In [7]:

```
import numpy as np
```

DataTypes and Attributes

In [8]:

```
#numpy main datatype is ndarray
```

In [13]:

```
a1=np.array([1,2,3])
a1
```

Out[13]:

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

In [15]:

```
type(a1)
```

Out[15]:

```
numpy.ndarray
```

In [16]:

```
a2=np.array([[1, 2.0, 3.3],
             [4,  5,   6.5]])
a2
```

Out[16]:

```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [19]:

```
a3=np.array([[[1,2,3],
              [4,5,6],
              [7,8,9]],
             [[10,11,12],
             [13,14,15],
             [16,17,18]]])
```

In [20]:

```
a3
```

Out[20]:

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

       [[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]]])
```

In [21]:

```
a1.shape
```

Out[21]:

```
(3,)
```

In [22]:

```
a2.shape
```

Out[22]:

```
(2, 3)
```

In [23]:

```
a3.shape
```

Out[23]:

```
(2, 3, 3)
```

In [24]:

```
#ndim tell dimension
a1.ndim, a2.ndim, a3.ndim
```

Out[24]:

```
(1, 2, 3)
```

In [25]:

```
a1.dtype, a2.dtype, a3.dtype
```

Out[25]:

```
(dtype('int32'), dtype('float64'), dtype('int32'))
```

In [26]:

```
a1.size, a2.size, a3.size
```

Out[26]:

```
(3, 6, 18)
```

In [27]:

```
#create a dataframe from a numpy array
import pandas as pd
df=pd.DataFrame(a2)
df
```

Out[27]:

	0	1	2
0	1.0	2.0	3.3
1	4.0	5.0	6.5

creating array

In [28]:

```
 sample_array=np.array([1, 2, 3])
```

In [29]:

```
sample_array
```

Out[29]:

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

In [30]:

```
sample_array.dtype
```

Out[30]:

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

In [31]:

```
#ones genrate atomatic array of 1zero
ones=np.ones((2, 3))
```

In [32]:

```
ones
```

Out[32]:

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

In [33]:

```
ones.dtype
```

Out[33]:

```
dtype('float64')
```

In [34]:

```
zeros=np.zeros((2,3))
```

In [35]:

```
zeros
```

Out[35]:

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

In [36]:

```
#arange
#give start value then give max value and then give diffrence
range_array=np.arange(0, 10, 2)
```

In [37]:

```
range_array
```

Out[37]:

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

In [38]:

```
#random
#it giive randam array of from given size and in given interval
randam_array=np.random.randint(0, 10, size=(3,5))
randam_array
```

Out[38]:

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

In [39]:

```
randam_array.shape
```

Out[39]:

```
(3, 5)
```

In [40]:

```
randam_array.size
```

Out[40]:

```
15
```

In [41]:

```
#another way to creating random array returnn floot in interval 0 to1
np.random.random((3,5))
```

Out[41]:

```
array([[8.89698695e-02, 8.69144285e-01, 4.42073479e-01, 1.28924972e-01,
        3.41609738e-01],
       [4.06793503e-01, 2.65779328e-04, 8.80804768e-01, 9.45769703e-01,
        9.35052517e-01],
       [5.58733911e-02, 9.63554042e-02, 8.86402477e-01, 3.98449195e-01,
        9.54222106e-01]])
```

In [42]:

```
#psudo randam number using seed random not change now when we use seed 2 in another is a
np.random.seed(seed=2)
random_array_4=np.random.randint(10, size=(5, 3))
random_array_4
```

Out[42]:

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

viewing array and matrices

In [43]:

```
#geting unique array from another array
np.unique(random_array_4)
```

Out[43]:

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

In [44]:

```
a1
```

Out[44]:

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

In [45]:

```
#viewing through index
a1[0]
```

Out[45]:

```
1
```

In [46]:

```
a2
```

Out[46]:

```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [47]:

```
a2[0]
```

Out[47]:

```
array([1. , 2. , 3.3])
```

In [48]:

```
a3
```

Out[48]:

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

       [[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]]])
```

In [49]:

```
a3[0]
```

Out[49]:

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

In [50]:

```
#geting first 2 element of every index
a3
```

Out[50]:

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

       [[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]]])
```

In [51]:

```
a3[:2, :2, :2]
```

Out[51]:

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

       [[10, 11],
        [13, 14]]])
```

In [52]:

```
a4=np.random.randint(10, size=(2, 3, 4, 5))
a4
```

Out[52]:

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

        [[2, 9, 8, 7, 1],
         [6, 8, 5, 9, 9],
         [9, 3, 0, 0, 2],
         [8, 8, 2, 9, 6]],

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


       [[[5, 8, 3, 0, 0],
         [5, 7, 5, 0, 8],
         [6, 5, 1, 7, 4],
         [3, 6, 1, 4, 0]],

        [[8, 5, 4, 2, 9],
         [7, 1, 9, 2, 1],
         [0, 7, 1, 8, 9],
         [0, 7, 0, 5, 2]],

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

In [53]:

```
#get the first 4 numbers of the inner most array
a4[:, :, :, :4]
```

Out[53]:

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

        [[2, 9, 8, 7],
         [6, 8, 5, 9],
         [9, 3, 0, 0],
         [8, 8, 2, 9]],

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


       [[[5, 8, 3, 0],
         [5, 7, 5, 0],
         [6, 5, 1, 7],
         [3, 6, 1, 4]],

        [[8, 5, 4, 2],
         [7, 1, 9, 2],
         [0, 7, 1, 8],
         [0, 7, 0, 5]],

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

Mainuplating array

Arithmetic

In [54]:

```
a1+1
```

Out[54]:

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

In [55]:

```
a1-1
```

Out[55]:

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

In [56]:

```
a1*1
```

Out[56]:

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

In [57]:

```
a1/1
```

Out[57]:

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

In [58]:

```
a1*a2
```

Out[58]:

```
array([[ 1. ,  4. ,  9.9],
       [ 4. , 10. , 19.5]])
```

In [59]:

```
a2*a3
#error due to diffrent shape
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
Cell In[59], line 1
----> 1 a2*a3

ValueError: operands could not be broadcast together with shapes (2,3) (2,
3,3)
```

In [60]:

```
a2/a1
```

Out[60]:

```
array([[1.        , 1.        , 1.1       ],
       [4.        , 2.5       , 2.16666667]])
```

In [61]:

```
a2//a1
#floor division removev the decimal
```

Out[61]:

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

In [62]:

```
a2**2
#power 2
```

Out[62]:

```
array([[ 1.  ,  4.  , 10.89],
       [16.  , 25.  , 42.25]])
```

In [63]:

```
np.square(a2)
```

Out[63]:

```
array([[ 1.  ,  4.  , 10.89],
       [16.  , 25.  , 42.25]])
```

In [64]:

```
np.exp(a1)
```

Out[64]:

```
array([ 2.71828183,  7.3890561 , 20.08553692])
```

In [65]:

```
np.log(a1)
```

Out[65]:

```
array([0.        , 0.69314718, 1.09861229])
```

Aggregation

it perform same operation on a nunmber of things

In [66]:

```
sum(a1)
#this is python sum
```

Out[66]:

```
6
```

In [67]:

```
np.sum(a1)
```

Out[67]:

```
6
```

In [68]:

```
#creating massive array
massive_array =np.random.random(1000)
```

In [69]:

```
massive_array[:10]
```

Out[69]:

```
array([0.17671216, 0.59125735, 0.48926617, 0.54790778, 0.69952062,
       0.24581116, 0.18662715, 0.11058315, 0.27405925, 0.01025004])
```

In [70]:

```
%timeit sum(massive_array) #python sum
%timeit np.sum(massive_array) #numpysum
```

```
103 µs ± 3.54 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
6.36 µs ± 84.5 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops eac
h)
```

In [71]:

```
a2
```

Out[71]:

```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [72]:

```
# np.mean(a2)
```

In [73]:

```
np.max(a2)
```

Out[73]:

```
6.5
```

In [74]:

```
np.std(a2)
#standerdeviation
```

Out[74]:

```
1.8226964152656422
```

In [75]:

```
np.var(a2)
#variance
```

Out[75]:

```
3.3222222222222224
```

In [76]:

```
np.sqrt(np.var(a2))
```

Out[76]:

```
1.8226964152656422
```

Reshaping and transposing

In [77]:

```
a2
```

Out[77]:

```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [78]:

```
a2.shape
```

Out[78]:

```
(2, 3)
```

In [79]:

```
a3
```

Out[79]:

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

       [[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]]])
```

In [80]:

```
a3.shape
```

Out[80]:

```
(2, 3, 3)
```

In [81]:

```
#reshape it reshape dimension bithout changing value
a2_reshape=a2.reshape(2,3,1)
a2_reshape
```

Out[81]:

```
array([[[1. ],
        [2. ],
        [3.3]],

       [[4. ],
        [5. ],
        [6.5]]])
```

In [82]:

```
a2_reshape*a3
```

Out[82]:

```
array([[[  1. ,   2. ,   3. ],
        [  8. ,  10. ,  12. ],
        [ 23.1,  26.4,  29.7]],

       [[ 40. ,  44. ,  48. ],
        [ 65. ,  70. ,  75. ],
        [104. , 110.5, 117. ]]])
```

In [83]:

```
#transpose
a2.T
```

Out[83]:

```
array([[1. , 4. ],
       [2. , 5. ],
       [3.3, 6.5]])
```

In [84]:

```
a2.T.shape
```

Out[84]:

```
(3, 2)
```

Dot product

In [87]:

```
np.random.seed(22)
mat1=np.random.randint(10, size=(5,3))
mat2=np.random.randint(10, size=(5,3))
mat1
```

Out[87]:

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

In [88]:

```
mat2
```

Out[88]:

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

In [89]:

```
# Eelment wise multiplication(Hardmard)
mat1*mat2
```

Out[89]:

```
array([[40, 20,  0],
       [ 8, 12,  6],
       [24, 72, 12],
       [ 6, 16, 49],
       [14, 63, 56]])
```

In [90]:

```
#DOT PRODUCT
np.dot(mat1, mat2)
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
Cell In[90], line 2
      1 #DOT PRODUCT
----> 2 np.dot(mat1, mat2)

File <__array_function__ internals>:180, in dot(*args, **kwargs)

ValueError: shapes (5,3) and (5,3) not aligned: 3 (dim 1) != 5 (dim 0)
```

In [91]:

```
#transpose
mat1.T
```

Out[91]:

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

In [92]:

```
mat2.shape ,mat1.shape
```

Out[92]:

```
((5, 3), (5, 3))
```

In [94]:

```
mat3 =np.dot(mat2, mat1.T)
mat3
```

Out[94]:

```
array([[ 60,  86,  88,  84,  93],
       [ 18,  26,  28,  27,  30],
       [ 66,  96, 108, 105, 117],
       [ 23,  66,  56,  71,  80],
       [ 63, 112, 112, 119, 133]])
```

In [95]:

```
mat2.shape
```

Out[95]:

```
(5, 3)
```

Dot product (Nut butter Sales)

In [96]:

```
np.random.seed(12)
#numbers of jars sold
sales_amount =np.random.randint(20, size=(5,3))
sales_amount
```

Out[96]:

```
array([[11,  6, 17],
       [ 2,  3,  3],
       [12, 16, 17],
       [ 5, 13,  2],
       [11, 10,  0]])
```

In [98]:

```
import pandas as pd
```

In [102]:

```
#creatively weekely_sales dataframe
weekly_sales = pd.DataFrame(sales_amount,
                            index=["Mon", "Tues", "Wed", "Thrus", "fri" ],
                            columns=["Almand butter", "Pensut butter", "Cashew butterr"])
```

In [103]:

```
weekly_sales
```

Out[103]:

	Almand butter	Pensut butter	Cashew butterr
Mon	11	6	17
Tues	2	3	3
Wed	12	16	17
Thrus	5	13	2
fri	11	10	0

In [104]:

```
#create price array
prices =np.array([10, 8, 12])
prices
```

Out[104]:

```
array([10,  8, 12])
```

In [106]:

```
#create butter prices dataframe
butter_prices =pd.DataFrame(prices.reshape(1,3),
                            index=["price"],
                            columns=["Almand butter", "Pensut butter", "Cashew butterr"])
butter_prices
```

Out[106]:

	Almand butter	Pensut butter	Cashew butterr
price	10	8	12

In [107]:

```
total_sales=prices.dot(sales_amount)
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
Cell In[107], line 1
----> 1 total_sales=prices.dot(sales_amount)

ValueError: shapes (3,) and (5,3) not aligned: 3 (dim 0) != 5 (dim 0)
```

In [108]:

```
prices.shape
```

Out[108]:

```
(3,)
```

In [109]:

```
sales_amount.shape
```

Out[109]:

```
(5, 3)
```

In [111]:

```
#shape are not aligned let's transpose
total_sales =prices.dot(sales_amount.T)
total_sales
```

Out[111]:

```
array([362,  80, 452, 178, 190])
```

In [112]:

```
#create daily sales
butter_prices.shape, weekly_sales.shape
```

Out[112]:

```
((1, 3), (5, 3))
```

In [115]:

```
weekly_sales.T.shape
```

Out[115]:

```
(3, 5)
```

In [116]:

```
daily_sales=butter_prices.dot(weekly_sales.T)
daily_sales
```

Out[116]:

	Mon	Tues	Wed	Thrus	fri
price	362	80	452	178	190

In [113]:

```
weekly_sales
```

Out[113]:

	Almand butter	Pensut butter	Cashew butterr
Mon	11	6	17
Tues	2	3	3
Wed	12	16	17
Thrus	5	13	2
fri	11	10	0

In [118]:

```
#Daoesn't work not the right shape
weekly_sales["Total ($)"] = daily_sales
weekly_sales
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
Cell In[118], line 2
      1 #Daoesn't work not the right shape
----> 2 weekly_sales["Total ($)"] = daily_sales
      3 weekly_sales

File ~\Machine_learning\project1\env\lib\site-packages\pandas\core\frame.p
y:3968, in DataFrame.__setitem__(self, key, value)
   3966     self._setitem_array(key, value)
   3967 elif isinstance(value, DataFrame):
-> 3968     self._set_item_frame_value(key, value)
   3969 elif (
   3970     is_list_like(value)
   3971     and not self.columns.is_unique
   3972     and 1 < len(self.columns.get_indexer_for([key])) == len(value)
   3973 ):
   3974     # Column to set is duplicated
   3975     self._setitem_array([key], value)

File ~\Machine_learning\project1\env\lib\site-packages\pandas\core\frame.p
y:4123, in DataFrame._set_item_frame_value(self, key, value)
   4120     return
   4122 if len(value.columns) != 1:
-> 4123     raise ValueError(
   4124         "Cannot set a DataFrame with multiple columns to the singl
e "
   4125         f"column {key}"
   4126     )
   4128 self[key] = value[value.columns[0]]

ValueError: Cannot set a DataFrame with multiple columns to the single col
umn Total ($)
```

In [122]:

```
weekly_sales["Total ($)"] =daily_sales.T
weekly_sales
```

Out[122]:

	Almand butter	Pensut butter	Cashew butterr	Total ($)
Mon	11	6	17	362
Tues	2	3	3	80
Wed	12	16	17	452
Thrus	5	13	2	178
fri	11	10	0	190

comparison operator

In [123]:

```
a1
```

Out[123]:

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

In [124]:

```
a2
```

Out[124]:

```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [125]:

```
a1>a2
```

Out[125]:

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

In [128]:

```
bool_array =a1>=a2
bool_array
```

Out[128]:

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

In [129]:

```
a1>5
```

Out[129]:

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

In [130]:

```
a1<5
```

Out[130]:

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

In [132]:

```
a1==a1
```

Out[132]:

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

sorting array

In [136]:

```
random_array = np.random.randint(10, size=(3,5))
random_array
```

Out[136]:

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

In [138]:

```
random_array.shape
```

Out[138]:

```
(3, 5)
```

In [139]:

```
np.sort(random_array)
```

Out[139]:

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

In [140]:

```
#sort accoding to index
np.argsort(random_array)
```

Out[140]:

```
array([[0, 4, 1, 3, 2],
       [3, 4, 0, 1, 2],
       [4, 3, 1, 0, 2]], dtype=int64)
```

In [141]:

```
a1
```

Out[141]:

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

In [143]:

```
np.argsort(a1)
```

Out[143]:

```
array([0, 1, 2], dtype=int64)
```

In [144]:

```
#min value at index
np.argmin(a1)
```

Out[144]:

```
0
```

In [145]:

```
#for max
np.argmax(a1)
```

Out[145]:

```
2
```

In [149]:

```
random_array
```

Out[149]:

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

In [147]:

```
np.argmax(random_array, axis=0)
```

Out[147]:

```
array([2, 2, 1, 2, 2], dtype=int64)
```

In [148]:

```
np.argmax(random_array, axis=1)
```

Out[148]:

```
array([2, 2, 0], dtype=int64)
```

Pratical example numpy in action

In [160]:

```
#turn an image into numpy array
from matplotlib.image import imread
panda = imread("panda.png")
print(type(panda))
```

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

In [163]:

```
panda.size, panda.shape, panda.ndim
```

Out[163]:

```
(30108672, (3872, 2592, 3), 3)
```

In [ ]:

```

```