```
Jupyter Note Book
- Open Source Project
- Run Python, R, spark etc. codes
- type "jupyter notebook" command in anaconda prompt to get the jupyter kernel opened
```

```
Four Modes
1) Code - where we code
2) MarkDown - for markup language
    single hash-'#'- main heading
    double hash- '##'- side heading
    hyphen - '-' - bullet points
3) Heading - similar to above
4) Raw NBConvert - The present cell is Raw NBConvert which is a raw text that doesn't get executed along with codes
```

```
Jupyter keyboard commands
Press Esc for command mode
Press Enter for Edit mode
Some command mode shortcuts
- Press a to insert above
- Press b to insert below
- Press d for delete
- Press l for numbered lines
- Press s to save or create a check point
- Press h to know all the shortcuts both in cmd and edit modes
- Press r for raw mode
- Press c for code mode
- Press m for markdown
Some Edit mode shortcuts
- ctrl+u undo
- ctrl+enter run selected cells
- shift+enter run present cell

Important thing to remember is
 "PRESS h to all the shortcuts"
```

```
Bool : True/False
int : signed integer types
float: decimals
complex:3+j4
String: "I want to be a Data Scientist"
time: Data/time type
```

In [1]:
```
a=10
type(a)
```

Out[1]:
```
int
```

In [2]:
```
b=True
type(b)
```

Out[2]:
```
bool
```

In [3]:
```
c=1.5
type(c)
```

Out[3]:
```
float
```

In [4]:
```
d="Sudheer"
e="S"
print(type(d))
type(e)
```

```
<class 'str'>
```

Out[4]:
```
str
```

In [5]:
```
import datetime
```

In [7]:
```
datetime.date.today()
```

Out[7]:
```
datetime.date(2019, 6, 19)
```

In [12]:
```
datetime.MAXYEAR
```

Out[12]:
```
9999
```

```
Use tab key on keyboard when you stuck up what to use in module
Ex: (type)datetime.(now press tab it shows a dropdown of the utilities present)
```

In [13]:
```
datetime.datetime.now()
#To display time also (year,month,date,hr,min,sec,ms)
```

Out[13]:
```
datetime.datetime(2019, 6, 19, 0, 46, 14, 857449)
```

In [14]:
```
from datetime import datetime
datetime.now()
```

Out[14]:
```
datetime.datetime(2019, 6, 19, 0, 47, 45, 119789)
```

In [16]:
```
#assign or create a date object
from datetime import date
my_bdy=date(1999,1,31)
```

In [17]:
```
date.today()-my_bdy
```

Out[17]:
```
datetime.timedelta(days=7444)
```

In [23]:
```
import datetime # this is req as we imported datetime sub module before else error will be displayed
date.today()+datetime.timedelta(days=20)# datetime.timedelta to convert integer 20 to 20 days
```

Out[23]:
```
datetime.date(2019, 7, 9)
```

```
Math Operators
- usual notations only
- // integer division, / float division
```

In [25]:
```
print(10+4,10-4,10/4,10%4,10//4)
```

```
14 6 2.5 2 2
```

In [27]:
```
print(int('9'),str(9),type(int('9')),type(str(9)))#type casting
```

```
9 9 <class 'int'> <class 'str'>
```

Logical operators

asusual

- <,>,>=,<=,==,!=

```
If you use '>' first in the previous markdown cell it won't display it because it is the symbol for intendation in mardown mode
```

Comparision Operators

and, or, not

Conditional Statements

if , elif etc.

- Indentation is required

In [31]:
```
x=5
if x>0:
    print('positive')
elif x==0:
    print('zero')
else:
    print('negative')
```

```
positive
```

```
lists
Tuple
String
Dictionaries
Sets
Functions and Loops
These are the things to be learnt in basic python
```

```
Python lists are very flexible and can hold heterogenous data and they can be manipulated efficiently
```

In [1]:
```
mylist=[1,1.5,"Sudheer"]
print(mylist)
print(mylist[0])
print(mylist[-1])
```

```
[1, 1.5, 'Sudheer']
1
Sudheer
```

In [2]:
```
mylist.extend(['is','a'])
```

In [3]:
```
mylist.append("good boy")
```

In [4]:
```
print(mylist)
```

```
[1, 1.5, 'Sudheer', 'is', 'a', 'good boy']
```

In [5]:
```
mylist.count("Sudheer")
```
Out[5]: 1

In [6]:
```
len(mylist)
```
Out[6]: 6

In [7]:
```
mylist.append(1.5)
mylist.remove(1.5)
#removes first instance of element
print(mylist)
```
```
[1, 'Sudheer', 'is', 'a', 'good boy', 1.5]
```

In [8]:
```
mylist.pop()
```
Out[8]: 1.5

In [9]:
```
print(mylist)
```
```
[1, 'Sudheer', 'is', 'a', 'good boy']
```

In [11]:
```
mylist[3]='the'
```

In [12]:
```
print(mylist)
```
```
[1, 'Sudheer', 'is', 'the', 'good boy']
```

```
Tuple is a sequence of python objects, it is "IMMUTABLE"
```

In [13]:
```
mytuple=(1,2,3.5,"Sudheer")
```

In [14]:
```
mytuple[1]
```
Out[14]: 2

In [15]:
```
len(mytuple)
```
Out[15]: 4

In [16]:
```
mytuple2=tuple([5,4,3])
```

In [17]:
```
mytuple2
```
Out[17]: (5, 4, 3)

```
mytuple[0]=2 is not possible as tuples are immutable and we can't assign values to them
```

Strings

In [19]:
```
mystring="I will become a good data scientist"
```

In [20]:
```
mystring
```
Out[20]: 'I will become a good data scientist'

In [22]:
```
mystring[5]
```
Out[22]: 'l'

In [23]:
```
mystring[6:]
```
Out[23]: ' become a good data scientist'

In [24]:
```
mystring[:6]
```
Out[24]: 'I will'

In [25]:
```
mystring[6:10]
```
Out[25]: ' bec'

In [26]:
```
mystring[-1]
```
Out[26]: 't'

In [34]:
```
mystring[0:-9]
```
Out[34]: 'I will become a good data '

In [35]:
```
mystring[-9:]
```
Out[35]: 'scientist'

Dictionaries

In [36]:
```
mydict={'name':"Sudheer",'age':'20','profession':'data scientist'}
```

In [37]:
```
mydict
```

Out[37]:
```
{'name': 'Sudheer', 'age': '20', 'profession': 'data scientist'}
```

In [39]:
```
mydict2=dict(name='Sudheer',age=20,profession='Data Scientist')
mydict2
```

Out[39]:
```
{'name': 'Sudheer', 'age': 20, 'profession': 'Data Scientist'}
```

In [41]:
```
mydict2['name']
```

Out[41]:
```
'Sudheer'
```

Sets

- Collection of Unique elements

In [42]:
```
myset={'Sudheer','Sudheer','loves',"loves"}
```

In [43]:
```
myset
```

Out[43]:
```
{'Sudheer', 'loves'}
```

In [44]:
```
languages={'python','R','java','C'}
snakes={'cobra','python','viper'}
print(languages.union(snakes))
print(languages.intersection(snakes))
```

```
{'C', 'java', 'viper', 'python', 'R', 'cobra'}
{'python'}
```

Functions

In [46]:
```
def myfunc(x):
    if(x%2==0):
        return 'even'
    else:
        return 'odd'
print(myfunc(5))
print(myfunc(4))
```

```
odd
even
```

Loops

In [47]:
```
fruits={'banana','apple','mango'}
for fruit in fruits:
    print(fruit)
```

```
mango
banana
apple
```

Numpy

Numerical Python

- Created by Traive Oliphant in 2005

In [48]:
```
import numpy as np
```

In [49]:
```
a=np.array([1,2,3,5,7])
```

In [50]:
```
a
```

Out[50]:
```
array([1, 2, 3, 5, 7])
```

In [51]:
```
a=np.array([1,2,3,4,5.0])
a
```

Out[51]:
```
array([1., 2., 3., 4., 5.])
```

```
i.e. if one element is array is float then others will also converted to float i.e. auto typecasting
```

In [55]:
```
b=np.arange(10) # array of 0 to 9 integers
b
```

Out[55]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [59]:
```
c=np.array([[1,1],[2,2]])
```

```
print(b,b.ndim)
```

```
[[1 1]
 [2 2]] 2
```

In [60]:
```
d=np.arange(20).reshape(5,4)
```

In [61]:
```
d
```

Out[61]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19]])
```

In [62]:
```
d.shape
```

Out[62]:
```
(5, 4)
```

In [63]:
```
d.size
```

Out[63]:
```
20
```

In [64]:
```
type(d)
```

Out[64]:
```
numpy.ndarray
```

In [65]:
```
type(a)
```

Out[65]:
```
numpy.ndarray
```

In [67]:
```
e=np.zeros((3,4))
e
```

Out[67]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [70]:
```
f=np.arange(9).reshape(3,3)
g=np.arange(12).reshape(4,3)
np.vstack((f,g))#no of columns must be same to get stacked
```

Out[70]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [72]:
```
np.hstack((f,np.transpose(g)))# no of rows must be same
```

Out[72]:
```
array([[ 0,  1,  2,  0,  3,  6,  9],
       [ 3,  4,  5,  1,  4,  7, 10],
       [ 6,  7,  8,  2,  5,  8, 11]])
```

In [73]:
```
np.random.random()
```

Out[73]:
```
0.2103857233987786
```

In [74]:
```
#random array
h=np.random.random((5,5))
```

In [75]:
```
h
```

Out[75]:
```
array([[0.074411  , 0.4158652 , 0.63962888, 0.69185481, 0.50881403],
       [0.11816211, 0.21952564, 0.47736086, 0.62248643, 0.71754538],
       [0.69847928, 0.0497263 , 0.4100113 , 0.07105497, 0.65003337],
       [0.98835775, 0.40871494, 0.44277338, 0.23047386, 0.76604698],
       [0.90269858, 0.82863819, 0.86348482, 0.56141328, 0.80703776]])
```

In [76]:
```
i=np.floor(100*h)
i
```

Out[76]:
```
array([[ 7., 41., 63., 69., 50.],
       [11., 21., 47., 62., 71.],
       [69.,  4., 41.,  7., 65.],
       [98., 40., 44., 23., 76.],
       [90., 82., 86., 56., 80.]])
```

In [77]:
```
j=np.ceil(100*h)
```

In [78]:
```
j
```

Out[78]:
```
array([[ 8., 42., 64., 70., 51.],
       [12., 22., 48., 63., 72.],
       [70.,  5., 42.,  8., 66.],
       [99., 41., 45., 24., 77.],
       [91., 83., 87., 57., 81.]])
```

In [86]:
```
#convert list to array
k=np.array([1,2,3,5,4])
```

In [87]:
```
l=k # creates a reference of k i.e. they both refer to same numpy array
m=k.copy() #creates a copy, to know the difference see the following 2 statemnts and run this cell
k[0]=5
print(k,l,m)
```

```
[5 2 3 5 4] [5 2 3 5 4] [1 2 3 5 4]
```

Pandas

- Used for data mugging/massagning/wrangling

In [88]:
```
import pandas as pd
from pandas import Series,DataFrame
```

```
    Series is similar to a time series
```

In [89]:
```
series=Series([1,2,3,4,5,6,54,3])
series
```

Out[89]:
```
0     1
1     2
2     3
3     4
4     5
5     6
6    54
7     3
dtype: int64
```

```
we got a series as we usually see in maths
```

In [90]:
```
series2=Series([1,2,3,4],index=['row1','row2','row3','row4'])
series2
```

Out[90]:
```
row1    1
row2    2
row3    3
row4    4
dtype: int64
```

Selecting and Retreiving data from Series Object

In [91]:
```
series2[2]
```

Out[91]:
```
3
```

In [93]:
```
series2['row2']
```

Out[93]:
```
2
```

In [94]:
```
series[2:]
```

Out[94]:
```
2     3
3     4
4     5
5     6
6    54
7     3
dtype: int64
```

Data Frame

- multiple series objects together

In [106]:
```
n=np.floor(np.random.random(36).reshape(6,6)*100)
```

In [107]:
```
df=DataFrame(np.floor(np.random.random(36).reshape(6,6)*100))
```

In [108]:
```
df
```

Out[108]:

	0	1	2	3	4	5
0	96.0	93.0	86.0	85.0	20.0	13.0
1	79.0	21.0	80.0	4.0	75.0	4.0
2	52.0	23.0	98.0	1.0	45.0	30.0
3	40.0	23.0	74.0	34.0	94.0	85.0
4	99.0	43.0	97.0	1.0	56.0	3.0
5	56.0	71.0	77.0	80.0	90.0	78.0

In [109]:
```
df2=DataFrame(n)
```

In [110]:
```
df2
```

Out[110]:

	0	1	2	3	4	5
0	36.0	46.0	85.0	47.0	28.0	24.0
1	98.0	21.0	30.0	76.0	85.0	70.0
2	27.0	79.0	19.0	8.0	20.0	25.0
3	39.0	19.0	29.0	6.0	27.0	87.0
4	30.0	87.0	6.0	4.0	28.0	84.0
5	90.0	82.0	42.0	24.0	94.0	53.0

In [112]:
```
r='row'
c='column'
row=[]
column=[]
for i in range(len(df.index)):
    row.append(r+" "+str(i))
    column.append(c+" "+str(i))
df3=DataFrame(n,index=row,columns=column)
```

In [113]:
```
df3
```

Out[113]:

	column 0	column 1	column 2	column 3	column 4	column 5
row 0	36.0	46.0	85.0	47.0	28.0	24.0
row 1	98.0	21.0	30.0	76.0	85.0	70.0
row 2	27.0	79.0	19.0	8.0	20.0	25.0
row 3	39.0	19.0	29.0	6.0	27.0	87.0
row 4	30.0	87.0	6.0	4.0	28.0	84.0
row 5	90.0	82.0	42.0	24.0	94.0	53.0

In [115]:
```
df3.loc['row 0']['column 5']
```

Out[115]:
```
24.0
```

In [116]:
```
df3.loc['row 0']
```

Out[116]:
```
column 0    36.0
column 1    46.0
column 2    85.0
column 3    47.0
column 4    28.0
column 5    24.0
Name: row 0, dtype: float64
```

In [117]:
```
df3.loc['row 1','column 0']
```

Out[117]:
```
98.0
```

In [120]:
```
df3.loc[['row 1','row 2'],['column 0','column 1','column 5']]
```

Out[120]:

	column 0	column 1	column 5
row 1	98.0	21.0	70.0
row 2	27.0	79.0	25.0

In [121]:
```
df3.iloc[2,3]
```

Out[121]:
```
8.0
```

In [128]:
```
df3.loc[ : , 'column 3': ]
```

Out[128]:

	column 3	column 4	column 5
row 0	47.0	28.0	24.0
row 1	76.0	85.0	70.0
row 2	8.0	20.0	25.0
row 3	6.0	27.0	87.0
row 4	4.0	28.0	84.0
row 5	24.0	94.0	53.0

In [ ]:
```
```