An Introduction to Python (for CompuCell3D)

Randy Heiland

heiland@indiana.edu

Aug 2011

Python - a scripting language

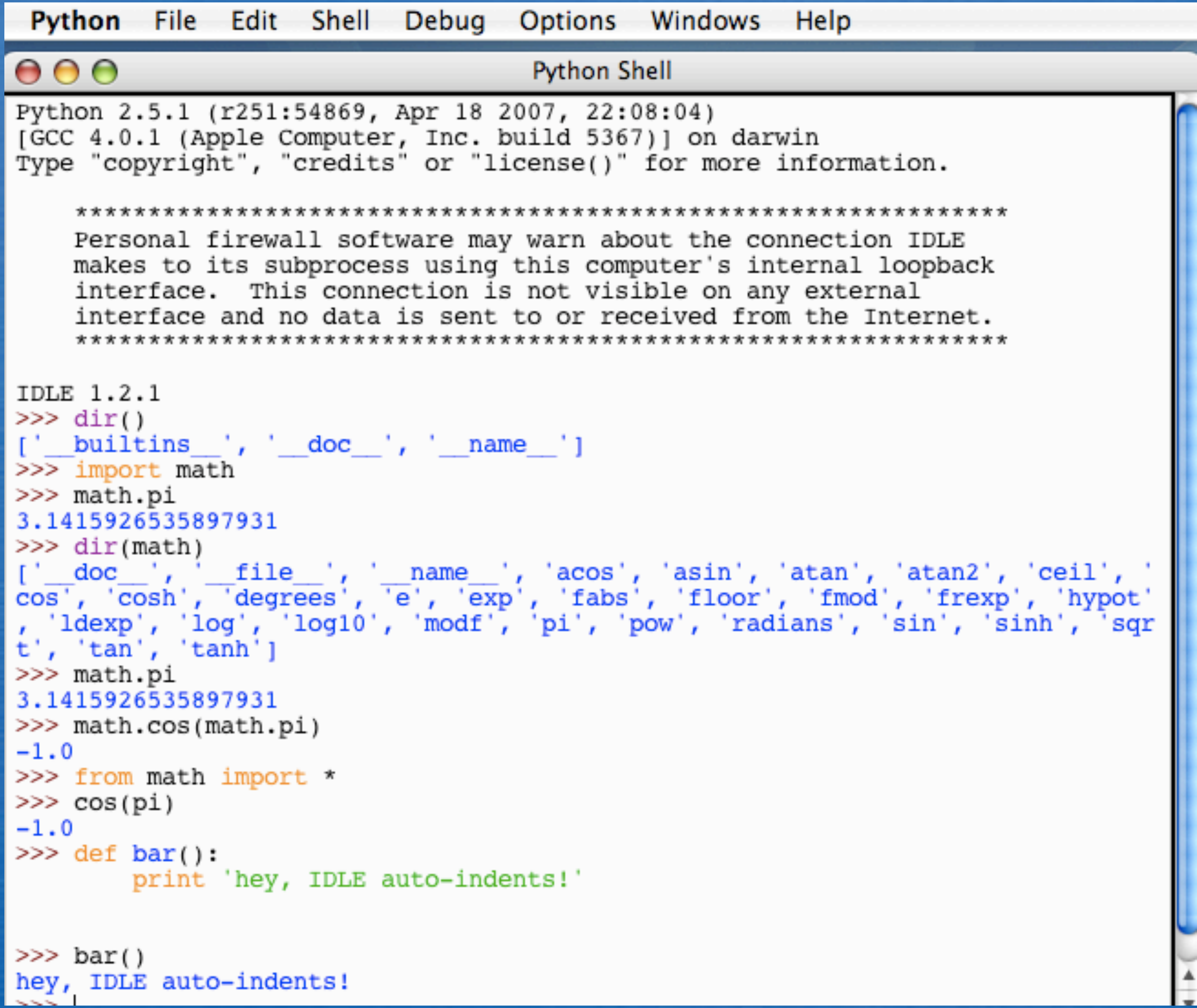
- www.python.org
- Released in 1991 by Guido van Rossum (at Google since 2005 where Python is heavily used)
- “Python is absolutely free, even for commercial use”
- Dynamically typed
- Strongly typed
- Auto memory mgt

Features

- High-level lng;   Syntax (minimal, clean)
- Interpreted; Interactive
  (--> rapid prototyping/development)
- Glue-iness, Wrap-ability (www.swig.org)
- Introspection
- History with science apps
- Vibrant (community) and evolving (language)

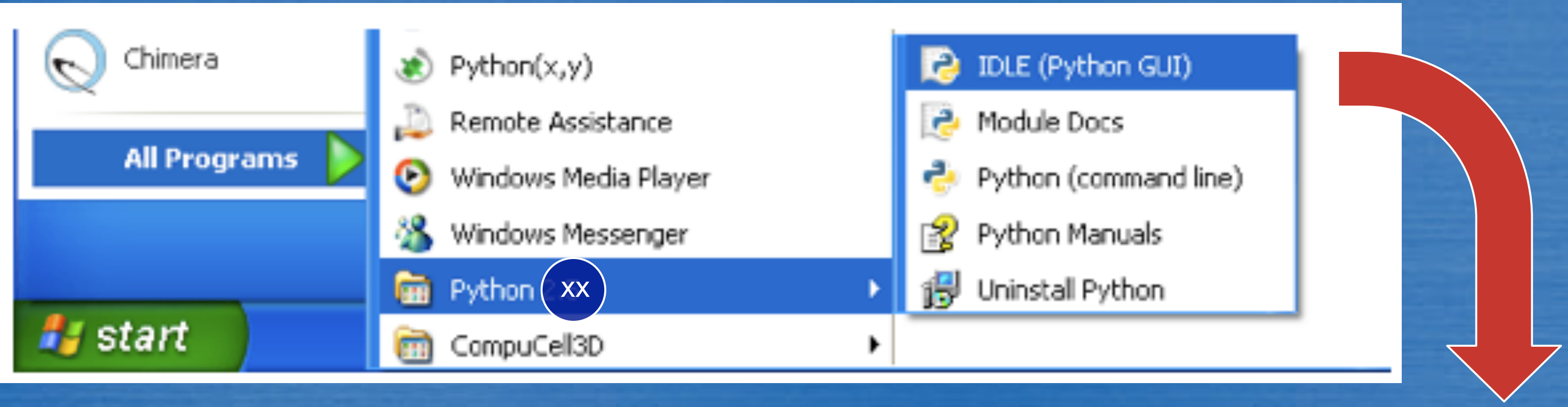
(free) Python shells

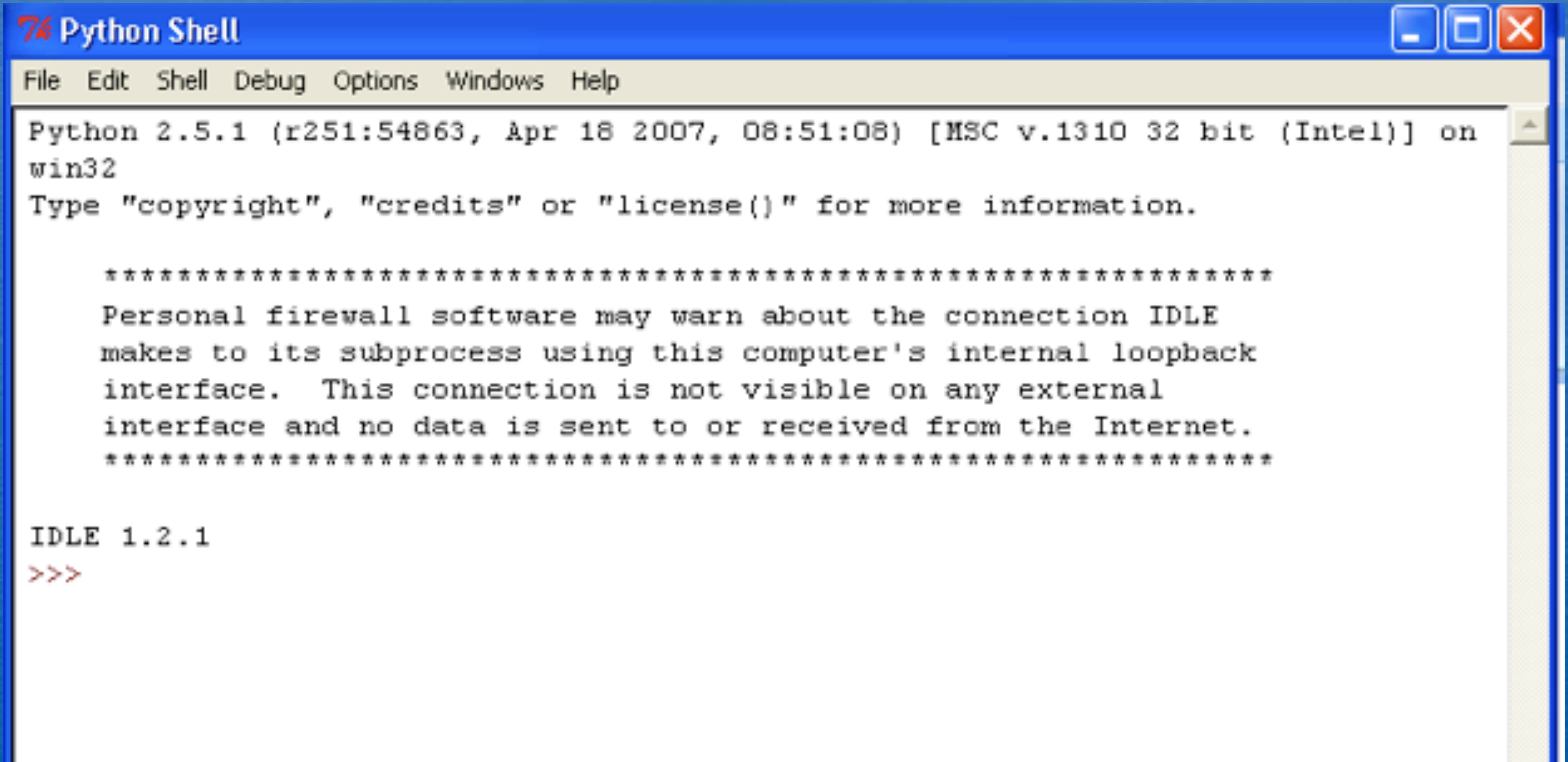
- IDLE
- Twedit++ (for CC3D)
- ConTEXT (Windows only)



```
Python 2.5.1 (r251:54869, Apr 18 2007, 22:08:04)
[GCC 4.0.1 (Apple Computer, Inc. build 5367)] on darwin
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2.1
>>> dir()
['__builtins__', '__doc__', '__name__']
>>> import math
>>> math.pi
3.1415926535897931
>>> dir(math)
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan', 'atan2', 'ceil', '
cos', 'cosh', 'degrees', 'e', 'exp', 'fabs', 'floor', 'fmod', 'frexp', 'hypot'
, 'ldexp', 'log', 'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqr
t', 'tan', 'tanh']
>>> math.pi
3.1415926535897931
>>> math.cos(math.pi)
-1.0
>>> from math import *
>>> cos(pi)
-1.0
>>> def bar():
        print 'hey, IDLE auto-indents!'


>>> bar()
hey, IDLE auto-indents!
```

Starting a Python shell





```
Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2.1
>>>
```

OSX/Linux

From a terminal window:

```
$ python
(…build info printed…)
Type "help", "copyright", "credits" or "license"
   for more information.
>>>
```

FYI

In the following slides, we will demonstrate various features of Python directly in the Python shell (the interactive interpreter).

But for a Python-based application, all Python code will reside in files and will simply get executed with the interpreter:

```
% python myproject.py
```

Python is Object-Oriented

- Object = attributes + methods
  - attributes = things you know
  - methods = things you can do

Getting started

http://docs.python.org/tutorial/

```
% python
Python 2.7.1 …
```

Start from the command line, or IDLE, or…

dynamic typing

```
>>> x = 'Euler'
>>> x
'Euler'
>>> x = 2.718

>>> x += 5j
>>> x
(2.718+5j)
>>> x.real, x.imag
(2.718, 5.0)
```

x.real

Python uses the ‘dot’ syntax to access attributes of objects

```
% python
Python 2.7.1 …
```

interpreter

dynamic typing

```
>>> x = 'Euler'
>>> x
'Euler'
>>> x = 2.718

>>> x += 5j
>>> x
(2.718+5j)
>>> x.real, x.imag
(2.718, 5.0)
```

strong typing

```
>>> x='2'
>>> y=2
>>> x+y
Traceback (most recent call last):
  File "<stdin>", line 1, in
     <module>
TypeError: cannot concatenate
     'str' and 'int' objects
>>> int(x) + y
4
```

Introspection – very handy

```
>>> x=2.718+5j
>>> x
(2.718+5j)
>>> type(x)
<type 'complex'>
>>> dir(x)
[ ‘’ …,'conjugate', 'imag', 'real']
>>> x.imag
5.0
```

```
>>> x.conjugate
<built-in method conjugate of
    complex object at 0x12110>
>>> x.conjugate()
(2.718-5j)
>>> type(x.imag)
<type 'float'>
>>> type(x.conjugate)
<type 'builtin_function_or_method'>
```

Python Function: ‘def’

```
>>> def foo(x,y):
...       z = x*y
...       return z
...
>>> print foo(3,4)
12

>>> foo(’fun',5)
funfunfunfunfun
```

}

Indentation and alignment required for statement blocks;
No {…} in Python

(get over it!)

If you're off by even 1 space

```
>>> def test():
...   x=3
...   y=4
...    return (x+y)
  File "<stdin>", line 4
    return (x+y)
    ^
IndentationError: unexpected indent
>>>
```

Also beware of mixing spaces and tabs

Control flow

```
>>> if x < 0:
...       x = 0
...       print 'Negative changed to zero'
... elif x == 0:
...       print 'Zero'
... elif x == 1:
...       print 'Single'
... else:
...       print 'More'
```

```
>>> for n in range(5):
...    for k in range(10,14):
...       print n,k
...
0 10
0 11
0 12
0 13
1 10
1 11
…
```

Comments

# this is a comment (in any column)

”””this triple-quote thingy is a comment

that may extend over

multiple lines”””

User input

```
>>> val = input('Hey genius, give me some some input: ')
Hey genius, give me some input: 13
>>> val
13
>>> val = input('Hey genius, give me some input: ')
Hey genius, give me some input: ”Python is fun"
>>> val
”Python is fun"
```

Python Modules in Files

- A module is a file containing Python definitions and statements
- <module-name>.py
- ‘import’ the module
- PYTHONPATH env var
- .pyc = byte-compiled, arch-independent file

- E.g. in file ‘foo.py’ :

```
def foo(x,y):
    z = x*y

    return z
```

File I/O

- http://docs.python.org/tutorial/inputoutput.html

```
fp = open('people.dat', 'r')
line = fp.readline()
print 'line=', line,      # "," suppresses additional line feed
items = line.split()
print 'items=', items
name = items[0]
birth = int(items[1])   # or float()
death= int(items[2])
print name,' lived ',(death-birth),' years'

# to read all lines, one at a time
#for line in fp:
#  …
fp.close()
```

people.py

```
% python people.py
line= Einstein 1879 1955
items= ['Einstein', '1879',
'1955']
Einstein  lived  76  years
```

Error handling: try-except

```
try:
    fp = open(“bogus.dat”, “r”)
except:
    print “Couldn’t open file”
```

Also use a try/except to check for null objects, division by zero, etc.

ImportError

```
>>> import Cell
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named Cell
>>> import sys
>>> print sys.path
```

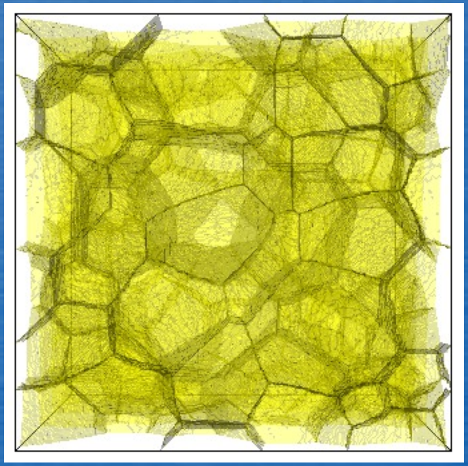
Prints all known locations for modules; check your PYTHONPATH env var

Simplify module names

```
import CompuCellSetup
or:
import CompuCellSetup as ccs

ccs.getScreenshotDirectoryName()
```

Python as “glue”

```
# renderAll.py
# Operate all on .vtk files in the current dir. In this case, we
# (1) invoke a C pgm (foam.c) that parses the file and creates newField.vtk, then
# (2) invoke a python script that renders that file to generate the 3D cells.
Import os

vtkDir = '/Users/heiland/Documents/Glazier/Vidhya’
for idx in range(0,9950,50):
    fname = 'Step_%05d.vtk' % idx                  # ~= C-style printf stmt
    count = idx / 50
    imageFile = 'image%04d' % count

    cmd = './foam ' + vtkDir+'/'+fname
    print cmd
    os.system(cmd)        # invoke any executable

    cmd = 'python renderFoam.py ' + imageFile
    print cmd
    os.system(cmd)
```



Basic Data Structures

- Lists
- Tuples
- Sets
- Dictionaries

```
>>> a = [2.718, 'fred', 13]
>>> dir(a)
[…'append', 'count', 'extend', 'index',
    'insert', 'pop', 'remove', 'reverse',
    'sort’]
>>> a[1] = 56              (mutable)
>>> a
[2.718, 56, 13]
>>> a.sort()
>>> a
[2.718, 13, 56]
>>> a.insert(2,'fun')
>>> a
[2.718, 13, 'fun', 56]
```

Tuples

- Similar to Lists, but immutable

```
>>> temp, pressure = (13.0, 98.3),(45,46)
>>> type(temp)
<type 'tuple'>
>>> temp
(13.0, 98.29999999999997)
>>> pressure
(45, 46)
>>> temp(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object is not callable
>>> temp[0]
13.0
```

Dictionary (key,value pairs)

```
>>> mydict= {'ecm' : 0, 'condensing' : 1,
    'noncondensing' : 2}
>>> mydict['ecm']
0
>>> type(mydict)
<type 'dict'>
>>> len(mydict)
2
>>> mydict["dim"] = (100,100,1)      insert new key,value
>>> mydict
{'dim': (100, 100, 1), 'ecm': 0, 'condensing': 1,
    'noncondensing': 1}
```

Sets

- Similar to Lists, except unordered and does not allow duplicate values.
- Elements of a set are neither bound to a number (like list and tuple) nor to a key (like dictionary).
- Much faster for huge number of items; fast data insertion, deletion, and membership testing

Array (in standard lib)

- Similar to Lists, but homogeneous elements

```
>>> from array import *
>>> x=array('f',[1.0,1.1,1.2,1.3])          float
>>> type(x)
<type 'array.array'>
>>> x[3]
1.2999999523162842
>>> x=array('d',[1.0,1.1,1.2,1.3])          double
>>> x[3]
1.3
```

http://docs.python.org/library/array.html

Large standard library (Python: “batteries included”)

Just one example module

- >>> import os
- >>> dir(os)
- […'_copy_reg', '_execvpe', '_exists', '_exit', '_get_exports_list', '_make_stat_result', '_make_statvfs_result', '_pickle_stat_result', '_pickle_statvfs_result', '_spawnvef', 'abort', 'access', 'altsep', 'chdir', 'chmod', 'chown', 'chroot', 'close', 'confstr', 'confstr_names', 'ctermid', 'curdir', 'defpath', 'devnull', 'dup', 'dup2', 'environ', 'error', 'execl', 'execle', 'execlp', 'execlpe', 'execv', 'execve', 'execvp', 'execvpe', 'extsep', 'fchdir', 'fdopen', 'fork', 'forkpty', 'fpathconf', 'fstat', 'fstatvfs', 'fsync', 'ftruncate', 'getcwd', 'getcwdu', 'getegid', 'getenv', 'geteuid', 'getgid', 'getgroups', 'getloadavg', 'getlogin', 'getpgid', 'getpgrp', 'getpid', 'getppid', 'getsid', 'getuid', 'isatty', 'kill', 'killpg', 'lchown', 'linesep', 'link', 'listdir', 'lseek', 'lstat', 'major', 'makedev', 'makedirs', 'minor', 'mkdir', 'mkfifo', 'mknod', 'name', 'nice', 'open', 'openpty', 'pardir', 'path', 'pathconf', 'pathconf_names', 'pathsep', 'pipe', 'popen', 'popen2', 'popen3', 'popen4', 'putenv', 'read', 'readlink', 'remove', 'removedirs', 'rename', 'renames', 'rmdir', 'sep', 'setegid', 'seteuid', 'setgid', 'setgroups', 'setpgid', 'setpgrp', 'setregid', 'setreuid', 'setsid', 'setuid', 'spawnl', 'spawnle', 'spawnlp', 'spawnlpe', 'spawnv', 'spawnve', 'spawnvp', 'spawnvpe', 'stat', 'stat_float_times', 'stat_result', 'statvfs', 'statvfs_result', 'strerror', 'symlink', 'sys', 'sysconf', 'sysconf_names', 'system', 'tcgetpgrp', 'tcsetpgrp', 'tempnam', 'times', 'tmpfile', 'tmpnam', 'ttyname', 'umask', 'uname', 'unlink', 'unsetenv', 'urandom', 'utime', 'wait', 'wait3', 'wait4', 'waitpid', 'walk', 'write']

Another standard lib module

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan', 'atan2',
    'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp', 'fabs', 'floor', 'fmod',
    'frexp', 'hypot', 'ldexp', 'log', 'log10', 'modf', 'pi', 'pow', 'radians',
    'sin', 'sinh', 'sqrt', 'tan', 'tanh']
```

```
>>> math.cos(math.pi)
-1.0
```

Alternatively:

```
>>> from math import *
>>> cos(pi)
```

But beware of namespace clashes

Yet another

```
>>> import random
>>> dir(random)
[…, 'randint', 'random', 'randrange', 'sample', 'seed', 'setstate', 'shuffle', 'triangular',
    'uniform', 'vonmisesvariate', 'weibullvariate']
>>> print random.random
<built-in method random of Random object at 0x100897420>
>>> print random.random()
0.656260480257
>>> print random.randint()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: randint() takes exactly 3 arguments (1 given)
>>> print random.randint.__doc__
Return random integer in range [a, b], including both end points.

>>> print random.randint(1,100)
13
```

Oops!

What’s in the standard lib??

http://docs.python.org/tutorial/stdlib.html

GUIs

- Tkinter – built-in Tcl/Tk module, but …

Alternative 3rd party:

- wxWidgets
- PyQt (GPL license)
- PyOpenGL

Summary

- Python is an open source scripting lng (interpreted language)
- Used frequently in science applications
- Can wrap C,C++,etc code in Python
- Great for gluing together applications

- …advanced topics follow…

Classes

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

‘self’ =ref to current Instance (~ ‘this’ in C++,Java

Beyond the standard lib - installing community modules

There are many freely available community (3rd party) modules available. You just need to install them - e.g., from a shell:

```
% python setup.py install
```

If the installation method is not obvious from the module’s download site, rf:

http://docs.python.org/inst/inst.html

http://peak.telecommunity.com/DevCenter/EasyInstall

Beyond the standard lib - community packages - e.g. NumPy

Google for ‘numpy’, download/install it.

```
>>> from numpy import arange
>>> tvals = arange(0.,5., 0.1)
>>> tvals
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,
        1.1,  1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9,  2. ,  2.1,
        2.2,  2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9,  3. ,  3.1,  3.2,
        3.3,  3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,  4.3,
        4.4,  4.5,  4.6,  4.7,  4.8,  4.9])
>>> tvals[1]
0.10000000000000001
```

MATLAB® vs. NumPy

www.scipy.org/NumPy_for_Matlab_Users

```
>>> from numpy import *
>>> b = array( [ (1.5,2,3), (4,5,6) ] )
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
>>> b.shape
(2, 3)
>>> dir(linalg)
['LinAlgError', '__builtins__', '__doc__', '__file__', '__name__',
    '__path__', 'cholesky', 'det', 'eig', 'eigh', 'eigvals', 'eigvalsh', 'info',
    'inv', 'lapack_lite', 'linalg', 'lstsq', 'norm', 'pinv', 'qr', 'solve', 'svd',
    'tensorinv', 'tensorsolve', 'test']
```

NOTE: NumPy is written in C and therefore quite fast

Basic Data Structures (cont.)

List comprehensions - concise way to map or filter lists

```
>>> a
[2.718, 13, 56]
>>> import math              # in the std lib
>>> [math.exp(x) for x in a]
[15.149991940878165, 442413.39200892049,
   2.0916594960129961e+24]
```

Basic Data Structures (cont.)

dictionary - 1:1 reln between key-value pairs

(~hash in Perl; ~Hashtable in Java)

```
>>> simParam = dict(lattice='square', xdim=100,ydim=100,hex=1)
>>> simParam= {'lattice':'square', 'xdim':100, 'ydim':100, 'hex':1}

>>> simParam['xdim']
100
```

Note 2 ways to create dict

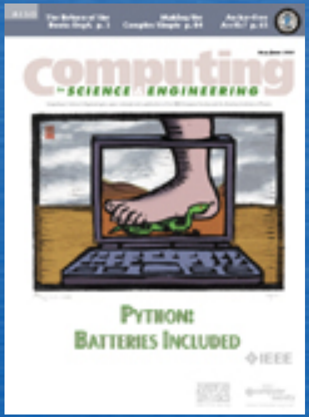
Python vs. Perl syntax
e.g. hash (dictionary vs. assoc array)

```
for i in range(5):
  x={}
  for j in range(3):
    x[j]=i + j
  print x

--> outputs:
{0: 0, 1: 1, 2: 2}
{0: 1, 1: 2, 2: 3}
{0: 2, 1: 3, 2: 4}
{0: 3, 1: 4, 2: 5}
{0: 4, 1: 5, 2: 6}
```

```
for $i (0 .. 6000-1) {
  %x=();
  for $j (0 .. 1000-1) {
    $x{$j}=$i + $j;
    $x{$j};
  }
}
```

Testimonials/Users

- Google
- YouTube
- LLNL, ANL, LBL, LANL
- …



May/June 2007 issue of Computing in Science & Engineering (CiSE)

Useful 3rd party Python pkgs

- Data/Analysis

DBs:	mysql-python
Numerics:	NumPy
RPy	R from Python
Storage (HDF5):	PyTables

- Visualization

SciVis:	VTK (Python bindings)
Plotting:	matplotlib
Workflows:	VisTrails

3rd party pkgs (cont)

- Science

Chemistry:	UCSF Chimera, PyMOL, VMD
Bioinfo:	BioPython
Physics:	PyROOT
Imaging:	ITK (Python bindings)
GPGPU:	PyCUDA

  - www.scipy.org/Topical_Software
  - www.python.org/about/apps

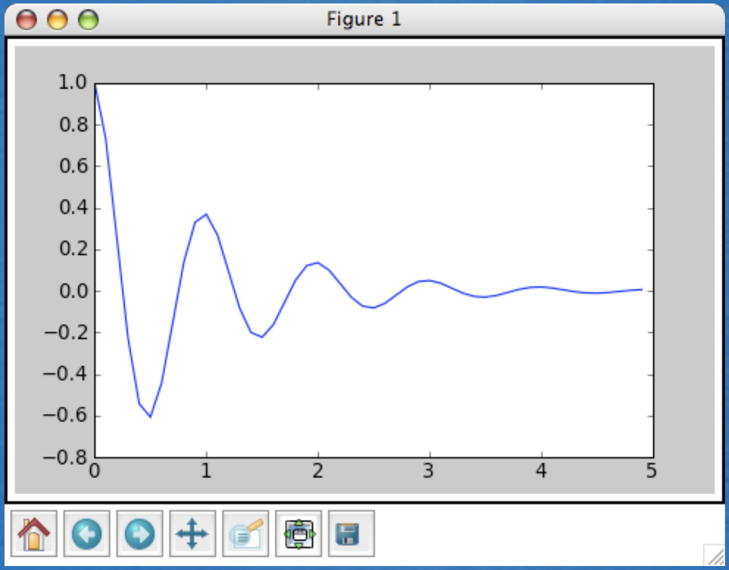
3rd party pkgs (cont)

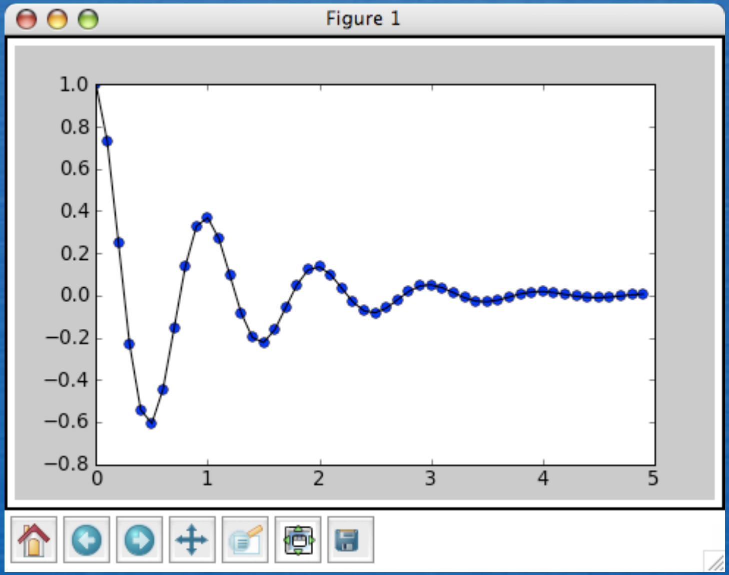
- Infrastructure

• Web dev:	django, TurboGears
• Web Services:	ZSI, xml
• HPC:	pyMPI, MYMPI, pp (parallelpython.com)
• Grid:	pyGlobus, pyGridWare
• Star-P	Python client (bought by MS)

matplotlib

```
>>> from pylab import *
>>> def my_func(t):
        s1 = cos(2*pi*t)
        e1 = exp(-t)
        return s1*e1
>>> tvals = arange(0., 5., 0.1)
>>> plot(tvals, my_func(tvals))
[<matplotlib.lines.Line2D object at 0x16ca9d50>]
>>> show()
```





```
>>> plot(tvals, my_func(tvals), 'bo', tvals, my_func(tvals), 'k')
>>> show()
```

Create a .pif

```
ct = ('Wall','Bacterium','Macrophage')
xdel,ydel = 20,20
xmax,ymax = 99,99
x0, y0 = 10,10
count = 0
maxSquares = 3
for icol in range(3):
  x1 = x0 + xdel
  if icol == 1:
    maxSquares = 4
    y0 = 0
  elif icol == 2:
    maxSquares = 3
    y0 = 10
  for idx in range(maxSquares):
    y1 = y0 + ydel
    if y1 > ymax: y1=ymax
    print count,ct[0],x0,x1, y0,y1,0,0
    y0 = y1 + 10
    count += 1
  x0 += 30
```

```
$ python bm.py
0 Wall 10 30 10 30 0 0
1 Wall 10 30 40 60 0 0
2 Wall 10 30 70 90 0 0
3 Wall 40 60 0 20 0 0
4 Wall 40 60 30 50 0 0
5 Wall 40 60 60 80 0 0
6 Wall 40 60 90 99 0 0
7 Wall 70 90 10 30 0 0
8 Wall 70 90 40 60 0 0
9 Wall 70 90 70 90 0 0
```

Exercises

- If you have a favorite mini-project, do it
- Calculate/print linearly interpolated values between (x1,y1) and (x2,y2) (and have user enter those points)
- Write a ‘Fruit’ class, create a default ‘color’ method, then subclass it with a ‘Banana’ class and override the method. Test.

fruit.py

```
class Fruit:
    def __init__(self):
        self.sweet = True
    def color(self):
        return 'red'

class Banana(Fruit):
    def color(self):
        return 'yellow’
    def shape(self):
        return 'shape is not round'

class Tomato(Fruit):
    def __init__(self):
        self.sweet = False
```

```
print '----- Fruit -----'
fruit = Fruit()
print 'Sweet? ',fruit.sweet
print fruit.color()

print '----- Banana -----'
fruit = Banana()
print fruit.sweet
print 'Sweet? ',fruit.sweet
print fruit.color()
print fruit.shape()

print '----- Tomato -----'
fruit = Tomato()
print 'Sweet? ',fruit.sweet
print fruit.color()
print fruit.shape()
```

```
% python fruit.py
----- fruit 1 -----
Sweet?  True
red
----- fruit 2 -----
True
Sweet?  True
yellow
shape is not round
----- fruit 3 -----
Sweet?  False
red
Traceback (most recent call last):
  File "fruit.py", line 35, in <module>
    print fruit.shape()
AttributeError: Tomato instance has no attribute 'shape'
```

- Create a class, Acct. Attributes -> __init__
- Add a method for deposits
- Add a method for debits
- Add a method for balance
- Create an instance of Acct and do stuff…