In [3]:
```
import pandas as pd
```

In [4]:
```
#series object in pandas
p=pd.Series(["A","B","C","D"])
print(p)
print("Values list of series=")
print(p.values)
print("Index array=")
print(p.index)
#accessing elements is similar to numpy
print("First element=",p[0])
#slicing pandas object array
print(p[1:3])
```

```
0    A
1    B
2    C
3    D
dtype: object
Values list of series=
['A' 'B' 'C' 'D']
Index array=
RangeIndex(start=0, stop=4, step=1)
First element= A
1    B
2    C
dtype: object
```

In [5]:
```
#customising indices in pandas series
a=pd.Series([1,2,3,4],index=["a","b","c","d"])#instead of integers we are using string as index
print(a)
print("element associated with index 'b'=",a["b"])#accessing element using custom index
```

```
a    1
b    2
c    3
d    4
dtype: int64
element associated with index 'b'= 2
```

In [6]:
```
#padas series as dictioaries in python
d={"Dairy milk":10,"5star_fuse":15,"Snickers":20,"Mars":50}#dictionary of chocolate names with price
p=np.Series(d)#passing in dictionary for series pandas
print(p)
print("Slicing in pandas=")
print(p["Dairy milk":"Snickers"])#Sinckers(max range) is also included in the slice unlike array
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5000\2030312467.py in <module>
      1 #padas series as dictioaries in python
      2 d={"Dairy milk":10,"5star_fuse":15,"Snickers":20,"Mars":50}#dictionary of chocolate names with price
----> 3 p=np.Series(d)#passing in dictionary for series pandas
      4 print(p)
      5 print("Slicing in pandas=")

NameError: name 'np' is not defined
```

In [8]:
```
#dataframe in pandas
age=pd.Series({"a":15,"b":19,"c":35,"d":70})#seperate series for age
height=pd.Series({"a":6,"b":5.2,"c":5.6,"d":6.1})#seperate series for height
bio=pd.DataFrame({"age":age,"height":height})#dataframe bio combining age and height series
print(bio)
print(bio.index)#returns all keys
print(bio.values)#returns values in this case age and height associated with each key
print(bio.columns)#returns the headings of columns
```

```
   age  height
a   15     6.0
b   19     5.2
c   35     5.6
d   70     6.1
Index(['a', 'b', 'c', 'd'], dtype='object')
[[15.   6. ]
 [19.   5.2]
 [35.   5.6]
 [70.   6.1]]
Index(['age', 'height'], dtype='object')
```

In [16]:
```
#constructing dataframes pandas
age=pd.Series({"a":15,"b":19,"c":35,"d":70})#seperate series for age
p=pd.DataFrame(age,columns=["age"])
print(p)
```

```
   age
a   15
b   19
c   35
d   70
```

In [29]:
```
#creating a dataframe from structured array
import numpy as np
chocolates=np.zeros(4,dtype=[("name","S10"),("price","i4")])#structured array
print(chcocolates)
panda=pd.DataFrame(chocolates)#passing structured array as argument will convert the array to pandas dataframe
print(panda)
```

```
[(b'', 0) (b'', 0) (b'', 0) (b'', 0)]
  name  price
0  b''      0
1  b''      0
2  b''      0
3  b''      0
```

In [43]:
```
#pandas index object is similar to immutable arrays
im=pd.Index([1,2,3,4,5])#implementing index from list of integers
print(im)
#pandas index object are similar to sets
ia=pd.Index([1,2,3,4,5,6])
ib=pd.Index([4,5,6,7,8,9])
iintersection=ia & ib#intersection of sets
print("Intersection of sets=")
print(iintersection)
iunion= ia | ib#union of sets
print("Union of sets=")
print(iunion)
unique= ia^ib#Unique elements
print("Unique elements of the sets=")
print(unique)
```

```
Int64Index([1, 2, 3, 4, 5], dtype='int64')
Intersection of sets=
Int64Index([4, 5, 6], dtype='int64')
Union of sets=
Int64Index([1, 2, 3, 4, 5, 6, 7, 8, 9], dtype='int64')
Unique elements of the sets=
Int64Index([1, 2, 3, 7, 8, 9], dtype='int64')

C:\Users\kdmag\AppData\Local\Temp\ipykernel_5000\1119667438.py:7: FutureWarning: Index.__and__ operating as a set operation is
deprecated, in the future this will be a logical operation matching Series.__and__.  Use index.intersection(other) instead.
  iintersection=ia & ib#intersection of sets
C:\Users\kdmag\AppData\Local\Temp\ipykernel_5000\1119667438.py:10: FutureWarning: Index.__or__ operating as a set operation is
deprecated, in the future this will be a logical operation matching Series.__or__.  Use index.union(other) instead.
  iunion= ia | ib#union of sets
C:\Users\kdmag\AppData\Local\Temp\ipykernel_5000\1119667438.py:13: FutureWarning: Index.__xor__ operating as a set operation is
deprecated, in the future this will be a logical operation matching Series.__xor__.  Use index.symmetric_difference(other) inst
ead.
  unique= ia^ib#Unique elements
```

In [44]:
```
#pandas index object is similar to immutable arrays
im=pd.Index([1,2,3,4,5])#implementing index from list of integers
print(im)
#pandas index object are similar to sets
ia=pd.Index([1,2,3,4,5,6])
ib=pd.Index([4,5,6,7,8,9])
#object methods
iintersection=ia.intersection(ib)#intersection of sets
print("Intersection of sets=")
print(iintersection)
iunion=ia.union(ib)#union of sets
print("Union of sets=")
print(iunion)
idifference= ia.difference(ib)#difference between sets
print("Difference between the sets=")
print(idifference)
```

```
Int64Index([1, 2, 3, 4, 5], dtype='int64')
Intersection of sets=
Int64Index([4, 5, 6], dtype='int64')
Union of sets=
Int64Index([1, 2, 3, 4, 5, 6, 7, 8, 9], dtype='int64')
Difference between the sets=
Int64Index([1, 2, 3], dtype='int64')
```