#Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.
#It is the fundamental package for scientific computing with Python.
#It save coding time. Example: No for loop for many vector and matrix operations.
#Faster Execution: It avoide type checking because it have single datatype for each field.
#In this memory size of each item in bytes.
#Data can be accessed by indexing.
#NumPy is the foundation of python datastructure.
#It is low level datastructure. # np.array
#Support multidimensional arrays and matrices.
#It also support wide range of mathematical operations.

#Numpy
import numpy as np
ndarray - an N-dimensional array, which describes a collection of â€œitemsâ€ of the same type
array(list) # constructor
asarray(a[, dtype, order]) # Convert the input to an array
Constants:
ndarray.shape #tuple of array dimensions
ndarray.size #number of elements in array
ndarray.itemsize #size of one element
ndarray.dtype #data type of elements
ndarray.flat #1D iterator over elements of array
Common Functions
np.tolist() #Return the array as a (possibly nested) list.
np.reshape(a, (3,2)) #Returns an array containing the same data with a new shape.
np.swapaxes(axis1, axis2) #Return a view of the array with axis1 and axis2 interchanged.
np.copy() #Return a copy of the array.
np.arange() #Return evenly spaced values within a given interval.
Statistics Functions:
np.sum(a, axis) #Sum of array elements over a given axis.
np.prod #Return the product of array elements over a given axis.
np.min #Return the minimum along a given axis.
np.max #Return the maximum along a given axis.
np.mean #Compute the arithmetic mean along the specified axis.
np.std #Compute the standard deviation along the specified axis.
np.var #Compute the variance along the specified axis.
np.sort(axis) #Return a sorted copy of an array.
Other Functions:
String operations #A set of vectorized string operations for arrays.
logical operations # AND, OR, XOR, NOT, >, <, =, ...
Trigonometric functions #sin, cos, tan, etc.
complex numbers #(real + imaginary)
polynomials #Polynomials in NumPy can be created, manipulated, and even fitted.

```
==========================================================================================
import numpy as np   #import the entire numpy module with a short name as np
x = [0,1,2,3,4,5]
a = np.array(x)    #multidimensional container of items of the same type and size.
print(a)

Output: [0 1 2 3 4 5]

index: a[2]  # Output: 2
```

```
#indexing with N integers returns an array scalar representing the corresponding item.
slice:  a[start:stop:step]
       a[1:4:2]  #Output: array([1, 3])
       a[3:]     #Output: array([3, 4, 5])
       a[:3]     #Output: array([0, 1, 2])
a.shape          #Output: (6,)
a.size           #Output: 6
a.itemsize       #Output: 8  # 64/8 = 8, here 64 comes from dtype. And 8 bits = 1 byte.
a.dtype          #Output: dtype('int64')

b = np.array([[1,2,3], [4,5,6]])
print(b)

Output: array([[1, 2, 3],
               [4, 5, 6]])

b.swapaxes(0,1)
print(b)

Output: array([[1, 4],
               [2, 5],
               [3, 6]])

a = np.arange(0,6)  #Output: array([0, 1, 2, 3, 4, 5])
a = np.arange(0,6).reshape(2,3)

Output: array([[0, 1, 2],
               [3, 4, 5]])
================================================================================
===
import numpy as np

a = np.array([2,3,4])

a = np.arange(1, 12, 2)         # (from, to, step)

Output: array([ 1,  3,  5,  7,  9, 11])

a = np.linspace(1, 12, 6)       # (first, last, num_elements) float data type

Output: array([  1. ,   3.2,   5.4,   7.6,   9.8,  12. ])

a.reshape(3,2)
a = a.reshape(3,2)
Output: array([[  1. ,   3.2],
               [  5.4,   7.6],
               [  9.8,  12. ]])
a.size   # 6

a.shape  # (6,)

a.dtype  # dtype('float64')

a.itemsize  # 8
```

```
# this works:
b = np.array([(1.5,2,3), (4,5,6)])
print(b)

Output:[[ 1.5  2.   3. ]
       [ 4.   5.   6. ]]

# but this does not work:
b = np.array(1,2,3)           # square brackets are required
print(b)

Output: raceback (most recent call last):
  File "<ipython-input-32-f61788523772>", line 1, in <module>
    b = np.array(1,2,3)
ValueError: only 2 non-keyword arguments accepted
================================================================================
import numpy as np
a = np.arange(0,6).reshape(2,3)
print(a)
Output: array([[0, 1, 2],
               [3, 4, 5]])

print(a < 4)                      # prints True/False
Output: [[ True  True  True]
         [ True False False]]
a * 3                             # multiplies each element by 3
a *= 3                            # saves result to a
Output: array([[ 0,  3,  6],
               [ 9, 12, 15]])
a = np.zeros((3,4))
Output: array([[ 0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.],
               [ 0.,  0.,  0.,  0.]])
a = np.ones((2,3))
Output: array([[ 1.,  1.,  1.],
               [ 1.,  1.,  1.]])

a = np.random.random((2,3))
Output: array([[ 0.01597309,  0.57744167,  0.8222098 ],
               [ 0.50195045,  0.70745541,  0.73827131]])

np.set_printoptions(precision=2) # show 2 decimal places
Output: array([[ 0.02,  0.58,  0.82],
               [ 0.5 ,  0.71,  0.74]])

a = np.random.randint(0,10,5)
Output: array([6, 5, 7, 0, 0])
================================================================================
import numpy as np
a = np.arange(0,6).reshape(2,3)
print(a)
Output: array([[0, 1, 2],
               [3, 4, 5]])
```

```
a.sum()   # 15
a.min()   # 0
a.max()   # 5
a.mean()  # 2.5
a.var()   # 2.9166666666666665  # variance
a.std()   # 1.707825127659933   # standard deviation

a.sum(axis=1)  # array([ 3, 12])
a.sum(axis=0)  # array([3, 5, 7])

a.argmin()  # 0        # index of min element
a.argmax()  # 5        # index of max element
a.argsort()
Output: array([[0, 1, 2],
           [0, 1, 2]]) # returns array of indices that would put the array in sorted order

# indexing, slicing, iterating
a = np.arange(10)**2 #Output: array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
a[2:5]  #Output: array([ 4,  9, 16])

for i in a:
      print (i ** 2)
Output:
0
1
16
81
256
625
1296
2401
4096
6561
==========================================================================
import numpy as np
a = np.arange(10)**2 #Output: array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
a[::-1] # reverses array. Output: array([81, 64, 49, 36, 25, 16,  9,  4,  1,  0])

a = np.arange(0,6).reshape(2,3)
Output: array([[0, 1, 2],
           [3, 4, 5]])
a.transpose()
Output: array([[0, 3],
           [1, 4],
           [2, 5]])
a.ravel() # flattens to 1D # array([0, 1, 2, 3, 4, 5])

# read in csv data file
data = np.loadtxt("data.txt", dtype=np.uint8, delimiter=",", skiprows=1 )
# loadtxt does not handle missing values. to handle such exceptions use genfromtxt instead.
Output: [[9 3 8 7 6 1 0 4 2 5]
       [1 7 4 9 2 6 8 3 5 0]
       [4 8 3 9 5 7 2 6 0 1]
       [1 7 4 2 5 9 6 8 0 3]
```

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

data = np.loadtxt("data.txt", dtype=np.uint8, delimiter=",", skiprows=1, usecols=[0,1,2,3])
Output:array([[9, 3, 8, 7],
           [1, 7, 4, 9],
           [4, 8, 3, 9],
           [1, 7, 4, 2],
           [0, 7, 5, 2],
           [5, 9, 1, 4]], dtype=uint8)
np.random.shuffle(data)
print(data)
Output: array([[0, 7, 5, 2, 8, 6, 3, 4, 1, 9],
           [4, 8, 3, 9, 5, 7, 2, 6, 0, 1],
           [9, 3, 8, 7, 6, 1, 0, 4, 2, 5],
           [1, 7, 4, 2, 5, 9, 6, 8, 0, 3],
           [1, 7, 4, 9, 2, 6, 8, 3, 5, 0],
           [5, 9, 1, 4, 7, 0, 3, 6, 8, 2]], dtype=uint8)

a = np.random.random(5) #5 random numbers between 0 and 1.
Output: array([ 0.1424916 ,  0.34602233,  0.10294581,  0.37575511,  0.04312772])

np.random.choice(a) #Output: 0.10294580703740197 # a random choice.

np.random.randint(5,10,2) #Output: array([5, 6])
================================================================================
#High level data structure.  #dataframe
#More streamlined handeling to tabular data and rich time series functionality.
#It support data alignment, missing data friendly statistics, groupby, merge and join methods.
#In Pandas datastructure we can freely draw NumPy and SciPy functions to manipulate them.
#Pandas

import numpy as np   #import the entire numpy module with a short name as np
import pandas as pd  #import the entire pandas module with a short name as pd

def header(msg):
      print('-' * 50)  # print 50 times "-"
      print('[ ' + msg + ' ]') # print the msg which are passed through header function.

# 1. load hard-coded data into a dataframe
header("1. load hard-coded data into a df")
df = pd.DataFrame(
      [['Jan',58,42,74,22,2.95],
      ['Feb',61,45,78,26,3.02],
      ['Mar',65,48,84,25,2.34],
      ['Apr',67,50,92,28,1.02],
      ['May',71,53,98,35,0.48],
      ['Jun',75,56,107,41,0.11],
      ['Jul',77,58,105,44,0.0],
      ['Aug',77,59,102,43,0.03],
      ['Sep',77,57,103,40,0.17],
      ['Oct',73,54,96,34,0.81],
      ['Nov',64,48,84,30,1.7],
      ['Dec',58,42,73,21,2.56]],
```

```
        index = [0,1,2,3,4,5,6,7,8,9,10,11],
        columns = ['month','avg_high','avg_low','record_high','record_low',
'avg_precipitation'])
print(df)
```

Output:

```
-------------------------------------------------
[ 1. load hard-coded data into a df ]
   month  avg_high  avg_low  record_high  record_low  avg_precipitation
0    Jan        58       42           74          22               2.95
1    Feb        61       45           78          26               3.02
2    Mar        65       48           84          25               2.34
3    Apr        67       50           92          28               1.02
4    May        71       53           98          35               0.48
5    Jun        75       56          107          41               0.11
6    Jul        77       58          105          44               0.00
7    Aug        77       59          102          43               0.03
8    Sep        77       57          103          40               0.17
9    Oct        73       54           96          34               0.81
10   Nov        64       48           84          30               1.70
11   Dec        58       42           73          21               2.56
```

```
# 2. read text file into a dataframe
header("2. read text file into a df")
filename = 'Fremont_weather.txt'
df = pd.read_csv(filename)
print(df)
```

Output:

```
-------------------------------------------------
[ 2. read text file into a df ]
   month  avg_high  avg_low  record_high  record_low  avg_precipitation
0    Jan        58       42           74          22               2.95
1    Feb        61       45           78          26               3.02
2    Mar        65       48           84          25               2.34
3    Apr        67       50           92          28               1.02
4    May        71       53           98          35               0.48
5    Jun        75       56          107          41               0.11
6    Jul        77       58          105          44               0.00
7    Aug        77       59          102          43               0.03
8    Sep        77       57          103          40               0.17
9    Oct        73       54           96          34               0.81
10   Nov        64       48           84          30               1.70
11   Dec        58       42           73          21               2.56
```

```
# 3. print first 5 or last 3 rows of df
header("3. df.head()")
print(df.head())
header("3. df.tail(3)")
print(df.tail(3))
```

Output:

```
-------------------------------------------------
[ 3. df.head() ]
```

```
  month  avg_high  avg_low  record_high  record_low  avg_precipitation
0   Jan        58       42           74          22               2.95
1   Feb        61       45           78          26               3.02
2   Mar        65       48           84          25               2.34
3   Apr        67       50           92          28               1.02
4   May        71       53           98          35               0.48
--------------------------------------------------
[ 3. df.tail(3) ]
   month  avg_high  avg_low  record_high  record_low  avg_precipitation
9    Oct        73       54           96          34               0.81
10   Nov        64       48           84          30               1.70
11   Dec        58       42           73          21               2.56

# 4. get data types, index, columns, values
header("4. df.dtypes")
print(df.dtypes)
Output:
--------------------------------------------------
[ 4. df.dtypes ]
month               object
avg_high             int64
avg_low              int64
record_high          int64
record_low           int64
avg_precipitation  float64
dtype: object

header("4. df.index")
print(df.index)

Output:
--------------------------------------------------
[ 4. df.index ]
RangeIndex(start=0, stop=12, step=1)

header("4. df.columns")
print(df.columns)

Output:
[ 4. df.columns ]
Index(['month', 'avg_high', 'avg_low', 'record_high', 'record_low', 'avg_precipitation'],
      dtype='object')

header("4. df.values")
print(df.values)

Output:
--------------------------------------------------
[ 4. df.values ]
[['Jan' 58 42 74 22 2.95]
 ['Feb' 61 45 78 26 3.02]
 ['Mar' 65 48 84 25 2.34]
 ['Apr' 67 50 92 28 1.02]
 ['May' 71 53 98 35 0.48]
```

```
 ['Jun' 75 56 107 41 0.11]
 ['Jul' 77 58 105 44 0.0]
 ['Aug' 77 59 102 43 0.03]
 ['Sep' 77 57 103 40 0.17]
 ['Oct' 73 54 96 34 0.81]
 ['Nov' 64 48 84 30 1.7]
 ['Dec' 58 42 73 21 2.56]]

# 5. statistical summary of each column
header("5. df.describe()")
print(df.describe())

Output:
--------------------------------------------------
[ 5. df.describe() ]
       avg_high    avg_low  record_high  record_low  avg_precipitation
count  12.000000  12.000000    12.000000   12.000000          12.000000
mean   68.583333  51.000000    91.333333   32.416667           1.265833
std     7.366488   6.060303    12.323911    8.240238           1.186396
min    58.000000  42.000000    73.000000   21.000000           0.000000
25%    63.250000  47.250000    82.500000   25.750000           0.155000
50%    69.000000  51.500000    94.000000   32.000000           0.915000
75%    75.500000  56.250000   102.250000   40.250000           2.395000
max    77.000000  59.000000   107.000000   44.000000           3.020000

# 6. sort records by any column
header("6. df.sort_values('record_high', ascending=False)")
print (df.sort_values('record_high', ascending=False))

Output:
--------------------------------------------------
[ 6. df.sort_values('record_high', ascending=False) ]
   month  avg_high  avg_low  record_high  record_low  avg_precipitation
5    Jun        75       56          107          41               0.11
6    Jul        77       58          105          44               0.00
8    Sep        77       57          103          40               0.17
7    Aug        77       59          102          43               0.03
4    May        71       53           98          35               0.48
9    Oct        73       54           96          34               0.81
3    Apr        67       50           92          28               1.02
2    Mar        65       48           84          25               2.34
10   Nov        64       48           84          30               1.70
1    Feb        61       45           78          26               3.02
0    Jan        58       42           74          22               2.95
11   Dec        58       42           73          21               2.56

# 7. slicing records
header("7. slicing -- df.avg_low")
print(df.avg_low)                       # index with single column
Output:
--------------------------------------------------
[ 7. slicing -- df.avg_low ]
0    42
1    45
```

```
2     48
3     50
4     53
5     56
6     58
7     59
8     57
9     54
10    48
11    42
Name: avg_low, dtype: int64

header("7. slicing -- df['avg_low']")
print(df['avg_low'])
Output:
--------------------------------------------------
[ 7. slicing -- df.avg_low ]
0     42
1     45
2     48
3     50
4     53
5     56
6     58
7     59
8     57
9     54
10    48
11    42
Name: avg_low, dtype: int64

header("7. slicing -- df[2:4]")              # index with single column
print(df[2:4])                          # rows 2 to 3

Output:
--------------------------------------------------
[ 7. slicing -- df[2:4] ]
  month  avg_high  avg_low  record_high  record_low  avg_precipitation
2   Mar        65       48           84          25               2.34
3   Apr        67       50           92          28               1.02

header("7. slicing -- df[['avg_low','avg_high']]")
print(df[['avg_low','avg_high']])

Output:
--------------------------------------------------
[ 7. slicing -- df[['avg_low','avg_high']] ]
    avg_low  avg_high
0        42        58
1        45        61
2        48        65
3        50        67
4        53        71
5        56        75
```

```
6       58       77
7       59       77
8       57       77
9       54       73
10       48       64
11       42       58

header("7. slicing scalar value -- df.loc[9,['avg_precipitation']]")
print(df.loc[9,['avg_precipitation']])

Output:
--------------------------------------------------
[ 7. slicing scalar value -- df.loc[9,['avg_precipitation']] ]
avg_precipitation    0.81
Name: 9, dtype: object

header("7. df.iloc[3:5,[0,3]]")  # index location can receive range or list of indices
print(df.iloc[3:5,[0,3]])

Output:
--------------------------------------------------
[ 7. df.iloc[3:5,[0,3]] ]
  month  record_high
3   Apr          92
4   May          98

# 8. filtering
header("8. df[df.avg_precipitation > 1.0]")     # filter on column values
print(df[df.avg_precipitation > 1.0])

Output:
--------------------------------------------------
[ 8. df[df.avg_precipitation > 1.0] ]
   month  avg_high  avg_low  record_high  record_low  avg_precipitation
0    Jan        58       42           74          22               2.95
1    Feb        61       45           78          26               3.02
2    Mar        65       48           84          25               2.34
3    Apr        67       50           92          28               1.02
10   Nov        64       48           84          30               1.70
11   Dec        58       42           73          21               2.56

header("8. df[df['month'].isin['Jun','Jul','Aug']]")
print(df[df['month'].isin(['Jun','Jul','Aug'])])

Output:
--------------------------------------------------
[ 8. df[df['month'].isin['Jun','Jul','Aug']] ]
  month  avg_high  avg_low  record_high  record_low  avg_precipitation
5   Jun        75       56          107          41               0.11
6   Jul        77       58          105          44               0.00
7   Aug        77       59          102          43               0.03

# 9. assignment -- very similar to slicing
header("9. df.loc[9,['avg_precipitation']] = 101.3")
```

```
df.loc[9,['avg_precipitation']] = 101.3
print(df.iloc[9:11])

Output:
--------------------------------------------------
[ 9. df.loc[9,['avg_precipitation']] = 101.3 ]
   month  avg_high  avg_low  record_high  record_low  avg_precipitation
9    Oct        73       54           96          34              101.3
10   Nov        64       48           84          30                1.7

header("9. df.loc[9,['avg_precipitation']] = np.nan")
df.loc[9,['avg_precipitation']] = np.nan
print(df.iloc[9:11])

Output:
--------------------------------------------------
[ 9. df.loc[9,['avg_precipitation']] = np.nan ]
   month  avg_high  avg_low  record_high  record_low  avg_precipitation
9    Oct        73       54           96          34              NaN
10   Nov        64       48           84          30                1.7

header("9. df.loc[:,'avg_low'] = np.array([5])")
df.loc[:,'avg_low'] = np.array([5])
print(df.head())

Output:
--------------------------------------------------
[ 9. df.loc[:,'avg_low'] = np.array([5]) ]
  month  avg_high  avg_low  record_high  record_low  avg_precipitation
0   Jan        58        5           74          22               2.95
1   Feb        61        5           78          26               3.02
2   Mar        65        5           84          25               2.34
3   Apr        67        5           92          28               1.02
4   May        71        5           98          35               0.48

header("9. df['avg_day'] = (df.avg_low + df.avg_high) / 2")
df['avg_day'] = (df.avg_low + df.avg_high) / 2
print(df.head())

Output:
[ 9. df['avg_day'] = (df.avg_low + df.avg_high) / 2 ]
  month  avg_high  avg_low  record_high  record_low  avg_precipitation  \
0   Jan        58        5           74          22               2.95
1   Feb        61        5           78          26               3.02
2   Mar        65        5           84          25               2.34
3   Apr        67        5           92          28               1.02
4   May        71        5           98          35               0.48

   avg_day
0     31.5
1     33.0
2     35.0
3     36.0
4     38.0
```

```
# 10. renaming columns
header("10. df.rename(columns = {'avg_precipitation':'avg_rain'}, inplace=True)")
df.rename(columns = {'avg_precipitation':'avg_rain'}, inplace=True)     # rename 1 column
print(df.head())

Output:
-------------------------------------------------
[ 10. df.rename(columns = {'avg_precipitation':'avg_rain'}, inplace=True) ]
  month  avg_high  avg_low  record_high  record_low  avg_rain  avg_day
0   Jan        58        5           74          22      2.95     31.5
1   Feb        61        5           78          26      3.02     33.0
2   Mar        65        5           84          25      2.34     35.0
3   Apr        67        5           92          28      1.02     36.0
4   May        71        5           98          35      0.48     38.0

# 11. iterate a df
header("11. iterate rows of df with a for loop")
for index, row in df.iterrows():
    print (index, row["month"], row["avg_high"])

Output:
-------------------------------------------------
[ 11. iterate rows of df with a for loop ]
0 Jan 58
1 Feb 61
2 Mar 65
3 Apr 67
4 May 71
5 Jun 75
6 Jul 77
7 Aug 77
8 Sep 77
9 Oct 73
10 Nov 64
11 Dec 58

# 12. write to csv file
df.to_csv('updated_data.csv')

================================================================================
#Python's most popular 2D plotting library.
#Produce dozens of different types of plots and charts with just few lines of code.
#We can easly plot NumPy arrays, Pandas dataframe and Python List.
#The PyPlot module provides a MATLAB like interface.

#Matplotlib

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 1. simple plot with 4 numbers
plt.plot([1, 3, 2, 4])
```

```
plt.show()

# 2. points have x and y values; add title and axis labels
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.title('Test Plot', fontsize=8, color='g')
plt.xlabel('number n')
plt.ylabel('n^2')
plt.show()

# 3. change figure size. plot red dots; set axis scales x: 0-6 and y: 0-20
plt.figure(figsize=(1,5))       # 1 inch wide x 5 inches tall
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro') # red-o
plt.axis([0, 6, 0, 20])                     # [xmin, xmax, ymin, ymax]
plt.annotate('square it', (3,6))
plt.show()

# 4. bar chart with four bars
plt.clf()             # clear figure
x = np.arange(4)
y = [8.8, 5.2, 3.6, 5.9]
plt.xticks(x, ('Ankit', 'Hans', 'Joe', 'Flaco'))
plt.bar(x, y)
# plt.bar(x, y, color='y')
# plt.bar(x, y, color=['lime', 'r', 'k', 'tan'])
plt.show()

# 5. two sets of 10 random dots plotted
d = {'Red O' : np.random.rand(10),
    'Grn X' : np.random.rand(10)}
df = pd.DataFrame(d)
df.plot(style=['ro','gx'])
plt.show()

# 6. random dots in a scatter
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
sizes = (30 * np.random.rand(N))**2  # 0 to 15 point radii
plt.scatter(x, y, s=sizes, c=colors, alpha=0.5)
plt.show()

# 7. subplots
fig = plt.figure()
fig.suptitle('My SubPlots')
fig.add_subplot(221)   #top left
plt.plot([np.log(n) for n in range(1,10)])
fig.add_subplot(222, facecolor='y')   #top right
fig.add_subplot(223)   #bottom left
fig.add_subplot(224)   #bottom right
plt.show()

fig, plots = plt.subplots(2, sharex=True)
fig.suptitle('Sharing X axis')
```

```
x = range(0,200,5)
y = [n**0.8 for n in x]
plots[0].plot(x, y, color='r')
plots[1].scatter(x, y)

# 8. save figure to image file
plt.figure(figsize=(4,3), dpi=100)
plt.plot([245, 170, 148, 239, 161, 196, 112, 258])
plt.axis([0, 7, 0, 300])
plt.title('Flight Data')
plt.xlabel('Speed')
# plt.savefig('Flights.png')
plt.show()
```