In [2]:

```
import numpy as np
import pandas as pd
```

In [14]:

```
x = np.arange(1,12,2)
```

In [16]:

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

Out[16]:

```
array([[ 1,  3,  5],
       [ 7,  9, 11]])
```

In [8]:

```
np.linspace(0,1,20).reshape(5,4)
```

Out[8]:

```
array([[ 0.        ,  0.05263158,  0.10526316,  0.15789474],
       [ 0.21052632,  0.26315789,  0.31578947,  0.36842105],
       [ 0.42105263,  0.47368421,  0.52631579,  0.57894737],
       [ 0.63157895,  0.68421053,  0.73684211,  0.78947368],
       [ 0.84210526,  0.89473684,  0.94736842,  1.        ]])
```

In [19]:

```
# Generates random numbers from o to 1 only. The argument specifies the number of rando
m numbers to desire.
np.random.rand(15)
```

Out[19]:

```
array([ 0.10828964,  0.51800358,  0.8797633 ,  0.36228769,  0.37555033,
        0.42044744,  0.54184559,  0.388982  ,  0.79652623,  0.95335399,
        0.29402575,  0.66385841,  0.30773754,  0.41846903,  0.11845899])
```

In [24]:

```
# Generates random integers. The first two defines the range and the third decides the
 number of randoms you wish to generate.
z = np.random.randint(0,10,6).reshape (3,2)
```

In [25]:

```
z
```

Out[25]:

```
array([[7, 7],
       [9, 5],
       [0, 1]])
```

In [30]:

```
z.argmin()
```

Out[30]:

```
4
```

In [31]:

```
z.dtype
```

Out[31]:

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

In [2]:

```
from numpy.random import rand
from numpy.random import randint
```

In [33]:

```
rand(5)
```

Out[33]:

```
array([ 0.57066638,  0.77376005,  0.19420016,  0.60717534,  0.95329291])
```

In [40]:

```
randint(0,10,4).reshape(2,2)
```

Out[40]:

```
array([[7, 9],
       [1, 8]])
```

In [42]:

```
x = np.arange(0,11)
```

In [43]:

```
x
```

Out[43]:

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

In [46]:

```
# THis will always generate a boolean value. Do not use in this way directly.
x>5
```

Out[46]:

```
array([False, False, False, False, False, False,  True,  True,  True,
        True,  True], dtype=bool)
```

In [47]:

```
# Always use this way to get the data and not the boolean value.
x[x>5]
```

Out[47]:

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

In [50]:

```
r = np.random.randint(0,50,50)
```

In [53]:

```
len(r)
```

Out[53]:

```
50
```

In [55]:

```
r = r.reshape(5,10)
```

In [65]:

```
r
```

Out[65]:

```
array([[17, 41, 42, 13, 15, 28, 39, 41, 10,  6],
       [28, 43, 16, 42,  6, 29, 33, 17, 47, 22],
       [30,  2,  7, 14,  7, 23, 19, 46, 16, 22],
       [11, 20, 16, 23, 45, 30, 36, 38, 48, 49],
       [34,  6, 27, 34, 38,  7, 20, 19, 31, 25]])
```

In [59]:

```
len(r[r>25])
```

Out[59]:

```
24
```

In [60]:

```
len(r[r<25])
```

Out[60]:

```
25
```

In [69]:

```
# Slicing and dicing
r[1:3,1:3]
```

Out[69]:

```
array([[43, 16],
       [ 2,  7]])
```

In [70]:

```
# Operations
np.sum(r)
```

Out[70]:

```
1278
```

In [3]:

```
import pandas as pd
```

In [76]:

```
data = np.random.randint(0,10,10)
```

In [77]:

```
data
```

Out[77]:

```
array([0, 5, 0, 7, 5, 7, 4, 7, 3, 7])
```

In [81]:

```
index = np.array(['a', 'b', 'c','d','e','f','g','h','i','t'])
```

In [82]:

```
index
```

Out[82]:

```
array(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 't'],
      dtype='<U1')
```

In [85]:

```
# The difference between Series and array is that in Series you specifically
# assign or have your own index value used to denote the specific data
pd.Series(data,index)
```

Out[85]:

```
a    0
b    5
c    0
d    7
e    5
f    7
g    4
h    7
i    3
t    7
dtype: int32
```

In [4]:

```
df = pd.DataFrame([randint(0,20,5)],['A','B','C','D','E'],['C1','C2','C3','C4','C5'])
```

In [89]:

```
df
```

Out[89]:

	C1	C2	C3	C4	C5
A	19	13	18	1	11
B	19	13	18	1	11
C	19	13	18	1	11
D	19	13	18	1	11
E	19	13	18	1	11

In [94]:

```
# When you provide two or more columns make sure to have two square brackets since the
 result is a dataframe.
df[['C1','C2']]
```

Out[94]:

	C1	C2
A	19	13
B	19	13
C	19	13
D	19	13
E	19	13

In [104]:

```
df['newCol'] = [1,2,3,4,5]
```

In [111]:

```
# While dropping a column, always remember to mention
# 1) Axis
# 2) Inplace = True

df.drop('newCol',1, inplace = True)
```

In [112]:

```
df
```

Out[112]:

	C1	C2	C3	C4	C5
A	19	13	18	1	11
B	19	13	18	1	11
C	19	13	18	1	11
D	19	13	18	1	11
E	19	13	18	1	11

In [117]:

```
# To extract a row loc method has to be used against the name of the dataframe.
df.loc['B']
```

Out[117]:

```
C1    19
C2    13
C3    18
C4     1
C5    11
Name: B, dtype: int64
```

In [119]:

```
df.iloc[2]
```

Out[119]:

```
C1    19
C2    13
C3    18
C4     1
C5    11
Name: C, dtype: int64
```

In [120]:

```
df.loc['B','C2']
```

Out[120]:

```
13
```

In [124]:

```
# To extract a subset of a DataFrame
df.loc[['B','C','D'],['C2','C1','C3']]
```

Out[124]:

	C2	C1	C3
B	13	19	18
C	13	19	18
D	13	19	18

In [132]:

```
# Conditional Selection in a DataFrame.
df[df>10]
```

Out[132]:

	C1	C2	C3	C4	C5
A	19	13	18	NaN	11
B	19	13	18	NaN	11
C	19	13	18	NaN	11
D	19	13	18	NaN	11
E	19	13	18	NaN	11

In [149]:

```
# Conditional Selection for one Column
df[df['C1']>15]['C1']
```

Out[149]:

```
A    19
B    19
C    19
D    19
E    19
Name: C1, dtype: int64
```

In [150]:

```
# Conditional Selection for multi Column
df[df['C1']>15][['C1','C2']]
```

Out[150]:

	C1	C2
A	19	13
B	19	13
C	19	13
D	19	13
E	19	13

In [153]:

```
# Conditional Selection for specific col and rows to return
df[df['C1']>15].loc[['A','B'],['C1','C2']]
```

Out[153]:

	C1	C2
A	19	13
B	19	13

In [160]:

```
df[(df['C1'] >10) & (df['C2'] >12)]['C2']
```

Out[160]:

```
A    13
B    13
C    13
D    13
E    13
Name: C2, dtype: int64
```

In [173]:

```
df
```

Out[173]:

	C1	C2	C3	C4	C5
A	19	13	18	1	11
B	19	13	18	1	11
C	19	13	18	1	11
D	19	13	18	1	11
E	19	13	18	1	11

In [177]:

```
df.reset_index(inplace = True)
```

In [183]:

```
df.drop('index',axis = 1)
```

Out[183]:

	C1	C2	C3	C4	C5
0	19	13	18	1	11
1	19	13	18	1	11
2	19	13	18	1	11
3	19	13	18	1	11
4	19	13	18	1	11

In [188]:

```
df.set_index('index',inplace = True)
```

In [189]:

```
df
```

Out[189]:

	C1	C2	C3	C4	C5
index					
A	19	13	18	1	11
B	19	13	18	1	11
C	19	13	18	1	11
D	19	13	18	1	11
E	19	13	18	1	11

In [192]:

```
df.loc[['A','E'],['C1','C3']]
```

Out[192]:

	C1	C3
index		
A	19	18
E	19	18

In [195]:

```
# Using the split method
newind = 'a b c d e'.split()
```

In [196]:

```
newind
```

Out[196]:

```
['a', 'b', 'c', 'd', 'e']
```

In [198]:

```
df.fillna(12)
```

Out[198]:

	C1	C2	C3	C4	C5
index					
A	19	13	18	1	11
B	19	13	18	1	11
C	19	13	18	1	11
D	19	13	18	1	11
E	19	13	18	1	11

In [5]:

```
df
```

Out[5]:

	C1	C2	C3	C4	C5
A	7	1	13	19	15
B	7	1	13	19	15
C	7	1	13	19	15
D	7	1	13	19	15
E	7	1	13	19	15

In [16]:

```
df.loc['A','C1']=  np.nan
```

In [17]:

```
df
```

Out[17]:

	C1	C2	C3	C4	C5
A	NaN	1	13	19	15
B	7	1	13	19	15
C	7	1	13	19	15
D	7	1	13	19	15
E	7	1	13	19	15

In [22]:

```
df.fillna(10, inplace =True)
```

In [23]:

```
df
```

Out[23]:

	C1	C2	C3	C4	C5
A	10	1	13	19	15
B	7	1	13	19	15
C	7	1	13	19	15
D	7	1	13	19	15
E	7	1	13	19	15

In [3]:

```
dic = {'A' : [1,2,3,np.nan], 'B': [2,np.nan,4,np.nan]}
```

See the usage of dictionary in Series and DataFrame.

In [9]:

```
data=pd.DataFrame(dic, index=['First','Sec','Third','Four'])
```

In [7]:

```
pd.Series(dic)
```

Out[7]:

```
A      [1, 2, 3, nan]
B    [2, nan, 4, nan]
dtype: object
```

In [18]:

```
data['NewCol2']=[10,5,3,5]
```

In [20]:

```
data.set_index('NewCol2') # No inplace True used. So does not affect the original dataf
rame.
```

Out[20]:

	A	B	NewCol
NewCol2			
10	1.0	2.0	1
5	2.0	NaN	2
3	3.0	4.0	3
5	NaN	NaN	5

In [55]:

```
data
```

Out[55]:

	A	B	NewCol	NewCol2
First	1.0	2.0	1	10
Sec	2.0	3.0	2	5
Third	3.0	4.0	3	3
Fouth	1.0	1.0	2	2

In [54]:

```
# Adding a row never taught in the class. Remember this !!
data.loc['Fouth']=[1,1,2,2]
```

In [59]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, First to Fouth
Data columns (total 4 columns):
A          4 non-null float64
B          4 non-null float64
NewCol     4 non-null int64
NewCol2    4 non-null int64
dtypes: float64(2), int64(2)
memory usage: 160.0+ bytes
```

In [48]:

```
data.dropna(inplace=True)
```