

Introduction to Programming with Python - Day 4

`vipin@cbio.mskcc.org`

Rätsch Laboratory, Computational Biology Center
Memorial Sloan Kettering Cancer Center

13 November 2014

Python for Scientific Computing

SciPy is a Python-based ecosystem of open-source software for science.

Python for Scientific Computing

SciPy is a Python-based ecosystem of open-source software for science.

NumPy - Powerful N-dimensional array object

Python for Scientific Computing

SciPy is a Python-based ecosystem of open-source software for science.

NumPy - Powerful N-dimensional array object

Matplotlib - 2D plotting library

Python for Scientific Computing

SciPy is a Python-based ecosystem of open-source software for science.

NumPy - Powerful N-dimensional array object

Matplotlib - 2D plotting library

IPython - Interactive python console

Python for Scientific Computing

SciPy is a Python-based ecosystem of open-source software for science.

NumPy - Powerful N-dimensional array object

Matplotlib - 2D plotting library

IPython - Interactive python console

pandas - Data structures

NumPy - multidimensional data arrays

The **numpy** package (module) is used in almost all numerical computation using Python.

NumPy - multidimensional data arrays

The **numpy** package (module) is used in almost all numerical computation using Python.

Provide high-performance vector, matrix and higher-dimensional data structures for Python.

NumPy - multidimensional data arrays

The **numpy** package (module) is used in almost all numerical computation using Python.

Provide high-performance vector, matrix and higher-dimensional data structures for Python.

In the **numpy** package the terminology used for vectors, matrices and higher-dimensional data sets is *array*.

NumPy - multidimensional data arrays

The **numpy** package (module) is used in almost all numerical computation using Python.

Provide high-performance vector, matrix and higher-dimensional data structures for Python.

In the **numpy** package the terminology used for vectors, matrices and higher-dimensional data sets is *array*.

```
1 from numpy import *
```

NumPy - multidimensional data arrays

The **numpy** package (module) is used in almost all numerical computation using Python.

Provide high-performance vector, matrix and higher-dimensional data structures for Python.

In the **numpy** package the terminology used for vectors, matrices and higher-dimensional data sets is *array*.

```
1 from numpy import *
```

The recommended convention to import numpy is:

NumPy - multidimensional data arrays

The **numpy** package (module) is used in almost all numerical computation using Python.

Provide high-performance vector, matrix and higher-dimensional data structures for Python.

In the **numpy** package the terminology used for vectors, matrices and higher-dimensional data sets is *array*.

```
1 from numpy import *
```

The recommended convention to import numpy is:

```
1 import numpy as np
```

NumPy - manual construction of arrays

1-Dimension:

```
1 xv = np.array([0, 1, 2, 3]) #Python list
2
3 xv.ndim #1
4
5 xv.shape #(4,)
6
7 len(xv) #4
```

NumPy - manual construction of arrays

1-Dimension:

```
1 xv = np.array([0, 1, 2, 3]) #Python list
2
3 xv.ndim #1
4
5 xv.shape #(4,)
6
7 len(xv) #4
```

2-Dimension:

```
1 xm = np.array([[0, 1, 2], [3, 4, 5]]) #a nested Python
    list
2
3 xm.ndim #2
4
5 xm.shape #(2,3)
6
7 len(xm) #2
```

NumPy - basic data types

Different data types allows to store data efficiently.

NumPy - basic data types

Different data types allows to store data efficiently.

```
1 xv.dtype #dtype('int64')
```


NumPy - basic data types

Different data types allows to store data efficiently.

```
1 xv.dtype #dtype('int64')
```

ValueError - assign a value of the wrong type to an element

```
1 xv[0] = 'hello' #ValueError
```

NumPy - basic data types

Different data types allows to store data efficiently.

```
1 xv.dtype #dtype('int64')
```

ValueError - assign a value of the wrong type to an element

```
1 xv[0] = 'hello' #ValueError
```

Other data types:

- Complex

- Bool

- String

NumPy - basic data types

Different data types allows to store data efficiently.

```
1 xv.dtype #dtype('int64')
```

ValueError - assign a value of the wrong type to an element

```
1 xv[0] = 'hello' #ValueError
```

Other data types:

Complex

Bool

String

```
1 xc = np.array([[1, 2], [3, 4]], dtype=complex)
```

NumPy - function for creating arrays

arange

```
1 xp = np.arange(10) #0 .. n-1
2
3 xq = np.arange(1, 9, 2) #arguments: start, stop (
    exclusive) step
```

NumPy - function for creating arrays

arange

```
1 xp = np.arange(10) #0 .. n-1
2
3 xq = np.arange(1, 9, 2) #arguments: start, stop (
    exclusive) step
```

linspace

```
1 xl = np.linspace(0, 1, 6) # start, end, num-points
```

NumPy - function for creating arrays

arange

```
1 xp = np.arange(10) #0 .. n-1
2
3 xq = np.arange(1, 9, 2) #arguments: start, stop (
    exclusive) step
```

linspace

```
1 xl = np.linspace(0, 1, 6) # start, end, num-points
```

random

```
1 xu = np.random.rand(4) #uniform in [0, 1]
2
3 xu = np.random.rand(2,2)
```

NumPy - function for creating arrays

arange

```
1 xp = np.arange(10) #0 .. n-1
2
3 xq = np.arange(1, 9, 2) #arguments: start, stop (
    exclusive) step
```

linspace

```
1 xl = np.linspace(0, 1, 6) # start, end, num-points
```

random

```
1 xu = np.random.rand(4) #uniform in [0, 1]
2
3 xu = np.random.rand(2,2)
```

```
1 xg = np.random.randn(4) # standard normal distributed
    random numbers
2
3 xg = np.random.rand(2,2)
```

NumPy - common arrays

zeros

```
1 xz = np.zeros((2, 2)) # (2,2) is a tuple!
```


NumPy - common arrays

zeros

```
1 xz = np.zeros((2, 2)) # (2,2) is a tuple!
```

ones

```
1 xo = np.ones((3, 3))
```

NumPy - common arrays

zeros

```
1 xz = np.zeros((2, 2)) # (2,2) is a tuple!
```

ones

```
1 xo = np.ones((3, 3))
```

diag

```
1 # a diagonal matrix  
2 xd = np.diag(np.array([1, 2, 3, 4]))
```

NumPy - common arrays

zeros

```
1 xz = np.zeros((2, 2)) # (2,2) is a tuple!
```

ones

```
1 xo = np.ones((3, 3))
```

diag

```
1 # a diagonal matrix  
2 xd = np.diag(np.array([1, 2, 3, 4]))
```

eye

```
1 # a identity matrix  
2 xe = np.eye(3)
```

NumPy - common arrays

zeros

```
1 xz = np.zeros((2, 2)) # (2,2) is a tuple!
```

ones

```
1 xo = np.ones((3, 3))
```

diag

```
1 # a diagonal matrix
2 xd = np.diag(np.array([1, 2, 3, 4]))
```

eye

```
1 # a identity matrix
2 xe = np.eye(3)
```

More details:

http://wiki.scipy.org/Tentative_NumPy_Tutorial

NumPy - indexing

Index elements in an array using the square bracket and indices.

```
1 # xv has only one dimension , taking one index  
2 xv[0]
```

NumPy - indexing

Index elements in an array using the square bracket and indices.

```
1 # xv has only one dimension , taking one index  
2 xv[0]
```

```
1 # xm is a 2 dimensional array , taking two indices  
2 xm[1 ,1]
```

NumPy - indexing

Index elements in an array using the square bracket and indices.

```
1 # xv has only one dimension , taking one index
2 xv[0]
```

```
1 # xm is a 2 dimensional array , taking two indices
2 xm[1,1]
```

If we omit an index of a multidimensional array it returns the whole row

```
1 xm[1] #array ([]) a N-1 dimensional array
2
3 xm[1,:] #row 1 same as the above line
4
5 xm[:,1] #accessing the column 1
```

NumPy - indexing

Index elements in an array using the square bracket and indices.

```
1 # xv has only one dimension , taking one index
2 xv[0]
```

```
1 # xm is a 2 dimensional array , taking two indices
2 xm[1,1]
```

If we omit an index of a multidimensional array it returns the whole row

```
1 xm[1] #array([]) a N-1 dimensional array
2
3 xm[1,:] #row 1 same as the above line
4
5 xm[:,1] #accessing the column 1
```

Assigning new values to elements in an array using indexing

```
1 xm[0,0] = 1 #new value
2
3 xm[1,:] = 0 # row 1
4
5 xm[:,2] = -1 # column 2
```


NumPy - slicing

Index slicing syntax **`xm[lower:upper:step]`** to extract part of an array.

```
1 xs = np.array([1,2,3,4,5])  
2  
3 xs[1:3]
```

NumPy - slicing

Index slicing syntax **xm[lower:upper:step]** to extract part of an array.

```
1 xs = np.array([1,2,3,4,5])  
2  
3 xs[1:3]
```

We can omit any of the three parameters in **xm[lower:upper:step]**

```
1 xs[:] # lower, upper, step all take the default values  
2 xs[::2] # step is 2, lower and upper defaults to the  
         # beginning and end of the array  
3 xs[:3] # first three elements  
4 xs[3:] # elements from index 3  
5 xs[-1] # the last element in the array  
6 xs[-3:] # the last three elements
```

NumPy - slicing

Index slicing syntax **xm[lower:upper:step]** to extract part of an array.

```
1 xs = np.array([1,2,3,4,5])
2
3 xs[1:3]
```

We can omit any of the three parameters in **xm[lower:upper:step]**

```
1 xs[:] # lower, upper, step all take the default values
2 xs[:2] # step is 2, lower and upper defaults to the
        # beginning and end of the array
3 xs[:3] # first three elements
4 xs[3:] # elements from index 3
5 xs[-1] # the last element in the array
6 xs[-3:] # the last three elements
```

Index slicing works exactly the same way for multidimensional arrays.

```
1 xm = np.array([[n+m*10 for n in range(5)] for m in range
                (5)])
2
3 xm[1:4, 1:4] # a block from the original array
4
5 xm[:,2, :2] # strides
```

Matplotlib - basic visualization

Matplotlib is a 2D plotting package.

Matplotlib - basic visualization

Matplotlib is a 2D plotting package.

```
1 import matplotlib.pyplot as plt
2
3 x = np.linspace(0, 3, 20)
4 y = np.linspace(0, 9, 20)
5
6 plt.plot(x, y)          # line plot
7 plt.plot(x, y, 'o')     # dot plot
8
9 plt.show()
```

Matplotlib - basic visualization

Matplotlib is a 2D plotting package.

```
1 import matplotlib.pyplot as plt
2
3 x = np.linspace(0, 3, 20)
4 y = np.linspace(0, 9, 20)
5
6 plt.plot(x, y)           # line plot
7 plt.plot(x, y, 'o')     # dot plot
8
9 plt.show()
```

```
1 image = np.random.rand(30, 30)
2 plt.imshow(image, cmap=plt.cm.hot)
3 plt.colorbar()
4 plt.show()
```

Matplotlib - basic visualization

Matplotlib is a 2D plotting package.

```
1 import matplotlib.pyplot as plt
2
3 x = np.linspace(0, 3, 20)
4 y = np.linspace(0, 9, 20)
5
6 plt.plot(x, y)          # line plot
7 plt.plot(x, y, 'o')     # dot plot
8
9 plt.show()
```

```
1 image = np.random.rand(30, 30)
2 plt.imshow(image, cmap=plt.cm.hot)
3 plt.colorbar()
4 plt.show()
```

Reference: <http://matplotlib.org/>

<http://web.stanford.edu/~mwaskom/software/seaborn/>

BioPython - python modules for bioinformatics

A set of free Python modules for working with sequence analysis.

BioPython - python modules for bioinformatics

A set of free Python modules for working with sequence analysis.

- AlignIO
- SearchIO
- BioSQL
- Seq
- SeqIO
- SeqRecord

BioPython - python modules for bioinformatics

A set of free Python modules for working with sequence analysis.

- AlignIO
- SearchIO
- BioSQL
- Seq
- SeqIO
- SeqRecord

Details: <http://biopython.org>

Seq - basic sequence tools

BioPython represents sequences with the **Seq** object.

Seq - basic sequence tools

BioPython represents sequences with the **Seq** object.

```
1
2 from Bio.Seq import Seq
3
4 # working with sequences
5 my_seq = Seq("AGTACACTGGT")
6
7 print my_seq
8
9 print "complement: " + my_seq.complement()
10 print "reverse complement: " + my_seq.reverse_complement
   ()
11 print "transcribe: " + my_seq.transcribe()
12 print "my_seq[2:4]: " + my_seq[2:4]
13
14 my_rna = my_seq.transcribe()
15 my_dna = my_rna.back_transcribe()
```

SeqIO - basic fileIO

parse function can handle GenBank, FastQ, Fasta formats.

SeqIO - basic fileIO

parse function can handle GenBank, FastQ, Fasta formats.

```
1
2 from Bio import SeqIO
3
4 for record in SeqIO.parse("genome.fasta", "fasta"):
5     print record.id
6
7     print record.seq
8     print len(record.seq)
9
10    print record.description
11    break
```

SeqIO - basic fileIO

parse function can handle GenBank, FastQ, Fasta formats.

```
1
2 from Bio import SeqIO
3
4 for record in SeqIO.parse("genome.fasta", "fasta"):
5     print record.id
6
7     print record.seq
8     print len(record.seq)
9
10    print record.description
11    break
```

```
1
2 for record in SeqIO.parse("reads.fq", "fastq"):
3     print record.id
```

SeqRecord - handling sequence records

Hold a sequence (as a Seq object) with identifiers (ID and name).

SeqRecord - handling sequence records

Hold a sequence (as a Seq object) with identifiers (ID and name).

```
1 # importing modules
2 import re
3 from Bio.Seq import Seq
4 from Bio.SeqRecord import SeqRecord
5
6 # doing an operation
7 rna = "AUG"
8 dna = re.sub(r'U', r'T', rna)
9
10 # making seqrecords
11 dna = Seq(dna)
12 fas_rec = SeqRecord(dna, id="codon_1", description="start
           codon")
13
14 # writing to a fasta file
15 fh = open("codon.fa", "w")
16 fh.write(fas_rec.format("fasta"))
17 fh.close()
```

GFF Parsing - parsing genome annotation file

Not yet integrated into the core biopython

<http://github.com/chapmanb/bcbb/tree/master/gff>

GFF Parsing - parsing genome annotation file

Not yet integrated into the core biopython

<http://github.com/chapmanb/bcbb/tree/master/gff>

Collection of different annotation conveter programs

<https://github.com/vipints/GFFtools-GX>

pip or **easy_install** - managing local python packages

Machine learning toolbox in python

SHOGUN - shogun-toolbox.org

scikit-learn - scikit-learn.org

pip or easy_install - managing local python packages

Machine learning toolbox in python

SHOGUN - shogun-toolbox.org

scikit-learn - scikit-learn.org

```
1 # execute the following command in a unix console
2 # creating a path for module installation
3 mkdir -p /Users/student/tmp/lib/python2.7/site-packages/
4
5 # adding the above path to the pythonpath env variable
6 export PYTHONPATH=$PYTHONPATH:/Users/student/tmp/lib/
   python2.7/site-packages/
7
8 # install the machine learning package
9 pip install freeze scikit-learn
10
11 # testing the installation
12 python -c "import sklearn"
13
14 # uninstalling the module
15 pip uninstall scikit-learn
```

cPickle - data persistence

Methods **dump()**, **load()**

cPickle - data persistence

Methods **dump()**, **load()**

```
1 import sys
2 import cPickle
3
4 xa = [ 'mango', 'carrots', 'garlic' ]
5
6 fh = open("python_data.pickle", "wb")
7
8 cPickle.dump(xa, fh, protocol=2)
9 fh.close()
```

cPickle - data persistence

Methods **dump()**, **load()**

```
1 import sys
2 import cPickle
3
4 xa = [ 'mango', 'carrots', 'garlic' ]
5
6 fh = open("python_data.pickle", "wb")
7
8 cPickle.dump(xa, fh, protocol=2)
9 fh.close()
```

```
1 fh = open("python_data.pickle", "rb")
2
3 xb = cPickle.load(fh)
4 fh.close()
5
6 xa == xb # checking are they same #True
```


Django - python web framework

Creating a project: `django-admin.py startproject mysite`

Django - python web framework

Creating a project: `django-admin.py startproject mysite`

Add names to *mysite settings.py* *mysite urls.py*

Django - python web framework

Creating a project: `django-admin.py startproject mysite`

Add names to *mysite settings.py* *mysite urls.py*

`python manage.py startapp app_name`

Django - python web framework

Creating a project: `django-admin.py startproject mysite`

Add names to *mysite settings.py* *mysite urls.py*

`python manage.py startapp app_name`

Edit `view.py`

```
1 from django.http import HttpResponse
2
3 def index(request):
4     """
5     Index page for the site
6     """
7     return HttpResponse("Hello, Python Class.")
8     #return render(request, 'tracks/index.html')
```

Django - python web framework

Creating a project: `django-admin.py startproject mysite`

Add names to *mysite settings.py* *mysite urls.py*

`python manage.py startapp app_name`

Edit `view.py`

```
1 from django.http import HttpResponse
2
3 def index(request):
4     """
5     Index page for the site
6     """
7     return HttpResponse("Hello, Python Class.")
8     #return render(request, 'tracks/index.html')
```

`python manage.py runserver`

Details: <https://www.djangoproject.com/>

Would love to hear your experience!

gabow@cbio.mskcc.org

vipin@cbio.mskcc.org