NumPy

NumPy (**Num**erical **Py**thon) is an open source Python library that aids in mathematical and scientific programming.

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

```
* a powerful N-dimensional array object
* extension package to Python for multi-dimensional arrays
* designed for scientific computation -- linear algebra and random number capabilit
ies
* NumPy is incredible library to perform mathematical and statistical operations du
e to it's fast and memory efficient.
* This allows NumPy to seamlessly and speedily integrate with a wide variety of dat
abases.
```

In [1]:
```
##Import NumPy
import numpy as np
```

In [2]:
```
a=[1,2,3,4,5]
```

In [3]:
```
print(a)
```

```
[1, 2, 3, 4, 5]
```

In [4]:
```
print(type(a))
```

```
<class 'list'>
```

ndarray object

The basic ndarray is created using an array function in NumPy as follows:

```
numpy.array(object, dtype = None)
```

NOTE: In NumPy, axis = 0 is cols and axis = 1 is rows.

In [5]:
```
np_a = np.array(a)
```

In [6]:
```
print(np_a)
```

```
[1 2 3 4 5]
```

In [7]:
```
print(type(np_a))
```

```
<class 'numpy.ndarray'>
```

Why it is useful: Memory-efficient container that provides fast numerical operations.

In [8]:
```
#python lists
L = range(1000)
%timeit [i**2 for i in L]
```

```
263 µs ± 11.2 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [9]:
```
a = np.arange(1000)
%timeit a**2
```

```
1.71 µs ± 57.8 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

Creating NumPy Arrays From a List

1-D

In [10]:
```
a = np.array([0, 1, 2, 3])
print(a)
```

```
[0 1 2 3]
```

In [11]:
```
#print dimensions
a.ndim
```

Out[11]:
```
1
```

In [12]:
```
#shape
a.shape
```

Out[12]:
```
(4,)
```

2-D

In [13]:
```
b = np.array([[0, 1, 2], [3, 4, 5]])
b
```

Out[13]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [14]:
```
b.ndim
```

Out[14]:
```
2
```

In [15]:
```
b.shape
```

Out[15]:
```
(2, 3)
```

In [16]:
```
# We can convert a matrix which is also a list to array
my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
b = np.array(my_matrix)
b
```

Out[16]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [17]:
```
b.shape
```

Out[17]:
```
(3, 3)
```

3D

In [18]:
```
c = np.array([[[0, 1], [2, 3]], [[4, 5], [6, 7]]])
c
```

Out[18]:
```
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
```

In [19]:
```
c.ndim
```

Out[19]:
```
3
```

In [20]:
```
c.shape
```

Out[20]:
```
(2, 2, 2)
```

Functions for creating arrays

arange

- arange is an array-valued version of the built-in Python range function
- **Syntax:** np.arange(start,end,step)

In [21]:
```
np.arange(10, 100, 10) #start, end (exclusive), step
```

Out[21]:
```
array([10, 20, 30, 40, 50, 60, 70, 80, 90])
```

linspace

- Return evenly spaced numbers over a specified interval.
- **Syntax:** np.linspace(start, end, num_of_samples)

```
In [22]: np.linspace(0, 5, 6)
```

```
Out[22]: array([0., 1., 2., 3., 4., 5.])
```

zeros

- Generate arrays of zeros
- **Syntax:** np.zeros(shape)

```
In [23]: np.zeros(3)
```

```
Out[23]: array([0., 0., 0.])
```

```
In [24]: np.zeros((3,3))  # zero matrix or null matrix
```

```
Out[24]: array([[0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.]])
```

ones

- Generate arrays of ones
- **Syntax:** np.ones(shape)

```
In [25]: np.ones((4,3))  # unity matrix
```

```
Out[25]: array([[1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.]])
```

identity matrix

- Return a 2-D array with ones on the diagonal and zeros elsewhere.
- **Syntax:** np.eye(shape)

```
In [26]: np.eye(2)   # identity matrix
```

```
Out[26]: array([[1., 0.],
                [0., 1.]])
```

```
In [27]: np.eye(3,2)
```

```
Out[27]: array([[1., 0.],
                [0., 1.],
                [0., 0.]])
```

diagnal matrix

- Return a 2-D array with diagonal values and zeros elsewhere.
- **Syntax:** np.diag(diagnal values)

```
In [28]: a = np.diag([7, 9, 3, 4])
         a
```

```
Out[28]: array([[7, 0, 0, 0],
                [0, 9, 0, 0],
                [0, 0, 3, 0],
                [0, 0, 0, 4]])
```

```
In [29]: np.diag(a)   #Extract diagonal
```

```
Out[29]: array([7, 9, 3, 4])
```

random.randint

- Return random integers from start (inclusive) to end (exclusive)
- **Syntax:** np.random.randint(start, end, num_of_samples)

```
In [30]: np.random.randint(1,40,2)
```

```
Out[30]: array([35, 26])
```

random.rand

- Create an array with random samples from a uniform distribution over (0, 1)
- **Syntax:** np.random.rand(num_of_samples)

```
In [31]: np.random.rand(40)
```

```
Out[31]: array([0.15968365, 0.10103724, 0.74335488, 0.96066994, 0.26934805,
                0.8884551 , 0.31741055, 0.97859335, 0.61937223, 0.20354284,
                0.69652774, 0.4224069 , 0.18444385, 0.12748784, 0.75832597,
                0.27040267, 0.14500459, 0.32695968, 0.04512633, 0.51910825,
                0.97725918, 0.01549304, 0.30444066, 0.95476832, 0.84277896,
                0.35330099, 0.04976558, 0.27740822, 0.83415149, 0.10576513,
                0.35660972, 0.50923459, 0.05434971, 0.6417897 , 0.1070533 ,
                0.04990192, 0.78016956, 0.2083521 , 0.91456974, 0.03827678])
```

random.normal

- Create an array of the given input and populate it with random samples from a normal distribution.
- **Syntax:** np.random.rand(mean_value, std_value, num_of_samples)

```
In [32]: np.random.normal(10,1,100) #mean=10, std.dev=1
```

```
Out[32]: array([ 9.2519856 ,  9.88568656, 10.53069748,  9.60640501,  9.38029484,
                 8.62014416, 12.62993012, 10.05362153, 10.28989837, 10.25986058,
                11.20238987, 10.00577682,  9.27764198,  9.55022348,  9.17623389,
                11.52214179,  9.70926248, 10.64414598,  9.24480658,  9.32420587,
                 9.73279543,  9.29621687,  9.03739096, 10.98896823, 11.30835943,
                11.29882984, 11.21593227, 11.02886698, 11.42913613,  8.21073102,
                10.75264072, 12.14991599,  9.32070331,  9.86434776, 10.1276733 ,
                10.62044623, 11.27497313,  8.82817136,  9.87249989,  9.8813644 ,
                10.78280971, 10.38980641, 10.4600693 ,  9.12948334,  9.31397425,
                11.21950835,  9.41585927,  7.40326127,  9.9387721 , 11.34396186,
                11.60145401, 11.60840649, 10.73343093,  9.46060605,  8.29029835,
                 9.95457239,  9.52078355, 10.1921334 ,  9.49608252,  9.15529378,
                10.48640243, 10.05622719,  9.73819979, 10.8924576 , 11.06789028,
                10.80159032, 10.02361934,  9.55717953,  9.71773057, 11.18940336,
                 9.23418238, 10.81406215,  8.6335664 , 10.37060308, 11.94899351,
                10.1693854 ,  9.20944851,  8.2884319 , 10.85905469,  9.54971309,
                 8.62332921,  9.17037822, 10.33593733,  9.0067163 , 10.43022183,
                 9.71086052,  9.61819985,  8.80892456, 11.0464651 ,  9.52446284,
                11.17117767, 10.1346511 , 10.08155151,  8.94251017, 10.33008117,
                 9.1487683 , 11.52131465,  9.86151899, 10.16629038, 10.76639116])
```

Operations

Flattening

```
In [33]: a = np.array([[1, 2, 3], [4, 5, 6]])
         a.ravel()
```

```
Out[33]: array([1, 2, 3, 4, 5, 6])
```

Reshape array

```
In [34]: arr = np.arange(25)
         print(arr)
         print(arr.shape)
```

```
[ 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,)
```

In [35]:
```
# Notice the two sets of brackets
a1=arr.reshape(5,5)
a1
```

Out[35]:
```
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]])
```

In [36]:
```
a = np.array([[1,2,3],[4,5,6]])
print(a)
```

```
[[1 2 3]
 [4 5 6]]
```

In [37]:
```
b = a.reshape(3,2)
print(b)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

Transpose array

In [38]:
```
a = np.array([[1,2,3],[4,5,6]])
print(a)
```

```
[[1 2 3]
 [4 5 6]]
```

In [39]:
```
c = a.T
print(c)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

Copy array

In [40]:
```
a = np.array([1,2,3])
b = a
b[0] = 100
print(a)
print(b)
```

```
[100   2   3]
[100   2   3]
```

In [41]:
```
a = np.array([1,2,3])
b = a.copy()
b[0] = 100
print(a)
print(b)
```

```
[1 2 3]
[100   2   3]
```

Sort

In [42]:
```
#Sorting along an axis: axis=1 --> row and axis=0 -->column
a = np.array([[5, 4, 6], [2, 8, 2]])
b = np.sort(a, axis=0)
b
```

Out[42]:
```
array([[2, 4, 2],
       [5, 8, 6]])
```

Add/Remove operations on a numpy array

In [43]:
```
a = np.array([4,7,8])
a = np.append(a,[2,4,5])
print('Numpy array after addition:',a)
```

```
Numpy array after addition: [4 7 8 2 4 5]
```

In [44]:
```
a = np.delete(a,2)
print('Numpy array after deletion of one element:',a)
```

```
Numpy array after deletion of one element: [4 7 2 4 5]
```

Indexing

In [45]:
```
a = np.arange(3,10)
print(a)
print(a[5])
```

```
[3 4 5 6 7 8 9]
8
```

In [46]:
```
a = np.diag([1, 2, 3])
print(a)
print(a[2,1])
```

```
[[1 0 0]
 [0 2 0]
 [0 0 3]]
0
```

In [47]:
```
a[2,1] = 5 #assigning value
a
```

Out[47]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 5, 3]])
```

Indexing with an array of integers

In [48]:
```
a = np.arange(0, 101, 10)
a
```

Out[48]:
```
array([  0,  10,  20,  30,  40,  50,  60,  70,  80,  90, 100])
```

In [49]:
```
a[[0,2,3,6,10]]
```

Out[49]:
```
array([  0,  20,  30,  60, 100])
```

In [50]:
```
#Indexing can be done with an array of integers, where the same index is repea
ted several time:
a[[2, 3, 2, 4, 2]]
```

Out[50]:
```
array([20, 30, 20, 40, 20])
```

In [51]:
```
# New values can be assigned
a[[9, 7]] = -200
a
```

Out[51]:
```
array([   0,   10,   20,   30,   40,   50,   60, -200,   80, -200,  100])
```

Mask indexing or Fancy indexing

In [52]:
```
a = np.array([15,  6,  2, 16,  4])
a
```

Out[52]:
```
array([15,  6,  2, 16,  4])
```

In [53]:
```
a % 2 == 0
```

Out[53]:
```
array([False,  True,  True,  True,  True])
```

In [54]:
```
a[a%2==0]
```

Out[54]:
```
array([ 6,  2, 16,  4])
```

Slicing

In [55]:
```
a = np.arange(4,18)
a
```

Out[55]:
```
array([ 4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17])
```

In [56]:
```
a[1:8:2]      #[startindex: endindex(exclusive):step]
```

Out[56]:
```
array([ 5,  7,  9, 11])
```

Arthimetic Operations

In [57]:
```
a = np.array([1, 2, 3, 4]) #create an array
print(a)
print(a+1)
```

```
[1 2 3 4]
[2 3 4 5]
```

In [58]:
```
a ** 2
```

Out[58]:
```
array([ 1,  4,  9, 16], dtype=int32)
```

In [59]:
```
b = np.ones(4) + 1
b
```

Out[59]:
```
array([2., 2., 2., 2.])
```

In [60]:
```
a+b
```

Out[60]:
```
array([3., 4., 5., 6.])
```

In [61]:
```
a - b
```

Out[61]:
```
array([-1.,  0.,  1.,  2.])
```

In [62]:
```
a * b
```

Out[62]:
```
array([2., 4., 6., 8.])
```

In [63]:
```
a = np.array([[1,2],[3,4],[5,6]])
b = np.array([[4,4],[4,4],[4,4]])

# adding a and b
print(a+b)
```

```
[[ 5  6]
 [ 7  8]
 [ 9 10]]
```

In [64]:
```
# multiply a and b - elementwise multiplication
print(a*b)
```

```
[[ 4  8]
 [12 16]
 [20 24]]
```

In [65]:
```
# matrix multiplication
a = np.array([[1,2,1],[3,4,3],[5,6,5]])
b = np.array([[4,4,1],[4,4,2],[4,4,3]])
print('Matrix multiplication:\n',np.matmul(a,b))
```

```
Matrix multiplication:
 [[16 16  8]
 [40 40 20]
 [64 64 32]]
```

Comparision operations

In [66]:
```
a = np.array([1, 2, 3, 4])
b = np.array([5, 2, 2, 4])
a == b
```

Out[66]:
```
array([False,  True, False,  True])
```

In [67]:
```
a > b
```

Out[67]:
```
array([False, False,  True, False])
```

In [68]:
```
#array-wise comparisions
a = np.array([1, 2, 3, 4])
b = np.array([5, 2, 2, 4])
c = np.array([1, 2, 3, 4])
```

In [69]:
```
np.array_equal(a, b)
```

Out[69]:
```
False
```

In [70]:
```
np.array_equal(a, c)
```

Out[70]:
```
True
```

Transcendental operations

In [71]:
```
a = np.arange(0,6)
np.sin(a)
```

Out[71]:
```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427])
```

In [72]:
```
np.log(a)
```

Out[72]:
```
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436,
       1.60943791])
```

In [73]:
```
np.exp(a)   #evaluates e^x for each element in a given input
```

Out[73]:
```
array([  1.        ,   2.71828183,   7.3890561 ,  20.08553692,
        54.59815003, 148.4131591 ])
```

Statistics

In [74]:
```
array = np.arange(1,11)
print(array)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

In [75]:
```
# Min
np.min(array)
```

Out[75]:
```
1
```

In [76]:
```
# index of minimum element
np.argmin(array)
```

Out[76]:
```
0
```

In [77]:
```
# Max
np.max(array)
```

Out[77]:
```
10
```

In [78]:
```
# index of maximum element
np.argmax(array)
```

Out[78]:
```
9
```

In [79]:
```
# Sum
np.sum(array)
```

Out[79]:
```
55
```

In [80]:
```
# Mean
np.mean(array)
```

Out[80]:
```
5.5
```

In [81]:
```
# Median
np.median(array)
```

Out[81]: 5.5

In [82]:
```
# variance
np.var(array)
```

Out[82]: 8.25

In [83]:
```
# Standard deviation
np.std(array)
```

Out[83]: 2.8722813232690143

In [84]:
```
# percentile
np.percentile(array,25)
```

Out[84]: 3.25

In [85]:
```
x = np.array([[1, 1], [2, 2]])
x
```

Out[85]:
```
array([[1, 1],
       [2, 2]])
```

In [86]:
```
x.sum(axis=0)   #column sum
```

Out[86]: array([3, 3])

In [87]:
```
x.sum(axis=1)   #row sum
```

Out[87]: array([2, 4])