Ch03_Data Collections Structure

September 5, 2018

1 Chapter : Collections

```
In [1]: # Create List
        List1 = [1, 24, 76]
        print (List1)

        colors=['red', 'yellow', 'blue']
        print (colors)

        mix=['red', 24, 98.6]
        print (mix)

        nested= [ 1, [5, 6], 7]
        print (nested)

        print ([])
```

```
[1, 24, 76]
['red', 'yellow', 'blue']
['red', 24, 98.6]
[1, [5, 6], 7]
[]
```

```
In [9]: list1 = ['Egypt', 'chemistry', 2017, 2018];
        list2 = [1, 2, 3, [4, 5] ];
        list3 = ["a", 3.7, '330', "Omar"]

        print (list1[2])
        print (list2 [3:])
        print (list3 [-3:-1])
        print (list3[-3])
```

```
2017
[[4, 5]]
[3.7, '330']
3.7
```

```
In [50]: courses=["OOP","Networking","MIS","Project"]
         students=["Ahmed", "Ali", "Salim", "Abdullah", "Salwa"]
         OOP_marks = [65, 85, 92]

         OOP_marks.append(50)   # Add new element
         OOP_marks.append(77)# Add new element
         print (OOP_marks[ : ])  # Print list before updateing

         OOP_marks[0]=70   # update new element
         OOP_marks[1]=45   # update new element
         list1 = [88, 93]
         OOP_marks.extend(list1)     # extend list with another list
         print (OOP_marks[ : ])  # Print list after updateing

[65, 85, 92, 50, 77]
[70, 45, 92, 50, 77, 88, 93]


In [28]:

[70, 45, 92, 50, 77]


In [48]: OOP_marks = [70, 45, 92, 50, 77, 45]
         print (OOP_marks)

         del OOP_marks[0]    # delete an element using del
         print (OOP_marks)

         OOP_marks.remove (45)   # remove an element using remove() method
         print (OOP_marks)


         OOP_marks.pop (2)   # remove an element using remove() method
         print (OOP_marks)

[70, 45, 92, 50, 77, 45]
[45, 92, 50, 77, 45]
[92, 50, 77, 45]
[92, 50, 45]


In [42]: len([5, "Omar", 3])  # find the list length.
         [3, 4, 1] + ["O", 5, 6] # concatenate lists.
         ['Hi!'] * 4   # repeate an element in a list.
         3 in [1, 2, 3]   # check if element in a list
         for x in [1, 2, 3]: print (x) # traverse list elements

1
2
```

```
3
```

```
In [46]: print (len([5, "Omar", 3]))          # find the list length.
         print ([3, 4, 1] + ["Omar", 5, 6])   # concatenate lists.
         print (['Eg!'] * 4)                  # repeate an element in a list.
         print (3 in [1, 2, 3])              # check if element in a list
         for x in [1, 2, 3]: print (x, end='  ') # traverse list elements
```

```
3
[3, 4, 1, 'Omar', 5, 6]
['Eg!', 'Eg!', 'Eg!', 'Eg!']
True
1  2  3
```

```
In [51]: #Built-in Functions and Lists
         tickets = [3, 41, 12, 9, 74, 15]
         print (tickets)
         print (len(tickets))
         print (max(tickets))
         print (min(tickets))
         print (sum(tickets))
         print (sum(tickets)/len(tickets))
```

```
[3, 41, 12, 9, 74, 15]
6
74
3
154
25.666666666666668
```

```
In [58]: #List sorting and Traversing
         seq=(41, 12, 9, 74, 3, 15)   # use sequence for creating a list
         tickets=list(seq)

         print (tickets)
         tickets.sort()
         print (tickets)

         print ("\nSorted list elements ")
         for ticket in tickets:
             print (ticket)
```

```
[41, 12, 9, 74, 3, 15]
[3, 9, 12, 15, 41, 74]

Sorted list elements
3
```

```
9
12
15
41
74
```

1.1 LISTS AND STRINGS

```
In [63]: # convert string to a list of characters
         Word = 'Egypt'
         List1 = list(Word)
         print (List1)
```

```
['E', 'g', 'y', 'p', 't']
```

```
In [70]: # we can break a string into words using the split method
         Greeting= 'Welcome to Egypt'
         List2 =Greeting.split()
         print (List2)
         print (List2[2])
```

```
['Welcome', 'to', 'Egypt']
Egypt
```

```
In [69]: # use the delimiter
         Greeting= 'Welcome-to-Egypt'
         List2 =Greeting.split("-")
         print (List2)

         Greeting= 'Welcome-to-Egypt'
         delimiter='-'
         List2 =Greeting.split(delimiter)
         print (List2)
```

```
['Welcome', 'to', 'Egypt']
['Welcome', 'to', 'Egypt']
```

```
In [73]: List1 = ['Welcome', 'to', 'Egypt']
         delimiter = ' '
         delimiter.join(List1)
```

```
Out[73]: 'Welcome to Egypt'
```

```
In [74]: List1 = ['Welcome', 'to', 'Egypt']
         delimiter = '-'
         delimiter.join(List1)
```

```
Out[74]: 'Welcome-to-Egypt'

In [105]: filesdata="From oembarak@hct.ac.ae Sat Jan 5 09:14:16 2016 \
              mak.jon@ec.ac.ae Sat Jan 5 09:14:16 2011 \
              From ossama.embarak@ar.ac.eg Sat Jan 5 09:14:16 2010\
                          From usa.mak@gmail.com Jan 5 09:14:16 2015"
          #print (filesdata)
          for line in filesdata:
              #line = line.rstrip()
              if not line.startswith('From ') : continue
              words = line.split()
              print (words[2])

In [117]: a = [1, 2, 3]
          b = a
          print (a)
          print (b)

[1, 2, 3]
[1, 2, 3]


In [118]: a.append(77)
          print (a)
          print (b)

[1, 2, 3, 77]
[1, 2, 3, 77]


In [119]: b is a

Out[119]: True

In [120]: a = [1, 2, 3]
          b = [1, 2, 3]
          print (a)
          print (b)

[1, 2, 3]
[1, 2, 3]


In [121]: a.append(77)
          print (a)
          print (b)

[1, 2, 3, 77]
[1, 2, 3]
```

```
In [122]: b is a

Out[122]: False

In [124]: Students =["Ahmed", "Ali", "Salim", "Abdullah", "Salwa"]
          def displaynames (x):
              for name in x:
                  print (name)

          displaynames(Students)  # Call the function displaynames

Ahmed
Ali
Salim
Abdullah
Salwa
```

2 Dictionaries

```
In [36]: Prices = {"Honda":40000, "Suzuki":50000, "Mercedes":85000, "Nissan":35000, "Mitsubishi"
         print (Prices)

{'Honda': 40000, 'Suzuki': 50000, 'Mercedes': 85000, 'Nissan': 35000, 'Mitsubishi': 43000}


In [37]: Staff_Salary = { 'Omar Ahmed' : 30000 , 'Ali Ziad' : 24000, 'Ossama Hashim': 25000, 'Ma
         print(Staff_Salary)
         STDMarks={"Salwa Ahmed":50, "Abdullah Mohamed":80, "Sultan Ghanim":90}
         print(STDMarks)

{'Omar Ahmed': 30000, 'Ali Ziad': 24000, 'Ossama Hashim': 25000, 'Majid Hatem': 10000}
{'Salwa Ahmed': 50, 'Abdullah Mohamed': 80, 'Sultan Ghanim': 90}


In [38]: STDMarks = dict()
         STDMarks['Salwa Ahmed']=50
         STDMarks['Abdullah Mohamed']=80
         STDMarks['Sultan Ghanim']=90
         print (STDMarks)

{'Salwa Ahmed': 50, 'Abdullah Mohamed': 80, 'Sultan Ghanim': 90}


In [39]: STDMarks={"Salwa Ahmed":50, "Abdullah Mohamed":80, "Sultan Ghanim":90}
         STDMarks['Salwa Ahmed'] = 85   # update current value of the key 'Salwa Ahmed'
         STDMarks['Omar Majid'] = 74 # Add a new item to the dictionary
         print (STDMarks)
```

```
{'Salwa Ahmed': 85, 'Abdullah Mohamed': 80, 'Sultan Ghanim': 90, 'Omar Majid': 74}
```

```
In [40]: STDMarks={"Salwa Ahmed":50, "Abdullah Mohamed":80, "Sultan Ghanim":90}
         print (STDMarks)
         del STDMarks['Abdullah Mohamed'] # remove entry with key 'Abdullah Mohamed'
         print (STDMarks)
         STDMarks.clear()     # remove all entries in STDMarks dictionary
         print (STDMarks)
         del STDMarks         # delete entire dictionary
```

```
{'Salwa Ahmed': 50, 'Abdullah Mohamed': 80, 'Sultan Ghanim': 90}
{'Salwa Ahmed': 50, 'Sultan Ghanim': 90}
{}
```

```
In [2]: Staff_Salary = { 'Omar Ahmed' : 30000 , 'Ali Ziad' : 24000, 'Ossama Hashim': 25000, 'Maj
        print('Salary package for Ossama Hashim is ', end='')
        print(Staff_Salary['Ossama Hashim'])                    # access specific dictionary ele
```

```
Salary package for Ossama Hashim is 25000
```

```
In [3]: # Define a function to return salary after dicount tax 5%
        def Netsalary (salary):
            return salary - (salary * 0.05) # also could be retunr salary *0.95

        #iterate all elements in a dcitionary
        print ("Name     " , '\t', "Net Salary" )
        for key, value in Staff_Salary.items():
            print (key , '\t', Netsalary(value))
```

```
Name               Net Salary
Omar Ahmed          28500.0
Ali Ziad           22800.0
Ossama Hashim          23750.0
Majid Hatem          9500.0
```

```
In [43]: Staff_Salary = { 'Omar Ahmed' : 30000 , 'Ali Ziad' : 24000, 'Ossama Hashim': 25000, 'Ma
         STDMarks={"Salwa Ahmed":50, "Abdullah Mohamed":80, "Sultan Ghanim":90}
```

```
In [52]: def cmp(a, b):
             for key, value in a.items():
                 for key1, value1 in b.items():
                     return (key >key1) - (key < key1)
```

```
In [54]: print (cmp(STDMarks,Staff_Salary) )
         print (cmp(STDMarks,STDMarks) )
         print (len(STDMarks) )
         print (str(STDMarks) )
         print (type(STDMarks) )
```

```
1
0
3
{'Salwa Ahmed': 50, 'Abdullah Mohamed': 80, 'Sultan Ghanim': 90}
<class 'dict'>
```

In [ ]:

In [71]:
```
Staff_Salary = { 'Omar Ahmed' : 30000 , 'Ali Ziad' : 24000, 'Ossama Hashim': 25000, 'Ma
STDMarks={"Salwa Ahmed":50, "Abdullah Mohamed":80, "Sultan Ghanim":90}
dic3 = Staff_Salary.copy()
Staff_Salary.clear()    # clear all elements in Staff_Salary dictionary
print (Staff_Salary)
print (dic3)

dict1= dict()
sequence=('Id' , 'Number' , 'Email')
print (dict1.fromkeys(sequence))
print (dict1.fromkeys(sequence, '####'))
```

```
{}
{'Omar Ahmed': 30000, 'Ali Ziad': 24000, 'Ossama Hashim': 25000, 'Majid Hatem': 10000}
{'Id': None, 'Number': None, 'Email': None}
{'Id': '####', 'Number': '####', 'Email': '####'}
```

In [89]:
```
Staff_Salary = { 'Omar Ahmed' : 30000 , 'Ali Ziad' : 24000, 'Ossama Hashim': 25000, 'Ma
STDMarks={"Salwa Ahmed":50, "Abdullah Mohamed":80, "Sultan Ghanim":90}
print (Staff_Salary.get('Ali Ziad') )
print (STDMarks.items())
print (Staff_Salary.keys())

print()
STDMarks.setdefault('Ali Ziad')
print (STDMarks)
print (STDMarks.update(dict1))
print (STDMarks)
```

```
24000
dict_items([('Salwa Ahmed', 50), ('Abdullah Mohamed', 80), ('Sultan Ghanim', 90)])
dict_keys(['Omar Ahmed', 'Ali Ziad', 'Ossama Hashim', 'Majid Hatem'])

{'Salwa Ahmed': 50, 'Abdullah Mohamed': 80, 'Sultan Ghanim': 90, 'Ali Ziad': None}
None
{'Salwa Ahmed': 50, 'Abdullah Mohamed': 80, 'Sultan Ghanim': 90, 'Ali Ziad': None}
```

In [96]:
```
Staff_Salary = { 'Omar Ahmed' : 30000 , 'Ali Ziad' : 24000, 'Ossama Hashim': 25000, 'Ma
print ("\nSorted by key")
```

```
        for k in sorted(Staff_Salary):
            print (k, Staff_Salary[k])
```

```
Sorted by key
Ali Ziad 24000
Majid Hatem 10000
Omar Ahmed 30000
Ossama Hashim 25000
```

```
In [97]: Staff_Salary = { 'Omar Ahmed' : 30000 , 'Ali Ziad' : 24000, 'Ossama Hashim': 25000, 'Ma
         print ("\nSorted by value")
         for w in sorted(Staff_Salary, key=Staff_Salary.get, reverse=True):
           print (w, Staff_Salary[w])
```

```
Sorted by value
Omar Ahmed 30000
Ossama Hashim 25000
Ali Ziad 24000
Majid Hatem 10000
```

3 Tuples

```
In [1]: Names = ('Omar', 'Ali', 'Bahaa')
        Marks = ( 75, 65, 95 )

        print (Names[2])
        print (Marks)
        print (max(Marks))
```

```
Bahaa
(75, 65, 95)
95
```

```
In [2]: for name in Names:
            print (name)
```

```
Omar
Ali
Bahaa
```

```
In [3]: Marks[1]=66
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-3-b225998b9edb> in <module>()
----> 1 Marks[1]=66


TypeError: 'tuple' object does not support item assignment
```

```
In [4]: Names = ( 'Omar Ahmed', 'Ali Ziad' , 'Ossama Hashim', 'Majid Hatem')
        print (Names)
        Names.sort(reverse=True)
        print (Names)
```

```
('Omar Ahmed', 'Ali Ziad', 'Ossama Hashim', 'Majid Hatem')
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-4-bdedc3bd1752> in <module>()
  1 Names = ( 'Omar Ahmed', 'Ali Ziad' , 'Ossama Hashim', 'Majid Hatem')
  2 print (Names)
----> 3 Names.sort(reverse=True)
  4 print (Names)


AttributeError: 'tuple' object has no attribute 'sort'
```

```
In [9]: MarksCIS=(70,85,90)
        print (MarksCIS)
```

```
(70, 85, 55)
```

```
In [14]: MarksCIS.sort(key=lambda x: int(x[0]))
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-14-4beb6e32a492> in <module>()
```

```
          1
----> 2 MarksCIS.sort(key=lambda x: int(x[0]))


AttributeError: 'tuple' object has no attribute 'sort'
```

```
In [1]: import operator
        MarksCIS = [(88,65),(70,90,85), (55,88,44)]
        print (MarksCIS)  # original tuples
        print (sorted(MarksCIS)) # direct sorting
```

```
[(88, 65), (70, 90, 85), (55, 88, 44)]
[(55, 88, 44), (70, 90, 85), (88, 65)]
```

```
In [2]: print (MarksCIS)  # original tuples
        #create a new sorted tuple
        MarksCIS2 = sorted(MarksCIS, key=lambda x: (x[0], x[1]))
        print (MarksCIS2)
```

```
[(88, 65), (70, 90, 85), (55, 88, 44)]
[(55, 88, 44), (70, 90, 85), (88, 65)]
```

```
In [3]: print (MarksCIS)  # original tuples
        MarksCIS.sort(key=lambda x: (x[0], x[1])) # sort in tuple
        print (MarksCIS)
```

```
[(88, 65), (70, 90, 85), (55, 88, 44)]
[(55, 88, 44), (70, 90, 85), (88, 65)]
```

```
In [4]: MarksCIS = (70, 85, 55)
        MarksCIN = (90, 75, 60)
        print ("The third mark in CIS is ", MarksCIS[2])
        print ("The third mark in CIN is ", MarksCIN[2])
```

```
The third mark in CIS is  55
The third mark in CIN is  60
```

```
In [5]: MarksCIN = (90, 75, 60)
        print (MarksCIN)
        del MarksCIN
        print (MarksCIN)
```

```
(90, 75, 60)
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-5-4c08fec39768> in <module>()
  2 print (MarksCIN)
  3 del MarksCIN
----> 4 print (MarksCIN)


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

```
In [6]: MarksCIS = (88, 65, 70,90,85,45,78,95,55)
        print ("\nForward slicing")
        print (MarksCIS[1:4])
        print (MarksCIS[:3])
        print (MarksCIS[6:])
        print (MarksCIS[4:6])

        print ("\nBackward slicing")
        print (MarksCIS[-4:-2])
        print (MarksCIS[-3])
        print (MarksCIS[-3:])
        print (MarksCIS[ :-3])
```

```
Forward slicing
(65, 70, 90)
(88, 65, 70)
(78, 95, 55)
(85, 45)

Backward slicing
(45, 78)
78
(78, 95, 55)
(88, 65, 70, 90, 85, 45)
```

```
In [8]: import operator
        MarksCIS = [(88,65),(70,90,85), (55,88,44)]
        print (MarksCIS)  # original tuples
        print (sorted(MarksCIS)) # direct sorting


        MarksCIS2 = sorted(MarksCIS, key=lambda x: (x[0], x[1]))
```

```
        print (MarksCIS2)

        MarksCIS.sort(key=lambda x: (x[0], x[1]))  # sorts in place
        print (MarksCIS)
```

```
[(88, 65), (70, 90, 85)]
[(70, 90, 85), (88, 65)]
[(70, 90, 85), (88, 65)]
```

```
        ---------------------------------------------------------------------------

        AttributeError                            Traceback (most recent call last)

        <ipython-input-8-718c88fe2e49> in <module>()
          9 print (MarksCIS2)
         10
    ---> 11 MarksCIS.sort(key=lambda x: (x[0], x[1].lower()))  # sorts in place
         12 print (MarksCIS)


        <ipython-input-8-718c88fe2e49> in <lambda>(x)
          9 print (MarksCIS2)
         10
    ---> 11 MarksCIS.sort(key=lambda x: (x[0], x[1].lower()))  # sorts in place
         12 print (MarksCIS)


        AttributeError: 'int' object has no attribute 'lower'
```

```
In [ ]: students = [
            ('John', 'A', 2),
            ('Zoro', 'C', 1),
            ('Dave', 'B', 3),
        ]
        print(students)
```

```
In [5]: MarksCIS=(70,85,55)
        MarksCIN=(90,75,60)
        Combind=MarksCIS+MarksCIN
        print (Combind)
```

```
(70, 85, 55, 90, 75, 60)
```

4 a series from ndarray with labels.

```
In [8]: import numpy as np
        import pandas as pd
        Series1 = pd.Series(np.random.randn(4), index=['a', 'b', 'c', 'd'])
        print(Series1)
        print(Series1.index)
```

```
a    0.350241
b   -1.214802
c    0.704124
d    0.866934
dtype: float64
Index(['a', 'b', 'c', 'd'], dtype='object')
```

```
In [9]: import numpy as np
        import pandas as pd
        Series2 = pd.Series(np.random.randn(4))
        print(Series2)
        print(Series2.index)
```

```
0    1.784219
1   -0.627832
2    0.429453
3   -0.473971
dtype: float64
RangeIndex(start=0, stop=4, step=1)
```

```
In [10]: print (" \n Series slicing ")
         print (Series1[:3])
         print ("\nIndex accessing")
         print (Series1[[3,1,0]])
         print ("\nSingle index")
         x = Series1[0]
         print (x)
```

```
 Series slicing
a    0.350241
b   -1.214802
c    0.704124
dtype: float64

Index accessing
d    0.866934
b   -1.214802
a    0.350241
```

```
dtype: float64

Single index
0.35024081401881596
```

```
In [19]: print ("\nSeries  Sample operations")
         print ("\n Series values greater than the mean: %.4f" % Series1.mean())
         print (Series1 [Series1> Series1.mean()])
         print ("\n Series values greater than the Meadian:%.4f" % Series1.median())
         print (Series1 [Series1> Series1.median()])
         print ("\nExponential value ")
         Series1Exp = np.exp(Series1)
         print (Series1Exp)
```

```
Series  Sample operations

 Series values greater than the mean: 0.1766
a    0.350241
c    0.704124
d    0.866934
dtype: float64

 Series values greater than the Meadian:0.5272
c    0.704124
d    0.866934
dtype: float64

Exponential value
a    1.419409
b    0.296769
c    2.022075
d    2.379604
dtype: float64
```

```
In [12]: dict = {'m' : 2, 'y' : 2018, 'd' : 'Sunday'}
         print ("\nSeries of non declared index")
         SeriesDict1 = pd.Series(dict)
         print(SeriesDict1)

         print ("\nSeries of declared index")
         SeriesDict2 = pd.Series(dict, index=['y', 'm', 'd', 's'])
         print(SeriesDict2)
```

```
Series of non declared index
d    Sunday
```

```
m         2
y      2018
dtype: object

Series of declared index
y      2018
m         2
d    Sunday
s       NaN
dtype: object
```

```
In [13]: print ("\nUse the get and set methods to access"
                 "a series values by index label\n")
         SeriesDict2 = pd.Series(dict, index=['y', 'm', 'd', 's'])
         print (SeriesDict2['y']) # Display the year
         SeriesDict2['y']=1999    # change the year vlaue
         print (SeriesDict2)      # Display all dictionary values
         print (SeriesDict2.get('y'))  # get specific value by its key
```

```
Use the get and set methods to accessa series values by index label

2018
y      1999
m         2
d    Sunday
s       NaN
dtype: object
1999
```

```
In [14]: print ("\n CREATE SERIES FORM SCALAR VALUE ")
         Scl = pd.Series(8., index=['a', 'b', 'c', 'd'])
         print (Scl)
```

```
 CREATE SERIES FORM SCALAR VALUE
a    8.0
b    8.0
c    8.0
d    8.0
dtype: float64
```

```
In [18]: SerX = pd.Series([1,2,3,4], index=['a', 'b', 'c', 'd'])
         print ("Addition");
         print( SerX + SerX)
         print ("Addition with non matched labels");
```

```
          print (SerX[1:] + SerX[:-1])
          print ("Multiplication");
          print (SerX * SerX)
          print ("Expponential");
          print (np.exp(SerX))
```

```
Addition
a    2
b    4
c    6
d    8
dtype: int64
Addition with non matched labels
a    NaN
b    4.0
c    6.0
d    NaN
dtype: float64
Multiplication
a     1
b     4
c     9
d    16
dtype: int64
Expponential
a     2.718282
b     7.389056
c    20.085537
d    54.598150
dtype: float64
```

```
In [17]: std = pd.Series([77,89,65,90], name='StudentsMarks')
          print (std.name)
          std = std.rename("Marks")
          print (std.name)
```

```
StudentsMarks
Marks
```

```
In [4]: #  read data from file and add it to dictionary for processing
        handle = open("Egypt.txt")
        text = handle.read()
        words = text.split()
        #print (words)
        counts = dict()
        for word in words:
            counts[word] = counts.get(word,0) + 1
```

```
        print (counts)
        bigcount = None
        bigword = None
        for word,count in counts.items():
            if bigcount is None or count > bigcount:
                bigword = word
                bigcount = count
        print ("\n bigword and  bigcount")
        print (bigword, bigcount)
```

```
{'Egypt,': 1, 'a': 2, 'country': 1, 'linking': 1, 'northeast': 1, 'Africa': 1, 'with': 1, 'the':

 bigword and  bigcount
the 6
```

```
In [14]: print ((100, 1, 2) > (150, 1, 2))
         print ((0, 1, 120) < (0, 3, 4))
         print (( 'Javed', 'Salwa' ) > ('Omar', 'Sam'))
         print (( 'Khalid', 'Ahmed') < ('Ziad', 'Majid'))
```

```
False
True
False
True
```

```
In [5]: import pandas as pd
        dict1 = {'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
                 'two' : pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}
        df = pd.DataFrame(dict1)
        df
```

```
Out[5]:    one  two
        a  1.0  1.0
        b  2.0  2.0
        c  3.0  3.0
        d  NaN  4.0
```

```
In [6]: # set index for DataFrame
        pd.DataFrame(dict1, index=['d', 'b', 'a'])
```

```
Out[6]:    one  two
        d  NaN  4.0
        b  2.0  2.0
        a  1.0  1.0
```

```
In [8]: # Control the labels appearance of the DataFrame
        pd.DataFrame(dict1, index=['d', 'b', 'a'], columns=['two', 'three', 'one'])
```

```
Out[8]:    two three  one
        d  4.0   NaN  NaN
        b  2.0   NaN  2.0
        a  1.0   NaN  1.0

In [11]: # without index
         ndarrdict = {'one' : [1., 2., 3., 4.],
            'two' : [4., 3., 2., 1.]}
         pd.DataFrame(ndarrdict)

Out[11]:    one  two
         0  1.0  4.0
         1  2.0  3.0
         2  3.0  2.0
         3  4.0  1.0

In [12]: # Assign index
         pd.DataFrame(ndarrdict, index=['a', 'b', 'c', 'd'])

Out[12]:    one  two
         a  1.0  4.0
         b  2.0  3.0
         c  3.0  2.0
         d  4.0  1.0

In [18]: import pandas as pd
         import numpy as np
         data = np.zeros((2,), dtype=[('A', 'i4'),('B', 'f4'),('C', 'a10')])
         data[:] = [(1,2.,'Hello'), (2,3.,"World")]
         pd.DataFrame(data)

Out[18]:    A    B              C
         0  1  2.0       b'Hello'
         1  2  3.0  b'WorldWorld'

In [16]: pd.DataFrame(data, index=['First', 'Second'])

Out[16]:          A    B         C
         First    1  2.0  b'Hello'
         Second   2  3.0  b'World'

In [17]: pd.DataFrame(data, columns=['C', 'A', 'B'])

Out[17]:           C  A    B
         0  b'Hello'  1  2.0
         1  b'World'  2  3.0

In [19]: data2 = [{'A': 1, 'B': 2}, {'A': 5, 'B': 10, 'C': 20}]
         pd.DataFrame(data2)
```

```
Out[19]:    A   B     C
         0  1   2   NaN
         1  5  10  20.0

In [20]: pd.DataFrame(data2, index=['First', 'Second'])

Out[20]:         A   B     C
         First   1   2   NaN
         Second  5  10  20.0

In [21]: pd.DataFrame(data2, columns=['A', 'B'])

Out[21]:    A   B
         0  1   2
         1  5  10

In [22]:  pd.DataFrame({('a', 'b'): {('A', 'B'): 1, ('A', 'C'): 2},
                        ('a', 'a'): {('A', 'C'): 3, ('A', 'B'): 4},
                        ('a', 'c'): {('A', 'B'): 5, ('A', 'C'): 6},
                        ('b', 'a'): {('A', 'C'): 7, ('A', 'B'): 8},
                        ('b', 'b'): {('A', 'D'): 9, ('A', 'B'): 10}})

Out[22]:         a              b
                 a    b    c    a     b
         A B   4.0  1.0  5.0  8.0  10.0
           C   3.0  2.0  6.0  7.0   NaN
           D   NaN  NaN  NaN  NaN   9.0

In [25]: # DATAFRAME COLUMN SELECTION, ADDITION, DELETION
         ndarrdict = {'one' : [1., 2., 3., 4.],
            'two' : [4., 3., 2., 1.]}
         df = pd.DataFrame(ndarrdict, index=['a', 'b', 'c', 'd'])
         df

Out[25]:    one  two
         a  1.0  4.0
         b  2.0  3.0
         c  3.0  2.0
         d  4.0  1.0

In [26]: df['three'] = df['one'] * df['two'] # Add column
         df['flag'] = df['one'] > 2          # Add column
         df

Out[26]:    one  two  three   flag
         a  1.0  4.0    4.0  False
         b  2.0  3.0    6.0  False
         c  3.0  2.0    6.0   True
         d  4.0  1.0    4.0   True
```

```
In [27]: df['Filler'] = 'HCT'
         df['Slic'] = df['one'][:2]
         df

Out[27]:    one  two  three   flag Filler  Slic
         a  1.0  4.0    4.0  False    HCT   1.0
         b  2.0  3.0    6.0  False    HCT   2.0
         c  3.0  2.0    6.0   True    HCT   NaN
         d  4.0  1.0    4.0   True    HCT   NaN

In [28]: # Delet columns
         del df['two']
         Three = df.pop('three')
         df

Out[28]:    one   flag Filler  Slic
         a  1.0  False    HCT   1.0
         b  2.0  False    HCT   2.0
         c  3.0   True    HCT   NaN
         d  4.0   True    HCT   NaN

In [29]: df.insert(1, 'bar', df['one'])
         df

Out[29]:    one  bar   flag Filler  Slic
         a  1.0  1.0  False    HCT   1.0
         b  2.0  2.0  False    HCT   2.0
         c  3.0  3.0   True    HCT   NaN
         d  4.0  4.0   True    HCT   NaN

In [54]: import numpy as np
         import pandas as pd
         df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
         df = df.assign(C=lambda x: x['A'] + x['B'])
         df = df.assign( D=lambda x: x['A'] + x['C'])
         df

Out[54]:    A  B  C   D
         0  1  4  5   6
         1  2  5  7   9
         2  3  6  9  12

In [55]: df = df.assign( A=lambda x: x['A'] *2)
         df

Out[55]:    A  B  C   D
         0  2  4  5   6
         1  4  5  7   9
         2  6  6  9  12
```

```
In [56]: df

Out[56]:    A  B  C   D
         0  2  4  5   6
         1  4  5  7   9
         2  6  6  9  12

In [61]: df['B']

Out[61]: 0    4
         1    5
         2    6
         Name: B, dtype: int64

In [59]: df.iloc[2]

Out[59]: A     6
         B     6
         C     9
         D    12
         Name: 2, dtype: int64

In [62]: df[1:]

Out[62]:    A  B  C   D
         1  4  5  7   9
         2  6  6  9  12

In [65]: df[df['C']>7]

Out[65]:    A  B  C   D
         2  6  6  9  12

In [69]: df1 = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
         df2 = pd.DataFrame({"A": [7, 4, 6], "B": [10, 4, 15]})
         print (df1)
         print()
         print(df2)

   A  B
0  1  4
1  2  5
2  3  6

   A   B
0  7  10
1  4   4
2  6  15
```

```
In [70]: df1+df2

Out[70]:    A   B
         0  8  14
         1  6   9
         2  9  21

In [71]: df1-df2

Out[71]:    A  B
         0 -6 -6
         1 -2  1
         2 -3 -9

In [72]: df2 - df1.iloc[2]

Out[72]:    A  B
         0  4  4
         1  1 -2
         2  3  9

In [75]: df2

Out[75]:    A   B
         0  7  10
         1  4   4
         2  6  15

In [78]: df2*2+1

Out[78]:     A   B
         0  15  21
         1   9   9
         2  13  31

In [3]: import pandas as pd
        import numpy as np
        P1 = pd.Panel(np.random.randn(2, 5, 4), items=['Item1', 'Item2'],
                        major_axis=pd.date_range('10/05/2018', periods=5),
                        minor_axis=['A', 'B', 'C', 'D'])
        P1

Out[3]: <class 'pandas.core.panel.Panel'>
        Dimensions: 2 (items) x 5 (major_axis) x 4 (minor_axis)
        Items axis: Item1 to Item2
        Major_axis axis: 2018-10-05 00:00:00 to 2018-10-09 00:00:00
        Minor_axis axis: A to D

In [4]: data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
                'Item2' : pd.DataFrame(np.random.randn(4, 2))}
        P2 = pd.Panel(data)
        P2
```

```
Out[4]: <class 'pandas.core.panel.Panel'>
        Dimensions: 2 (items) x 4 (major_axis) x 3 (minor_axis)
        Items axis: Item1 to Item2
        Major_axis axis: 0 to 3
        Minor_axis axis: 0 to 2

In [5]: p3 = pd.Panel.from_dict(data, orient='minor')
        p3

/home/nbuser/anaconda3_501/lib/python3.6/site-packages/ipykernel/__main__.py:1: DeprecationWarni
Panel is deprecated and will be removed in a future version.
The recommended way to represent these types of 3-dimensional data are with a MultiIndex on a Da
Alternatively, you can use the xarray package http://xarray.pydata.org/en/stable/.
Pandas provides a `.to_xarray()` method to help automate this conversion.

  if __name__ == '__main__':


Out[5]: <class 'pandas.core.panel.Panel'>
        Dimensions: 3 (items) x 4 (major_axis) x 2 (minor_axis)
        Items axis: 0 to 2
        Major_axis axis: 0 to 3
        Minor_axis axis: Item1 to Item2

In [26]: df = pd.DataFrame({'Item': ['TV', 'Mobile', 'Laptop'],
                            'Price': np.random.randn(3)**2*1000})
         df

Out[26]:      Item        Price
         0      TV  3704.932147
         1  Mobile  1348.142561
         2  Laptop   336.985518

In [29]: data = {'stock1': df, 'stock2': df}
         panel = pd.Panel.from_dict(data, orient='minor')
         panel['Item']

/home/nbuser/anaconda3_501/lib/python3.6/site-packages/ipykernel/__main__.py:2: DeprecationWarni
Panel is deprecated and will be removed in a future version.
The recommended way to represent these types of 3-dimensional data are with a MultiIndex on a Da
Alternatively, you can use the xarray package http://xarray.pydata.org/en/stable/.
Pandas provides a `.to_xarray()` method to help automate this conversion.

  from ipykernel import kernelapp as app


Out[29]:    stock1  stock2
         0      TV      TV
         1  Mobile  Mobile
         2  Laptop  Laptop
```

```
In [30]: wp['Price']
```

```
Out[30]:         stock1       stock2
         0  3704.932147  3704.932147
         1  1348.142561  1348.142561
         2   336.985518   336.985518
```

```
In [33]: import pandas as pd
         import numpy as np
         P1 = pd.Panel(np.random.randn(2, 5, 4), items=['Item1', 'Item2'],
                         major_axis=pd.date_range('10/05/2018', periods=5),
                         minor_axis=['A', 'B', 'C', 'D'])
         P1['Item1']
```

```
Out[33]:                    A         B         C         D
         2018-10-05 -0.794656  1.082396 -0.368632  0.360976
         2018-10-06 -0.281474  0.070584 -0.012636 -0.388089
         2018-10-07  1.653752  0.487939  1.838114 -0.832078
         2018-10-08 -0.145535  1.856141  0.107239  0.462018
         2018-10-09 -0.816565  2.195793 -0.871674 -1.226616
```

```
In [34]: P1.major_xs(P1.major_axis[2])
```

```
Out[34]:       Item1     Item2
         A  1.653752 -0.496110
         B  0.487939  0.990550
         C  1.838114  1.492156
         D -0.832078 -0.197148
```

```
In [35]: P1.minor_axis
```

```
Out[35]: Index(['A', 'B', 'C', 'D'], dtype='object')
```

```
In [36]: P1.minor_xs('C')
```

```
Out[36]:                Item1     Item2
         2018-10-05 -0.368632 -0.989085
         2018-10-06 -0.012636  0.266520
         2018-10-07  1.838114  1.492156
         2018-10-08  0.107239 -0.555847
         2018-10-09 -0.871674 -0.468046
```

```
In [28]: data = {'Omar': 2.5, 'Ali': 3.5, 'Osama': 3.0}
         pd.Series(data)
```

```
Out[28]: Ali      3.5
         Omar     2.5
         Osama    3.0
         dtype: float64
```

```
In [30]: pd.Series(data, index = ['Omar', 'Ali', 'Osama'])

Out[30]: Omar     2.5
         Ali      3.5
         Osama    3.0
         dtype: float64

In [31]: data = {'Omar': [90, 50, 89],
                 'Ali': [78, 75, 73],
                 'Osama': [67, 85, 80]}
         df1 = pd.DataFrame (data, index= ['Course1', 'Course2', 'Course3'])
         df1

Out[31]:           Ali  Omar  Osama
         Course1   78    90     67
         Course2   75    50     85
         Course3   73    89     80

In [32]: df1['Omar']

Out[32]: Course1    90
         Course2    50
         Course3    89
         Name: Omar, dtype: int64

In [33]: df1['Mean'] = (df1['Ali'] + df1['Omar'] + df1['Osama'])/3
         df1

Out[33]:           Ali  Omar  Osama       Mean
         Course1   78    90     67  78.333333
         Course2   75    50     85  70.000000
         Course3   73    89     80  80.666667
```