In [1]:

```
my_list=[1,2,3,4]
```

In [2]:

```
my_list
```

Out[2]:

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

In [3]:

```
import numpy as np
```

In [4]:

```
arr=np.array(my_list)   #converting the list into array
```

In [5]:

```
print(arr)
```

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

In [6]:

```
arr
```

Out[6]:

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

In [7]:

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

In [8]:

```
my_mat
```

Out[8]:

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

In [9]:

```
print(my_mat)
```

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

In [10]:

```
twoD_arr=np.array(my_mat)
```

In [11]:

```
twoD_arr
```

Out[11]:

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

In [12]:

```
print(twoD_arr)
```

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

In [13]:

```
np.arange(1,12)    #array of element betwwen 1 and 12 where 1 is inclusive and 12 is exclu
sive
```

Out[13]:

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

In [14]:

```
np.zeros(4)
```

Out[14]:

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

In [15]:

```
np.zeros((4,5))
```

Out[15]:

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

In [16]:

```
print(np.ones(5))
```

```
[1. 1. 1. 1. 1.]
```

In [17]:

```
np.ones((5,6))
```

Out[17]:

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

In [18]:

```
np.linspace(0,5,10)
```

Out[18]:

```
array([0.        , 0.55555556, 1.11111111, 1.66666667, 2.22222222,
       2.77777778, 3.33333333, 3.88888889, 4.44444444, 5.        ])
```

In [19]:

```
np.linspace(0,5,100)    #100 equi distanced element betwwen 0 and 5
```

Out[19]:

```
array([0.        , 0.05050505, 0.1010101 , 0.15151515, 0.2020202 ,
       0.25252525, 0.3030303 , 0.35353535, 0.4040404 , 0.45454545,
       0.50505051, 0.55555556, 0.60606061, 0.65656566, 0.70707071,
       0.75757576, 0.80808081, 0.85858586, 0.90909091, 0.95959596,
       1.01010101, 1.06060606, 1.11111111, 1.16161616, 1.21212121,
       1.26262626, 1.31313131, 1.36363636, 1.41414141, 1.46464646,
       1.51515152, 1.56565657, 1.61616162, 1.66666667, 1.71717172,
       1.76767677, 1.81818182, 1.86868687, 1.91919192, 1.96969697,
       2.02020202, 2.07070707, 2.12121212, 2.17171717, 2.22222222,
       2.02020202, 2.07070707, 2.12121212, 2.17171717, 2.22222222,
       2.27272727, 2.32323232, 2.37373737, 2.42424242, 2.47474747,
       2.52525253, 2.57575758, 2.62626263, 2.67676768, 2.72727273,
       2.77777778, 2.82828283, 2.87878788, 2.92929293, 2.97979798,
       3.03030303, 3.08080808, 3.13131313, 3.18181818, 3.23232323,
       3.28282828, 3.33333333, 3.38383838, 3.43434343, 3.48484848,
       3.53535354, 3.58585859, 3.63636364, 3.68686869, 3.73737374,
       3.78787879, 3.83838384, 3.88888889, 3.93939394, 3.98989899,
       4.04040404, 4.09090909, 4.14141414, 4.19191919, 4.24242424,
       4.29292929, 4.34343434, 4.39393939, 4.44444444, 4.49494949,
       4.54545455, 4.5959596 , 4.64646465, 4.6969697 , 4.74747475,
       4.7979798 , 4.84848485, 4.8989899 , 4.94949495, 5.        ])
```

In [20]:

```
np.eye(3)    #identy matrix
```

Out[20]:

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

In [21]:

```
np.random.rand(5)      #find the array of 5 random numbers linearlly distributed
```

Out[21]:

```
array([0.00322549, 0.64072628, 0.20687644, 0.87432955, 0.13757461])
```

In [22]:

```
np.random.rand(4,5)    #find the 2 D array of linearlly distributed 4x5=20 random number
```

Out[22]:

```
array([[0.84693275, 0.00818016, 0.14767013, 0.89050774, 0.87618785],
       [0.47981192, 0.01904577, 0.38487578, 0.81589291, 0.95664793],
       [0.28033651, 0.69500251, 0.6315152 , 0.27608121, 0.55838491],
       [0.85914009, 0.06852554, 0.20260292, 0.96825048, 0.64386797]])
```

In [23]:

```
np.random.randn(7)
```

Out[23]:

```
array([ 1.52887664,  0.34529595,  2.08754438, -1.25327819, -1.11562492,
       -0.30310816, -0.04395607])
```

In [24]:

```
rand2D_array=np.random.randn(5,6)
print(rand2D_array)
```

```
[[ 0.10505786 -1.23732482 -1.60871883 -0.75039738  0.13687532  0.68230716]
 [ 0.65787396  0.30227727  1.31788249 -1.16457079  1.88240877  0.71457015]
 [ 0.33306318  0.18480131  0.25439252  1.25638308  1.71588084  1.26124861]
 [-0.15422166  0.06345717 -0.71190211  0.72899991 -2.33561417  0.94892255]
 [-0.15162084  0.63766634 -0.3004974  -0.96106249  0.4484168  -1.29364397]]
```

In [25]:

```
np.random.randint(1,100,10)   #array of 10 random element between 1 and 100
```

Out[25]:

```
array([87, 72, 28, 14, 25, 85, 70, 95, 64,  5])
```

In [26]:

```
arr1=np.arange(30)
```

In [27]:

```
arr1
```

Out[27]:

```
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 [28]:

```
arr1.reshape(5,6)    # reshape the array with exact numbe of element in original array
```

Out[28]:

```
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 [29]:

```
arr1.reshape(3,4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-29-ed6e303fc59c> in <module>
----> 1 arr1.reshape(3,4)

ValueError: cannot reshape array of size 30 into shape (3,4)
```

In [30]:

```
ranarr1=np.random.randint(1,50,10)
```

In [31]:

```
ranarr1
```

Out[31]:

```
array([ 4, 19, 12, 12, 30,  5, 41, 26, 16, 36])
```

In [32]:

```
ranarr1.max()
```

Out[32]:

```
41
```

In [33]:

```
ranarr1.min()
```

Out[33]:

```
4
```

In [34]:

```
ranarr1.argmax()    #find the location of max value
```

Out[34]:

```
6
```

In [35]:

```
ranarr1.argmin()   #find the location of min value
```

Out[35]:

```
0
```

In [36]:

```
arr1.shape   #return the sahpe of vector.. here it is arr1 is 1-D vector
```

Out[36]:

```
(30,)
```

In [37]:

```
arr1=arr1.reshape(5,6)
```

In [38]:

```
arr1.shape
```

Out[38]:

```
(5, 6)
```

In [39]:

```
arr1.dtype    #return the data type of array
```

Out[39]:

```
dtype('int32')
```

Indexing and selection in NumPy

In [40]:

```
arr1
```

Out[40]:

```
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 [41]:

```
arr1=arr1.reshape(30)
```

In [42]:

```
arr1
```

Out[42]:

```
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 [43]:

```
arr1[8]        #[] with index number return the value at that index
```

Out[43]:

```
8
```

In [44]:

```
arr1[1:8]    # start and end index mentioned where last is not inclusive
```

Out[44]:

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

In [45]:

```
arr1[:10]
```

Out[45]:

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

In [46]:

```
arr1[3:]
```

Out[46]:

```
array([ 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 [47]:

```
arr1[0:3]= 99   #broadcast
```

In [48]:

```
arr1
```

Out[48]:

```
array([99, 99, 99,  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 [50]:

```
arr1=np.arange(30)    #reset
arr1
```

Out[50]:

```
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 [51]:

```
slice_of_arr1=arr1[0:7]     #creating the sub array using slicing
```

In [52]:

```
slice_of_arr1
```

Out[52]:

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

In [53]:

```
slice_of_arr1[:]=80   #broadcasting
```

In [54]:

```
slice_of_arr1
```

Out[54]:

```
array([80, 80, 80, 80, 80, 80, 80])
```

In [55]:

```
arr1      #values in original arr1 are also changed; so data is not copied, numpy does no
t keep the copies of array
```

Out[55]:

```
array([80, 80, 80, 80, 80, 80, 80,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])
```

In [56]:

```
arr1_copy=arr1.copy()    # specifically creating a copy of array using copy method
```

In [57]:

```
arr1_copy
```

Out[57]:

```
array([80, 80, 80, 80, 80, 80, 80,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])
```

In [58]:

```
arr1
```

Out[58]:

```
array([80, 80, 80, 80, 80, 80, 80,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])
```

In [59]:

```
arr1_copy[:]=100
```

In [60]:

```
arr1_copy
```

Out[60]:

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

In [61]:

```
arr1         #arr1 is now unaffected
```

Out[61]:

```
array([80, 80, 80, 80, 80, 80, 80,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])
```

Indexing in 2-D array

In [62]:

```
arr_2d=np.array([[5,10,15],[20,25,30],[35,40,45]])
```

In [63]:

```
arr_2d
```

Out[63]:

```
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
```

In [64]:

```
arr_2d[1][1]      #finding the particular element
```

Out[64]:

```
25
```

In [66]:

```
arr_2d[1,1]
```

Out[66]:

```
25
```

In [77]:

```
arr_2d=arr_2d[:2, 1:]                # for creating submatrix, use slicing (explain one b
y one)
```

In [78]:

```
arr_2d
```

Out[78]:

```
array([[10, 15],
       [25, 30]])
```

In [79]:

```
arr_2d[1:,1:]
```

Out[79]:

```
array([[30]])
```

In [80]:

```
arr1=np.arange(1,11)
arr1
```

Out[80]:

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

In [81]:

```
bool_arr=arr1>5             # boolean array
bool_arr
```

Out[81]:

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

In [82]:

```
arr1[bool_arr]       # conditionally selected array depending on bool_arr
```

Out[82]:

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

In [83]:

```
arr1[arr1>5]       #doing the condistional selection in one line
```

Out[83]:

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

In [84]:

```
arr1[arr1<6]
```

Out[84]:

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

In [86]:

```
arr_2d_2=np.arange(50).reshape(10,5)    #using two method in one line (try to find any c
hunck of sub-matrix for exercise)
arr_2d_2
```

Out[86]:

```
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],
       [30, 31, 32, 33, 34],
       [35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44],
       [45, 46, 47, 48, 49]])
```

NumPy Operations

In [87]:

```
arr=np.arange(11)
arr
```

Out[87]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [88]:

```
arr+arr
```

Out[88]:

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [89]:

```
arr-arr
```

Out[89]:

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

In [90]:

```
arr*arr
```

Out[90]:

```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100])
```

In [91]:

```
1/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-91-9e1622b385b6> in <module>
----> 1 1/0

ZeroDivisionError: division by zero
```

In [92]:

```
arr/arr
```

```
<ipython-input-92-50b4ced5627e>:1: RuntimeWarning: invalid value encountered in true_divi
de
  arr/arr
```

Out[92]:

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

In [93]:

```
arr*3
```

Out[93]:

```
array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27, 30])
```

In [94]:

```
arr+6
```

Out[94]:

```
array([ 6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16])
```

In [95]:

```
arr-30
```

Out[95]:

```
array([-30, -29, -28, -27, -26, -25, -24, -23, -22, -21, -20])
```

In [96]:

```
arr**2
```

Out[96]:

```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100], dtype=int32)
```

In [97]:

```
1/arr
```

```
<ipython-input-97-016353831300>:1: RuntimeWarning: divide by zero encountered in true_div
ide
  1/arr
```

Out[97]:

```
array([       inf, 1.        , 0.5       , 0.33333333, 0.25      ,
       0.2       , 0.16666667, 0.14285714, 0.125     , 0.11111111,
       0.1       ])
```

In [98]:

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

Out[98]:

```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ,
       3.16227766])
```

In [99]:

```
np.exp(arr)
```

Out[99]:

```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03, 2.20264658e+04])
```

In [100]:

```
np.log(arr)
```

```
<ipython-input-100-a67b4ae04e95>:1: RuntimeWarning: divide by zero encountered in log
  np.log(arr)
```

Out[100]:

```
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436,
       1.60943791, 1.79175947, 1.94591015, 2.07944154, 2.19722458,
       2.30258509])
```

In [101]:

```
np.max(arr)
```

Out[101]:

```
10
```

In [102]:

```
arr.max()
```

Out[102]:

```
10
```

In [103]:

```
np.min(arr)
```

Out[103]:

```
0
```

In [104]:

```
arr.min()
```

Out[104]:

```
0
```

In [ ]: