Introduction to Python

Yang Liu

June 7, 2016

Bioinformatics Workshop
High Performace Research Computing
AgriLife Genomics & Bioinformatics Services
College of Veterinary Medicine & Biomedical Services

Python Short History

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

- Python Conceived --- late 1980s
- Python 1.0 --- 1994
- Python 2.0 --- 2000
  - 2.6 --- 2008
  - 2.7 --- 2010 (**last** version of Python 2)
- Python 3.0 --- 2008
  - 3.5 --- 2015

Philosophy

There should be **one**—and perferable only one—obvious way to do it

- Python 3 was designed to reduce feature duplications (in Python 2) by removing old ways of doing things.

Why Python

Quick for protoype

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

Access Python on Ada

```
$ module load Anaconda/3-4.0.0
    ###Load Anaconda module including Python 3.5.1

$ python -V
Python 3.5.1 :: Anaconda 4.0.0 (64-bit)
   ###check python version

$ which python
/software/tamusc/Anaconda/3-4.0.0/bin/python
   ###check which python to use
```

Data Type (1): *Numbers*

```
>>> a = 3;          ### variable a is an integer, no need to declare variable and its type
>>> b = 5.5;        ### variable b is a floating point number
>>> c = True;       ### variable c is a boolean (subset of integers)
>>> a + b - c;
7.5
>>> c
True
>>> -c
-1
>>> type(c)
<class 'bool'>
```

Data Type (2): Strings

```
### Strings can be quoted by double/single/tripple quotes
>>> name = "Ada cluster"                                    ### String using double quotes
>>> print(name);
Ada cluster
>>> two_line_string = 'Ada Cluster \n Texas A&M University' ### String using single quotes
>>> print(two_line_string)
Ada Cluster
 Texas A&M University
>>> string_with_tripple_quote = '''Ada Cluster      ### String using tripple(either single or double) quotes
... Texas A&M University'''
>>> print(string_with_tripple_quote)
Ada Cluster
Texas A&M University
```

Data Type (3): *Lists*

```
>>> NBA_Players = ['Michael Jordan', 'Scottie Pippen', 'Shaquille O\'Neal']
        ### a list defined by '[' and ']'
>>> print(NBA_Players)
['Michael Jordan', 'Scottie Pippen', "Shaquille O'Neal"]
>>> NBA_Players[0] = 'Yang Liu'                    ### change an element (mutable)
>>> print(NBA_Players)
['Yang Liu', 'Scottie Pippen', "Shaquille O'Neal"]
>>> 'Micahel Jordan' in NBA_Players                ### operator 'in' check list has an element or not
False
>>> NBA_Players = NBA_Players + ['Michael Joran', 'Jeremy Lin']
        ### operator '+' concatenate two lists
>>> print(NBA_Players)
['Yang Liu', 'Scottie Pippen', "Shaquille O'Neal", 'Michael Joran', 'Jeremy Lin']
```

Slicing

- **x[start:end:step]** where x is a sequence (string, list, etc)
  - A list with elements from x: x[start], x[start+step], ...until x[start + m * step] where start + m * step < end and start + (m+1) * step >= end

```
>>> 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[0:8:2]
[0, 2, 4, 6]
>>> x[3:5]
[3, 4]
```

List Comprehension

```
>>> [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]
```

Flow Control (1): *If/Else*

```
>>> x = 3
>>> if x%2 == 0:                          ###If ends with ':'
...     print(x," is an even number.")###Space needed before print

... else:                                  ###Else ends with ':'
...     print(x," is an odd number.")  ###Space needed before print
...
3  is an odd number.
```

Flow Control (2): For

```
>>> print(NBA_Players)
['Yang Liu', 'Scottie Pippen', "Shaquille O'Neal", 'Michael Joran', 'Jeremy Lin']
>>> for player in NBA_Players:                    ###For ends with ':'
...     print('The length of player\'s name (' + player + ') is ', len(player))
...
The length of player's name (Yang Liu) is  8
The length of player's name (Scottie Pippen) is  14
The length of player's name (Shaquille O'Neal) is  16
The length of player's name (Michael Joran) is  13
The length of player's name (Jeremy Lin) is  10
```

Flow Control (3): *While*

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

Flow Control (4): *Break*

```
>>> x = 237
>>> i = 2
>>> while i < x:
...     if x%i == 0:
...         print(x,'=', i, '*', x/i)
...         break   ###exit the while loop when if condition is true, i.e, x%i is 0.
...     i = i + 1
...
237 = 3 * 79.0
```

List Comprehension

```
>>> [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]
```

Functions

```
>>> def isPrime(n):                    ###'def' ending with ':' defines a function
...     '''Checke whether the integer n is a prime number of not'''
...     i = 2
...     while i < n:
...         if n%i == 0:
...             print(n,'=', i, '*', n/i)
...             return True             ###'return' returns a value when the function terminates
...         i = i + 1
...     return False                    ###'return' returns a value when the function terminates
...
>>> isPrime(237)
237 = 3 * 79.0
True
>>> isPrime(227)
False
```

Scripts

- Python code saved in a file
- Can be run again when needed
- Good for batch jobs
- Example: isPrime.py

```
def isPrime(n):
    i = 2
    while i < n:
        if n%i == 0:
            print(n,'=', i, '*', n/i)
            return True
        i = i + 1
    return False

isPrime(237)
isPrime(227)
```

$ module load Anaconda/3-4.0.0

$ **python isPrime.py** ### tells computer to interpret/execute the contents in isPrime.py using python

237 = 3 * 79.0

Scripts

- Python code saved in a file
- Can be run again when needed
- Good for batch jobs
- Example: isPrime.py

```
def isPrime(n):
    i = 2
    while i < n:
        if n%i == 0:
            print(n,'=', i, '*', n/i)
            return True
        i = i + 1
    return False

isPrime(237)
isPrime(227)
```

$ module load Anaconda/3-4.0.0

$ **python isPrime.py** ### tells computer to interpret/execute the contents in isPrime.py using python

237 = 3 * 79.0

Executable Scripts

- A script with executable bit set
- Example for isPrime.py
  - $ **chmod +x isPrime.py**

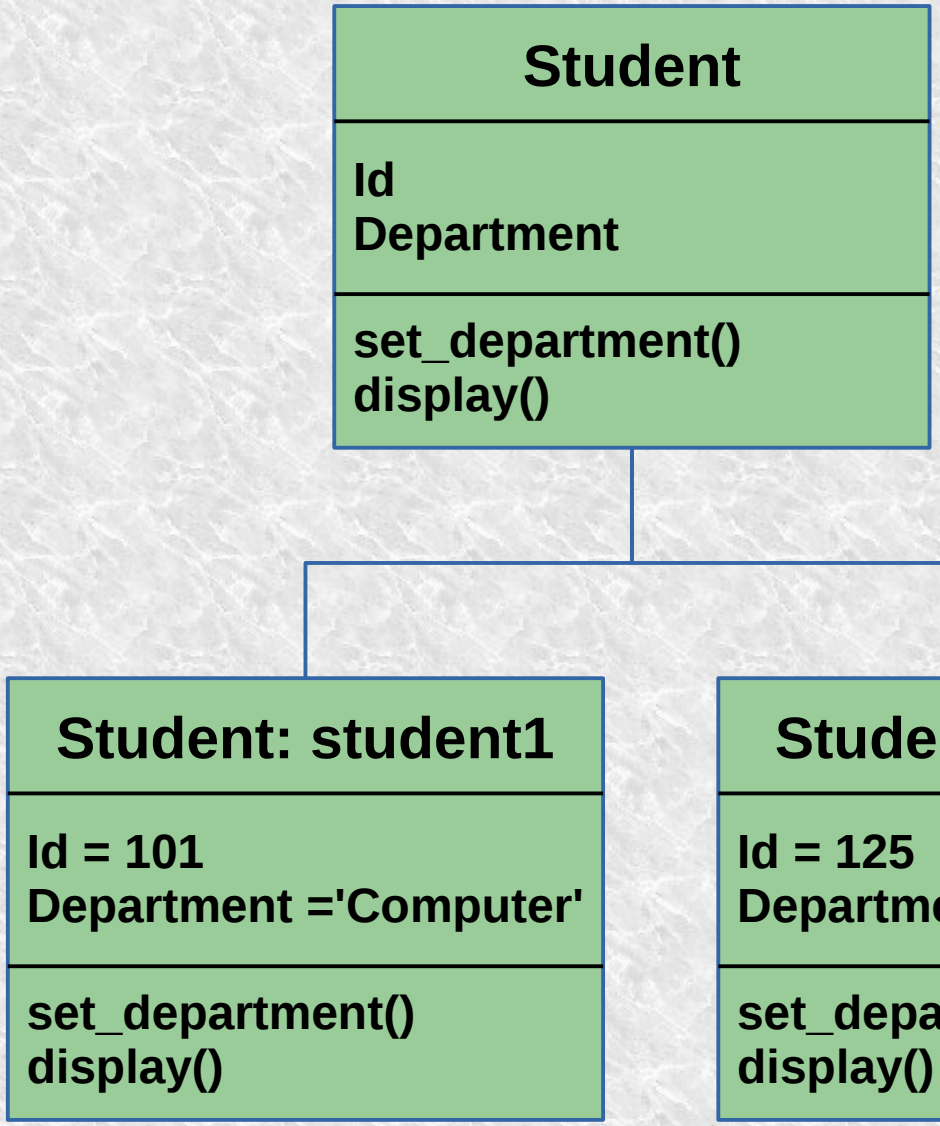
    ### make isPrime.py to be executable
  - Add '**#!/bin/env python**'

    ### use python found in env to execute program
  - module load Anaconda/3-4.0.0
  - $ **./isPrime.py**

    ### execute the executable file
  - 237 = 3 * 79.0

```
#!/bin/env python
def isPrime(n):
    i = 2
    while i < n:
        if n%i == 0:
            print(n,'=', i, '*', n/i)
            return True
        i = i + 1
    return False

isPrime(237)
isPrime(227)
```

Object Oriented Programming



Class:

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

Instance:

- Has own data

Class: Example

```
#!/bin/env python

class Student:
    count = 0

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

    def display(self):
        print(“Studnet ID:”, self.id)
        print("Student Name:",self.name)
        print("Student Department:",self.department)
        print("Total Students:", Student.count)
        print("Student Count?:", self.count)

student1 = Student('Yang Liu', 'Computer Science')
student2 = Student(“Michael Jordan', 'Basketball')
student1.display()
student2.display()
```

A class consists of data and methods/functions.

Example:
student.py

```
$ ./student.py
Student ID: 1
Student Name: Yang Liu
Student Department: Computer Science
Total Students: 2
Student Count?: 0
Student ID: 2
Student Name: Michael Jordan
Student Department: Basketball
Total Students: 2
Student Count?: 0
```

Class: Example Details

```
#!/bin/env python

class Student:                                  ### defined by Class and ':'
   count = 0                                    ### class variable (a data member)

   def __init__(self, name, department):        ### Constructor (one method member called when an object is created)
      Student.count = Student.count + 1
      self.name = name
      self.department = department
      self.id = Student.count
      self.count = 0                            ### instace variable 'count'

   def display(self):                           ### another meothd member
      print(“Studnet ID:”, self.id)
      print("Student Name:",self.name)
      print("Student Department:",self.department)
      print("Total Students:", Student.count)
      print("Student Count?:", self.count)

student1 = Student('Yang Liu', 'Computer Science')
student2 = Student(“Michael Jordan', 'Basketball')
student1.display()
student2.display()
```

```
$ ./student.py
Student ID: 1
Student Name: Yang Liu
Student Department: Computer Science
Total Students: 2
Student Count?: 0
Student ID: 2
Student Name: Michael Jordan
Student Department: Basketball
Total Students: 2
Student Count?: 0
```

Class Inheritance: Example

```
class GraduateStudent(Student):
    count = 0
    advisor = ''

    def set_advisor(self, advisor):
        self.advisor = advisor

    def display(self):
        print("Graduate Student ID:", self.id)
        print("Student Name:", self.name)
        print("Student Department:", self.department)
        print("Student Advisor:", self.advisor)
        print("Total Graduate Students:", Student.count)

graduateStudent1 = GraduateStudent("Yang Liu", 'Computer Science')
graduateStudent2 = GraduateStudent("Michael Jordan", 'Basketball')
graduateStudent1.set_advisor('Alan Turing');
graduateStudent2.set_advisor('Michael Young');
graduateStudent1.display()
graduateStudent2.display()
```

Example:
student.py (incomplete)

$ ./student.py (partial output below)
Graduate Student ID: 3
Student Name: Yang Liu
Student Department: Computer Science
Student Advisor: Alan Turing
Total Graduate Students: 4
Graduate Student ID: 4
Student Name: Michael Jordan
Student Department: Basketball
Student Advisor: Michael Young
Total Graduate Students: 4

Class Inheritance: Example Details

```
class GraduateStudent(Student):        ### inherite from parent class Student
    count = 0
    advisor = ''

    def set_advisor(self, advisor):        ### add new method in child class (GraduateStudent)
        self.advisor = advisor             ### instance variable (not GraduateStudent.advisor which is a class variable)

    def display(self):                     ### overload themethodfrom parent Student
        print("Graduate Student ID:", self.id)
        print("Student Name:", self.name)
        print("Student Department:", self.department)
        print("Student Advisor:", self.advisor)
        print("Total Graduate Students:", Student.count)

graduateStudent1 = GraduateStudent("Yang Liu", 'Computer Science')
graduateStudent2 = GraduateStudent("Michael Jordan", 'Basketball')
graduateStudent1.set_advisor('Alan Turing');
graduateStudent2.set_advisor('Michael Young');
graduateStudent1.display()
graduateStudent2.display()
```

File I/O

```
#!/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 output 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))                  ### write to the output file
input_file.close()                                        ### close the input file
output_file.close()                                      ### close the output file
```

File I/O: Automatically Close

```
with open('integers_sum.txt', 'w') as output_file:
    with open('integers.txt', 'r') as input_file:
        total_sum = 0
        for line in input_file:
            total_sum = total_sum + int(line)
    output_file.write(str(total_sum))
```

File I/O: Automatically Close

```
with open('integers_sum.txt', 'w') as output_file:
    with open('integers.txt', 'r') as input_file:
        total_sum = 0
        for line in input_file:
            total_sum = total_sum + int(line)
    output_file.write(str(total_sum))
```

Module: Import

Is a file containing python code (not a Lmodule for software on Ada!)

Example: module_student.py

module_test.py

```
#!/bin/env python
import module_student

graduate_student1 =
module_student.GraduateStudent('Yang
Liu', 'Computer Science')
graduate_student1.display()
```

```
#!/bin/env python

class Student:
    count = 0

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

    def display(self):
        print("Student ID:", self.id)
        print("Student Name:", self.name)
        print("Student Department:", self.department)
        print("Total Students:", Student.count)

class GraduateStudent(Student):
    count = 0
    advisor = ''

    def set_advisor(self, advisor):
        self.advisor = advisor

    def display(self):
        print("Graduate Student ID:", self.id)
        print("Student Name:", self.name)
        print("Student Department:", self.department)
        print("Student Advisor:", self.advisor)
        print("Total Graduate Students:", Student.count)
```

Module: From … Import

Example: module_student.py

module_test.py

```
#!/bin/env python
from module_student import
GraduateStudent

graduate_student1 =
GraduateStudent('Yang Liu', 'Computer
Science')
graduate_student1.display()
```

```
#!/bin/env python

class Student:
    count = 0

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

    def display(self):
        print("Student ID:", self.id)
        print("Student Name:", self.name)
        print("Student Department:", self.department)
        print("Total Students:", Student.count)

class GraduateStudent(Student):
    count = 0
    advisor = ''

    def set_advisor(self, advisor):
        self.advisor = advisor

    def display(self):
        print("Graduate Student ID:", self.id)
        print("Student Name:", self.name)
        print("Student Department:", self.department)
        print("Student Advisor:", self.advisor)
        print("Total Graduate Students:", Student.count)
```

Module: Search Path

- Current directory
- Path specified by enviorment variable **PYTHONPATH**
- Default path

Matplotlib

- a python 2D plotting library
- produces publication quality figures
- support 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

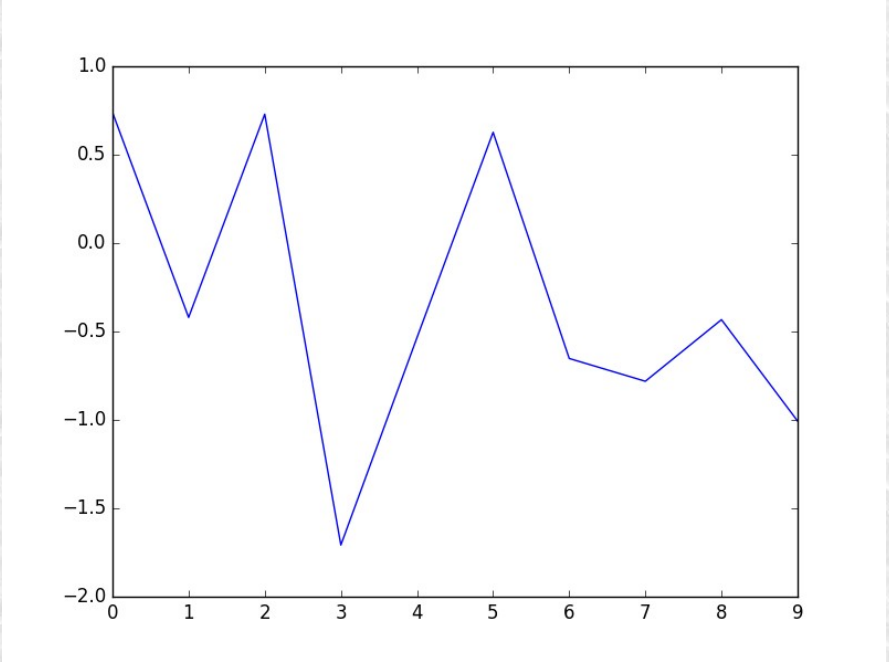
  matplotlib 1.3 supports Python 2.6, 2.7, 3.2, and 3.3

  matplotlib 1.2 supports Python 2.6, 2.7, and 3.1

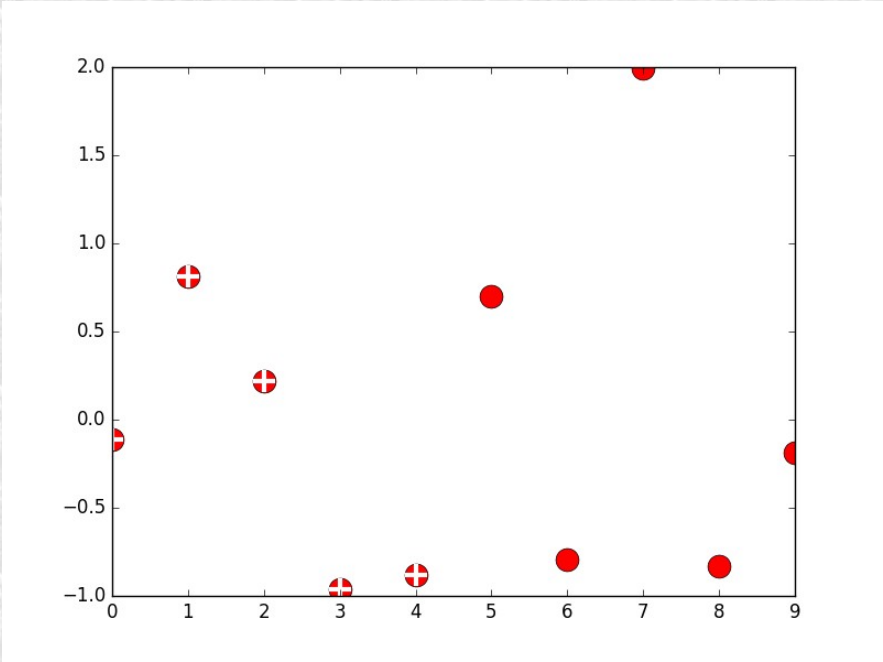
  matplotlib 1.1 supports Python 2.4 to 2.7

Matplotlib: *pyplot*

```
#!/bin/env python

from matplotlib import pyplot
from numpy.random import randn

z = randn(10)    ### 10 random numbers

red_dot = pyplot.plot(z)    ### plot the 10 numbers
pyplot.show()                ### display the figure
```

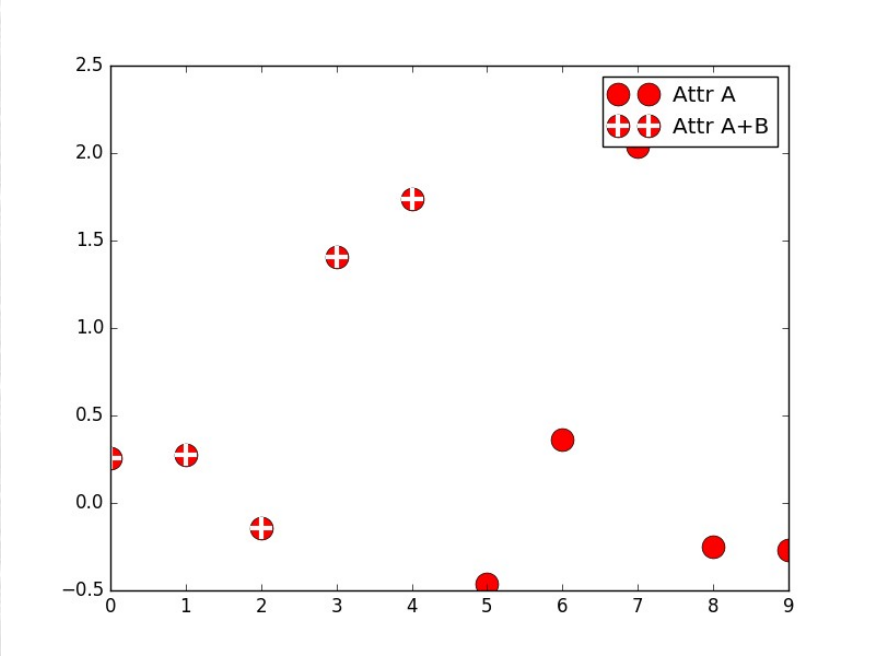


Matploglib:*Marker*




```
#!/bin/env python

from matplotlib import pyplot
from numpy.random import randn

z = randn(10)

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

pyplot.show()
```

Matplotlib: *Lengend*




```
#!/bin/env python

from matplotlib import pyplot
from numpy.random import randn

z = randn(10)

red_dot, = pyplot.plot(z, "ro", markersize=15) ### ',' : a single-element tuple
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()
```

Numpy and Scipy

- Numpy: **functions** for the manipulation of **arrays**
- Scipy: **functions** for **scientific data** such as Fourier transforms.

Numpy: Array

```
>>> from numpy import *
>>> a = array([[1,2,3], [4,5,6]])     ### create a 2x3 array
>>> print(a)
[[1 2 3]
 [4 5 6]]
>>> c = ones((2,3))
>>> a = array([[1,2,3], [4,5,6]])
>>> print(a)
[[1 2 3]
 [4 5 6]]
>>> b = ones((2,3))                   ### create a 2x3 array with 1s
>>> print(b)
[[ 1.  1.  1.]
 [ 1.  1.  1.]]
>>> b = ones((2,3), int)
>>> print(b)
[[1 1 1]
 [1 1 1]]
```

```
>>> b = 2 * b
>>> print (b)
[[2 2 2]
 [2 2 2]]
>>> a + b
array([[3, 4, 5],
       [6, 7, 8]])
>>> a - b
array([[-1,  0,  1],
       [ 2,  3,  4]])
>>> a * b
array([[ 2,  4,  6],
       [ 8, 10, 12]])
>>> sqrt(a)
array([[ 1.        ,  1.41421356,
1.73205081],
       [ 2.        ,  2.23606798,
2.44948974]])
```

Biopython

Tools for computational molecular biology

- Parsers for various file formats (BLAST Clustalw, FASTA, GEBank, etc).
- Access to online service (NCBI, and ExPASy)
- Interfaces to common programs (Blast, Clustalw, etc)
- And more

Biopython:Sequence (Transcription)

```
>>> from Bio.Seq import Seq

>>> from Bio.Alphabet import IUPAC

>>> coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG", IUPAC.unambiguous_dna)


>>>coding_dna.transcribe()

Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPACUnambiguousRNA())

>>>help(Seq)
```

Biopython: Sequence (Translation)

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG", IUPAC.unambiguous_dna)

>>> coding_dna.translate()
Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
```

For Further References

- Python Tutorial: https://docs.python.org/3/tutorial/
- Matplotlib Beginner's Guide: http://matplotlib.org/users/beginner.html
- Numpy Quickstart Tutorial: https://docs.scipy.org/doc/numpy-dev/user/quickstart.html
- Biopython Tutorial and Cookbook: http://biopython.org/DIST/docs/tutorial/Tutorial.html