In [ ]:
```
import pandas as pd
data = {'color' : ['blue','green','yellow','red','white'],
 'object' : ['ball','pen','pencil','paper','mug'],
 'price' : [1.2,1.0,0.6,0.9,1.7]}
f = pd.DataFrame(data)
print(f)
```

In [ ]:
```
#If the dict object from which you want to create a dataframe contains more data
#than you are interested in, you can make a selection.
f = pd.DataFrame(data, columns=['object','price'])
print(f)
```

In [ ]:
```
#if you want to assign labels to the indexes of a dataframe, you have to use the index
#an array containing the labels
import numpy as np
f = pd.DataFrame(data, index=['one','two','three','four','five'])
print(f)
```

In [ ]:
```
#define three arguments in the constructor, in the following order—
#a data matrix, #an array containing the labels assigned to the index option,
#and an array containing the names of the columns assigned to the columns option
import numpy as np
f = pd.DataFrame(np.arange(16).reshape((4,4)),
                     index=['red','blue','yellow','white'], columns=['ball','pen','pen
print(f)
```

In [ ]:
```
print(f)
print(f.index,'\n')
print(f.columns,'\n',)
print(f.values,'\n',)
#retrieve column as a series  using dict like notation
print(f['pencil'])
#retrieve column as a series  by attribute
print(f.pencil)
```

In [ ]:
```
#For rows within a dataframe, it is possible to use the loc attribute with the index
#value of the row that you want to extract.
data = {'color' : ['blue','green','yellow','red','white'],
 'object' : ['ball','pen','pencil','paper','mug'],
 'price' : [1.2,1.0,0.6,0.9,1.7]}
f = pd.DataFrame(data)
print(f)
print(f.loc[2],'\n')
print(f.loc[[2,4]],'\n')
print(f[1:3],'\n')
```

In [ ]:
```
#you can also assign a label, using the name attribute, to these two substructures to i
f.index.name = 'id'
f.columns.name = 'item'
print(f)
```

In [ ]:
```
import pandas as pd
data = {'color' : ['blue','green','yellow','red','white','black','pink'],
 'object' : ['ball','pen','pencil','paper','mug','car','flower'],
 'price' : [1.2,1.0,0.6,0.9,1.7,5.5,2.2]}
f = pd.DataFrame(data)
print(f)
print(f.head())

#Change the order of columns
f1=pd.DataFrame(f,columns=['price','color','object'])
print(f1)
```

In [ ]:
```
f1=pd.DataFrame(f,columns=['price','color','object','size'])
print(f1)
print(f1.columns)
```

In [ ]:
```
#To add a new column,assign a value to the instance of the dataframe and
#specifying a new column name
f['new'] = 12
print(f)
```

In [ ]:
```
f['new1'] = [3.0,1.3,2.2,0.8,1.1]
print(f)
f['new1']=np.arange(5,)
print(f)

del f['new']
print(f)
```

In [ ]:
```
#Assgning values to some objects
import pandas as pd
s=pd.Series([10,20,30],index=[3,1,5])
print(f)
f['size']=s
print(f)
```

In [ ]:
```
f['select']=f.object=='pen'
print(f)
del f['select']
print(f.columns)
```

In [ ]:
```
#Membership of a Value
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
#DataFrame from Nested dictIf the nested dict is passed to the data frame,
#pandas will interpret the outer dict keys as the column indices and
# the inner keys as the row indices
nestdict = { 'red': { 2012: 22, 2013: 33 },
 'white': { 2011: 13, 2012: 22, 2013: 16},
 'blue': {2011: 17, 2012: 27, 2013: 18}}
f=pd.DataFrame(nestdict)
print(f)
```

In [ ]:
```
print(f.T)
```

In [ ]:
```
#Reindexing
#to create a new object with the data conformed to a new index
s=pd.Series([10,20,30],index=['a','b','c'])
print(s)
s1=s.reindex(['x','y','a'])
print(s1)
```

In [ ]:
```
#interpolation or filling of values
a=pd.Series(['blue','purple','yellow'],index=[0,2,4])
print(a)
b=a.reindex(range(8),method='ffill')
print(b)
```

In [ ]:
```
#with Data Frame, reindex can alter either the (row)index, columns, or both.
#when a sequence is passed, it reindexes the rows in the result
f=pd.DataFrame(np.arange(9).reshape((3,3)), index=['a','c','d'],
               columns=['Ohio','Texas','California'])
print(f)
f1=f.reindex(['a','b','c','d'])
print(f1)
states=['California','Texas','Ohio']
f2=f.reindex(columns=states)
print(f2)
```

In [ ]:
```
s=pd.Series([10,20,30,40])
print(s)
print(s.idxmin())
print(s.idxmax())
print(s.index.is_unique)
```

In [ ]:
```
#Dropping
s = pd.Series(np.arange(4.), index=['red','blue','yellow','white'])
print(s)
s1=s.drop('yellow')
print(s1)
# ser.drop(['blue','white'])
```

In [ ]:
```
f=pd.DataFrame(np.arange(16).reshape((4,4)), index=['red','blue','yellow','white'],
   columns=['ball','pen','pencil','paper'])
print(f)

#To delete rows, you just pass the indexes of the rows.
#f.drop(['blue','yellow'])
f1=f.drop(['blue'])
print(f1)

#To delete columns, you always need to specify the indexes of the columns, but you
#must specify the axis from which to delete the elements,
f2=f.drop(['pen','pencil'],axis=1)
print(f2)
```

In [ ]:
```
#Arithmetic and Data Alignment
#pandas can align indexes coming from two different data structures.
s1 = pd.Series([3,2,5,1],['white','yellow','green','blue'])
s2 = pd.Series([1,4,7,2,1],['white','yellow','black','blue','brown'])
print(s1+s2)
```

In [ ]:
```
frame1 = pd.DataFrame(np.arange(16).reshape((4,4)),
 index=['red','blue','yellow','white'],
 columns=['ball','pen','pencil','paper'])
frame2 = pd.DataFrame(np.arange(12).reshape((4,3)),
 index=['blue','green','white','yellow'],
 columns=['mug','pen','ball'])
print(f1)
print(f2)
print(f1+f2)

# frame1.add(frame2)
#sub mul div
```

In [ ]:
```
#Operation Between DataFrame and Series
frame = pd.DataFrame(np.arange(16).reshape((4,4)),
 index=['red','blue','yellow','white'],
  columns=['ball','pen','pencil','paper'])
s = pd.Series(np.arange(4), index=['ball','pen','pencil','paper'])
print(frame - s)
```

In [ ]:
```
#universal functions
frame = pd.DataFrame(np.arange(16).reshape((4,4)),
 index=['red','blue','yellow','white'],
 columns=['ball','pen','pencil','paper'])
#calculate the square root of each value in the dataframe
print( np.sqrt(frame))
```

In [ ]:
```
frame = pd.DataFrame(np.arange(16).reshape((4,4)),
 index=['red','blue','yellow','white'],
 columns=['ball','pen','pencil','paper'])

print(frame.sum())
```

```
print(frame.mean())
print(frame.describe())
```

In [ ]: