NUMPY (CONTINUE)

1. numpy stands for numerical python
2. numpy is for fasting speed of python by using c++, java

```
import numpy as np
arr1 = np.array([1,2,3,4,5])
arr1
```

```
array([1, 2, 3, 4, 5])
```

Double-click (or enter) to edit

Matrix

zero matrix

```
arr1 = np.zeros(9).reshape(3,3)
arr1
```

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

ones matrix

```
arr1 = np.ones(9).reshape(3,3).astype("int") #by default takes float
arr1.ndim
arr1
# arr2 = np.arange(9).reshape(3,3)
# arr2
```

```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

Eye or identitiy matrix

```
arr_1 = np.eye(3)
arr_1
```

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

```
arr_1 = np.ones(25).reshape(5,5).astype("int")
print(arr_1)
#this is the actual logic
for i in range(0,5) :
  for j in range(0,5):
    if(i == 0 or j == 0) :
      print(arr_1[i][j],end=" ")
    elif(i == len(arr_1)-1 or j == len(arr_1)-1):
      print(arr_1[i][j],end=" ")
    else :
      print("0",end=" ")
  print()
#this is done using slicing
arr_1[1:4, 1:4] = 0
print(arr_1)
```

```
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
[[1 1 1 1 1]
 [1 0 0 0 1]
 [1 0 0 0 1]
 [1 0 0 0 1]
 [1 1 1 1 1]]
```

Diagonal Matrix

```
arr_1 = np.diag([1,2,3,4])
arr_1
```

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

Random module in Numpy

```
arr_1 = np.random.rand(3,3) # create a random values between 0 and 1
arr_1
```

```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

```
arr_1 = np.random.randint(0,100, size=[3,3]) # creates integer values between 0 and 100 , range can be modified
arr_1
```

```
array([[44, 57, 54],
       [69, 19, 15],
       [13,  9, 22]])
```

Some operations on numpy arrays

```
arr_1 = np.arange(9).reshape(3,3)
arr_1
arr_1 > 1
```

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

conditional slicing

```
arr_1 = np.arange(9).reshape(3,3)
arr_1[arr_1 > 2].reshape(3,2)
```

```
array([[3, 4],
       [5, 6],
       [7, 8]])
```

```
arr_1 = np.random.randint(10,60, size=[4,4])
arr_1
# arr_1[arr_1 > 30]
```

```
array([[31, 32, 36, 27],
       [53, 59, 59, 11],
       [45, 44, 24, 41],
       [25, 35, 26, 42]])
```

```
arr_1[arr_1 > 30].reshape(3,3)
```

```
array([[40, 39, 37],
       [59, 38, 51],
       [57, 39, 48]])
```

Aggregation operations on arrays in numpy

```
np.sum(arr_1, axis = 0) #0 axis for column wise sum and 1 for row wise sum of the array
```

```
array([154, 170, 145, 121])
```

Product of two arrays

```
a = np.arange(1,7).reshape(2,3)
b = np.linspace(1,100,9).reshape(3,3).astype("int")
print(a)
print(b)
```

```
[[1 2 3]
 [4 5 6]]
[[  1  13  25]
 [ 38  50  62]
 [ 75  87 100]]
```

```
np.dot(a,b) # multiply two matrices by dot function , remember the product rules
```

```
array([[ 302,  374,  449],
       [ 644,  824, 1010]])
```

some other important operations

1. max
2. min
3. mean
4. std (standard deviation)
5. etc etc

```
print(np.std(arr_1))
np.max(arr_1, axis = 1) #max element from every row
np.min(arr_1, axis=0) # min element from every col
np.mean(arr_1)
np.corrcoef(arr_1)
```

```
array([[ 1.        ,  0.85212718, -0.70737527, -0.75124572],
       [ 0.85212718,  1.        , -0.23302388, -0.7834756 ],
       [-0.70737527, -0.23302388,  1.        ,  0.35053251],
       [-0.75124572, -0.7834756 ,  0.35053251,  1.        ]])
```

Ravel fucntion in array

```
arr_1 = np.random.randint(1,100,size=[3,3,3,3]) #4d array
arr_1
arr_2 = np.ravel(arr_1) # convertes any n-d array into 1-d array
arr_2
```

```
array([89, 46, 63, 97, 50, 34, 68, 64, 25, 34, 68, 28, 90,  3, 47, 40, 35,
       26, 28, 18, 84, 25, 82, 59, 36, 12, 72, 41, 82, 48, 92, 54, 62, 15,
       50, 18, 67, 39, 69, 83, 86, 14, 50,  7, 69, 53, 95, 87, 76, 84, 99,
        7, 18, 46, 36, 49,  9, 60,  3, 56, 28, 46, 39, 12, 83, 96, 10, 14,
       92, 61, 66, 72, 93, 94, 17, 77, 84, 96, 76,  7, 49])
```

Sorting operations

```
np.sort(arr_1, axis=1)
```

```
array([[[[28, 18, 28],
         [25,  3, 34],
         [36, 12, 25]],

        [[34, 46, 63],
         [90, 50, 47],
         [40, 35, 26]],

        [[89, 68, 84],
         [97, 82, 59],
         [68, 64, 72]]],


       [[[41, 39, 48],
               [76, 54, 14],
               [ 7,  7, 18]],

              [[53, 82, 69],
               [83, 84, 62],
               [15, 18, 46]],

              [[67, 95, 87],
               [92, 86, 99],
               [50, 50, 69]]],


             [[[12, 49,  9],
               [10,  3, 56],
               [28,  7, 39]],

              [[36, 83, 17],
               [60, 14, 92],
               [61, 46, 49]],

              [[93, 94, 96],
               [77, 84, 96],
               [76, 66, 72]]]]])
```

```
arr_2[3:6] = arr_2[3:6] + 10 # add 10 to the every element from index 3 to 5
arr_2
```

```
array([ 89,  46,  63, 107,  60,  44,  68,  64,  25,  34,  68,  28,  90,
         3,  47,  40,  35,  26,  28,  18,  84,  25,  82,  59,  36,  12,
        72,  41,  82,  48,  92,  54,  62,  15,  50,  18,  67,  39,  69,
        83,  86,  14,  50,   7,  69,  53,  95,  87,  76,  84,  99,   7,
        18,  46,  36,  49,   9,  60,   3,  56,  28,  46,  39,  12,  83,
        96,  10,  14,  92,  61,  66,  72,  93,  94,  17,  77,  84,  96,
        76,   7,  49])
```

```
np.argsort(arr_2) #give indexes of the sorted matrix, read more about it
```

```
array([13, 58, 43, 51, 79, 56, 66, 25, 63, 67, 41, 33, 74, 35, 19, 52, 21,
        8, 17, 11, 60, 18,  9, 16, 24, 54, 62, 37, 15, 27,  5, 53,  1, 61,
       14, 29, 55, 80, 34, 42, 45, 31, 59, 23,  4, 57, 69, 32,  2,  7, 70,
       36, 10,  6, 38, 44, 71, 26, 48, 78, 75, 28, 22, 39, 64, 20, 49, 76,
       40, 47,  0, 12, 68, 30, 72, 73, 46, 65, 77, 50,  3])
```

```
np.argmax(arr_2) #returns the index of the max element in the array
```

```
np.int64(3)
```

Broadcasting means when we multiply 2x3 and 3x3 matrix then result will be a 2x3 matrix

Most important questions

```
#Question1
my_array = np.arange(1,10).reshape(3,3)
print(my_array)
#question2
row_sum = np.sum(my_array, axis=1)
# print(row_sum)
#Question3
column_sum = np.mean(my_array,axis=0)
column_sum
```

```
#Question4
squared_array = my_array[0:3,0:3] ** 2
print(squared_array)
#question5
filtered_array = squared_array[squared_array > 30]
filtered_array
#question6
transposed_array = np.transpose(my_array)
transposed_array
#question7
diagonaled_array = np.diag(my_array)
diagonaled_array
#question8
flattened_array = np.ravel(my_array)
flattened_array
#question9
reshaped_array = my_array[my_array > 1].reshape(2,4)
reshaped_array
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[ 1  4  9]
 [16 25 36]
 [49 64 81]]
array([[2, 3, 4, 5],
       [6, 7, 8, 9]])
```

PANDAS LIBRARY

1. to work with data
2. two types of datatypes:- series and dataframe
3. series are for single column and for multiple column we use dataframes
4. it is used for data analyst , for extracting data, for data scientist also
5. we use pyspark for bigData

```
pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8
```

```
import pandas as pd
import numpy as np
```

```
#series = List, np.array
mylist = [1,2,3,4]
array = np.random.randint(10,20,4)
print(array)
mylist
```

```
[18 15 17 15]
[1, 2, 3, 4]
```

```
import pandas as pd
s1 = pd.Series(mylist)
s1
```

```
   0
0  1
1  2
2  3
3  4

dtype: int64
```

```
s2 = pd.Series(array)
s2
```

```
    0
0  18
1  15
2  17
3  15

dtype: int64
```

```
s2[0:3]
```

```
     0
0  5.4
1  6.1
2  1.7

dtype: float64
```

```
s2.add(s1)
```

```
    0
0  17
1  19
2  22
3  15

dtype: int64
```

```
import numpy as np
mylist = [5.4,6.1,1.7,99.8]
array = np.array(mylist)
print(mylist)
array
```

```
[5.4, 6.1, 1.7, 99.8]
array([ 5.4,  6.1,  1.7, 99.8])
```

```
import pandas as pd
b = ["One", "Two", "Three", "Four"]
a = pd.Series(mylist,b)
a
a["One"]
a[a>6]
s1,s2
s1 + s2
a + s2
pd.concat([s1,s2],axis=1)
# pd.concat([s1,a], axis=1) #1 means two cols will be created and 0 means only one row
```

```
   0   1
0  1  18
1  2  15
2  3  17
3  4  15
```

```
lables = ["one", "two", "three", "four"]
myseries4 = pd.Series(lables, mylist)
myseries4
```

```
          0
5.4     one
6.1     two
1.7   three
99.8   four

dtype: object
```

how to make different dataframes

```
arr = np.arange(1,10).reshape(3,3)
pd.DataFrame(arr) #dataframes has multiple col values
```

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

```
student_data = {
    "idno" : [x for x in range(1,4)],
    "Name" : ["aman", "gaurav", "sohan"],
    "Grade" : ["A", "B", "C"],
    "RollNo" : [301, 302, 303]
}
df = pd.DataFrame(student_data)
df #generally dataframes name contain df like studentdf,universitydf etc ...
```

	idno	Name	Grade	RollNo
0	1	aman	A	301
1	2	gaurav	B	302
2	3	sohan	C	303

Next steps: Generate code with df View recommended plots New interactive sheet

```
df["new_col_1"] = ["ones", "dos", "tres"] #add new col in dataframe
df
```

	idno	Name	Grade	RollNo	new_col_1
0	1	aman	A	301	ones
1	2	gaurav	B	302	dos
2	3	sohan	C	303	tres

Next steps: Generate code with df View recommended plots New interactive sheet

```
df["RollNo"]
```

	RollNo
0	301
1	302
2	303

dtype: int64

```
df.Grade
```

	Grade
0	grade
1	grade
2	grade

dtype: object

```
df_1 = pd.DataFrame(np.random.randint(0,100,size=(4,4)),index=["ek", "do", "tin", "chaar"], columns=["One", "Two"
df_1
```

	One	Two	Three	Four
ek	15	85	17	8
do	62	77	85	99
tin	33	90	73	42
chaar	26	49	1	76

Next steps: Generate code with df_1 View recommended plots New interactive sheet

```
df_1["One"] #works because it acts as a series in data frame but we cannot do this :- df_1["One", "Two"]
```

	One
ek	15
do	62
tin	33
chaar	26

dtype: int64

```
df_1[["One", "Two"]] #it is valid as acts as a dataframe (2-d array)
```

	One	Two
ek	15	85
do	62	77
tin	33	90
chaar	26	49

Conditional slicing

```
df_1[df_1 > 10] #conditional slicicing
```

	One	Two	Three	Four
ek	15	85	17.0	NaN
do	62	77	85.0	99.0
tin	33	90	73.0	42.0
chaar	26	49	NaN	76.0

DataFrame functions

loc() function

```
df_1.loc["ek","Four"]
```

```
np.int64(8)
```

```
df_1.loc[["ek","tin"],["One", "Three"]] #it returns the matching element like ek -> One and tin -> Three
```

	One	Three
ek	15	17
tin	33	73

iloc() function

```
df.iloc[1:3, 3:] #it works as same as slicing in DataFrame
```

	RollNo	new_col_1
1	302	dos
2	303	tres

```
df_1.iloc[1:3, 1:3]
```

	Two	Three
do	77	85
tin	90	73

columns() function

```
list(df_1.columns) #returns all the column names
```

```
['One', 'Two', 'Three', 'Four']
```

```
list(df_1.index) #returns the value of index
```

```
['ek', 'do', 'tin', 'chaar']
```

```
df_1.shape #returns the sizexsize of the dztaframe
```

```
(4, 4)
```

```
df_1.info() #reurns a short info of dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, ek to chaar
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   One     4 non-null      int64
 1   Two     4 non-null      int64
 2   Three   4 non-null      int64
 3   Four    4 non-null      int64
dtypes: int64(4)
memory usage: 332.0+ bytes
```

```
df_1.describe() #returns all the info of dataframe
```

	One	Two	Three	Four
count	4.00000	4.000000	4.000000	4.000000
mean	34.00000	75.250000	44.000000	56.250000
std	20.08316	18.300729	41.231056	39.785885
min	15.00000	49.000000	1.000000	8.000000
25%	23.25000	70.000000	13.000000	33.500000
50%	29.50000	81.000000	45.000000	59.000000
75%	40.25000	86.250000	76.000000	81.750000
max	62.00000	90.000000	85.000000	99.000000

```
df_1.describe(include="all")
```

	One	Two	Three	Four
count	4.00000	4.000000	4.000000	4.000000
mean	34.00000	75.250000	44.000000	56.250000
std	20.08316	18.300729	41.231056	39.785885
min	15.00000	49.000000	1.000000	8.000000
25%	23.25000	70.000000	13.000000	33.500000
50%	29.50000	81.000000	45.000000	59.000000
75%	40.25000	86.250000	76.000000	81.750000
max	62.00000	90.000000	85.000000	99.000000

```
df_1.describe(exclude="int") #unknown
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipython-input-70-4159882794.py in <cell line: 0>()
----> 1 df_1.describe(exclude="int")

5 frames

/usr/local/lib/python3.11/dist-packages/pandas/core/reshape/concat.py in _clean_keys_and_objs(self, objs,
keys)
    505 
    506         if len(objs_list) == 0:
--> 507             raise ValueError("No objects to concatenate")
    508 
    509         if keys is None:

ValueError: No objects to concatenate
```

Next steps: Explain error

```
df_1.dtypes
```

	0
One	int64
Two	int64
Three	int64
Four	int64

dtype: object

Practice

```
import seaborn as sns
df = sns.load_dataset("flights")
df
#read about pd.read_csv etc etc
```

	year	month	passengers
0	1949	Jan	112
1	1949	Feb	118
2	1949	Mar	132
3	1949	Apr	129
4	1949	May	121
...	...	...	...
139	1960	Aug	606
140	1960	Sep	508
141	1960	Oct	461
142	1960	Nov	390
143	1960	Dec	432

144 rows × 3 columns

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.head() #returns starting 5 values of data
```

	year	month	passengers
0	1949	Jan	112
1	1949	Feb	118
2	1949	Mar	132
3	1949	Apr	129
4	1949	May	121

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.tail() #returns last 5 values of data
```

	year	month	passengers
139	1960	Aug	606
140	1960	Sep	508
141	1960	Oct	461
142	1960	Nov	390
143	1960	Dec	432

```
df.columns
```

```
Index(['year', 'month', 'passengers'], dtype='object')
```

```
df.index
```

```
RangeIndex(start=0, stop=144, step=1)
```

```
df.rename(columns = {"year" : "Year"},inplace=True) #inplace is used to get a deep copy instead of shalow copy
df.rename(columns = {"month" : "Month", "passengers" : "Passengers"},inplace=True)
df
```

	Year	Month	Passengers
0	1949	Jan	112
1	1949	Feb	118
2	1949	Mar	132
3	1949	Apr	129
4	1949	May	121
...	...	...	...
139	1960	Aug	606
140	1960	Sep	508
141	1960	Oct	461
142	1960	Nov	390
143	1960	Dec	432

144 rows × 3 columns

Next steps: Generate code with df   View recommended plots   New interactive sheet

```
df.isnull().sum() #gives us is any value in our column is null or not
```

```
              0
Year          0
Month         0
Passengers    0

dtype: int64
```

```
df.isnull() #returns True if any missing value is there , we hav to use sum() to get the total sum
```

```
      Year  Month  Passengers
0    False  False       False
1    False  False       False
2    False  False       False
3    False  False       False
4    False  False       False
...    ...    ...         ...
139  False  False       False
140  False  False       False
141  False  False       False
142  False  False       False
143  False  False       False

144 rows × 3 columns
```

```
df.loc[1, "Month"] = np.nan #NaN is a function of numpy not pandas
df
```

```
     Year  Month  Passengers
0    1949    Jan         112
1    1949    NaN         118
2    1949    Mar         132
3    1949    Apr         129
4    1949    May         121
...   ...    ...         ...
139  1960    Aug         606
140  1960    Sep         508
141  1960    Oct         461
142  1960    Nov         390
143  1960    Dec         432

144 rows × 3 columns
```

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.reset_index()
```

	index	Year	Month	Passengers
0	0	1949	Jan	112
1	1	1949	Feb	118
2	2	1949	Mar	132
3	3	1949	Apr	129
4	4	1949	May	121
...	...	...	...	...
139	139	1960	Aug	606
140	140	1960	Sep	508
141	141	1960	Oct	461
142	142	1960	Nov	390
143	143	1960	Dec	432

144 rows × 4 columns

```
df.isnull().sum().reset_index().rename(columns={0:"Count"}) #coulmn 0 is renamed to count and year is replaced wi
```

	index	Count
0	Year	0
1	Month	1
2	Passengers	0

```
#kaggle.com :- pick up any dataset and perform operations
```

kaggle.com :- pick up any dataset and perform operations