Introduction to Python

Yang Liu
Associate Research Scientist
High Performance Research Computing
Texas A&M University

June 6, 2017

Why Python

Quick for a prototype:

- General purpose high-level programming language (file, network, web, database, GUI, etc.)
- Abundant packages (numpy, scipy, biopython, pandas, ntlk, tensorflow, etc.)

Short History

Created by **Guido Van Rossum** in early 1990s.

- Python conceived – last 1980s
- Python 1.0 – 1994
- Python 2.0 – 2000
  - 2.7 – 2010 (**last** version of Python 2)
- Python 3.0 – 2008 (if no legacy issue, use python 3. In this short course, we will learn python 3).
  - 3.5 – 2015

Access Python on Ada

Recall: “ssh your_net_id@ada.tamu.edu“ to login to ada cluster

```
[yangliu@ada2 ~]$ module load Anaconda/3-4.0.0
[yangliu@ada2 ~]$ python -V
Python 3.5.1 :: Anaconda 4.0.0 (64-bit)
[yangliu@ada2 ~]$ which python
/software/tamusc/Anaconda/3-4.0.0/bin/python
```

load Anaconda module
list python version

list which python to use

Data Type: Number

A number can be an integer, a floating pointer number, or a boolean. (Everything in Python is an object)

```
>>> a = 3
>>> b = 5.5
>>> c = True
>>> a + b + c
9.5
>>> c
True
>>> -c
-1
>>> type(c)
<class 'bool'>
```

```
# integer
# floating point number
# boolean
# automatic conversion for number operations
```

```
# everything is an object in python
```

Data Type: String

A string is a sequence of characters quoted by double/single/triple quotes.

```
>>> name = "Ada Cluster"
>>> print(name)
Ada Cluster
>>> two_line_string = 'Ada Cluster \n Texas A&M University'
>>> print(two_line_string)
Ada Cluster
 Texas A&M University
>>> string_with_tripple_quote="""Ada Cluster
... Texas A&M University"""
>>> print(string_with_tripple_quote)
Ada Cluster
Texas A&M University
>>>
```

#string in single quote
#string in double quote (with escaped character ‘\n’)

string in triple quote which preserve its format

Data Type: List

A list is a collection of items (elements) which are enclosed by '[' and ']'.

```
>>> department = ['Biology', 'Civil Engineering', 'Math']
>>> print(department)
['Biology', 'Civil Engineering', 'Math']
>>> department[0] = 'English'
>>> print(department)
['English', 'Civil Engineering', 'Math']
>>> 'Biology' in department
False
>>> department = department + ['geoscient', 'finance']
>>> print(department)
['English', 'Civil Engineering', 'Math', 'geoscient', 'finance']
>>>
```

define a list
print (function) a list
access a list element

operator "+" concatenate two lists

Slicing

Let x be a sequence (string, list, etc). Then a slicing x[start:end:step] (step is positive) is

- A sequence of elements: x[start], x[start + step], x[start + 2 * step], … , x[start + m * step] where
  - start + m * step < end and
  - start + (m+1) * step >= end

slicing definition is different when step is negative (not covered in this course)

```
>>> x = list(range(10))
>>> print(x)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x[0:10:2]
[0, 2, 4, 6, 8]
>>> x[3:5]
[3, 4]
>>>
```

List Comprehension

List comprehension is to create a new list from an old list by

- selecting elements from old list with/without conditions,
- And applying operations to those selected elements.

```
>>> [x + 100 for x in range(10)]
[100, 101, 102, 103, 104, 105, 106, 107, 108, 109]
>>> [x + 2 for x in range(10) if x%2 == 0]
[2, 4, 6, 8, 10]
>>>
```

slicing without condition

slicing with condition

Exercise:
Use list comprehension to create a new list: its elements are the double of those positive integers from a given list [-1, 5, 32, -98, 3912, -238, 72, 666].
Answer:
[10, 64, 7824, 144, 1332]

Flow Control: If Else

When the if condition is true, statements for the ‘if’ are executed. Otherwise, statements for the ‘else’ are executed.

```
>>> x = 3
>>> if x % 2 == 0:
...     print(x, " is an even number.")
... else:
...     print(x, " is an odd number.")
...
3  is an odd number.
```

do not forgot the colon “:”
condition “x %2 == 0” is true, the print statement following ‘if’ is executed

```
>>> x = 4
>>> if x % 2 == 0:
...     print(x, " is an even number.")
... else:
...     print(x, " is an odd number.")
...
4  is an even number.
```

condition “x %2 == 0” is false, the print statement following ‘else’ is executed

Indention in Python

Python use indentation (white spaces) to group program statements (C/C++ uses '{' and '}').

- Adjacent statements with the same indentations belong to the same group.

```
>>> x = 3
>>> if x % 2 == 0:
...     print(x, " is an even number.")
...     print("Can you see me")
...
>>> x = 4
>>> if x % 2 == 0:
...     print(x, " is an even number.")
...     print("Can you see me")
...
4  is an even number.
Can you see me
```

The two print statements have the same indention. So they both belong to the if statement

operator '%' is an modular operation: a % b is the remainder of a/b. For example: 10 % 3 = 1 since 10 = 3* 3 + 1, that is, the remainder of 10 / 3 is 1.

'==' is a comparison operator. 'a == b' is true is a is equal to b, and false otherwise. Note it is different to the assignment operator '='.

Flow Control: For

Statements of ‘for’ loop are executed for each element in the sequence as specified in the ‘for’ loop.

```
>>> print(department)
['English', 'Civil Engineering', 'Math', 'geoscient', 'finance']
>>> for d in department:
...     print("The length of department (" + d + ") is ", len(d))
...
The length of department (English) is  7
The length of department (Civil Engineering) is  17
The length of department (Math) is  4
The length of department (geoscient) is  9
The length of department (finance) is  7
```

do not forgot the colon “:”

Flow Control: While

While statement continues to execute the statements belong to this loop until the condition is not true

```
>>> x = 0
>>> while x < 5:
...     print(x, x*x)
...     x = x + 1
...
0 0
1 1
2 4
3 9
4 16
```

reminder of the colon ":" and indention

Flow Control: Break

Break statement in a loop exits from that loop

```
>>> x = 237
>>> i = 2
>>> while i < x:
...     if x % i == 0:
...         print(x, '=', i, '*', int(x / i))
...         break
...     i = i + 1
...
237 = 3 * 79
```

‘break’ statement exits the while loop when it is executed

Exercise:
What will be the output if there is no ‘break’ statement in the example?

Example: Right Triangle

```
>>> x = 8
>>> for i in range(x):
...     for j in range(i + 1):
...         print('*', end = '')
...     print('')
...
*
**
***
****
*****
******
*******
********
```

```
>>> x = 8
>>> for i in range(x):
...     j = 0
...     while j <=i:
...         print('*', end = '')
...         j = j + 1
...     print('')
...
*
**
***
****
*****
******
*******
********
```

print() without ‘end =’ in it, it adds new line character ‘\n’ in the end of its output

print() with ‘end = x‘ in it, it adds character x in the end of its output

Exercise:
What will be the output if there is no ‘j = j + 1’ statement in the example above?

Problem: Finding Factors (1)

Write a python program to find the factors of 1000: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500. (range(1,1000) returns a list of integer from 1 to 1000: no 0)

- Using for loop
- Using while loop
- Using list comprehension

Problem Solution: Finding Factors (1)

```
>>> a = 1000
>>> for factor in range(1, 1000):
...     if a % factor == 0:
...         print(factor, end = ',')
...
1,2,4,5,8,10,20,25,40,50,100,125,200,250,500,>>>
>>>
>>> a = 1000
>>> i = 1
>>> while i < a:
...     if a % i == 0:
...         print(i, end = ',')
...     i = i + 1
...
1,2,4,5,8,10,20,25,40,50,100,125,200,250,500,>>>
>>>
>>> a = 1000
>>> [x for x in range(1, 1000) if a %x == 0]
[1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500]
```

Python Scripts

A python script is a file with python statements.

```
[yangliu@ada2 2017-Summer-Research-Week]$ cat rightTriangle.py
x = 8
for i in range(x):
    for j in range(i+1):
        print('*', end = '')
    print('')
[yangliu@ada2 2017-Summer-Research-Week]$ ll rightTriangle.py
-rw-rw-r-- 1 yangliu tamusc 94 May 31 09:14 rightTriangle.py
[yangliu@ada2 2017-Summer-Research-Week]$ python rightTriangle.py
*
**
***
****
*****
******
*******
********
```

'python scrip_name' executes the python program in file script_name

Exercise:
What will happen if run 'module purge' before executing the script?

Python Executable Scripts

A python executable script is a python script with executable bit set.

```
[yangliu@ada2 2017-Summer-Research-Week]$ cat rightTriangle_executable.py
#!/bin/env python

x = 8
for i in range(x):
    for j in range(i+1):
        print('*', end = '')
    print('')
[yangliu@ada2 2017-Summer-Research-Week]$ chmod +x rightTriangle_executable.py
[yangliu@ada2 2017-Summer-Research-Week]$ ll rightTriangle_executable.py
-rwxrwxr-x 1 yangliu tamusc 113 May 31 09:33 rightTriangle_executable.py
[yangliu@ada2 2017-Summer-Research-Week]$ ./rightTriangle_executable.py
*
**
***
****
*****
******
*******
********
```

"#!/bin/env python" is to use python found in environment (env)

"chmod +x" sets the file to be an executable file
no need of 'python' in front of the script name to execute the script

Exercise:
What will happen if run 'module purge' before executing the script?

Edit Python Scripts on Ada

Depends on which operating systems you use, there are various ways to connect to Ada to edit python scripts on Ada.

- “ssh” to ada and use an editor (vi, emacs, etc)
- “ssh -X” to ada and use gedit (slow sometimes)
- Edit python scripts on your computer (windows, mac, etc) and copy to ada.
- …

Built-in Functions

The Python interpreter provides a number of built-in functions:
https://docs.python.org/3/library/functions.html

```
>>> sorted([1, -2, 3, -4])
[-4, -2, 1, 3]
>>> abs(-5.5)
5.5
>>> int(-5.5)
-5
>>> sum([1, -2, 3, -4])
-2
```

function ‘sorted’ sorts all elements (in a list)

function ‘abs’ returns the absolute value of a number

funciton ‘int’ convert its parameter to integer

function ‘sum’ returns the sum of all elements

User Defined Functions

A user may define a function other than just using built-in functions

```
[yangliu@ada2 2017-Summer-Research-Week]$ cat function_factor.py
#!/bin/env python

def factor(n):
    product = 1
    for i in range(1,n):
        product = product * i
    return product

print("5! = ", factor(5))
print("10! = ", factor(10))
[yangliu@ada2 2017-Summer-Research-Week]$ ./function_factor.py
5! =  24
10! =  362880
```

‘def’ defines a function. Do not forgot the colon “:”

‘n’ is the argument/parameter for function factor

reminder of indention

Exercise:
What will happen if range(1,n) is changed to range(n)?

File I/O

A python program can read from or write to a file.

```
[yangliu@ada2 2017-Summer-Research-Week]$ cat file_io.py
#!/bin/env python

input_file = open('integers.txt', 'r')       #open the input file to read only
output_file = open('integers_sum.txt', 'w')  #open the ouptut file to write only
total_sum = 0
for line in input_file:                      #read the input file line by line:
    total_sum = total_sum + int(line)
output_file.write(str(total_sum) + "\n")
input_file.close()                           #close the input file
output_file.close()                          #close the output file
[yangliu@ada2 2017-Summer-Research-Week]$ cat integers.txt
1
3
5
7
9
11
[yangliu@ada2 2017-Summer-Research-Week]$ ./file_io.py
[yangliu@ada2 2017-Summer-Research-Week]$ cat integers_sum.txt
36
```

File I/O: Automatically Close

A file is automatically closed after ‘with’ statement is finished if that file is opened in the ‘with’ statement.

```
[yangliu@ada2 2017-Summer-Research-Week]$ cat file_io_auto_close.py
#!/bin/env python

with open('integers_sum.txt', 'w') as output_file:  #open the ouptut file to write only
    with open('integers.txt', 'r') as input_file:   #open the input file to read only
        total_sum = 0
        for line in input_file:                     #read the input file line by line:
            total_sum = total_sum + int(line)
    output_file.write(str(total_sum) + "\n")
[yangliu@ada2 2017-Summer-Research-Week]$ cat integers.txt
2
4
6
8
10
12
[yangliu@ada2 2017-Summer-Research-Week]$ ./file_io_auto_close.py
[yangliu@ada2 2017-Summer-Research-Week]$ cat integers_sum.txt
42
```

Problem: Finding Factors (2)

Write a program (script) factors.py which reads input from input file number.txt of positive integers (one per line) and write the factors of those integers to output file number_factors.txt (all factors for an integer are in the same line). Your program should have a function finding_factors which takes an integer as a parameter and return a list of integers.

Example number.txt:
6
10
25

Example number_factor.txt:
1 2 3 6
1 2 5 10
1 5 25

Problem Solution: Finding Factors (2)

```
#!/bin/env python

def finding_factors(n):
    return [x for x in range(1, n) if n %x == 0]

with open("number_factors.txt", 'w') as output_file:
    with open("numbers.txt", 'r') as input_file:
        for number in input_file:
            factors = finding_factors(int(number))
            for factor in factors:
                output_file.write(str(factor) + ' ')
            output_file.write('\n')
```

Module

A module is a python script. We use the terminology ‘module’ when importing a python script (or library)

```
[yangliu@ada2 module]$ cat ../function/function_factor.py
#!/bin/env python

def factor(n):
    product = 1
    for i in range(1,n):
        product = product * i
    return product

print("5! = ", factor(5))
print("10! = ", factor(10))
[yangliu@ada2 module]$ python
Python 3.5.1 |Anaconda 4.0.0 (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.
>>> factor(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'factor' is not defined
>>>
```

the function_factor.py is not in current directory and it includes function factor().

failed to call the function factor() which is in a program not at current directory

Module: Import

We can import a module which is found in PYTHONPATH

```
[yangliu@ada2 module]$ echo $PYTHONPATH

[yangliu@ada2 module]$ export PYTHONPATH=/scratch/training/Python/2017-Summer-Research-Week/Examples/function/:$PYTHONPATH
[yangliu@ada2 module]$ echo $PYTHONPATH
/scratch/training/Python/2017-Summer-Research-Week/Examples/function/:
[yangliu@ada2 module]$ python
Python 3.5.1 |Anaconda 4.0.0 (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.
>>> import function_factor
5! =  24
10! =  362880
>>> function_factor.factor(5)
24
>>> factor(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'factor' is not defined
```

PYTHONPATH sets the paths/directories to search for python modules

Once we add the directory where the function-factor is to PYTHONPATH (by 'export PYTHONPATH=...'), the factor function can be found and is available (with module name is a prefix to the function name)

Module: Import … From

"Import" make all contents from the imported module available. "Import … From" can just import needed contents from the module to import.

```
[yangliu@ada2 module]$ python
Python 3.5.1 |Anaconda 4.0.0 (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.
>>> from function_factor import factor
5! =  24
10! =  362880
>>> factor(5)
24
>>> function_factor.factor(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'function_factor' is not defined
```

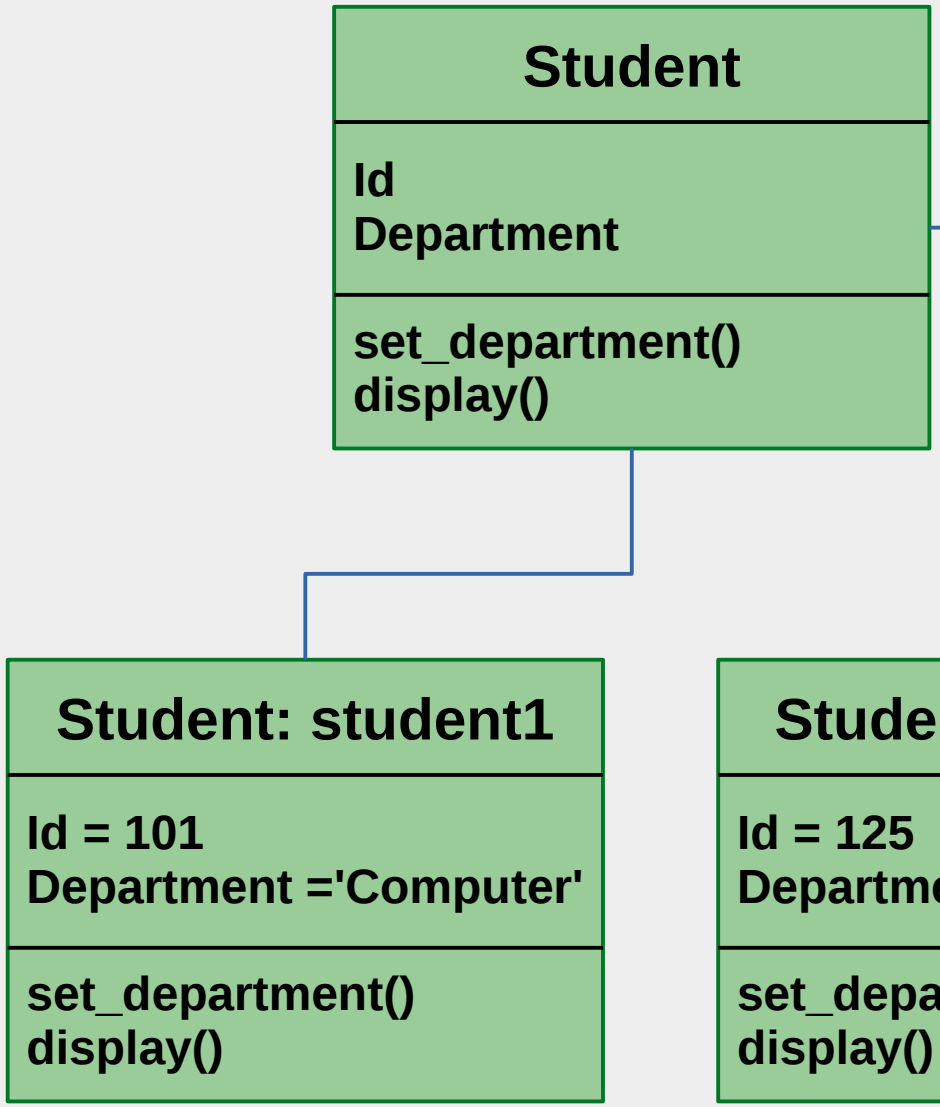
only import 'factor' from module function_factor

module name is not available (not imported)

Problem: Finding Factors (3)

Go to a directory different to where your solution to problem 2 is. Import the module factors.py and call its function to return the factors for integer 2018.

Object Oriented Programming



Class:

- Include both data and methods (functions)
- Used as a prototype to create instances

Instance:

- Has own data

Class: Example

```
[yangliu@ada2 class]$ cat student.py
#!/bin/env python

class Student:
    def __init__(self, name, department):
        self.name = name
        self.department = department

    def display(self):
        print('*****Student Information---begin*****')
        print('Student Name: ', self.name)
        print('Department: ', self.department)
        print('*****Student Information---end*******')

student_1 = Student('Mike Williams', 'Physics')
student_2 = Student('Eric Garcia', 'English')
student_1.display()
student_2.display()
[yangliu@ada2 class]$ ./student.py
*****Student Information---begin*****
Student Name:  Mike Williams
Department:  Physics
*****Student Information---end*******
*****Student Information---begin*****
Student Name:  Eric Garcia
Department:  English
*****Student Information---end*******
```

Class constructor "__**init**__" executed when an instance is created

instance student_1 has its own data

instance student_2 has its own data

Class Variable

A class variable is shared among all instances: one instance change a class variable, the other instances see the change

```
[yangliu@ada2 class]$ cat student_with_class_variable.py
#!/bin/env python

class Student:
    count = 0
    def __init__(self, name, department):
        Student.count = Student.count + 1
        self.id = Student.count
        self.name = name
        self.department = department

    def display(self):
        print('*****Student Information---begin*****')
        print('Student ID: ', self.id)
        print('Student Name: ', self.name)
        print('Department: ', self.department)
        print('Student Count: ', Student.count)
        print('*****Student Information---end*******')

student_1 = Student('Mike Williams', 'Physics')
student_1.display()
student_2 = Student('Eric Garcia', 'English')
student_1.display()
student_2.display()
```

‘count’ is a class variable

```
[yangliu@ada2 class]$ ./student_with_class_variable.py
*****Student Information---begin*****
Student ID:  1
Student Name:  Mike Williams
Department:  Physics
Student Count:  1
*****Student Information---end*******
*****Student Information---begin*****
Student ID:  1
Student Name:  Mike Williams
Department:  Physics
Student Count:  2
*****Student Information---end*******
*****Student Information---begin*****
Student ID:  2
Student Name:  Eric Garcia
Department:  English
Student Count:  2
*****Student Information---end*******
```

Note that the ‘Student Count’ in output from the first student_1.display() is different to that from the second student_1.display(). The change is caused by the creation of stduent_2.

Class Inheritance

```
[yangliu@ada2 class]$ cat student_inheritance.py
#!/bin/env python

class Student:
    def __init__(self, name, department):
        self.name = name
        self.department = department

    def display(self):
        print('*****Student Information---begin*****')
        print('Student Name: ', self.name)
        print('Department: ', self.department)
        print('*****Student Information---end*******')

class Graduate_Student(Student):
    def set_advisor(self, advisor):
        self.advisor = advisor
    def display(self):
        print('*****Graduate Student Information---begin*****')
        print('Student Name: ', self.name)
        print('Department: ', self.department)
        print('*****Graduate Student Information---end*******')

graduate_student_1 = Graduate_Student('Mike Williams', 'Physics')
graduate_student_1.display()
[yangliu@ada2 class]$ ./student_inheritance.py
*****Graduate Student Information---begin*****
Student Name:  Mike Williams
Department:  Physics
*****Graduate Student Information---end*******
```

A class can inherit the data and functions in its parent class. But it can also override its parent functions and add new data

Graduate_Student inherits from its parent Student

Child class Graduate_student can have new function ‘set_advisor’

Child class Graduate_Student override its parent function ‘display’

Matplotlib

Matplotlib is a Python 2D plotting library. It

- produces publication quality figures
- supports different Python versions.

  matplotlib 1.5 supports Python 2.7, 3.4, and 3.5

  matplotlib 1.4 supports Python 2.6, 2.7, 3.3, and 3.4

  matplotlib 1.3 supports Python 2.6, 2.7, 3.2, and 3.3

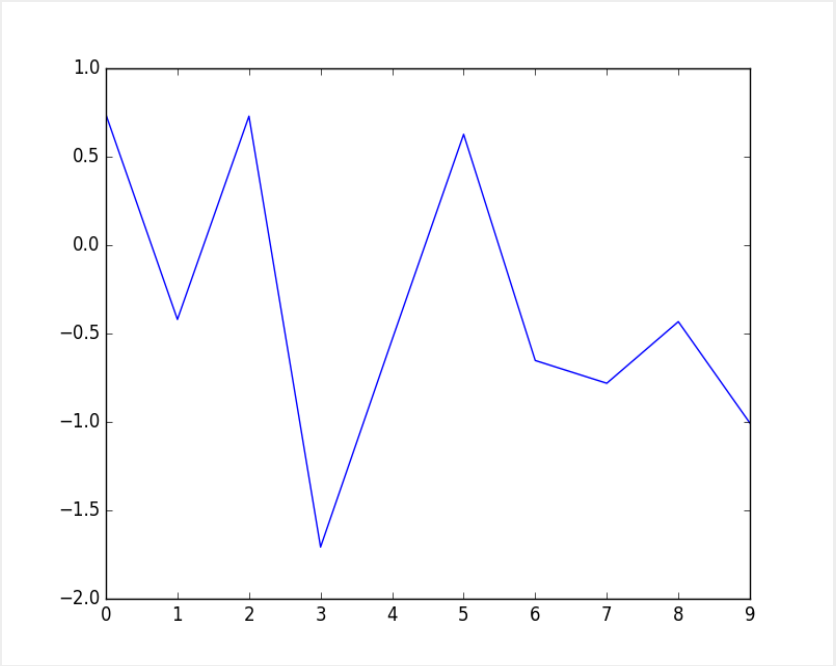
  matplotlib 1.2 supports Python 2.6, 2.7, and 3.1

  matplotlib 1.1 supports Python 2.4 to 2.7

Matplotlib: pyplot

Matplotlib.pyplot is a module with functions to make figures like MATLAB

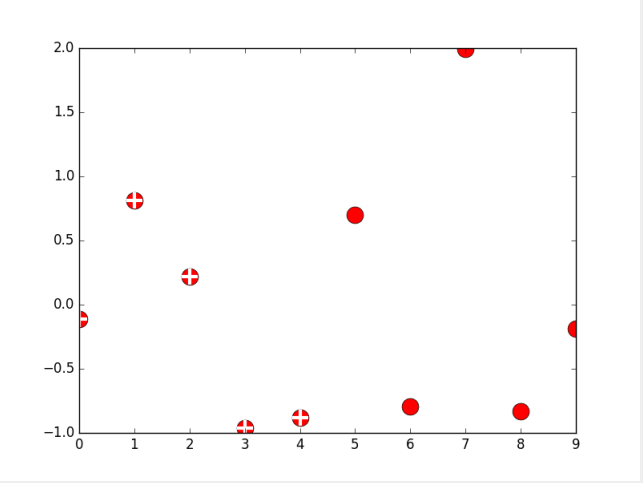
```
[yangliu@ada2 matplotlib]$ cat example_pyplot.py
#!/bin/env python

from matplotlib import pyplot
from numpy.random import randn

z = randn(10) #generates 10 random numbers

red_dot = pyplot.plot(z) #plot the 10 numbers
pyplot.show()  #show the figure
[yangliu@ada2 matplotlib]$ ./example_pyplot.py
```



10 random points are connected via lines

Matplotlib: Marker

```
[yangliu@ada2 matplotlib]$ cat example_marker.py
#!/bin/env python

from matplotlib import pyplot
from numpy.random import randn

z = randn(10)

#red 'o' marker with size 15
red_dot = pyplot.plot(z, "ro", markersize=15)
#white '+' marker with size 15 and edgewidth 3
white_cross = pyplot.plot(z[:5], "w+", markeredgewidth=3, markersize=15)

pyplot.show()
[yangliu@ada2 matplotlib]$ ./example_marker.py
```

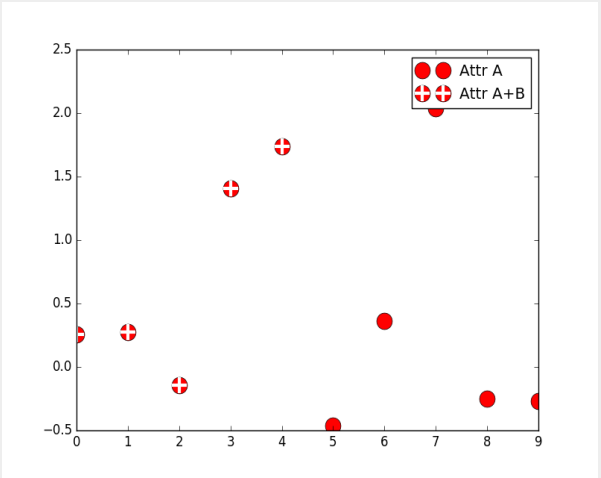


all 10 points have marker of red dot

first 5 points have marker of white cross

Matplotlib: Lengend

```
[yangliu@ada2 matplotlib]$ cat example_legend.py
#!/bin/env python

from matplotlib import pyplot
from numpy.random import randn

z = randn(10)

#red 'o' marker with size 15
red_dot, = pyplot.plot(z, "ro", markersize=15)  #',': a single element tuple
#white '+' marker with size 15 and edgewidth 3
white_cross, = pyplot.plot(z[:5], "w+", markeredgewidth=3, markersize=15)

pyplot.legend([red_dot, (red_dot, white_cross)], ["Attr A", "Attr A+B"])

pyplot.show()
[yangliu@ada2 matplotlib]$ ./example_legend.py
```



legend shows the meaning of two different markers

Parallel Processing

- Multithreading
  - Multiple threads sharing memory
  - Great for I/O-bound applications
- Multiporcessing
  - Multiple processes
  - Good for CPU-bound applications
- Mpi4py
  - mpi for python
  - Great for CPU-bound applications

Other Python Modules

Module spider Python

- Python/2.7.12-intel-2017A
- Python/2.7.12-iomkl-2016D
- Python/2.7.12-2017.0.035
- Python/3.4.3-intel-2015B
- Python/3.5.1-foss-2016a
- Python/3.5.1-intel-2016a
- Python/3.5.2-foss-2016b
- Python/3.5.2-intel-2016a
- Python/3.5.2-intel-2016b
- Python/3.5.2-2017.0.035
- and more...

If you need python with particular version/compiler, those modules are available for you to select.
Otherwise, anaconda and its virtual environments are easier to use.

Upcoming Schedule (HPRC Research Week)

- Tuesday
  - Nvidia Deep Learning Institute lecture, 1:15 – 3:30
  - Data Literacy and Data Management, 3:45 – 4:45
- Wednesday
  - **Keynote talk**: NSF's computing and Information Science and Engineering Directorate (CISE) Research and Cyberinfrastructure Priorities, 10:00 – 11:00
  - Matlab Workshop on Data Analytics, Machine Learning, and Code Optimization, 1:00 – 4:30
- More on Thursday and Friday.
- Details: https://sites.google.com/a/tamu.edu/rcompweek/home/agenda