An Introduction to Python

Prof. Thomas Breuel

Introduction to Python and NumPy

- **developed in 1991 by Guido van Rossum**
- **originally developed as a teaching language**

- **used at...**
  - Google
  - Gnome desktop
  - Yahoo
  - Google AppEngine
  - open source numerical computing, AI
  - bioinformations, astrophysics

Implementations

- CPython — de facto standard implementation
- Jython — Python+libs on the JVM (can call Java)
- IronPython — Python+libs on the CLR (can call C#)
- Pyrex — Python with declarations (fast code)
- (Google Python implementation based on LLVM)

Python is multi-paradigm

- **procedural / imperative programming**
  - assignment, loops, procedures
- **object-oriented**
  - classes, inheritance, encapsulation, mutability
- **functional programming**
  - closures, recursion, list comprehensions

Python libraries

- **“batteries included”**
  - strings, regex, networking, ...
- **scientific programming**
  - NumPy, SciPy, Matplotlib, ...
- **3D visualization**
  - VTK, VPython, Mayavi
- **gaming**
  - SDL
- **GUI**
  - xlib, Tk, Gtk, Qt, wx, Windows, Cocoa
- **web applications**
  - TurbuGears, Django, Twisted
- **unit testing**

Python code

```
def fib(x):
    if x<2: return 1
    return fib(x-1) + fib(x-2)
```

- **note**
  - block structure indicated by indentation
  - not a lot of special characters

Python code

```
def quicksort(l):
    if len(l)<2: return l
    pivot = l[0]
    low = [x for x in l if x<pivot]
    mid = [x for x in l if x==pivot]
    high = [x for x in l if x>pivot]
    return quicksort(low)+mid+quicksort(high)
```

- **note**
  - just typed it in and it worked (test edge cases, though)
  - garbage collection
  - dynamic typing
  - strong typing
  - list comprehensions

Python command line programs

```
#!/usr/bin/python
# simple grep-like program

import os,sys,re

regex = re.compile(sys.argv[1])
stream = open(sys.argv[2])
for line in stream.readlines():
    line = line[:-1]
    if regex.search(line)>=0:
        print line
```

- often need to deal with lots of data → command line programs
- use optparse for command line parsing

Python command line programs

```
>>> 3+4
7
>>> import os
>>> os.popen("ls","r")
<open file 'ls', mode 'r' at 0xb29468>
>>>  help(os.popen)
Help on built-in function popen in module posix:
popen(...)
    popen(command [, mode='r' [, bufsize]]) -> pipe}
    Open a pipe to/from a command returning a file 
object.
>>>
```

- write small functions and test out interactively
- experiment with APIs
- use the interactive help function
- use reload(module) when necessary

Python objects

```
class Counter:
    def __init__(self):
        self.count = 0
    def increment(self,by=1):
        self.count += by
    def reset(self,to=0):
        self.count = 0
    def report(self):
        return self.count

c = Counter()
c.increment()
c.report()
c.reset()
```

- single dispatch
- multiple inheritance
- self is an explicit arg
- constructor is __init__
- note default args

Python datatypes

- **scalar datatypes**
  - integers, floating point numbers, immutable strings
- **lists / arrays: l = [1,2,3] ; print l[1]**
  - mutable, extensible 1D arrays
- **tuples: t = (1,2,3) ; print t[1]**
  - immutable, non-extensible 1D arrays
- **numerical arrays: a = array([1,2,3]) ; print a[1]**
  - mutable, multidimensional, SIMD
- **dictionaries: d = {1:1,2:4,3:9} ; print d[1]**
  - implemented as hash tables
- **objects**
- **modules**

Python modules

- **there are many standard modules**
- **you need to import things before you can use them**
  - e.g., import os,sys,string,re
- **after import os**
  - os is a module object
  - os.popen is the popen function in the os module
- **you can import selectively**
  - e.g., from os import popen
  - then you can use the function directly
  - the module remains not imported (no os object)
- **if you don’t do anything else, a source file “foo.py” is considered a “module” and you can import it via import foo**

Python types

- Assignment never copies, always passes references.
- x==y tests for value equality
- x is y tests for identity
- isinstance(a,t) tests for whether a is of type t
- int(x) converts to an int or raises an error
- list(x) converts to a list or raises an error
- type(x) returns the type (e.g., str, list, tuple)

```
>>> isinstance(3,list)
False
>>> isinstance(3,int)
True
```

Errors and Exceptions

- Python catches runtime errors (type errors, numerical errors, etc.)
- runtime errors are indicated by exceptions
- exceptions can be handled with try ... except ... else and try ... finally (like Java)

```
try:
    x = 1 + None
except TypeError:
    print "oops"
else:
    print "no oops"
finally:
    print "finally"
```

Python code

```
for animal in ["lion"]:
    print animal\newline

for i in range(99,0,-1):
    print i,"bottles of beer on the wall"
```

- **note**
  - iteration in Python is a for loop over a collection
  - iterations over numerical ranges are like iterations over collections

Python iterators

```
def fibonacci():
    a,b = (0,1)
    while 1:
        a,b = (b,a+b)
        yield b

for x in fibonacci():
    if x>1000000: break
    print x
```

- **note**
  - you can generate a sequence in a loop and then “yield”
  - (also observe parallel tuple assignment)

array subscripting syntax

- a[10] – return the 11th element of a
- a["foo"] – return the value associated with "foo" (only dicts)
- a[-1] – return the last element of a
- a[1:3] – return a list consisting of the second and third elements (NB: end is exclusive)
- a[1:] – return a list of everything other than the first element
- a[:-1] – return everything other than the last element
- a[:] – return a copy of the entire list

NumPy arrays

```
>>> from numpy import *
>>> a = array( [ 10, 20, 30, 40 ]
>>> a
array([10, 20, 30, 40])
>>> b = arange( 4 )
>>> print b
array([0 1 2 3])
```

- **see on-line tutorial for more information**

  http://www.scipy.org/Tentative_NumPy_Tutorial

NumPy arrays

```
>>> from numpy import *

# Element-wise operations
>>> array([1,2,3]) * array([3,2,1])
array([3, 4, 3])

# Dot product
>>> dot(array([1,2,3]),array([3,2,1]))
8

# zeros and ones
>>> zeros(5)
array([ 0.,  0.,  0.,  0.,
>>> ones(3)
array([ 1.,  1.,  1.])
```

NumPy multidimensional arrays

```
>>> zeros((2,2))
array([[ 0.,  0.],
       [ 0.,  0.]])

>>> array([[1,2],[3,4]])
array([[1, 2],
       [3, 4]])

>>> a.dtype
dtype('int64')

>>> a = array([[1,2],[3,4]])
>>> a.shape
(2, 2)

>>> a.shape=4
>>> a
array([1, 2, 3, 4])
```

arrays have multiple views

```
>>> a = array([[1,2],[3,4]])
>>> a.shape
(2, 2)>>> a.reshape(4,1)
array([[1],
       [2],
       [3],
       [4]])
>>> a
array([1, 2, 3, 4])}
>>> a.reshape(2,2)[1,1]=99
>>> a
array([ 1,  2,  3, 99])
>>>
```

mismatches and recycling

```
# recycling
>>> ones(2)+ones((2,2))
array([[ 2.,  2.],
       [ 2.,  2.]])

# shape mismatch
>>> ones(4)+ones((2,2))
Traceback (most recent call last):
File "<stdin>", line 1, in <module>}
ValueError: shape mismatch: objects cannot be
broadcast to a single shape
>>>
```

mismatches and recycling

```
# These are equivalent in expressions

ones((5,1))

ones(5)[:,newaxis]

# combine powerfully with other ops

arange(5)[:,newaxis] * arange(5)[newaxis,:]
```

arrays as images

```
from numpy import *
from pylab import imshow

imshow(arange(10)[:,newaxis]
        * arange(10)[newaxis,:])

show()
```

array shifting by subranges

```
>>> a = zeros((100,100))
>>> a[30:40,30:40] =
ones((10,10))
>>> a[:-1,:] -= a[1:,:]
>>> imshow(a)
<matplotlib.image.AxesImage instance at
0x159bd88>
>>> show()
```

animation in NumPy/Matplotlib

```
# see www.scipy.org/Cookbook/Matplotlib/Animations

from numpy import *
from pylab import *
import time

ion() # turn on interactive

tstart = time.time()
image = zeros((100,100))
x = linspace(0,10,100)
y = linspace(0,10,100)
image = sin(x)[:,newaxis]*sin(y)[newaxis,:]
im = imshow(image)

n = 500
for i in linspace(0,10,n):
 image = sin(x+i)[:,newaxis]*sin(y+i)[newaxis,:]
 im.set_array(image)
 draw()

print 'FPS:' , n/(time.time()-tstart)
```

arrays as images

arguments are recycled