Python for Scientific Computing

Jan Van lent

BICS, University of Bath, UK

Thursday 22 November 2007
University of Tartu

What is Python?

- programming language
- buzz words: object oriented, functional, imperative, interactive, dynamic, reflective
- simple, uniform, consistent, elegant syntax
- highly readable
- batteries included: extensive standard libraries
- many additional packages available, e.g. for scientific computing
- large user community
- several implementations: CPython, Jython, IronPython, PyPy
- edit - test vs edit - compile - test
- comparison to other languages
- personal experience: tried many languages, very productive in Python, so far one of my favourites

History

- designed by Guido Van Rossum
- start late 80s, early 90s
- at CWI (Amsterdam)
- inspired by ABC (still around)
- influenced by and influence on many languages
- Van Rossum is the 'Benevolent Dictator for Life' (BDFL)
- currently working at Google
- current stable version 2.5, future 2.6 and Python 3000

The Python Command Line

- interactive command interpreter

```
$ python
Python 2.5.1 (r251:54863, Jul 24 2007, 00:09:45)
[GCC 4.0.1 (Apple Computer, Inc. build 5367)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 + 1
2
>>> (1 + 2 +
... 5)
8
>>>
```

- Ctrl-D to quit
- alternatives:
  - pycrust, ipython
  - editor, IDE

A Python Program

- python program `test.py`

```
#! /usr/bin/env python

# Everything from hash to end of line is comment.
print "This is a test."
```

- running python program
  - command line: `$ python test.py`
  - unix shell: `$ ./test.py`
  - python shell: `>>> execfile("test.py")`
  - editor or IDE: IDLE, emacs, vi, eclipse, spe, etc.

Shell Commands

- semicolon to separate statements
- example one-liner: list of directories in PATH

```
$ echo $PATH
/usr/bin:/usr/local/bin
$ python -c 'import os; print "\n".join(os.getenv("PATH").split(":"))'
/usr/bin
/usr/local/bin
```

- can be handy, but almost always better to create a script

```
# pathlist.py

import os

PATH = os.getenv("PATH")
pathlist = PATH.split(":")
print "\n".join(pathlist)'
```

Overview of Datatypes

- basic datatypes: booleans, numbers, strings, tuples
- powerful compound datatypes: lists, dictionaries
- also: queues, sets
- multidimensional arrays from separate package
- files
- python language: functions, classes, instances, modules, exceptions, iterators, generators

Booleans

- **type:** `bool`

```
True
False
not True
True and True
True or False
```

Numbers

- **types**: `int`, `long`, `float`, `complex`

```
1
2 + 3
2**64
2.0 + 3.0
(2.0+1.0j) * (2-1j)
1 / 2
1 // 2
from __future__ import division
1 / 2
1 // 2
```

Strings

- **types:** `str,unicode`

```
"double quoted"
'single quoted'
"handy for 'quotes' in quotes"
"""multiline
strings
"""
'''escape characters:
newline \n,
tab \t,
backslash \\'''
r"raw strings useful for e.g. \texcommands"
```

Tuples

- immutable, heterogenous sequence
- type: `tuple`

```
x = (1, "a")
y = (1.0,)
z = ("a", ("b", 2), 4)
print x + y
print x[0], len(x)
x[0] = 2                    # error
```

Lists

- mutable, heterogenous sequence
- type: `list`

```
list
x = [ 0, 1, 2, 3 ]
print x[0]
x[1] = 7
x[-1]
x[1:3]
x[1:-1]
y = ["a", 10]
print x + y
x.sort()
print x
print sorted(y)
```

Dictionaries

- mapping from keys to values
- keys must be immutable (hashable)
- type: `dict`

```
d = { "a": 1, "b": 2 }
d["a"]
d["c"] = 3
d = dict(a=1, b=2)
d = {}
d[(1,2)] = [8, 9]
print d.keys(), d.values()
print len(d), d.items()
```

Statements and Control Flow

- import: working with modules
- '=': assignment
- print: output
- if: conditional statement
- while, for: looping
- functions
- classes
- iterators, generators, generator expressions
- raise, try, except: exceptions

Working with Modules

- every python file can be a module
- file `polynomial.py`

```
p0 = 1

def add(p, q):
    ...
...
```

- `import polynomial`
- `from polynomial import p0, add`
- `from polynomial import *`: to be used with care
- executable statements in module file

```
if __name__ == "__main__":
    run_test()
```

Assignment

- the equal sign '=' is used to give a name to an object

```
a = 4
```

- note: mutable objects

```
a = [1, 2]
b = a
b[1] = 8
print a
```

- syntactic sugar for item assignment

```
a[1] = 2
a.__setitem__(1, 2)
```

- not used for comparison: `5 == 1`
- only allowed at statement level

```
a = 1
if (a = 5):
    print "five"
```

Assigning Multiple Values

- unpacking of sequences

```
a, b = 1, 2
a, b = b, a
c = [1, 2, 3]
u, v, w = c
[(a, b), c] = [(1, 2), 3]
```

Input/Output

- `print "Some output"`
- string formatting:
  `print "a %s number %d" % ("big", 2**64)`
- output to a file

```
file = open("results.dat", "w")
print >> file, "a + b =", 3+4
file.write("no newline")
file.close()
```

- input from file

```
file = open("results.dat", "r")
print file.readline()
print file.readlines()[-2:]
file.close()
```

Conditional Statement

- keywords: `if`, `else`, `elif`

```
a = 1
if a >= 0:
    print "pos"

if a == 1:
    print "yes"
else:
    print "no"
```

```
if a == 1:
   print "pos"
elif a == -1:
   print "neg"
else:
   print "zero"
```

Basic Looping

- keywords: `while`, `for`

```
i = 0
while i < 1:
    print i
    i += 1

for i in range(10):
    print i
```

- further keywords: `break`, `continue`, `else`

Looping for Sequences

```
a = ["one", "two", "three" ]
for word in a:
    print word

for (word, i) in enumerate(a):
    print i, word

d = { "one": 1, "two": 2, "seven": 7 }
for word in sorted(d.keys()):
    print word, d[word]
for word, i in d.items():
    print word, i
```

Functions

- keyword: `def`

```
def add_positive(x, y):
    if y > 0:
        z = x + y
    else:
        z = x
    return z
```

- functions are objects

```
flist = [ add_positive, add_negative ]
```

- functions can be nested

```
def adder(x):
    def add(y):
        return x + y
    return add
add2 = adder(2)
print add2(5)
```

Classes

```
class poly:
    def __init__(self, coeff):
        self.coeff = coeff
    def degree(self):
        return len(self.coeff)
    def add(self, other):
        cs = [ a + b for (a, b) in
               zip(self.coeff, other.coeff) ]
        return poly(cs)

a = poly([0,1,0])
b = poly([1,1,1])
c = a.add(b)
print c.coeff
```

Exceptions

- keywords: `raise, try, except, finally`

```
a = [ 1, 2, 3 ]
try:
    print a[7]
except KeyError, e:
    print e

try:
    file = open("data.txt", "r")
except IOError:
    print "Could not open data.txt."
```

Iterators

- any object with `__iter__` and `next` methods
- raise `StopIteration` exception when no more elements

```
class squares:
    def __init__(self, data):
        self.data = data
        self.index = 0
    def __iter__(self): return self
    def next():
        if self.index < len(self.data):
            x = self.data[self.index]
            self.index += 1
            return x * x
        else:
            raise StopIteration

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

Generators

- creating iterators using functions
- keyword `yield`

```
def squares(data):
    for x in data:
        yield x * x

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

List Comprehension and Generator Expressions

- compact way to create lists

```
xs = [ -2, -1, 0, 1, 2, 3, 4 ]
ys = [ x*x for x in xs ]
```

- generator expressions create iterators

```
ys = ( x*x for x in xs )
for y in ys:
    print y
```

- can be used in many places instead of a sequence

```
sum(x * x for x in xs)
dict((x, x*x) for x in xs)
set(x*x for x in xs)
```

Getting Help

- a lot of functionality available
- how to find what you need?
- excellent documentation
  - tutorial, library, reference
  - usually comes with installation
  - also online: `http://www.python.org/doc/`
- internet, books
- within python
  - very dynamic, introspective, reflective
  - important tools: `help`, `info`, `dir`, `type`, `str`
  - work on any value (number, list, dict, function, class, instance, module, etc.)
  - a discoverable language
- Ask me!

Providing Help

- comments
- docstrings
- used by the help command
- guidelines on writing good docstrings
- other uses: automatic documentation generation (several tools), examples (automatically checked), unit tests, parser generators, etc.

Some interesting packages

- math, random
- list, collections
- text processing: string, re
- os, sys, fileinput, subprocess
- networking: mail, html, xml, http, ftp, socket
- data persistence: pickle, dbm, sqlite3
- gui: tkinter
- scientific computing: numpy, scipy, scientificpython, pypar, mpi4py, petsc, trilinos
- visualisation: pil, pyx, matplotlib, gnuplot, pyvtk, mayavi, vpython
- maths: sage

Advanced Topics

- interfacing to other languages (C, C++, Fortran, Matlab, R, etc.)
- embedding python
- python extension modules
- swig, f2py, pyfort, psyco, pyrex, weave, inline
- gui

Multidimensional Arrays

- history: numeric, numarray, numpy
- adds homogeneous multidimensional array (`[u]int{8,16,32,64}`, `float{32,64,96}`, `complex{64,128,192}`)
- storage in row major order (like C)
- used for vectors, matrices, grids, etc.
- used to interface to other libraries: BLAS, LAPACK, SuperLU, UMFPack, MPI
- scipy: many useful functions, many use existing C and Fortran packages

Numpy Arrays

```
from numpy import *
x = array([1.0, 1.0, 2.0])
x = zeros((4, 5))
x = ones((4, 2))
print x[3, 1]
x[2, 0] = 9
x = random.uniform(0.0, 1.0, 4)
I = identity(4)
A = I + random.normal(0.0, 0.1, (4, 4))
y = dot(A, x)
ew, ev = linalg.eig(A)
```

Array Slices

- provide view onto part of array

```
a = reshape(arange(4*7), (4, 7))
print a
b = a[1:2, :]
b[:,:] = b * 2
print a
print a[0,1:6:2]
```

Useful Numpy Functionality

- **try** `import numpy; dir(numpy)`
- `mgrid, ogrid`
- `arange, array, concatenate, dot, where, zeros`
- `all, allclose, alltrue, amax, amin, any`
- `asarray, choose, compress, copy, cumsum`
- `diag, diagonal, product, put, rank, ravel`
- `reshape, resize, sort, squeeze, sum, take, tile`
- `abs, absolute, add, arcos, bitwise_and, ceil, conj, conjugate, less, log, log10, logical_and, maximum`

Scipy

- fft
- integration
- ordinary differential equations
- interpolation
- numerical linear algebra for dense and sparse matrices
- image processing
- optimization
- random
- signal processing
- special functions
- statistics

Further Reading

- style guide PEP 8
- documentation: tutorial, library reference, language reference
- `python.org`, `diveintopython.org`, `greenteapress.com/thinkpython`, `scipy.org`
- books
- Python 3000 (py3k)

Euler Method

- ordinary differential equation (ODE)

$$y'(x) = f(x, y(x)) \tag{1}$$
$$y(0) = y_0 \tag{2}$$

- grid

$$x_i = ih$$

- forward Euler method

$$y_i = y_{i-1} + hf(x_{i-1}, y_{i-1})$$

- `euler.py`

Trapezoidal Rule

- integral

$$I = \int_{-1}^{1} f(x)dx$$

- grid of points

$$x_i = a + hi, \qquad h = \frac{b-a}{n}$$

- approximation

$$I \approx \frac{h}{2}f(x_0) + h\sum_{i=1}^{n-1} f(x_i) + \frac{h}{2}f(x_n)$$

Double Exponential Quadrature

- integral

$$I = \int_{-1}^{1} f(x)dx$$

- transform $x = \phi(t) = \tanh(\frac{\pi}{2}\sinh(t))$

$$I = \int_{-\infty}^{\infty} f(\phi(t))\phi'(t)dt$$

- trapezoidal rule with stepsize h, $x_i = ih$

$$I \approx h\sum_{i=\infty}^{\infty} f(\phi(x_i))\phi'(x_i)$$

- truncate ($2n+1$ points)

$$I \approx h\sum_{i=-n}^{n} f(\phi(x_i))\phi'(x_i)$$

Gauss-Legendre Quadrature

- quadrature rule

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

- Gauss-Legendre: points x_i and weights w_i s.t. exact for polynomials of degree $2n-1$
- define the symmetric, tridiagonal matrix

$$T = \begin{bmatrix} 0 & \beta_1 & & \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & 0 \end{bmatrix}, \qquad \beta_i = \frac{i}{\sqrt{(2i-1)(2i+1)}}$$

- eigenvalues λ_i, eigenvectors $\mathbf{q}_i$ s.t. $T\mathbf{q}_i = \lambda_i \mathbf{q}_i$, $\mathbf{q}_i^T \mathbf{q}_i = 1$
- points $x_i = \lambda_i$, weights $w_i = 2(\mathbf{e}_1^T \mathbf{q}_i)^2$

Tasks

- Implement a function `trap_quad(f, a, b, n)` that calculates the integral $\int_a^b f(x)dx$ using the trapezoidal rule with n intervals.
- Implement a function `de_quad(f, h, n)` that calculates the the integral $\int_{-1}^{1} f(x)dx$ using double exponential quadrature with stepsize h and $2n + 1$ function evaluations.
- Implement a class `gauss_legendre` such that

```
gl5 = gauss_legendre(n) # n = 5
print "points", gl5.x, "weights", gl5.w
print "approximation", gl5.integrate(f)
```

gives the expected result. Use `numpy.linalg.eig` to find the evaluation points and weights.

Tasks (continued)

- Calculate $\int_{-1}^{1} \sqrt{1-x^2}dx$ exactly and using trapezoidal, double exponential and Gauss-Legendre quadrature. Take $h = \alpha/n, \alpha = 3$ for the double exponential quadrature. Make a table of the errors for each method. Use a column for each method and for each row the same number of function evaluations for each method ($N = n + 1$, $N = 2n + 1, N = n$)
- Repeat for the integral $\int_{-1}^{1} \sin(\pi x)dx$.
- Use all three methods to find a approximations for the integrals $\int_{-1}^{1} \exp(\sin(\pi x))dx$, $\int_{-1}^{1} \exp(\frac{1}{1-x^2})dx$.
- Comment on your results.