Numpy Practice Session

In [1]:
```
import numpy as np
```

In [2]:
```
#1-D Array
a = np.array([1,2,3,4])
a
```

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

In [9]:
```
food = np.array(["samosa", "pakora", "raita"])
food
```

Out[9]:
```
array(['samosa', 'pakora', 'raita'], dtype='<U6')
```

In [4]:
```
price = np.array([5,5,5])
price
```

Out[4]:
```
array([5, 5, 5])
```

In [5]:
```
type(price)
```

Out[5]:
```
numpy.ndarray
```

In [6]:
```
len(price)
```

Out[6]:
```
3
```

In [7]:
```
price[0:]
```

Out[7]:
```
array([5, 5, 5])
```

In [10]:
```
food[1]
```

Out[10]:
```
'pakora'
```

In [11]:
```
price.mean()
```

Out[11]:
```
5.0
```

In [12]:
```
#Zeros
np.zeros(6)
```

Out[12]:
```
array([0., 0., 0., 0., 0., 0.])
```

In [13]:
```
#Ones
np.ones(5)
```

Out[13]:
```
array([1., 1., 1., 1., 1.])
```

In [22]:
```
#empty
np.empty(5)
```

Out[22]:
```
array([1., 1., 1., 1., 1.])
```

In [18]:
```
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-1df33ba9512e> in <module>
----> 1 np.empty(shape, dtype=float, order='C', like=None)

NameError: name 'shape' is not defined
```

In [25]:
```
#range
np.arange(10)
```

Out[25]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [26]:
```
#specify range
np.arange(2, 20)
```

Out[26]:
```
array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19])
```

In [27]:
```
#specific interval
np.arange(2,20,5)
```

Out[27]:
```
array([ 2,  7, 12, 17])
```

In [30]:
```
#table of 5
np.arange(0,55,5)
```

Out[30]:
```
array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45, 50])
```

In [33]:
```
#linepace
np.linspace(0, 10, num=5)
```

Out[33]:
```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

In [36]:
```
#specify your data type
np.ones(50, dtype=np.int64)
```

Out[36]:
```
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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1], dtype=int64)
```

In [37]:
```
#specify your data type
np.ones(50, dtype=np.float64)
```

Out[37]:
```
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., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

Array Function

In [39]:
```
a = np.array([1,23,45,67,8.0])
a
```

Out[39]:
```
array([ 1., 23., 45., 67.,  8.])
```

In [41]:
```
a.sort()
a
```

Out[41]:
```
array([ 1.,  8., 23., 45., 67.])
```

In [43]:
```
b = np.array([3.4,76.5,432.34,43.2])
b
```

Out[43]:
```
array([  3.4 ,  76.5 , 432.34,  43.2 ])
```

In [46]:
```
c = np.concatenate((a,b))
c
```

Out[46]:
```
array([  1.  ,   8.  ,  23.  ,  45.  ,  67.  ,   3.4 ,  76.5 , 432.34,
        43.2 ])
```

In [48]:
```
c.sort()
c
```

Out[48]:
```
array([  1.  ,   3.4 ,   8.  ,  23.  ,  43.2 ,  45.  ,  67.  ,  76.5 ,
       432.34])
```

2-D Array

In [55]:
```
a = np.array([[1,2],[5,4]])
a
```

Out[55]:
```
array([[1, 2],
       [5, 4]])
```

In [56]:
```
b = np.array([[4,5],[5,6]])
b
```

Out[56]:
```
array([[4, 5],
       [5, 6]])
```

In [58]:
```
c = np.concatenate((a,b), axis=0)
c
#We can concatenate 2-D array, if both arrays have same dimensions
```

Out[58]:
```
array([[1, 2],
       [5, 4],
       [4, 5],
       [5, 6]])
```

In [59]:
```
c = np.concatenate((a,b), axis=1)
c
#when we chane the axis value it changes the stack positioning
```

Out[59]:
```
array([[1, 2, 4, 5],
       [5, 4, 5, 6]])
```

In [62]:
```
a = np.array([[[0,1,2,3], [4,5,6,7]],

              [[0,1,2,3], [4,5,6,7]],
```

```
            [[0,1,2,3,], [4,5,6,7]]])
a
```

Out[62]:
```
array([[[0, 1, 2, 3],
        [4, 5, 6, 7]],

       [[0, 1, 2, 3],
        [4, 5, 6, 7]],

       [[0, 1, 2, 3],
        [4, 5, 6, 7]]])
```

In [64]:
```
#to find the number of dimensions
a.ndim
```

Out[64]: 3

In [66]:
```
b = np.array([[1,2,3],
              [4,5,6],
               [7,8,9]])
b
#It is a 2-D array of order 3 x 3
```

Out[66]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [67]:
```
b.ndim
```

Out[67]: 2

In [69]:
```
#size is equal to number of elements
b.size
```

Out[69]: 9

In [71]:
```
a = np.array([[[0,1,2,3], [4,5,6,7]],

              [[0,1,2,3], [4,5,6,7]],

              [[0,1,2,3,], [4,5,6,7]]])
a
```

Out[71]:
```
array([[[0, 1, 2, 3],
        [4, 5, 6, 7]],

       [[0, 1, 2, 3],
        [4, 5, 6, 7]],

       [[0, 1, 2, 3],
        [4, 5, 6, 7]]])
```

In [72]:
```
#shape
a.shape
```

Out[72]: (3, 2, 4)

In [80]:
```
a = np.arange(9) # 3 x 3
a
```

Out[80]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [79]:
```
#to reshape
b = a.reshape(3,3)
b
```

Out[79]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [82]:
```
# reshape
np.reshape(a, newshape=(1,9), order='C')
```

Out[82]:
```
array([[0, 1, 2, 3, 4, 5, 6, 7, 8]])
```

In [86]:
```
#Convert 1-D into 2-D
a = np.array([1,2,3,4,5,6,7,8,9])
a
```

Out[86]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [87]:
```
a.shape
```

Out[87]:
```
(9,)
```

In [89]:
```
#row wise convertion
b = a[np.newaxis, :]
b
```

Out[89]:
```
array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

In [90]:
```
b.shape
```

Out[90]:
```
(1, 9)
```

In [92]:
```
#column wise convertion
b = a[:, np.newaxis]
b
```

Out[92]:
```
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
       [8],
       [9]])
```

In [93]:
```
a
```

Out[93]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [95]:
```
a[2:9]
# here we are printing array a from index 2 to index 6
```

Out[95]:
```
array([3, 4, 5, 6, 7, 8, 9])
```

In [97]:
```
a
```

Out[97]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [98]:
```
a*6
```

Out[98]:
```
array([ 6, 12, 18, 24, 30, 36, 42, 48, 54])
```

In [99]:
```
a+6
```

Out[99]:
```
array([ 7,  8,  9, 10, 11, 12, 13, 14, 15])
```

In [100…
```
a.sum()
```

Out[100…
```
45
```

In [101…
```
a.mean()
```

Out[101…
```
5.0
```

In [ ]: