Programming in Python

Sarath Babu

Session-III

Indian Institute of Space Science and Technology
Thiruvananthapuram, Kerala, India 695547

25th October, 2018

IEEE Student Branch & Computer Science and Engineering Club, IIST

IEEE CSE CLUB

Outline

Introduction to NumPy

- Stands for **Num**erical **Py**thon
- Developed for scientific computing
- Predecessors: *Numeric* and *Numarray*
- Primarily created by **Travis Oliphant** in 2005-06
- Latest stable release is 1.15.2
- Core feature is **ndarrays**, multi-dimensional array with homogeneous data type
- **SciPy** + **Matplotlib** can be a replacement for MATLAB

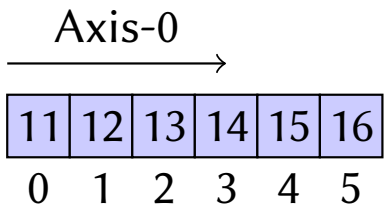
```
>>> import numpy as np
>>> a = np.array(11, 12, 13, 14, 15, 16)
>>> a = np.array([11, 12, 13, 14, 15, 16])
>>> print a
>>> b = np.array([[11, 12], [13, 14], [15, 16]])
>>> print b
>>> print type(a), type(b)
```

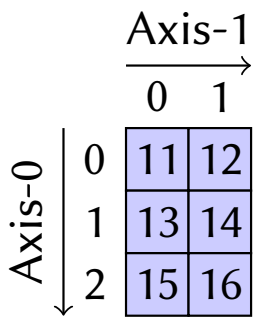
The type *ndarray*

- Represents table of elements of **same type**
- Elements are indexed by integers
- Dimensions are referred as **axes**

```
>>> b = np.array([11, 12, 13, 14, 15, 16])
>>> print b.ndim
>>> print b.shape
>>> print b.dtype

>>> c = np.array([[11, 12], [13, 14], [15, 16]])
>>> print c.ndim
>>> print c.shape
```





Playing with *ndarray*

Empty array

```
>>> a = np.ndarray((2,3), dtype=float)
>>> print a
>>> b = np.empty((2,3), dtype=int)
>>> print b
```

Array initialized with 0s

```
>>> c = np.zeros((2,3), dtype=int)
>>> print c
```

Array initialized with 1s

```
>>> d = np.ones((2,3))
>>> print d
```

Array with range of values

```
>>> e = np.arange(1, 20, 2)
>>> print e
```

Array with *n* elements with equal spacing

```
>>> f = np.linspace(1, 20, 8)
>>> print f
>>> g = np.linspace(1, 20, 8, endpoint=True)
>>> print g
```

Reshaping arrays

```
>>> h = np.arange(0, 12)
>>> i = np.reshape(h, (3, 4))
>>> print i
>>> j = np.reshape(i, (2, 3, 2))
>>> print j
```

Arrays with dimensions of other arrays

```
>>> k = np.empty_like(j)
>>> l = np.zeros_like(j)
>>> m = np.ones_like(j)
```

Arithmetic operations with *ndarrays*

- The operations are primarily carried out **elementwise**

Arithmetic operations

```
>>> a = np.arange(1, 11)
>>> b = np.arange(11, 21)
>>> c = np.arange(21, 26)
>>> print a + b
>>> print a + c
>>> print a + 5
>>> print a * b
>>> print a - b
>>> print a / b
>>> print a ** 2
>>> print a < 5
```

Functions on *ndarrays*

```
>>> a = np.arange(1, 13).reshape((3,4))
>>> print a.sum()
>>> print a.min()
>>> print a.max()
>>> print a.argmax(axis=0)
>>> print a.argmin(axis=0)
>>> print a.sum(axis=0) # Sum along Axis-0
>>> print a.sum(axis=1) # Sum along Axis-1
```

Other NumPy functions

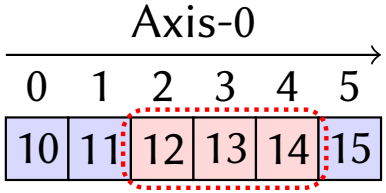
```
>>> x = np.linspace(0, 2*np.pi, 100,
endpoint=True)
>>> y = np.sin(x)
>>> z = np.exp(x)
>>> print np.sqrt(a)
>>> p = np.arange(1,7)
>>> print np.mean(p)
>>> print np.std(p)
>>> a = np.array([1, 2, 3])
>>> b = np.array([4, 5, 6])
>>> print np.dot(a, b)
>>> print np.cross(a, b)
```

Slicing and shaping *ndarrays*

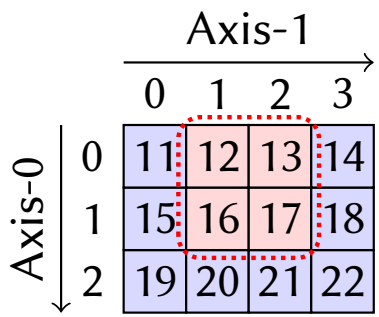
- Slicing attributes need to be provided for **each axis**
- Return **views**

Slicing

```
>>> a = np.arange(11, 16)
>>> b = a[2:5] # Returns view
```




```
>>> b[0] = 500
>>> print b
>>> print a
>>> c = np.arange(11, 23).reshape((3,4))
>>> d = c[0:2, 1:3] # Returns view
>>> print d
```




```
>>> print c[-1]
>>> p = np.arange(11, 71).reshape((3,4,5))
>>> q = p[1:, 2:4, 1:4] # Returns view
>>> print q
```

Shaping arrays

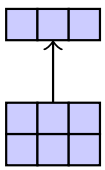
```
>>> a = np.arange(11, 23)
>>> print a
>>> b = a.reshape((3,4)) # Returns view
>>> print b
>>> c = a.ravel() # View in 1-dimension
>>> print c
>>> a.resize((2, 6)) # Modifies a itself
>>> print a
>>> d = a.T # Transpose as view
>>> print d
```

Stacking and splitting *ndarrays*

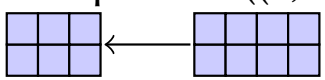
- Dimensions need to be in order for stacking ndarrays

Stacking

```
>>> a = np.arange(11, 17).reshape((2, 3))
>>> b = np.arange(21, 27).reshape((3, 2))
>>> c = a.T
>>> d = b.T
>>> e = np.vstack((a, b))
>>> f = np.vstack((a, d))
```

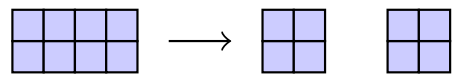


```
>>> g = np.hstack((b, a))
>>> h = np.hstack((a, d))
```

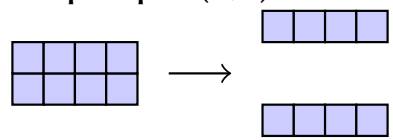


Splitting

```
>>> a = np.arange(11,35).reshape((4,6))
>>> b = np.hsplit(a, 4) # Returns view
>>> c = np.hsplit(a, 2) # Returns view
```



```
>>> d = np.vsplit(a,2) # Returns view
```



Split at specific points

```
>>> print np.hsplit(a, (3, 5)) # Returns view
>>> print np.vsplit(a, (1, 3)) # Returns view
```

File I/O with NumPy

- Arrays can be read from and written to files
- Two methods
  1. Using ***.npy*** extension for *platform independent* data storage

```
>>> a = np.arange(11, 23).reshape((3,4))
>>> np.save('datafile', a)
>>> b = np.load('datafile.npy')
```

  2. Using text files

```
>>> a = np.arange(11, 23).reshape((3,4))
>>> np.savetxt('datafile.txt', a, delimiter=' , ')
>>> b = np.loadtxt('datafile.txt', delimiter=' , ')
```

Matrix in NumPy

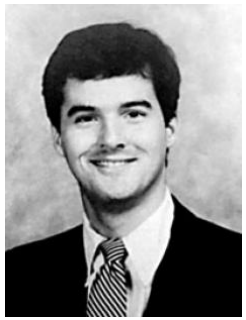
- Data arranged in rows and columns
- NumPy uses the type ***matrix*** to represent matrices

```
>>> a = np.matrix('1, 2, 3; 4, 5, 6')
>>> b = np.array([[1,0], [2,3], [4,1]])
>>> c = np.matrix(b)
>>> print b[1,1]
>>> d = a * c
>>> print d
>>> e = a.T # Returns view
>>> p = np.matrix('3, 6; 1, 4')
>>> q = p.I # Multiplicative inverse
>>> s = np.matrix(np.random.random((2,3)))
>>> print s.max()
>>> print s.max(0)
>>> print s.max(1)
>>> print s.argmax(0)
>>> print s.argmax(1)
```

For more functions: *https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.matrix.html*

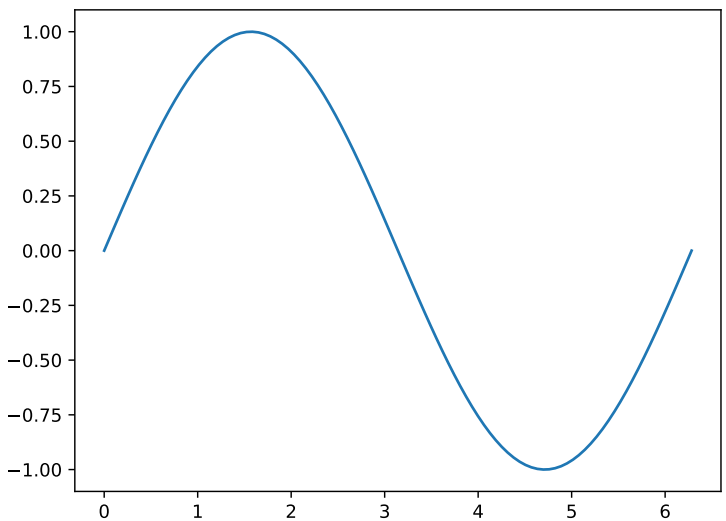
Introduction to Matplotlib

- Library for plotting graphs
- Developed by **John D. Hunter**
- Equipped with GUI using Pyplot



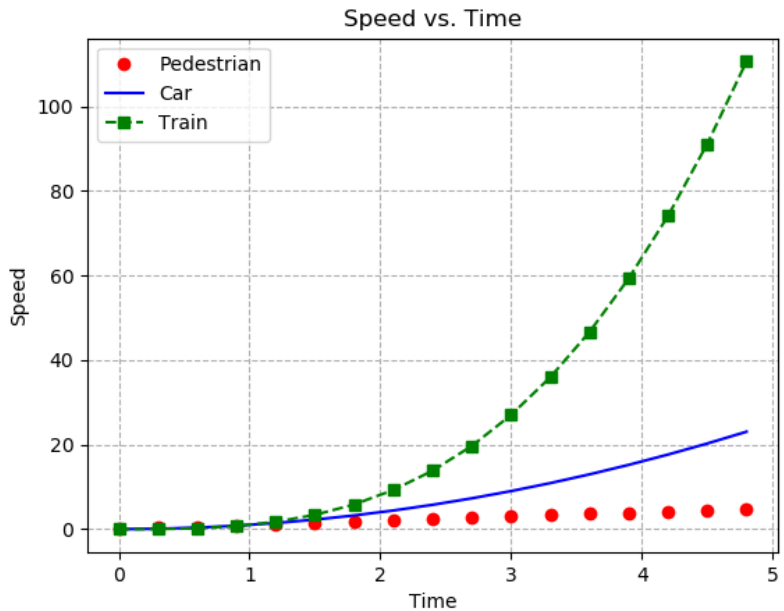
John D. Hunter
(1968–2012)

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> x = np.linspace(0, 2*np.pi, 100)
>>> y = np.sin(x)
>>> plt.plot(x, y)
>>> plt.show()
```



Matplotlib: Plot with points and lines

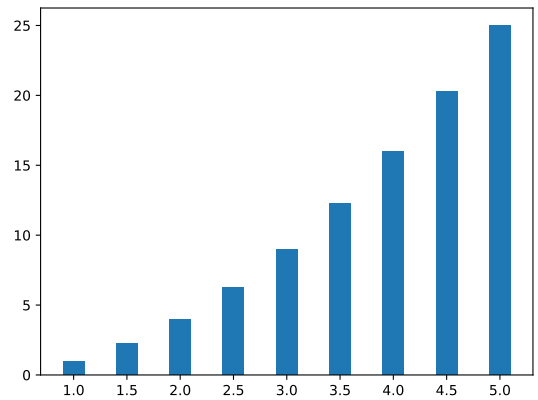
```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> x = np.arange(0, 5.0, 0.3)
>>> y1 = x
>>> y2 = x ** 2
>>> y3 = x ** 3
>>> plt.title('Speed vs. Time')
>>> plt.xlabel('Time')
>>> plt.ylabel('Speed')
>>> plt.plot(x, y1, 'ro', label='Pedestrian')
>>> plt.plot(x, y2, 'b-', label='Car')
>>> plt.plot(x, y3, 'gs--', label='Train')
>>> plt.grid(linestyle='--')
>>> plt.legend()
>>> plt.show()
```



Matplotlib: Bar charts

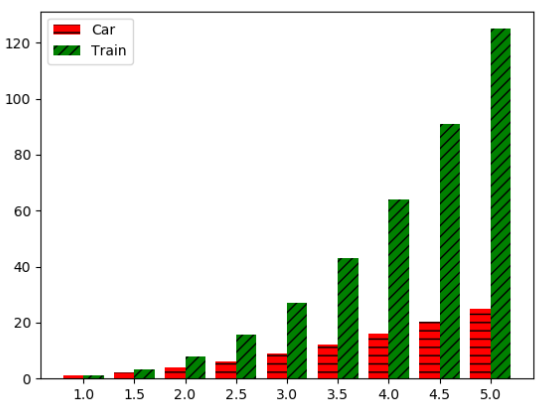
With single y value

```
>>> x = np.arange(1, 5.1, 0.5)
>>> y = x ** 2
>>> plt.bar(x, y, width=0.2)
>>> plt.show()
```



With multiple y values

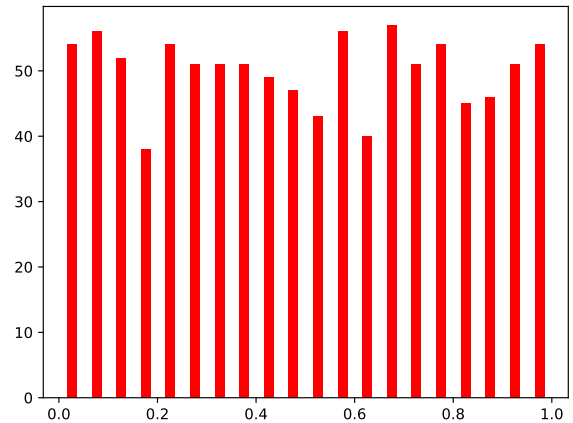
```
>>> x = np.arange(1, 5.1, 0.5)
>>> y1 = x ** 2
>>> y2 = x ** 3
>>> plt.bar(x-0.1, y1, color='r', width=0.2,
label='Car', hatch='-')
>>> plt.bar(x+0.1, y2, color='g', width=0.2,
label='Train', hatch='///')
>>> plt.legend()
>>> plt.show()
```



Matplotlib: Histograms

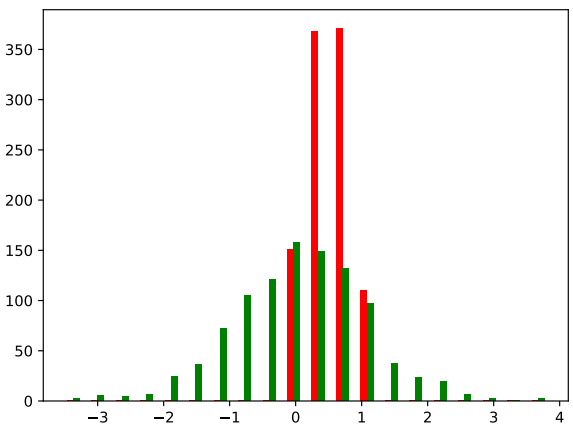
Single set of samples

```
>>> y = np.random.random(1000)
>>> plt.hist(y, bins=20, color='r', rwidth=0.4)
>>> plt.show()
```



Multiple set of samples

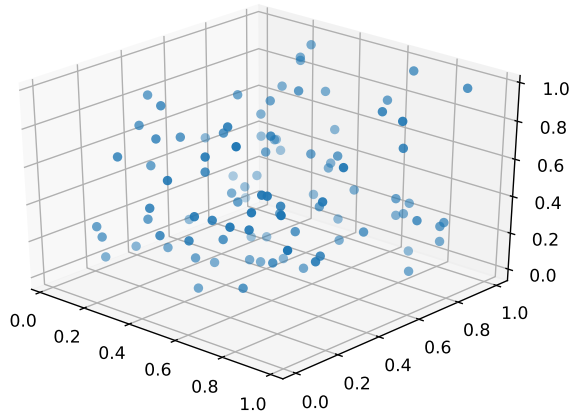
```
>>> y1 = np.random.random(1000)
>>> y2 = np.random.randn(1000)
>>> plt.hist([y1,y2], bins=20, color=['r', 'g'],
histtype='bar', rwidth=0.4)
>>> plt.show()
```



Matplotlib: Plotting in 3D environment

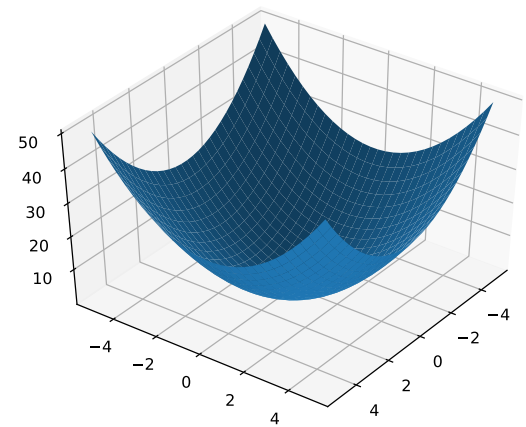
3D Scatter plot

```
>>> import matplotlib.pyplot as plt
>>> from mpl_toolkits import mplot3d
>>> import numpy as np
>>> x = np.random.random(100)
>>> y = np.random.random(100)
>>> z = np.random.random(100)
>>> ax = plt.axes(projection='3d')
>>> ax.scatter3D(x, y, z)
>>> plt.show()
```



3D Surface plot

```
>>> def findZ(x, y):
        return x ** 2 + y ** 2
>>> x = np.linspace(-5,5,30)
>>> y = np.linspace(-5,5,30)
>>> X, Y = np.meshgrid(x, y)
>>> Z = findZ(X, Y)
>>> ax = plt.axes(projection='3d')
>>> ax.plot_surface(X, Y, Z)
>>> plt.show()
```

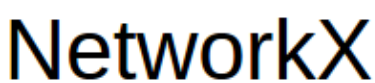


Useful libraries

1. **math:** For operations in mathematics
2. **string:** String operations
3. **Tkinter:** Graphical User Interface (GUI) toolkit
4. **sys:** Interpreter related modules
5. **os:** Operating system specific functions
6. **socket:** Network programming

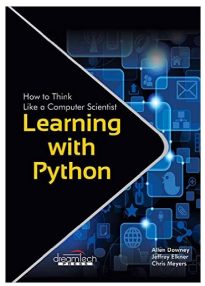
Useful libraries

1. **math:** For operations in mathematics
2. **string:** String operations
3. **Tkinter:** Graphical User Interface (GUI) toolkit
4. **sys:** Interpreter related modules
5. **os:** Operating system specific functions
6. **socket:** Network programming



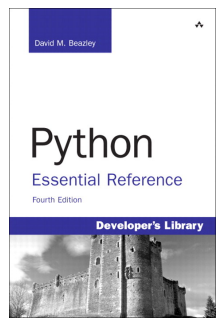
Learning materials

Official Python documentation is available at https://docs.python.org



"How to Think Like a Computer Scientist: Learning with Python"
- Allen Downey, Jeffrey Elkner, and Chris Meyers

"Python Essential Reference: Developer's Library" - David Beazley



"Dive into Python" - Mark Pilgrim

"A Byte of Python" - Swaroop C. H.

"Learn Python" - tutorialspoint

"Programs must be written for people to read, and only incidentally for machines to execute." – **Harold Abelson**

Questions?

sarath.babu.2014@ieee.org

Thank you.