1.

```
# masking  on the values to extract subsets of data
s1= pd.Series([10,20,30,40,50], index=['A','B','C','D','E'])
print(s1)
print('Masking')
print("s1[(s1>10) & (s1<40)] \n", s1[(s1>10) & (s1<40)])
#print('Fancy indexing')
#print("s1[['A', 'C']] \n" , s1[['A', 'C']])
```

2.Series Operations

```
s1=pd.Series([6,7,8,9,5])
s3 = pd.Series([1,2,3,4], index = ['a','b','c','d'])
print('s1: \n',s1,'s3: \n',s3)
s4 = s1.append(s3)    #Observe no copy is created
print('Appended  series: \n',s4 )
# Delete a row with a particular element
s4.drop(['c'])
print("Series s4  after dropping 'c':\n",  s4)
```

3.Aritmetic Functions

```
import pandas as pd
#Create two series
s1=pd.Series([6,7,8,9,5])
s2=pd.Series([0,1,2,3,4,5,7])
print('Series are : \n',s1, '\n', s2)
```

4.

```
print("\nMedian of series s2 is", s2.median())
print("\n Mean of series s2 is " , s2.mean())
print("\n Maximum of series s2 is", s2.max())
print("\n Minimum of series s2 is", s2.min())
```

5.

```
#Series with char/ string elements
string=pd.Series(['a','b','c','S','e','J','g','B','P','o'])
print('A Series wih String values: \n ', string)
print('string.str.upper(): \n',string.str.upper())
print('string.str.lower(): \n',string.str.lower())
```

6.Pandas DataFrames

Creation of DataFrames

```
#Dataframe as a stack of Series.  we create two columns using series and then make
a DataFrame
population_d= {'California': 3833, 'Texas': 8193,
                'New York': 6511, 'Florida': 5560, 'Ohio': 1135}    #Statewise
population
print(population_d, type(population_d))
population = pd.Series(population_d)
print(population)
```

7.

```
area_d = {'California': 423967, 'Texas': 695662, 'New York': 141297,
             'Florida': 170312, 'Ohio': 149995}
area = pd.Series(area_d)
print(area)
states = pd.DataFrame({'Population': population,  'Area': area})  #two series with
same index
print("Data Frame of States: \n", states)
```

8.DataFrame Attributes

```
print('\n', states.index)     #row indices
print('\n', states.columns)    #column names
print('\n', states.values)
print('\n', states['Area'])     #access a column on a DataFrame like a key value
pair
```

```
print('\n',states.Area)        #Columns can also be accessed as an Attribute
#print('\n',states.Area is states['Area'])
print('\n',states.loc['California'])   #accessing row of a dataframe with explicit
index
print('\n', states.iloc[3])
print('\n', states.loc['California','Area'])
print('\n', states.iloc[3,1])
```

9.

```
import numpy as np
num_arr=np.random.randn(6,4) #random delection of numbers following a standard
normal distribution
print("Array is : \n", num_arr)
cols=['A','B','C','D']    #arrays will not have index and columns
df1=pd.DataFrame(num_arr, columns=cols, index = ['i', 'ii', 'iii', 'iv', 'v',
'vi'])
#array of values, index, column
print('\n Data Frame from numpy array is : \n')
print(df1)
```

10.

```
# create a dataframe using a dictionary of Lists, values are lists and column
names are keys
data= {'city' : ['Bombay', 'Chennai', 'Chennai', 'Delhi', 'Mysore' ], 'year' :
[2001, 2005, 2003, 2001, 2000],
        'pop' : [25, 35, 20, 40, 15]}
df2= pd.DataFrame(data)
print(df2)
#observe index is assigned automatically
```

11.

```
# create a dataframe using a dictionary of Lists, values are lists and column
names are keys
data= {'city' : ['Bombay', 'Chennai', 'Chennai', 'Delhi', 'Mysore' ], 'year' :
[2001, 2005, 2003, 2001, 2000],
        'pop' : [25, 35, 20, 40, 15]}   #this will have only columns no index
labels=['a', 'b', 'c', 'd', 'e']
df2= pd.DataFrame(data, index=labels)
print(df2)
#observe index is assigned automatically
```

12. Exercise

```
#create a dataframe from a list of dictionaries
df3=pd.DataFrame([{'a': 1, 'b': 2, 'c':3, 'd':4}, {'a': 10, 'b': 20, 'c': 30},
{'a': 11, 'b': 21, 'c': 41, 'd': 51}])
print(df3)  # creating a dataframe from a list of dictionaries
```

13.Visualizing DataFrames

```
#First Create a DataFrame
data={'Animals':
['cat','cat','turtle','dog','dog','cat','turtle','cat','dog','dog'],
          'Age': [2.5,3,0.5,np.nan,5,2,4.5,np.nan,7,3],
       'Visits' : [1,3,2,3,2,3,1,1,2,1],
    'Priority' : ['y','y','n','y','n','n','n','y','n','n']}
labels=['a','b','c','d','e','f','g','h','i','j']
animals_data=pd.DataFrame(data,index=labels)
print(animals_data)
print(type(animals_data))
```

14. DataFrame Attributes - index, cols, values, datatype of values

```
print("\n animals_data.index:\n ", animals_data.index)
print("\n animals_data.columns:\n", animals_data.columns)
print("\n animals_data.values:\n", animals_data.values)      #will show only
values without index and column names
```

```
print("\n animals_data.dtypes:\n", animals_data.dtypes)    #will show the datatype
of each column

print(animals_data.head())   # will display top 5 lines of the dataFrame
print(animals_data.tail())     # will display bottom 5 lines of the dataframe
```

15.

```
# Information about the whole dataframe
print( animals_data.info())
#nrows, ncols, index, datatype of each column, number of nonnull values

#statistical data of dataframe
print('\n Statistical Description : \n',animals_data.describe())
#mean std max min quartiles for columns with numeric type

print('\n Description for object values: \n',animals_data.describe(include =
['object']))
#count, unique values, mode , freq
```

16. Exercise
Accessing/Slcing Data in a DataFrame

```
print("\n animals_data.index:\n ", animals_data.index)    #accessing index
print("\n animals_data.columns:\n", animals_data.columns)    #accessing column
names

#Accessing columns of a DataFrame  2 ways
print("\n animals_data['Animals']:\n",animals_data['Animals'] )
print("\n animals_data['Age'] :\n", animals_data['Age'])

print("\n animals_data.Animal:\n",animals_data.Animals)
animals_data[['Age','Visits']]   #Displaying particular Columns

print("\n animals_data.loc['b', 'Age']:\n",animals_data.loc['b', 'Age'])
#accessing by row and column
animals_data.loc['b', 'Age'] =50

print("\n animals_data.loc['b', 'Age']:\n",animals_data.loc['b', 'Age'])
#updatinng a value in a Dataframe
```

17.

```
#Accessing rows of a DataFrame by implict and explicit index
print("\n animals_data.loc['a', :] :\n", animals_data.loc['a', :])   #values of a
row are given as columns
print("\ Rows 1 to 3 using slicing:\n",animals_data.iloc[1:3, 2:3] )      #iloc
for implicit indexing
```

18.

```
# Exercise
#Acessing individual elements in the table by row and column
print(animals_data)
print("\n animals_data.loc['b', 'Age']:\n",animals_data.loc['b', 'Age'])
#accessing by row and column
animals_data.loc['b', 'Age'] =50
print("\n animals_data.loc['b', 'Age']:\n",animals_data.loc['b', 'Age'])
#updatinng a value in a Dataframe
print("\n animals_data.loc['b', 'Age']:\n",animals_data.iloc[2,2])   #accessing by
row and column

print(animals_data.iloc[:5, 2:4] )
print(animals_data.loc['b':'e', 'Animals':'Visits'] )
```

19.

```
print("Transpose of the Data Frame :")
animals_data.T
```

```
20.
Sorting DataFrames
#Methods in DataFrame object Sort By Values
print(animals_data)
print("\n Sorting the Data Agewise:\n", animals_data.sort_values(by = 'Age',
ascending = False))   #sort by which column
#Any missing value is sorted at end by default
animals_data.sort_values(by='Age')[1:4]

21.
#Sort by index
print("\n Sorting the Data by Index:\n", animals_data.sort_index(axis=1))
#Since it is already sorted you dont see the change


22.
#. Create a new index  Reindexing
print(animals_data)
animals_data_reindex = animals_data.reindex(['d', 'e', 'g', 'f', 'a', 'b', 'c',
'i', 'j'])
print("\n ReIndexed Data: \n",animals_data.reindex)
print("\n Sorted by row index:\n", animals_data_reindex.sort_index(axis=0))
print("\n Sorted by Column Index:\n", animals_data_reindex.sort_index(axis=1))

23. Creating a copy
animals_data_c=animals_data.copy()
print("\n Copy of animals_data:\n", animals_data_c)

24. Deleting a row or Column of a DataFrame
print(animals_data)
print("Drop rows with names 'a; and 'b':\n", animals_data.drop(['a', 'b']))
#dropping rows

#to drop the columns permanently use inplace - True
print("Drop rows with names 'a; and 'b':\n", animals_data_c.drop(['a', 'b'],
inplace=True))

print(animals_data.drop('Visits', axis=1))
#dropping column  columns are axis=1 for drop() default is row
#So if we dont mention axis = 1 it will search for a row with name 'Visits'
print(animals_data.drop('Visits', axis='columns'))


25.Aggregate Functions

#Why doing an Aggregation on a Row doesnt make sense
print("Mean of the Dataframe is: \n",animals_data.mean())    #mean of values in
columns containing numeric data
print("\nMean of 'Age' is: ",animals_data[['Age']].mean())
print("\nTotal visits :",animals_data[['Visits']].sum())
print("\nMax visits: ",animals_data[['Visits']].max())
print("\nMin visits: ",animals_data[['Visits']].min())
print("\n Index of Max visits: ",animals_data[['Visits']].idxmax())
print("\n Index of Min visits: ",animals_data[['Visits']].idxmin())
print("\nSum: \n",animals_data.sum())     #for strings sum is string concatenation

26. Missing Values
#Trouble with missing data
#Why we need to drop missing values
import numpy as np
arr1 = np.array([1, None, 3, 4])    #observe None is a NoneType
print(arr1,  arr1.dtype)
print(arr1.sum())    #unsupported operand type(s) for +: 'int' and 'NoneType
print(arr1.mean())
```

```
arr2 = np.array([1, np.nan, 3,4])   #np.nan is a float type
print(arr2,  arr2.dtype)
print(arr2.sum())                  #so np.nan is handled by  numpy  but not None
print(arr2.mean())


27.
print(pd.Series([1, np.nan, 2, None]))
ser_null = pd.Series(range(5,8), dtype=int)
print('\n',ser_null)
ser_null[0] = None
print('\n',ser_null)
print('\n',ser_null.sum())
#Series datatype converts a None also to a nan and it can do the aggregation even
with the nan values .
#it ignores the nan values

28.
#Dataframe aggregation methods ignore nan values and find the sum
data = pd.DataFrame([[1,      np.nan, 2],
                     [2,      3,      5],
                     [np.nan, 4,      6]])
print(data)
print(data.sum())    #sum by default is column sum axis =0
print(data.sum(axis=1))   #sum across columns

29.Detecting and dropping Null Values
print(pd.isnull(animals_data))    #isnull() function in pandas library
#print(animals_data.isnull())      #isnull() in DataFrame object
#Observe that Age has two missing values
print(pd.notnull(animals_data))

data = pd.DataFrame([[1,      np.nan, 2],
                     [2,      3,      5],
                     [np.nan, 4,      6]])
print('\n data.isnull(): \n',data.isnull())
print('\n data.notnull(): \n',data.notnull())
data[data.notnull()]

30.
data = pd.DataFrame([[1,      np.nan, 2],
                     [2,      3,      5],
                     [np.nan, 4,      6]])
print(data)
#print(data.fillna(0))     we can fill with column mean or mode for categorical
data
#print(data.fillna(method='ffill'))
print(data.fillna(method='bfill'))
print(data)
# original data will not change, to change we need to set inplace = True
#find mean of each column and fill each individually

31.  Series Concatenation
ser1 = pd.Series(['A', 'B', 'C'], index=[1, 2, 3])
ser2 = pd.Series( ['D', 'E', 'F'], index=[4, 5, 6] )     #test with the same index
print("Series 1 : \n",ser1, "\nSeries 2 : \n",ser2)
print("Concatenating series: \n", pd.concat([ser1, ser2]))

32. DataFrame Concatenation

df1 = pd.DataFrame({'A' : ['axe', 'art', 'ant'], 'B' : ['bat', 'bar', 'bin'], 'C'
: ['cap', 'cat', 'car']},
                    index = [1,2,3])
df2 = pd.DataFrame({'D' : ['dam', 'den', 'dot'], 'E': [ 'ear', 'eat', 'egg'], 'F':
['fan', 'fog', 'fat']},
                   index =[ 2, 3, 6])
print("Data frame 1 : \n", df1,'\n Data Frame 2: \n', df2)
print("Concatenating Data Frames: \n",pd.concat([df1,df2], axis=0))
 # axis =0 is stacking one below the other
print("Concatenating Data Frames along axis 1: \n",pd.concat([df1,df2], axis = 1))
#will consider common indices
```

33.

```
df_concat = pd.concat([df1, df2], ignore_index = True)
print("Concatenation of dataframes while ignoring the index: \n", df_concat)
```

34.

```
print(" Inner Join on dataframes : \n", pd.concat([df1, df2], join = 'inner')) #no
overlapping columns
```

35. Append (similar to concat but a dataframe method)

```
print(df1)
print(df2)
print(df1.append(df2))    # append is same as concat stocks dataframes one below
another
```