Attributes & Function in Numpy

In [1]:

```
#pip install numpy
```

In [3]:

```
import numpy as np
```

In [4]:

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

In [5]:

```
x
```

Out[5]:

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

In [6]:

```
type(x)
```

Out[6]:

```
list
```

In [7]:

```
a = np.array(x)
```

In [8]:

```
a
```

Out[8]:

```
array([1, 2, 3, 4, 5])
```

In [9]:

```
type(a)
```

Out[9]:

```
numpy.ndarray
```

Creating a numpy array of 2D

In [10]:

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

Out[10]:

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

Attributes & Methods of Numpy

1. Shape -: return the shape of an array

In [11]:

```
a.shape  #row, col inside a tuple
```

Out[11]:

```
(2, 3)
```

2.size -: returns the total number of elements present in an array

In [12]:

```
a.size
```

Out[12]:

```
6
```

3.Transpose -: returns an by array by swaping the row into col & viz

In [13]:

```
a
```

Out[13]:

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

In [14]:

```
a.T
```

Out[14]:

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

4.NDim -: return the number of dimensions an array

In [15]:

```
a
```

Out[15]:

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

In [16]:

```
a.ndim
```

Out[16]:

```
2
```

5. Reshape-: used to change the shape of an array

In [17]:

```
a
```

Out[17]:

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

In [18]:

```
a.shape
```

Out[18]:

```
(2, 3)
```

In [19]:

```
a.reshape(1,6)
```

Out[19]:

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

In [20]:

```
a.reshape(3,2)
```

Out[20]:

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

In [21]:

```
a
```

Out[21]:

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

In [22]:

```
a= a.reshape(3,2)
```

In [23]:

```
a
```

Out[23]:

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

In [24]:

```
a.shape
```

Out[24]:

```
(3, 2)
```

In [25]:

```
a.shape = (2, 3)
```

In [26]:

```
a
```

Out[26]:

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

Functions in Numpy

In [27]:

```
np.array([1,2,3,4,5,6]).reshape(2, 3)
```

Out[27]:

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

In [28]:

```
a = np.array([1,2,3,4,5,6]).reshape(2, 3)
b = np.array([7,8,9,10,11,12]).reshape(2,3)
```

In [29]:

```
a
```

Out[29]:

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

In [30]:

```
b
```

Out[30]:

```
array([[ 7,  8,  9],
       [10, 11, 12]])
```

1. Concatenate((arr1, arr2)) -: used to join two different array together

In [31]:

```
np.concatenate((a,b))
```

Out[31]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

2. Vstack((arr1, arr2)) -: used to join two different array together vertically

In [32]:

```
np.vstack((a,b))
```

Out[32]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

3. Hstack((arr1, arr2)) -: used to join two different array together horizontally

In [33]:

```
np.hstack((a,b))
```

Out[33]:

```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

In [34]:

```
c = np.array([21, 22, 23, 24, 25, 26 ,26])
```

In [35]:

```
c
```

Out[35]:

```
array([21, 22, 23, 24, 25, 26, 26])
```

4. Append(arr, listofelement) -: used to add the elements in an array

In [36]:

```
np.append(c, [27, 28, 29])
```

Out[36]:

```
array([21, 22, 23, 24, 25, 26, 26, 27, 28, 29])
```

5. Insert(arr, index, element) -: used to add the elemenet in a specified position

In [37]:

```
c
```

Out[37]:

```
array([21, 22, 23, 24, 25, 26, 26])
```

In [38]:

```
np.insert(c, 3, [1,2,3])
```

Out[38]:

```
array([21, 22, 23,  1,  2,  3, 24, 25, 26, 26])
```

6. Delete(arr, listofelementindexvalue) -: used to delete the elements of specified index

In [39]:

```
c
```

Out[39]:

```
array([21, 22, 23, 24, 25, 26, 26])
```

In [40]:

```
np.delete(c,[-1])
```

Out[40]:

```
array([21, 22, 23, 24, 25, 26])
```

In [41]:

```
c
```

Out[41]:

```
array([21, 22, 23, 24, 25, 26, 26])
```

In [42]:

```
np.delete(c, [1, 2])
```

Out[42]:

```
array([21, 24, 25, 26, 26])
```

Array Creation Routine in Numpy

In [2]:

```
import numpy as np
```

1. np.empty -: It creates an uninitialise array of specified shape with random values ¶

In [3]:

```
np.empty(5)
```

Out[3]:

```
array([2.12199579e-314, 1.17770766e-311, 4.62445445e-321, 2.88591428e-312,
       1.17885383e-311])
```

In [4]:

```
np.empty((3,3))
```

Out[4]:

```
array([[0.00000000e+000, 0.00000000e+000, 0.00000000e+000],
       [0.00000000e+000, 0.00000000e+000, 3.69561103e-321],
       [1.05699242e-307, 8.01097888e-307, 0.00000000e+000]])
```

In [5]:

```
np.empty((3,3), dtype=int)
```

Out[5]:

```
array([[      0,       1,       0],
       [5111897,     936,       0],
       [    768,       0, 7602288]])
```

2. np.zeros - It returns a new array of specified shape, filled with zeros

In [6]:

```
np.zeros(5)
```

Out[6]:

```
array([0., 0., 0., 0., 0.])
```

In [7]:

```
np.zeros((3,3), dtype=int)
```

Out[7]:

```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

3. np.ones - It returns a new array of specified shape, filled with ones

In [8]:

```
np.ones(10)
```

Out[8]:

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [9]:

```
np.ones((3,3))
```

Out[9]:

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

4. np.eye - it return an identity matrix which is a square matrix

In [10]:

```
np.eye(3)
```

Out[10]:

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [11]:

```
np.eye(2)
```

Out[11]:

```
array([[1., 0.],
       [0., 1.]])
```

5. arange -: It returns an evenly spaced numbers just like python range function

In [12]:

```
list(range(1, 11))
```

Out[12]:

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

In [13]:

```
np.arange(1,11) #start: inclusive, end: exclusive
```

Out[13]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [14]:

```
np.arange(1, 11, 2)
```

Out[14]:

```
array([1, 3, 5, 7, 9])
```

In [15]:

```
np.arange(1,7).reshape(2,3)
```

Out[15]:

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

6. np.linspace - This is a function similar to arange, instead of step size, the number of evenly spaced values between an interval is specified

In [16]:

```
np.linspace(11, 20, 10)
```

Out[16]:

```
array([11., 12., 13., 14., 15., 16., 17., 18., 19., 20.])
```

In [17]:

```
np.linspace(11, 20, 20)
```

Out[17]:

```
array([11.        , 11.47368421, 11.94736842, 12.42105263, 12.89473684,
       13.36842105, 13.84210526, 14.31578947, 14.78947368, 15.26315789,
       15.73684211, 16.21052632, 16.68421053, 17.15789474, 17.63157895,
       18.10526316, 18.57894737, 19.05263158, 19.52631579, 20.        ])
```

In [18]:

```
11.47368421 - 11
```

Out[18]:

```
0.47368421000000005
```

In [19]:

```
 11.94736842 -   11.47368421
```

Out[19]:

```
0.47368421000000005
```

In [20]:

```
np.linspace(0, 1, 100)
```

Out[20]:

```
array([0.        , 0.01010101, 0.02020202, 0.03030303, 0.04040404,
       0.05050505, 0.06060606, 0.07070707, 0.08080808, 0.09090909,
       0.1010101 , 0.11111111, 0.12121212, 0.13131313, 0.14141414,
       0.15151515, 0.16161616, 0.17171717, 0.18181818, 0.19191919,
       0.2020202 , 0.21212121, 0.22222222, 0.23232323, 0.24242424,
       0.25252525, 0.26262626, 0.27272727, 0.28282828, 0.29292929,
       0.3030303 , 0.31313131, 0.32323232, 0.33333333, 0.34343434,
       0.35353535, 0.36363636, 0.37373737, 0.38383838, 0.39393939,
       0.4040404 , 0.41414141, 0.42424242, 0.43434343, 0.44444444,
       0.45454545, 0.46464646, 0.47474747, 0.48484848, 0.49494949,
       0.50505051, 0.51515152, 0.52525253, 0.53535354, 0.54545455,
       0.55555556, 0.56565657, 0.57575758, 0.58585859, 0.5959596 ,
       0.60606061, 0.61616162, 0.62626263, 0.63636364, 0.64646465,
       0.65656566, 0.66666667, 0.67676768, 0.68686869, 0.6969697 ,
       0.70707071, 0.71717172, 0.72727273, 0.73737374, 0.74747475,
       0.75757576, 0.76767677, 0.77777778, 0.78787879, 0.7979798 ,
       0.80808081, 0.81818182, 0.82828283, 0.83838384, 0.84848485,
       0.85858586, 0.86868687, 0.87878788, 0.88888889, 0.8989899 ,
       0.90909091, 0.91919192, 0.92929293, 0.93939394, 0.94949495,
       0.95959596, 0.96969697, 0.97979798, 0.98989899, 1.        ])
```

7. np.random.rand - returns an array of specified shape with random values between 0-1

In [21]:

```
np.random.rand(10)
```

Out[21]:

```
array([0.59404774, 0.83296271, 0.78496044, 0.93988312, 0.53707497,
       0.66061446, 0.02276956, 0.52204954, 0.94821119, 0.99877182])
```

In [22]:

```
np.random.rand(3,3)
```

Out[22]:

```
array([[0.28660907, 0.61206151, 0.05638213],
       [0.12222886, 0.72693016, 0.97313313],
       [0.20421363, 0.47756959, 0.7623363 ]])
```

8. np.random.randn - returns an array of spacified shape with random values between -ve to +ve

In [23]:

```
np.random.randn(10)
```

Out[23]:

```
array([-0.35135465,  1.72366338, -1.34133507,  0.48316616,  0.70087154,
       -1.06884536, -0.12996615, -0.16462106, -0.91187202,  0.14018605])
```

In [24]:

```
np.random.randn(3,3)
```

Out[24]:

```
array([[-0.19036406,  1.79072348, -0.51879149],
       [-1.06205004, -0.43224411, -1.50662694],
       [ 0.58380694,  0.19318176,  0.35828306]])
```

Slicing & Indexing of an array

In [2]:

```
import numpy as np
```

In [3]:

```
a =np.arange(21, 31)
a
```

Out[3]:

```
array([21, 22, 23, 24, 25, 26, 27, 28, 29, 30])
```

In [4]:

```
a[2]
```

Out[4]:

```
23
```

In [5]:

```
a[-1]
```

Out[5]:

```
30
```

In [6]:

```
a[3: 6]
```

Out[6]:

```
array([24, 25, 26])
```

Indexing & Slicing on 2D array

In [7]:

```
a = np.arange(1,10).reshape(3,3)
```

In [8]:

```
a
```

Out[8]:

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

In [9]:

```
#index -> 4 => var[row, col]
a[1, 0]
```

Out[9]:

```
4
```

In [10]:

```
a[1, 1]
```

Out[10]:

```
5
```

In [11]:

```
a[0, 2]
```

Out[11]:

```
3
```

In [12]:

```
#slicing => var[startrow: endrow, startcol: endcol]
#[[4,5],
#[7,8]]
a[1:, 0:2]
```

Out[12]:

```
array([[4, 5],
       [7, 8]])
```

In [13]:

```
a = np.arange(1,17).reshape(4,4)
a
```

Out[13]:

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

In [14]:

```
#[[6,7],
#[10,11]]

a[1:3, 1:3]
```

Out[14]:

```
array([[ 6,  7],
       [10, 11]])
```

In [15]:

```
#[[11, 12],
#[15, 16]]

a[2:, 2:]
```

Out[15]:

```
array([[11, 12],
       [15, 16]])
```

In [16]:

```
a[0:2, 1:3]
```

Out[16]:

```
array([[2, 3],
       [6, 7]])
```

Tip

In [17]:

```
a = np.arange(1,11)
```

In [18]:

```
a
```

Out[18]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [19]:

```
a[0:5]
```

Out[19]:

```
array([1, 2, 3, 4, 5])
```

In [20]:

```
a[:5] #start is empty, this mean start from the begining, i.e 0
```

Out[20]:

```
array([1, 2, 3, 4, 5])
```

In [21]:

```
a[5: ] #end is empty, this mean go till the end i.e include the very last element of the li
```

Out[21]:

```
array([ 6,  7,  8,  9, 10])
```

Vectorization & Selection in an array

In [1]:

```
x = list(range(1,11))
x
```

Out[1]:

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

In [2]:

```
x*2 #replication
```

Out[2]:

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

In [4]:

```
import numpy as np
```

In [5]:

```
a = np.arange(1,11)
a
```

Out[5]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [6]:

```
a*2
```

Out[6]:

```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [7]:

```
a
```

Out[7]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [8]:

```
a+100
```

Out[8]:

```
array([101, 102, 103, 104, 105, 106, 107, 108, 109, 110])
```

In [9]:

```
a
```

Out[9]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [10]:

```
a-2
```

Out[10]:

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

In [11]:

```
a
```

Out[11]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [12]:

```
a/10
```

Out[12]:

```
array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])
```

In [13]:

```
a/2
```

Out[13]:

```
array([0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
```

Vectorization with comparison operators

In [14]:

```
a
```

Out[14]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [15]:

```
3>4
```

Out[15]:

```
False
```

In [16]:

```
a>4
```

Out[16]:

```
array([False, False, False, False,  True,  True,  True,  True,  True,
        True])
```

In [17]:

```
a
```

Out[17]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [18]:

```
a%2==0
```

Out[18]:

```
array([False,  True, False,  True, False,  True, False,  True, False,
        True])
```

masking

In [19]:

```
a
```

Out[19]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [20]:

```
#what are all those numbers which are greater then 4
a>4
```

Out[20]:

```
array([False, False, False, False,  True,  True,  True,  True,  True,
        True])
```

In [21]:

```
mask = a>4
```

In [22]:

```
a[mask]
```

Out[22]:

```
array([ 5,  6,  7,  8,  9, 10])
```

In [23]:

```
#what are the even number in my array
a%2==0
```

Out[23]:

```
array([False,  True, False,  True, False,  True, False,  True, False,
        True])
```

In [24]:

```
mask = a%2==0
a[mask]
```

Out[24]:

```
array([ 2,  4,  6,  8, 10])
```

- vectorisation with comparison operators will give you the boolean value i.e True or False
- Masking will give you the actual value for which the condition was True

Selection in array

In [25]:

```
names =np.array(["jay", "raj", "jay", "kumar", "mrityunjay", "kumar", "jay", "rohit", "sura
```

In [26]:

```
#select all the names, where the value is jay
names=="jay"
```

Out[26]:

```
array([ True, False,  True, False, False, False,  True, False, False,
       False])
```

In [27]:

```
mask = names=="jay"
```

In [28]:

```
names[mask]
```

Out[28]:

```
array(['jay', 'jay', 'jay'], dtype='<U10')
```

In [29]:

```
names[names=="jay"]
```

Out[29]:

```
array(['jay', 'jay', 'jay'], dtype='<U10')
```

In [30]:

```
#select all the names, where the values is not jay
names[names!="jay"]
```

Out[30]:

```
array(['raj', 'kumar', 'mrityunjay', 'kumar', 'rohit', 'suraj', 'ravi'],
      dtype='<U10')
```

In [31]:

```
names
```

Out[31]:

```
array(['jay', 'raj', 'jay', 'kumar', 'mrityunjay', 'kumar', 'jay',
       'rohit', 'suraj', 'ravi'], dtype='<U10')
```

In [32]:

```
#select all the names except jay and kumar
cond1= (names!="jay")
cond2= (names!="kumar")
```

In [33]:

```
#=> and operator
#here in numpy
#and => &
#or  => |
```

In [34]:

```
cond1 & cond2
```

Out[34]:

```
array([False,  True, False, False,  True, False, False,  True,  True,
        True])
```

In [35]:

```
names[cond1 & cond2]
```

Out[35]:

```
array(['raj', 'mrityunjay', 'rohit', 'suraj', 'ravi'], dtype='<U10')
```

In [36]:

```
names[(names!="jay")&(names!="kumar")]
```

Out[36]:

```
array(['raj', 'mrityunjay', 'rohit', 'suraj', 'ravi'], dtype='<U10')
```

In [37]:

```
#select all those naems where name is jay and kumar
names[(names=="jay")|(names=="kumar")]
```

Out[37]:

```
array(['jay', 'jay', 'kumar', 'kumar', 'jay'], dtype='<U10')
```

In [38]:

```
#select all the values which is not mrityunjay, and assign then as shiv
#step1

names!="mrityunjay"
```

Out[38]:

```
array([ True,  True,  True,  True, False,  True,  True,  True,  True,
        True])
```

In [39]:

```
#step2
names[names!="mrityunjay"]
```

Out[39]:

```
array(['jay', 'raj', 'jay', 'kumar', 'kumar', 'jay', 'rohit', 'suraj',
       'ravi'], dtype='<U10')
```

In [40]:

```
#step3
names[names!="mrityunjay"] = "shiv"
```

In [41]:

```
names
```

Out[41]:

```
array(['shiv', 'shiv', 'shiv', 'shiv', 'mrityunjay', 'shiv', 'shiv',
       'shiv', 'shiv', 'shiv'], dtype='<U10')
```

In [42]:

```
np.unique(names)
```

Out[42]:

```
array(['mrityunjay', 'shiv'], dtype='<U10')
```

Summary

In [43]:

```
a #array
```

Out[43]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [44]:

```
a*2 #vectorization with arithmetic operator
```

Out[44]:

```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [45]:

```
a>5 #vectorization with comparison operator
```

Out[45]:

```
array([False, False, False, False, False,  True,  True,  True,  True,
        True])
```

In [46]:

```
a[a>5] #masking
```

Out[46]:

```
array([ 6,  7,  8,  9, 10])
```

Broadcasting & Fancy Indexing in an Array

In [2]:

```
a = np.arange(1, 11)
a
```

Out[2]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [3]:

```
a[3:6]
```

Out[3]:

```
array([4, 5, 6])
```

In [4]:

```
b = a[3:6] #data is not copied, we are passing the reference
```

In [5]:

```
b
```

Out[5]:

```
array([4, 5, 6])
```

In [6]:

```
b[:] = 100
```

In [7]:

```
b
```

Out[7]:

```
array([100, 100, 100])
```

In [8]:

```
a
```

Out[8]:

```
array([  1,   2,   3, 100, 100, 100,   7,   8,   9,  10])
```

In [10]:

```
c = a[3:6].copy()
```

In [11]:

```
c
```

Out[11]:

```
array([100, 100, 100])
```

In [12]:

```
c[:] = 0
```

In [13]:

```
c
```

Out[13]:

```
array([0, 0, 0])
```

In [14]:

```
a
```

Out[14]:

```
array([  1,   2,   3, 100, 100, 100,   7,   8,   9,  10])
```

Fancy Indexing

In [15]:

```
arr = np.zeros((10,10))
arr
```

Out[15]:

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

In [16]:

```
arrlen = arr.shape[1]
arrlen
```

Out[16]:

```
10
```

In [17]:

```
for i in range(arrlen):
    arr[i]=i

arr
```

Out[17]:

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

In [18]:

```
arr[[3, 4, 5]]
```

Out[18]:

```
array([[3., 3., 3., 3., 3., 3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [5., 5., 5., 5., 5., 5., 5., 5., 5., 5.]])
```

In [19]:

```
arr[[2, 4, 6, 8]]
```

Out[19]:

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

Arithmetical & Statistical Operations on Numpy Array

In [2]:

```
import numpy as np
```

In [3]:

```
a = np.array([[1, 5, 6], [1, 8, 9], [0, -1, 6]])
b = np.array([[4, 8, 4], [1, 0, 5], [6, -8, 3]])
```

In [4]:

```
a
```

Out[4]:

```
array([[ 1,  5,  6],
       [ 1,  8,  9],
       [ 0, -1,  6]])
```

In [5]:

```
b
```

Out[5]:

```
array([[ 4,  8,  4],
       [ 1,  0,  5],
       [ 6, -8,  3]])
```

Arithmetics on numpy

In [6]:

```
np.add(a,b)
```

Out[6]:

```
array([[ 5, 13, 10],
       [ 2,  8, 14],
       [ 6, -9,  9]])
```

In [7]:

```
np.subtract(a,b)
```

Out[7]:

```
array([[-3, -3,  2],
       [ 0,  8,  4],
       [-6,  7,  3]])
```

In [8]:

```
np.multiply(a,b)
```

Out[8]:

```
array([[ 4, 40, 24],
       [ 1,  0, 45],
       [ 0,  8, 18]])
```

In [9]:

```
np.divide(a,b)
```

```
<ipython-input-9-c364992e28ce>:1: RuntimeWarning: divide by zero encountered
in true_divide
  np.divide(a,b)
```

Out[9]:

```
array([[0.25 , 0.625, 1.5  ],
       [1.   ,   inf, 1.8  ],
       [0.   , 0.125, 2.   ]])
```

In [10]:

```
np.remainder(a,b)
```

```
<ipython-input-10-75e769c51b49>:1: RuntimeWarning: divide by zero encountere
d in remainder
  np.remainder(a,b)
```

Out[10]:

```
array([[ 1,  5,  2],
       [ 0,  0,  4],
       [ 0, -1,  0]], dtype=int32)
```

Matrics product

In [11]:

```
#dot product or inner product i.e 1st row to 1st col and so on...
np.matmul(a,b)
```

Out[11]:

```
array([[ 45, -40,  47],
       [ 66, -64,  71],
       [ 35, -48,  13]])
```

In [12]:

```
np.kron(a,b)
```

Out[12]:

```
array([[  4,   8,   4,  20,  40,  20,  24,  48,  24],
       [  1,   0,   5,   5,   0,  25,   6,   0,  30],
       [  6,  -8,   3,  30, -40,  15,  36, -48,  18],
       [  4,   8,   4,  32,  64,  32,  36,  72,  36],
       [  1,   0,   5,   8,   0,  40,   9,   0,  45],
       [  6,  -8,   3,  48, -64,  24,  54, -72,  27],
       [  0,   0,   0,  -4,  -8,  -4,  24,  48,  24],
       [  0,   0,   0,  -1,   0,  -5,   6,   0,  30],
       [  0,   0,   0,  -6,   8,  -3,  36, -48,  18]])
```

In [13]:

```
arr = np.arange(6)
```

In [14]:

```
arr
```

Out[14]:

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

In [15]:

```
arr*10
```

Out[15]:

```
array([ 0, 10, 20, 30, 40, 50])
```

In [16]:

```
np.sqrt(arr)
```

Out[16]:

```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798])
```

In [17]:

```
nums = np.sqrt(arr)
```

In [18]:

```
np.ceil(nums)
```

Out[18]:

```
array([0., 1., 2., 2., 2., 3.])
```

In [19]:

```
np.floor(nums)
```

Out[19]:

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

Statistical Operations

In [20]:

```
arr = np.random.randn(4,2)
arr
```

Out[20]:

```
array([[ 0.10025631, -0.4894232 ],
       [ 0.34079317, -1.35955441],
       [-1.4908254 ,  0.95162809],
       [-1.14273089, -1.06812343]])
```

In [21]:

```
arr.mean()
```

Out[21]:

```
-0.5197474700467734
```

In [22]:

```
arr.std()
```

Out[22]:

```
0.8387410993119755
```

In [23]:

```
arr.max()
```

Out[23]:

```
0.9516280946524784
```

In [24]:

```
arr.argmax()
```

Out[24]:

```
5
```

In [25]:

```
arr.min()
```

Out[25]:

```
-1.4908253968685017
```

In [26]:

```
arr.argmin()
```

Out[26]:

```
4
```

In [27]:

```
arr
```

Out[27]:

```
array([[ 0.10025631, -0.4894232 ],
       [ 0.34079317, -1.35955441],
       [-1.4908254 ,  0.95162809],
       [-1.14273089, -1.06812343]])
```

In [28]:

```
arr.sum()
```

Out[28]:

```
-4.157979760374187
```

In [29]:

```
arr.sum(axis=0)
```

Out[29]:

```
array([-2.19250681, -1.96547295])
```

In [30]:

```
arr.sum(axis=1)
```

Out[30]:

```
array([-0.3891669 , -1.01876124, -0.5391973 , -2.21085432])
```

In [31]:

```
arr
```

Out[31]:

```
array([[ 0.10025631, -0.4894232 ],
       [ 0.34079317, -1.35955441],
       [-1.4908254 ,  0.95162809],
       [-1.14273089, -1.06812343]])
```

In [32]:

```
arr.flatten()
```

Out[32]:

```
array([ 0.10025631, -0.4894232 ,  0.34079317, -1.35955441, -1.4908254 ,
        0.95162809, -1.14273089, -1.06812343])
```

Computing Euclidean Distance between 2 vectors

In [33]:

```
vec1 = np.random.randn(3,3)
vec2 = np.random.randn(3,3)
```

In [34]:

```
vec1
```

Out[34]:

```
array([[-0.64890736, -2.79620102,  1.46321099],
       [ 0.16062447,  1.05830729,  0.56010725],
       [ 0.34757447, -0.65325639,  1.44400429]])
```

In [35]:

```
vec2
```

Out[35]:

```
array([[-3.18750322, -0.83745151,  0.49355973],
       [ 0.19610056, -0.32092387, -0.61016066],
       [ 0.24307821, -1.3587111 ,  1.67535366]])
```

In [36]:

```
dist = np.sqrt(np.sum(vec1-vec2)**2)
```

In [37]:

```
dist
```

Out[37]:

```
4.642122191766501
```

Trigonometric Functions

In [38]:

```
np.sin(90)
```

Out[38]:

```
0.8939966636005579
```

In [39]:

```
np.cos(90)
```

Out[39]:

```
-0.4480736161291701
```

In [40]:

```
np.tan(30)
```

Out[40]:

```
-6.405331196646276
```

In [41]:

```
np.log(10)
```

Out[41]:

```
2.302585092994046
```

In [42]:

```
np.log(1)
```

Out[42]:

```
0.0
```

In [43]:

```
np.log10(10)
```

Out[43]:

```
1.0
```

In [44]:

```
np.log(np.log10(5))
```

Out[44]:

```
-0.35814744992084513
```