53.) import numpy as np } to create numpy
a = np.array([0,1,2,3]) } array using python
print(a) | a
→ [0 1 2 3] | array([0 1 2 3]) list.

print(np.arange(10)) } to create numpy array
→ [0 1 2 3] from any integer
[0 1 2 3 4 5 6 7 8 9]

54.) To calculate time to run a program -
# python list
l = range(1000)
% timeit [i**2 for i in l] } normally.
→ 237 µs ± 1.36 µs per loop

a = np.arange(1000)
% time it a**2 } using numpy.
→ 1.03 µs ± 12.3 ns per loop

55.) Creating arrays in numpy -
# 1-D (array) (1-Dimensional) → list (it is called)
a = np.array([0,1,2,3])
a
→ array([0,1,2,3])

56.) 1-D array → list
2-D array → matrix
n-D array → Tensor

57.) # print dimension { find dimension }
a.ndim
→ 1

58.) # shape { find shape }
a.shape
→ (4,)

59.) len(a) { in tensor there is nothing like length its only shape }
4.

60.) #= 2-D, 3-D, ----
b = np.array([[0,1,2], [3,4,5]])
b
→ array([[0,1,2],
[3,4,5]])

61.) b.ndim	62.) b.shape	63.) len(b)
→ 2	→ (2,3)	→ 2
	1st dim. 2nd dim.	(only return the size of 1st dim.)

64.) in 3-D array
n.shape
→ (2,2,2)
n=([[1,2], [2,3], [3,4]])

65.) function for creating array-
a = np.arrange(2)
→ array([0,1])

DATE

66.) start, end (exclusive), step

```
b= np.arrange (1, 10, 2)
b
→ array([1, 3, 5, 7, 9])
```

67.) linespace →

start, end (inclusive), no. of points (divide in 6 parts)

```
a = np.linespace (0, 1, 6)
a
→ array ([0. , 0.2 , 0.4 , 0.6 , 0.8 , 1.0])
```

1st part, 2nd part, 3rd, 4th, 5th.

68.) create array of ones

```
a= np.ones ((2,3))
a
→ array ([[1. , 1. ,
           1. , 1. ]])
```

69.) create array of zeros

```
b= np.zeros ((2,2))
b
→ array ([[0., 0.],
          [0., 0.]])
```

70.) diagonal matrix -

```
c= np.eye (2)
c
→ array ([[1.e, 0.],
          [0., 1.]])
```

71.) more dim.

```
d = np.eye (3,2)
d
```

row, column

```
→ array ([[1.0, 0.],
          [0., 1.],
          [0., 0.]])
```

MON TUE WED THU FRI SAT SUN

DATE .....

72.) create array using diag f^n -

```
a = np.diag([1,2,3,4])
a
→ array([[1,0,0,0],
         [0,2,0,0],
         [0,0,3,0],
         [0,0,0,4]])
```

73.) np.diag(a) # Extract diagonal

→ array([1,2,3,4])

74.) # create array using random. {it will be uniform distribution over [0,1)} {in this prob. of each value to occur is same}

```
a = np.random.rand(3)
a
→ array([0.85434586, 0.05106692, 0.37337949])
```

75.) a = np.random.randn(4). (# std. normal distribution. in this the prob. of values near 0 is high)

```
a
→ array([1.99407539e+00,
         -1.33836224e+00,
          3.07395030e-04])
```

76.) Basic data type -

```
a = np.arange(10)
a.dtype
→ dtype('int64')
```

MON TUE WED THU FRI SAT SUN

DATE

77.) to specify which data type we want -

```
a = np.arange(10, dtype='float64')
a
```

bits/size.

→ array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])

78.) # The default data type is float for zeros & ones -

```
a = np.zeros((2,2))
print(a)
a.dtype
```

→ [[0., 0.
0., 0.]]
dtype('float64')

79.) Complex datatype.

```
d = np.array([1+2j, 2+4j])
print(d.dtype)
```

→ complex128

80.) # Boolean datatype.

```
b = np.array([True, False])
print(b.dtype)
```

→ bool

81.) # string datatype

```
s = np.array(['Ram', 'Rom'])
s.dtype
```

→ dtype('S6')

MON TUE WED THU FRI SAT SUN

DATE .....

82.) Each built-in data type has a character code that uniquely identifies it.

'b' - boolean
'i' - (signed) integer [range → -27149 to + 27149]
'u' - (unsigned) integer [range 0 to 4....]
'f' - floating pt.
'c' - complex-floating pt
'm' - time delta } in time format or date format.
'M' - date time
'O' - (Python) objects → if you don't know which datatype it is
'S', 'a' - (byte) string
'V' - raw data (void)
'U' - Unicode

83.) samplearray = np.array([1,2,3,4,5])
samplearray * 5
→ array([5, 10, 15, 20, 25])

84.) samplearray / 2
→ array([0.5, 1.0, 1.5, 2., 2.5])

85.) print(type(samplearray))
→ <class 'numpy.ndarray'>

86.) arr = np.array([[1,2], [3,4]])
arr
→ array([[1,2],
[3,4]])

MON TUE WED THU FRI SAT SUN

DATE

87.) arr.shape
→ (2,2)

88.) Reshape the array - (used to gain a particular shape).
~~arr = a.reshape~~ arr = arr.reshape(1,4).
arr.shape
→ (1,4).

89.) Random -
from numpy import random
{import only random} {import random from 1st file}.
{from numpy import *} {import everything}

90.) x = random.randint(100, size=(3,5))
x
→ array([[48, 27, 38, 57, 27],
[26, 14, 26, 46, 61],
[87, 80, 3, 20, 86]])

~~Indexing and slicing~~ -

91.) a = np.ones((2,2)) + 1
a
→ array([[2., 2.]
[2., 2.]])

92.) to get normal distribution of $\mu = 100$, $\sigma = 15$
⇒ import numpy as np
import matplotlib.pyplot as plt
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
→

Indexing and slicing-

Indexing

93) a = np.arrange(10)
print(a[5])
→ 5

print element at 5th position.
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
address 1 2 ------ 5

94) a = np.diag [(1, 2, 3])
print(a)
print(a[2,2])
→ a[[1 0 0]
1→ [0 2 0]
2→ [0 0 3]]
3 ? 1 2

95) Assign any a value-
a [2,1] = 5
a
→ array([[1,0,0]
[0,2,0]
[0,5,3]]).

96) slice out value from 1-D array.
a[1:8:2]
→ array([1, 3, 5, 7])

MON TUE WED THU FRI SAT SUN

DATE

97.) # we can also combine assignment & slicing:

```
a = np.arange(10)
a[5:] = 10
a
```

→ array([0, 1, 2, 3, 4, 10, 10, 10, 10, 10])

98.)

```
b = np.arange(5)
a[5:] = b[::-1]   # assigning
a
```

→ array([0, 1, 2, 3, 4, 4, 3, 2, 1, 0])

Copies & views-

99.)

```
a = np.arange(10)
a
```

→ array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

```
b = a[::2]   → view
b
```

→ array([0, 2, 4, 6, 8])

```
np.shares_memory(a,b)
```

→ True

{# are the two arrays share memory} i.e., ~~they have same values~~ }

```
b[0] = 10
b
```

→ array([10, 2, 4, 6, 8])

```
a
```

→ ([10, 1, 2, 3, 4, 5, 6, 7, 8, 9])

it change the value in a also it change in b.

MON TUE WED THU FRI SAT SUN

DATE

100.) To overcome problem of pt (99) a copy is created -

```
a = np.arange(5)
c = a[::2].copy()
→ array([0, 2, 4])

np.shares_memory(a,c)          # now it does not share memory as 'c' is a new copy of 'a'. there will be no rel" b/w a & c.
→ False

c[0] = 10
a
→ array([0,1,2,3,4])
```

Fancy Indexing - It by default creates copy.

101.) using Boolean mask -

```
a = np.random.randint(0, 20, 15)
a
array([18, 17, 1, 18, 5, 17, 0, 14, 12, 11, 4, 15, 16, 8, 7])
```

102.)

```
mask = (a%2 == 0)
mask
→ array([True, False, T, F, T, T, F, T, T, F, T, F, T, T, F])
```

MON TUE WED THU FRI SAT SUN

DATE .....

can write anything masking.

103.) extract_from_a = a[mask] # we get value where the cond^n is True.

extract_from_a.

→ array([18, 18, 0, 14, 12, 4, 16, 8])

104.) a[mask] = -1

a

→ array([-1, 17, 1, -1, 5, 17, -1, -1, -1, 11, -1, 15, -1, -1, 7])

every even/True element get the value '-1'.

105.) s = np.array([[1,2,3],[4,5,6]])

s.flatten()

→ array([1, 2, 3, 4, 5, 6])

convert a 2-D array into 1-D array.

106.) import numpy as np

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

s.shape

s.reshape(2,2,3) → 3rd dim.

→ array([[[1,2,3], [4,5,6]], → 2nd dim.

[[2,4,5], [1,4,2]]]) → 2nd dim.

→ these two in 1st dimension

3rd dim.

MTWTHFSSAT SUN

DATE

107.) in above eg -.

s. reshape (2, 2, 3, 1)

→ array ([[[[1],
[2],
[3],

[[4],
[5],
[6]]],

[[[2],
[4],
[5]]]]

[[[[1]],
[4]],
[[2]]]]).

108.) in pt. (106).

s. reshape (1, 2, 2, 3)

→ array ([[[[1, 2, 3],
[4, 5, 6]],

[[2, 4, 5],
[1, 4, 2]]]])

109.) in pt (106)

s. reshape (2, 3, 2)

→ array ([[[1, 2],
[3, 4],
[5, 6]],

[[2, 4],
[5, 1],
[4, 2]]]).

109) - 110.) s. ravel ()

→ array ([1, 2, 3, 4, 5, 6]).

convert n-D array into 1-D array.

111.) → flatten always returns a copy.

Difference

→ Ravel returns a view of the original array whenever possible. In this we don't know it return a "copy" or "view".

MON TUE WED THU FRI SAT SUN

DATE

Element wise operation-

→ if we add any no. then it will add to whole array
→ if we multiply any no. it multiple will whole array
→ square → all.
→ arithmetic operation operate elementwise

112.) matrix multiplication -

```
c = np.diag([1,2,3,4])
print(c*c)
print(c.dot(c))  → element wise multiplication.
print(np.matmul(c,c)) → matrix multiply as we done in maths.
```

→
```
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
```
} dot multiplication

```
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
```
} maths matrix multiplication.

~~113.~~ Comparisions - (element wise)

113.)
```
a = np.array([1,2,3])
b = np.array([5,2,2])
```

a.
→ → array([False, True, False], dtype = bool)

MON TUE WED THU FRI SAT SUN

DATE

114.) a>b

→ array ([False, False, True]), dtype = bool)

115.) #Array wise comparisions -

a = np.array ([1, 2, 3, 4])

b = np.array ([5, 2, 2, 4])

c = np.array ([1, 2, 3, 4])

np.array_equal(a, b)

→ False

np.array_equal (a, c)

→ True

Logical Operations -

116.) a = np.array ([1, 1, 0, 0], dtype = bool)

b = np.array ([1, 0, 1, 0], dtype = bool)

print (a)

np.logical_or (a, b)

→ [True True False False]

array ([True, True, True, False])

0 → False

1 → True

117.) np.logical_and (a, b)

→ array ([True, False, False, False], dtype = bool).

MON TUE WED THU FRI SAT SUN

DATE

Transcendental function:

118.) a = np.arrange (4) # find value of sine.

np.sin(a)

→ array ([0. , 0.84147098, 0.90929743, 0.14112000])

119.) np.log(a)

→ array ([-inf , 0. , 0.69314718, 1.09861229])

$-\infty$

120.) np.exp(a)

→ array ([1. , 2.71828183, 7.3890561, 20.08553692])

121.) Shape mismatch.

a = np.arange (4)

a + np.array ([1,2])

→ broadcasting error.

arrays with different shape cannot be add.

Basic Reduction -

122.) x = np.array ([1,2,3,4])

np.sum(x)

→ 10

MON TUE WED THU FRI SAT SUN

DDATE

123.) # sum by rows & by columns.

```
x = np.array([[1,1],[2,2]])
x
→ array([[1,1],
         [2,2]])

x.sum(axis=0)   # columns 1st dim.
→ array([3,3])

x.sum(axis=1)   # rows (2nd dim.)
→ array([2,4])
```

Other reductions -

```
                    0 1 2 ← index
                    ↓ ↓ ↓
124.) x = np.array([1,3,2])

x.min()                    x.argmin()   # index of
→ 1                        → 1          min. element

x.max()                    x.argmax()   # index of
→ 3                        → 0          max. element
```

Logical operations :

```
                  ← and
125.) np.all([True, True, False])   { If any one is false it gives false }
→ false

                  ← or
126.) np.any([True, False, False])  { If any element is true it gives true }
→ True.
```

MON TUE WED THU FRI SAT SUN

DATE .....

127.) np.any (a != b) # can be used for array comparision.
→ False.

128.) np.all (a == a) {lets say - a = [1,2,3,4]}.
→ True

129.) a = np.array ([1,2,3,2])
b = np.array ([2,2,3,2])
c = np.array ([6,4,4,5])
((a <= b) & (b <= c)).all()
→ True.

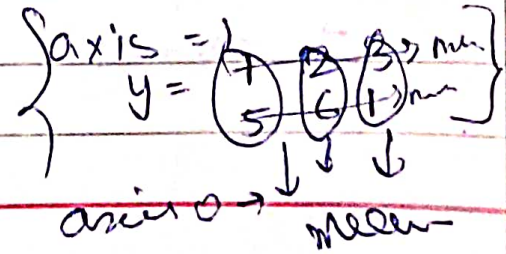
Statistics -

130.) x = np.array ([1,2,3,1])
y = np.array ([[1,2,3], [5,6,1]])
x.mean() | np.meanx()
→ 1.75

131.) np.medianx()
→ 1.75

132.) np.median (y, axis = 1) # last axis
→ array ([2., 5.])

133.) np.median (y, axis = 0)
→ array ([3., 4., 2.])



DATE

134.) x.std() # full population std. dev.
→ 0.8291561975884995

135.) # load data into numpy array object
data = np.loadtxt('filename.txt')
Data
→ data of file.
array([[1900., 30000., 4000., 48300.],
[1901., 47200., 6100., 50000.],
[1902., 70200., 5000., 57000.],
[1903., 77400., 6000., 58200.]]).

yr.	Rabbit	Cat	Carrot
1900	30000	4000	4830
1901	47200	6100	5000
1902	70200	5000	5700
1903	77400	6000	5820

↳ if this is our data

136.) year, rabbit, cat, carrot = data.T → Transpose
→ [1900. 1901. 1902. 1903] # data to columns
print(rabbit) variables.
→ [30000. 47200. 70200. 77400.]
print(cat)
→ [4000. 6100. 5000. 57000.]
print(carrot).
→ [48300. 50000. 57000. 38200.]

137.) # The mean population over time
populations = data[(:), (1:)] → take all rows
↳ start from 1st leave 0 index
populations
→ array([[30000., 4000., 48300.],
[47200., 6100., 50000.],
[70200., 5000., 57000.],
[77400., 6000., 38200.]])

DATE

138.) # sample std. dev.
populations.std (axis=0)
→ array([

139.) # which species has the highest population each yr.?
np.argmax (populations, axis=1) (horizontal)
→ array ([2, 2, 0, 0]).

140.) np.argmax (populations, axis=0)
→ array([3, 1, 2]) (vertically max. index)

Broadcasting -

Rule -

1) Two dimensions are compatible when -
1) They are equal, or (eg. 4×3 - 4×3, 2×4, 2×4).
2) one of them is 1. (i.e, either rows or column is 1 eg: 3×1, 1×3, 4×1, 1×4 etc).

141.) a = np.tile (np.arange (0,40,10), (3,1))
print (a)
a = a.T
print (a)

→ [[0 10 20 30] (3 rows, 4 cols)
[0 10 20 30]
[0 10 20 30]]
(1 column)

[[0 0 0]
[10 10 10]
[20 20 20]
[30 30 30]]

MON TUE WED THU FRI SAT SUN

DATE .....

142.) b = np.array([10, 1, 2])
b
→ array([0, 1, 2])

143.) a + b
→ array([[0, 1, 2],
[10, 11, 12],
[20, 21, 22],
[30, 31, 32]])

144.) a = np.arrange(0, 40, 10)
a.shape
→ array([0, 10, 20, 30])

145.) # add a new axis → 2D array.
a = a[:, np.newaxis]
a.shape
→ (4, 1)
a
→ ([[0],
[10],
[20],
[30]])

in then
1D → 2D
2D → 3D
3D → 4D
and so on.

146.) a + b
→ array([[0, 1, 2],
[10, 11, 12],
[20, 21, 22],
[30, 31, 32]])

DDATE

Array Shape Manipulation.

Flattening-

147.) a = np.array([[1,2,3],[4,5,6]])
a.ravel() # Return a contiguous flattened array. A 1-D array, containing the elements of i/p.
→ array([1,2,3,4,5,6])

148.) a.T # Transpose
→ array([[1,4],
[2,5],
[3,6]])

149.) a.T.ravel()
→ array([1,4,2,5,3,6])

150.) Reshape (done previously).
→ inverse opn of flattening.
→ it may also return a copy.

151.) Dimension Shuffling -
a = np.arange(4*3*2).reshape(4,3,2).
a.shape
→ (4,3,2)
a
→ array([[[0,1],
[2,3],
[4,5]],
[[6,7],
[8,9],
[10,11]],
[[12,13],
[14,15],
[16,17]],
[[18,19],
[20,21],
[22,23]]])

MON TUE WED THU FRI SAT SUN

DATE .....

152.) a[0, 2, 1]

→ 5

153.) a[[0, 2, 1]]

→ array([[0, 0, 0],
[20, 20, 20],
[10, 10, 10]])

154.) Resizing -

```
a = np.arrange(4)
a.resize((8, 0))
a
→ array ([0, 1, 2, 3, 0, 0, 0, 0])
```

155.)

```
b = a
a.resize((4, ))
```

→ Errors (as b also refer to a. so, when it a resize b also change).

Sorting -

156.)

```
# sorting along an axis:
a = np.array ([[ 5, 4, 6 ], [2, 3, 2]])
b = np.sort (a, axis = 1)
b
→ array ([[ 4, 5, 6 ],
[ 2, 2, 3]])
```

157.)

```
# in-place sort
a.sort (axis = 1)
a
→ array [[[ 4 5 6]
[2 2 3]]]
```

MON TUE WED THU FRI SAT SUN

DATE

158.) # sorting with fancy indexing -

```
a = np.array([4,3,1,2])
j = np.argsort(a)
j
```

→ array ~~[4,5,6],~~ ([2,3,1,0]). ← index are sorted.

~~[2,2,3]]~~

159.) a[j]

array([1,2,3,4])

~~PANDAS~~ (without pandas) ↓

~~160.) parsed_rows = []~~

⇒ csv → a module in python.

⇒ open → a fn to open any type of file.

Eg. with open(file_path, "r") as f:

f = open(file_path...) → after this we have to close this file

↳ read format

⇒ after ":" a ff block is started

⇒ reader.next() { change reader pointer to next row } move to next line.

⇒ .append → to add anything.

⇒ if __name__ == '__main__':

↳ to create main fn in python.

MON TUE WED THU FRI SAT SUN

DATE .....

PANDAS: Data frame: it is main obj. in pandas used to represent data with rows & columns.

Creating Dataframes -

160.) import pandas as pd

df = pd.read_csv("weather-data.csv") #read weather.csv data

df

→

	day	temp	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

(file already created)

161.) # list of tuples (make list by yourself).

```
weather_data = [('1/1/2017', 32, 6, 'Rain'),
                ('1/2/2017', 35, 7, 'Sunny'),
                ('1/3/2017', 28, 2, 'Snow')].
df = pd.DataFrame(weather_data, columns=['day',
     'temperature', 'windspeed', 'event']
```

df

→

	day	temp.	windspeed	event.
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow.

MON TUE WED THU FRI SAT SUN

DATE

162.) # get dimensions of the table.
df.shape # total no. of rows & columns.
→ (3, 4)

163.) # if you want to see initial some rows then use head command (default: 5 row
df.head() / df.head(1)
→ same table till 5 rows. ↳ only one row.

164.) # if you want to see last few rows then use tail command (default: last 5 rows).
df.tail() / df.tail(2) → last 2 rows.
→ above table till 5 last 5 rows

165.) # slicing
df[2:5]
↳ start from 2nd & end to 5

166.) df.columns # print columns in a table
→ Index(['day', 'temp', 'windspeed', 'event'],
dtype='object') {# print headings}

167.) df.day # print particular column data
→ 0 1/1/2017
1 1/2/2017
2 1/3/2017
Name: day, dtype: object

MON TUE WED THU FRI SAT SUN

DATE .....

168.) # another way of accessing column.
df['day'] # df.day (both are same).
→ same o/p as (167)

169.) # get 2 or more column.
df [['day', 'event']]
→

	day	event
0	1/1/2017	Rain
1	1/2/2017	sunny
2	1/3/2017	snow

max. or min.

170.) # get ~~all~~ temp.
df['temperature'].max() / .min()
→ 35 / 28

171.) df['temperature'].describe()
→ count
mean
std
min
25%
50%
75%
max
Name: temp. , dtype: float64

MON TUE WED THU FRI SAT SUN

DATE

172.) # select rows which has max. temp.

```
df[df.temp. == df.temperature.max()]
```

→

	day	temp	ws	event
1	1/2/2017	35	7	Sunny.

173.) # select rows which has max. temp.

```
df[df.temp. < df.temp.max()]
```

→

	day	temp.	ws	event
0	1/1/2017	32	6	Rain
2	1/3/2017	28	2	Snow

174.) Read & write XLS file. (Excel)

```
# Install : pip3 install xlrd   (or pip.)
# read excel file                (a library is required.)
df = pd.read_excel('weather_data.xlsx')
df.
```

→ excel file will open.

175.) # write DF to CSV → save to csv.

```
df.to_csv('new.csv')
```

→ file with index (0, 1, 2...)

```
df.to_csv('new_no Index.csv', index = False)
```

→ file w/o index.

with index

	day	temp.	ws	event
0				
1				
2				

w/o index

day	temp.	ws	ev

176.) # Install : pip3 install openpyxl
write DF to Excel } save dataframe
df.to_excel('new.xlsx', to excel }
Sheet_name = 'weather_data').

Groupby → a table divide in diff parts acc to columns.

177.) import pandas as pd
df = pd.read_csv ('weather_data_cities.csv')
df # weather by cities.

		day	city	temp.	ws	event
→	0	1/1/2017	newyork	32	6	Rain
	1	1/2/2017	mumbai	87	15	Fog
	2	1/3/2017	paris	50	20	Sunny

178.) g = df.groupby('city')
g
→ <pandas.core.groupby.DataFrameGroupBy obj at 0x1 }

179.) for city, city_df in g:
print (city)
print (city_df)
→ o/p will be data acc to cities
mumbai ____ data

N.Y ____ data

____ data

MON TUE WED THU FRI SAT SUN

DATE

180.) # ou to get specific group.
g.get_group ('mumbai')
→ only get data of mumbai.

181.) # find max. temp in each cities
print (g.max())
→ get max temp of all cities.

182.) # print (g.mean())
→ get mean of temp & ws of all cities.

183.) print (g.describe())
→ get the mean, std, min., 25%, 50%, 75%, max.
(acc to temp & w.s).

Concatenate Data frames-

184.) import pandas as pd
india_weather = pd.DataFrame ({
after space { "city": ["mumbai", "delhi", "pune"],
"temp": [32, 45, 30]
"humidity": [80, 60, 78]
})

india_weather

→

	city	temp	humidity
0	mumbai	32	80
1	delhi	45	60
2	pune	30	78