In [1]:
```
import numpy as np
np.eye(5)
```

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

In [9]:
```
ar2=np.arange(25).reshape(5,5)
ar2
```

Out[9]:
```
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 [10]:
```
ar2.shape
```

Out[10]:
```
(5, 5)
```

In [6]:
```
import numpy as np
ar1 = np.arange(13)
ar1
reshaped_array=ar1.reshape(13,1)
ar1
```

Out[6]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [12]:
```
ar2[0:3,1:4]
```

Out[12]:
```
array([[ 1,  2,  3],
       [ 6,  7,  8],
       [11, 12, 13]])
```

In [14]:
```
ar2[1:2,1:3]
```

Out[14]:
```
array([[6, 7]])
```

In [22]:
```
ar2[-4:-2,-4:-2]
```

Out[22]:
```
array([[ 6,  7],
       [11, 12]])
```

In [102]:
```
np.sum(ar2)
```

Out[102]:
```
300
```

In [26]:
```
np.sum(ar2,axis=1)
```

Out[26]:
```
array([ 10,  35,  60,  85, 110])
```

In [ ]:
```
#mean --> average
#median--> center value
#mode--> max num of repeated element
#variance--> median / total num of elements
#std --> square root of variance
```

In [93]:
```
np.random.randint(60,65,2)-->#np.random.randint(start,end,no.of value)
```

Out[93]:
```
array([64, 63])
```

In [95]:
```
np.linspace(10,20,3,dtype=int)
```

Out[95]:
```
array([10, 15, 20])
```

In [101]:
```
np.random.rand(3)
```

Out[101]:
```
array([0.78527841, 0.33064382, 0.29074487])
```

In [103]:
```
ar2.min()
```

Out[103]:
```
0
```

In [104]:
```
ar2.max()
```

Out[104]:
```
24
```

In [107]:
```
ar2.min()
```

Out[107]:
```
0
```

In [109]:
```
a1=np.arange(24)
a1
```

Out[109]:
```
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])
```

In [110]:
```
min(a1)
```

Out[110]:
```
0
```

In [111]:
```
max(a1)
```

Out[111]:
```
23
```

In [112]:
```
a1.max()
```

Out[112]:
```
23
```

In [113]:
```
print(np.argmax(a1))
```

```
23
```

In [116]:
```
a2=np.arange(1,25)
a2
print(np.argmax(a2))
```

```
23
```

In [120]:
```
np.sin(a2)/np.cos(a1)
```

Out[120]:
```
array([ 8.41470985e-01,  1.68294197e+00, -3.39111092e-01,  7.64452759e-01,
        1.46704449e+00, -9.85029068e-01,  6.84239669e-01,  1.31231634e+00,
       -2.83242879e+00,  5.97083791e-01,  1.19178183e+00, -1.21240292e+02,
        4.97914399e-01,  1.09164237e+00,  4.75574864e+00,  3.78975776e-01,
        1.00390328e+00,  2.72924166e+00,  2.26977764e-01,  9.23375125e-01,
        2.05021420e+00,  1.61600081e-02,  8.46253555e-01,  1.69955376e+00])
```

In [129]:
```
import numpy as np
a2=np.array([1,2,3])
a4=np.array([4,5,6])
(a2[0]*a4[0]+a2[1]*a4[1]+a2[2]*a4[2])

fname=np.array(["sid","fn1","fn2"])
gname=np.array(["hi",])
```

Out[129]:
```
32
```

In [146]:
```
arr11=np.array([1,2,3,4])
arr22=np.array([1,2,3])

a=len(arr11)
b=len(arr22)
if(a!=b):
    c=np.abs(a-b)
    if(a>b):
        np.append(arr22,1)
    else:
        np.append(arr11,1)
    ar3=np.sum(np.sum(arr11) + np.sum(arr22))
ar3
```

Out[146]:
```
16
```

In [ ]:
```
#pandas
import pandas as pd
#Pandas is defines as an open source library that provides high performance da
#data analysis reqires lot of processing such as restructuring,
#cleaning ,mergin,manipulating..ect.
#above functionalies coz it is fast ,simple than other tools,
#pandas is built on numpy ,numpy is required for operating pandas
```

In [ ]:
```
#pandas series
#pandas series is a data structure with one
#dimensional labelled array
#it is a primary building block of data frame ,marking its
#rows and columns
```

In [149]:
```
import numpy as np
labels = ["a","b","c"]
my_data = [10,20,30]
arr = np.array(my_data)
d = {"a":100,"b":200,"c":300}
```

In [150]:
```
#syntax -->pandas.series(data=none,index=none,dtype=none,name=none,copy=true o
#eg :1
import pandas as pd
pd.Series(my_data)#Series -->S= capital
```

Out[150]:
```
0    10
1    20
2    30
dtype: int64
```

In [152]:
```
type(pd.Series(my_data))
```

Out[152]:
```
pandas.core.series.Series
```

In [153]:
```
#series with labels
pd.Series(data=my_data,index=labels) #type 1
```

Out[153]:
```
a    10
b    20
c    30
dtype: int64
```

In [154]:
```
pd.Series(my_data,labels)#type  2 ..output is same
```

Out[154]:
```
a    10
b    20
c    30
dtype: int64
```

In [157]:
```
pd.Series(data=[print(),len,sum])
```

Out[157]:
```
0                       None
1    <built-in function len>
2    <built-in function sum>
dtype: object
```

In [165]:
```
ser1 = pd.Series([1,2,3,4,5],["USA","INDIA","CANADA","UK","EGYPT"])
ser1
```

Out[165]:
```
USA       1
INDIA     2
CANADA    3
UK        4
EGYPT     5
dtype: int64
```

In [166]:
```
#access value using index
ser1[0:3]
```

Out[166]:
```
USA       1
INDIA     2
CANADA    3
dtype: int64
```

In [170]:
```
ser2=pd.Series([5,6,7,8],['USA','BRAzIL','CANADA','UK'])
```

In [171]:
```
ser1+ser2
```

Out[171]:
```
BRAzIL     NaN
CANADA    10.0
EGYPT      NaN
INDIA      NaN
UK        12.0
USA        6.0
dtype: float64
```

In [172]:
```
ser2
```

Out[172]:
```
USA       5
BRAzIL    6
CANADA    7
UK        8
dtype: int64
```

In [173]:
```
ser2['china'] = 'duplicate'
```

In [174]:
```
ser2
```

Out[174]:
```
USA               5
BRAzIL            6
CANADA            7
UK                8
china     duplicate
dtype: object
```

In [176]:
```
ser2.drop("USA")
```

Out[176]:
```
BRAzIL            6
CANADA            7
UK                8
china     duplicate
dtype: object
```

In [ ]:
```
#dataframe
#pd.DataFrame(datas,row_label,col_label)
```

In [181]:
```
import numpy as np
df = pd.DataFrame(np.random.randn(5,4))
```

In [182]:
```
df
```

Out[182]:

	0	1	2	3
0	-0.668311	2.727698	-0.680305	-0.791817
1	-0.989895	-0.776215	1.243206	-0.574766
2	-3.550476	-0.172434	0.327725	0.726852
3	1.459425	2.205909	-0.091625	1.260669
4	0.707667	-0.633950	1.528386	-0.283621

In [200]:
```
import numpy as np
df=pd.DataFrame(np.random.randn(5,4),['A','B','C','D','E'],['W','X','Y','Z'])
df
```

Out[200]:

	W	X	Y	Z
A	-0.444279	1.450715	-0.772272	-1.501108
B	0.974060	-0.411133	1.239587	-1.143623
C	-0.469667	0.132065	0.832259	-0.450190
D	-0.914487	0.148341	-0.220224	0.596668
E	0.192435	0.130477	0.327733	0.290064

In [193]:
```
#to convert dictionary to dataframe
d = {"col1":[1,2],"col2":[3,4],"col3":[5,6]}
```

In [194]:
```
d
```

Out[194]:
```
{'col1': [1, 2], 'col2': [3, 4], 'col3': [5, 6]}
```

In [195]:
```
df = pd.DataFrame(data=d)
```

In [196]:
```
df
```

Out[196]:

	col1	col2	col3
0	1	3	5
1	2	4	6

In [203]:
```
df.index
```

Out[203]:
```
Index(['A', 'B', 'C', 'D', 'E'], dtype='object')
```

In [204]:
```
df
```

Out[204]:

	W	X	Y	Z
A	-0.444279	1.450715	-0.772272	-1.501108
B	0.974060	-0.411133	1.239587	-1.143623
C	-0.469667	0.132065	0.832259	-0.450190
D	-0.914487	0.148341	-0.220224	0.596668
E	0.192435	0.130477	0.327733	0.290064

In [205]:
```
df.T
```

Out[205]:

	A	B	C	D	E
W	-0.444279	0.974060	-0.469667	-0.914487	0.192435
X	1.450715	-0.411133	0.132065	0.148341	0.130477
Y	-0.772272	1.239587	0.832259	-0.220224	0.327733
Z	-1.501108	-1.143623	-0.450190	0.596668	0.290064

In [206]:
```
df.columns
```

Out[206]:
```
Index(['W', 'X', 'Y', 'Z'], dtype='object')
```

In [207]:
```
type(df)
```

Out[207]:
```
pandas.core.frame.DataFrame
```

In [208]:
```
df.dtypes
```

Out[208]:
```
W    float64
X    float64
Y    float64
Z    float64
dtype: object
```

In [209]:
```
df.info()#--> get information about
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5 entries, A to E
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   W       5 non-null      float64
 1   X       5 non-null      float64
 2   Y       5 non-null      float64
 3   Z       5 non-null      float64
dtypes: float64(4)
memory usage: 372.0+ bytes
```

In [210]:
```
df.values
```

Out[210]:
```
array([[-0.44427928,  1.45071493, -0.77227204, -1.50110752],
       [ 0.97406041, -0.41113301,  1.23958694, -1.14362309],
       [-0.4696667 ,  0.1320653 ,  0.83225864, -0.45019007],
       [-0.91448713,  0.14834076, -0.22022425,  0.596668  ],
       [ 0.1924348 ,  0.13047694,  0.32773282,  0.29006398]])
```

In [211]:
```
df.axes
```

Out[211]:
```
[Index(['A', 'B', 'C', 'D', 'E'], dtype='object'),
 Index(['W', 'X', 'Y', 'Z'], dtype='object')]
```

In [212]:
```
df.ndim
```

Out[212]:
```
2
```

In [213]:
```
df.size
```

Out[213]:
```
20
```

In [217]:
```
df
```

Out[217]:

	W	X	Y	Z
A	-0.444279	1.450715	-0.772272	-1.501108
B	0.974060	-0.411133	1.239587	-1.143623
C	-0.469667	0.132065	0.832259	-0.450190
D	-0.914487	0.148341	-0.220224	0.596668
E	0.192435	0.130477	0.327733	0.290064

In [215]:
```
#To access specific column in df
df['W']
```

Out[215]:
```
A   -0.444279
B    0.974060
C   -0.469667
D   -0.914487
E    0.192435
Name: W, dtype: float64
```

In [216]:
```
type(df['W'])
```

Out[216]:
```
pandas.core.series.Series
```

In [218]:
```
#To access multiple columns
df[['W','X','Y']]
```

Out[218]:

	W	X	Y
A	-0.444279	1.450715	-0.772272
B	0.974060	-0.411133	1.239587
C	-0.469667	0.132065	0.832259
D	-0.914487	0.148341	-0.220224
E	0.192435	0.130477	0.327733

In [219]:
```
#To create new column
df['NEW']=df['W']+df['Y']
```

In [220]:
```
df
```

Out[220]:

	W	X	Y	Z	NEW
A	-0.444279	1.450715	-0.772272	-1.501108	-1.216551
B	0.974060	-0.411133	1.239587	-1.143623	2.213647
C	-0.469667	0.132065	0.832259	-0.450190	0.362592
D	-0.914487	0.148341	-0.220224	0.596668	-1.134711
E	0.192435	0.130477	0.327733	0.290064	0.520168

In [224]:
```
#To delete a column (it's show delete but it is not delete)
df.drop('NEW',axis=1)
```

Out[224]:

	W	X	Y	Z
A	-0.444279	1.450715	-0.772272	-1.501108
B	0.974060	-0.411133	1.239587	-1.143623
C	-0.469667	0.132065	0.832259	-0.450190
D	-0.914487	0.148341	-0.220224	0.596668
E	0.192435	0.130477	0.327733	0.290064

In [225]:
```
df
```

Out[225]:

	W	X	Y	Z	NEW
A	-0.444279	1.450715	-0.772272	-1.501108	-1.216551
B	0.974060	-0.411133	1.239587	-1.143623	2.213647
C	-0.469667	0.132065	0.832259	-0.450190	0.362592
D	-0.914487	0.148341	-0.220224	0.596668	-1.134711
E	0.192435	0.130477	0.327733	0.290064	0.520168

In [226]:
```
#delete the column
df.drop('NEW',axis=1,inplace=True)
```

In [227]:
```
df
```

Out[227]:

	W	X	Y	Z
A	-0.444279	1.450715	-0.772272	-1.501108
B	0.974060	-0.411133	1.239587	-1.143623
C	-0.469667	0.132065	0.832259	-0.450190
D	-0.914487	0.148341	-0.220224	0.596668
E	0.192435	0.130477	0.327733	0.290064

In [ ]:
```
```