Python for Scientific Computing

Konrad Hinsen
Centre de Biophysique Moléculaire, CNRS
Rue Charles Sadron; 45071 Orléans
E-Mail: hinsen@cnrs-orleans.fr

8 October 2003

Overview

- Common data types
- Loops and conditions
- File handling
- Text processing
- Arrays
- Functions and modules

1 Modules

Python code is structured by **modules**. Modules can contain functions, variables, classes, and other modules.
To use the contents of a module, they must be **imported**, either individually:

```
from Module import function1, function2
```

or collectively:

```
from Module import *
```

2 Numbers

Integer	0, 1, 2, 3, -1, -2, -3
Real	0., 3.1415926, -2.05e30, 1e-4 (must contain dot or exponent)
Imaginary/Complex	1j, -2.5j, 3+4j (the last one is a sum)
Addition	3+4, 42.+3, 1+0j
Subtraction	2-5, 3.-1, 3j-7.5
Multiplication	4*3, 2*3.14, 1j*3j
Division	1/3, 1./3., 5/3j
Power	1.5**3, 2j**2, 2**-0.5

3 Mathematical functions

The standard mathematical functions (`sqrt`, `log`, `log10`, `exp`, `sin`, `cos`, `tan`, `arcsin`, `arccos`, `arctan`, `sin`, `cosh`), as well as the constants `pi` and `e`, are not part of the basic language, but contained in a module called `Numeric`:

```
from Numeric import sin, pi
print sin(pi/3.)
```

4 Text strings

Two forms: `'abc'` or `"abc"`

Line breaks are indicated by `\n`: `"abc\ndef"`

Concatenation: `"abc"+'def'` gives `"abcdef"`

Repetition: `3*"ab"` gives /"ababab"/

Triple quotes permit multi-line strings:

```
comment = '''This string
is split over
```

```
three lines.
'''
```

5 Lists

Lists are very important in Python. They can store any kind of element:

```
some_prime_numbers = [2, 3, 5, 7, 11, 13]
names = ['Smith', 'Jones']
a_mixed_list = [3, [2, 'x'], '']
```

Concatenation	`[0,1]+['a','b']`
	→ `[0,1,'a','b']`
Append	`names.append('Python')`
Sort	`names.sort()`
Reverse	`names.reverse()`

6 Tuples

Tuples are **immutable** lists. Once created, they can not be changed. They are more efficient than lists, and can be used in places where immutable objects are required.

Syntax: `(1, 2, 3)`

Note: single-element tuples require a comma: `(1,)`

7 Vectors

```
from Scientific.Geometry import Vector
x = Vector(1,0,0)
print x[0], x[1], x[2]
```

Addtition	Vector(1,0,0)+Vector(0,-1,3)
Subtraction	Vector(,1,0)-Vector(1.5,4,0)
Mult. by scalar	3.5*Vector(1,1,0)
Div. by scalar	Vector(1,1,0)/2
Dot product	Vector(1,2.5,0)*Vector(0,-1,3.1)
Cross product	Vector(1,2.5,0).cross(Vector(0,-1,3.1))
Length	Vector(2.5, 3.4, 1.).length()
Normalization	Vector(1.,4.,2.).normal()
Angle	Vector(1,2.5,0).angle(Vector(0,-1,3.1))

8 Objects

An **object** is a collection of data. The **type** of an object determines what **operations** are available for it.

Type	Operations
Integer	`+ - * /`
List	`+ append sort` ...
Vector	`+ - * / cross` ...

Object-oriented programming: Programming by creating problem-specific object types

9 Variables

Fortran, C:
A variable **stands for** a memory area holding data.

Python:
A variable **refers to** a memory area holding data.
Two variables can refer to **the same** memory area.

```
a = [1, 2, 3]
b = a
a.append(4)
print b
```

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

10 Indexing

Sequence objects (lists, strings, vectors, ...) permit **indexing**:

```
text = 'abc'
print text[1]
print text[-1]
```

and **slicing**:

```
print text[0:2]
print text[1:-1]
```

Note: The first element always has index `0`.

11 Indexing

Mutable sequence objects (e.g. lists) permit **indexed assignment**:

```
names = ['Smith', 'Jones']
names[1] = 'Python'

some_prime_numbers = [2, 3, 5, 7, 11, 13]
some_prime_numbers[3:] = [17, 19, 23, 29]
```

12 Loops

```
for item in sequence:
    print item
```

Counting loop ("do loop"):

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

Note: `range(5)` yields `[0, 1, 2, 3, 4]`

13 File philosophy

Fortran: A file is a sequence of records, which can be "formatted" (text) or "unformatted" (binary).

C/Python: A file is a sequence of bytes. Text files use **control characters** for formatting (line breaks etc.)

14 Reading files

Read whole file into a string object:
```
contents = file('filename').read()
```

Read whole file into a list of lines:
```
contents = file('filename').readlines()
```

Read line by line:
```
for line in file('filename'):
    print line
```

15 Writing files

```
output_file = file('filename', 'w')
output_file.write('line1\n')
output_file.write('line2\n')
output_file.close()
```

Replace `'w'` by `'a'` to append to an existing file.

16 Binary files

The module `struct` provides conversion from string data to other data types and back:

```
from struct import unpack
binary_data = file('data.bin').read(8)
a, b = unpack('ii', binary_data)
```

```
from struct import pack
binary_data = pack('id', 1, 2.5)
file('data.bin', 'w').write(binary_data)
```

17 Fortran formatted files

```
from Scientific.IO.FortranLine
s = '   59999'
line = FortranLine(s, '2I4')
print line[0]
print line[1]
```

```
from Scientific.IO.FortranLine
line = FortranLine([3.14159, 2.718], '2D15.5')
print str(line)
```

18 Example

This program counts the lines and words in a file.

```
lines = 0
words = 0
for line in file('some_file'):
    lines = lines + 1
    words = words + len(line.split())

print lines, " lines, ", words, " words."
```

19 Conditions

equal	`a == b`
not equal	`a != b`
greater than	`a > b`
less than	`a < b`
greater than or equal	`a >= b`
less than or equal	`a <= b`

The result of a comparison is 0 (false) or 1 (true).

Logical operators: `not`, `and`, `or`

20 Conditions

```
if a > 0:
   print 'greater'
elif a == 0:
   print 'equal'
else:
   print 'smaller'


while n > 0:
    n = n - 1
```

21 Text processing

```
'ab cd e'.split()     ['ab','cd','e']
'1, 2, 3'.split(',')  ['1', ' 2', ' 3']
','.join(['1','2'])   '1,2'
' a b c  '.strip()    'a b c'
'text'.find('ex')     1
'Abc'.upper()         'ABC'
'Abc'.lower()         'abc'
```

Convert to numbers: `int('2')`, `float('2.1')`

Any data as text: `str(3)`, `str([1, 2, 3])`

22 Dictionaries

A dictionary stores a **mapping** from (almost) arbitrary **keys** to arbitrary **values**.

```
dictionary = {'a': 1, 'b': 2, 'c': 3}
dictionary['z'] = 26
print dictionary['c']
print dictionary.get('h', 'not found')
print dictionary.keys()
print dictionary.values()
print dictionary.items()
```

23 Example

This program reads a file that describes an atomic system. Each line has a chemical element name, followed by three coordinates. The program calculates the center of mass.

```
from Scientific.Geometry import Vector
masses = {'h': 1, 'c': 12, 'o': 16}
```

```
tot_mass = 0.
sum = Vector(0., 0., 0.)
for line in file('atoms'):
    element, x, y, z = line.split()
    position = Vector(float(x), float(y), float(z))
    mass = masses[element.lower()]
    tot_mass = tot_mass + mass
    sum = sum + mass*position

print 'Center of mass: ', sum/tot_mass
```

24 Arrays

Properties:

- multidimensional
- all elements of same type
- compact storage
- efficient computations
- define arithmetic operations
- flexible indexing options
- indexing creates references
- easy interfacing to C and Fortran

All array functions are in module `Numeric`.

25 Creating arrays

zero	`zeros((2, 3), Int)`
from lists	`array([[1, 2], [-1, -2]])`
range	`arange(0., 2., 0.4)`
repetition	`resize([0, 1], (3,3))`

26 Array attributes

Shape: `a.shape`
the dimensions of the array

Element type: `a.typecode()`
character, byte, short/long integer, short/long float, short/long complex

27 Array indexing

One index expression per dimension:

element	`a[1, 1]`
row	`a[0]`
column	`a[:, 0]`

Slices:

range	`a[3:7]`
every second element	`a[::2]`
reverse order	`a[::-1]`
negative indices	`a[-2:-8:-2]`

28 Array operations

- standard arithmetic (elementwise)
- standard maths functions (applied elementwise)
- reduction (`add.reduce`, `multiply.reduce`)
- scalar, dot, and outer product
- structural operations (selection, rearrangement)

29 Functions

Like C, but unlike Fortran, Python makes no difference between functions and subroutines. Functions can have zero, one, or more return values.

```
from Scientific.Geometry import Vector

def distance(r1, r2):
    return (r1-r2).length()

print distance(Vector(1., 0., 0.),
               Vector(3., -2., 1.))
```

30 Functions

Multiple return values:

```
import Numeric

def cartesianToPolar(x, y):
    r = Numeric.sqrt(x**2+y**2)
    phi = Numeric.arccos(x/r)
    if y < 0.:
        phi = 2.*Numeric.pi-phi
    return r, phi

radius, angle = cartesianToPolar(1., 1.)
```

31 Modules

Modules collect related code and variables. Modules are completely isolated from each other, access to data in other modules is available only by explicit **import**.

Functions in a module can use, but not modify, variables in the same module. Any variable that is changed in a function is automatically local to that function.

Modules are Python code files whose name is the module name plus '.py'. A script that is directly executed by the interpreter defines the module `__main__`.

32 Command line arguments

If you start your script by typing

```
python my_script.py arg1 arg2 arg3
```

you can access the arguments via the list `argv` in the module `sys`:

```
import sys
print sys.argv
```