Engineering Analytics and Machine Learning Lab 2

Python Data structure, Numpy and Panda

The most basic data structure in Python is the sequence. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.

Python has six built-in types of sequences, but the most common ones are lists and tuples, which we would see in this tutorial.

There are certain things you can do with all the sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership. In addition, Python has built-in functions for finding the length of a sequence and for finding its largest and smallest elements.

1 Python Lists

The list is the most versatile datatype available in Python, which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that the items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets.

In [1]:
```
#list creation

list1=["hello","world","myfriend"]
list2=[1,2,3,4,5,6]
list3=["a","b","c","d"]

print(list1)
print(list2)
print(list3)
```

```
['hello', 'world', 'myfriend']
[1, 2, 3, 4, 5, 6]
['a', 'b', 'c', 'd']
```

2 Accessing values in List

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. Please note that the index of list start from 0.

In [2]:
```
#examples of access values in list
print(list1[0])
print(list2[3])
print(list3[2])
```

```
hello
4
c
```

In [3]:
```
#example to update the value of a particular element in the list

list1[0]='how'
list2[3]=10
list3[2]='z'

#print the  content of the three  lists
#take note of respective element been updated

print(list1)
print(list2)
print(list3)
```

```
['how', 'world', 'myfriend']
[1, 2, 3, 10, 5, 6]
['a', 'b', 'z', 'd']
```

In [4]:
```
#example of deleting element in a list

list4=[2,3,4,5,6,7]
print("list content before delete",list4)
del list4[1]
print("list content after delete",list4)
```

```
list content before delete [2, 3, 4, 5, 6, 7]
list content after delete [2, 4, 5, 6, 7]
```

3 Basic List Operation

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

In fact, lists respond to all of the general sequence operations we used on strings in the prior chapter.

In [12]:
```
#example of operation for List

list4=[2,3,4,5,6,7]
list5=[12,13,14]

print("The lenght of the list is",len(list4))
print("We could performation Concatenation of two lists with result",list4+lis
t5)
print("Reptition by * operator,result of list4*3 is ",list4*3)
print("We could find whether a particular value is an member of a list by. Exa
mple if 3 is in list4 is ",2 in list4)
print("We could find whether a particular value is an member of a list by. Exa
mple if 3 is in list5 is ",2 in list5)

#iteration can be done by

for x in list4 : print (x,end = ' ')  # print for every value in list4 with a
 space at the end
```

```
The lenght of the list is 6
We could performation Concatenation of two lists with result [2, 3, 4, 5, 6,
7, 12, 13, 14]
Reptition by * operator,result of list4*3 is  [2, 3, 4, 5, 6, 7, 2, 3, 4, 5,
6, 7, 2, 3, 4, 5, 6, 7]
We could find whether a particular value is an member of a list by. Example i
f 3 is in list4 is  True
We could find whether a particular value is an member of a list by. Example i
f 3 is in list5 is  False
2 3 4 5 6 7
```

In [19]:
```
#slicing and other function which is similar to string

list6=['This','is','Python']

print(list6[0])   #zero based
print(list6[-1])  #count from behind
print(list6[:2])  #slicing from start to index 2 (exclude)
print(list6[1:3]) #slicing from index 1 (inclusive) to 3 (exclusive)
print(len(list6)) #lenght of list6
print(max(list5)) #max value of list5
print(min(list5)) #min value of list5
```

```
This
Python
['This', 'is']
['is', 'Python']
3
14
12
```

Exercise 1

Write a function to compute the mean and standard deviation of a given list of number.

In [52]:
```
def com_value(inlist):
    #your code start here
    mean=sum(inlist)/len(inlist)
    std=0.0
    for x in inlist:
        std=(x-mean)**2+std
    std=math.sqrt(std/(len(inlist)-1))

    return mean,std
    #end here

#test code, do not change anything here
test1=[1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0]
ans=com_value(test1)
print(ans)
if(ans[0]==5.5 and ans[1]==3.0276503540974917):
    print("Passed, correct answer")
else:
    print("Wrong, try harder")
```

```
(5.5, 3.0276503540974917)
Passed, correct answer
```

4 List methods

Below are a few commonly used Python list methods that are useful.

In [65]:
```
#list methhods
list7=['I','Love','Python']
list7.append('like')  #append a new value to the list
print('Object like is appended to the end of list7 reuslt in: ',list7)
list7.append('mad')   #append another new  value to the list
print('Append object mad in list7, result: ',list7)

print('The number of count mad occour in list 7 is: ',list7.count('mad'))   #o
unt of how many times obj 'mad' occurs in list
print('The lowest index of mad appear in list7 is: ',list7.index('mad'))

list8=list(range(6))      #create a list of number from 0 to 5
print('list with number from 0 to 5', list8)

list7.extend(list8)     #extend list7 with list8
print('Extended list7 contain: ',list7) #print the extended list7

list7.insert(1,'hate')  #insert a new object 'hate' at index 1
print('Content of list7 after Object hate is added: ', list7)

list7.remove('Love')   #remove the object 'Love'
print('Remove the oject Love from list7: ',list7)

list7.reverse()   #reverse list 7
print('Reversed list7 result: ',list7)
```

```
Ojbect like is appended to the end of list7 reuslt in:  ['I', 'Love', 'Pytho
n', 'like']
Append object mad in list7, result:  ['I', 'Love', 'Python', 'like', 'mad']
The number of count mad occour in list 7 is:  1
The lowest index of mad appear in list7 is:  4
list with number from 0 to 5 [0, 1, 2, 3, 4, 5]
Extended list7 contain:  ['I', 'Love', 'Python', 'like', 'mad', 0, 1, 2, 3,
4, 5]
Content of list7 after Object hate is added:  ['I', 'hate', 'Love', 'Python',
'like', 'mad', 0, 1, 2, 3, 4, 5]
Remove the oject Love from list7:  ['I', 'hate', 'Python', 'like', 'mad', 0,
1, 2, 3, 4, 5]
Reversed list7 result:  [5, 4, 3, 2, 1, 0, 'mad', 'like', 'Python', 'hate',
'I']
```

Exercise 2

Write a function to compute do a words count of unique words provided in a list. The function should return a list of unique words and a list of corresponding count.

In [79]:
```
def word_count(nlist):
    unique=[]
    unique_count=[]
    for x in nlist:
        if x not in unique:
            unique.append(x)

    for y in unique:
        c=nlist.count(y)
        print(y)
        unique_count.append(c)
    return unique,unique_count

test=['a','ab','a','c','ab','a','v','c']
q,q_count=word_count(test)
print(q)
print(q_count)
```

```
a
ab
c
v
['a', 'ab', 'c', 'v']
[3, 2, 2, 1]
```

Exercise 3

Write a program that accept a string with number seperated by space and produce a list wit these number. E.g. "1 2 4 5 34 55" -->[1,2,4,5,34,55] (Hint: use split method)

In [2]:
```
str1='1 2 4 5 34 55'
num=[int(x) for x in str1.split()]
print(num)
```

```
[1, 2, 4, 5, 34, 55]
```

1.5 Python Tuples

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The main difference between the tuples and the lists is that the tuples cannot be changed unlike lists. Tuples use parentheses, whereas lists use square brackets.

In [2]:
```
#tuples creation

tup1 = (1, 2, 3, 4, 5 )    #round  bracket is used to initialze tuples
tup2 = "a", "b", "c", "d"
tup3 = ('electrical', 'digitial', 2007, 2018)  #tuples can have a mixed  type
 of content
tup4=()    #a empty tuple

print(tup1)
print(tup2)
print(tup3)
print(tup4)
```

```
(1, 2, 3, 4, 5)
('a', 'b', 'c', 'd')
('electrical', 'digitial', 2007, 2018)
()
```

Accessing Access Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain the value available at that index.

In [3]:
```
#access value in tuples please make sure that the previous code cell is run be
fore running this one

print(tup1[4])
print(tup2[:3])
print(tup3[1:])
```

```
5
('a', 'b', 'c')
('digitial', 2007, 2018)
```

Updating Tuples

Tuples are immutable, which means you cannot update or change the values of tuple elements. You are able to take portions of the existing tuples to create new tuples

In [7]:
```
#tup1[0]=2 #this command would give us error

tup5=tup1[2:]+tup3[:3]

print(tup5)
```

```
(3, 4, 5, 'electrical', 'digitial', 2007)
```

Exercise 4

An email address is provided in variable called email. Use tuple assignment to seperate the username and domain from the email address. (Hint: Use split method)

In [2]:
```
#initialize the email address
email='hello@enganlyml.com'

#your code start there
username,domain=email.split('@')

print(username)
print(domain)
```

```
hello
enganlyml.com
```

1.6 Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values. Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Let's see an example of a dictionary shown in the cell below:

In [10]:
```
thisdict =      {
  "apple": "green",
  "banana": "yellow",
  "cherry": "red"
}
print(thisdict)
```

```
{'banana': 'yellow', 'apple': 'green', 'cherry': 'red'}
```

Dictionary with mixed keys

In [4]:
```
dict1={
    'name':'John',
    3:['hello',1,2],
}
print(dict1)
```

```
{3: ['hello', 1, 2], 'name': 'John'}
```

In [6]:
```
#more example of dictionary initialization

d1={1:'amber',2:'Jenny',3:'Amy'}
print(d1)
#to access individual value, we need to use the key to access
print(d1[1])
print(d1[2])

#for mixed key like dict1 (please make sure you had already run previous cell)
print(dict1['name'])
print(dict1[3])
```

```
{1: 'amber', 2: 'Jenny', 3: 'Amy'}
amber
Jenny
John
['hello', 1, 2]
```

Update of Dictionary

Dictionary in python are mutable. That mean the value unlike those in tuple and string can be updated, added or deleted. If the key specified is present in the dictionary, then the associated value with that key is updated or change, otherwiese a new key:value pair is added or created.

In [7]:
```
#please ensure the previous is run successfully as we are using dictionary cre
ated in previous cell

d1[1]='Alice'   #update the value of key 1 with 'Alice'
print(d1)
d1[4]='Mary'    #Add a new key pair 4:Mary
print(d1)
m=d1.pop(4)      #use pop method to remove key:value pair with key=2, the meth
od return the value of the deleted item
print(d1)
print(m)
```

```
{1: 'Alice', 2: 'Jenny', 3: 'Amy'}
{1: 'Alice', 2: 'Jenny', 3: 'Amy', 4: 'Mary'}
{1: 'Alice', 2: 'Jenny', 3: 'Amy'}
Mary
```

Tranversing Dictionary

We have learn about tranversing strings and etc, it is perhap right for us to learn about transversing Dictionary too.

In [9]:
```
#lets use the d1 dictionary defined in the previous cell

for c in d1:
    print(c,d1[c])
```

```
1 Alice
2 Jenny
3 Amy
```

Membership

Using the membership operator in and not in, we can test whether a key is in the dictionary or not.

In [11]:
```
1 in d1  # test whether 1 is key of d1
10 in d1 # test whether 10 is key of d1
```

Out[11]: False

Exercise 5

A company wanted to write a simple python program to score potential candiates skill according to the company skill matrix as shown below:

skill	Weightage
python	3
deeplearning	5
excel	1
c++	3
tensorflow	4

The candidates would be stored in tuple with their skills such as ['python','deeplearning','excel','c++','tensorflow'] would would achieve a scare of 2.3333

In [18]:
```
d2={'python':3,
    'deeplearning':5,
   'excel':1,
   'c++':3,
    'tensorflow':4
   }

skill=['python',
     'excel',
     'c++']
score=0
for a in skill:
    if a in d2:
        score=score+d2[a]
score=score/len(skill)
print(score)
```

```
2.3333333333333335
```

2 Numpy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities
- Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.
- Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

http://www.numpy.org/ (http://www.numpy.org/)

Array types and conversions between types NumPy supports a much greater variety of numerical types than Python does. This section shows which are available, and how to modify an array's data-type.

Data type Description

- bool_ Boolean (True or False) stored as a byte
- int_ Default integer type (same as C long; normally either int64 or int32)
- intc Identical to C int (normally int32 or int64)
- intp Integer used for indexing (same as C ssize_t; normally either int32 or int64)
- int8 Byte (-128 to 127)
- int16 Integer (-32768 to 32767)
- int32 Integer (-2147483648 to 2147483647)
- int64 Integer (-9223372036854775808 to 9223372036854775807)
- uint8 Unsigned integer (0 to 255)
- uint16 Unsigned integer (0 to 65535)
- uint32 Unsigned integer (0 to 4294967295)
- uint64 Unsigned integer (0 to 18446744073709551615)
- float_ Shorthand for float64.
- float16 Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
- float32 Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
- float64 Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
- complex_ Shorthand for complex128.
- complex64 Complex number, represented by two 32-bit floats (real and imaginary components)
- complex128 Complex number, represented by two 64-bit floats (real and imaginary components)

source: https://docs.scipy.org/doc/numpy/user/basics.types.html (https://docs.scipy.org/doc/numpy/user/basics.types.html)

2.1 Basic of Numpy Arrays

In [30]:
```
# to use numpy we need to import the library
import numpy as np

#Lets define some numpy arrays

np.random.seed(0)  #seed for reproducibility

#initialize a numpy one-dimension array with 6 elements with random number fro
m 0 to 9
x1=np.random.randint(10,size=6)
#each numpy array come with a few data
print("x1 information")
print(x1.ndim)    #ndim (number of dimension)
print(x1.shape)   #shape (size of each dimension)
print(x1.size)    #size (the total size of the array)
print(x1)         #content of x1

#initialize a numpy two-dimension array with 3x4 elements with random number f
rom 0 to 9
x2=np.random.randint(10,size=(3,4))
#initialize a numpy three-dimension array with 3x4x5 elements with random numb
er from 0 to 9
x3=np.random.randint(10,size=(3,4,5))

#each numpy array come with a few data
print("x2 information")
print(x2.ndim)    #ndim (number of dimension)
print(x2.shape)   #shape (size of each dimension)
print(x2.size)    #size (the total size of the array)
print(x2)         #content of x1

print("x3 information")
print(x3.ndim)    #ndim (number of dimension)
print(x3.shape)   #shape (size of each dimension)
print(x3.size)    #size (the total size of the array)
print(x3)         #content of x1
```

```
x1 information
1
(6,)
6
[5 0 3 3 7 9]
x2 information
2
(3, 4)
12
[[3 5 2 4]
 [7 6 8 8]
 [1 6 7 7]]
x3 information
3
(3, 4, 5)
60
[[[8 1 5 9 8]
  [9 4 3 0 3]
  [5 0 2 3 8]
  [1 3 3 3 7]]

 [[0 1 9 9 0]
  [4 7 3 2 7]
  [2 0 0 4 5]
  [5 6 8 4 1]]

 [[4 9 8 1 1]
  [7 9 9 3 6]
  [7 2 0 3 5]
  [9 4 4 6 4]]]
```

In [37]:
```
# to access single elements
print(x1)
print(x1[4]) #access the 4th elements of x1
print(x1[1]) #access the 1st element of x1
print(x1[-1]) #to access from the end of array
print(x1[-2]) #to access the seoncd last element of the x1
```

```
[5 0 3 3 7 9]
7
0
9
7
```

In [46]:
```
#to access single elements in multi-dimension array using comma-seperated tupl
e of indices

print(x2)
print(x2[0,0]) #top left corner element
print(x2[2,0]) # access row 2 column 0 element
print(x2[1,-3]) # row 1 column three from end behind
```

```
[[3 5 2 4]
 [7 6 8 8]
 [1 6 7 7]]
3
1
6
```

Exercise 6

Below would be code to initize a three-dimensional array. Please write code to access (print) all those elements with value of 3.

In [61]:
```
#numpy array initialization -- do not change the code here
np.random.seed(78)  #seed for reproducibility
x4=np.random.randint(10,size=(3,4,5))
print(x4)


#your code start here
print(x4[0,1,-1])
print(x4[0,2,0])
print(x4[1,1,0])
print(x4[2,3,2])
```

```
[[[5 6 7 8 9]
  [6 8 4 4 3]
  [3 8 4 4 6]
  [6 0 7 3 6]]

 [[4 9 7 2 7]
  [3 5 5 1 2]
  [4 9 0 0 6]
  [8 4 1 8 1]]

 [[0 9 6 6 6]
  [6 8 1 6 0]
  [9 6 4 4 0]
  [8 0 3 2 1]]]
3
3
3
3
```

2.2 Slicing of Numpy to access Subarrays

Very often in Data Analytics, we need to slice our data input different subarrays for processing and analysis. Numpy slicing syntax follows that of the standard Python list, to access a slice of an array x,

x[start:steop:step]

In [77]:
```
np.random.seed(0)  #seed for reproducibility

#initialize a numpy one-dimension array with number from 0 to 9
x5=np.arange(10)
print(x5)
print(x5[:5])         # get first 5 element
print(x5[5:])         # get subarray from element 5 to end of array
print(x5[2:6])        # get subarray from element 2 to element 6 (not inclusiv
e)
print(x5[::2])        #alternate element
print(x5[1::2])       #alternate element starting from index 1
print(x5[::-1])       #all element but reverse
print(x5[::-2])       #alternate all but reverse order
print(x5[0:9:3])      #start from element 1 to 8 (not  inclusive) with step of
 3
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4]
[5 6 7 8 9]
[2 3 4 5]
[0 2 4 6 8]
[1 3 5 7 9]
[9 8 7 6 5 4 3 2 1 0]
[9 7 5 3 1]
[0 3 6]
```

In [79]:
```
#Lets work on multi-dimension slicing
#initialize x6 for use
np.random.seed(0)  #seed for reproducibility

x6=np.random.randint(10,size=(3,4))
print(x6)         #content of x6
```

```
[[5 0 3 3]
 [7 9 3 5]
 [2 4 7 6]]
```

In [102]:
```
# multidimensional slices work in the same way, with multiple slices seperated
 by commas. Example

print(x6[:2,:3]) # first two rows and three columns
print(x6[:,1])   #slice column index 1
print(x6[:3,::2]) #all rows, every other column
print(x6[::-1,::-1]) #reverse all in once
```

```
[[5 0 3]
 [7 9 3]]
[0 9 4]
[[5 3]
 [7 3]
 [2 7]]
[[6 7 4 2]
 [5 3 9 7]
 [3 3 0 5]]
```

Note that Numpy array slices return views rather than copies of the array data as in standard Python list operation. This default behavior is usefufl as it means that when we deal with hug datsets, we can access and process these datasets without the need to keep creating copy in the data buffer. Lets illustrate this behavior with a simple example.

In [88]:
```
print(x2) #recall the two-dimensional array x2
subx2=x2[:2,:2] #slice a 2x2 subarray from x2
print(subx2)
subx2[0,0]=99 #modify the element 0,0 to 99
print(subx2)
print(x2)   #the correspending element in x2 is also modified
```

```
[[3 5 2 4]
 [7 6 8 8]
 [1 6 7 7]]
[[3 5]
 [7 6]]
[[99  5]
 [ 7  6]]
[[99  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
```

In [92]:
```
#if we want to create a copy instead of a view, we could use the copy method
subx2copy=x2[:2,:2].copy()
print(subx2copy)
subx2copy[0,0]=-1
print(subx2copy)  #the subx2copy element 0,0 is modified to -1
print(x2)         #the  corresponding element in x2 is not modified
```

```
[[99  5]
 [ 7  6]]
[[-1  5]
 [ 7  6]]
[[99  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
```

Exercise 7

Slice the multidimensional array create below into:

1. A vector make up of all column 1
2. All of row 3
3. A subarry consist of first array with row 1 to 3 and column 0 to 2

In [122]:
```
#code to initialize the array, do not modify
np.random.seed(78)  #seed for reproducibility
xtest=np.random.randint(10,size=(4,6,5))

#your code start here

print(xtest)
#print(xtest[:,:,1:2]) #!
#print(xtest[:,3:4,:]) #2
print(xtest[0,1:4,0:3])
```

```
[[[5 6 7 8 9]
  [6 8 4 4 3]
  [3 8 4 4 6]
  [6 0 7 3 6]
  [4 9 7 2 7]
  [3 5 5 1 2]]

 [[4 9 0 0 6]
  [8 4 1 8 1]
  [0 9 6 6 6]
  [6 8 1 6 0]
  [9 6 4 4 0]
  [8 0 3 2 1]]

 [[6 8 3 6 7]
  [3 6 3 1 5]
  [6 8 3 1 4]
  [4 7 7 2 6]
  [1 0 5 5 7]
  [9 1 5 8 8]]

 [[6 6 7 0 4]
  [7 6 2 9 4]
  [5 3 0 5 6]
  [7 2 9 9 5]
  [5 0 8 4 0]
  [6 3 2 4 6]]]
[[6 8 4]
 [3 8 4]
 [6 0 7]]
```

Another operation useful in data analytic is the reshaping operation. The most flexible way of doing this is with the reshape() method. If you want to put the number 1 through 9 in a 3x3 grid, it can be done in

In [124]:
```
#arange is to create a one dimensiona array with number from 1 to 10
grid=np.arange(1,10).reshape((3,3)) #reshape method is used to convert the one
 dimensional array to a 3x3 matrix
print(grid)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Numpy array can be concatente and split. Let try a few examples.

In [148]:
```
x=np.array([1,2,3])
y=np.array([4,5,6])
z=np.array([7,8,9])
xy=np.concatenate([x,y])
print(xy)
xyz=np.concatenate([x,y,z])
print(xyz)
x2=np.array([x,y,z])
print(x2)
xyz2=np.concatenate([x2,x2],axis=0)
print(xyz2)
xyz3=np.concatenate([x2,x2],axis=1)
print(xyz3)
```

```
[1 2 3 4 5 6]
[1 2 3 4 5 6 7 8 9]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[1 2 3]
 [4 5 6]
 [7 8 9]
 [1 2 3]
 [4 5 6]
 [7 8 9]]
[[1 2 3 1 2 3]
 [4 5 6 4 5 6]
 [7 8 9 7 8 9]]
```

3 Data Manipulation with Pandas

Python Data Analysis Library pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a NumFOCUS sponsored project. This will help ensure the success of development of pandas as a world-class open-source project, and makes it possible to donate to the project.

https://pandas.pydata.org/ (https://pandas.pydata.org/)

Again to use Panda, we need to install either by pip or conda depend on your environment. To use Panda, "import pandas as pd" is essential.

In [9]:
```
import  numpy as np
import pandas as pd
```

3.1 Pandas series

Pandas series wrap both a sequenceof values and a sequence of indices, which can be access with the value and index attributes. The values are simply a Numpy array as shown below:

In [8]:
```
data=pd.Series([0.25,0.5,0.75,1.0])
print(data)
print(data.values)  #access the values of a pandas series
print(data.index)   #acess the index of a pandas series
print(data[2])      #access individual value
print(data[1:3])    #access subset of a series
```

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
[0.25 0.5  0.75 1.  ]
RangeIndex(start=0, stop=4, step=1)
0.75
1    0.50
2    0.75
dtype: float64
```

In [7]:
```
# Constructing Series objects

p1=pd.Series([2, 4, 6])  #created from array by default the index is integer
print(p1)
p2=pd.Series(5, index=[100, 200, 300])   #value is scalar which repeat itself
 while indexes are changed
print(p2)
p3=pd.Series({2:'a', 1:'b', 3:'c'})  #pandas series created from dictionary
print(p3)
p4=pd.Series({2:'a', 1:'b', 3:'c'}, index=[3, 2])  #even if created from dicti
onary we could still specify the index
print(p4)               #Notice that in this case, the Series is populated onl
y with the explicitly identified keys.
```

```
0    2
1    4
2    6
dtype: int64
100    5
200    5
300    5
dtype: int64
1    b
2    a
3    c
dtype: object
3    c
2    a
dtype: object
```

3.2 Pandas DataFrame Object

The next fundamental structure in Pandas is the DataFrame. Like the Series object discussed in the previous section, the DataFrame can be thought of either as a generalization of a NumPy array, or as a specialization of a Python dictionary. We'll now take a look at each of these perspectives.

DataFrame as a generalized NumPy array

If a Series is an analog of a one-dimensional array with flexible indices, a DataFrame is an analog of a two-dimensional array with both flexible row indices and flexible column names. Just as you might think of a two-dimensional array as an ordered sequence of aligned one-dimensional columns, you can think of a DataFrame as a sequence of aligned Series objects. Here, by "aligned" we mean that they share the same index.

In [15]:
```
#some fictional data for illustration purposes
area_dict = {'Bedok': 423967, 'Tampines': 695662, 'Katong': 141297,
             'Sengkang': 170312, 'Punggol': 149995}
pop_dict = {'Bedok': 12345, 'Tampines': 678542, 'Katong': 435678,
             'Sengkang': 34568, 'Punggol': 345796}
area = pd.Series(area_dict)
pop = pd.Series(pop_dict)
print(area)

town = pd.DataFrame({'population': pop,
                                'area': area})
print(town)
print(town.index)
```

```
Bedok       423967
Katong      141297
Punggol     149995
Sengkang    170312
Tampines    695662
dtype: int64
            area  population
Bedok     423967       12345
Katong    141297      435678
Punggol   149995      345796
Sengkang  170312       34568
Tampines  695662      678542
Index(['Bedok', 'Katong', 'Punggol', 'Sengkang', 'Tampines'], dtype='object')
```

In [28]:
```
#lets read in some real data from the town_area_pop.csv

df=pd.read_csv('town_area_pop.csv')
print("First five data for previw")
print(df.head())
print("The columns names")
print(df.columns)
print("some information on the dataframe")
print(df.info(verbose=True))
```

```
First five data for previw
          Town  area Population
0   Ang Mo Kio  6.38    149,800
1        Bedok  9.37    204,300
2       Bishan  6.90     65,700
3  Bukit Batok  7.85    113,800
4  Bukit Merah  8.58    147,000
The columns names
Index(['Town', 'area', 'Population'], dtype='object')
some information on the dataframe
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23 entries, 0 to 22
Data columns (total 3 columns):
Town          23 non-null object
area          23 non-null float64
Population    23 non-null object
dtypes: float64(1), object(2)
memory usage: 632.0+ bytes
None
```

In [29]:
```
#data selection

print(df["area"])  #access only the area
```

```
0      6.38
1      9.37
2      6.90
3      7.85
4      8.58
5      4.89
6      5.83
7      4.12
8      6.78
9     13.09
10     3.84
11     9.87
12     7.99
13     6.01
14     9.57
15     6.94
16     7.08
17    10.55
18     7.37
19    12.00
20     5.56
21    11.98
22     7.78
Name: area, dtype: float64
```

As we can see that currently the index is just number and it would be quite inconvenience to use in this case. We may want to replace the index by Town name.

In [31]:
```
df.index=df['Town']
print(df)
```

```
                            Town   area Population
Town
Ang Mo Kio            Ang Mo Kio   6.38    149,800
Bedok                      Bedok   9.37    204,300
Bishan                    Bishan   6.90     65,700
Bukit Batok          Bukit Batok   7.85    113,800
Bukit Merah          Bukit Merah   8.58    147,000
Bukit Panjang      Bukit Panjang   4.89    119,300
Choa Chu Kang      Choa Chu Kang   5.83    161,100
Clementi                Clementi   4.12     72,500
Geylang                  Geylang   6.78     91,900
Hougang                  Hougang  13.09    179,800
Jurong East          Jurong East   3.84     80,300
Jurong West          Jurong West   9.87    260,000
Kallang/Whampoa  Kallang/Whampoa   7.99    105,500
Pasir Ris              Pasir Ris   6.01    111,000
Punggol                  Punggol   9.57     99,500
Queenstown            Queenstown   6.94     82,100
Sembawang              Sembawang   7.08     71,600
Sengkang                Sengkang  10.55    186,500
Serangoon              Serangoon   7.37     73,000
Tampines                Tampines  12.00    239,100
Toa Payoh              Toa Payoh   5.56    107,500
Woodlands              Woodlands  11.98    243,100
Yishun                    Yishun   7.78    186,600
```

In [ ]:
```
A better way is to set the index column to 'Town' when reading in the csv as s
hown below.
```

In [33]:
```
df2=pd.read_csv('town_area_pop.csv',index_col='Town')
print(df2)
```

```
                  area Population
Town
Ang Mo Kio        6.38    149,800
Bedok             9.37    204,300
Bishan            6.90     65,700
Bukit Batok       7.85    113,800
Bukit Merah       8.58    147,000
Bukit Panjang     4.89    119,300
Choa Chu Kang     5.83    161,100
Clementi          4.12     72,500
Geylang           6.78     91,900
Hougang          13.09    179,800
Jurong East       3.84     80,300
Jurong West       9.87    260,000
Kallang/Whampoa   7.99    105,500
Pasir Ris         6.01    111,000
Punggol           9.57     99,500
Queenstown        6.94     82,100
Sembawang         7.08     71,600
Sengkang         10.55    186,500
Serangoon         7.37     73,000
Tampines         12.00    239,100
Toa Payoh         5.56    107,500
Woodlands        11.98    243,100
Yishun            7.78    186,600
```

Exercise 8

Read DAILYDATA_S24_201801.csv file into a Pandas dataframe. Read in using the 'Day' as index

In [50]:
```
#your code start here

import pandas as pd
df=pd.read_csv('DAILYDATA_S24_201801.csv',index_col='Day')
print(df.head())
```

```
    Station  Year  Month  DailyRainfallTotal  Highest30MinRainfall  \
Day
1    Changi  2018      1                29.4                   6.0
2    Changi  2018      1                 1.0                   0.4
3    Changi  2018      1                 2.8                   1.8
4    Changi  2018      1                 0.4                   0.2
5    Changi  2018      1                 1.0                   1.0

     Highest60MinRainfall  Highest120MinRainfall  MeanTemperature  \
Day
1                    11.6                   13.4             24.8
2                     0.4                    0.4             25.5
3                     2.0                    2.0             26.5
4                     0.2                    0.2             26.1
5                     1.0                    1.0             26.0

     MaximumTemperature  MinimumTemperature  MeanWindSpeed  MaxWindSpeed
Day
1                  26.7                23.6            5.4          36.0
2                  27.3                24.2            7.9          24.5
3                  31.1                24.1            7.2          29.2
4                  28.2                25.0            8.3          23.8
5                  29.0                24.8            4.3          18.7
```

As we can see that column station and Year seem to be all the same value (you could try verifying them by display more data to convince yourself), we may want to remove these two columns so as not to waste memory space. Lets try to remove the two columns

In [51]:
```
df=df.drop(columns=['Station','Year'])
print(df.head())
```

```
     Month  DailyRainfallTotal  Highest30MinRainfall  Highest60MinRainfall  \
Day
1        1                29.4                   6.0                  11.6
2        1                 1.0                   0.4                   0.4
3        1                 2.8                   1.8                   2.0
4        1                 0.4                   0.2                   0.2
5        1                 1.0                   1.0                   1.0

     Highest120MinRainfall  MeanTemperature  MaximumTemperature  \
Day
1                     13.4             24.8                26.7
2                      0.4             25.5                27.3
3                      2.0             26.5                31.1
4                      0.2             26.1                28.2
5                      1.0             26.0                29.0

     MinimumTemperature  MeanWindSpeed  MaxWindSpeed
Day
1                  23.6            5.4          36.0
2                  24.2            7.9          24.5
3                  24.1            7.2          29.2
4                  25.0            8.3          23.8
5                  24.8            4.3          18.7
```