In [11]:
```
#missing data in python is represented by 2 types NaN and None
#the none object type
import numpy as np
import pandas as pd
a=np.array([1,2,None,4])
print("Array with missing data=")
print(a)
#since none is an object all elements are treated as objects and computation of oeprations are much slower than primitive array
#aggregates wont work with None object type
```

```
Array with missing data=
[1 2 None 4]
```

In [12]:
```
#the NaN data representation
a=np.array([1,2,np.nan,4])
print("Array with missing data=")
print(a)
#computation with nan data type will result in nan data type and array with nan is considered as float array
print("Adding integer and nan data type=")
print(1+np.nan)
#aggregatrs with nan data types will result in nan
print("Sum aggregate of array with nan=")
print(a.sum())
#nan ufuncs will ignore nan
print("Sum aggregate of array with nan=")
print(np.nansum(a))
#nan is only for floating point and not for other data types
```

```
Array with missing data=
[ 1.  2. nan  4.]
Adding integer and nan data type=
nan
Sum aggregate of array with nan=
nan
Sum aggregate of array with nan=
7.0
```

In [16]:
```
#type casting of none to nan
#when a value in ax = pd.Series(range(2), dtype=int)integer is replaced by none it is type casted to nan
#to accomodate nan the array is made floating type
a=pd.Series([1,2,3,4])
print("Original array=")
print(a)#printing original array
a[1]=None
print("Array after replacing value with none=")
print(a)#printing array after replacing value with none
print("Boolean array=")
b=pd.Series([True,False,True])
print(b)#printing boolean array
b[1]=np.nan
print(b)#printing array after replacing a value in boolean array with nan
#this time array is type casted to object
```

```
Original array=
0    1
1    2
2    3
3    4
dtype: int64
Array after replacing value with none=
0    1.0
1    NaN
2    3.0
3    4.0
dtype: float64
Boolean array=
0     True
1    False
2     True
dtype: bool
0    True
1     NaN
2    True
dtype: object
```

In [22]:
```
#operating on null values
#using isnull to create boolean mask
x=pd.Series([0,1,2,None,3,4,np.nan])
print("Boolean mask to find if null values are there are not=")
print(x.isnull())
#notnull will return values that are not missing
print("Printing values that are not null=")
print(x[x.notnull()])
```

```
Boolean mask to find if null values are there are not=
0    False
1    False
2    False
3     True
4    False
5    False
6     True
dtype: bool
Printing values that are not null=
0    0.0
1    1.0
2    2.0
4    3.0
5    4.0
dtype: float64
```

In [31]:
```
#dropna and fillna in Series
x=pd.Series([0,1,2,None,3,4,np.nan])
print("Replacing nullvalues by '$'=")
print(x.fillna("$"))
print("Removing null values form series object=")
print(x.dropna())
#in dataframe dropna will drop all rows in which null values is present
print("Dataframe with missing values=")
y=pd.DataFrame([[1,np.nan,3],[np.nan,5,6],[7,8,9]])#creating dataframe from 2 dimensional array
print(y)
print("dataframe after using dropna()=")
print(y.dropna())#using dropna on dataframe
print("dataframe after uswing dropna() with column")
print(y.dropna(axis=1))
```

```
Replacing nullvalues by '$'=
0    0.0
1    1.0
2    2.0
3      $
4    3.0
5    4.0
6      $
dtype: object
Removing null values form series object=
0    0.0
1    1.0
2    2.0
4    3.0
5    4.0
dtype: float64
Dataframe with missing values=
     0    1  2
0  1.0  NaN  3
1  NaN  5.0  6
2  7.0  8.0  9
dataframe after using dropna()=
     0    1  2
2  7.0  8.0  9
dataframe after uswing dropna() with column
   2
0  3
1  6
2  9
```

In [38]:
```
#how in pandas
y=pd.DataFrame([[1,np.nan,3],[np.nan,5,6],[7,8,9]])#creating dataframe from 2 dimensional array
y[3]=np.nan
print(y)
print("dataframe after dropping columnbs which only have nan=")
print(y.dropna(axis=1,how="all"))#dropping columns which have all data as nan
print("dataframe after dropping rows with more than 2 missing data=")
print(y.dropna(thresh=3))#thresh will specify minimumn nan values to drop row
```

```
     0    1  2   3
0  1.0  NaN  3 NaN
1  NaN  5.0  6 NaN
2  7.0  8.0  9 NaN
dataframe after dropping columnbs which only have nan=
     0    1  2
0  1.0  NaN  3
1  NaN  5.0  6
2  7.0  8.0  9
dataframe after dropping rows with more than 2 missing data=
     0    1  2   3
2  7.0  8.0  9 NaN
```

In [50]:
```
#forward fill and backward fill
print(y)
print("Null values repalcing along row by forward propogation=")
print(y.fillna(method="ffill",axis=1))#forward propogating along row
print("Null values repalcing along row by backward propogation=")
print(y.fillna(method="bfill",axis=1))#backward propogating along row
print("Null values repalcing along column by forward propogation=")
print(y.fillna(method="ffill",axis=0))#forward propogating along column
```

```
     0    1  2   3
0  1.0  NaN  3 NaN
1  NaN  5.0  6 NaN
2  7.0  8.0  9 NaN
Null values repalcing along row by forward propogation=
     0    1    2    3
0  1.0  1.0  3.0  3.0
1  NaN  5.0  6.0  6.0
2  7.0  8.0  9.0  9.0
Null values repalcing along row by backward propogation=
     0    1    2   3
0  1.0  3.0  3.0 NaN
1  5.0  5.0  6.0 NaN
2  7.0  8.0  9.0 NaN
Null values repalcing along column by forward propogation=
     0    1  2   3
0  1.0  NaN  3 NaN
1  1.0  5.0  6 NaN
2  7.0  8.0  9 NaN
```