A guide to Numerical Python (Numpy) Library
by Malay Mitra

NumPy provides two fundamental objects:

- an N-dimensional array object (ndarray) and
- a universal function object (ufunc).

In addition, there are other objects that build on top of these which you may find useful in your work. An N-dimensional array is a homogeneous collection of “items” indexed using N integers. There are two essential pieces of information that define an N-dimensional array:

- the shape of the array, and
- the kind of item the array is composed of.

Numpy ndarray – Numpy N-dimensional array. Which is a multidimensional array object. While using 'numpy' library in your script, one has to import the library.

Numpy ndarray is generic multi-dimensional container of homogeneous data i.e. all elements must be same type.

Adding NumPy to your namespace

If your program is using Numpy library primarily while doing computation, place the 'import' statement at the top of your script, and it will add all the names from the numpy module to your name space

```
from numpy import *
```

You can then refer directly to the constants like pi and functions like array() and arange().

```
print(pi)
```

Otherwise you have to write print(numpy.pi)

Note : For all Example programs calling 'numpy' library, one has to import as follows which works like alias.

```
import numpy as np              # short name ‘np’ is your choice.
```

Examples:

a)

```
# Create a row vector and column vector in Numpy
import numpy as np
row_vect = np.array([1,2,3])
col_vect = np.array([[1],[2],[3]])
print(row_vect)
print(col_vect)
```

b)

```
# Generate an array of 2 rows and 4 columns. 2D array
ar1 = np.array([[2,4,6,8], [1,3,5,7]])
print(ar1)              # output        [[2 4 6 8]
                                         [1 3 5 7]]
# Element wise Batch operation on equal size arrays without any for – loop.
#This is called 'Vectorization'
print(ar1*2)            # Output        [[ 4  8 12 16]
                                         [ 2  6 10 14]]
# Similarly we can write ar1+2, ar-2 etc.
```

```
print(ar1-ar1)                          # Prints all elements as 0
print(1/ar1)                            # Prints the reciprocals
print('Dim :', ar1.ndim)                # Dim :2
print('Shape:', ar1.shape)              # Shape:(2,4), 2 rows, 4 columns
print('Dtype :', ar1.dtype)             # Shows data type as int32 (32 bit integer)
ar2 = ar1 * 0.5
print(ar2)                              # Output [[1.  2.  3.  4. ]
                                              [0.5 1.5 2.5 3.5]]
print(ar2.dtype)                        # Each element is float64, 64 bit float
                                        # Returns float64 when at least one element is float
```

c)

```
# Numpy 2D array of string data types
import numpy as np
arr_str = np.array([['unix', 'windows', 'linux'],
                    ['C++', 'Python', 'Java']])
print(arr_str)

Output:
        [['unix' 'windows' 'linux']
         ['C++' 'Python' 'Java']]
```

Note: Alternatively we can write np.mat([[1,2], [3,4], [5,6]]). Majority operations on Numpy return array objects but not matrix objects. So np.array is preferred.

- **Numpy array from Python List data type**

Examples:

a)

```
# Create ndarray from a list using 'array' function
lst = [2,4,6,8,10]
arr  = np.array(lst)
print(arr)              # Outputs : array([ 2,  4,  6,  8, 10])
print(arr.ndim)         # Dimension of arr = 1
```

b)

```
# Create 2D numpy array from list of lists (2D)
import numpy as np
lst = [ [1,3,5,7], [2,4,6, 8] ]
arr = np.array(lst)
print(arr)                      # array([[1, 3, 5, 7],
                                # [2, 4, 6, 8]])
print(arr.shape)                # (2, 4)
print(arr.ndim)                 # 2, 2D
```

- **Numpy array from Python Tuple data type**

Examples:

a)

```
import numpy as np
tup1 = (2, 4, 6, 8, 10)     # tup1 is a tuple data type
arr = np.array(tup1)        # arr is
print(arr)                  # [ 2  4  6  8 10]
```

- **<u>'arange' function in Numpy</u>**

  Like 'range' in Python, we have 'arange' in Numpy

  Syntax of 'arange':
  np.arange(start, stop, step)   where default values of start= 0 and step=1

Examples :

```
print(np.arange(1.0, 4.0))          # prints [ 1. 2. 3.]
print np.arange(5.0, -0.5, -0.5)    # [ 5. 4.5 4. 3.5 3. 2.5 2. 1.5 1. 0.5 0. ]
print(np.arange(4))                 # prints [0 1 2 3]
# arange with dtype
print(np.arange(10, dtype=np.float64))  # Output : [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]

a= np.arange(5, 10, 0.5)
print(a)                            #  starts at 5, goes up to 9.5, step 0.5
print(np.arange(0.0, 0.4, 0.1))      # [ 0. , 0.1,  0.2,  0.3]
print(np.arange(5.0, -0.5, -0.5))   #[ 5. ,  4.5,  4. ,  3.5,  3. ,  2.5,  2. ,  1.5,  1. ,  0.5,  0. ])
# Reverse print
print(np.arange(10,1,-2))           # [10  8  6  4  2]
```

- **<u>Type casting in Numpy array using ‘astype’ method</u>**

Examples:

```
import numpy as np
# While casting floating point to integers,
# there would be truncation of values
float_arr = np.array([3.75, -1.26, 0.99, 71.09])
# convert to int array
int_arr  = float_arr.astype(np.int32)
print(int_arr)              # [ 3 -1  0 71]    .Prints integers, decimals truncated

# Create a array of strings and then convert to float 64bit
num_str= np.array(['1.25', '90.00', '-15.75', '0.75'])
print(num_str)      # ['1.25' '90.00' '-15.75' '0.75']
num_arr = num_str.astype(np.float64)
print(num_arr)     # [  1.25  90.   -15.75   0.75]
# Converting numerical array to Boolean array where we know True=1 and False=0
Import numpy as np
num_arr = np.array([1, 0, 1])
print(num_arr)                     # [1 0 1]
bool_arr = num_arr.astype(np.bool)
print(bool_arr)                    # [ True False  True]
```

Note: During type casting if there is any data type mismatch, ‘ValueError’ exception is raised as given in the below code.

# This code gives ‘ValueError’ because 'arr1' contains non-numeric strings

```
arr1 = np.array(['unix', 'windows', 'linux'])
print(arr1)                          # ['unix' 'windows' 'linux'] with dtype = ‘<U7’
arr2 = arr1.astype(np.int32)
```

- **<u>The following table shows NumPy numerical data types:</u>**

Type	Description
bool	Boolean (True or False) stored as a bit
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2 ** 31 to 2 ** 31 -1)
int64	Integer (-2 ** 63 to 2 ** 63 -1)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 2 ** 32 - 1)
uint64	Unsigned integer (0 to 2 ** 64 - 1)
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64 or float	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128 or complex	Complex number, represented by two 64-bit floats (real and imaginary components)

- **We can also mention ‘dtype’ (data type) as argument, which is often optional as follows:**

Example:

```
import numpy as np
arr = np.array([2,4,6,8], dtype='uint16')
# itemsize gives size taken by each element
# len gives total number of elements

size     # 2 bytes which is 16 bits
elements = len(arr)     # 4
print('Total size :', size * elements)     # Total size : 8
```

Note : You cannot convert a complex number into an integer or float (TypeError).
However, you can convert a int/float to a complex number, e.g. complex(1.0).

- **<u>Sorting an Array</u>**

```
import numpy as np
a = np.array([2, 7, 10, 1, -8, 5])
a.sort()                  # Ascending sort
print(a)
a[::-1].sort()            # Reverse sort. Descending order
print(a)
```

- **<u>Basic Indexing and Slicing (:)</u>**

Slicing Operator ':' which takes the form → **[start:end:step].**
By default: Start takes 0, stop goes up to last element, step is 1

<u>Indexing/Slicing in One dimensional array</u>
Examples:

```
import numpy as np
arr = np.arange(0,15,2)
print(arr)              # [ 0  2  4  6  8 10 12 14]
# Accessing 5th (i.e. 6th, since index starts from 0 )
print(arr[5])           # 10
# Now slicing the array
print(arr[2:6])         # [ 4  6  8 10]. Starts from 2nd stops before 6th
# Prints odd numbered elements since starts from 1 and step is 2
print(arr[1::2])        # [ 2  6 10 14]
# Print even numbered elements since starts from 0 and step is 2
print(arr[0::2])        # [ 0  4  8 12]
# Starts at 0, stops at end, step 3
print(arr[::3])         # [ 0  6 12]
# Whole array is printed
print(arr[::])            # [ 0  2  4  6  8 10 12 14]. Same as arr
```

- **<u>Reverse a ndarray</u>**

```
import numpy as np
arr = np.array([5, -9, 0, 11, 3, 1])
print(arr)                          # [ 5 -9  0 11  3  1]
arr1 = arr[::-1]                    # Starts from 0, stops at last element, step -1
print(arr1)                         # [ 1  3 11  0 -9  5]
```

- **<u>Update selective elements of ndarray via slice</u>**

```
arr[5:8] = 99           # elements at 5,6,7 become 99
print(arr)              # [ 0  2  4  6  8 99 99 99]
# Slice is just a view of the original array.
# Any change in the sliced array would reflect in the original array

ar_slice = arr[5:8]                 # ar_slice is [99, 99, 99]
ar_slice[1]=100                     # ar_slice becomes [ 99, 100,  99]
# Changes take place in the original array
print(arr)                          # [  0   2   4   6   8 100  99  99]
```

- **<u>Explicit copy of 'ndarray' by 'copy()' method</u>**

  Example:

```
    # Explicit copy instead of view
    arr1 = np.arange(10)              # arr1 becomes [0,1,2,3,4,5,6,7,8,9]
    slice  = arr1[5:8].copy() # Sliced array is copied physically
    slice[:] = 100                    # make all elements in the sliced as 100
    print(slice)                      # [100,100,100]
    # Original array remains unchanged.
    print(arr1)                       # [0 1 2 3 4 5 6 7 8 9]
```

- **<u>Indexing/Slicing in two dimensional array</u>**

<u>Row slicing of 2D array</u>

Examples:

```
    import numpy as np
    arr = np.array([[1,2,3], [4,5,6], [7,8,9]])          # 2D array. 3X3
    print(arr[2])                                        # [7,8,9]. 3rd Row
    print(arr[0])                                        # [1,2,3]. 1st Row
    # Select individual element
    print(arr[0][2])                                     # 0th row, 2nd element
    # Slicing
    # Note that [0:1] is same as [:1]
    print(arr[0:1])             # array([1,2,3]). 0th  row
    print(arr[:1])              # prints [[1 2 3]], 0th  row
    print(arr[0:2])             # prints 0th  and 1st row
    print(arr[-1])              # prints the last row. array([7,8,9])
    print(arr[:2])              # prints Row 0 and Row 1 but not 2.

    print(arr[0::2])            # starts at 0, step 2.
                                # Outputs:        [[1 2 3]
                                                   [7 8 9]]
```

- **<u>Row Slicing & Column Slicing of 2D Array</u>**

  For higher dimensional array objects, we can slice row wise & column wise. Slicing parameter is separated by ','.

  Examples:

```
    # Row slice and column slice parameter separated by comma
    # Row parameters are mentioned before column parameters

    import numpy as np
    arr = np.array([[1, 2, 3],[4, 5, 6], [7, 8, 9]])
    print('Original array :', arr)
    print(arr[:, 0:2])          # ':' before comma indicates all rows,
                                #  0:2 after comma indicates start from col 0 to 1, but not 2
                                # Output:        [[1 2]
                                 #                [4 5]
                                #               [7 8]]
    print(arr[::2, :])        # '::2' before comma means starts from 0, ends at last,
                                # steps 2 means Rows 0,2
                                # After comma ‘:’ means all columns
                                # Output:    [[1, 2, 3],
                                #             [7, 8, 9]]
    print(arr[::2, ::2])    # Alternate rows [0,2] and alternate columns (0,2)
                                # Outputs:   [[1 3]
                                #             [7 9]]
    print(arr[1:2, :])      # starts from row 1 stops before 2 so Row 1 only.
                                # Colon after comma means all columns.
                                # Output : [[4,5,6]]

    print(arr[:2, :1])      # Row 0,1 but not 2. Starts at column 0 and stops at 1 but not 1
                                # Output:   [[1]
                                #            [4]]
    print(arr[2:, 2:])      # Prints [[9]].
                                # Starts at row 2 up to end which is row 2.
                                # Starts at col 2 and stops at end which is 9
                                # Only row slice
    print(arr[:2])          # prints   [[1 2 3]
                                #  [4 5 6]]

    # '-1' is last, '-2' is second last
    print(arr[:, -1])       # Prints last column from all rows [3,6,9]
    print(arr[:, -2])       # prints second last column from all rows [2, 5, 8]
```

- **Some functions on Numpy array.**

'sum' function is used to sum the elements in a 'ndarray' object.
either row wise (axis=1) or column wise(axis=0).

Examples:

a)

```
import numpy as np
print(np.sum([0.5, 1.5]))                 # 2, sum of all elements
print(np.sum([[0, 1], [0, 5]]))           # 6, sum of all elements
print(np.sum([[0, 1], [0, 5]], axis=0))   # array([0,6]). Col wise sum.
print(np.sum([[0, 1], [0, 5]], axis=1))   # array([1, 5]). Row wise sum.
```

b)

```
import numpy as np
matrix = np.array([[1,2,3], [4,5,6], [7,8,9]])
s = np.sum(matrix)
x = np.max(matrix)
y= np.min(matrix)
print(s, x, y)                            # 45, 9, 1

print(np.sum(matrix, axis=0))     # Column wise sum, [12 15 18]
print(np.sum(matrix, axis=1))     # Row wise sum, [ 6 15 24]
print(np.max(matrix,axis=0))      # Col wise max
print(np.max(matrix, axis=1))     # Row wise max
```

- **Some Statistical Functions on Numpy array**

```
# mean,  variance, standard deviations
print(np.mean(matrix))                # 5.0
print(np.std(matrix))                 # 2.581988897471611
print(np.var(matrix))                 # 6.666666666666667
```

Note: one can use axis=-0 (Col wise) or axis=1 (Row wise) on these functions

- **Universal Functions (ufunc) in Numpy**

'ufunc' is a function that performs element wise operation on ndarray.

Examples:

```
import numpy as np
# sqrt and exp
ar = np.array([4, 16, 25])
sqrt_ar = np.sqrt(ar)
print(sqrt_ar)                 # [2, 4, 5]
print(np.square(ar))           # [ 16 256 625]

# We can also use  mathematical operators like '*', '+', '-', '/' with ndarray
import numpy as np
a = np.array([10,6,3])
b = np.array([3,2,3])
print(a*b)                     # [30 12  9]
print(a+b)                     #[13  8  6]
print(a-b)                     # [7 4 0]
# Arithmetic operation with a scalar
print(a+1)                     # [11  7  4]
print(2 ** b)                  # [8, 4, 8], 2 is raised to power of each element of 'b'
```

- **'arange' with 'reshape' on numpy array**

Examples:

```
import numpy as np
y = np.arange(4,10).reshape(2,3)      # from 4 up to 9. total 6 elements. 2 rows, 3 cols
print(y)                              # Output : array([[4, 5, 6],
                                                        [7, 8, 9]])
a = np.arange(10).reshape(2,5)        # 2 rows, 5 cols
print(a)                              # [[0 1 2 3 4]
                                      #   [5 6 7 8 9]]
```

- **Column wise and Row wise sorting of a Numpy Array**

Sorting an ndarray object in Numpy makes the original array changed as per sort. By default, it makes ascending sort.

Examples:

```
# Column wise sort of 2D array
b = np.array([[2, 30, 1], [5, 9, 0]])          # Output :    array([[ 2, 30,  1],
                                                                    [ 5,  9,  0]])
c = np.sort(b, axis=0)
print(c)                                       # Output : array([[ 2,  9,  0],
                                                                 [ 5, 30,  1]])
# Row wise sort
d = np.sort(b, axis=1)
print(d)                                       # Output : array([[ 1,  2, 30],
                                                                 [ 0,  5,  9]])
# Without 'axis' clause, it makes row wise sort
print(np.sort(b))                              # Output : array([[ 1  2 30]
                                                                 [ 0  5  9]]
```

- **<u>Random number generator function in Numpy :</u>**

Examples:

```
# 'randint' – Random integers

# To avoid lengthy write 'numpy.random.randint'
# we import 'random' under numpy as 'rnd'
import numpy as np
import numpy.random as rnd

print(rnd.randint(low=1, high=100, size=4))  # Creates 4 random integers between 1 to 100
print(rnd.randint(0,10,6))                    # Generate 6 random integers between 0 to 9


# Creates a 2D array with 2 X 4 with numbers between 0 to 4. low = 0 as default
arr1 = np.random.randint(5, size=(2, 4))          # low=0, high=5, between 0 to 4
arr2 = np.random.randint(2, 10, size=(2, 4))      # Creates an array 2x4 with nos. 2 to 9
print(arr1)
print(arr2)

#  numpy.random.rand() - creates an array of specified shape and filled with random values.
print(np.random.rand())                     # Generates a single random number

# 1D Array
print(np.random.rand(2))          # Generates two random numbers
array = np.random.rand(5)         # 5 random numbers
print("1D Array with random values : \n", array)

# 2D Array
array = np.random.rand(3, 4)                # 3 rows , 4 cols
print("\n\n2D Array with random values : \n", array)

# Generate random floating point numbers between 2 values
print(np.random.uniform(1.5, 1.9))      # Generates a random float number between 1.5 and 1.9
print(round(rnd.uniform(1.5, 1.9), 2))   # Same but with 2 decimal places
```