Python – AN IntroduCtion

Hans Fangohr, fangohr@soton.ac.uk, BioSimGrid Meeting 25/11/2004

Overview

- Why Python
- How to get started
- Interactive Python (IPython)
- Installing extra modules
- Lists
- For-loops
- if-then
- modules and name spaces
- while
- string handling
- file-input, output
- functions
- Numerical computation
- some other features
  - ▷ long numbers
  - ▷ exceptions
  - ▷ dictionaries

  - ▷ default parameters
  - ▷ self documenting code
- Example for extensions: Gnuplot
- The `wc` program in Python
- Summary
- Outlook

Literature:

- M. Lutz & D. Ascher: *Learning Python*
  ISBN: 1565924649 (1999) (new edition 2004, ISBN: 0596002815). We point to this book (1999) where appropriate: → Chapter 1 in *LP*

- Alex Martelli: *Python in a Nutshell*
  ISBN: 0596001886

- Deitel & Deitel et al: *Python – How to Program*
  ISBN: 0130923613

Other resources:

- `www.python.org` provides extensive documentation, tools and download.

Why Python?

→ Chapter 1, p3 in *LP*

- All sorts of reasons ;-)
  - ▷ Object-oriented scripting language
  - ▷ power of high-level language
  - ▷ portable, powerful, free
  - ▷ mixable (glue together with C/C++, Fortran, . . .)
  - ▷ easy to use (save time developing code)
  - ▷ easy to learn
  - ▷ (in-built complex numbers)
- Today:
  - ▷ easy to learn
  - ▷ some interesting features of the language
  - ▷ use as tool for small sysadmin/data processing/collecting tasks

How to get started: The interpreter and how to run code

Two options: → Chapter 1, p12 in *LP*

- interactive session
  - ▷ start Python interpreter (`python.exe`, `python`, double click on icon, . . .)
  - ▷ prompt appears (`>>>`)
  - ▷ can enter commands (as on MATLAB prompt)
- execute program
  - ▷ Either start interpreter and pass program name as argument:

    `python.exe myfirstprogram.py`
  - ▷ Or make python-program executable (Unix/Linux):

    `./myfirstprogram.py`
  - ▷ Note: python-programs tend to end with `.py`, but this is not necessary.
  - ▷ On Unix/Linux: best if first line in file reads `#!/usr/bin/env python`
- Note: Python programs are interpreted (but use platform independent byte-code, as Java)

Interactive Python (IPython)

- small tool (written in Python)
- provides "more interactive" interpreter
- command and variable name completion
- browsing of objects
- easy access to help
- command line history stays over different sessions
- other (more advanced tools), such as very simple profiling etc

IPython can be downloaded from `ipython.scipy.org`.

Note: installing extra Python programs/modules

The standard Python distribution contains many modules and tools.

The standard procedure for installation of additional Python modules or stand-alone programs written in Python is:

- unpack tar or zip file
- cd into new subdirectory
- run `python setup.py install`

(This is the equivalent to the normal `./configure; make; make install` in UNIX, and a double-click on an icon in MS Windows.)

Lists

→ Chapter 2, p44 in *LP*

- important data structure, similar to vector in MATLAB

- Creating and extending lists
  - ▷ Creation of empty list
    ```
    >>>a=[]
    ```
  - ▷ Adding elements
    ```
    >>>a.append( 10 )
    ```
  - ▷ Or initialise with several values:
    ```
    >>>a=[10, 20, 45, 50, -400]
    ```

- length of a list is given by `len` (but don't need this often)

  ```
  >>>len(a)
  5
  ```

- Accessing (and changing) individual elements
  - ▷ Access individual elements (indexing as in C, starting from 0):
    ```
    >>>a[0]
    10
    ```

▷ Fetch the first element from end of list:

```
>>>a[-1]            #same as a[len(a)-1]
-400
```

▷ Change an element in list

```
>>>a[0]=20
```

- Access parts of the list (using "slicing")

→ Chapter 2, p44 in *LP*

▷ Leave out first element

```
>>>a[1:]
[20, 45, 50, -400]
```

▷ show first three elements only

```
>>>a[:3]
[10, 20, 45]
```

▷ Leave out first and last element

```
>>>a[1:-1]
[10, 20, 45]
```

- Elements (objects) in a list can be of any type, including lists

▷ `animals = ['cat','dog','mouse']`

▷ `wired_mix = ['ape', 42, [0,1,2,3,4], 'spam']`

- The `range(a,b,d)` command returns a list ranging from a up-to but not including b in increments of d. (a defaults to 0, d defaults to 1):

```
>>>range(4,8)        #similar to MATLAB's
                     #colon-operater 4:7
[4, 5, 6, 7]
>>>range(8)
[0, 1, 2, 3, 4, 5, 6, 7]
```

- More methods for lists including sorting, reversing, searching, inserting → Chapter 2, p46 in *LP*

  Example: `>>> a.sort()`

```
>>> a=range(-10,10,2)
>>> a
[-10, -8, -6, -4, -2, 0, 2, 4, 6, 8]
>>> a.reverse()
>>> a
[8, 6, 4, 2, 0, -2, -4, -6, -8, -10]
>>> a.sort()
>>> a
[-10, -8, -6, -4, -2, 0, 2, 4, 6, 8]
>>>
```

For-loop

→ Chapter 3, p87 in *LP*

- For-loops are actually for-each-loops
  For-loops iterate (generally) over lists

- Typical example, increasing counter

```
for i in range(5):
     print i
```

This is equivalent to:

```
for i in [0, 1, 2, 3, 4]:
     print i
```

Compare with MATLAB equivalent:

```
for i=0:4                    %this is MATLAB code
     disp( i )               %this is MATLAB code
end                          %this is MATLAB code
```

- Grouping of statements is done by indentation.
  - ▷ sounds strange (at first)
  - ▷ very easy to get used to
  - ▷ enforces code that is easy to read

- Example: print list of strings
  - ▷ Conventional way (for-loop)

```
animals=['dog','cat','mouse']
for i in range( len(animals) ):
   print animals[i]
```

Output:

```
dog
cat
mouse
```

  - ▷ Use for-each-loop

```
animals=['dog','cat','mouse']
for animal in animals:
   print animal
```

if-then

→ Chapter 3, p77 in *LP*

- if-then

```
a=10
b=20

if a == b:
    print "a is the same as b"
```

- if-then-else

```
a=10
b=20

if a == b:
    print "a is the same as b"
else:
    print "a and be are not equal"
```

Modules and namespaces

→ Chapter 5, p126 in *LP*

- Lots of libraries for python (called "modules")
  - ▷ string-handling
  - ▷ maths
  - ▷ more maths (LA-pack, fftw, ...)
  - ▷ data bases
  - ▷ XML
  - ▷ . . .
- access by `import`ing them
- Example: for simple maths
  - ▷ Either

    ```
    import math        #only once at start of prog.
    a = math.sqrt( 2 )
    ```

  - ▷ or import into current namespace

    ```
    from math import sqrt
    a = sqrt( 2 )
    ```

  - ▷ Can also be lazy:

    ```
    from math import *
    ```

While

→ Chapter 3, p84 in *LP*

- example 1:

```
a=1
while a<100:
    a = a*2

print "final a =",a
```

Output: `final a = 128`

- example 2: determine the square root of a using $\sqrt{a} = \lim_{n\to\infty} x_n$ and $x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$

```
a = 2.0      # want to find sqrt(a)
x = a        # take x ≈ a as a first guess for √a

while abs( x**2 - a ) > 1e-5:
    x = 0.5*( x + a / x )

print x
```

Output: `1.41421568627`

- example 3: determine the square root of a, but store x_n in list:

```
a = 2
x = [a]           # use x for sqrt_a
                  # create list with a as 1st entry

while abs( x[-1]**2 - a ) > 1e-5:
    x.append( 0.5*( x[-1] + a / x[-1])  )

print x
```

Output: [2, 1.5, 1.4166666666666665, 1.4142156862745097]

Strings

→ Chapter 2, p35 in *LP*

- use singe (') or double (") quotes, or triple double (""") quotes as limiters:
  - ▷ a = `"string 1"`
  - ▷ a = `'string 1'`
  - ▷ a = `"""string 1"""`

- Handy to include quotes in string

```
matlab_help = "in Matlab the single quote ' is used a lot"
matlab_help = """... the single quote "'" is used a lot"""
```

- "Slicing" and indexing as for lists:

```
>>> text = "this is a test."
>>> text[-1]
'.'
>>> text[:]
'this is a test.'
>>> text[:10]
'this is a '
>>> text[4:-4]
' is a t'
```

- Simple conversion functions available

```
>>> str(10)          #convert 10 into a string
'10'
>>> int(str(10))     #and back to an int
10
>>> float(str(10))   #and back to a float
10.0
>>> complex('10')    #and convert to complex number
(10+0j)
```

- Can concatenate strings using `+`

```
>>> a = "this is"; b="a test"
>>> c = a + " " + b
>>> c
'this is a test'
```

- Variety of string processing commands

  (replace, find, count, index, split, upper, lower, swapcase, . . .)

  To find out available methods for a string, try in `ipython`:

```
>>> a="test"
>>> a.                [and then press tab]
```

- `print` is the general purpose command for printing
  - ▷ separate objects to print by comma (automatic conversion to string):

```
>>> dogs = 10
>>> cats = 2
>>> print cats,"cats and",dogs,"dogs"
2 cats and 10 dogs
```

  - ▷ add comma at the end of print-statement to avoid new line
  - ▷ Explicit formatting similar to C's printf-syntax:

```
>>> print "%d cats and %d dogs" % ( cats, dogs )
```

File input/output

→ Chapter 9, p249 in *LP*

- Example: print file line by line

```
f = open('printfile.py','r')  #open file
lines = f.readlines()         #read lines,
                              #lines is list
f.close()                     #close file

for line in lines:
    print line,    # comma means: don't do linefeed
```

- Example: print file word by word in one line

```
f = open('printfile.py','r')
lines = f.readlines()
f.close()

import string

for line in lines:
    #split splits line into words.
    #Uses whitespace as separator (default)
    words = string.split( line )
    for word in words:
        print word,
```

File conversion example

Convert comma-separated data given in radian to space separated data in degree

Input

```
phi,sin(phi)
0,0,rad
0.2,0.198669,rad
0.4,0.389418,rad
0.6,0.564642,rad
0.8,0.717356,rad
1,0.841471,rad
1.2,0.932039,rad
```

Output

```
0 0.0 deg
0.2 11.3828952201 deg
0.4 22.3120078664 deg
0.6 32.3516035358 deg
0.8 41.1014712084 deg
1 48.2127368827 deg
1.2 53.4019010416 deg
```

Python programme

```
import string, math

filename = "data.csv"        #input file
f = open( filename, 'r' )    #open
lines = f.readlines()        #read
f.close()                    #close

g = open( filename[0:-4]+".txt", "w")  #open output file

for line in lines[1:]:                 #skip first line (header)
    bits = string.split( line, ',' )   #separate columns into list
                                       #use comma as separator

    # now convert numerical value from rad to deg
    bits[1] = str( float( bits[1] ) / 2.0 / math.pi * 360.0 )
    bits[2] = "deg"                    #change rad->deg

    for bit in bits:                   #write new data
        g.write( "%s " % bit )
    g.write( "\n" )
g.close()                              #close output file
```

Functions

→ Chapter 4, p97 in *LP*

- define functions `atan_deriv`, `gauss` and `trapez`

```
import math

def atan_deriv(x):
    return 1.0/(1+x**2)

def gauss(x):
    return math.exp(-x**2)

def trapez( f, a ,b):
    """ computes trapezoidal approximation of
    int_a^b f(x)dx """

    fa = f(a)
    fb = f(b)

    return (b-a)*0.5*(fa+fb)


# this is the main program:

print trapez( atan_deriv, 1, 2)
print trapez( gauss, 1, 2)
```

Functions 2 (advanced features)

```
import math
def atan_deriv(x):
    return 1.0/(1+x**2)
def gauss(x):
    return math.exp(-x**2)
def trapez( f, a ,b):
    """computes trapezoidal approximation of int_a^b f(x) dx """

    # functions are objects and have property .__name__
    print "Integrating",f.__name__, "from",a,"to",b

    fa = f(a)
    fb = f(b)
    return (b-a)*0.5*(fa+fb)

print trapez( atan_deriv, 1, 2)
print trapez( gauss, 1, 2)
#lambda defines function on the spot
print trapez( lambda x : x^2 , 1, 2)

#first string in object is stored as .__doc__
print trapez.__doc__
```

Output:

```
Integrating atan_deriv from 1 to 2
0.35
Integrating gauss from 1 to 2
0.19309754003
Integrating <lambda> from 1 to 2
1.5
computes trapezoidal approximation of int_a^b f(x) dx
```

Numerical Computation

- The Numeric module provides Linear Algebra routines (from LAPack)
- usage similar to MATLAB (i.e. vector and matrix computations in one line)
- computations are performed using compiled library (fast)

```
import Numeric, LinearAlgebra

x = Numeric.arange(0,10,0.1)  #in MATLAB: x=0:0.1:10
y = Numeric.sin(x) + 42.11
z = Numeric.outerproduct(x,y) #matrix multiplication of x and y

eigval = LinearAlgebra.eigenvalues(z)
eigvec = LinearAlgebra.eigenvectors(z)

u,x,v  = LinearAlgebra.singular_value_decomposition(z)

#Note that there is a Matlab compatibility mode
import MLab
eigval2, eigvec2 = MLab.eig( z )
[eigval2, eigvec2] = MLab.eig( z )  #if you like this notation

if eigval2 == eigval:       #check that MLab and LinearAlgebra
    print "All is well"     #produce same result
else:
    print "Problem"

u2,x2,v2 = MLab.svd(z)
```

Some other features

- Exact integer arithmetic if required

```
def fac(n):             # recursive implementation
    if n == 1:          # of n!
        return 1
    else:
        return n*fac(n-1)

print "3! =",fac(3)
print
print "150! =",fac(150)
print "150! =",float( fac(150) )
print
print "500! =",fac(500)
```

Output:

```
3! = 6

150! = 571338395644585459047893286526105400318955357860112641825483758331 7
98291248453983931265744886753111453771078787468542041626662501986845044663
55949195922066574942592095735778929325357290444962472405416790722118445437
122269675520000000000000000000000000000000000000
150! = 5.71338395645e+262

500! = 12201368259911100687012387854230469262535743428031928421924135883858
45373153881997605496447502203281863013616477148203584163378722078177200480
78520515932928547790757193933060377296085908627042917454788242491272634430
567017327076946106280231045264421887878946575477714986349436778103764427403
382736539747138647787849543848959553753799042324106127132698432774571554630
997720278101456108118837370953101635632443298702956389662891165897476957208
792692887128178007026517450776841071962439039432253642260523494585012991857
150124870696156814162535905669342381300885624924689156412677565448188650659
38479517753608940057452389403357984763639449053130623237490664450..
```

- Exceptions: → Chapter 7, p194 in *LP*

  ▷ catch exception for opening file

```
try:
    f = open("this file does really not exist","r")
    lines = f.readlines()
    f.close()
except:
    print("Can't open file - skip reading")
    lines = []
```

  Output: `Can't open file - skip reading`

  ▷ can raise exceptions

```
def my_f( x ):
    if x == 0:
        raise ZeroDivisionError, "Attempt to divide by 0 in my_f"

    return 1.0/x

print my_f( 1 )
print my_f( 0.5 )
print my_f( 0 )
```

  Output

```
1.0
2.0
Traceback (most recent call last):
  File "except2.py", line 11, in ?
    print my_f( 0 )
  File "except2.py", line 3, in my_f
    raise ZeroDivisionError,"Attempt to divide by 0 in my_f"
ZeroDivisionError: Attempt to divide by 0 in my_f
```

  ▷ can create our own set of exceptions

- Dictionary: built-in data type → Chapter 2, p49 in *LP*

```
dic = {}          #create empty dictionary

dic["Hans"] = "room 1033"      #fill dic
dic["Andy C"] = "room 1031"    #"Andy C" is key
dic["Ken"] = "room 1027"       #"room 1027" is value

for key in dic.keys():
    print key,"works in",dic[key]
```

Output:

```
Hans works in room 1033
Andy C works in room 1031
Ken works in room 1027
```

- Without dictionary:

```
people = ["Hans","Andy C","Ken"]
rooms  = ["room 1033","room 1031","room 1027"]

#possible inconsistency here since we have two lists
if not len( people ) == len( rooms ):
    raise "Oops","people and rooms differ in length"

for i in range( len( rooms ) ):
    print people[i],"works in",rooms[i]
```

- Default parameters for functions

```
def example_f( x, a=0, b=1, c=2 ):
    print "x =",x,", a =",a,", b =",b,", c =",c


example_f( 10 )                  #x=10, others default
example_f( 10, 5 )               #x=10, a=5
example_f( 10, c = 5 )           #x=10, c=5
example_f( 10, c = 5, b = 6 )    #if name is specified
                                 #then order of a,b,c
                                 #doesn't matter
```

Output:

```
x = 10 , a = 0 , b = 1 , c = 2
x = 10 , a = 5 , b = 1 , c = 2
x = 10 , a = 0 , b = 1 , c = 5
x = 10 , a = 0 , b = 6 , c = 5
```

- Self-documenting code
  - ▷ similar to MATLAB's help (first comment is used for help)
  - ▷ can list methods for object using `dir` (remember: everything in Python is an object, including functions and modules)

```
>>> import string
>>> dir(string)
['_StringType', '__builtins__', '__doc__', '__file__', '__name__',
'_float', '_idmap', '_idmapL', '_int', '_long', 'ascii_letters',
'ascii_lowercase', 'ascii_uppercase', 'atof', 'atof_error',
'atoi', 'atoi_error', 'atol', 'atol_error', 'capitalize',
'capwords', 'center', 'count', 'digits', 'expandtabs',
'find', 'hexdigits', 'index', 'index_error', 'join',
'joinfields', 'letters', 'ljust', 'lower', 'lowercase',
'lstrip', 'maketrans', 'octdigits', 'printable', 'punctuation',
'replace', 'rfind', 'rindex', 'rjust', 'rstrip', 'split',
'splitfields', 'strip', 'swapcase', 'translate', 'upper',
'uppercase', 'whitespace', 'zfill']
```

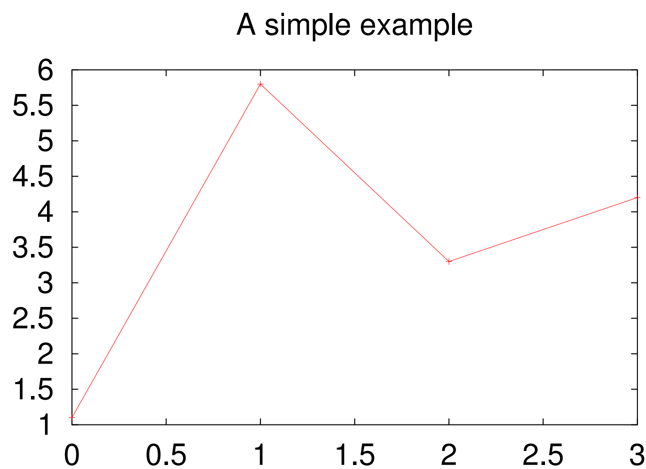
  - ▷ If method has comment after its first line, then it is stored in `__doc__` (this is what IPython uses to display help):

```
>>> print string.split.__doc__
split(s [,sep [,maxsplit]]) -> list of strings

    Return a list of the words in the string s, using sep as the
    delimiter string.  If maxsplit is given, splits into at most
    maxsplit words.  If sep is not specified, any whitespace string
    is a separator.

    (split and splitfields are synonymous)
```

Extensions, example Gnuplot

- 2d-plots, *i.e.* $y = f(x)$.

```
import Gnuplot

def demo():
    """Demonstrate the Gnuplot package."""

    g = Gnuplot.Gnuplot()
    g.title('A simple example')
    g('set data style linespoints') # any gnuplot command

    #create window with curve of following 4 points
    g.plot([[0,1.1], [1,5.8], [2,3.3], [3,4.2]])

    #write data to postscript file
    g.hardcopy('gp_test.ps', enhanced=1, color=1)

    raw_input('Please press return to continue...')

demo()
```



The `wc` program in Python

- Passing of command line arguments similar to C:
  - ▷ `sys.argv` is a list of command line arguments
  - ▷ `len(sys.argv)` returns the number of arguments (in C: argc)
- Example: `wc.py`

```
#!/usr/bin/env python
"""wc.py: program that copies the behaviour of the UNIX wordcount
programm wc.  SYNTAX: wc.py FILENAME"""

import sys, string

# has user provided command line argument, i.e. filename
if len( sys.argv ) < 2:          # if not
    print __doc__                # print documentation string
    sys.exit(0)                  # and exit (return to OS)

filename = sys.argv[1]           # otherwise take first argument
                                 # as filename

f = open(filename,'r')           # open and read file
lines = f.readlines()
f.close()

linecounter = 0; wordcounter = 0; charcounter = 0

for line in lines:                      #iterate over lines
    linecounter = linecounter + 1
    charcounter = charcounter + 1       #count new-line char

    words = string.split( line )
    for word in words:                  #iterate over words
        wordcounter = wordcounter + 1
        charcounter = charcounter + len( word ) + 1

print "%7d %7d %7d %s" % (linecounter, wordcounter, charcounter, filename)
```

Output from `./wc.py wc.py` is `39 151 955 wc.py`

Summary

- Python is
  - ▷ easy to learn
  - ▷ quick to code
  - ▷ code often works straight away
  - ▷ easily re-used (clear structure)

- Typical areas of use
  - ▷ data collection/processing/conversion
  - ▷ sys-admin
  - ▷ prototyping
  - ▷ coding big projects
  - ▷ glue language
  - ▷ numeric computation
  - ▷ visualisation

- There is much more to say but not within 45 minutes.

Outlook

Extension modules exist for

- Gnuplot/Grace/... for 2d plots
- Scientific computation (Numeric/Scientific)
- VTK, OpenDX, ... for demanding visualisation
- Tk for GUIs
- XML
- Databases
- Finite element calculations
- . . .