LONI Python tutorial

HPC@LSU

Bhupender Thakur

Outline

- **Introduction**
  Why python?
- **Installation: Know your python setup**
  Setting up python and modules
- **Python language reference**
  Programming in python
- **Modules: Basic**
  A powerful aspect of python programming: Sys, Os
- **Modules: Advanced**
  A look at some advanced modules: Numpy, Scipy, Ipython, mpi4py

Goals

- **Understand its advantages and disadvantages**
  Especially if you are primarily a C/Fortran programmer
- **Understand how to set up modules you need**
  Don’t come asking for “Somepy” to be installed
- **Learn python programming basics**
  Stop using Fortran/C/bash calls in python
- **Learn to take advantage of Modules**
  Learn to find useful tools which do more than just display colored text

Introduction

Python is a dynamic programming language : executes at runtime many common behaviors that other languages might perform during compilation.

Why would you use python?

- Open source code and libraries;
- Plenty of useful modules to satisfy varying tasks;
- Faster to code in, shorter development time;
- Portable across various platforms.

Why you would not use python?

- Slower at runtime;
- Modules can be taxing on memory;
- Module objects are somewhat opaque and harder to dig into.

Installation

Know what version of you have/want?
Current production versions are Python 2.7.5 and Python 3.3.2.
You will find Python 2.7.3 on most LONI and HPC machines

Python 3.3.2 (May 15, 2013)
Python 3.2.5 (May 15, 2013)
Python 3.1.5 (April 10, 2012)
Python 3.0.1 (February 13, 2009)
Python 2.7.5 (May 15, 2013)
Python 2.7.3 (April 10, 2012)
Python 2.6.8 (April 10, 2012)
Python 2.5.6 (May 26, 2011)
Python 2.4.6 (December 19, 2008)
Python 2.3.7 (March 11, 2008)
Python 2.2.3 (May 30, 2003)
Python 2.1.3 (April 8, 2002)
Python 2.0.1 (June 2001)

Python on LONI and HPC

Python on Queenbee: 2.7.3 *Recommended*

```
$soft add +python-2.7.3-gcc-4.3.2
```

or add the key to your ~/.soft file

```
$ vi ~/.soft
+gcc-4.3.2
+python-2.7.3-gcc-4.3.2
@default
```

On SuperMikeII look for +Python-2.7.3-gcc-4.3.2 key

Installation

Get Python

```
$ wget http://www.python.org/ftp/python/2.7.3/Python-2.7.3.tar.bz2
$ tar -jxvf Python-2.7.3.tar.bz2
$ cd Python-2.7.3
```

Your usual make and install

```
$ ./configure --prefix=$HOME/python --exec-prefix=$HOME/python
$ make
$ make install
```

If you choose one compiler(gcc or Intel), stick to it for building all your modules

Installation: Modules

You can add modules locally/globally depending on sudo rights

```
$ wget http://sourceforge.net/projects/numpy/files/NumPy/1.6.2/numpy-1.6.2.tar.gz/download
$ wget http://sourceforge.net/projects/scipy/files/scipy/0.11.0/scipy-0.11.0.tar.gz/download

$ tar -zxvf numpy-1.6.2.tar.gz
$ tar -zxvf scipy-0.11.0.tar.gz

$ cd numpy-1.6.2
$ $HOME/python/bin/python setup.py config build --fcompiler=gnu95 install

$export BLAS=/usr/local/packages/lapack/3.4/lib/libblas.a
$export LAPACK=/usr/local/packages/lapack/3.4/lib/liblapack.a

$cd scipy-0.11.0
$ $HOME/python/bin/python setup.py config build --fcompiler=gnu95 install
```

Alternate Installation

Python modules: You can locally add modules

Most Python tools can be installed by just

```
$python setup.py config build install
```

Use home/user build for local install

```
$python setup.py install –prefix= --exec-prefix=
                         --user=
                         --home=
```

Alternate Installation

Python modules: user scheme:

This scheme is designed to be the most convenient solution for users that don’t have write permission to the global site-packages directory or don’t want to install into it..

```
$python setup.py install –user=<dir>
```

Type of file	**Installation directory(unix)**	**Under Windows**
modules	*base*/lib/python*X.Y*/site-packages	*base*\Python*XY*\site-packages
scripts	*base*/bin	*base*\Scripts
data	*base*	*base*
C headers	*base*/include/python*X.Y*/*distname*	*base*\Python*XY*\Include *distname*

Alternate Installation

Python modules: home scheme:

Useful for maintaining a personal stash of Python modules.

```
$python setup.py install –home=<dir>
```

Type of file	Installation directory
modules	*home*/lib/python
scripts	*home*/bin
data	*home*
C headers	*home*/include/python/*distname*

(Supported on windows beyond 2.4. Mentally replace slashes with backslashes if you’re on Windows.)

Alternate Installation

Python modules: prefix scheme:

The “prefix scheme” is useful when you wish to use one Python installation to perform the build/install (i.e., to run the setup script), but install modules into the third-party module directory of a different Python installation

```
$python setup.py install –prefix=/usr/local/packages/python
```

Type of file	Installation directory
Python modules	*prefix*/lib/python*X.Y*/site-packages
extension modules	*exec-prefix*/lib/python*X.Y*/site-packages
scripts	*prefix*/bin
data	*prefix*
C headers	*prefix*/include/python*X.Y/distname*

(On Windows, --prefix option has traditionally been used to install additional packages in separate locations on Windows.)

Running python

Interactive python shell

```
[bthakur@qb3 ]$ python
Python 2.7.3 (default, Jun 17 2012, 16:26:01)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

As a script

```
[bthakur@qb3 ]$ python -c 'import sys;  print sys.version'
[bthakur@qb3 ]$ python script.py
```

For the adventurous: Try ipython

Python: Data Structures

Standard python data types:

Numeric Types :	int, float, long, complex
Sequence Types :	immutable: str, unicode, tuple
	mutable: list, bytearray, buffer
Iterator Types:	generators
Set Types:	set(mutable), frozenset
Mapping types :	dictionary
Others:	File Objects, memoryview type, Context Manager Types Other Built-in Types: Modules, Classes and class instances, functions, Methods, code objects, type objects,

http://docs.python.org/2/reference/datamodel.html#types

Note: Not all types may be available in prior releases

Python: Data Structures

Practical python data types:

Numbers : integers(normal, long), float(C- doubles), hexadecimal, binary, octal, complex

String : collection of characters

Tuple : ordered collection of arbitrary immutable objects

List : ordered collection of arbitrary mutable objects

Dictionary : unordered mutable collection of objects

Others: fractions, sets

http://docs.python.org/2/library/stdtypes.html

http://docs.python.org/2/tutorial/datastructures.html

- Note: python 2.x vs python 3.0 important differences exist. 3.0 only has one integer type which supports unlimited precision
- type(x) can be used to check the type of a variable

Basic python: Numbers

Integers and floats:

```
>>> a=3.142
>>> int(a), float(int(a))
(3, 3.0)
```

Hex, bin oct:

```
>>> print hex(64), oct(64), bin(64)
0x400 02000 0b1000000
```

Operations:

Basic arithmetic :

x+y, x*y,   x/y , x//y, x**y, x%y

//: Floor or integer division

Comparison(normal, chained) :

x<y , x<y<z

Bitwise operations :

x^2, x<<2 , x|1 , x&1

Math module:

```
>>> import math
>>> math.sqrt(144)
12
```

Random module:

```
>>> import random
>>> random.random()
>>> random.randint(1, 10)
>>> random.choice( [ ‘a’, ’b’ ,’c’ ] )
```

Sets:

```
>>> x= set('abc'); print x
set(['a', ’c’, ‘b'])
```

x - y, x | y, x & y

Basic python: Numbers

Integers and floats:

```
>>> a=3.142
>>> >>> int(a), float(int(a))
(3, 3.0)
```

Hex, bin oct:

```
>>> print hex(64), oct(64), bin(64)
0x400 02000 0b1000000
```

Operations:

Basic arithmetic :

x+y, x*y,   x/y , x//y, x**y, x%y

Comparison(normal, chained) :

x<y , x<y<z

Bitwise operations :

x<<2 , x|1 , x&1

Math module:

```
>>> import math
>>> math.sqrt(144), math.factorial(5)
(12.0, 120)
```

Random module:

```
>>> import random
>>> random.random()
>>> random.randint(1, 10)
>>> random.choice( [ ‘a’, ’b’ ,’c’ ] )
```

Sets:

A set object is an unordered collection of distinct hashable objects

```
>>> x= set('abc'); print x
set(['a', ’c’, ‘b'])
```

x - y, x | y, x & y

difference, union, intersection

Basic python: Strings

Python strings:

```
>>> str = 'Hello World!’;  print str
Hello World!
```

Indexing and slicing:

```
>>> print str[0:4], str[6:11]
Hell World
```

Repetition and concatenation:

```
>>> n=2;  print str*n
Hello World!Hello World!

>>> print str + ', ' + 'Good Morning'
Hello World!, Good Morning
```

Python strings:

```
>>> len(str)
26

>>> str=str.replace( 'Morn', 'Even' );
>>> print str
Hello World!, Good Evening

>>> str.find('o');
>>> str.count('l')
>>> str.upper()
```

Basic python: Strings

Python strings:

```
>>> str = 'Hello World!’;  print str
Hello World!
```

Indexing and slicing:

```
>>> print str[0:4], str[6:11]
Hell World
```

Repetition and concatenation:

```
>>> n=2;  print str*n
Hello World!Hello World!
```

```
>>> print str + ', ' + 'Good Morning'
Hello World!, Good Morning
```

Python strings:

```
>>> len(str)
26
```

```
>>> str=str.replace( 'Morn', 'Even' );
>>> print str
Hello World!, Good Evening
```

```
>>> str.find('o');
>>> str.count('l')
>>> str.upper()
```

Python strings are immutable: New copies are generated when using replace

Basic python: Strings

Example:

```
>>> import os
>>> a=os.uname()
>>> b=str(os.uname())
>>> b.strip(‘(’)
>>> b[0:2]=‘try’
TypeError

>>> c=list(os.uname)
>>> type(c)
<type ‘list’>
>>>c[0]=“*nix”
>>>type(c[0])
```

Python strings:

```
>>> len(str)
26

>>> str=str.replace( 'Morn', 'Even' );
>>> print str
Hello World!, Good Evening

>>> str.find('o');
>>> str.count('l')
>>> str.upper()
```

Basic python: Strings

Example:

```
>>> import os
>>> a=os.uname()
>>> b=str(os.uname())
>>> b.strip(‘(’)
>>> b[0:2]=‘try’
TypeError

>>> c=list(os.uname)
>>> type(c)
<type ‘list’>

>>>c[0]=“*nix”
```

Example:

```
>>> a=‘abc’
>>> b=‘uvw’
>>> c=[ (m+n) for m in a for n in b ]
>>> d=‘’
>>> for n in c:
>>>    d=d+str(n)
>>> print d
'auavawbubvbwcucvcw'
```

Exercise: Strings

Exercise 1:

Assign a string:

“a quick little brown fox jumps over the lazy dog”

a) Write a python code to capitalize the first letter of each word.

b) Write a python code to output a string with each word from the sentence spelled backwards, but in the same location.

Exercise 2:

A palindrome is a word, phrase, number, or other sequence of symbols or elements, whose meaning may be interpreted the same way in either forward or reverse direction. Examples: "Amore, Roma.", "A man, a plan, a canal: Panama."

Write a python script to check if a string is a palindrome

Exercise:Dictionary

Exercise 3:

Dictionary

Write a python program to create a contact list searchable by name to have phone numbers

Exercise 5:

List/functions

Create a list to generate few elements (smaller than 1000) which are products of two prime numbers

Basic python: Lists

Python lists:

- Variable length, heterogeneous ordered collection of arbitrary objects accessible by offset
- Mutable and nestable sequences
- Array of object references
- Powerful iterator and generator facilities
- Builtins

Random objects, Nesting:

```
 >>> L = [ 307, “Frey”,  “80900”, [“BT”,”AP”] ];
L[1]
  ‘Frey‘
```

Arbitrary, mutable, ordered objects

Shared references:

```
  L = []; M = L
   # modify both lists
  L.append(obj)
```

Ordered: indexing/slicing,

List iteration and generation

Built-ins: Searching/sorting

Building lists a.k.a List comprehension:

```
    >>> a = [ n for n in range(10) ];
    >>> b = [ x**2 for x in range(10) ];
   >>> c= [ x+y for x in “abc” for y in “123”];
    >>> b = ( [m,n] for m in a for n in a )
```

Basic python: Lists

Python lists:

- Variable length, heterogeneous ordered collection of arbitrary objects accessible by offset
- Mutable and nestable sequences. Shared references
- List comprehension and generators
- Built-ins: Searching/Sorting

Arbitrary objects:

```
>>> def f(a):a=a*2;return a;
>>> L=[ 307, “Frey”, “0900”, (“BT”,”AP”) , f ];
 L[1];  L[4](4)
 ‘Frey’; 8
```

Variable length, list modification:

```
>>> L.append(“a”)
>>> L.extend(M)
>>> L.insert(i,x), L.remove(x)
>>> L.pop([i])
```

Nestable:

```
 >>> L = [ [“a”, “b”, “c”], [1, 2] ];
L[0][1]
“b”
```

Shared references:

```
L = []; M = L
L.append(“a”)  # modifies both lists
```

Basic python: Lists

Python lists:

- Variable length, heterogeneous ordered collection of arbitrary objects accessible by offset
- Mutable and nestable sequences. Shared references
- List comprehension and generators
- Built-ins: Searching/Sorting

Arbitrary objects:

```
>>> def f(a):a=a*2;return a;
>>> L=[ 307, “Frey”, “0900”, (“BT”,”AP”) , f ];
 L[1];  L[4](4)
 ‘Frey’; 8
```

Variable length, list modification:

```
>>> L.append(“a”)
>>> L.extend(M)
>>> L.insert(i,x), L.remove(x)
>>> L.pop([i])
```

Nestable:

```
 >>> L = [ [“a”, “b”, “c”], [1, 2] ];
L[0][1]
“b”
```

Shared references:

```
L = []; M = L
L.append(“a”)  # modifies both lists
```

Basic python: Lists

Python lists:

- Variable length, heterogeneous ordered collection of arbitrary objects accessible by offset
- Mutable and nestable sequences. Shared references
- List comprehension and generators
- Built-ins: Searching/Sorting

List comprehension:

```
>>> L = ['a', 'b', ‘c']
>>> for s in L:
...     print s
```

More list comprehension:

```
>>>[ x**2 for x in range(10) ]
[0, 1, 4, 9, 19, 25, 36, 49, 64, 81]

 >>>  L = ['a', 'b', 'e']; M=[‘1', ‘2', ’3’]
 >>> [ x+y for x in L for y in M ]
['a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'e1', 'e2', 'e3']

>>> [(x, y) for x in [1,2] for y in [3,1] if x != y]
[(1, 3), (2, 3), (2, 1)]

>>> def is_even(a): return a%2 == 0
>>> [x for x in range(10) if is_even(x)]
[0, 2, 4, 6, 8]
```

Basic python: Lists

Python lists:

- Variable length, heterogeneous ordered collection of arbitrary objects accessible by offset
- Mutable and nestable sequences. Shared references
- List comprehension and generators
- Built-ins: Searching/Sorting

List iterators:

```
>>> s = 'abc'
>>> a=iter(s)
>>> a.next(),a.next(),a.next()
('a', 'b', 'c')
>>> a.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Built-in functions:

```
>>> L = ['a', 'b', ‘c']
>>> L.sort()
>>> L.append
>>> L.extend
```

Generators:

```
>>> [n**2 for n in range(10)]
>>> list(n**2 for n in range(10))
```

Basic python: Lists

Advanced list comprehension:

- Built-ins: Functional programming tools for list comprehension

Functional programming tools:

map:

Apply function to every item of iterable and return a list of the results.

```
>>> items = [1,2,3,4,5]
>>> def sqr(x): return x**2
>>> list (map(sqr, items))
[1,4,9,16,25]
```

filter:

Construct a list from those elements of iterable for which function returns true.

```
>>> def f(x): return x%2 != 0 and x%3 != 0
>>> filter(f, range(2, 25))
[5, 7, 11, 13, 17, 19, 23]
```

reduce:

Apply function of two arguments cumulatively to the items of iterable, from left to right, so as to reduce the iterable to a single value. calculates ((((1+2)+3)+4)+5).

```
>>> reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])
15
```

Basic python: functions

Python functions:

A reusable set of statements which can compute a result based on input parameters.

- Maximize code reuse and eliminate redundancy
- Procedural decomposition

```
def <name> (arg1, arg2, …):
      “doc string”
      <statements>
      return <value>
```

Example:

Typeless behavior

```
>>>def  mult(x, y):
        …   return x * y
>>> mult( 2,4 ); mult( ‘Hi’,2 )
8
‘HiHi’
```

Basic python: functions

Example: Passing values by reference:

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

def modify_list(la):
    "Modify a list"
    la=[10,11,12];
    print " List inside modify_list call \t" , la
    return

def modify_indx(la):
    for i in range(len(la)):
        la[i]+=10
    print " List inside modify_indx call \t" , la
    return

la=[0,1,2];
```

```
print " List initial \t\t\t", la
print "+-----------+"
modify_list(la);
print " List after  modify_list call\t", la
print "+-----------+"
modify_indx(la);
print " List after  modify_indx call\t", la
```

The first function creates a new list with a local scope which does not get passed to main routine.

Basic python: functions

Example: Passing values by reference:

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

def modify_list(la):
    "Modify a list"
    la=[10,11,12];
    print " List inside modify_list call \t" , la
    return

def modify_indx(la):
    for i in range(len(la)):
        la[i]+=10
    print " List inside modify_indx call \t" , la
    return

la=[0,1,2];
```

```
print " List initial \t\t\t", la
print "+-----------+"
modify_list(la);
print " List after  modify_list call\t", la
print "+-----------+"
modify_indx(la);
print " List after  modify_indx call\t", la
```

```
$ python func0.py
 List initial                       [0, 1, 2]
+-----------+
 List inside modify_list call       [10, 11, 12]
 List after  modify_list call       [0, 1, 2]
+-----------+
 List inside modify_indx call       [10, 11, 12]
 List after  modify_indx call       [10, 11, 12]
```

Exercise: functions

Example: Passing values by reference:

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

def modify_list(la):
    "Modify a list"
    la=[10,11,12];
    print " List inside modify_list call \t" , la
    return

def modify_indx(la):
    for i in range(len(la)):
        la[i]+=10
    print " List inside modify_indx call \t" , la
    return

la=[0,1,2];
```

```
print " List initial \t\t\t", la
print "+-----------+"
modify_list(la);
print " List after  modify_list call\t", la
print "+-----------+"
modify_indx(la);
print " List after  modify_indx call\t", la
```

Exercise:

a) Analyze and run script func0.py

b) Analyze and run func1.py using output from disassembly module included therein. Find the in-place addition.

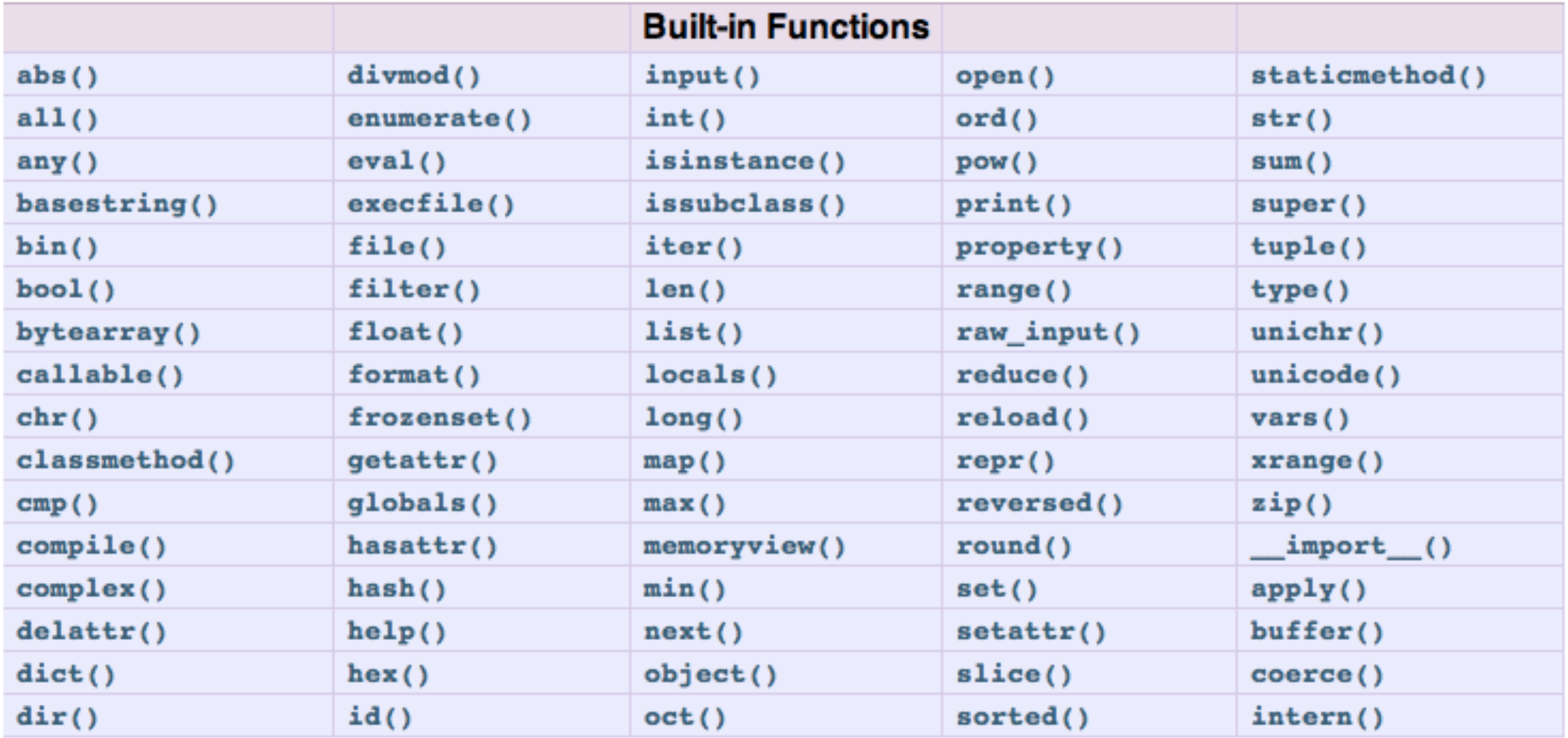
c) Run symb0.py to find the difference between the three functions calls in func1.py. In which function is the list declared global? In which function is it assigned?

Basic python: lambda functions

lambda functions

```
>>> g = lambda x: x**2
>>> print g(8)

>>> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>> print filter(lambda x: x % 3 == 0, foo)
[18, 9, 24, 12, 27]
>>> print map(lambda x: x * 2 + 10, foo)
[14, 46, 28, 54, 44, 58, 26, 34, 64]>>>
>>> print reduce(lambda x, y: x + y, foo)
139
```

Basic python: functions



		Built-in Functions		
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	apply()
delattr()	help()	next()	setattr()	buffer()
dict()	hex()	object()	slice()	coerce()
dir()	id()	oct()	sorted()	intern()

Source: http://docs.python.org/2/library/functions.html

Basic python: Dictionary

Python dictionaries:

- Unordered collection of arbitrary objects
- Accessed by key, not offset: slicing and indexing operations unavailable
- Variable length, heterogeneous and nestable
- Mutable mapping (map keys to values)
- Tables(hash) of object references

Example:

```
>>>  table = { ‘Python’: ‘Guido van Rossum’,
               ‘Perl’:     ‘Larry Wall’,
               ‘C++’:      ‘Bjarne Stroustrup’ }
>>> language = ‘Python’
>>> creator= table[ language ]
‘Guido van Rossum’
```

Basic python: Dictionary

Python dictionaries:

- Unordered collection of arbitrary objects
- Accessed by key, not offset: slicing and indexing operations unavailable
- Variable length, heterogeneous and nestable
- Mutable mapping (map keys to values)
- Tables(hash) of object references

Key-value pair assignment:

```
>>>D= {‘phys’:1, ‘chem’:2, math:3}; D
{'chem': 2, 'phys': 1, 'math': 3}
```

Membership: find value by key:

```
>>> D[‘math’]
3
>>> D.keys(); D.values(); D.items()
[1, 2, 3]
['chem', 'phys', 'math']
```

Attributes of object D:

```
>>> dir(D)
```

Basic python: Dictionary

Python dictionaries:

- Unordered collection of arbitrary objects
- Accessed by key, not offset: slicing and indexing operations unavailable
- Variable length, heterogeneous and nestable
- Mutable mapping (map keys to values)
- Tables(hash) of object references

List to dictionary:

```
>>> l= [ ‘a’, ‘b’, ‘c’, ‘d’ ]
>>> d={}
>>> for k in l:
>>>       d{k}=k*2
>>> print d
{’a': ‘aa’, ’b': ‘bb’, ’c': ‘cc’, ‘d’: ‘dd’}
```

Or :

```
>>> d= {l[n]:’’ for n in range(0,len(n)) }
```

Alternatively:

```
>>> from itertools import izip
>>> n=iter(l)
>>> b=dict( izip(n,n) )
```

Basic python: Files

Python files

- Use file iterators for reading files
- Content is strings, not objects
- Files are buffered and seekable.
  Flush forces buffered data to disk

```
>>> f = open( ‘test_file’, ‘w’ )
>>> print f
<open file 'test_file', mode 'w' at
0x1004bd250>


f.read(),  f.readline(),  f.readlines(),
f.seek(0)


>>> for line in f:
            print line
>>> with open(’test_file', 'r') as f:
...     read_data = f.read()


line.rstrip,
```

Basic python: Files

pickle:
Stores native python objects

struct: packed binary data
Store packed binary data

Example pickle:

```
  >>> D= {‘a’:1, ‘b’:2}
  >>> F= open(data.pkl, ‘wb’)
  >>> import pickle
  >>> pickle.dump(D, F)
  >>> F.close()

 >>> F= open(data.pkl, ‘rb’)
 >>> E= pickle.load(F)
 >>> E
{'a': 1, 'b': 2}
```

Conditionals: If-else

Python if else:

```
 if test1:
            statements1
  elif test2:
            statements2
  …
  else:
            statements
```

Example:

```
    >>>x= int(raw_input(”Enter x: ”))
     >>> if x < 0:
      ...      print 'x is –ve’
     ...    elif x == 0:
     ...       print 'x is 0'
     ...    else:
     ...       print 'x is +ve’’
```

Basic python: While Loop

While executes a block of statements until a condition is satisfied

```
while <test>:
        <statements1>
else:
        <statements2>
```

```
while True:
        print ‘This can go on forever’
```

```
a=0; b=9
while a<b:
        #print (a, end=‘ ‘) # python 3.0
        print a
        a+=1
```

Basic python: While Loop

continue:
lets you jump to top of the loop

pass:

placeholder statement

break:

immediate exit from the loop

```
while <tests1>:
        <statements1>
        if <tests2>: break
        if <tests3>: continue
else:
        <statements2>
```

```
a=10
while a:
        a=a-1
        if a%2 !=0: continue
        print a
```

```
while True: pass
```

```
while True:
        name=input(‘Enter name:’)
        if name==‘stop’: break
        age=input(‘Enter age:’)
        print(‘Hello’, name, int(age)*2)
```

Basic python: For Loop

For loop steps through items in an ordered sequence

```
for <target> in <object>:
          <statements1>
else:
          <statements2>
```

```
for x in [1,2,3,4,5]:
                sum += x
                prod *= x
```

```
values= [“abc”, 123, (1,2), 3.14]
keys= [(1,2), 2.1]
for key in keys:
                if key in values:
                    print (key, “found”)
                else:
                    print (key, “not found”)
```

Basic python: For Loop

Range can be use to iterate over sequence

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> range(1,11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> range(0,10,3)
[0, 3, 6, 9]
```

```
for n in range(2,10):
    for x in range(2,n):
        if n%x ==0:
            print n,’is’, x, ‘*’, n/x
            break
    else:
        print n, “is prime”
```

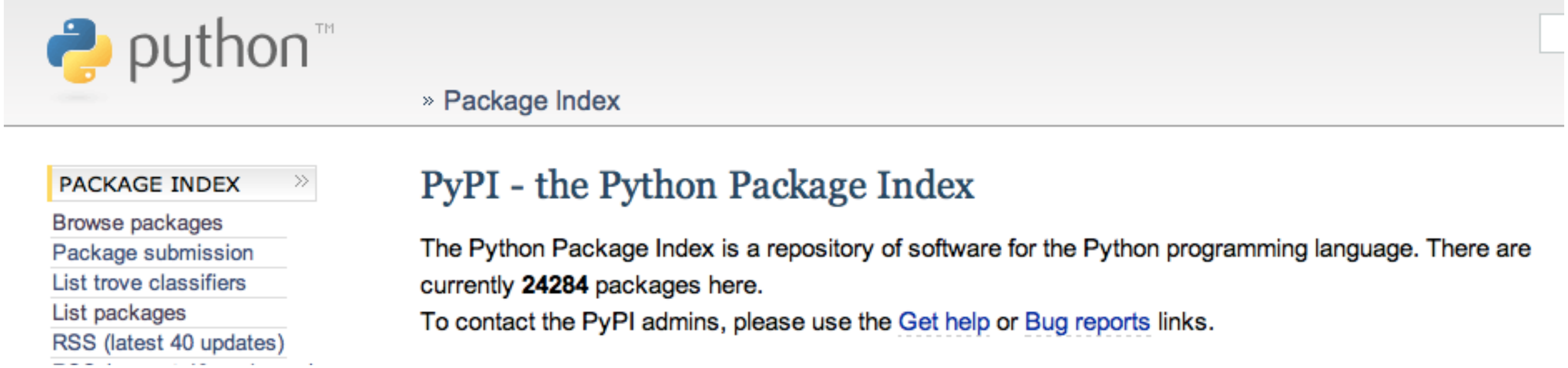
Basic python: Exception

```
try:
    mult( ‘Hi’, 2.5 )
except TypeError:
    print ‘There is an error here’
```

```
+-- Exception
     +-- StopIteration
     +-- StandardError
     |    +-- BufferError
     |    +-- ArithmeticError
     |    |    +-- FloatingPointError
     |    |    +-- OverflowError
     |    |    +-- ZeroDivisionError
     |    +-- AssertionError
     |    +-- AttributeError
     |    +-- EnvironmentError
     |    |    +-- IOError
     |    |    +-- OSError
     |    |         +-- WindowsError (Windows)
     |    |         +-- VMSError (VMS)
     |    +-- EOFError
     |    +-- ImportError
     |    +-- LookupError
     |    |    +-- IndexError
     |    |    +-- KeyError
     |    +-- MemoryError
     |    +-- NameError
     |    |    +-- UnboundLocalError
     |    +-- ReferenceError
     |    +-- RuntimeError
     |    |    +-- NotImplementedError
     |    +-- SyntaxError
     |    |    +-- IndentationError
     |    |         +-- TabError
     |    +-- SystemError
     |    +-- TypeError
     |    +-- ValueError
     |         +-- UnicodeError
     |              +-- UnicodeDecodeError
     |              +-- UnicodeEncodeError
     |              +-- UnicodeTranslateError
     +-- Warning
          +-- DeprecationWarning
```

Python: Using modules

http://pypi.python.org/pypi



PyPI - the Python Package Index

The Python Package Index is a repository of software for the Python programming language. There are currently **24284** packages here.

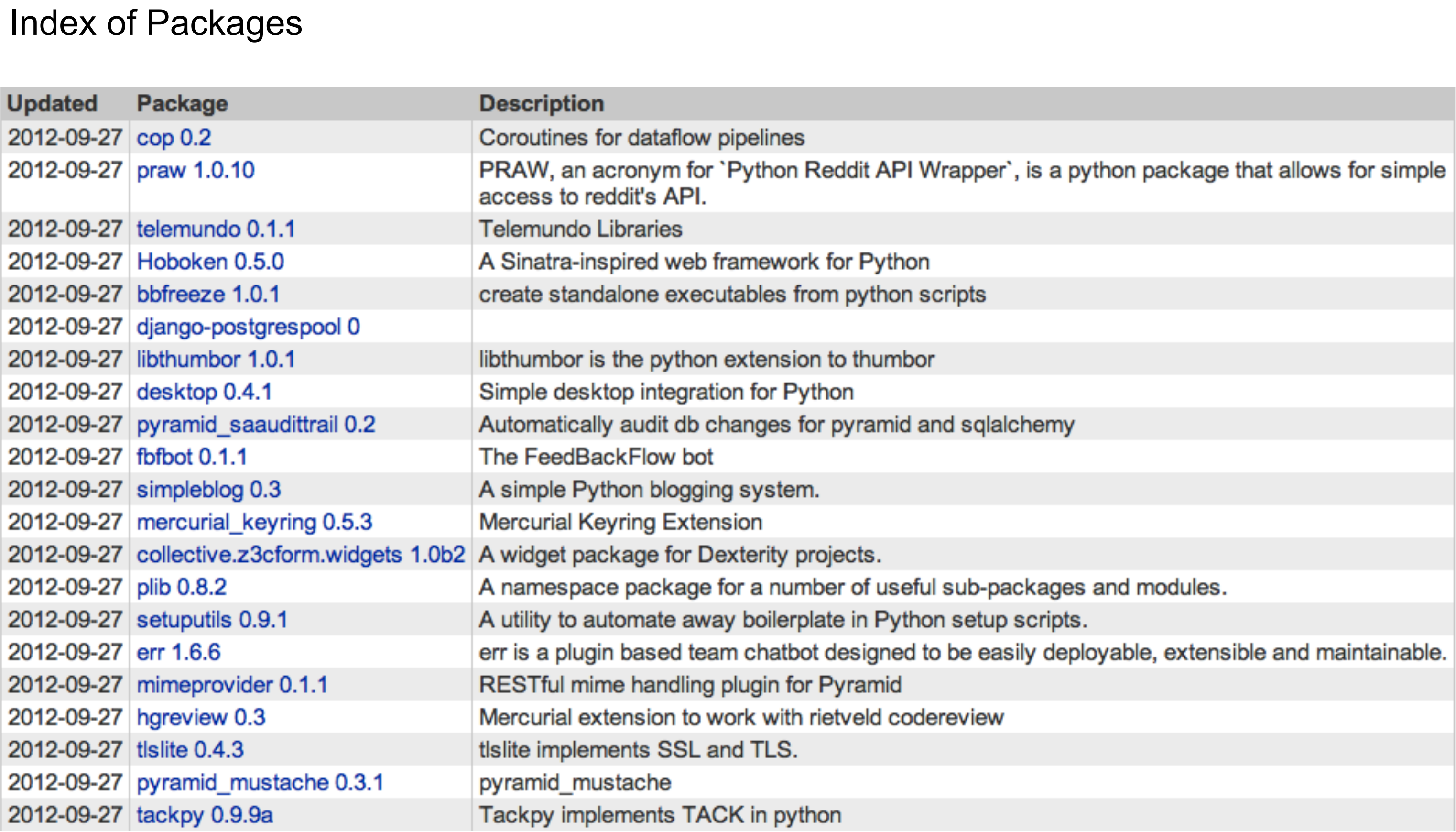
To contact the PyPI admins, please use the Get help or Bug reports links.

Python: Using modules

Modules

- sys, os, re
- math, numpy, scipy
- subprocess, ipython, re
- paramiko, sqlite
- Mpi4py, pyxml, matplotlib

Python: Using modules



Index of Packages

Updated	Package	Description
2012-09-27	cop 0.2	Coroutines for dataflow pipelines
2012-09-27	praw 1.0.10	PRAW, an acronym for `Python Reddit API Wrapper`, is a python package that allows for simple access to reddit's API.
2012-09-27	telemundo 0.1.1	Telemundo Libraries
2012-09-27	Hoboken 0.5.0	A Sinatra-inspired web framework for Python
2012-09-27	bbfreeze 1.0.1	create standalone executables from python scripts
2012-09-27	django-postgrespool 0	
2012-09-27	libthumbor 1.0.1	libthumbor is the python extension to thumbor
2012-09-27	desktop 0.4.1	Simple desktop integration for Python
2012-09-27	pyramid_saaudittrail 0.2	Automatically audit db changes for pyramid and sqlalchemy
2012-09-27	fbfbot 0.1.1	The FeedBackFlow bot
2012-09-27	simpleblog 0.3	A simple Python blogging system.
2012-09-27	mercurial_keyring 0.5.3	Mercurial Keyring Extension
2012-09-27	collective.z3cform.widgets 1.0b2	A widget package for Dexterity projects.
2012-09-27	plib 0.8.2	A namespace package for a number of useful sub-packages and modules.
2012-09-27	setuputils 0.9.1	A utility to automate away boilerplate in Python setup scripts.
2012-09-27	err 1.6.6	err is a plugin based team chatbot designed to be easily deployable, extensible and maintainable.
2012-09-27	mimeprovider 0.1.1	RESTful mime handling plugin for Pyramid
2012-09-27	hgreview 0.3	Mercurial extension to work with rietveld codereview
2012-09-27	tlslite 0.4.3	tlslite implements SSL and TLS.
2012-09-27	pyramid_mustache 0.3.1	pyramid_mustache
2012-09-27	tackpy 0.9.9a	Tackpy implements TACK in python

Python: Using modules

Get help:

pydoc <name> ...

Show text documentation on something.

dir():

Without arguments, return the list of names in the current local scope. With an argument, attempt to return a list of valid attributes for that object.

help(‘scipy')

Get help on any python object

Python: Using modules

Module search

Python module sys:

This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.

Python module pp:

The pprint module provides a capability to “pretty-print” arbitrary Python data structures in a form which can be used as input to the interpreter.

```
>>> import pprint as pp
>>> import sys

>>> a=sys.modules

>>> type(a)
>>> pp.pprint (a.keys())
>>> pp.pprint (a.values())

>>> a[‘os’]
```

Python: module ‘sys’

Python module sys:

This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.

sys.modules:	List modules
sys.path:	Show path
sys.argv:	List arguments
sys.platform:	Platform type
sys.executable:	Python executable
sys.maxint:	

```
>>> import sys
>>> sys.modules

>>> sys.path
>>> sys.path.append(newpath)

>>> sys.platform
```

Python: module ‘os’

Python module os:

- os.getcwd()
- os.environ[‘HOME’],
- os.environ[‘PWD’],
- os.chdir(), os.fchdir()
- os.fdopen()
- os.popen()
- os.getloadavg()
- os.fchmod(), os.fchown(fd, uid, gid)

```
#!/usr/bin/python

import os, sys

# First go to the "/var/www/html" directory
os.chdir("/var/www/html" )

# Print current working directory
print "Current working dir : %s" % os.getcwd()

# Now open a directory ”dir"
fd = os.open( “dir", os.O_RDONLY )

# Use os.fchdir() method to change the dir
os.fchdir(fd)

# Print current working directory
print "Current working dir : %s" % os.getcwd()

# Close opened directory.
os.close( fd )
```

Python: module ‘os’

Python module os:

```
os.path.isfile()
os.path.isdir()
os.path.islink()

os.walk()
```

Example:

```
>>> path = os.getcwd()
>>> files = os.listdir(path)
>>> os.path.isfile( files[10] )
```

Example:

```
>>> for top, dirs, files in os.walk(path)
>>>     print top
>>>     print dirs
>>>     print files
```

Python: subprocess

Python module subprocess:

The **subprocess** module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. This module intends to replace several other, older modules and functions, such as:

This module defines one class called Popen:
class subprocess.Popen(args, bufsize=0, executable=None, stdin=None, stdout=None, stderr=None, preexec_fn=None, close_fds=False, shell=False, cwd=None, env=None, universal_newlines=False, startupinfo=None, creationflags=0)

os.system
os.spawn*
os.popen*
popen2.*
commands.*

Python: subprocess

Python module subprocess:

```
#!/usr/bin/env python
import os,sys
import subprocess
subprocess.check_call(['ls','-al']);
subprocess.call('echo $HOME', shell=True);

$python sub1.py
total 16
drwxr-xr-x 2 bthakur Admins 4096 Oct 23 05:12 .
drwxr-xr-x 9 bthakur Admins 4096 Oct 23 05:12 ..
-rw-r--r-- 1 bthakur Admins  136 Oct 23 00:50 sub1.py
-rw-r--r-- 1 bthakur Admins  345 Oct 23 00:58 sub2.py
/home/bthakur
```

Python: Regular expression

Python regular expressions:

Load re module

Compile a search, or not

Find search string in a given string

re.match, re.search

- Module level functions take two arguments: search string and string to be searched
- backslashes are not handled in any special way in a string literal prefixed with 'r'. to match a literal backslash, one might have to write '\\\\' as the pattern string

Example:

```
>>> a=re.search('ain', 'The rain in spain')
>>> a.group(), a.start(), a.end()
('ain', 5, 8)

>>> a=re.match('ain', 'The rain in spain')
>>> print a
None
```

Python: Regular expression

re metacharacters: **. ^ $ * + ? { } [ ] \ | ( )**

[abc$^y]	Character class: Metacharacters are not active inside this ^ compliments the set by excluding the character
\	the backslash can be followed by various characters to signa various special sequences. It’s also used to escape all the metacharacters
\d \D	match a digit, non-digit characters (equivalent to [0-9] [^0-9])
\s \S	match whitespace, non-white space characters respectively
\w \W	match alpha-numeric [a-zA-Z0-9_], non-alpha numeric respectively
.	It matches anything except a newline character, also there’s (re.DOTALL)

```
>>> str = ’send email to sys-help@loni.org'
>>> match = re.search(r'[\w.-]+@[\w.-]+', str)
>>> match.group()
'sys-help@loni.org’
```

```
>>> str = ’send email to sys-help@loni.org’
>>> match = re.search('([\w.-]+)@([\w.-]+)', str)
>>> match.group(), match.groups()
('sys-help@lsu.edu', ('sys-help', 'lsu.edu'))
```

Python: Regular expression

Python regular expressions:

re.findall

- returns a list of matching strings
- has to create the entire list before it can be returned as the result.

Example:

```
>>> import sys,os, re
>>> str='axbaybcazbd'
>>> a=re.compile('a.b')
>>> m=a.findall(str)
>>> m
['axb', 'ayb', 'azb']
```

Python: module ‘re’

Python regular expressions:

```
import re


re.search
re.findall
re.finditer
```

```
str='The rain in spain stays mainly in the plains.'

a0='ain'
a1='.ain.'
a2='[a-z]ain'
a3='[a-z]+ain'
a4='[a-z]+ain[a-z]+'
a5='[a-z]*ain[a-z]*’
```

```
$ python re0.py
['ain', 'ain', 'ain', 'ain']
['rain ', 'pain ', 'mainl', 'lains']
['rain', 'pain', 'main', 'lain']
['rain', 'spain', 'main', 'plain']
['mainly', 'plains']
['rain', 'spain', 'mainly', 'plains']
```

Exercise: module ‘re’

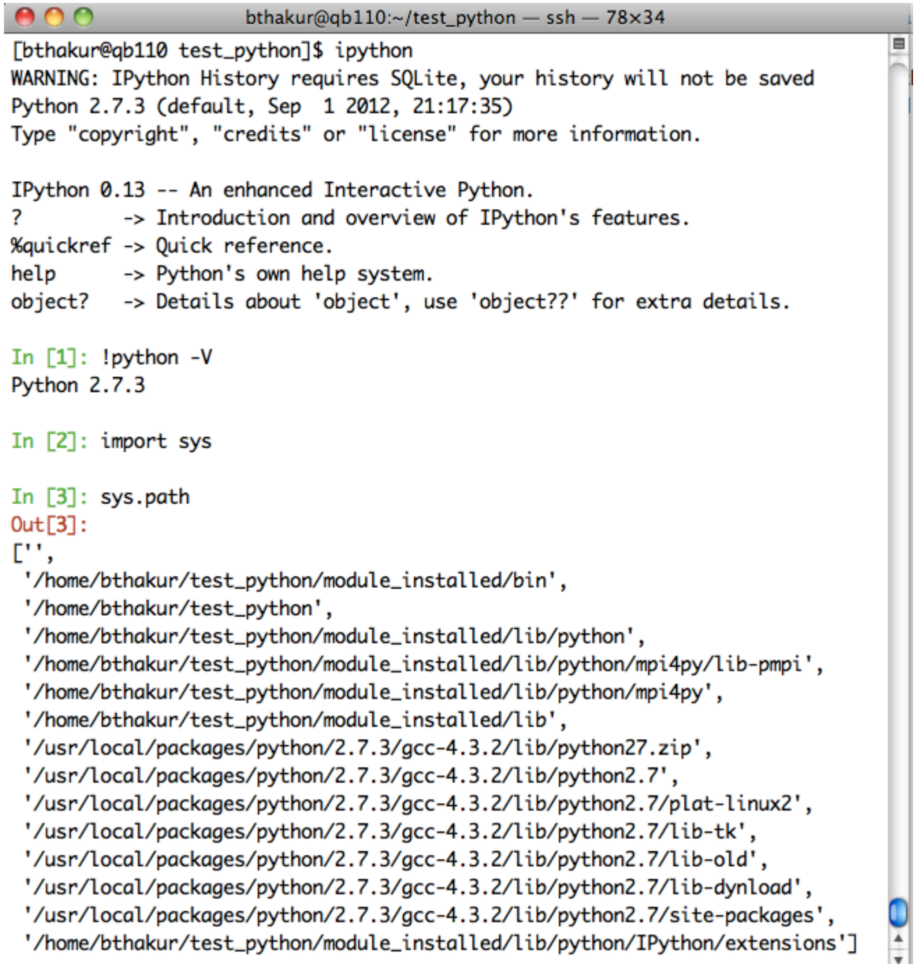
Exercise 1:

Exercise 2:

Using modules: Ipython

Ipython: A powerful Interactive interface to python

- System calls and tab completion
- GUI support
- Easy to use, high performance tools for parallel computing.



```
[bthakur@qb110 test_python]$ ipython
WARNING: IPython History requires SQLite, your history will not be saved
Python 2.7.3 (default, Sep  1 2012, 21:17:35)
Type "copyright", "credits" or "license" for more information.

IPython 0.13 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: !python -V
Python 2.7.3

In [2]: import sys

In [3]: sys.path
Out[3]:
['',
 '/home/bthakur/test_python/module_installed/bin',
 '/home/bthakur/test_python',
 '/home/bthakur/test_python/module_installed/lib/python',
 '/home/bthakur/test_python/module_installed/lib/python/mpi4py/lib-pmpi',
 '/home/bthakur/test_python/module_installed/lib/python/mpi4py',
 '/home/bthakur/test_python/module_installed/lib',
 '/usr/local/packages/python/2.7.3/gcc-4.3.2/lib/python27.zip',
 '/usr/local/packages/python/2.7.3/gcc-4.3.2/lib/python2.7',
 '/usr/local/packages/python/2.7.3/gcc-4.3.2/lib/python2.7/plat-linux2',
 '/usr/local/packages/python/2.7.3/gcc-4.3.2/lib/python2.7/lib-tk',
 '/usr/local/packages/python/2.7.3/gcc-4.3.2/lib/python2.7/lib-old',
 '/usr/local/packages/python/2.7.3/gcc-4.3.2/lib/python2.7/lib-dynload',
 '/usr/local/packages/python/2.7.3/gcc-4.3.2/lib/python2.7/site-packages',
 '/home/bthakur/test_python/module_installed/lib/python/IPython/extensions']
```

Using modules: Ipython

Ipython: A powerful Interactive interface to python

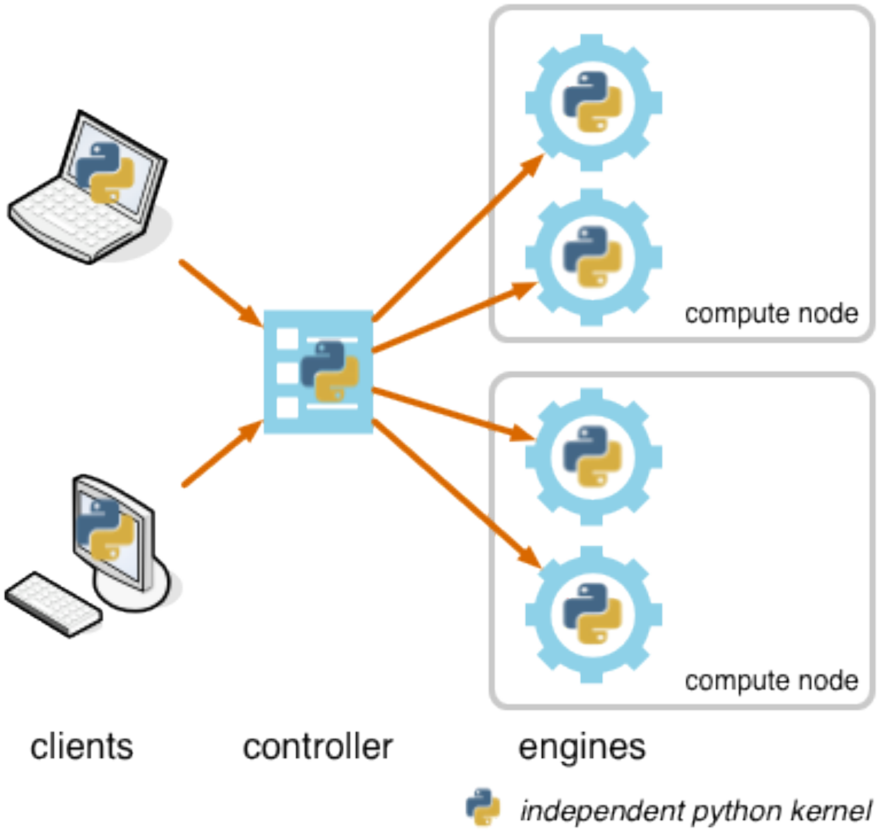
- Easy shell system calls and tab completion
- GUI support
- Easy to use hpc-tools for parallel computing.

```
>>> a = ! ls –al
>>> for I in a:
>>>     try:
>>>         j=i.split(); j[0], j[3], j[-1]
>>>     except:
>>>         print (“Error reading %s”, j[0])
```

Using modules: Ipython

Parallel programming using ipcluster: The three main components of IPCluster:

- **Controller:** the central process that manages all communications and dispatching of messages and data
- **Client:** the user process that connects to the Controller and issues commands or requests results
- **Engine:** a “slave” process that responds to commands sent by the Client (via the Controller)



Scipy Numpy

Testing Scipy

```
>>> import scipy
>>> scipy.test()
Running unit tests for scipy
NumPy version 1.6.2
...
Ran 5482 tests in 79.026s

OK (KNOWNFAIL=15, SKIP=41)
<nose.result.TextTestResult run=5482 errors=0 failures=0>
```

Numpy: Organization

```
[bthakur@qb3 ~]$ ls /usr/local/packages/python/2.7.3/gcc-4.3.2/lib/python2.7/site-packages/numpy/
```

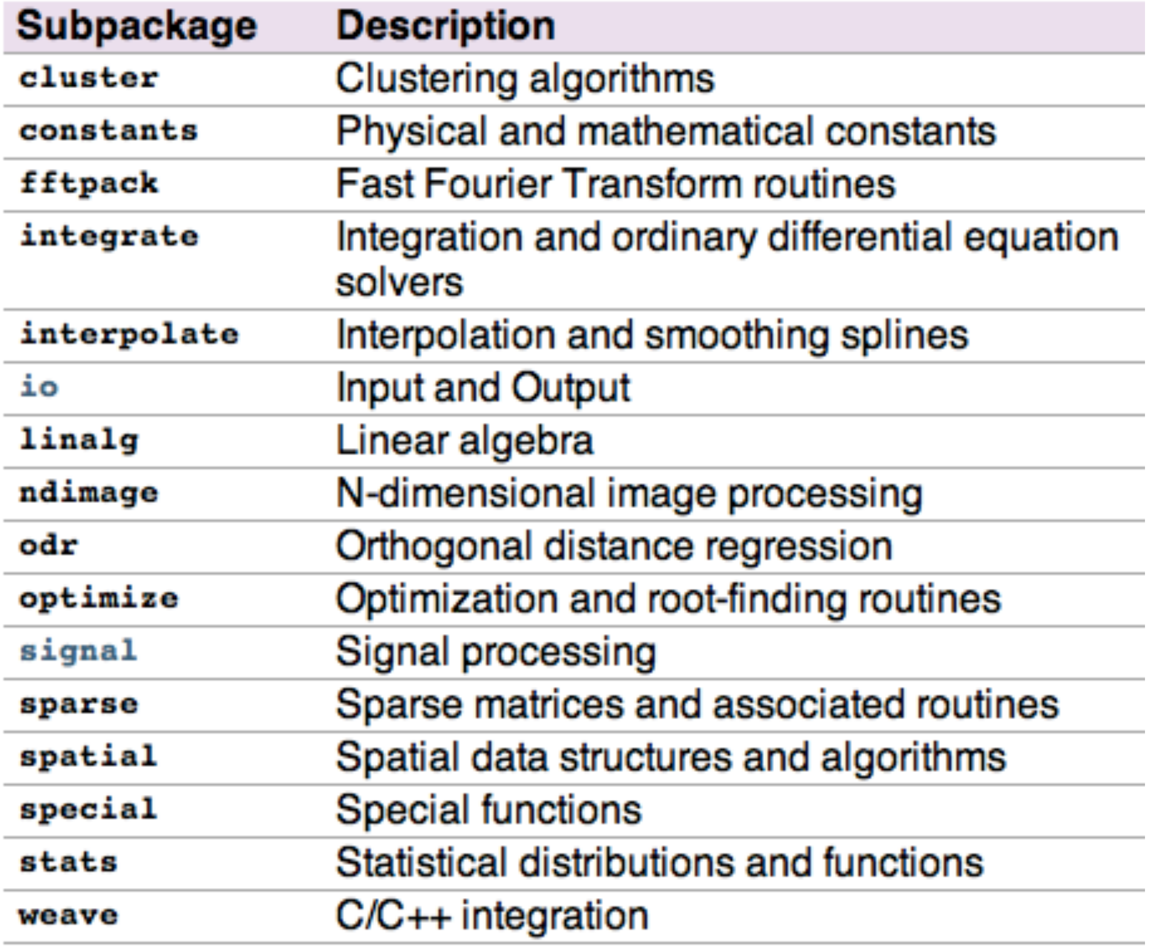
```
add_newdocs.py     fft                polynomial          __config__.pyc     __init__.pyc
ctypeslib.py       linalg             testing             dual.py            matrixlib
f2py               oldnumeric         __config__.py       __init__.py        setup.pyc
lib                setupscons.pyc     doc                 matlib.pyc         version.pyc
numarray           compat             _import_tools.pyc   setup.py
setupscons.py      distutils          matlib.py           version.py
add_newdocs.pyc    _import_tools.py   random              core
ctypeslib.pyc      ma                 tests               dual.pyc
```

scipy: Organization

```
[bthakur@qb3 ~]$ ls /usr/local/packages/python/2.7.3/gcc-4.3.2/lib/python2.7/site-packages/scipy/
```

```
BENTO_BUILD.txt   __init__.py         optimize          sparse          version.pyc
HACKING.rst.txt   LATEST.txt   odr    signal            version.py      fftpack
io                setupscons.pyc      TOCHANGE.txt      constants       interpolate
ndimage           THANKS.txt          __config__.pyc    integrate       misc
setupscons.py     __config__.py       INSTALL.txt       linalg          setup.pyc
stats             __init__.pyc        LICENSE.txt       setup.py        special
cluster           lib                 README.txt        spatial         weave
```

scipy: Organization



Subpackage	Description
`cluster`	Clustering algorithms
`constants`	Physical and mathematical constants
`fftpack`	Fast Fourier Transform routines
`integrate`	Integration and ordinary differential equation solvers
`interpolate`	Interpolation and smoothing splines
`io`	Input and Output
`linalg`	Linear algebra
`ndimage`	N-dimensional image processing
`odr`	Orthogonal distance regression
`optimize`	Optimization and root-finding routines
`signal`	Signal processing
`sparse`	Sparse matrices and associated routines
`spatial`	Spatial data structures and algorithms
`special`	Special functions
`stats`	Statistical distributions and functions
`weave`	C/C++ integration

Numpy: array class

Creating numpy array class

```
>>> import numpy as np


>>> x = np.array([2,3,1,0]); print x
array([2, 3, 1, 0])


>>>  np.zeros( (12), dtype=int )
>>>  np.arange(10)
>>>  np.linspace(1., 4., 6)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int8)
array([ 1. ,  1.6,  2.2,  2.8,  3.4,  4. ])
```

Type, size, trace, shape, dimensions

```
>>> x.dtype, x.ndim, x.shape, x.size
```

Indexing and slicing, Max, min, sum

```
x.max(), x.min(), x.sum()
```

Reshape, resize, inverse

```
reshape(), resize(), a.getI()
```

Relational operators

```
b= x>3;  c= x[(x>2) & (x<7)]
sum( (a%2) == 0 )
any( a == 10 )
```

Numpy: matrix class

Numpy matrix class

- Returns a matrix from an array-like object, or from a string of data.
- Specialized 2-D array that retains its 2-D nature through operations.
- Special operators, such as (*) multiplication and (**) power.

Create and view matrix

```
>>> import numpy as np
>>> a=np.matrix('[1 2 3; 4 5 6; 7 8 9]')
>>> a
matrix( [[1, 2, 3],
         [4, 5, 6],
         [7, 8, 9]])
```

Sum, trace, shape, dimensions

```
>>> np.sum(a), np.trace(a), np.ndim(a)
(45, 15, 2)
>>> np.shape(a)
(3, 3)
```

Diagonal, transpose

```
>>> np.diagonal(a); np.transpose(a)
array([1, 5, 9])
matrix([[1, 4, 7],
        [2, 5, 8],
        [3, 6, 9]])
```

Reshape, resize, inverse

reshape(), resize(), a.getI()

Scipy: Linear Algebra support

Scipy tutorial:
Solving a system of linear equations

$$\begin{aligned} x + 3y + 5z &= 10 \\ 2x + 5y + z &= 8 \\ 2x + 3y + 8z &= 3 \end{aligned}$$

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 8 \\ 3 \end{bmatrix}$$

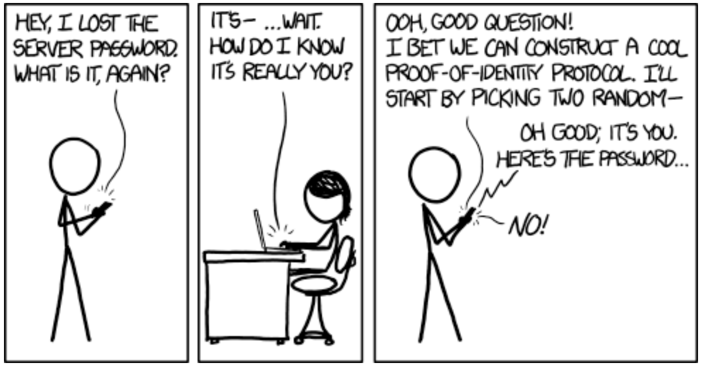
http://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html

```
>>> from numpy import *
>>> from scipy import linalg

>>> A = mat('[1 3 5; 2 5 1; 2 3 8]’)
>>> b = mat('[10;8;3]')
>>> linalg.det(A)
-25.000000000000007

>>> linalg.solve(A,b)
matrix([[-9.28],
        [ 5.16],
        [ 0.76]])
```

Using modules: SSH

SSH using Paramiko:

Setup ssh connection to remote machine and execute commands


HEY, I LOST THE SERVER PASSWORD. WHAT IS IT, AGAIN?
IT'S– ...WAIT. HOW DO I KNOW IT'S REALLY YOU?
OOH, GOOD QUESTION! I BET WE CAN CONSTRUCT A COOL PROOF-OF-IDENTITY PROTOCOL. I'LL START BY PICKING TWO RANDOM–
OH GOOD; IT'S YOU. HERE'S THE PASSWORD...
NO!

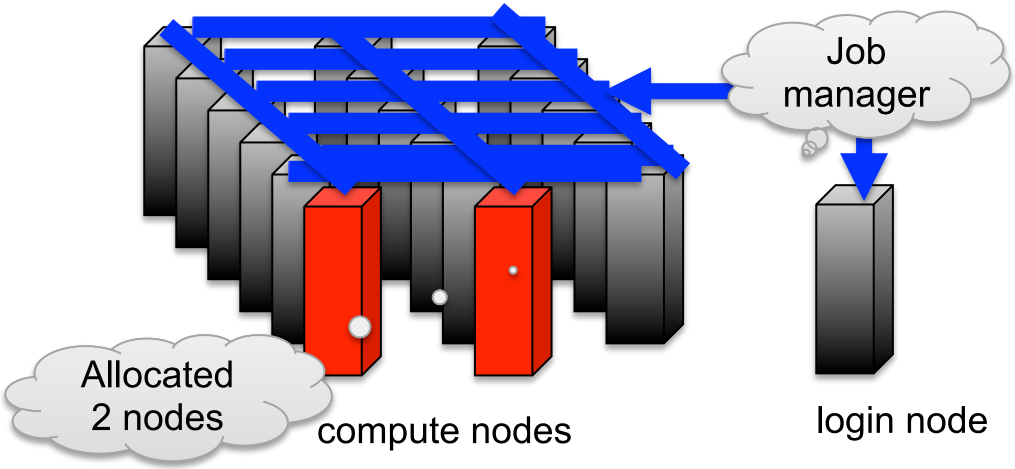
```
>>> import paramiko

>>> pol=    paramiko.AutoAddPolicy()
>>> user=   ‘myid’
>>> pass=   ‘mypass’
>>> server= ‘mymachine.lsu.edu’

>>> ssh= paramiko.SSHClient()
>>> ssh.set_missing_host_key_policy( pol)

>>> ssh.connect(server, username=user, password=pass)
>>>sin,sou,serr= ssh.exec_command(‘hostname’)
>>> ssh.close()
```

Using mpi: mpi4py

Mpi using mpi4py:

MPI library for running parallel jobs

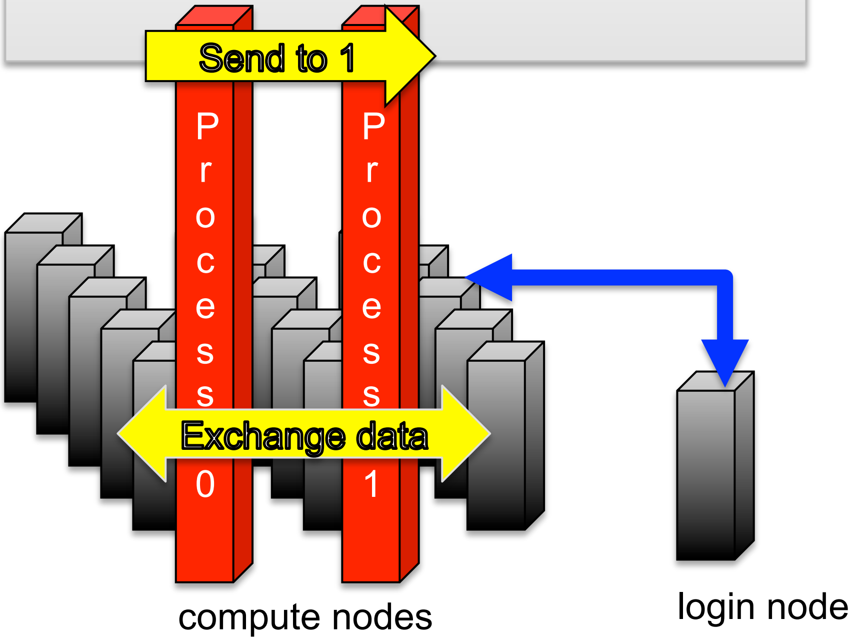
- Job manager allocates resources on compute nodes.
- Users usually submit jobs in batch or interactive mode from the login node.



Using mpi: mpi4py

Mpi using mpi4py:

MPI library for running parallel jobs



```
>>> from mpi4py import MPI

>>>  comm = MPI.COMM_WORLD
>>>  size = comm.Get_size()
>>>  rank = comm.Get_rank()

print ”My process id is ", rank, "of", size
```

```
My process id is  2 of 4
My process id is  0 of 4
My process id is  3 of 4
My process id is  1 of 4
```

Using mpi: mpi4py

Mpi using mpi4py:

MPI Broadcast example using dictionary

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()

# Initialize data
data = {'key1' : [0, 0.0, 0+0j],
        'key2' : ( '000', '000')}

# Change values on root node
if rank == 0:
   data = {'key1' : [7, 2.72, 2+3j],
           'key2' : ( 'abc', 'xyz')}

# data on two processes before broadcast
if rank == 0 or rank == 1:
   print "Before we receive", rank, data
```

```
# Broadcast from root
data = comm.bcast(data, root=0)

# Print data on two processes after broadcast
if rank == 0 or rank == 1:
   print "After we receive", rank, data
```

```
Before we receive 0
{'key2': ('abc', 'xyz'), 'key1': [7, 2.72, (2+3j)]}
Before we receive 1
{'key2': ('000', '000'), 'key1': [0, 0.0, 0j]}

After we receive 0
{'key2': ('abc', 'xyz'), 'key1': [7, 2.72, (2+3j)]}
After we receive 1
{'key2': ('abc', 'xyz'), 'key1': [7, 2.72, (2+3j)]}
```

Using sqlite: pysqlite

http://www.sqlite.org/

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain.

Using sqlite: pysqlite

sqlite using pysqlite:

```
>>> import sqlite3 as sq
>>> con=sq.connect('testpy.db')

>>> cur=con.cursor()
>>> cur.execute("""CREATE TABLE contacts(name text, phone integer)""")

>>> cur.execute("INSERT INTO contacts VALUES('HPC help', '2255780900')")
>>> cur.execute("INSERT INTO contacts VALUES(‘ITS help’,  '2255783375')")
>>> con.commit()
>>> con.close()
```

```
$ sqlite3 testpy.db
  sqlite> select * from contacts;
  HPC help|2255780900
  ITS help|2255783375
```

```
$ python sqlite_read.py
  (u'HPC help', 2255780900L)
  (u'ITS help', 2255783375L)
```

Using xml: xml.dom

Using xml.dom:

```
<result>

  <value>
       <name> John Doe </name>
       <age>  42 </age>
       <sex> M </sex>
  </value>

  <value>
     <name> Tom Doe </name>
     <age>  21 </age>
     <sex> M </sex>
 </value>

</result>
```

```
>>> from xml.dom.minidom import parse
>>> doc=parse("parse.xml”)

>>> values = doc.getElementsByTagName("value")

>>> for entry in values:
>>>       name=entry.childNodes[1]
>>>       for f in name.childNodes:
>>>             print f.data
```

```
 $ python parse.py
 John Doe
 Tom Doe
```

Broader than you think!

- Database: mysql-python, PyMySQL, PyGreSQL, pysqlite
- More parallel programming: Multiprocessing, Mpi4py, ipython
- Graphics/GPU programming: PyOpenGL, PyOpenCL, PyCUDA, Pygame
- Networking
- Regular expressions
- The list is endless ……..

Broader than you think!

- Database: mysql-python, PyMySQL, PyGreSQL, pysqlite
- More parallel programming: Multiprocessing, Mpi4py, Parallel Python
- Graphics/GPU programming: PyOpenGL, PyOpenCL, PyCUDA, Pygame
- Networking

Conclusion

- **Python is powerful and easy**: Understand its advantages(user friendly/object oriented) and disadvantages (impact on performance) for your project.
- **Module let you have useful tools and libraries at your disposal with minimal effort**
- **Interface to various languages, libraries and tools.**