Python Language Basics I

Georgia Advanced Computing Resource Center (GACRC)

Enterprise Information Technology Services(EITS)

The University of Georgia

Outline

- GACRC
- Python World
- General Lexical Conventions
- Basic Built-in Data Types

GACRC

- A high-performance-computing (HPC) center at the UGA
- Provide to the UGA research and education community an advanced computing environment:
  - HPC computing and networking infrastructure located at the Boyd Data Center
  - Comprehensive collection of scientific, engineering and business applications
  - Consulting and training services

Wiki: http://wiki.gacrc.uga.edu

Support: https://uga.teamdynamix.com/TDClient/Requests/ServiceCatalog?CategoryID=11593

Web Site: http://gacrc.uga.edu

Python World

- What is Python
- Scientific Python Modules
- Scientific Python Distributions
- Run Python Interactively on Sapelo2

What is Python

- Open source general-purpose scripting language (https://www.python.org/)
- Working with *procedural*, *object-oriented*, and *functional* programming
- Glue language with Interfaces to other languages, like C/C++ (via SWIG), Object-C (via PyObjC), Java (Jython), and Fortran (via F2PY) , etc. (https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages)
- Last Python2 version is 2.7.16; Latest Python3 version is 3.8.2; Current Python3 version on Sapelo2 is 3.7.4

Scientific Python Modules

- Python has a large collection of built-in modules included in standard distributions, e.g., io, os, sys, datetime, argparse, etc.:

  https://docs.python.org/3/index.html

  https://docs.python.org/3/library/index.html

- Packages for scientific modules:

  - NumPy
  - Biopython
  - SciPy
  - TensorFlow
  - Matplotlib
  - PyTorch

Scientific Python Modules

- NumPy: Matlab-ish capabilities, fast N-D array operations, linear algebra, etc. (http://www.numpy.org/)
- SciPy: Fundamental library for scientific computing (http://www.scipy.org/)
- matplotlib: High quality plotting (http://matplotlib.org/)
- TensorFlow: Open source platform for machine learning (https://www.tensorflow.org/)
- PyTorch: Open source machine learning library (https://pytorch.org/)

Scientific Python Distributions

- Anaconda
  - Comes with 1,500 packages selected from PyPI as well as the conda package and virtual environment manager
  - Supports Linux, Mac and Windows (https://www.anaconda.com/)
- Python(x,y)
  - A scientific-oriented Python Distribution based on Qt and Spyder
  - Windows only (https://python-xy.github.io/)
- WinPython
  - A free open-source portable distribution of the Python
  - Windows only (https://github.com/winpython)

Anaconda with Spyder IDE on my local computer:

Spyder (Python 3.5)

File Edit Search Source Run Debug Consoles Tools View Help

/home/MosesHou

Editor - /home/MosesHou/python scripts/numpy.py

numpy.py

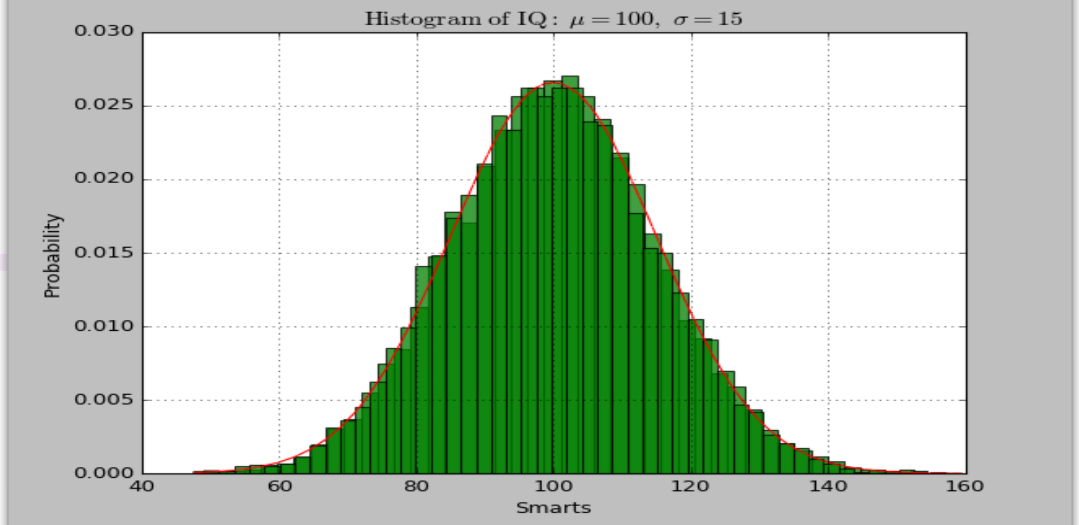
```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 14 10:56:30 2016

@author: MosesHou
"""

#!/usr/bin/env python
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

mu, sigma = 100, 15

x = mu + sigma*np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='green', alpha=0.75)

# add a 'best fit' line
y = mlab.normpdf( bins, mu, sigma)
l = plt.plot(bins, y, 'r--', linewidth=1)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title(r'$\mathrm{Histogram\ of\ IQ:}\ \mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)

plt.show()
```

Console

Python 1

```
Python 3.5.1 |Anaconda 2.5.0 (64-bit)| (default, Dec  7 2015, 11:16:01)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> runfile('/home/MosesHou/python scripts/numpy.py', wdir='/home/MosesHou/python scripts')
>>> runfile('/home/MosesHou/python scripts/numpy.py', wdir='/home/MosesHou/python scripts')
>>>
```

Figure 1



Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 30 Column: 1 Memory: 8 %

Run Python Interactively on Sapelo2

- Run python interactively on interactive node (use qlogin from login node)

```
zhuofei@sapelo2-sub2 ~$ qlogin
qsub: waiting for job 2367783.sapelo2 to start
qsub: job 2367783.sapelo2 ready

zhuofei@n204 ~$ module load Python/3.7.4-GCCcore-8.3.0
zhuofei@n204 ~$ python
Python 3.7.4 (default, Jan 30 2020, 18:11:14)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information. >>>  a = 7
>>>  e = 2
>>>  a**e
49
>>>
```

Run Python Interactively on Sapelo2

- script.py:

```
print("Hello, World!")
a = 7
e = 2
print(a**e)
```

- Run a Python script on interactive node (use qlogin from login node):

```
zhuofei@sapelo2-sub2 ~$ qlogin
qsub: waiting for job 2367783.sapelo2 to start
qsub: job 2367783.sapelo2 ready

zhuofei@n204 ~$ module load Python/3.7.4-GCCcore-8.3.0
zhuofei@n204 ~$ python  script.py
Hello, World!
49
```

General Lexical Conventions

- A Python code clip:

```
x = 10; y = "Hello!”                    # this is a comment
z = 3.14                                # z is a floating number

if z == 3.14 or y == "Hello!":
   x = x + 1
   y = y + " Python!"

print x
print y
```

- Semicolon ; to separate statements on the same line
- Hash # denotes a comment
- Assignment uses = ; comparison uses ==
- Logical operators are words: and, or, not
- Consistent indention within a block (4 spaces)
- For numbers: + - * / % are as expected

  For strings: + means concatenation
- The basic printing statement: print

Basic Built-in Data Types

- "Python is a dynamically typed language where variable names are bound to different values, possibly of varying types, during program execution. Variables names are untyped and can be made to refer to any type of data."

—*Python Essential Reference, 4th ed.*

```
a = 10          # a is created to refer to an integer
a = 3.24        # a is referring to a floating-point number now
a = "Hello!"    # a is referring to a string now
a = True        # a is referring to a boolean (True/False) now
```

Basic Built-in Data Types

Type Category	Type Name	Description
Numbers	int	i = 10; integer
	long	l = 73573247851; arbitrary-precision integer (Python 2 only!)
	float	f = 3.14; floating point
	complex	c = 3 + 2j; complex
	bool	b = True; Boolean (True or False)
Sequences	str	s = “Hello! Python”; character string
	list	lst = [1, 2, ”abc”, 2.0]; list of any typed elements (mutable!)
	tuple	t = (1, 2, “abc”, 2.0); record of any typed elements (immutable!)
Mapping	dict	d = {1:”apple”, 2:””}; mapping dictionary of any typed pairs of key:value

Basic Built-in Data Types

- **List**: A mutable sequence of arbitrary objects of any type

list1 = [1, “David”, 3.14, “Mark”, “Ann”]

index : 0 1 2 3 4 ➔ $Index_{max} = Length - 1$

- Indexed by integer, starting with zero:

```
a = list1[1]          # returns the 2nd item “David” ; a = “David”
list1[0] = “John”     # changes the 1st item 1 to “John” ; list1 = [“John”, “David”, 3.14, “Mark”, “Ann”]
```

- Empty list is created by:

```
list2 = []            # an empty list
list2 = list()        # an empty list
```

- Append and insert new items to a list:

```
list1.append(7)       # appends a new item to the end ; list1 = [“John”, “David”, 3.14, “Mark”, “Ann”, 7]
list1. insert(2, 0)   # inserts a new item into a middle ; list1 = [“John”, “David”, 0, 3.14, “Mark”, “Ann”, 7]
```

Basic Built-in Data Types

- Extract and reassign a portion of a list by slicing operator [i, j], with an index range of i<=k<j:

```
a = list1[0:2]          # returns [“John”, “David”] ; the 3rd item 0 is NOT extracted!
b = list1[2:]           # returns [0, 3.14, “Mark”, “Ann”, 7]
list1[0:2] = [-3, -2, -1]   # replaces the first two items with the list on the right
                        # list1 = [-3, -2, -1, 0, 3.14, “Mark”, “Ann”, 7]
```

- Delete items:

```
del list1[0]            # deletes the 1st item ; list1 = [-2, -1, 0, 3.14, “Mark”, “Ann”, 7]
del list1[0:4]          # delete a slice of the first 4 items ; list1 = [“Mark”, “Ann”, 7]
```

- Concatenate and multiply lists:

```
list2 = [8, 9]          # creates a new list
list3 = list1 + list2   # list3 = [“Mark”, “Ann”, 7, 8, 9]
list4 = list1 * 3       # list4 = [“Mark”, “Ann”, 7, “Mark”, “Ann”, 7, “Mark”, “Ann”, 7]
```

Basic Built-in Data Types

- Count occurrences of items:

```
list4.count("Mark")          # returns 3
```

- Remove an item from a list:

```
list1.remove("Ann")          # Search for "Ann" and remove it from list1 ; list1 = ["Mark", 7]
```

- Sort a list in place:

```
list5 = [10, 34, 7, 8, 9]    # creates a new list
list5.sort()                 # list5 = [7, 8, 9, 10, 34]
```

- Reverse a list in place:

```
list5.reverse()              # list5 = [34, 10, 9, 8, 7]
```

- Copy a list (*shallow copy*):

```
list6 = list(list5)          # list6 is a shallow copy of list5
```

Basic Built-in Data Types

- **Tuple**: A immutable record of arbitrary objects of any type

t1 = (1, “David”, 3.14, “Mark”, “Ann”)

index : 0 1 2 3 4

- Indexed by integer, starting with zero:

```
a = t1[1]          # returns the 2nd item “David” ; a = “David”
t1[0] = “John”     # Wrong operations! Tuple is immutable!
```

- 0-tuple (empty tuple) and 1-tuple:

```
t2 = ()            # an empty tuple ; same as t2 = tuple()
t3 = (“apple”,)    # a tuple containing 1 item ; note the trailing comma!
```

- Extract a portion of a list by slicing operator [i, j], with an index range of i<=k<j:

```
a = t1[0:2]        # returns (1, “David”) ; the 3rd item 3.14 is NOT extracted!
b = t1[2:]         # returns (3.14, “Mark”, “Ann”)
```

Basic Built-in Data Types

- Concatenate and multiply tuples:

```
t4 = t1 + t3          # t4 = (1, “David”, 3.14, “Mark”, “Ann”, “apple”)
t5 =  t3 * 3          # t5 = (“apple”, “apple”, “apple”)
```

- Count occurrences of items:

```
t5.count(“apple”)     # returns 3
```

- Extract values in a tuple without using index:

```
t6 =  (1, 2, 3)                          # create a new tuple
a, b, c = t6                              # a = 1 ; b = 2 ; c = 3
person = (“John”, “Smith”, 30)            # another example
first_name, last_name, age = person       # first_name = “John” ; last_name = “Smith” ; age = 30
```

Basic Built-in Data Types

- **String**: A immutable sequence of characters

s = “HELLO”

index : 0 1 2 3 4

- To create a string, enclose characters in single(‘ ’), double(“ ”), or triple(“““ ””” or ‘‘‘ ’’’) quotes:

```
a = ‘Mark’                  # ‘ ’ is usually for short strings
b = “Python is good!”       # “ ” is usually for string messages to be visible to human
c = “““This function        # “““ ””” or ‘‘‘ ’’’ is usually for Python doc strings ; can be used for a string
is for                      # spanning multiple lines
calculation of PI”””

d = ‘we say “yes!”’         # same type of quotes used to start a string must be used to terminate it!
d = “we say ‘yes!’”
d = “““we say ‘yes!’”””
d = ‘‘‘we say “yes!”’’’
```

Basic Built-in Data Types

- Indexed by integer, starting with zero:

```
a = "Hello Python!"      # a string a[0] = 'H' , a[1] = 'e' , a[2] = 'l' , a[3] = 'l' , ..... , a[11] = 'n' , a[12] = '!'
b = a[4]                 # b = 'o'
```

- Extract a portion of a string by slicing operator [i, j], with an index range of i<=k<j:

```
b = a[0:5]               # b = 'Hello'
b = a[6:]                # b = 'Python!'
b = a[4:7]               # b = 'o P'
```

- Concatenate and multiply strings:

```
c = "My name is John."   # a new string
d = a + ' ' + c          # d = "Hello Python! My name is John."
d = a * 2                # d = "Hello Python!Hello Python!"
```

Basic Built-in Data Types

- Conversion between numbers and strings :

```
a = '77' ; b = '23'          # two numeric strings
c = a + b                    # c = '7723' ; string concatenation ; NO numeric evaluation!
c = int(a) + int(b)          # c = 100
c = float(a) + int(b)        # c = 100.0

i = 77 ; f = 23.0            # two numbers
a = str(i)                   # a = '77'
b = str(f)                   # b = '23.0'
```

- Common string methods:

Next Page!

Basic Built-in Data Types

s = “python is good!”

String Methods	Description	Examples
s.capitalize()	Capitalize the 1st character	“Python is good!”
s.center(*w*, *p*) s.ljust(w, p) s.rjust(w, p)	Centers s in a field of length *w*, padding with *p* Left-align/Right-align s with *w* and *p*	(w=30, p=‘-’) : -------python is good!-------- python is good!----------
s.count(*substr*)	Counts occurrences of *substr*	s.count(‘o’) returns 3
s.isalpha() s.isdigit() s.isalnum() s.islower() s.isupper()	True if all characters in s are alphabetic/digits/alphanumeric/lowercase/uppercase	s.isalpha() returns True s.islower() returns True
s.find(*substr*)	Finds the 1st occurrence of *substr* or returns -1	s.find(‘good’) returns 10
s.index(*substr*)	Finds the 1st occurrence of *substr* or raises an error	s.index(‘good’) returns 10
s.replace(old, new)	Replaces a substring	s.replace(‘good’, ‘bad’) returns “python is bad!”
s.split(*sep*)	Splits a string using *sep* as a delimiter	s.split(‘is’) returns ['python ', ' good!']
s.partition(*sep*)	Partitions a string based on *sep*; returns (head, sep, tail)	s.partition(‘is’) returns ('python ', 'is', ' good!')

Basic Built-in Data Types

- Built-in operations common to all sequences: list, tuple, and string

s = “python is good!”

list1 = [0, 1, 2, 3, 4]

Operations	Description	Examples
seq[i] seq[i:j]	Returns the element at index i Returns a slice with an index range of i<=k<j	s[0] returns ‘p’ s[0:6] returns ‘python’
len(seq)	Number of elements in seq	len(s) returns 15
min(seq)	Minimum value in seq	min(s) returns ‘ ’
max(seq)	Maximum value in seq	max(s) returns ‘y’
sum(seq)	Sum of items in seq ; ONLY working for numeric list or tuple!	sum(list1) returns 10
all(seq)	True if all items in seq are True	all(list1) returns False
any(seq)	True if any item in seq is True	any(list1) returns True

Thank You!

Let's talk about *Python function and class*
on next class!

I : Python introduction, running python, Python built-in data types

II : function (procedural and functional programming) and class (OOP)

III: module, package, and practical code sample