NUMPY

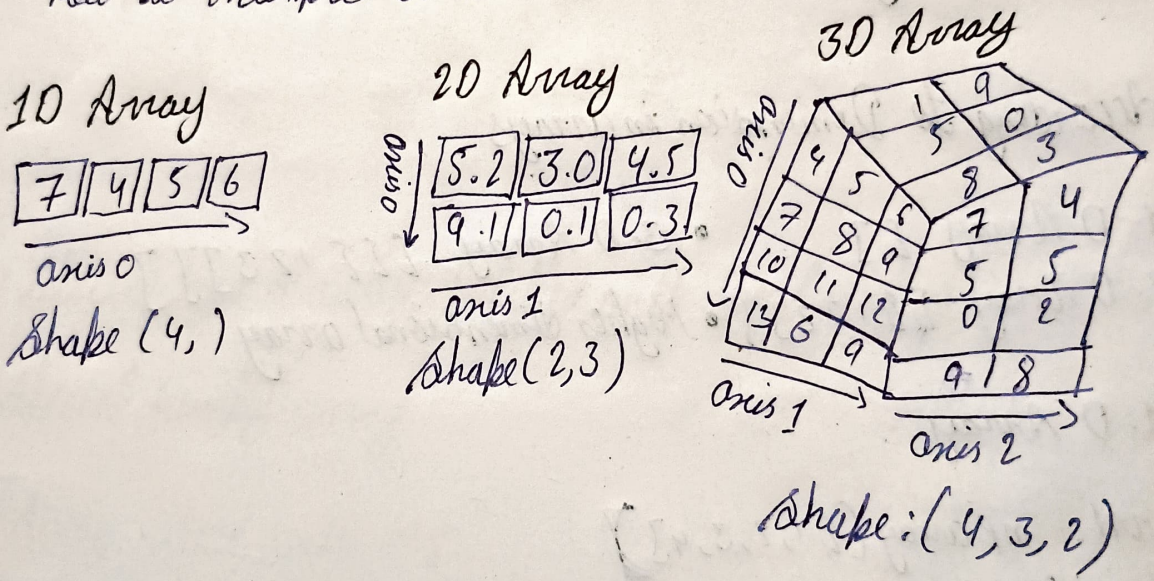
- What is numpy?

Ans: Numpy is the fundamental package for scientific computing in Python.

-> Numpy is a python libraries that provides a multidimensional array object, various derived object

- What is numpy Array?

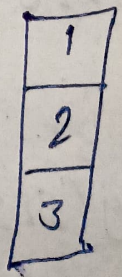
Ans: An array is a grid of value and it contains information about the raw data, how to locate an element, and how to interpret an element.



=> Creating Numpy Arrays

- To create a numpy Array, you can use the function np.array()

```
import numpy as np
a = np.array([1,2,3])
Print(a)
```



Taking input numpy array:

```
list = [ ]
For i in range(1,5):
    int_1 = int(input("Enter number:"))
    list.append(int_1)
print(np.array(list))
```

→ it convert list into array

```
print(list.ndim)
```

→ It help us to check the dimension of array

Type of array:

=> According to Dimension in arrays

- 1-D Array [ ]
- 2-D Array [[1 2 3]]
- 3-D Array [[[1 2 3]]]
- Higher dimensional array

- 1-D Array

```
arr1 = np.array([1,2,3,4])
print(arr1)
```

- 2-D Array

```
arr2 = np.array([[1,2,3,4],[1,2,3,4]])
print(arr2)
```

- 3-D Array

```
arr3 = np.array([[[1,2,3,4],[1,2,3,4],[1,2,3,4]]])
print(arr3)
```

Note: The data should be equal in array. else it give error.

Ex: arr2 = np.array([[1,2,3,4],[1,2,3]])

print(arr2) # output = error

- Multi dimension Array

ndmin → It helps to create multi-D array

```
arr = np.array([1,2,3,4], ndmin=10)
print(arr)
print(arr.ndim)
```

output: [[[[[[[[[1,2,3,4]]]]]]]]]

10

Special type of arrays (Filled with specific value)

=> Special numpy array

- Array filled with 0's

```
import numpy as np
ar_zero = np.zeros(4)
ar_zero1 = np.zero((3,4))
print(ar_zero)
print(arr_zero1)
```

→ [0. 0. 0. 0.]

output →

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

- Array filled with 1's

```
ar_one = np.ones(4)        → element
Print(ar_one)
```

output: [1. 1. 1. 1.]

- Creating an Empty array

```
ar_emp = np.empty(4)
Print(ar_em)
```

output: [1. 1. 1. 1.] It come because in empty array previous data is come out

- An array with a range of elements

```
ar_rn => np.arange(4)
Print(ar_rn)
```

output = [0 1 2 3]

- Array diagonal element filled with 1's identity matrix

```
Dig = np.eye(3)
Print(Dig)
```

(3,5)

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Creating an array with values that are spaced linearly in a specified interval

```
ar_line = np.linspace(0,10, num=5)        num → element
Print(ar_line)                  0, 20, num = 5
```

output = [0. 2.5 5. 7.5 10.]
[0. 5. 10. 15. 20.]

Creating Numby Arrays with Random numbers.

- rand() : The function is used to generate a random value between 0 to 1.

```
import numby as np
Var = np.random.rand(4)
Print(Var)
```

Output: [0.245467 0.1258932 0.5671234 0.9875456]
1-D Array

For 2-D = (3,5)

- randn() : The function is used to generate a random value close to zero. This may return positive or negative number as well.

```
var2 = np.random.randn(4)
print(var2)
```

Output: [0.7437 0.3754 -1.4567 -0.2549]

- **randn**: This function for doing random sampling in numpy. It return an array of specified shape and fill it with random floats in the half-open interval

  [0.0, 1.0) → This sign mean it not include 1.

  This include 0.0

```
var3 = np.random.rand(4)
print(var3)
```

Output: [0.02956 0.007812 0.917456 0.530491]

- **randint()**: The function is used to generates a random between a given range.

```
var4 = np.random.randint(05, 20, 5)
print(var4)
```

(05 → min, 20 → max, 5 → total number)

Output: [5 14 8 12 9]

Data type Numpy Arrays

Sr.No	Data Type	Description
1.	bool_	Boolen (True or False) store as 1 byte
2.	int	Default integer type (same as C long; normally either int64 or int32)
3.	intc	Identical to C int (normally int32 or int64)
4.	intp	Integer used for indexing (same as C size_t, normally either int32 or int64)
5.	int8	Byte (-128 to 127)
6.	int16	Integer (-32768 to 32767)
7.	int32	integer (-2147483648 to 2147483647)
8.	int64	integer (-9223372036854775808 to 9223372036854775807)
9.	uint8	Unsigned integer (0 to 255)
10.	uint16	Unsigned integer (0 to 65535)
11.	uint32	Unsigned integer (0 to 4294967295)
12.	uint64	Unsigned integer (0 to 18446744073709551615)

13.	Float_	Shorthand for float64
14.	Float16	Half precision float: Signed bit, 5 bits exponent, 10 bits mantissa
15.	Float32	Single precision float: Sign bit, 8 bits exponents, 16 bits mantissa
16.	Float64	Double Precision float: Single bit, 11 bits exponents, 22 bits mantissa
17.	Complex_	Shorthand for complex128
18.	Complex64	Complex number, represented by two 32-bit floats (Real and imaginary components)
19.	Complex128	Complex number, represented by two 64-bits floats (reals and imaginary components

-> List of characters that are used to represent dtype in Numpy

- i – integer
- b – boolean
- u – unsigned integer
- f – float
- c – complex float
- m – timedelta
- M – datetime
- O – object
- S – string
- U – Unicode string
- V – The fixed chunks of memory for other type (void)

(Capitals)

-> Import numpy as np

Var = np.array([1,2,3,4,5,16,17])

Print("Data type:", var.dtype)

output: Data type: int64

2) Var = np.array([1.1, 2.3, 3.4, 6.8, 9.4])

Print("Data type:", var.dtype

Output: Data type:

3) For string

```
var = np.array(["s", "r", "w"])
print("Data type: ", var.dtype)
```

Output: <U1

4) For mixing

```
var = np.array(["s","b","i", 4, 3, 7])
print("Data type: ", var.dtype)
```

output: <21

To convert in another dtype.

```
x = np.array([1,2,3,4,5], dtype = np.int8)
print("Data type: ", x.dtype)
```

Output: It convert int64 bit into int8

: Another way to convert

```
x = np.array([1,2,3,4], dtype = "f")
print("Data type: ", x.dtype
```

output: float32

Shape & Reshaping in Numpy array

→ import numpy as np

```
var = np.array([[1,2,3,4],[1,2,3,4]])
print(var.shape)
```

output: (2, 4)

• shape : is used to find the shape of array

```
var = np.array([1,2,3,4], ndmin=4)
print(var)
print(var.ndim)        # is used to find dimension of array
print()                # for gap
print(var.shape)       # To find the shape
```

output: [[[[1, 2, 3, 4]]]]

(1, 1, 1, 4) → 4 column

: Reshape

```
var2 = np.array([[1,2,3,4,5,6])
print(var2)
print(var.ndim)
```

Print()

```
x = var2.reshape(3,2)
Print(x)
Print(x.ndim)
```

Output: [1 2 3 4 5 6]
1

[[1 2]
[3 4]
[5 6]]
2

Note: Reshape करने के लिए Reshape में पास किया हुआ element size of Array element के बराबर होना चाहिए।

Array element = 6

(3,2) ✓ → 6

(3,3) ✗ = 9

but we have only 6 element in array

* For Multidimension Array

```
var3 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
Print(var3)
Print(var3.ndim)
```

Print()

```
x1 = var3.reshape(2,3,2)
Print(x1)
Print(x1.ndim)
```

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

[[[1 2]
[3 4]
[5 6]]

[[7 8]
[9 10]
[11 12]]]

3

To convert into multi to one dimension

Print()

```
one = x1.reshape(-1)
Print(one)
Print(one.ndim)
```

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

Arithmetic operation in Numpy Arrays

- a+b ⟶ np.add(a,b)
- a-b ⟶ np.substract(a,b)
- a*b ⟶ np.multiply(a,b)
- a/b ⟶ np.divide(a,b)
- a%b ⟶ np.mod(a,b)
- a**b ⟶ np.power(a,b)
- 1/a ⟶ np.reciprocal(a)

```
import numpy as np
var = np.Array([1,2,3,4])
varadd = var+3          # np.add(var,3)
print(varadd)
```

Output: [3, 5, 6, 7]

```
var1 = np.array([1,2,3,4])
var2 = np.array([1,2,3,4])
varadd = var1+var2      # np.add(var1+var2)
print(varadd)
```

Output: [2, 4 6 8]

2D Array

```
var21 = np.array([[1,2,3,4],[1,2,3,4]])
var22 = np.array([[1,2,3,4],[1,2,3,4])
print(var21)
print()
print(var22)
print()
~~print++~~
varadd = var21 + var22
print(varadd
```

Output: [[1 2 3 4]
[1 2 3 4]]

[[1 2 3 4]
[1 2 3 4]]

[[2 4 6 8]
[2 4 6 8]]

Arithmetic Function

- np.min(x)
- np.max(x)
- np.argmin(x) → Find position
- np.sqrt(x)
- np.sin(x)
- np.cos(x)
- np.cumsum(x)
  Like Fibonacci number

```
Var = np.array([4,4,1,4,5,3,2])
Print("min:", np.min(var), np.argmin(var))
Print("max:", np.max(var), np.argmin(var))
Print("Sqrt:", np.Sqrt(var))
```

Output: Min : 1 Position = 2

Max = 5 4

Sqrt: [2. 2. 1. 2. 2.236068 1.7320508
1.4142135]

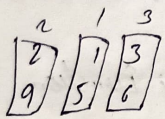
-> Var1 = np.array([[2,1,3],[9,5,6]])

Print(np.min(Var1, axis=0)

Output: [2 1 3]

axis = 0 Col

axis = 1 Row



-> Var2 = np.array([1,2,3])

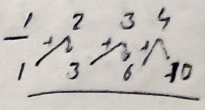
Print(np.Sqrt(Var2))

Output: [0.8417 0.9092 0.14112001]

-> Var3 = np.array([1,2,3,4])

Print(np.cumsum(Var3))

Output: 1 3 6 10



Broadcasting Numpy Array

```
import numpy as np
Var1 = np.array([1,2,3])
Print(Var1.shape)
Print()
Print(Var1)
Print()
Var2 = np.array([[1],[2],[3]])
Print(Var2.shape)
Print()
Print(Var2)
Print()
Print(Var1+Var2)
```

— output : (3,)
means (1,3)

— output : (3,1)

— [[2 3 4]
[3 4 5]
[4 5 6]]

For Broadcasting we have two condition

V = [1 2 3]
(1×3)

$V_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ (3×1)

1) Same dimension
2) (1×3) (3×1) — 3×3

1 होना चाहिए दोनों में से किसी एक में ।

How it work

V = [1 2 3] → [1, 2, 3] (column)

$\begin{bmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix}$ final output

→ [1 2 3] [1 2 3 4] dimes same but
(1×3) (1×4) — error ✗

(1×3) (1×3) — ✓

```
-> x = np.array([[1], [2]])          (2×1)
print(x.shape)
print()
y = np.array([[1,2,3],[1,2,3]])      (2×3)
print(y.shape)
print(x+y)
```

output: (2,1)
(2,3)

[[2 3 4
3 4 5]]

(2×1) (2×3) — (2, 3)

Indexing Numpy Arrays

[1 2 3 4]
0 1 2 3 - Indexing
-4 -3 -2 -1

1D

[[1 2] — 0
[1 2]] — 1
(columns: 0 1)

2D

[[[1 2] — 0
[1 2]] — 0
[[1 2] — 1
[1 2]]] — 1
(columns: 0 1)

3D

: For 1D Array

```
Import numby as np
Var = np array([1,2,3,4])
                0 1 2 3  -> Indexing number
Print(var[1])
Print(var[-3])
```

Output: 2
2

: For 2D Array

```
Var1 = np array([[1,2,3,4],[5,6,7,8]])
                 0 Row      1 Row
Print(var[1,2])
          ! 2 Index
```

1 => Mean finding in 1 Row
2 => 2nd Index

Output: 7

: For 3D Array

```
Var2 = np array([[[1,2],[6,7]]])
Print(Var2)
Print(Var2.ndim)
Print()
Print(Var2(0,1,1)
```

```
      0
[ [ [ 1 2]
    [ 6 7 ] ] ]
0       1
        0 1
```

Slicing in Numby Array

```
import numpy as np
Var = np array([9,8,7,6,5,4,3])

Print("2 to 5:", Var[1:5])
Print("2 to End:", var[1:])
Print("Start to 5:", Var[:5])
Print("Step:", Var[1:5:2]
```

Output:

For 2D

```
Var1 = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
                 Row 0     Row 1     Row 2
Print(var1)
Print(var1[1, 0:3])
          Row  Indexing number
```

Iterating Numby Array

```
Import numby as np
Var = np.array([9,8,7,6,5,4])
Print(var)
Print()
For i in Var:
    Print(var)
```

Output: 9 8 7 6 5 4

```
Var1 = np.array([[1,2,3,4],[5,6,7,8]])
Print(Var1)
Print()

For J in Var1:
    Print(J)
```

-> [1 2 3 4] [5 6 7 8]

```
Print()

For k in Var1:
    For J in k:
        Print(J)
```

->

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

```
Var3 => np.array([[[1,2,3,4],[5,6,7,8]]])
Print(Var3)
Print(Var3.ndim)
Print()

For i in Var3:
    For J in i:
        For k in J:
            Print(k)
```

->

Output:

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

=> nditer() function

```
For i in np.nditer(Var3):
    Print(i)
```

=>

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

```
=> For i,d in np.ndenumerate(Var3):
    Print(i,d)
```

Output: (0,0,0) 1
(0,0,1) 2
(0,0,2) 3
Index value -> (0,0,3) 4
(0,1,0) 5
(0,1,1) 6
(0,1,2) 7
(0,1,3) 8

Copy and View

```
import numpy as np
Var = np.array([1,2,3,4])
CO = Var.Copy()
Print ("Var :", Var)
Print("Copy :", CO)
```

output:
Var : [1 2 3 4]
Copy : [1 2 3 4]

```
Var1 = np.array([5,6,7,8])
Vi = Var1.View()
Print("Var:", V)
Print("View :", Vi)
```

output:
Var1 : [5,6 7 8]
View : [5 6 7 8]

Basic difference

- The copy owns the data
- The copy of an array is a new array.
- The changes made in the copy data does not reflect in the original array

- The view does not own the data
- A view of the original array
- Any change made to the view will affect the original array, and any change made to the original array will affect the view.

Example:

```
import numpy as np
Var = np.array([1,2,3,4])
Co = Var.Copy()
Var[1] = 40
Print("Var :", Var)
Print("Copy :", Co)
```

output:
Var : [1 40 3 4]
Copy : [1 2 3 4]

```
X = np.array([9,8,7,5,6])
Vi = X.View()
X[1] = 40
Print("X :", X)
Print("View :", Vi)
```

output:
X = [9 40 7 6 5]
Vi = [9 40 7 6 5]

Join & Split Function Numpy array

Join Array: Joining means putting contents of two or more arrays in a single array.

```
import numpy as np
Var =np.array([1,2,3,4])
Var1 = np.array([5,6,7,8])
ar = np.Concatenate((Var,Var1))
Print(ar)
```

output:
[1 2 3 4 5 6 7 8]

```
Var = np.array([[1,2],[3,4]])
Var1 = np.array([[5,6],[7,8]])

ar_new = np.concatenate((Var, Var1), axis=0)

print(Var)
print()
print(Var1)
print()
print(ar_new)
```

output:

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

[[5 6]
 [7 7]]

[[1 2 3 6]
 [3 4 7 8]]
```

When axis=0

then the output is =>

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

Note: we have also one more function name (stack) that we use instead of concatenate.

We have also three type of stack we use:

```
a_new = np.stack((Var_1, Var_2), axis=1)
a_new1 = np.hstack((Var_1, Var_2))   # Row
         np.vstack((Var_1, Var_2))   # Column
         np.dstack((Var_1, Var_2))   # height
```

output:

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

[1 2 3 4 9 8 7 6]

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

Split Array

- Splitting breaks one array into multiple

```
import numpy as np
Var = np.array([1,2,3,4,5,6])
print(Var)
ar = np.array_split(Var, 3)
print(ar)
print(type(ar))
print(Var[0])
```

output:

- [array([1,2]), array([3,4]), array([5,6])]
- type <list>
- [1 2]

```
Var1 = np.array([[1,2],[3,4],[5,6]])
Print(Var)
ar1 = np.array_split(Var1, 3)
ar2 = np.array_split(Var1, 3, axis=1)
Print()
Print(ar1)
Print()
Print(ar2)
```

output:

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

[array([[1],
        [3],
        [5]]), array([[2],
        [4],
        [6]]), array([], shape=(3,0), dtype=int32)]
```

Numpy Arrays Function
(Search, Sort, Search Sorter, Filter)

Search Array: Search an array for a certain value and return the indexes that get a match

```
import numpy as np
Var = np.array([1,2,3,4,2,5,2,5,6,7])
                0,1,2,3,4,5,6,7,8,9
#index
x = np.where(Var == 2)
Print(x)
```

output: (array[1, 4, 6], dtype = int64).) :

```
Var1 = np.array([4,2,3,4,2,5,2,5,6,7])
x = np.where(Var1 > 2 == 0)
Print(x)
```

output: (array[0, 1, 3, 4, 6, 8], dtype int64)

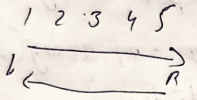
If we use (Var1 > 2) 0

If give array[]
because it give
reminder

Search Sorted Array: Which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.

```
Var2 = np.array([1,2,3,4,6,7,8])
x1 = np.searchsorted(var2, 5)
Print(x1)
```

Output : 4 index value.




```
Var3 = np.array([1,2,3,4,8,9,10])
x2 = np.searchsorted(var3, [5,6,7], side = "right")
Print(x2)
```

Output : [4 4 4]

Sort Array : Ordered Sequence is any sequence that has an order corresponding to elements, like numeric or Alphabetical ascending or descending.

```
Var1 = np.array([1,2,3,22,45,6,56,8,9])
Print(np.sort(var1))
```

Output: [1 2 3 6 8 9 22 45 56]

2D

```
Var2 => np.array([[4,2,3],[1,12,15],[22,52,63]])
Print(np.sort(var2))
```

Output : [[2 3 4]
[1 5 17]
[6 22 52]]

```
Var3 = np.array(["o","s","d","f"])
Print(np.sort(var3))
```

Output: ['o' 'd' 'f' 's']

Filter Array : Getting some elements out of an existing array and creating a new array out of them.

```
Var_3 = np.array(["a","s","d","f"])
f = [True, False, False, True]
new_a = var_3[f]
Print(new_a)
Print(type(new_a))
```

Output: ['a' 'f']
<class 'numpy.ndarray'>

Numpy Function

Arithmetic Function

=> Shuffle: import numpy as np

```
var = np.array([1,2,3,4,5])
np.random.shuffle(var)
print(var)
```

output: [2 1 4 5 3]

when we run it again it gave new shuffle array.

=> Unique: var1 = np.array([1,2,3,4,2,5,2,6,2,7])

```
x = np.unique(var1, return_index=True, return_counts=True)
print(x)
```

output: (array([1,2,3,4,5,6,7]), array([0,1,2,3,5,7,9]), array([1,4,1,1,1,1,1]))

=> Resize: var2 = np.array([1,2,3,4,5,6])

```
x = np.resize(var2, (3,2))
print(x)
```

output:
[[1 2]
 [3 4]
 [5 6]]

(2,3)
[[1 2 3]
 [4 5 6]]

=> Flatten: To convert 2D Array into 1D Array. We also do it with the help of conversion order.

order: {'C', 'F', 'A', 'K'}, optional

- 'C' means to flatten in row-major (C-style) order.
- 'F' means to flatten in column-major (Fortran-style) order.
- 'A' means to flatten in column-major order if 'a' is Fortran *contiguous* in memory, row-major order otherwise.
- 'K' means to flatten 'a' in the order the element occur in memory.
- The default is 'C'.

```
var2 = np.array([1,2,3,4,5,6])
y = np.resize(var2, (3,2))
print(y)
print()
print("Flatten: ", y.flatten(order="F"))
print("Ravel: ", np.ravel(y, order="F"))
```

output:
[[1 2]
 [3 4]
 [5 6]]

Flatten: [1 3 5 2 4 6]

Ravel: [1 2 3 4 5 6]

: (Insert and delete Function)

Insert : import numpy as np

```
var = np.array([1,2,3,4])
print(var)
v = np.insert(var, 2, 40)
                   Index number
```

Output: [1 2 3 4]
[1 2 40 3 4]

We can also insert multiple

```
v = np.insert(var, (2,4), 40)
```

Output = [1 2 40 3 4 40]

Note: It insert only integer value 2, 4, 6 not float 5.4, 6.4

2D

```
var1 = np.array([[1,2,3],[1,2,3]])
v1 = np.insert(var1, 2, 6, axis=0)
print(v1)
```

Output: [[1 2 3]
[1 2 3]
[6 6 6]]

position
axis=1
[[1 2 6 3
1 2 6 3]]

For multiple data

```
v1 = np.insert(var1, 2, [22,23], axis=1)
```

Output: [[1 2 22 3]
[1 2 23 3]]

axis=0
[22, 23, 24]

[[1 2 3]
[1 2 3]
[22 23 24]]

We can also use append function

-> Delet :

```
var2 = np.array([1, 2, 3, 4])
d = np.delete(var2, 2)
print(d)
```

output : [1 2 4]

Matrix in Numpy Arrays

Matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

3×3

→

Array

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

3×3

.dat

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 1\times1+2\times3 & 1\times2+2\times4 \\ 3\times1+3\times4 & 3\times2+4\times4 \end{bmatrix}$$

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 5 \\ 6 & 7 \end{bmatrix} \begin{bmatrix} 3 & 2 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 6 & 7 \end{bmatrix}$$

```
import numpy as np
Var = np.matrix([1,2,3],[1,2,3])
Print(Var)
Print(type(Var))
```

Output : [[1 2 3]
[1 2 3]]
<class 'numpy.matrix'

```
Var1 = np.array([1,2,3],[1,2,3])
Print(Var1)
Print(type(Var1))
```

Output: [[1 2 3]
[1 2 3]]
<class 'numpy.ndarray'>

=> Dot Function

```
Var = np.matrix([[1,2],[1,2]])
Var1 = np.matrix([[1,2],[1,2]])
Print(Var.dot(Var1))
```

Output [[3 6]
[3 6]]

Matrix Function in Numpy Arrays

=> Transpose :

```
import numpy as np
Var = np.array([[1,2,3],[3,4,5]])
Print(Var)
Print()
Print(np.transpose(Var))
Print()
Print(Var.T)
```

Output: [[1 2 3]
[3 4 5]]

=>[[1 3]
[2 4]
[3 5]]

Var.T
- [[1 3]
[2 4]
[3 5]]

=> Swapaxes : It is also work as transpose

```
Var2 = np.matrix([[1,2],[3,4]])
Print(Var2)
Print(np.swapaxes(Var2,0,1))
```

Output: [[1 2]
[3 4]]

[[1 3]
[2 4]]

=> Inverse :

$A^{-1} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \frac{1}{-2} \begin{bmatrix} -4 & -2 \\ -3 & 1 \end{bmatrix}$

```
Var3 = np.array([[1,2],[3,4]])
Print(Var3)
Print(np.linalg.inv(Var3))
```

Output: `[[1 2]`
`[3 4]]`

`[[-2 1]`
`[1.5 -0.5]]`

=> Power: $A^2 = A \cdot A$ dot function

```
np.linalg.matrix_power(Var, n)
```

n<0 n=0 n>0

n=0 I

n>0 power(Mult)

n<0 Inverse * power

```
Var4 = np.matrix([[1,2],[3,4]])
Print(Var4)
Print()
Print(np.linalg.matrix_power(Var4,2))
                            (Var4,0))
                            (Var4,-2))
```

Output: `[[1 2]`
`[3 4]]`

`[[7 10]`
`[15 22]]`

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

=> Determinate

$A = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$ $|A| = a[ei - hf] - b[di - gf] + c[dh - g$

```
Var5 = np.array([[1,2],[3,4]])
Print(Var5)
Print()
Print(np.linalg.det(Var5))
```

Output: `[[1 2]`
`[3 4]]`

-2.0000

=> `[[-3.5 -2.5]`
`[-3.75 1.75]]`

=> np.intersect1d(var, var1)

is used to find common element in array.

Prob Sort 2D Array by the second column

```
import numpy as np
arr = np.array([[3,2,7],[9,5,7],[5,1,9]])
Print(arr)
Sorted_arr = arr[arr[:,1].argsort()]
Print(Sorted_arr)
```

$$\begin{bmatrix} 3 & 2 & 7 \\ 9 & 5 & 7 \\ 5 & 1 & 9 \end{bmatrix} = \begin{bmatrix} 5 & 1 & 9 \\ 3 & 2 & 7 \\ 9 & 5 & 7 \end{bmatrix}$$

Column: 0 1 2

Row 0, Row 1, Row 2

arr[:,1) : => all row

(Row, Column) 1 = Colum ~~second~~ at index 1

.argsort() => Short the index