NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the BSD (http://www.numpy.org/license.html#license) license, enabling reuse with few restrictions.

Installation Guide

It is strongly recommended that you install Python using the Anaconda distribution to make sure all underlying dependencies work with conda install. If you have already installed Anaconda, install NumPy by opening your terminal(for MAC Users) or ANACONDA Command Prompt(for Windows Users) and type the following command -

```
conda install numpy
```

If you are unable to install ANACONDA due to restrictions, please refer to Numpy's official documentation on installation steps. (http://docs.scipy.org/doc/numpy-1.10.1/user/install.html)

Using NumPy Package

Once NumPy Package is installed successfully, you can import it to the current Python session using the following code:

In [1]:
```
import numpy as np        # Call the NumPy package using np. notation after ex
ecuting the same
```

PART 1 - Intro to Arrays in NumPy

Numpy has a ton of built-in functions that are useful for Data Scientists & Python Programmers alike.

We shall cover some of the most important topics in Numpy:

- Arrays (using Vectors & Matrices)
- Number Generation Concepts

Numpy Arrays

NumPy arrays are the one of the most widely used data structuring techniques by Data Scientists.

Numpy arrays are of two types: Vectors and Matrices.

Vectors are 1-dimensional arrays and matrices are 2-dimensional arrays(A Matrix can still possess a single row or a column).

We shall begin our learning with how to create NumPy Arrays.

Creating simple NumPy Array Structures

We could create a simple Array by using a list of values or a list of "List of values".

In [ ]:
```
simple_list = [101,102,103,104,105,106,107,108,109,110]
simple_list
```

In [ ]:
```
np.array(simple_list)
```

In [ ]:
```
simple_list_of_lists = [[10,11,12],[20,21,22],[30,31,32]]
simple_list_of_lists
```

In [ ]:
```
np.array(simple_list_of_lists)
```

There are multiple built-in methods to generate Arrays

arange

Return evenly spaced values within a given interval as input.

In [ ]:
```
np.arange(0,20)         # Return values 0 to 19. Start value is 0 which is inc
luded, the stop value provided is 20 which is not included
```

In [ ]:
```
np.arange(0,20,4)       # Specify start, stop and step values as input
```

0's and 1's

Generate arrays of 0's or 1's

In [ ]:
```
np.zeros(10)            # Specify the count of 0's required in the array
```

In [ ]:
```
np.zeros((4,3))         # Specify the number of rows by columns - 4 rows and 3
cols in this example
```

In [ ]:
```
np.ones(10)            # Specify the count of 1's required in the array
```

In [ ]:
```
np.ones((4,5))         # Specify the number of rows by columns - 4 rows and 5
cols in this example
```

linspace

Return evenly spaced numbers over a specified interval.

In [ ]:
```
np.linspace(0,20,5)    # Specify the start, stop and number values needed betw
een them. Please note the stop value is also considered in this case
```

In [ ]:
```
np.linspace(0,20,100)
```

eye

Return a 2-D array with ones on the diagonal and zeros elsewhere. Also called an identity matrix

In [2]:
```
np.eye(10)
```

Out[2]:
```
array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

Random

Numpy has lots of options to create random numbered arrays:

rand

Create an array of the given shape and populate it with random variables derived from a uniform distribution between `[0, 1)` .

```
In [3]: np.random.rand(5)
```

```
Out[3]: array([0.68137   , 0.8354984 , 0.7162431 , 0.67343426, 0.35486654])
```

```
In [4]: np.random.rand(3,2)
```

```
Out[4]: array([[0.54639647, 0.01601404],
               [0.69119265, 0.62032506],
               [0.20936566, 0.18221765]])
```

randn

Return a variable (or a set of variables) from the "Standard Normal" distribution. Unlike rand which is from a uniform distribution:

A standard Normal Distribution has mean 0 and SD of 1 as we know.

```
In [5]: np.random.randn(5)
```

```
Out[5]: array([-0.16960704, -1.03152155, -0.42765096,  0.07191182, -0.85161314])
```

```
In [6]: np.random.randn(3,2)
```

```
Out[6]: array([[-0.77587499, -0.51237864],
               [-0.54740058,  0.95151471],
               [ 1.40077439,  0.25820996]])
```

randint

Return random integers from `low` (inclusive) to `high` (exclusive).

```
In [7]: np.random.randint(5,20)       # Returns one rand integer between the values 5
        & 19(20 is excluded)
```

```
Out[7]: 15
```

In [8]:
```
np.random.randint(20,50,5)  # Returns  5 rand integers between 20 & 49(50 is e
xcluded)
```

Out[8]:
```
array([46, 27, 39, 39, 41])
```

Array Attributes and Methods for an array

Let us look at some important attributes and methods for an array.

In [9]:
```
sample_array = np.arange(30)
rand_array = np.random.randint(0,100,20)
```

In [10]:
```
sample_array
```

Out[10]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])
```

In [11]:
```
rand_array
```

Out[11]:
```
array([97, 48, 74, 81,  6, 23,  2, 10, 45, 33, 26, 13, 16, 62, 77, 86, 94,
       37, 96, 98])
```

Reshape

Returns an array containing the same data with a new shape.

In [12]:
```
sample_array.reshape(5,6)
```

Out[12]:
```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29]])
```

max,min,argmax,argmin

These are useful methods for finding max or min values. Or to find their index locations using argmin or argmax

In [13]:
```
rand_array
```

Out[13]:
```
array([97, 48, 74, 81,  6, 23,  2, 10, 45, 33, 26, 13, 16, 62, 77, 86, 94,
       37, 96, 98])
```

In [14]:
```
rand_array.max()
```

Out[14]:
```
98
```

In [15]:
```
rand_array.argmax()
```

Out[15]: 19

In [ ]:
```
rand_array.min()
```

In [ ]:
```
rand_array.argmin()
```

Shape

Shape is an attribute that arrays have. It is not a method.

In [ ]:
```
# Vector
sample_array.shape
```

In [ ]:
```
# Output has two sets of brackets - which indicates a matrix and not a vector
sample_array.reshape(1,30)
```

In [ ]:
```
sample_array.reshape(1,30).shape
```

In [ ]:
```
sample_array.reshape(30,1)
```

In [ ]:
```
sample_array.reshape(30,1).shape
```

dtype

You could retrieve the data type of the object in an array using dtype.

In [ ]:
```
sample_array.dtype
```

In [ ]:
```
sample_array2 = np.random.randn(4,4)

sample_array2
```

In [ ]:
```
sample_array2.T          # Transpose a matrix
```

Part 2 - NumPy Indexing & Selection

We will now learn how to select objects or groups of objects from an array

In [ ]:
```
#Create a sample array with 11 values
sample_array = np.arange(10,21)
```

In [ ]:
```
#Show
sample_array
```

Using brackets for Indexing & Selection

In [ ]:
```
#Get the value at index position 8 from above sample array - Indexing starts f
rom 0 and not 1
sample_array[8]
```

In [ ]:
```
#Get values from a range selection
sample_array[0:3]
```

In [ ]:
```
#Get values from specific index positions
sample_array[[0,4,7]]
```

Broadcasting

NumPy arrays have the capability of Broadcasting values as seen below

In [ ]:
```
#Setting a fixed value 100 using index range (Broadcasting)
sample_array[1:2]=100

#Show
sample_array
```

In [ ]:
```
# Reset array, we shall see the need for reset
sample_array = np.arange(10,21)

#Show
sample_array
```

In [ ]:
```
#Subsetting an array
subset_sample_array = sample_array[0:7]

#Output of the subset
subset_sample_array
```

In [ ]:
```
#Change the values in the subset
subset_sample_array[:]=1001

#Show subset output again
subset_sample_array
```

Please Note that the changes in the subset has affected the original sample_array as well

In [ ]:
```
sample_array
```

Data is not copied when we subset, it is a sub-view of the original array

This is a feature by default in order to avoid memory problems

In [ ]:
```
#To get a copy, we need to use the copy function
copy_array = sample_array.copy()

copy_array
```

Indexing a Matrix - 2 dimensional arrays

The general formats used are

sample_matrix[row][col]

or

sample_matrix[row,col]

We will use the second option as standard.

In [ ]:
```
sample_matrix = np.array(([50,200,5,10],[10,35,50,15],[25,100,145,120],[105,2
5,65,80]))

#Show output
sample_matrix
```

In [ ]:
```
#Indexing rows
sample_matrix[1]
```

In [ ]:
```
# Getting an individual element value from the matrix - Method 1
sample_matrix[1][2]
```

In [ ]:
```
# Getting an individual element value from the matrix - Method 2
sample_matrix[1,2]
```

In [ ]:
```
# Slicing matrix

#Shape (3,3) from top right corner
sample_matrix[:3,1:]
```

In [ ]:
```
#Shape bottom row - Extract values from last row only
sample_matrix[3]
```

In [ ]:
```
#Shape bottom row - Including column selection (Alternate to above)
sample_matrix[3,:]
```

Custom Indexing of Matrix

Custom or Fancy indexing allows you to select entire rows or columns specifically in interest.

In [ ]:
```
#Set up a sample matrix
sample_matrix = np.array(([50,200,5,10,160],[170,10,35,50,15],[125,25,100,145,
120],[100,105,25,65,80],[250,15,25,30,100]))

sample_matrix
```

In [ ]:
```
sample_matrix[:,(1,3)]
```

In [ ]:
```
#Allows to retrieve in any order
sample_matrix[:,(3,1)]
```

Selection

Using brackets for selection based on operators for comparison

In [ ]:
```
simple_array = np.arange(1,31)
simple_array
```

In [ ]:
```
simple_array <10
```

In [ ]:
```
boolean_array = simple_array<10
```

In [ ]:
```
boolean_array
```

In [ ]:
```
simple_array[boolean_array]
```

In [ ]:
```
simple_array[simple_array>15]
```

In [ ]:
```
a = 11
simple_array[simple_array>a]
```

Part 3

NumPy Operations

Arithmetic Operations

In [ ]:
```
simple_array = np.arange(0,21)
```

In [ ]:
```
simple_array + simple_array
```

In [ ]:
```
simple_array * simple_array
```

In [ ]:
```
simple_array - simple_array
```

In [ ]:
```
# You get a warning message when dividing 0 by 0, and the value is replaced wi
th nan
simple_array/simple_array
```

In [ ]:
```
# Here, 10/0 is infinity and not nan - You get a warning message along with it
10/simple_array
```

In [ ]:
```
simple_array**2
```

Universal Array Functions

Numpy has numerous universal array functions (http://docs.scipy.org/doc/numpy/reference/ufuncs.html)

They are mathematical operations that can be performed to the entire array

In [ ]:
```
#Calculating Square Root
np.sqrt(simple_array)
```

In [ ]:
```
#Calclating exponential values (e^)
np.exp(simple_array)
```

In [ ]:
```
np.max(simple_array) #Can be found out using simple_array.max() format as well
```

In [ ]:
```
np.argmax(simple_array)  #Gives the index position of max value in the array
```

In [ ]:
```
np.sin(simple_array)
```

In [ ]:
```
np.log(simple_array)
```

In [ ]:
```
np.cos(simple_array)
```

In [ ]:
```
np.square(simple_array)
```

In [ ]:
```
array_2 = np.random.randn(5,5)    #Create a 5x5 matrix with decimal numbers
```

In [ ]:
```
array_2
```

In [ ]:
```
np.round(array_2,decimals=2)     #Rounding off the values to 2 decimal places
```

In [ ]:
```
np.round(array_2)               #Rounding off values to zero decimal places
```

In [ ]:
```
np.std(array_2)                 # Calculate standard deviation
```

In [ ]:
```
np.var(array_2)                 # Calculate variance
```

In [ ]:
```
np.mean(array_2)                # Calculate mean
```

In [ ]:
```
#Using the unique function
sports = np.array(['Golf', 'Cricket', 'Football', 'Football','Cricket','cricke
t','Basketball', 'Baseball'])

np.unique(sports)
```

Why do we see cricket twice in above output??

In [ ]:
```
# in1d we can test values in one array - returns boolean output
np.in1d(['Fooseball','Cricket','Baseball'],sports)
```

Part 4

NumPy - Array Input and Output

In [ ]:
```
import numpy as np
```

In [ ]:
```
#Create a simple array
simple_array = np.arange(11)

simple_array
```

In [ ]:
```
#Save array on hard disk in binary format (file extension is .npy)
np.save('array1',simple_array)
```

In [ ]:
```
#Change arr
simple_array = np.arange(21)
#Show
simple_array
```

In [ ]:
```
#Lets see the original saved copy - This does not get affected by the changes
we have made to the simple_array
np.load('array1.npy')
```

In [ ]:
```
#Saving multiple arrays into a zip file using savez (file extension is .npz an
d not .npy)
np.savez('2_arrays.npz',a=simple_array,b=simple_array)
```

In [ ]:
```
#Now loading multiple arrays using load
archived_arrays = np.load('2_arrays.npz')

#Show
archived_arrays['a']
```

In [ ]:
```
#Saving the arrays as text files

simple_array2 = np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
np.savetxt('my_text.txt',simple_array2,delimiter=',')
```

In [ ]:
```
simple_array2 = np.loadtxt('my_text.txt',delimiter = ',')

simple_array2
```

End of NumPy Section