Python – D6S1

Contents

Python Libraries

Python Libraries

Python Standard Library

- The library also contains built-in functions and exceptions — objects that can be used by all Python code without the need of an import statement.
- Built-in Functions
- abs() dict() help() min() setattr() all() dir() hex() next() slice() any() divmod() id() object() sorted() ascii() enumerate() input() oct() staticmethod() bin() eval() int() open() str() bool() exec() isinstance() ord() sum() bytearray() filter() issubclass() pow() super() bytes() float() iter() print() tuple() callable() format() len() property() type() chr() frozenset() list() range() vars() classmethod() getattr() locals() repr() zip() compile() globals() map() reversed() __import__() complex() hasattr() max() round() delattr() hash() memoryview() set()
- Let's discuss some of them:
- abs(*x*) -- Return the absolute value of a number. The argument may be an integer or a floating point number.

Python Libraries

Python Standard Library

- enumerate(*iterable*, *start=0*) -- Return an enumerate object. *iterable* must be a sequence, an iterator, or some other object which supports iteration.

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
```

- range(*start*, *stop*[, *step*]) -- Rather than being a function, range is actually an immutable sequence type

- reversed(*seq*) Return a reverse iterator. *seq* must be an object which has a __reversed__() method or supports the sequence protocol (the __len__() method and the __getitem__() method with integer arguments starting at 0).

- For further details, consult Documentation.

Python Libraries

Numeric and Mathematical Modules

- There are lot's of functions that are available, like factorial(x), exp(x), log(x), pow(x,y), sqrt(x), cos(x), sin(x), etc.
- The decimal module provides support for fast correctly-rounded decimal floating point arithmetic. It offers several advantages over the float datatype.
- The 'random' module implements pseudo-random number generators for various distributions.

```
>>> random.random()          # Random float x, 0.0 <= x < 1.0
0.37444887175646646
>>> random.randrange(10)     # Integer from 0 to 9
7
>>> items = [1, 2, 3, 4, 5, 6, 7]
>>> random.shuffle(items)
>>> items
[7, 3, 2, 5, 6, 4, 1]
```

Python Libraries

Statistical Functions

- There are quite a few functions that are available, like mean(), median(), mode(), variance(), stdev() etc.

Functional Programming Modules

- These are a set of functions under 'itertools', functools, 'operator'.
- Please refer Documentation for the above.

File and Directory Access

- Though a lot of functions are there , one quite useful are categorized under os.path.
- Some of them are : abspath(path) , commonpath(paths), dirname(path), join(paths, *paths) [-- This joins one or more paths intelligently], split(path), etc.
- File related – like : filecmp.cmp(f1, f2, shallow=True) – Compare the files named *f1* and *f2*, returning True if they seem equal, False otherwise.

Python Libraries

Some other broad groups are :

- Data Compression and Archiving
- File Formats [here CSV and other types can be handled]
- Cryptographic Services
- Generic Operating System Services [misc. os system interfaces]
- Concurrent Execution [sub-process, thread]
- InterProcess Communication and Networking [socket, ssl, signal, asyncio]
- Internet Data Handling [email, json, mailbox]
- Structured Markup Processing Tools [html, xml]
- Internet Protocols and Support [webbrowser, cgi, urllib, http, telnetlib]

- This is not a Comprehensive list – please refer Python 3.5 Documentation.

Other Interesting Python Modules

- NumPy - **NumPy** is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

- SciPy -- **SciPy** (pronounced "Sigh Pie") is an open source Python library used by scientists, analysts, and engineers doing scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

- matplotlib -- **matplotlib** is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like wxPython, Qt, or GTK+.

Other Interesting Python Modules

- BeautifulSoup -- **Beautiful Soup** is a Python package for parsing HTML and XML documents (including having malformed markup, i.e. non-closed tags, so named after Tag soup). It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping.

```
# anchor extraction from html document
from bs4 import BeautifulSoup
import urllib.request, urllib.error, urllib.parse
webpage = urllib2.request.urlopen('http://en.wikipedia.org/wiki/Main_Page')
soup = BeautifulSoup(webpage)
for anchor in soup.find_all('a'):
    print(anchor.get('href', '/'))
```

Other Interesting Python Modules

- BeautifulSoup
- Consider another example :

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
html = urlopen("http://www.pythonscraping.com/exercises/exercise1.html")
bsObj = BeautifulSoup(html.read());
print(bsObj.h1)
```

- The output is :

```
<h1>An Interesting Title</h1>
```

The <h1> tag got extracted from the web page.

Other Interesting Python Modules

Requests

- Requests ranks #1 in the list of most popular and useful libraries for Python development.
- This library is used in projects of Spotify, Microsoft, NPR, Heroku, Amazon, BuzzFeed, Reddit, Twitter, Mozzila, and the list is very long.
- Requests takes off the manual labor work and automates the task of adding query strings to URLs, form encoding of post data, multipart files, keep alive and Http connection pooling etc. by utilizing urllib3.
- Automatic content decoding is also provided to cut down the development efforts.
- It offers loads of other features like digest authentication, elegant/value cookies, Unicode response bodies, streaming downloads, connection timeouts and so on.
- In short, Requests is a good to have library for any web project.

Other Interesting Python Modules

Simplejson

- Simplejson is another library in the wild which is highly popular and happens to be one of the most downloaded python library.
- Simplejson is fast, correct and extensible json encoder and decoder, and supports Python 3.3+ with backward compatibility to Python 2.5.
- Simplejson is written in python and has no external dependencies.
- It also packages an extension written in C for lightning fast performance.
- Apart from the default UTF-8, Simplejson decoder can handle JSON strings coming in with any encoding.

Other Interesting Python Modules

SQL Alchemy

- Databases are integral to application development, and in the world of Python, SQLAlchemy is the go to library for working with databases.
- It acts as SQL toolkit and Object Relational mapper by offering full suite of enterprise persistent patterns for high performance database access.
- The cool thing is the data mapper pattern, in which different classes are mapped to the database in different ways, it allows the object model and database schema to be decoupled from the start itself.
- Developers get full control and visibility of the SQL construction, nothing is hidden behind the walls of wrappers.
- The approach of this library is far more efficient and modern when compared to other SQL / ORM tools available out there in the wild, and for that reason, it ranks much higher in our list of most useful Python libraries.
- With the recent release, just a couple of months back, SQLAlchemy is far ahead in the game and is used by organizations like Freshbooks, Survey Monkey, Mozilla, reddit, Yelp and many more.

Other Interesting Python Modules

TensorFlow

- TensorFlow goes beyond the basics and takes you to the world of machine intelligence.
- This is an open source python library that is useful for doing calculations using data flow graphs.
- The computation is initially represented in the form of graphs with every node of the graph meant to do some mathematical operation.
- The actual computation, however, is done on demand, which allows for great performance boost in complex calculations.
- The library caters to the need of complex computations by catering to distributed computing across CPUs/GPUs, and multiple systems while taking care of redundancy.
- TensorFlow is available as open source library and free to use, it was originally developed by Google's engineers working on Google Brain project.

Other Interesting Python Modules

Scikit-learn

- Scikit-learn is your high-level machine learning library and provides off the shelf algorithms like random forests, ready to be used in machine learning projects.
- Scikit-learn is written mainly in Python but it also uses Cython for performance enhancements in some of the core algorithms.
- Cython wrapper around LIBSVM is used for support vector implementation and LIBLINEAR for logistical regression and linear support vector machines.
- Scikit-learn also makes use of CBLAS which is a C interface to utilize Basic Linear Algebra Subprograms (CBLAS) library.
- This library is built on top of SciPy and distributed under the 3-Clause BSD license for open source, for research as well as for commercial use, and is one of the best python libraries.

Other Interesting Python Modules

wxPython

- wxPython is one of the three popular GUI libraries for Python, the top three names are PyQT, Tkinter and wxPython.
- wxPython is intuitive to python developers and simple to use, it is a perfect blend of C++ wxWidgets with Python programming.
- wxPython is implemented as a Python extension module and is a cross-platform toolkit that runs on different platforms without the need for modification.
- It is supported on many platforms including Unix distributions, Macintosh OS X and Microsoft Windows (32 bit).
- wxPython offers tons of features that allow to create robust and functional application GUIs in Python with ease.

Other Interesting Python Modules

- Twisted -- **Twisted** is an event-driven network programming framework written in Python and licensed under the MIT License. Twisted projects variously support TCP, UDP, SSL/TLS, IP multicast, Unix domain sockets, a large number of protocols (including HTTP, XMPP, NNTP, IMAP, SSH, IRC, FTP, and others), and much more.

- Scrapy – **Scrapy** is a free and open source web crawling framework, written in Python. Originally designed for web scraping, it can also be used to extract data using APIs or as a general purpose web crawler.

- IPython -- **IPython** is a command shell for interactive computing in multiple programming languages, originally developed for the Python programming language, that offers introspection, rich media, shell syntax, tab completion, and history. IPython provides the following features like Interactive shells (terminal and Qt-based), A browser-based notebook with support for code, text, mathematical expressions, inline plots and other media.

Other Interesting Python Modules

- nltk -- The **Natural Language Toolkit**, or more commonly **NLTK**, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for the Python programming language. NLTK includes graphical demonstrations and sample data. NLTK is intended to support research and teaching in NLP or closely related areas, including empirical linguistics, cognitive science, artificial intelligence, information retrieval, and machine learning.

- Let's try to see an example : [next page]

Other Interesting Python Modules

Nltk example to tokenize a text into sentences and words.

>>> from nltk import sent_tokenize, word_tokenize

>>> text = "Machine learning is the science of getting computers to act without being explicitly programmed. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome. Machine learning is so pervasive today that you probably use it dozens of times a day without knowing it. Many researchers also think it is the best way to make progress towards human-level AI."

>>> sents = sent_tokenize(text)

>>> sents

['Machine learning is the science of getting computers to act without being explicitly programmed.', 'In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome.', 'Machine learning is so pervasive today that you probably use it dozens of times a day without knowing it.', 'Many researchers also think it is the best way to make progress towards human-level AI.']

>>> len(sents)

4

Other Interesting Python Modules

Nltk example to tokenize a text into sentences and words.

```
>>>  tokens = word_tokenize(text)
>>> tokens
['Machine', 'learning', 'is', 'the', 'science', 'of', 'getting', 'computers', 'to', 'act', 'without', 'being', 'explicitly', 'programmed', '.', 'In', 'the', 'past', 'decade', ',', 'machine', 'learning', 'has', 'given', 'us', 'self-driving', 'cars', ',', 'practical', 'speech', 'recognition', ',', 'effective', 'web', 'search', ',', 'and', 'a', 'vastly', 'improved', 'understanding', 'of', 'the', 'human', 'genome', '.', 'Machine', 'learning', 'is', 'so', 'pervasive', 'today', 'that', 'you', 'probably', 'use', 'it', 'dozens', 'of', 'times', 'a', 'day', 'without', 'knowing', 'it', '.', 'Many', 'researchers', 'also', 'think', 'it', 'is', 'the', 'best', 'way', 'to', 'make', 'progress', 'towards', 'human-level', 'AI', '.']
>>> len(tokens)
82
```

Other Interesting Python Modules

- Pandas -- is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.
- It has several very useful features :
- DataFrame object for data manipulation with integrated indexing
- Tools for reading and writing data between in-memory data structures and different file formats
- Data alignment and integrated handling of missing data
- Reshaping and pivoting of data sets
- Label-based slicing, fancy indexing, and subsetting of large data sets
- Data structure column insertion and deletion
- Group by engine allowing split-apply-combine operations on data sets
- Data set merging and joining

OK….. So that was just a Glimpse of some Python Modules/Libraries. There are Lots of them actually. Use net to search and find out.

Other Interesting Python Modules

- Example program using Pandas:
- Let's assume a CSV file named "msft.csv", which contains a snapshot of Yahoo Finance data for MSFT Ticker.
- The format looks like this:

Date, Open, High, Low, Close, Volume, Adj Close

2014-07-21, 83.46, 83.53, 81.81, 81.93, 2359300, 81.93

2014-07-18, 83.30, 83.40, 82.52, 83.35, 4020800, 83.35

2014-07-17, 84.35, 84.63, 83.33, 83.63, 1974000, 83.63

2014-07-16, 83.77, 84.91, 83.66, 84.91, 1755600, 84.91

- We can simply read this data into panda's Dataframe as:

```
# read in msft.csv into a DataFrame
msft = pd.read_csv("msft.csv")
msft.head()
```

- It will show you the first few lines.

Other Interesting Python Modules

- You can alternatively read as below :

```
import pandas.io.data as web
import datetime
# start end end dates
start = datetime.datetime(2012, 1, 1)
end = datetime.datetime(2014, 1, 27)
# read the MSFT stock data from yahoo! and view the head
yahoo = web.DataReader('MSFT', 'yahoo', start, end)
yahoo.head()
```

- Instead of Yahoo, you can read the data for Google as well:

```
goog = web.DataReader('MSFT', 'google', start, end)
```

Other Interesting Python Modules

- Let's see another use of pandas.
- Consider a Loan File structure as [filename : train.csv;; no of records = 615]

Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
LP001002	Male	No	0	Graduate	No	5849	0		360	1	Urban	Y
LP001003	Male	Yes	1	Graduate	No	4583	1508	128	360	1	Rural	N
LP001005	Male	Yes	0	Graduate	Yes	3000	0	66	360	1	Urban	Y
LP001006	Male	Yes	0	Not Graduate	No	2583	2358	120	360	1	Urban	Y
LP001008	Male	No	0	Graduate	No	6000	0	141	360	1	Urban	Y
LP001011	Male	Yes	2	Graduate	Yes	5417	4196	267	360	1	Urban	Y
LP001013	Male	Yes	0	Not Graduate	No	2333	1516	95	360	1	Urban	Y
LP001014	Male	Yes	3+	Graduate	No	3036	2504	158	360	0	Semiurban	N
LP001018	Male	Yes	2	Graduate	No	4006	1526	168	360	1	Urban	Y
LP001020	Male	Yes	1	Graduate	No	12841	10968	349	360	1	Semiurban	N
LP001024	Male	Yes	2	Graduate	No	3200	700	70	360	1	Urban	Y
LP001027	Male	Yes	2	Graduate		2500	1840	109	360	1	Urban	Y
LP001028	Male	Yes	2	Graduate	No	3073	8106	200	360	1	Urban	Y
LP001029	Male	No	0	Graduate	No	1853	2840	114	360	1	Rural	N
LP001030	Male	Yes	2	Graduate	No	1299	1086	17	120	1	Urban	Y
LP001032	Male	No	0	Graduate	No	4950	0	125	360	1	Urban	Y
LP001034	Male	No	1	Not Graduate	No	3596	0	100	240		Urban	Y
LP001036	Female	No	0	Graduate	No	3510	0	76	360	0	Urban	N
LP001038	Male	Yes	0	Not Graduate	No	4887	0	133	360	1	Rural	N

Other Interesting Python Modules

- Now we can do the following :

```
import pandas as pd
data = pd.read_csv("train.csv", index_col="Loan_ID")
# list of all females who are not graduate and got a loan
data.loc[(data["Gender"]=="Female") & (data["Education"]=="Not Graduate") & (data["Loan_Status"]=="Y"), ["Gender","Education","Loan_Status"]]

# now to find out all missing values in each row and column
#Create a new function
def num_missing(x):
        return sum(x.isnull())
#Applying per column
print "Missing values per column:"
print data.apply(num_missing, axis=0) #axis=0 defines that function is to be applied on each column
#Applying per row:
print "\nMissing values per row:"
print data.apply(num_missing, axis=1).head() #axis=1 defines that function is to be applied on each column
```

Other Interesting Python Modules

- The output will look like :

```
Missing values per column:
Gender                13
Married                3
Dependents            15
Education              0
Self_Employed         32
ApplicantIncome        0
CoapplicantIncome      0
LoanAmount            22
Loan_Amount_Term      14
Credit_History        50
Property_Area          0
Loan_Status            0
dtype: int64

Missing values per row:
Loan_ID
LP001002    1
LP001003    0
LP001005    0
LP001006    0
LP001008    0
dtype: int64
```

Other Interesting Python Modules

- Let's see some simple yet powerful use of nltk:

```
from nltk.corpus import stopwords
stoplist = stopwords.words('english')
text = "This is just a test text"
cleanwordlist = [word for word in text.split() if word not in stoplist ]
print(cleanwordlist)
```

- The output will be :

['This', 'test', 'text']

Numpy Library

Contents

Introduction to numpy and scipy

- NumPy and SciPy are open-source add-on modules to Python that provide common mathematical and numerical routines in pre-compiled, fast functions.
- These are growing into highly mature packages that provide functionality that meets, or perhaps exceeds, that associated with common commercial software like MatLab.
- The basic operations used in scientific programming include arrays, matrices, integra- tion, differential equation solvers, statistics, and much more.
- The NumPy (Numeric Python) package provides basic routines for manipulating large arrays and matrices of numeric data.
- The SciPy (Scientific Python) package extends the functionality of NumPy with a substantial collection of useful algorithms, like minimization, Fourier transformation, regression, and other applied mathematical techniques.

Introduction to numpy and scipy

- NumPy specializes in numerical processing through multi-dimensional arrays, where the arrays allow element-by-element operations.
- If needed, linear algebra formalism can be used without modifying the NumPy arrays beforehand. Moreover, the arrays can be modified in size dynamically. This takes out the worries that usually mire quick programming in other languages. Rather than creating a new array when you want to get rid of certain elements, you can apply a mask to it.
- SciPy is built on the NumPy array framework and takes scientific programming to a whole new level by supplying advanced mathematical functions like integration, ordinary differential equation solvers, special functions, optimizations, and more.
- To list all the functions by name in SciPy would take several pages at minimum.
- When looking at the plethora of SciPy tools, it can sometimes be daunting even to decide which functions are best to use.

Numpy

Numpy array objects

- An array is a memory-efficient container that provides fast numerical operations

```
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

- Creating arrays --- 1-D:

```
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
>>> a.ndim
1
>>> a.shape
(4,)
>>> len(a)
4
```

Numpy array objects

- Creating arrays --- 2-D, 3-D, ….. :

```
>>> b = np.array([[0, 1, 2], [3, 4, 5]])  # 2 or 3 array
>>> b
array([[0, 1, 2], [3, 4, 5]])
>>> b.ndim
2
>>> b.shape
(2, 3)
>>> len(b) # returns the size of the first dimension
2
>>> c = np.array([[[1], [2]], [[3], [4]]])
>>> c
array([[[1], [2]],
              [[3], [4]]])
>>> c.shape
(2, 2, 1)
```

Numpy array objects

- Functions for creating arrays: -- evenly spaced :

```
>>> a = np.arange(10)   # 0 .. n-1 (!)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> b = np.arange(1, 9, 2)   # start, end (exclusive), step
>>> b
array([1, 3, 5, 7])
```

- Or by number of points :

```
>>> c = np.linspace(0, 1, 6) # start, end, num-points
>>> c
array([ 0. , 0.2, 0.4, 0.6, 0.8, 1. ])
>>> d = np.linspace(0, 1, 5, endpoint=False)
>>> d
array([ 0. , 0.2, 0.4, 0.6, 0.8])
```

Numpy array objects

- Common arrays :

```
>>> a = np.ones((3, 3)) # reminder: (3, 3) is a tuple
>>> a
array([[ 1., 1., 1.],
       [ 1., 1., 1.],
       [ 1., 1., 1.]])
>>> b = np.zeros((2, 2))
>>> b
array([[ 0., 0.],
       [ 0., 0.]])
>>> c = np.eye(3)
>>> c
array([[ 1., 0., 0.],
       [ 0., 1., 0.],
       [ 0., 0., 1.]])
```

Numpy array objects

- np.random usage :

```
>>> a = np.random.rand(4)        # uniform in [0, 1]
>>> a
array([ 0.95799151, 0.14222247, 0.08777354, 0.51887998])

>>> b = np.random.randn(4)     # Gaussian
>>> b
array([ 0.37544699, -0.11425369, -0.47616538, 1.79664113])

>>> np.random.seed(1234)       # Setting the random seed
```

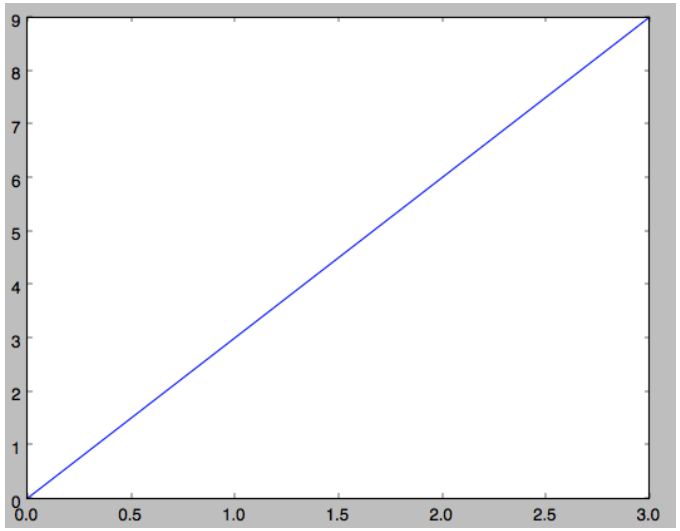
- You can explicitly specify which data-type you want:

```
>>> c = np.array([1, 2, 3], dtype=float)
>>> c.dtype
dtype('float64')
```

Numpy array objects

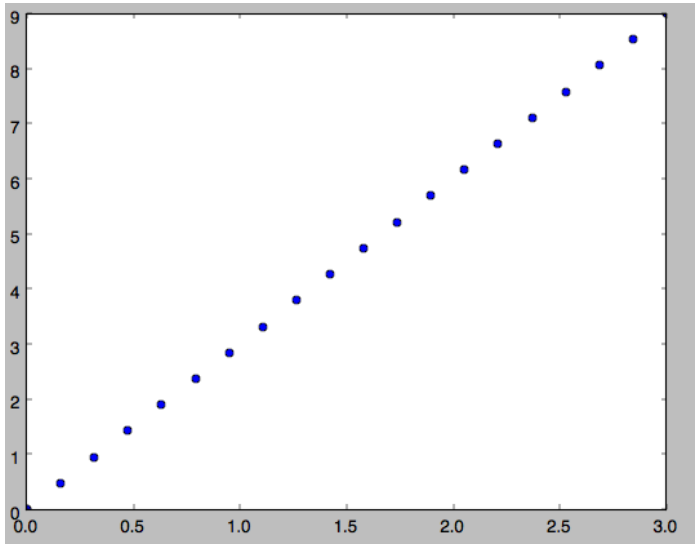
- Basic visualization :

```
>>> import matplotlib.pyplot as plt # the tidy way
```



- 1D plotting :

```
>>> x = np.linspace(0, 3, 20)
>>> y = np.linspace(0, 9, 20)
>>> plt.plot(x, y) # line plot
[<matplotlib.lines.Line2D object at ...>]
>>> plt.show()
>>> plt.plot(x, y, 'o') # dot plot
[<matplotlib.lines.Line2D object at ...>]
>>> plt.show()
```



Indexing and slicing

- The items of an array can be accessed and assigned to the same way as other Python sequences (e.g. lists):

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[0], a[2], a[-1]
(0, 2, 9)
```

Note : Indices begin at 0, like other Python sequences.

- The usual python idiom for reversing a sequence is supported:

```
>>> a[ : : -1]
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

Indexing and slicing

- For multidimensional arrays, indexes are tuples of integers:

```
>>> a = np.diag(np.arange(3))
>>> a
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 2]])
>>> a[1, 1]
1
>>> a[2, 1] = 10    # third line, second column
>>> a
array([[ 0, 0, 0],
       [0, 1, 0],
       [ 0, 10, 2]])
>>> a[1]
array([0, 1, 0])
```

- In2D,the first dimension corresponds to **rows**, the second to **columns**.

Indexing and slicing

- **Slicing**: Arrays, like other Python sequences can also be sliced:

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[2:9:3]    # [start:end:step]
array([2, 5, 8])
```

- Note that the last index is not included! :

```
>>> a[ : 4]
array([0, 1, 2, 3])
```

- All three slice components are not required: by default, start is 0, end is the last and step is 1:

```
>>> a[1:3]
array([1, 2])
>>> a[::2]
array([0, 2, 4, 6, 8])
>>> a[3:]
array([3, 4, 5, 6, 7, 8, 9])
```

Indexing and slicing

- You can also combine assignment and slicing:

```
>>> a = np.arange(10)
>>> a[5:] = 10
>>> a
array([ 0, 1, 2, 3, 4, 10, 10, 10, 10, 10])
>>> b = np.arange(5)
>>> a[5:] = b[::-1]
>>> a
array([0, 1, 2, 3, 4, 4, 3, 2, 1, 0])
```

Copies and views

- A slicing operation creates a **view** on the original array, which is just a way of accessing array data. Thus the original array is not copied in memory.
- **When modifying the view, the original array is modified as well**:

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> b = a[::2]
>>> b
array([0, 2, 4, 6, 8])
>>> np.may_share_memory(a, b)
True
>>> b[0] = 12
>>> b
array([12, 2, 4, 6, 8])
>>> a # (!)
array([12, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a = np.arange(10)
>>> c = a[::2].copy()    # force a copy >>> c[0] = 12
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.may_share_memory(a, c)
False
```

Numerical Operations on arrays

- Element-wise operations – Basic operations

```
>>> a = np.array([1, 2, 3, 4])
>>> a + 1
array([2, 3, 4, 5])
>>> 2**a
array([ 2, 4, 8, 16])
```

- All arithmetic operates elementwise:

```
>>> b = np.ones(4) + 1
>>> a - b
array([-1., 0., 1., 2.])
>>> a * b
array([ 2., 4., 6., 8.])

>>> j = np.arange(5)
>>> 2**(j + 1) - j
array([ 2, 3, 6, 13, 28])
```

- Note : These operations are of course much faster than if you did them in pure python

Numerical Operations on arrays

- Array multiplication is NOT matrix multiplication (!!!)

```
>>> c = np.ones((3, 3))
>>> c * c          # NOT matrix multiplication!
array([[ 1., 1., 1.],
       [ 1., 1., 1.],
       [ 1., 1., 1.]])
```

- **Matrix multiplication:**

```
>>> c.dot(c)
array([[ 3., 3., 3.],
       [ 3., 3., 3.],
       [ 3., 3., 3.]])
```

Numerical Operations on arrays

- Other operations : Comparisons :

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array([4, 2, 2, 4])
>>> a == b
array([False, True, False, True], dtype=bool)
>>> a > b
array([False, False, True, False], dtype=bool)
```

- Array-wise comparisons:

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array([4, 2, 2, 4])
>>> c = np.array([1, 2, 3, 4])
>>> np.array_equal(a, b)
False
>>> np.array_equal(a, c)
True
```

Numerical Operations on arrays

- Logical Operations :

```
>>> a = np.array([1, 1, 0, 0], dtype=bool)
>>> b = np.array([1, 0, 1, 0], dtype=bool)
>>> np.logical_or(a, b)
array([ True, True, True, False], dtype=bool)
>>> np.logical_and(a, b)
array([ True, False, False, False], dtype=bool)
```

- Transcendental functions:

```
>>> a = np.arange(5)
>>> np.sin(a)
array([ 0. , 0.84147098, 0.90929743, 0.14112001, -0.7568025 ])
>>> np.log(a)
array([ -inf, 0.  , 0.69314718, 1.09861229, 1.38629436])
>>> np.exp(a)
array([ 1.  , 2.71828183, 7.3890561 , 20.08553692, 54.59815003])
```

Numerical Operations on arrays

- Transposition :

```
>>> a = np.triu(np.ones((3, 3)), 1)
>>> a
array([[ 0., 1., 1.],
       [ 0., 0., 1.],
       [ 0., 0., 0.]])
>>> a.T
array([[ 0., 0., 0.],
       [ 1., 0., 0.],
       [ 1., 1., 0.]])
```

- Note : **The transposition is a view**
  As a results, the following code **is wrong** and will **not make a matrix symmetric**:

```
>>> a += a.T
```

Note : The sub-module numpy.linalg implements basic linear algebra, such as solving linear systems, singular value decomposition, etc. However, it is not guaranteed to be compiled using efficient routines, and thus we recommend the use of scipy.linalg, as detailed in section *Linear algebra operations: scipy.linalg*

Basic Reductons

- Computing sums :

```
>>> x = np.array([1, 2, 3, 4])
>>> np.sum(x)
10
>>> x.sum()
10
```

- Sum by rows and by columns:

```
>>> x = np.array([[1, 1], [2, 2]])
>>> x
array([[1, 1],
       [2, 2]])
>>> x.sum(axis=0)     # columns (first dimension)
array([3, 3])
>>> x[:, 0].sum(), x[:, 1].sum()
(3, 3)
>>> x.sum(axis=1)     # rows (second dimension)
array([2, 4])
>>> x[0, :].sum(), x[1, :].sum()
(2, 4)
```

Basic Reductons

- Other reductions :

```
>>> x = np.array([1, 3, 2])
>>> x.min()
1
>>> x.max()
3
>>> x.argmin()        # index of minimum
0
>>> x.argmax()        # index of maximum
1
```

- Logical operations:

```
>>> np.all([True, True, False])
False
>>> np.any([True, True, False])
True
```

Basic Reductons

- Can be used for array comparisons:

```
>>> a = np.zeros((100, 100))
>>> np.any(a != 0)
False
>>> np.all(a == a)
True
>>> a = np.array([1, 2, 3, 2])
>>> b = np.array([2, 2, 3, 2])
>>> c = np.array([6, 4, 4, 5])
>>> ((a <= b) & (b <= c)).all()
True
```

Basic Reductons

- Statistics:

```
>>> x = np.array([1, 2, 3, 1])
>>> y = np.array([[1, 2, 3], [5, 6, 1]])
>>> x.mean()
1.75
>>> np.median(x)
1.5
>>> np.median(y, axis=-1)      # last axis
array([ 2.,  5.])
>>> x.std()       # full population standard dev.
0.8291561975888499
```

... and many more

Array Shape manipulation

- Flattening :

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> a.ravel()
array([1, 2, 3, 4, 5, 6])
>>> a.T
array([[1, 4], [2, 5], [3, 6]])
>>> a.T.ravel()
array([1, 4, 2, 5, 3, 6])
```

- Reshaping : The inverse operation to flattening:

```
>>> a.shape
(2, 3)
>>> b = a.ravel()
>>> b = b.reshape((2, 3))
>>> b
array([[1, 2, 3],
       [4, 5, 6]])
```

Array Shape manipulation

- Adding a dimension : Indexing with the np.newaxis object allows us to add an axis to an array

```
>>> z = np.array([1, 2, 3])
>>> z
array([1, 2, 3])
>>> z[ :, np.newaxis]
array([[1],
       [2],
       [3]])
>>> z[np.newaxis, : ]
array([[1, 2, 3]])
```

Array Shape manipulation

- Dimension shuffling :

```
>>> a = np.arange(4*3*2).reshape(4, 3, 2)
>>> a.shape
(4, 3, 2)
>>> a[0, 2, 1]
5
>>> b = a.transpose(1, 2, 0)
>>> b.shape
(3, 2, 4)
>>> b[2, 1, 0]
5
```

- Resizing :

```
>>> a = np.arange(4)
>>> a.resize((8,))
>>> a
array([0, 1, 2, 3, 0, 0, 0, 0])
```

Sorting data

- Sorting along an axis :

```
>>> a = np.array([[4, 3, 5], [1, 2, 1]])
>>> b = np.sort(a, axis=1)
>>> b
array([[3, 4, 5],
       [1, 1, 2]])
```

- Sorts each row separately! -- In-place sort:

```
>>> a.sort(axis=1)
>>> a
array([[3, 4, 5],
       [1, 1, 2]])
```

- Rounding :

```
>>> a = np.array([1.2, 1.5, 1.6, 2.5, 3.5, 4.5])
>>> b = np.around(a)
>>> b      # still floating-point
array([ 1., 2., 2., 2., 4., 4.])
>>> c = np.around(a).astype(int)
>>> c
array([1, 2, 2, 2, 4, 4])
```

Loading Data files

- Text files [Say we have a text file : populations.txt]

```
# year    hare      lynx      carrot
1900      30e3      4e3       48300
1901      47.2e3    6.1e3     48200
1902      70.2e3    9.8e3     41500
1903      77.4e3    35.2e3    38200
```

```
>>> data = np.loadtxt('data/populations.txt')
>>> data
array([[ 1900., 30000., 4000., 48300.],
        [ 1901., 47200., 6100., 48200.],
        [ 1902., 70200., 9800., 41500.],
……
```

```
>>> np.savetxt('pop2.txt', data)
>>> data2 = np.loadtxt('pop2.txt')
```

Loading Data files

- Load CSV File with Numpy [say the file is : pima-indians-diabetes.data.csv]

```
import numpy
filename = 'pima-indians-diabetes.data.csv'
raw_data = open(filename, 'rt')
data = numpy.loadtxt(raw_data, delimiter=",")
print(data.shape)
```

Introduction to Scipy

Introduction to Scipy

- The scipy package contains various toolboxes dedicated to common issues in scientific computing.
- Its different submodules correspond to different applications, such as interpolation, integration, optimization, image processing, statistics, special functions, etc.
- scipy can be compared to other standard scientific-computing libraries, such as the GSL (GNU Scientific Library for C and C++), or Matlab's toolboxes.
- scipy is the core package for scientific routines in Python; it is meant to operate efficiently on numpy arrays, so that numpy and scipy work hand in hand.
- Before implementing a routine, it is worth checking if the desired data processing is not already implemented in Scipy.
- Scipy's routines are optimized and tested, and should therefore be used when possible.

Introduction to Scipy

- scipy is composed of task-specific sub-modules:

scipy.cluster	Vector quantization / Kmeans
scipy.constants	Physical and mathematical constants
scipy.fftpack	Fourier transform
scipy.integrate	Integration routines
scipy.interpolate	Interpolation
scipy.io	Data input and output
scipy.linalg	Linear algebra routines
scipy.ndimage	n-dimensional image package
scipy.odr	Orthogonal distance regression
scipy.optimize	Optimization
scipy.signal	Signal processing
scipy.sparse	Sparse matrices
scipy.spatial	Spatial data structures and algorithms
scipy.special	Any special mathematical functions
scipy.stats	Statistics

- They all depend on numpy, but are also mostly independent of each other.

Introduction to Scipy

- For further discussion , please refer to scipy manual
- Scipy Lecture - simple – creative commons

Reference

- Scipy Lecture notes – creative commons
- An Introduction to numpy and scipy
- Scipy and Numpy – Eli Bressert – [O'Reilly]
- Numpy – Ivan Idris – [PACKT]

End of Presentation

Python – Numpy