Python Crash course

Håkon Enger

Dept. of Mathematical Sciences and Technology
Norwegian University of Life Sciences

January 29, 2008

Python

- General-purpose, high-level programming language
- Published by Guido van Rossum in 1991
- Named after Monty Python
- Open source, BSD-like license
- Current production version is 2.5.1 (April 2007)
- Python 3.0 will be major revision
  - not backwards compatible
- Supports object-oriented and functional programming
- Originally intended to make programming more accessible to more people

Creating python programs

hello.py

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

print "Hello, world!"
```

Creating python programs

hello.py

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

print "Hello, world!"
```

```
$ python hello.py
Hello, world!
$
```

Creating python programs

hello.py

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

print "Hello, world!"
```

```
$ python hello.py
Hello, world!
$ chmod ugo+x hello.py
$ ./hello.py
Hello, world!
$
```

Datatypes

- Python is dynamically typed
  Type is implicit, not declared
- `x = 17` makes `x` an integer with value 17
- `x = 17.0` makes `x` a float (double) with value 17
- Can also use `x = float(17)`

Datatypes

- Python is dynamically typed
  Type is implicit, not declared
- `x = 17` makes `x` an integer with value 17
- `x = 17.0` makes `x` a float (double) with value 17
- Can also use `x = float(17)`
- Division of integers gives integer result:
  `1 / 2` $\longrightarrow$ 0
  `1. / 2` $\longrightarrow$ 0.5
- Complex numers are always doubles:
  `z = 17 + 23j` or `z = complex(17,23)`
- Long integers have unlimited precision

```
p = 191561942608236107294793378084303638130997321548169216L
```

Datatypes

- Python is dynamically typed
  Type is implicit, not declared
- `x = 17` makes `x` an integer with value 17
- `x = 17.0` makes `x` a float (double) with value 17
- Can also use `x = float(17)`
- Division of integers gives integer result:
  `1 / 2` $\longrightarrow$ 0
  `1. / 2` $\longrightarrow$ 0.5
- Complex numers are always doubles:
  `z = 17 + 23j` or `z = complex(17,23)`
- Long integers have unlimited precision

```
p = 191561942608236107294793378084303638130997321548169216L
```

- Duck typing:
  If it walks like a duck and quacks like a duck, I would call it a duck.

– James Whitcomb Riley

Datatypes

- string: `s = 'Hello'` or `s = "Hello"`
  - unicode: `len("Håkon")==6`, `len(u"Håkon")==5`
- list: `L = [ 17, 23, 42 ]` ⟶ `L[0] == 17`
  - Inhomogeneous:
    `L = [ 17, 4.5, 'Hello', [ 'x', 3.14 ] ]`
- dictionary: `D = { 'Spam': 'Eggs',`
  `42: 'Life, the universe & everything' }`
- tuple: `T = (1, 4.5, 'x')` – immutable

Whitespace

test.py

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

 print "Hello, world!"
```

Whitespace

test.py

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

 print "Hello, world!"
```

```
$ python test.py
  File "test.py", line 3
    print "Hello, world!"
    ^
IndentationError: unexpected indent
```

Block constructs

blocks.py

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

for i in range(0,4):
    print i
    if i == 2:
        print "This is number two"
print "Outside of loop, i =",i
```

Block constructs

blocks.py

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

for i in range(0,4):
    print i
    if i == 2:
        print "This is number two"
print "Outside of loop, i =",i
```

```
$ python blocks.py
0
1
2
This is number two
3
Outside of loop, i = 3
$
```

Interactive Python

```
$ python
Python 2.5.1 (r251:54863, Oct  5 2007, 13:50:07)
[GCC 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)
Type "help", "copyright", "credits" or "license" for more
>>> 2 + 2
4
>>> ⟨Ctrl-D⟩
$
```

Interactive Python

```
$ python
Python 2.5.1 (r251:54863, Oct  5 2007, 13:50:07)
[GCC 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)
Type "help", "copyright", "credits" or "license" for more
>>> 2 + 2
4
>>> ⟨Ctrl-D⟩
$ ipython
Python 2.5.1 (r251:54863, Oct  5 2007, 13:50:07)
Type "copyright", "credits" or "license" for more informa

IPython 0.8.1 -- An enhanced Interactive Python.
?         -> Introduction to IPython's features.
%magic    -> Information about IPython's 'magic' % functior
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?

In [1]: 2 + 2
Out[1]: 4
```

Functions

Defining functions in python:

```
def fact(x):
    if x==0:
        return 1
    else:
        return x*fact(x-1)
```

Functions

Defining functions in python:

```
def fact(x):
    if x==0:
        return 1
    else:
        return x*fact(x-1)
```

```
>>> fact(100)
933262154439441526816992388562667004907159682
643816214685929638952175999932299156089414639
761565182862536979208272237582511852109168640
0000000000000000000000L
```

- No type on input argument, remember duck typing!

Functions

Defining functions in python:

```
def fact(x):
    if x==0:
        return 1
    else:
        return x*fact(x-1)
```

```
>>> fact(100)
933262154439441526816992388562667004907159682
643816214685929638952175999932299156089414639
761565182862536979208272237582511852109168640
0000000000000000000000L
```

- No type on input argument, remember duck typing!
- Using lambda expression:

```
fact=lambda x:(x==0 and 1) or (x*fact(x-1))
```

Functions

Documentation string:

```
def fact(x):
    "Factorial of x"
    if x==0:
        return 1
    else:
        return x*fact(x-1)
```

Functions

Documentation string:

```
def fact(x):
    "Factorial of x"
    if x==0:
        return 1
    else:
        return x*fact(x-1)

>>> help(fact)
Help on function fact in module __main__:

fact(x)
    Factorial of x
```

Functions

Default parameters:

```
def fact(x,k=1):
    "kth factorial of x"
    if x<=0:
        return 1
    else:
        return x*fact(x-k,k)
```

Functions

Default parameters:

```
def fact(x,k=1):
    "kth factorial of x"
    if x<=0:
        return 1
    else:
        return x*fact(x-k,k)

>>> fact(6)
720
>>> fact(6,2)
48
>>> fact(k=2,x=6)
48
```

Modules

- Most interesting things are available in separate modules/packages.
- Two “import” statements:
  - `import` module — imports in separate namespace
  - `from` module `import` fn — current namespace
  - `from` module `import *` — imports everything

```
>>> import math
>>> math.sqrt(25)
5.0
>>> from math import sqrt
>>> sqrt(25)
5.0
>>> from math import *
>>> sin(1.57)
0.99999968293183461
```

Modules

`math`

- ceil(x), fabs(x), floor(x), ...
- exp(x), log(x,base=e), sqrt(x), ...
- cos(x), sin(x), tan(x), ...

`cmath`

- Same as `math`, but for complex numbers

`os`

- popen(command), system(command), ...
- chdir(path), listdir(path), rename(src,dst), stat(path), ...

`time`

- time(), localtime(secs), mktime(t), ...

NumPy

- "The fundamental package needed for scientific computing with Python"
- Module is Free, documentation costs (Book: "Guide to NumPy")
- Main class: `numpy.array`
- Homogeneous arrays, all entries must have same type
- Can give type explicitly:
  ```
  a = array( [ 1, 2, 3 ], float )
  ```
- Multidimensional:
  ```
  A = array( [[ 1, 2, 3 ], [ 4, 5, 6 ]] )
  ```
- Can change shape: `A.shape=(3, 2)` $\longrightarrow$
  ```
  A == array( [[1, 2], [3, 4], [5, 6]] )
  ```

Lists and arrays

Arrays act differently from lists:

```
>>> x=[1,2,3,4]
>>> x
[1, 2, 3, 4]
>>> x*2
[1, 2, 3, 4, 1, 2, 3, 4]
>>> from numpy import array
>>> y=array(x)
>>> y
array([1, 2, 3, 4])
>>> y*2
array([2, 4, 6, 8])
```

Creating arrays

```
>>> from numpy import *
>>> arange(0, 3, 0.5)
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5])
>>> zeros((2,3))
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> ones(4, int)
array([ 1,  1,  1,  1])
>>> fromfunction(lambda x,y:x**2+y, (3,4))
array([[ 0.,  1.,  2.,  3.],
       [ 1.,  2.,  3.,  4.],
       [ 4.,  5.,  6.,  7.]])
>>> eye(2)
array([[ 1.,  0.],
       [ 0.,  1.]])
```

Using arrays

```
>>> A = fromfunction(lambda x,y:x**2+y, (3,4))
>>> A[1,0:4]
array([ 1.,  2.,  3.,  4.])
>>> A[:,::2]
array([[ 0.,  2.],
       [ 1.,  3.],
       [ 4.,  6.]])
>>> A.transpose()
array([[ 0.,  1.,  4.],
       [ 1.,  2.,  5.],
       [ 2.,  3.,  6.],
       [ 3.,  4.,  7.]])
>>> A.diagonal()
array([ 0.,  2.,  6.])
```

Using arrays

The `matrix` class: like `array` but does matrix multiplication

```
>>> M=matrix(A)
>>> A*A
array([[  0.,   1.,   4.,   9.],
       [  1.,   4.,   9.,  16.],
       [ 16.,  25.,  36.,  49.]])
>>> M*M
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python2.5/site-packages/numpy/
    return N.dot(self, asmatrix(other))
ValueError: matrices are not aligned
>>> M*M.transpose()
matrix([[  14.,   20.,   38.],
        [  20.,   30.,   60.],
        [  38.,   60.,  126.]])
```

Pylab (matplotlib)

- Matplotlib: 2D plotting library
- Commands similar to matlab plotting commands
- Python module is called `pylab`
- Use with ipython: `ipython -pylab`
- Examples:

```
In [1]: x=arange(0,5.1,0.1)
In [2]: y=sin(x)
In [3]: plot(x,y,'ro-')
Out[3]: [<matplotlib.lines.Line2D instance at 0x17b6ef0>]
In [4]: hist(y,10,hold=False)
Out[4]:
(array([10,  3,  2,  2,  2,  5,  4,  5,  5, 13]),
 array([ -9.99923258e-01,  -7.99973572e-01,  -6.00023885e
        -4.00074199e-01,  -2.00124513e-01,  -1.74827261e-
         1.99774859e-01,   3.99724545e-01,   5.99674231e-
         7.99623917e-01]),
 <a list of 10 Patch objects>)
```

Pylab (matplotlib)

```
In [5]: [X,Y]=meshgrid(x,x)
In [6]: Z=sin(X)*cos(Y)
In [7]: clf()
In [8]: contourf(X,Y,Z)
Out[8]: <matplotlib.contour.ContourSet instance at 0x1dd2
In [9]: plot(t/5*sin(t)+2.5,t/5*cos(t)+2.5,'k-')
Out[9]: [<matplotlib.lines.Line2D instance at 0x1ddeef0>]
```

You can use some TEX-like commands in text, labels, etc.:

```
In [10]: xlabel('$\\sum x(\\theta)+y(\\theta)$')
Out[10]: <matplotlib.text.Text instance at 0x1dd2f38>
```

Logarithmic axes:

```
In [11]: semilogx()
Out[11]: []
In [12]: loglog()
Out[12]: []
```

NEST

- PyNest: Python bindings for NEST
- `./configure --enable-pynest`
- Intended as substitute for SLI

```
>>> import nest
>>> neuron = nest.Create("iaf_neuron")
>>> vm = nest.Create("voltmeter")
>>> nest.Connect(vm, neuron)
>>> nest.Simulate(2)
-70
-70
-70
-70
-70
-70
-70
-70
-70
-70
-70
```

NEST

- Moving data between SLI and PyNest:

```
>>> nest.slirun("/a 1 def")
>>> nest.slirun("a")
>>> a = nest.slipop()
>>> a
1
```

Neuron

- Neuron also has a python interface
- Intended as complement of hoc language(?)

```
>>> import hoc
>>> h = hoc.HocObject?()
>>> h('obfunc newvec() { return new Vector($1) }')
>>> v = h.newvec(5)
>>> h('objref m')
>>> h('m = new Matrix(2,3)')
>>> h.m.x[1][2] = 12
```

Classes

Creating classes in python:

mymodule.py

```
class myclass:
    def __init__(self,x):
        self.value=x
        print "New myclass instance created"

    def get_value(self):
        return self.value

    def __add__(self, other):
        return myclass(self.value + other.get_value())

    def __repr__(self):
        return str(myclass)+'('+str(self.value)+')'
```

```
>>> import mymodule
>>> o=mymodule.myclass(17)
New myclass instance created
```

Gtk

```
import gtk

window = gtk.Window(gtk.WINDOW_TOPLEVEL)
window.set_title("Example")
window.connect("destroy", gtk.main_quit)

hbox = gtk.HBox()
window.add(hbox)

button = gtk.Button("Click me")
hbox.add(button)

def when_clicked(widget):
    print "Button was clicked"

button.connect("clicked",when_clicked)

window.show_all()
gtk.main()
```

Threads

Two threading modules available:

- `thread`
- `threading`

threading

```
import threading

class MyThread(threading.Thread):
    def run(self):
        print "This will run in a thread"

MyThread().start()
```

thread

```
import thread

def threadfunc(value):
    print "This function will run in a thread"
    print "Value: "+str(value)

thread.start_new_thread( threadfunc, ('spam',) )
```

Gtk with threads

```
import gtk
import gobject
import thread

gobject.threads_init()

window = gtk.Window(gtk.WINDOW_TOPLEVEL)
window.set_title("Example")
window.connect("destroy", gtk.main_quit)
hbox = gtk.HBox()
window.add(hbox)
button = gtk.Button("Click me")
hbox.add(button)
def when_clicked(widget):
    print "Button was clicked"

button.connect("clicked",when_clicked)

window.show_all()
thread.start_new_thread(gtk.main,())
```