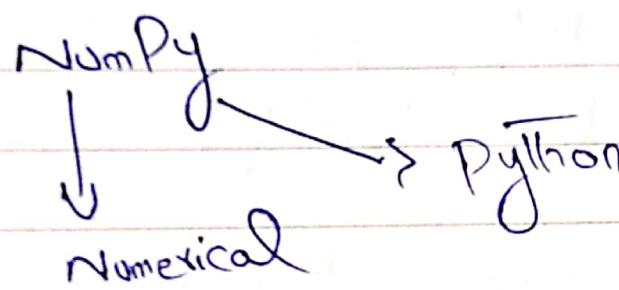


⇒ NumPy

- (i) NumPy is a Python Library
- (ii) Numpy is used for working with arrays.
- (iii) NumPy is short for "Numerical Python".



Numpy was created in 2005

by Travis Oliphant.

Q1: Why NumPy is Faster Than Lists?

Ans:

Numpy arrays are stored at one continuous place in memory + unlike lists,

So processes can access and manipulate them very efficiently. This behaviour is called locality of reference in computer science.

This is the main reason why Numpy is faster than lists.

Q2: Main difference between Array and List?

Ans:- The main difference between an array and a list is that the elements in lists are separated by commas and in array elements can't be separated by commas.

Examples:-

$a = [10, 20, 30] \Rightarrow$ list

$a = [10 20 30] \Rightarrow$ Array

⇒ Creating Numpy Arrays:

• جاں کیا لیے اسے کہا جائے گا؟

- i فنکشن array()
- ii فنکشن linspace()
- iii فنکشن logspace()
- iv فنکشن arange()

(i) linspace () :-

Syntax :-

np.linspace(start, stop, num=50, endpoint=True, type)

By default

* start ⇒ جو اسے کہا جائے گا اسے کہا جائے گا

* stop ⇒ جو اسے کہا جائے گا اسے کہا جائے گا

* num ⇒ کتنے حصے میں کرنے کا کام کرنا ہے

(By default 50), جو اسے کہا جائے گا 50 elements کرنے کے لئے 50 حصے میں کرنے کا کام کرنا ہے

* Endpoint ⇒ کہا جائے گا کہ اسے کہا جائے گا کہ اسے کہا جائے گا

Example :

```
from numpy import * → optional
a = linspace(1, 8, num=5, endpoint=True)
print(a)
```

output $\Rightarrow [1.0 \ 2.75 \ 4.5 \ 6.25 \ 8.0]$

\Rightarrow logspace ():

Syntax, \rightarrow $\{ \text{np. logspace}(\text{start}, \text{stop}, \text{num=50}, \text{endpoint=True}, \text{base=10, dtype)}$ $\}$
 اسکرپٹ میں start اور stop کو چھوڑ کر جیسا کہ np.logspace() میں دیا گیا تھا۔
 اس کا فرم $\{ \text{Base-10}\}$ ہے جیسا کہ np.logspace() میں دیا گیا تھا۔

Syntax:

```
np.logspace(start, stop, num=50, endpoint=True,
            base=10, dtype)
```

→ $\{ \text{np. logspace}(\text{start}, \text{stop}, \text{num=50}, \text{endpoint=True}, \text{base=10, dtype})\}$

iii) Arange () :

Syntax :

np.arange(start, stop, step size, dtype)

discuss \rightarrow if \downarrow \downarrow \downarrow stop, 91 start
 - \downarrow \downarrow \downarrow logspace, 91 linspace
 - \downarrow \downarrow \downarrow step size

* Step size(\downarrow) \downarrow \downarrow stop, start index
 - \downarrow \downarrow \downarrow sequence in elements

Example :

import numpy as np \uparrow stop Step size
 a = np.arange(0, 100, 10)
 ↓ start

print(a)

output \Rightarrow [0 10 20 30 40 50 60
 70 80 90]

(iv) Array () :

اے ای جس کو "Array" پر Mostly
Array خود کو array() پر Simple جس
ہے بناتے ہیں۔

Example :

```
import numpy as np
a = np.array([10, 20, 30, 40, 50])
```

print(a)

output \Rightarrow [10 20 30 40 50]

جس میں Array ہے تو اسے پرfer کرنے والے اور
کوئی نہیں اسے پرfer کرنے والے اور اسے
array() کو خود جس کو فنکشن اسے array
کرنے والے اس کو اسے اسے اسے
array() کو خود جس کو فنکشن اسے array
کرنے والے اسے اسے اسے اسے اسے

\Rightarrow Dimensions in Arrays :

(i) 0-D Array :

ج: ج: اے لیکس اریئ (لیکس) اے ڈیزے-ڈی
ج: کولمن ج: اے ڈیزے، ج: راؤ ج: اے ڈیزے

Example :

$$a++ = np.array([42])$$

(ii) 2-D and 3-D Array :

Arrays Multidimensional ج: اے ڈیزے 3D اے ڈیزے 2D
ج: ڈیزے اے ڈیزے اے ڈیزے Multidimensional Array ج: اے ڈیزے
ج: Rows اے ڈیزے اے ڈیزے ج: اے ڈیزے ج: اے ڈیزے
ج: Columns اے ڈیزے اے ڈیزے اے ڈیزے

Examples :

2-D Array :-

import numpy as np
~~a = np.array([[0, 10, 20, 30]])~~

a = np.array([[0, 10, 20], [30, 40, 50]])

print(a)

Output \Rightarrow [[0 10 20], [30 40 50]]

3-D Array :-

import numpy as np

a = np.array([[[0, 10, 20], [30, 40, 50]]])

print(a)

Output # [[0 10 20] [30 40 50]]

Close Jupyter Notebook 3-D Array

Ctrl + Shift + F5 Run 3-D Array

Mo	Tu	We	Th	Fr	Sa	Su
----	----	----	----	----	----	----

Date: _____

○ Array is called 3-D if 2D
• If 2D array 4-D or 3D close

1-D Array :-

↳ Row is called 1-D Array
• If 2D, Columns Multiple of

Example :-

```
a = np.array([0, 10, 20, 30])
```

Accessing Arrays :-

Any type of array 1-D, 2-D
3-D will be accessed by
index number.

1-D Array is simple. I
will discuss about 2-D, 3-D
array or next page.
Come on to next page. 😊

3-D, 2-D

Accessing Elements:

$a = np.array([[0, 10, 20], [30, 40, 50]])$

1) Rows (نحوں کی تکمیل میں تو پہلے ہے row,
وہ کسی 2D arr کی نیچے کی تکمیل کی جاتی ہے Column (کالونیں)
 ↓ ↓
 3 columns 10 20
 → 0 10 20
 2 Rows 30 40 50 ↑ column
 → 30 40 50

Index No.:

	0	1	2	
0	0	10	20	Row
1	30	40	50	

کوئی 2 یا 3 Access کی Element کی کی
 اس کو 1 Column کی 1 Row کی طرف رکھو

Example:

i) $z = a[0][0]$

print(z)

output $\Rightarrow 0$

(ii) `print(a[1][2])`

Output => 50

- لیست 2 پر 50 same گز 3 - D Array

Example :-

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9],
[10, 11, 12]]])
```

`print(arr[0, 0, 2])`

Output => 3

لیست index یا دیکسٹری میں 3 کا جگہ
- لیست کے ایک 2 - D ایڈجیٹ میں 2 کے تین
Array 2 - D کو 2 کے تین واحدهیں کہا جاتا ہے
کہ یہ 2 کے تین واحدهیں کہا جاتا ہے
کہ یہ 2 کے تین واحدهیں کہا جاتا ہے
کہ یہ 2 کے تین واحدهیں کہا جاتا ہے

⇒ Slicing Arrays :

Slicing in Python means taking elements from one given index to another given index.

Examples

(i)

$\underline{1-D}$

$$a = \text{array}([1, 2, 3, 4, 5])$$

(Print [a])

$\text{print}(a[2])$

Output \Rightarrow 3

(ii)

$\underline{2-D}$

$$a = \text{array}([[1, 2, 3, 4], [5, 6, 7, 8]])$$

$\text{print}(a[0:2, 1:3])$

Output $\Rightarrow [[2, 3], [6, 7]]$

Q-D :-

import numpy as np

```
a = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])  
print(a[0,:,:1:3])  
# output [[2 3][5 6]]
```

* Here ^{In} a[0,:,:1:3] → Shows columns
in respective dimension
↳ Shows rows in respective dimension
↳ Shows how many 2-D dimensions

Copy Us View ::

Copy :: The copy() owns

data and any changes made to copy will not affect the

original array, and changes made in original array not affect the copy.

Example:

i import numpy as np

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

x = a.copy

a[0] = 40

[Sorry Comma
use ni hoga]

print(x) # [1, 2, 3, 4]

print(a) # [40, 2, 3, 4]

⇒ Same if changes made in copy as:

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

x = a.copy()

x[0] = 50 # [50, 2, 3, 4]

print(x) # [1, 2, 3, 4]

View ():-

کوئی میں view() سے اگر ہے میں View() میں اسی Array original 09 ترکی لائی ہے اور اسی میں Original کا اور 32 ترکی کی ترکی ہے اسی میں View() کا ترکی دیتے ہیں۔

Example :-

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

$$x = a.view()$$

$$a[0] = 40$$

$$x[1] = 50$$

$$\text{print}(x) \# [40, 50, 3, 4]$$

$$\text{print}(a) \# [40 50 3 4]$$

⇒ Shape of Arrays:

જેવું એ લિએ એ એ શેપ ઓફ એરેઝ
જીન્હે કોમ્લન્સ એ રોઝ એ એરેઝ.

$$a = \begin{array}{c} \text{array} \\ \left[[1, 2, 3], [4, 5, 6] \right] \end{array}$$

print(a.shape)

output (2, 3)

Rows Column

-> એ એ શેપ એન્ધ

Reshape of Arrays:

Reshaping means Changing The
Shape of an array.

Examples ::

(i)

```
a = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
```

```
print(a.reshape(3,4))
```

output
[[1 2 3 4
 5 6 7 8
 9 10 11 12]]

Here In (a.reshape(3,4))

Rows Columns

```
(ii) a = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
```

```
print(a.reshape(2,2,3))
```

Columns

Rows

output
[[[1 2 3
 4 5 6]]
 [[7 8 9
 10 11 12]]]

How many dimensions

Array Iterating:

For 1-D:

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

for i in a:

print(i)

output 1
2
3
4

For 2-D:

(ii) $a = np.array([[1, 2, 3], [4, 5, 6]])$

for i in a:

for z in i:

print(z)

output =>

1
2
3
4
5
6

(iii) For 3-D :-

$a = np.array([[[1, 2], [3, 4], [[5, 6], [7, 8]]]])$

for i in a:

 for x in i:

 for z in x:

 print(z)

output \Rightarrow

1
3
4
5
6
7
8

\Rightarrow Array Joining :-

Join 2 arrays

Join 2

- Concatenate (i)

- Stack (ii)

i) Concatenate () :-

Join Arrays 2 یعنی اس کے
لئے اس کے 2 دیگر اس کے
arrays کو join کر دیا جائے۔

Ex:-

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

```
b = np.array([4,5,6])
```

```
print(np.concatenate((a,b)))
```

output : [1 2 3 4 5 6]

Join 2-D arrays یعنی 2-D array
کو 3-D array کے لئے axis wise

Example:-

```
a = array([[1,2,3],[4,5,6]])
```

```
b = array([[7,8,9],[10,11,12]])
```

```
print(concatenate((a,b),axis=1))
```

Output \Rightarrow $\begin{bmatrix} [1, 2, 3] \\ [4, 5, 6] \end{bmatrix}$

Stack () :

میں کے پہلے دو گھنیں Stack()
 (i) (ii) Horizontal (iii) Vertical
 (iv) Join (v) \rightarrow Stack()

Vertical \leftarrow vstack(i)
 Horizontal \leftarrow hstack(ii)
 depth \leftarrow dstack() (iii)

Examples:

i) $a = \text{array}([1, 2, 3])$
 b = $\text{array}([4, 5, 6])$
 $\text{print}(\text{vstack}(a, b))$
 # output $\begin{bmatrix} [1, 2, 3] \\ [4, 5, 6] \end{bmatrix}$



(ii)

hstack :- $a = \text{array}([1, 2, 3])$ $b = \text{array}([4, 5, 6])$ $\text{print}(hstack(a, b))$

Output [1 2 3 4 5 6]

(iii)

dstack :- $a = \text{array}([1, 2, 3])$ $b = \text{array}([4, 5, 6])$ $\text{print}(dstack(a, b))$ # Output [[[1 4]
[2 5]
[3 6]]]]

Array Split ::

Splitting is reverse operation of joining.

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

`x = np.array_split(a, 3)`

`print(x)`

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

Searching Arrays ::

Search میں طریقہ ہے Arrays

i) `.where()`
ii) `.Searchsorted()`

where() :

3 Element Certain جو julie where()
Element 0, 1, 2, 3, 4, 5, 6, 7 indexes
- 9, 10, Repeat 1, 0, 1, 2, 3, 4, 5, 6, 7

Ex:

```
a = np.array([1, 2, 3, 4, 3, 5, 3])
```

```
print(np.where(a == 3))
```

```
# output(array([2, 4, 6]),)
```

Set (,) for condition
Ans (,) b

Searchsorted ::

Searchsorted() performs a binary search in the array and returns the index where specified value would be inserted to maintain the search order.

index (will Searchsorted() کے لئے اسکے
insert value پر جاگزئی کرتا ہے
جسے بے طے کرنے کے لئے)

`a = np.array([5,6,7,8,9])`

`print(np.searchsorted(a, 4))`

output # (0) or (zero)

insert \rightarrow Zero for 4, Value \rightarrow یعنی
 \rightarrow کریں

Multiples values can also be
searchsorted.

`(a = np.searchsorted())`

`a = array([1,3,5,7])`

`print(searchsorted(a, [2,4,6]))`

output \Rightarrow [1 2 3]

.. بحث

الدالة `Searchsorted()`

- `Sorted array` \rightarrow مُرتّب
- `Sorted 1D array` \rightarrow مُرتّب `where()`
- `On which` \rightarrow `Elements`

Sorting Arrays :

Sorting means putting elements in an ordered sequence.

In NumPy we use `sort()`, That will sort a specified array.

Example:

```
a = np.array([3, 1, 2, 0])
print(np.sort(a))
```

output [0123]

(ii)

```
a = np.array(['banana', 'cherry', 'apple'])
print(np.sort(a))
```

output ['apple' 'banana' 'cherry']

(iii)

```
a = np.array([True, False, True])
print(np.sort(a))
```

output [False, True, True]

اے جاں پر آریو جس کے لئے sort()
اوے جاں پر ایسے لئے sorted() کے لئے Use
گے 2D - 3D آریو

Filtering Arrays :

Getting some elements out of an existing array and creating a new array out of them is called Filtering.

i) Creating The Filter Array:

```
a = np.array([41, 42, 43, 44])
```

```
filter_array = []
```

```
for i in a:
```

```
    if i > 42:
```

```
        filter_array.append(True)
```

```
    else:
```

```
        filter_array.append(False)
```

```
new_array = a[filter_array]
```

```
print(new_array)
```

```
print(filter_array)
```

Output # [False, False, True, True]
[43, 44]

=> Creating Filter Directly From

Array :

a = np.array ([41, 42, 43, 44])

filter-arr = a > 42

newarr = a [filter-arr]

Print (filter-arr)

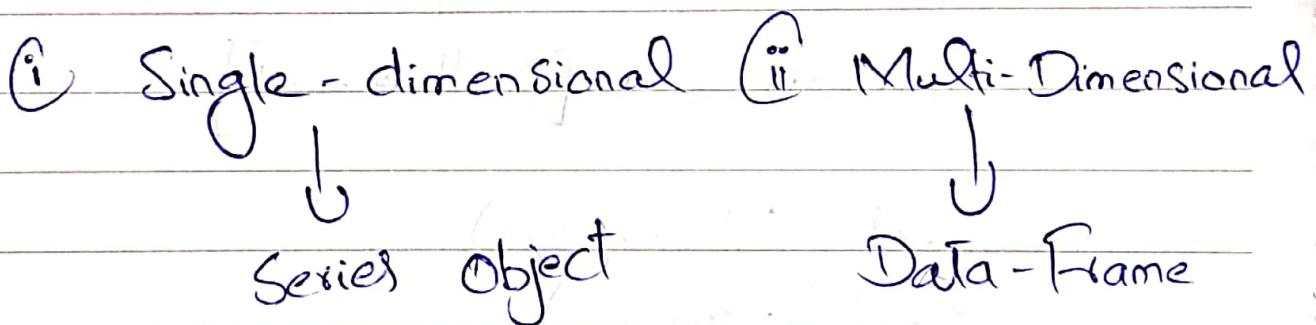
Print (newarr)

Output # [False, False, True, True]
[43, 44]

Pandas

Pandas stands for panel Data and is the core library for data manipulation and data analysis.

It consists of single and multiple dimensional data structures for data manipulation.



→ i) Pandas Series object:

Series object is one-dimensional labeled array.

Example:

```
import pandas as pd  
s = pd.Series([1, 2, 3, 4, 5])  
print(s)
```

Output: 0 1
1 2
2 3
3 4
4 5

In numpy array, array is not labeled but here in pandas it is labeled.

→ Changing Index:

```
import pandas as pd  
s = pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e'])  
print(s)
```

Output: a 1
b 2
c 3
d 4
e 5

⇒ Creating Series Object From Dictionary:

```
import pandas as pd
```

```
z = pd.Series({'a': 20, 'b': 30, 'c': 40})
```

```
print(z)
```

Output :-

```
a    20
```

```
b    30
```

```
c    40
```

Here "Keys" becomes Indexes as a label.

⇒ Changing Index Position:

```
import pandas as pd
```

```
z = pd.Series({'a': 10, 'b': 20, 'c': 30}, index=['b',  
        'c', 'd', 'a'])
```

```
print(z)
```

Mo Tu We Th Fr Sa Su

Date: _____

Output :-

b	20.0
c	30.0
a	10.0
d	NaN

Extracting Elements :-

```
import pandas as pd  
s = pd.Series([1,2,3,4,5])  
print(s[2])
```

Output : 3

If we :

```
print(s[2:])
```

Output : 2 3

3 4

4 5

If we print :
print(s[2:4])

Output : 1 2
2 3
3 4

we can also use negative indexing

print(s[-3:])

Output: 2 3
3 4
4 5

Basic Math operations on Series Obj.:

i

```
import pandas as pd  
z = pd.Series([1, 2, 3, 4, 5])
```

print(z+5) # Adding scalar Value to Series Element

Output: 0 6
1 7
2 8
3 9
4 10

Adding two Series object:

```
import pandas as pd  
z = pd.Series([1, 2, 3, 4, 5])  
s = pd.Series([6, 7, 8, 9, 10])
```

print(z+s)

Output :

0	7
1	9
2	11
3	13
4	15

ii Multi-Dimensional (Data Frame) :-

Data-Frame is a 2-Dimensional Labelled data-structure. Data Frame Comprises of rows and columns.

```
import pandas as pd  
z=pd.DataFrame( {"Name":['Ali','Moni','Mudasir'],  
"Marks": [90, 80, 92]})
```

print(z)

Output :-

	Name	Marks
0	Ali	90
1	Moni	80
2	Mudasir	92

In-Built DataFrame Functions ::

(i) head() ::

Value میں سے Data Frame کی head() فنکشن اسے head() لفظ کو کرواتا ہے اس کو indexers پر بھی بارہے ہے اسے values کو بخوبی کہتا ہے

(ii) tail() ::

آخری پانچ سے آخری پانچ Data Frame کی tail() فنکشن اسے tail() لفظ کو کرواتا ہے اس کو indexers پر بخوبی کہتا ہے values

(iii) shape ::

پہلے یہیں - ہے جس کی shape فنکشن اسے shape کہتا ہے اس کے columns اور rows کی DataFrame کو columns اور rows کے تعداد کے برابر ہے

(iv) describe() ::

DataFrame پر یہیں describe() فنکشن اسے describe() کہتا ہے info کی نیز میں ہے اسے older کہتا ہے

count

mean

std

min

25%

50%

75%

max

⇒ Extracting Rows and Columns ::

(i) .iloc

(ii) .loc

(i) iloc ::→ (index Location)

index of columns, \ or Rows f, G. iloc
. O. i ↗ Extract }z.iloc [0:3, 0:2] Exclusive (not included)
 } }
 Rows columns

Mo Tu We Th Fr Sa Su

Date: _____

(ii)

.loc ::

↳ Columns او indexes' Rows فی جو Loc
↳ Extract میں کسی ایسے جو indexes

z.loc[0:3, ("name", "Marks")]
↓
inclusive (included)

⇒ Dropping Columns ::
↳ کسی جو column (جس کو DataFrame
- جس کو حذف کرنا Drop()

z.drop('name', axis=1)
↓

Column Name
(کسی drop کرنے کا جس کو)

:: "1" Shows Columns

:: "0" Shows Row

⇒ Dropping Rows ::

↳ کسی جو Row (Column) (جس کو DataFrame
- کو حذف کرنا Drop() سامنے)

`z.drop([1,2,3], axis=0)`

↓
Rows Name
(ر�جھ کو ختم کرو جو?)

More Pandas Functions :-

- (i) `mean()` - ↗ \bar{x} \hat{x} : Average ^{value}
- (ii) `median()` - ↗ \tilde{x} : Value(s) \rightarrow میانہ
- (iii) `min()` - ↗ \underline{x} : Minimum Value
- (iv) `max()` - ↗ \overline{x} : Maximum ^{value}

$\Rightarrow z.mean()$

$\Rightarrow z.median()$

$\Rightarrow z.min()$

$\Rightarrow z.max()$