LSU INFORMATION TECHNOLOGY SERVICES

LONI

Introduction to Python

Wei Feinstein
HPC User Services
LSU HPC & LONI
sys-help@loni.org
Louisiana State University
April, 2018

XSEDE
Extreme Science and Engineering Discovery Environment

Overview

- What is Python
- Python programming basics
- Control structures and functions
- Python modules

What is Python?

- A general-purpose programming language (1980) by Guido van Rossum
- Intuitive and minimal coding
- Dynamically typed, no type declarations, data type is tracked at runtime
- Automatic memory management
- Interpreted not compiled

Why Python?

Advantages

- Ease of programming
- Minimal time to develop and maintain codes
- Modular and object-oriented
- Large standard and user-contributed libraries
- Large user community

Disadvantages

- Interpreted and therefore slower than compiled languages
- Not great for 3D graphic applications requiring intensive computations

IPython

- Python: a general-purpose programming language (1980)
- IPython: an interactive command shell for Python (2001) by Fernando Perez
  - Enhanced Read-Eval-Print Loop (REPL) environment
  - Command tab-completion, color-highlighted error messages..
  - Basic Linux shell integration (cp, ls, rm…)
  - Great for plotting!

http://ipython.org

Jupyter Notebook

IPython Notebook was introduced in 2011

- Web interface to Python
- Rich text, improved graphical capabilities
- Integrate many existing web libraries for data visualization
- Allow to create and share documents that contain live code, equations, visualizations and explanatory text.

Jupyter Notebook (2015)

- Interface with over 40 languages, such as R, Julia and Scala

Python Installed on HPC/LONI clusters

soft add key

Machine	Version	Softenv Key
supermike2	2.7.3	+Python-2.7.3-gcc-4.4.6
supermike2	anaconda-3-4.0.0	+Python-3.5.1-anaconda-3.4.0

module load key

Machine	Version	Module
qb	2.7.10-anaconda	python/2.7.10-anaconda
qb	2.7.12-anaconda-tensorflow	python/2.7.12-anaconda-tensorflow
qb	2.7.7-anaconda	python/2.7.7-anaconda
qb	3.5.2-anaconda-tensorflow	python/3.5.2-anaconda-tensorflow
smic	2.7.10-mkl-mic	python/2.7.10-mkl-mic
smic	2.7.13-anaconda-tensorflow	python/2.7.13-anaconda-tensorflow
smic	2.7.7	python/2.7.7/GCC-4.9.0
smic	2.7.7-anaconda	python/2.7.7-anaconda
philip	2.7.10-anaconda	python/2.7.10-anaconda
philip	2.7.7	python/2.7.7/GCC-4.9.0

Notations

>>> IPython command shell

$ Linux command shell

Comments

=> Results from Python statements/programs

How to Use Python

Run commands directly within a Python interpreter

Example

```
$ python
>>> print('hello, Smith')
>>> hello, Smith
```

To run a program named 'hello.py' on the command line

Example

```
$ cat hello.py
  import sys
  def main():
      print('Hello, there', sys.argv[1])
  if __name__ == '__main__':
      main()
$ python hello.py Smith
```

Overview

- What is Python
- **Python programming basics**
- Control structures and functions
- Python modules

Python Programming Basic

- Variable
- Build-in data types
- File I/O

Variables

- Variable type is dynamically determined from the value it is assigned, no data type is declared
- Assign meaningful variable names
- Some keywords are reserved such as ‘print’, ‘assert’, ‘while’, ‘lambda’.
- A variable is assigned using the ‘=’ operator

https://docs.python.org/2.5/ref/keywords.html

Variable Types

Example

```
>>> x = 3.3
>>> type(x)
<type 'float'>

>>> y=int(x)
>>> type(y)
<type 'int'>

>>> my_file=open("syslog","r")
>>> type(my_file)
<type 'file'>
```

Operators

- Arithmetic operators +, -, *, /, // (integer division for floating point numbers), '**' power
- Boolean operators and, or and not
- Comparison operators >, <, >= (greater or equal), <= (less or equal), == equality

Build-in Data Types

- Number
- String
- List
- Tuple
- Dictionary
- File

Numbers

Example

```
>>> int(3.3)              => 3
>>> complex(3)            => (3+0j)
>>> float(3)              => 3.0
>>> sqrt(9)               => 3.0
>>> sin(30)               => -0.9880316240928618
>>> abs(-3.3)             => 3.3
>>> pi=3.14
>>> type(pi)              => type 'float'
>>> x=str(pi)
>>> type(x)               => type 'str'
```

Strings

Example

```
>>> my_str= “Hello World”
>>> len(my_str)   => 12
>>> my_str
>>> print(my_str)          => Hello World
>>> my_str[0]       #string indexing
#string slicing
>>>my_str[1:4], my_str[1:],my_str[:]
   => (‘ello’,‘ello World’,‘Hello World’
>>> my_str.upper()     =>”HELLO WORLD”
>>> my_str.find(‘world’) =>6   #return index
>>> my_str + ‘!!’      =>" Hello World!!”
>>> s1,s2=my_str.split()
>>> s1          => Hello
>>> s2          => World
>>> my_str.isalpha()              =>True
```

Multiple line spanning """

Example

```
>>> lines="""This is
...a multi-line block
...of text"""

>>> lines
   this is\na multi-line block\nof text'

>>>print(lines)
  This is
  a multi-line block
  of text
```

String % (printf-like)

Example

% take string on the left(%d int, %s string, %f/%g float point),
And the matching values in a tuple on the right

```
>>> text="%d pigs,I'll %s and %s" %(3,'huff','puff')
     => 3 pigs,I'll huff and puff

>>> text = "33.9 after formatting is %.3f" % 33.9
    =>33.9 after formatting is 33.900

>>> total="my share=%.2f tip=%d"%(24.5, 5.3)
>>> print(total)
    => 'my share=24.50 tip=5'
```

Lists

- Collection of data [ ]
- Often used to store homogeneous values
  e.g., Numbers, names with one data type
- Members are accessed as strings
- Mutable: modify in place without creating a new object

List

Example

```
>>> my_list=[1,2,9,4]
>>> my_list                                  => [1,2,9,4]
>>> my_list[0]              #indexing        => 1

#slicing [start:end]  [start to (end-1)]
>>> my_list[0:4] or my_list[:]               => [1,2,9,4]

>>> type(my_list)                            => <type, 'list'>
>>> my_list+my_list         #concatenate     => [1,2,9,4,1,2,9,4]
>>> my_list*2               #repeat          => [1,2,9,4,1,2,9,4]

>>> friends = ['john', 'pat', 'gary', 'michael']
>>> for index, name in enumerate(friends):
        print index, name
    =>  0 john
        1 pat
        2 gary
        3 michael
```

Lists

lists are mutable

Example

```
>>> my_list=[1,2,9,4]
>>> my_list.index(1)          => 0
>>> my_list.append(0)         => [1,2,9,4,0]
>>> my_list.insert(0,22)      => [22,1,2,9,4,0]
>>> del my_list[0]            => [1,2,9,4,0]
>>> my_list.remove(9)         => [1,2,4,0]
>>> my_list.sort()            => [0,1,2,4]
>>> my_list.reverse()         => [4,2,1,0]
>>> my_list[:]          =>list slicing
>>> len(my_list)              => 4
>>> my_list[0]=100
>>> print(my_list)            => [100,2,1,0]

>>> my_list=list(range(6)) => [0,1,2,3,4,5]
>>> for index in range(len(my_list)):
        my_list[index] += 10
>>> print my_list => [10,11,12,13,14,15]
```

List Sorting

Example

```
>>> my_list=[1,2,9,4]
>>> sorted(my_list)   #ascending order    =>1,2,4,9
>>> sorted(my_list, reverse=True)         =>9,4,2,1

strs = [‘ccc’, ‘AAAA’,  ‘d’, ‘BB’]
sorted(strs, key=len)       =>[‘d’,’BB’,’ccc’,’AAAA’]
sorted(strs, key=str.lower, reverse=True)

                        =>[’d’,’ccc’,’BB’,’AAAA’]

Sort by key function
def my_fun(s):
   return s[-1]

my_strs=[‘xc’,’zb’,’yd’,’wa’]
sorted(my_strs, key=my_fun)   =>['wa’,'zb’,'xc’,'yd']
```

List Comprehensions

Example

```
[expr for var in list]

>>> nums =[1,2,9,4]
>>> square = [ n*n for n in nums]
                              =>[1,4,81,16]
>>> small=[n for n in nums if n<=2]
                              =>[1,2]

>>> strs = ['hello', 'and', 'goodbye']
>>> shouting=[s.upper()+'!!' for n in strs]
             =>['HELLO!!', 'AND!!', 'GOODBYE!!']
>>> sub=[s.upper()+'!!' for s in strs if 'o' in s]
             =>['HELLO!!', 'GOODBYE!!']
```

Tuples

- Collection of data ()
- Not immutable
- Why Tuple?
  - Processed faster than lists
  - Sequence of a Tuple is protected
- Sequence unpacking

Tuples

Example

```
>>> my_tuple=(1,2,9,4)
>>> print(my_tuple)          => (1,2,9,4)
>>> print(my_tuple[0])       => 1
>>> my_tuple[0] = 10
```

TypeError: 'tuple' object does not support item assignment

```
>>> x,y,z,t = my_tuple       #Unpacking
>>> print(x)            => 1
>>> print(y)            => 2

Switching btw list and tuple
>>> my_l=[1,2]  >>> type(my_l)    => <type 'list'>
>>> my_t=tuple(my_l)  >>> type(my_t)  =><type 'tuple'>
```

Dictionary

- List of key-value pairs { }
- Unordered collections of objects, not indexed
- Store objects in a random order to provide faster lookup
- Data type are heterogeneous, unlike list
- Element are accessed by a keyword, not index
- Elements are mutable

Dictionaries

dictionary = {"key1": value1, "key2": value2}

Example

```
>>> my_dict={"new":1,"old":2}
>>> my_dict['new']    #indexing by keys   => 1
>>> my_dict.has_key('new')          => True
>>> my_dict['new'] = 9
>>> my_dict['new']          => 9
>>> del my_dict['new']
>>> my_dict             => {'old': 2}
>>> my_dict["young"] = 4   #add new entry
             => {"old":2,"young":4}
```

Dictionaries

dictionary = {"key1": value1, "key2": value2}

Example

```
>>> my_table={"python":'red',"linux":'blue'}
>>> my_table.keys()          =>['python','linux']
>>> my_table.values()        =>['red','blue']

>>> my_table.items()
              =>[('python','red'),('linux','blue')]

>>> for key in my_table.keys():
        print(key, my_table[key])
              =>('python','red')
                ('linux','blue')

>>> for key, value in my_table.items():
         print key, value   =>python red
                              linux blue
```

File Operations

- file_handle = open("file_name", 'mode')
- Modes:
  - a:append
  - r: read only (error if not existing)
  - w: write only
  - r+: read/write (error if not existing)
  - w+: read/write
  - b: binary

File Operations

Example

```
>>>  input =open("data",'r')
>>> content=input.read()
>>> line=input.readline()
>>> lines=input.readlines()
>>> input.close()
>>>  output =open("result",'w')
>>> output.write(content)
>>> output.close()
```

- Python has a built-in garbage collector
- Object memory space is auto reclaimed once a file is no longer in use

Overview

- What is Python
- Python programming basics
- **Control structures and functions**
- Python modules

Control Structures

- if-else
- while loops, for loops
- break: jump out of the current loop
- continue: jump to the top of next cycle within the loop
- pass: do nothing

Indentation

- Indentation: signify code blocks (very important)

Example (loop.py)

```
n=2
while n < 10:
    prime = True
    for x in range(2,n):
        if n % x == 0:
            prime = False
            break
    if prime:
        print n, 'is a prime #'
        pass
    else:
        n=n+1
        continue
    n=n+1
```

```
$ python loop.py
2 is a prime #
3 is a prime #
5 is a prime #
7 is a prime #
```

Define a Function

```
def func_name(param1,param2, ..):
    body
```

Example

```
>>> def my_func(a,b):
        return a*b

>>> x, y = (2,3)
>>> my_func(x,y)          => 6

>>> def greet(name):
      print 'Hello', name

>>> greet('Jack')         => Hello Jack
>>> greet('Jill')         => Hello Jill
```

Return multiple values

Example

```
>>> def power(number):
        return number**2, number**3
>>> squared, cubed = power(3)

>>> print(squared)          => 9
>>> print(cubed)            => 27
```

Exceptions

- Events to alter program flow either intentionally or due to errors, such as:
  - open a non-existing file
  - zero division
- Catch the faults to allow the program to continue

Without Exception Handling

exception_absent.py

Example

```
f_num = raw_input("Enter the 1st number:" )
s_num = raw_input("Enter the 2nd number:")
num1,num2 = float(f_num), float(s_num)
result = num1/num2
print str(num1) + "/" + str(num2) + "=" + str(result)
```

```
$ python exception_absent.py
Enter the 1st number:3
Enter the 2nd number:0
Traceback (most recent call last):
  File "exception_absent.py", line 4, in <module>
    result = num1/num2
ZeroDivisionError: float division by zero
```

Exception Handling

exception.py

Example

```
f_num = raw_input("Enter the 1st number:" )
s_num = raw_input("Enter the 2nd number:")
try:
    num1,num2 = float(f_num), float(s_num)
    result = num1/num2
except ValueError:          #not enough numbers entered
    print "Two numbers are required."
except ZeroDivisionError:     #divide by 0
    print "Zero can't be a denominator . "
else:
    print str(num1) + "/" + str(num2) + "=" + str(result)
```

```
$ python exception.py
Enter the 1st number:3
Enter the 2nd number:0
Zero can't be a denominator.
```

```
$ python exception.py
Enter the 1st number:3
Enter the 2nd number:
Two numbers are required.
```

Sample Program (game.py)

```
import random
guesses_made = 0
name = raw_input('Hello! What is your name?\n')
number = random.randint(1, 20)
print 'Well, {0}, I am thinking of a number between 1 and 20.'.format(name)

while guesses_made < 6:
    guess = int(raw_input('Take a guess: '))
    guesses_made += 1
    if guess < number:
        print 'Your guess is too low.'
    if guess > number:
        print 'Your guess is too high.'
    if guess == number:
        break
if guess == number:
    print 'Good job, {0}! You guessed my number in {1} guesses!'.format(name,
    guesses_made)
else:
    print 'Nope. The number I was thinking of was {0}'.format(number)
```

Overview

- What is Python
- Python programming basics
- Control structures and functions
- **Python modules**

Python Modules

- Module: a Python script with Python functions and statements
- Import the module
- Now you have access to functions and variables in that module

Python Modules

Most Python distributions come with plenty of build-in modules

- math, sys, os…
- NumPy: high performance in vector & matrix(vector computation)
- SciPy: base on NumPy, include many scientific algorithms
- pandas
- Matplotlib, Pyplot, Pylab
- .....

Reference to Python 2.x standard library of modules at
http://docs.python.org/2/library/

Module Namespaces

- Namespaces: containers for mapping names to objects, a system to make sure all names are unique
- foo() defined in module1 and module2. Fully qualified names to access foo():
  - module1.foo
  - module2.foo

Get Info on Modules (dir)

Example

```
>>> import math                 #import the whole module
>>> math.sin(math.pi)           => 1.2246467991473532e-16

>>> from math import *          #import all symbols from math
>>> sin(pi)                     => 1.2246467991473532e-16

>>>dir(math)   #all the symbols from math module
['__doc__', '__file__', '__name__', '__package__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil',
'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc',
'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'gamma', 'hypot', 'isinf', 'isnan',
'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'modf', 'pi',
'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',
'trunc']
```

Get Info on Modules (help)

Example

```
>>> help(math)
```

Help on module math:

NAME

math

…

DESCRIPTION

This module is always available. It provides access to the mathematical functions defined by the C standard.

FUNCTIONS

…

```
>>> help(math.sin)
Help on built-in function sin in module math:
sin(...)
    sin(x)
    Return the sine of x (measured in radians).
```

Utility Modules (os, sys)

os: includes many functions to interact with the file system
sys: System-specific parameters and functions.

util.py

```
import sys, os

def printdir(dir):
  filenames = os.listdir(dir)
  for filename in filenames:
    print filename
    print os.path.abspath(os.path.join(dir, filename))

def main():
    if len(sys.argv) < 2:
        printdir(os.getcwd())
    else:
        printdir(sys.argv[1])

if __name__ == '__main__':
    main()
```

```
$ pwd
 /Users/wei/intro_python
$ python util.py
exception.py
/Users/wei/intro_python/exception.py
exception_absent.py
/Users/wei/intro_python/
…
```

```
$ python util.py ~
CIPRES
/Users/wei/CIPRES
composer.phar
/Users/wei/composer.phar
….
```

Running External Processes Module (commands)

cmd.py

```
import commands, sys, os

def listdir(dir):
  cmd = 'ls -l ' + dir
  print('command to run:', cmd)
  status, output = commands.getstatusoutput(cmd)
  if status:  ## Error case, print output to stderr and exit
     sys.stderr.write(output)
     sys.exit(status)
  print(output)

def main():
    if len(sys.argv) < 2:
     listdir(os.getcwd())
    else:
        listdir(sys.argv[1])

if __name__ == '__main__':
    main()
```

```
$ python cmd.py
('command to run:', 'ls -l /Users/wei/
intro_python')
total 88
-rw-r--r--  1 wei  staff  369 Apr 10 21:53
cmd.py
-rw-r--r--  1 wei  staff  381 Mar 14  2017
exception.py ….
```

```
$ python cmd.py ~
('command to run:', 'ls -l /Users/wei')
total 556904
-rw-r--r--   1 wei  staff     8834 Jun  7  2017
1_notmnist.ipynb
-rw-r--r--   1 wei  staff    39083 Oct 24 22:27
1st book.ipynb ….
```

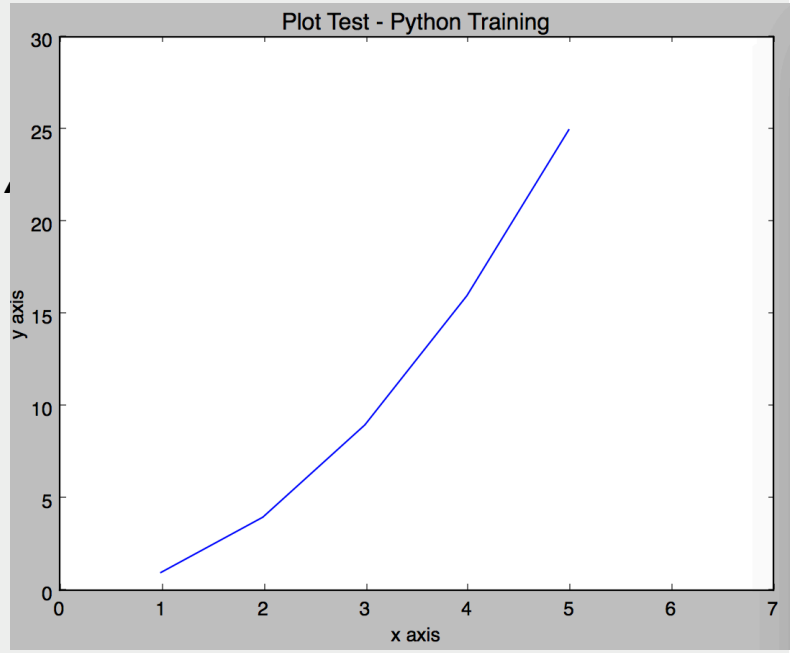
Plotting

- Matplotlib: Python library for plotting
- Pyplot: a wrapper module to provide a Matlab-style interface to Matplotlib
- Pylab: NumPy+Pyplot

Example (plot.py)

```
import numpy as np
import pylab as pl
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
# use pylab to plot x and y
pl.plot(x, y)
# give plot a title
pl.title('Plot Test Python Training')
# make axis labels
pl.xlabel('x axis')
pl.ylabel('y axis')
# set axis limits
pl.xlim(0.0, 7.0)
pl.ylim(0.0, 30.)
# show the plot on the screen
pl.show()

$ python plot.py
```



Conclusions

- Python is an interpreted language, concise yet powerful
- Built-in data types
- Indentation is used to mark code blocks in control structures, functions and classes
- Object-Oriented Programming language, with classes as building blocks
- Rich repository of Python libraries and modules
- Create your own modules and classes
- Rich plotting features

Upcoming Training

- April 18,2018: Introduction to Deep Learning and Software

http://www.hpc.lsu.edu/training/tutorials.php#upcoming