Introduction to Python

Wei Feinstein
HPC User Services
LSU HPC & LONI
sys-help@loni.org
Louisiana State University
March, 2017

Overview

- What is Python
- Python programming basics
- Control structures, functions
- Python modules, classes
- Plotting with Python

What is Python?

- A general-purpose programming language (1980) by Guido van Rossum
- Intuitive and minimalistic coding
- Dynamically typed
- Automatic memory management
- Interpreted not compiled

Why Python?

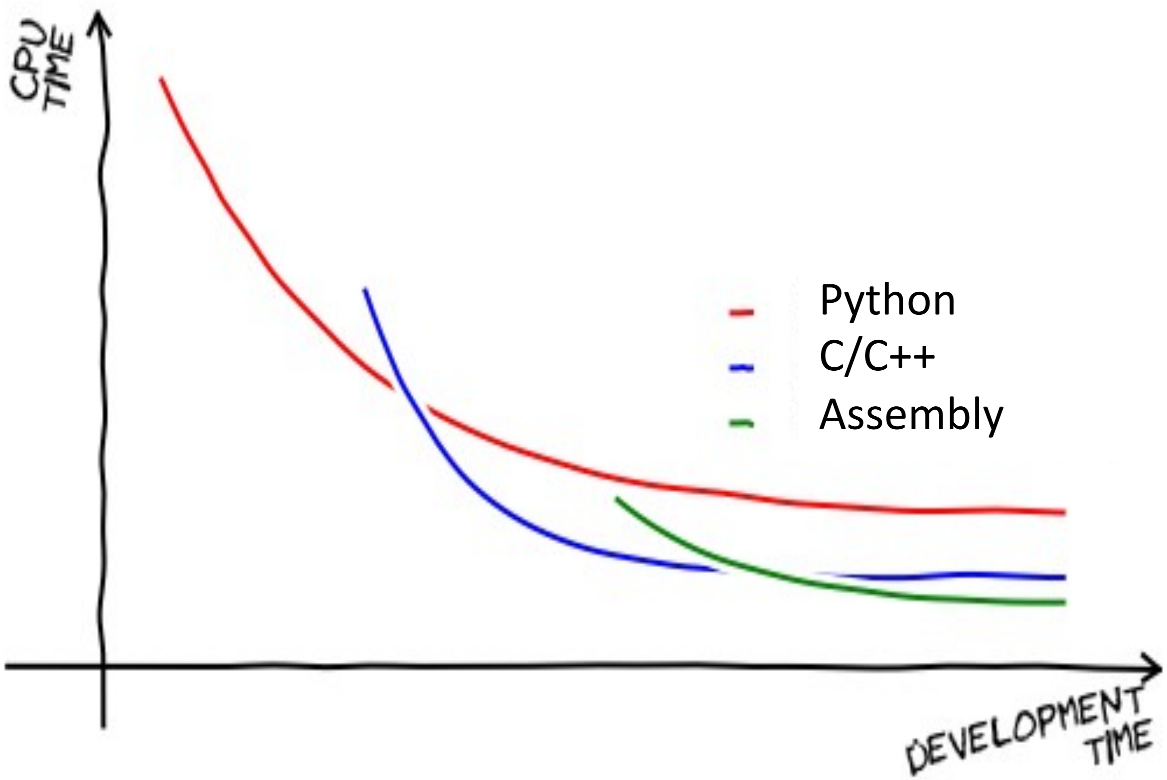
Advantages

- Ease of programming
- Minimizes the time to develop and maintain code
- Modular and object-oriented
- Large standard and user-contributed libraries
- Large community of users

Disadvantages

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

Code Performance vs. Development Time



Python 2.x vs 3.x

- Final Python 2.x is 2.7 (2010)
- First Python 3.x is 3.0 (2008)
- Major cleanup to better support Unicode data formats in Python 3.x
- Python 3 not backward-compatible with Python 2
- Rich packages available for Python 2z

```
$ python -- version
```

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 introduced a new tool Notebook (2011)

- Bring modern and powerful 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

LSU INFORMATION TECHNOLOGY SERVICES

Python packages on LSU/LONI clusters

soft add xxx

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 xxx

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	2.7.7	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

XSEDE Extreme Science and Engineering Discovery Environment

How to Use Python

Run commands directly within a Python interpreter

Example

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

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

Example

```
$ cat hello.py
   print('hello')

$ python hello.py
```

Notations

>>> IPython command shell

$ Linux command shell

Comments

=> Results from Python statements or programs

Python Programming

- Syntax
- Variables
- Strings
- Lists
- Tuples
- Dictionaries
- File I/O

Variables

- Variable names can contain alphanumerical characters and some special characters
- It is common to have variable names start with a lower-case letter and class names start with a capital letter
- Some keywords are reserved such as 'and', 'assert', 'break', 'lambda'.
- A variable is assigned using the '=' operator
- Variable type is dynamically determined from the value it is assigned.

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’>
```

LSU INFORMATION TECHNOLOGY SERVICES

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

XSEDE Extreme Science and Engineering Discovery Environment

Data types

Build-in data types in Python

- Numbers
- Strings
- Lists
- Dictionary
- Tuples
- Files

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
```

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[:-1]
   => ('ello', 'ello World', 'Hello Worl'
>>> 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
```

Multiple line spanning """

Example

```
>>> lines="""This is
...a multi-line block
...of text"""
>>> lines
   this is\na multi-line block\nof text'
```

LSU INFORMATION TECHNOLOGY SERVICES

Print Strings

Example

```
>>> print("hello"+" World")
     => Hello World
>>> print("33.9 after formatting = %.3f" %33.9)
    =>33.9 after formatting = 33.900

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

>>> total="my share={:.2f} tip={:.0f}".format(24.5, 5.3)
    =>  'my share=24.50 tip=5'
```

XSEDE Extreme Science and Engineering Discovery Environment

Lists

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

Lists

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.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]
>>> len(my_list)            => 4

>>> my_list[0]=100
>>> print(my_list)          => [100,2,1,0]
```

Tuples

- Collection of data ()
- Not immutable
- Why Tuples?
  - Tuples are processed faster than lists
  - Sequences of a Tuple are 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’>
```

Dictionaries

- Lists 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
- Accessed by 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_ditc[‘new’]                    => 9
>>> del my_dict[‘new’]
>>> my_dict                           => {'old': 2}
>>> my_dict[“young”] = 4   #add new entry
             => {“old”:2,”young”:4}

>>> table={“python”:’red’, “linux”: ‘blue’}
>>> for key in table.keys():
        print(key, table[key])  =>(‘python’,’red’)
                                  (‘linux’,‘blue’)
```

Files

- 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

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 used to 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 #
```

Exceptions

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

Without Exceptions 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
```

Exceptions 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.
```

Define a Function

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

Function arguments

Call-by-value: integers, strings, tuples (no change)

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
```

Function arguments

- Call-by-reference: pass mutable arguments
- Arguments can be changed but can't be re-assigned to new object

example

```
>>> def Fruits(lists):
        lists.insert(0, 'new')   #modify list
>>> my_list=['apple','pear']
>>> Fruits(my_list)
>>> my_list                    => ['new','apple','pear']

>>> def Grains(lists):
        lists = ['new']        #reassign list
        print lists
>>> my_list=['rice','wheat']
>>> Grains(my_list)            => ['new']
>>> my_list                    => ['apple','pear']
```

LSU INFORMATION TECHNOLOGY SERVICES

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)
```

XSEDE Extreme Science and Engineering Discovery Environment

Object Oriented Programming (OOP)

- Python is a OOP language, like C++
- Object: collection of data and methods
- Class is prototype of describing an object
- Why use classes?
  - Define an object once, reuse it multiple times
  - Inheritance: a new class born from an old class
  - Method or operator overloads: redefine functions/ methods in the newly defined class

Classes

- You have seen/used classes
- Build-in python classes: strings, lists, tuples, dictionaries, files …
  - file class: input.readline(), output.write()
  - list class: list.append()
  - string class: string.replace(), string.len()

A Python class

Example (point.py)

```
class Point:
        '''This is my Point class'''
        def __init__(self, x, y):
                self.x = x
                self.y = y
        def translate(self, dx, dy):
                self.x += dx
                self.y += dy
        def display(self):
                return "point at [%s, %s]" % (self.x, self.y)

origin = Point(0,0)  # create a Point object
print("Orignal", origin.display())
origin.translate(1,2)
print('after translate', origin.display())
```

```
$ python point.py
('Orignal', 'point at [0, 0]')
('after translate', 'point at [1, 2]')
```

Python Classes

Class inheritance

Example

```
>>> class Shape:
        def area(self):
                print “now in Shape.area
    class Ball(Shape):
        def area(self):
                Shape.area(self)
                print “ending in Ball.area”

>>> s=Shape()
>>> s.area()        =>now in Shape.area
>>> b.Ball()
>>> b.area()        =>now in Shape.area
                      ending in Ball.area
```

Python Modules

- Module: a Python script (xxx.py) with Python definitions and statements
- Import the module
- Now you have access to functions and variables in that module

Python Modules

Most Python installations 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/

How to Use Modules

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

>>>print(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 Module Info

Example

```
>>> help(math.sin)

Help on built-in function sin in module math:

sin(...)
    sin(x)

    Return the sine of x (measured in radians).
```

Create Python Modules

Example (my_module.py)

```
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b
def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
#make sure the path of the new
>>> import sys
>>> sys.path
['/Users/wei/python',
 '',
 '/Users/wei/anaconda/bin',
 '/Users/wei/anaconda/lib/python27.zip',
 '/Users/wei/anaconda/lib/python2.7',....
>>> sys.path.append('/Users/wei/intro_python')
```

```
>>> import my_module
>>> my_module.fib(60)
1 1 2 3 5 8 13 21 34 55
>>> my_module.fib2(60)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
>>> my_module.__name__
'my_module'
```

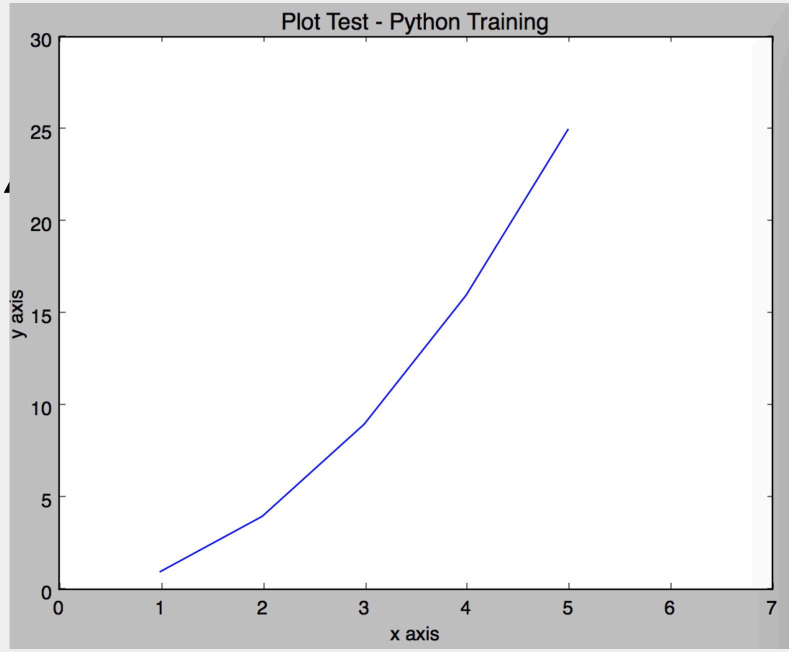
Plotting

- Matplotlib: Python library for plotting
- Pyplot: a wrapper module providing 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
- Python has rich sets of built-in data types
- Indentation is used to mark code blocks in control structures, functions and classes
- Python is an object-oriented programming language, with classes as building blocks
- Rich repository of Python libraries, modules
- Create your own modules and classes
- Rich plotting features

Upcoming Trainings

1. March 22, 2017: Parallel computing with R
2. March 28, 2017: Agave: Ordering a Science Gateway To Go
3. March 29, 2017: Intermediate Python Programming
4. April 5, 2017: Machine Learning in HPC Environments

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