1]
5 # 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.

---

```
1 d = {'key1':23,
2      'key2':'test',
3      5:[2,3],
4      (3,4):'tuple value'}
5
6 print d['key1']
7 print d['key2']
8 print d[5]
9 print d[(3,4)]
10
11 defaultvalue = None
12 print d.get('invalidkey', defaultvalue) # default value for nonexistent 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 (>), less than or equal to (<=), greater than or equal to (>=) and not equal (!=). These generally work on numbers and strings.

---

```
1 print 4 == 4.  
2 print 'a' != 'A'  
3 print 4 > 3  
4 print 4 <= 3  
5 print 'a' < 'b' # hmmm...
```

---

```
True  
True  
True  
False  
True
```

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

---

```
1 a = 4  
2 b = 5  
3  
4 if a < b:  
5     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.

---

```
1 a = 14  
2 b = 5  
3  
4 if a > b:  
5     print 'a is greater than b'  
6 else:  
7     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.

---

```
1 a = 4  
2 b = 4  
3 if a > b:  
4     print 'a is greater than b'
```

---

```
5 elif a == b:
6     print 'a is equal to b'
7 else:
8     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:

---

```
1 a = 4
2 b = 4
3 if a > b:
4     print 'a is greater than b'
5 elif a >= b:
6     print 'a is greater than or equal to b'
7 elif a == b:
8     print 'a is equal to b'
9 elif a <= b:
10    print 'a is less than or equal to b'
11 else:
12    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

---

```
1 for i in [0,1,2,3]:
2     print i
3
4
5 for i in range(4):
6     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

---

```
1 def myfunc(x):  
2     return x*x  
3 print myfunc(3)  
4 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 in the module with the `.` operator.

---

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

---

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

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

---

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

---

```
L01-intro-molecular-simulations.pptx
L01-intro-to-dft.pdf
L02-intro-software.aux
L02-intro-software.html
L02-intro-software.log
L02-intro-software.org
L02-intro-software.out
L02-intro-software.pdf
L02-intro-software.tex
L02-intro-software.toc
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.

---

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

---

```
L01-intro-molecular-simulations.pptx
L01-intro-to-dft.pdf
L02-intro-software.aux
L02-intro-software.html
L02-intro-software.log
L02-intro-software.org
L02-intro-software.out
L02-intro-software.pdf
L02-intro-software.tex
L02-intro-software.toc
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.

---

```
1 try:
2     1/0
3 except ZeroDivisionError, e:
4     print e
5     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/>

---

```
1 import numpy as np
2 a = np.array([1,2,3,4])
3
4 print a*a           # element-wise operation
5
6 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.

---

```
1 import numpy as np
2 a = np.array([1, 2, 3, 4])
3 print a**2
4 print np.sin(a)
5 print np.exp(a)
6 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.          ]
```

---

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

---

```
1 4
10
2.5
1.11803398875
```

### 1.9.2 scipy

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

scipy provides all the functionality we need for fitting, root finding, statistics, ODEs, interpolation, and optimization. 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.

---

```
1 from scipy.optimize import fsolve
2
3 def f(x):
4     y = 2 - x**2
5     return y
6
7 x0 = 1.4
8 x = fsolve(f, x0)
9 print x
10 print type(x)
```

---

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

### 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 mat-

plotlib 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.

---

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x = np.linspace(0,2*np.pi)
5 y = np.sin(x)
6
7 plt.plot(x,y)
8 plt.plot(x,np.cos(x))
9 plt.xlabel('X axis')
10 plt.ylabel('Y axis')
11 plt.legend(['sin(x)', 'cos(x)'], loc='best')
12 plt.savefig('L02-plot1.png')
13 plt.show()
```

---

[./L02-plot1.png](#)

## 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.

## 3 emacs

A recent pre-compiled version of emacs for windows is available at <http://ftp.gnu.org/gnu/emacs/windows/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.