# numpy-24th-aprl-1

May 7, 2024

## 1 Introduction to NUMPY

- NumPy is a Python library used for scientific computing and data analysis. It provides support for arrays, matrices, and mathematical functions, making it useful for numerical computations such as linear algebra, Fourier transforms, and random number generation.
- NumPy is used in a wide range of fields, including scientific computing, data science, machine learning, and artificial intelligence. Some popular applications of NumPy include image and audio processing, statistical analysis, and simulations.

#Here are some of the major operations that NumPy offers:

- Creating arrays: NumPy provides a number of ways to create arrays, including from lists or tuples using numpy.array(), using functions like numpy.zeros() or numpy.ones() to create arrays of a specific shape filled with zeros or ones, or generating random arrays using numpy.random.rand() or numpy.random.randn().
- Indexing and slicing arrays: NumPy allows you to select specific elements or sections of an array using indexing and slicing. For example, you can select the first two elements of an array using my_array[0:2].
- Mathematical operations: NumPy provides a wide range of mathematical functions for working with arrays, including arithmetic operations like addition, subtraction, and multiplication, as well as functions like numpy.sin() and numpy.cos() for trigonometric calculations.
- Linear algebra operations: NumPy provides a number of linear algebra functions, including matrix multiplication, matrix inversion, and eigenvalue and eigenvector calculations.

## 2 Why are they important in data analysis?

NumPy is important for data analysis because it provides fast and efficient array operations, making it a powerful tool for manipulating large datasets. Here are some methods and functions associated with NumPy that demonstrate its usefulness for data analysis:

[ ]:
```
# Import numpy package
import numpy as np
```

## 3 How to create array?

- 1D array

- 2D array

[ ]:
```
# create an array to store single element 10
arr = np.array(10)
print(arr)
print(type(arr))
print(arr.ndim)
```

```
10
<class 'numpy.ndarray'>
0
```

[ ]:
```
# To create 1-D array we can use either list or tuple
A = [1,2,3,4,5]
arr_1=np.array(A)
print(arr_1)
print(type(arr_1))
print(arr_1.ndim)
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
1
```

[ ]:
```
# create array using tuple
B = tuple(range(1,11))
print(B)
arr_2 = np.array(B)
print(arr_2)
print(type(arr_2))
print(arr_2.ndim)
```

```
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
[ 1  2  3  4  5  6  7  8  9 10]
<class 'numpy.ndarray'>
1
```

[ ]:
```
# 2d arrays: Matrices which will have rows and columns
# 2d array are created using nested list or nested tuple
A = [[1,2],[3,4]] #2d array
arr_2d = np.array(A)
print(arr_2d)
print(type(arr_2d))
print(arr_2d.ndim)
```

```
[[1 2]
 [3 4]]
<class 'numpy.ndarray'>
2
```

[ ]:
```
B = [[2,3,7],[4,5,6]] # 2d array
arr2=np.array(B)
print(arr2)
print(type(arr2))
print(arr2.ndim)
```

```
[[2 3 7]
 [4 5 6]]
<class 'numpy.ndarray'>
2
```

## 4 Basic functions

- size : It is used to find number of elements in a array
- dtype: To find data type of elements
- type: To find type of object
- shape : It is used to find number of rows and columns
- ndim : Dimension of an array

[ ]:
```
arr = np.array([1,2,3,4,5])
print(arr)
print("data type",arr.dtype)
print("type",type(arr))
print("no of elements",arr.size)
print("dimension" , arr.ndim)
print("Shape",arr.shape)
```

```
[1 2 3 4 5]
data type int64
type <class 'numpy.ndarray'>
no of elements 5
dimension 1
Shape (5,)
```

[ ]:
```
arr_2d= np.array([[1,2],[3,4]])
print(arr_2d)
print("data type",arr_2d.dtype)
print("type",type(arr_2d))
print("no of elements",arr_2d.size)
print("dimension" , arr_2d.ndim)
print("Shape",arr_2d.shape)
```

```
[[1 2]
 [3 4]]
data type int64
type <class 'numpy.ndarray'>
no of elements 4
```

```
dimension 2
Shape (2, 2)
```

## 5 Basic math functions

- sum(): Sum of array elements.
- mean(): Mean of array elements.
- median(): Median of array elements.
- std(): Standard deviation of array elements.
- min(): Minimum value in an array.
- max(): Maximum value in an array.
- linalg.inv(): Inverse of a matrix.
- linalg.det(): Determinant of a matrix.
- linalg.eig(): Eigenvalues and eigenvectors of a matrix.

[ ]:
```
# 1d array
arr = np.array([10,20,60,80,100,700,4])
print("sum of array elements",np.sum(arr))
print("minimum element in array",np.min(arr))
print("maximum element in array",np.max(arr))
print("Number of elements",len(arr))
print("Average of array",np.mean(arr))
print("Median",np.median(arr))
print("standard deviation",np.std(arr))
```

```
sum of array elements 974
minimum element in array 4
maximum element in array 700
Number of elements 7
Average of array 139.14285714285714
Median 60.0
standard deviation 231.45220337896893
```

[ ]:
```
# 2d array
arr_2d=np.array([[2,30],[4,90]])
print(arr_2d)
print("sum of array elements",np.sum(arr_2d))
print("minimum element in array",np.min(arr_2d))
print("maximum element in array",np.max(arr_2d))
print("Number of elements",arr_2d.size) # size for 2d
print("Average of array",np.mean(arr_2d))
print("Median",np.median(arr_2d))
print("standard deviation",np.std(arr_2d))
print("determinant",np.linalg.det(arr_2d))
print("Inverse",np.linalg.inv(arr_2d))
print("Eigen values",np.linalg.eig(arr_2d))
```

```
[[ 2 30]
```

```
 [ 4 90]]
sum of array elements 126
minimum element in array 2
maximum element in array 90
Number of elements 4
Average of array 31.5
Median 17.0
standard deviation 35.53519382246282
determinant 59.999999999999986
Inverse [[ 1.5        -0.5       ]
 [-0.06666667  0.03333333]]
Eigen values EigResult(eigenvalues=array([ 0.6568638, 91.3431362]),
eigenvectors=array([[-0.99899927, -0.31831791],
       [ 0.0447264 , -0.94798402]]))
```

[ ]:

## 6 Functions to generate array with different elements

- arange(): This function is used to create an array with evenly spaced values within a specified range. It takes three arguments: start, stop, and step.
- linspace(): This function is used to create an array with a specified number of evenly spaced values within a specified range. It takes three arguments: start, stop, and num.
- random.rand(): Create an array of random values (uniform distribution).
- random.rand(): Create an array of random values (uniform distribution).
- random.randint(): create a array with integer values
- reshape(): This function is used to change the shape of an array. It takes a tuple as input that specifies the new shape of the array.
- transpose(): This function is used to transpose an array, which means swapping its rows and columns.
- flatten(): In numpy, flatten() is a method that can be used to convert a multi-dimensional array into a one-dimensional array. It returns a copy of the original array, with all of its dimensions flattened into a single dimension.
- sort(): Sort an array.
- concatenate(): Concatenate arrays.

## 7 arange():

This function is used to create an array with evenly spaced values within a specified range. It takes three arguments: * np.arange(start, stop,step): stop is exclusive

[ ]:
```
# create an array with 20 elements 1 to 20
arr = np.arange(1,21)
arr
```

[ ]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20])
```

[ ]:
```
# create an array with first 10 even numbers
arr = np.arange(2,21,2)
arr
```

[ ]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

[ ]:
```
# create an array with first 10 numbers in reverse order
arr1=np.arange(10,0,-1)
arr1
```

[ ]:
```
array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1])
```

[ ]:
```
# create an array with first 10 multiples of 5 in reverse direction
np.arange(50,0,-5)
```

[ ]:
```
array([50, 45, 40, 35, 30, 25, 20, 15, 10,  5])
```

[ ]:
```
# create 2-d array of order 2,2 using using arange function
arr = np.arange(1,5).reshape(2,2)
arr
```

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

[ ]:
```
arr=np.arange(1,5)
print(arr)
arr_2d=arr.reshape(2,2)
print(arr_2d)
```

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

[ ]:
```
# create an 2d array of order 4,5
# no of elemts =4*5=20
arr= np.arange(1,21).reshape(4,5)
print(arr)
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]]
```

[ ]:
```
# create 2d array of any order using given elements
arr = np.arange(1,16).reshape(3,5)
print(arr)
'''reshape(): This function is used to change the shape of an array.
```

```
It takes a tuple as input that specifies the new shape of the array. '''
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]]
```

## 8 linspace():

This function is used to create an array with a specified number of evenly spaced values within a specified range. It takes three arguments: start, stop, and num.

[ ]:
```
np.arange(1,3)
```

[ ]:
```
array([1, 2])
```

[ ]:
```
np.linspace(1,3)
```

[ ]:
```
array([1.        , 1.04081633, 1.08163265, 1.12244898, 1.16326531,
       1.20408163, 1.24489796, 1.28571429, 1.32653061, 1.36734694,
       1.40816327, 1.44897959, 1.48979592, 1.53061224, 1.57142857,
       1.6122449 , 1.65306122, 1.69387755, 1.73469388, 1.7755102 ,
       1.81632653, 1.85714286, 1.89795918, 1.93877551, 1.97959184,
       2.02040816, 2.06122449, 2.10204082, 2.14285714, 2.18367347,
       2.2244898 , 2.26530612, 2.30612245, 2.34693878, 2.3877551 ,
       2.42857143, 2.46938776, 2.51020408, 2.55102041, 2.59183673,
       2.63265306, 2.67346939, 2.71428571, 2.75510204, 2.79591837,
       2.83673469, 2.87755102, 2.91836735, 2.95918367, 3.        ])
```

[ ]:
```
np.linspace(1,10,5)
# It is generating 5 evenly spaced value between 1 to 100b
```

[ ]:
```
array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ])
```

[ ]:
```
# Generate 10 evenly spaced values between 1 to 100
np.linspace(1,100,10,retstep=True)
# retstep: It give equal space/difference
```

[ ]:
```
(array([  1.,  12.,  23.,  34.,  45.,  56.,  67.,  78.,  89., 100.]), 11.0)
```

[ ]:
```
#. gernerate 5 linearly spaced numbers between 1to 1000 and find equal spacing
np.linspace(1,1000,5,retstep=True)
```

[ ]:
```
(array([   1.  ,  250.75,  500.5 ,  750.25, 1000.  ]), 249.75)
```

[ ]:
```
'''generate 2d array using elements generated by linspace which
includes 4 numbers between 1 to 100'''
np.linspace(1,100,4).reshape(2,2)
```

```
array([[  1.,  34.],
       [ 67., 100.]])
```

```
# np.linspace(start , end(include), num_of_elements , space{retstep=True})
```

## 9 Random functions

- random.rand(): Create an array of random values (uniform distribution).
- random.rand(): Create an array of random values (uniform distribution).
- random.randint(): create a array with integer values

```
# rand : Uniformly distribution--> It will generate random values within [0,1]
np.random.rand(5)
```

```
array([0.76725642, 0.52391426, 0.20506297, 0.93452754, 0.72092201])
```

```
np.random.rand(10)
```

```
array([0.64597089, 0.89298437, 0.40573882, 0.70522476, 0.05306132,
       0.92061216, 0.35263085, 0.05124318, 0.39811944, 0.71861968])
```

```
# generate 3 * 3 array using rand function
np.random.rand(9).reshape(3,3)
```

```
array([[0.94449088, 0.41650809, 0.09958857],
       [0.61554606, 0.70409841, 0.34837335],
       [0.9815293 , 0.19580832, 0.27023569]])
```

```
# generate 4 * 3 array using rand function
np.random.rand(12).reshape(4,3)
```

```
array([[0.14323958, 0.66578384, 0.34979421],
       [0.57080194, 0.98762029, 0.34552743],
       [0.52227931, 0.0420611 , 0.32701006],
       [0.18635825, 0.61487264, 0.50453298]])
```

```
# randn: It is used to generate normally distributed values(-3 , 3)
np.random.randn(10)
```

```
array([ 1.98542814,  0.99505165,  0.84932951, -0.229731  ,  0.91909753,
       -1.43176462, -0.43735771, -0.44334447, -0.05840733, -0.18661495])
```

```
# generate 20 normally distributed values
np.random.randn(20)
```

```
array([-0.47579272,  1.10910574, -0.21145247, -0.68494896,  1.45285667,
        0.44042578, -0.32040901,  1.21469428,  1.0995975 , -0.61877837,
```

```
 0.85108408,  0.36906061, -0.15739013,  0.42606046, -0.11611182,
-1.14824983,  0.29661245,  0.73203162, -0.0230496 , -0.8513491 ])
```

[ ]:
```
# randint: It is used generate random integers
# generate 1 random integer between 1 to 10
np.random.randint(1,10)
```

[ ]:
```
2
```

[ ]:
```
# generate 10 random integers between 1 to 100
np.random.randint(1,100,10)
```

[ ]:
```
array([14, 91,  3, 40, 84, 21, 46, 91, 46, 49])
```

[ ]:
```
# generate 5 random integers between 1 to 1000
np.random.randint(1,1000,5)
```

[ ]:
```
array([930, 570, 493, 939, 730])
```

#transpose(): This function is used to transpose an array, which means swapping its rows and columns.

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

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

[ ]:
```
np.transpose(arr)
```

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

## 10 flatten():

In numpy, flatten() is a method that can be used to convert a multi-dimensional array into a one-dimensional array. It returns a copy of the original array, with all of its dimensions flattened into a single dimension.

[ ]:
```
arr = np.array([[1,2],[3,4]])
print(arr)
arr.flatten()
```

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

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

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

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

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

## 11 sort():

Sort an array.

[ ]:
```
arr= np.random.randint(1,100,10)
print(arr)
np.sort(arr)# increasing order/ascending
```

```
[73 95  4 20 13 13 66 53 24 76]
```

[ ]:
```
array([ 4, 13, 13, 20, 24, 53, 66, 73, 76, 95])
```

[ ]:
```
np.sort(arr)[::-1]# descending
```

[ ]:
```
array([95, 76, 73, 66, 53, 24, 20, 13, 13,  4])
```

[ ]:
```
arr1 = np.array([[300,3,20],[10,2,40],[300,4,1000]])
arr1
```

[ ]:
```
array([[ 300,    3,   20],
       [  10,    2,   40],
       [ 300,    4, 1000]])
```

[ ]:
```
np.sort(arr1) #ascending: sorting rowwise
```

[ ]:
```
array([[   3,   20,  300],
       [   2,   10,   40],
       [   4,  300, 1000]])
```

[ ]:
```
np.sort(arr1,axis=0) # ascending : sorting column wise
```

[ ]:
```
array([[  10,    2,   20],
       [ 300,    3,   40],
       [ 300,    4, 1000]])
```

[ ]:
```
arr1 = np.array([[300,3,20],[10,2,40],[300,4,1000]])
arr1
```

[ ]:
```
array([[ 300,    3,   20],
       [  10,    2,   40],
       [ 300,    4, 1000]])
```

[ ]:
```
# sort array in 1D
np.sort(arr1,axis=None)
# axis=None, It will sort 2d array elements in ascending order
```

[ ]:
```
array([   2,    3,    4,   10,   20,   40,  300,  300, 1000])
```

[ ]:
```
np.sort(arr1,axis=None)[::-1]
```

[ ]:
```
array([1000,  300,  300,   40,   20,   10,    4,    3,    2])
```

[ ]:
```
arr1
```

[ ]:
```
array([[ 300,    3,   20],
       [  10,    2,   40],
       [ 300,    4, 1000]])
```

[ ]:
```
# sort array row wise in ascending order (rowwise)
np.sort(arr1,axis=1) # axis=1 ,row wise sorting
```

[ ]:
```
array([[   3,   20,  300],
       [   2,   10,   40],
       [   4,  300, 1000]])
```

[ ]:
```
np.sort(arr1,axis=1)[:,::-1] # rowwise sorting in descending order
```

[ ]:
```
array([[ 300,   20,    3],
       [  40,   10,    2],
       [1000,  300,    4]])
```

[ ]:
```
# sort array column wise in ascending and descending
```

[ ]:
```
arr1
```

[ ]:
```
array([[ 300,    3,   20],
       [  10,    2,   40],
       [ 300,    4, 1000]])
```

[ ]:
```
np.sort(arr1,axis=0)# column wise sorting in ascending order
```

[ ]:
```
array([[  10,    2,   20],
       [ 300,    3,   40],
       [ 300,    4, 1000]])
```

[ ]:
```
# column wise sorting in descending order
np.sort(arr1,axis=0)[::-1] # [::-1]: It is to reverse column
                           # [:,::-1] : It is to revers erow
```

[ ]:
```
array([[ 300,    4, 1000],
       [ 300,    3,   40],
       [  10,    2,   20]])
```

[ ]:
```
# rowise sorting in ascending=np.sort(arr,axis=1)
# rowwise sorting in descending = np.sort(arr,axis=1)[:,::-1]
#[:,::-1]: It reverses each row
# column wise sorting in ascending = np.sort(arr,axis=0)
# column wise sorting in descending = np.sort(arr,axis=0)[::-1]
# [::-1]: It is used to reverse each column
```

## 12 concatenate():

Concatenate arrays.

[ ]:
```
arr1 = np.array([1,2,3,4])
arr2=np.array([10,20,30,40])
new_arr=np.concatenate((arr1,arr2)) # two brackets
new_arr
```

[ ]:
```
array([ 1,  2,  3,  4, 10, 20, 30, 40])
```

## 13 Indexing and Slicing

- arr[row , col]

[ ]:
```
arr= np.array([1,2,3,4])
arr[::-1]
```

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

[ ]:
```
arr= np.array([[3,2,1],[1,5,6],[7,8,10]])
arr
```

[ ]:
```
array([[ 3,  2,  1],
       [ 1,  5,  6],
       [ 7,  8, 10]])
```

[ ]:
```
np.sort(arr,axis=0)
```

[ ]:
```
array([[ 1,  2,  1],
       [ 3,  5,  6],
```

```
       [ 7,  8, 10]])
```

[ ]:
```
np.sort(arr,axis=0)[::-1]
```

[ ]:
```
array([[ 7,  8, 10],
       [ 3,  5,  6],
       [ 1,  2,  1]])
```

## 14 Indexing and Slicing

[ ]:
```
arr = np.array([100,200,300,400,500,600])
arr
```

[ ]:
```
array([100, 200, 300, 400, 500, 600])
```

[ ]:
```
# fetch 400
arr[3]
```

[ ]:
```
400
```

[ ]:
```
arr[-3]
```

[ ]:
```
400
```

[ ]:
```
# fetch 300,400,500
arr[2:5]
```

[ ]:
```
array([300, 400, 500])
```

## 15 Indexing and Slicing for 2d arrays

[ ]:
```
arr = np.array([[10,20,30],[40,50,60],[70,80,90]])
arr
```

[ ]:
```
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])
```

[ ]:
```
# Indexing: fetching single elements from a given array
# fetch 10
arr[0,0]
```

[ ]:
```
10
```

[ ]:
```
#fetch 90
arr[2,2]
```

[ ]:
```
90
```

[ ]:
```
# fetch 80
arr[2,1]
```

[ ]:
```
80
```

[ ]:
```
# fetch 40
arr[1,0]
```

[ ]:
```
40
```

[ ]:
```
# fetch first rows
arr[0]
```

[ ]:
```
array([10, 20, 30])
```

[ ]:
```
# fetch third row
arr[2]
```

[ ]:
```
array([70, 80, 90])
```

[ ]:
```
# fetch first column: arr[: , column_index]
arr[: ,[0]]
```

[ ]:
```
array([[10],
       [40],
       [70]])
```

[ ]:
```
# fetch third column
arr[:,[2]]
```

[ ]:
```
array([[30],
       [60],
       [90]])
```

[ ]:
```
# fetch first row : arr[row_index,:]
arr[0,:]
```

[ ]:
```
array([10, 20, 30])
```

[ ]:
```
A = np.array([[10,20,30,40],[50,60,70,80],[5,7,8,9],[1,2,3,4]])
A
```

[ ]:
```
array([[10, 20, 30, 40],
       [50, 60, 70, 80],
       [ 5,  7,  8,  9],
       [ 1,  2,  3,  4]])
```

[ ]:
```
# fetch 8
A[2,2]
```

[ ]:
```
8
```

[ ]:
```
# fetch 70
A[1,2]
```

[ ]:
```
70
```

[ ]:
```
# fetch 3
A[3,2]
```

[ ]:
```
3
```

[ ]:
```
# fetch second row
A[1] #A[1,:]
```

[ ]:
```
array([50, 60, 70, 80])
```

[ ]:
```
# fetch fourth row
A[3] # A[3,:]
```

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

[ ]:
```
# fetch second column
A[:,[1]]
```

[ ]:
```
array([[20],
       [60],
       [ 7],
       [ 2]])
```

[ ]:
```
# fetch 4th column
A[:,[3]]
```

[ ]:
```
array([[40],
       [80],
       [ 9],
       [ 4]])
```

[ ]:
```
# fetch 1st and 2nd row
A[0:2]
```

[ ]:
```
array([[10, 20, 30, 40],
       [50, 60, 70, 80]])
```

[ ]:
```
# fetch 1st and 4th row
A[[0,3]] # A[0:5:3]
```

[ ]:
```
array([[10, 20, 30, 40],
       [ 1,  2,  3,  4]])
```

[ ]:
```
A[0:5:3]
```

[ ]:
```
array([[10, 20, 30, 40],
       [ 1,  2,  3,  4]])
```

[ ]:
```
# fetch first and third row using step
A[0:3:2]
```

[ ]:
```
array([[10, 20, 30, 40],
       [ 5,  7,  8,  9]])
```

[ ]:
```
# fetch first and third row without step
A[[0,2]]
```

[ ]:
```
array([[10, 20, 30, 40],
       [ 5,  7,  8,  9]])
```

[ ]:
```
# fetch 1st and 2nd column (slicing)
A[: ,0:2]
```

[ ]:
```
array([[10, 20],
       [50, 60],
       [ 5,  7],
       [ 1,  2]])
```

## 16 Indexing and slicing

[ ]:
```
import numpy as np
arr = np.array([[1,2,3,4,1],[5,6,7,8,2],[9,10,11,12,3],[13,14,15,16,4]])
arr
```

[ ]:
```
array([[ 1,  2,  3,  4,  1],
       [ 5,  6,  7,  8,  2],
       [ 9, 10, 11, 12,  3],
```

```
       [13, 14, 15, 16,  4]])
```

[ ]:
```
# fetch 11
arr[2,2]
```

[ ]:
```
11
```

[ ]:
```
# fetch 8
arr[ 1,3 ]
```

[ ]:
```
8
```

[ ]:
```
# fetch first row
arr[0]
```

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

[ ]:
```
# fetch 4 th row
arr[3]
```

[ ]:
```
array([13, 14, 15, 16,  4])
```

[ ]:
```
# fetch 1st and 2nd row
arr[0:2]
```

[ ]:
```
array([[1, 2, 3, 4, 1],
       [5, 6, 7, 8, 2]])
```

[ ]:
```
# fetch 2nd 3rd and 4th row
arr[1:4]
```

[ ]:
```
array([[ 5,  6,  7,  8,  2],
       [ 9, 10, 11, 12,  3],
       [13, 14, 15, 16,  4]])
```

[ ]:
```
# fetch 1 st , 3rd and 4th row
arr[[0,2,3]]
```

[ ]:
```
array([[ 1,  2,  3,  4,  1],
       [ 9, 10, 11, 12,  3],
       [13, 14, 15, 16,  4]])
```

[ ]:
```
# fetch 1st ,2nd row and 4th row
arr[[0,1,3]]
```

[ ]:
```
array([[ 1,  2,  3,  4,  1],
       [ 5,  6,  7,  8,  2],
```

```
       [13, 14, 15, 16,  4]])
```

[ ]:
```
# fetch 1row and 3rd row
arr[[0,2]]
```

[ ]:
```
array([[ 1,  2,  3,  4,  1],
       [ 9, 10, 11, 12,  3]])
```

[ ]:
```
# fetch 1st column:arr[:,column_index]
arr[:,0]
```

[ ]:
```
array([ 1,  5,  9, 13])
```

[ ]:
```
arr[:,[0]]
```

[ ]:
```
array([[ 1],
       [ 5],
       [ 9],
       [13]])
```

[ ]:
```
# fetch 4th column
arr[: ,[3]]
```

[ ]:
```
array([[ 4],
       [ 8],
       [12],
       [16]])
```

[ ]:
```
# fetch 2nd column
arr[:,[1]]
```

[ ]:
```
array([[ 2],
       [ 6],
       [10],
       [14]])
```

[ ]:
```
# fetch 1st,2nd and 3rd column
arr[: ,0:3]
```

[ ]:
```
array([[ 1,  2,  3],
       [ 5,  6,  7],
       [ 9, 10, 11],
       [13, 14, 15]])
```

[ ]:
```
# fetch 2nd , 3rd ,4th column
arr[:,1:4]
```

[ ]:
```
array([[ 2,  3,  4],
       [ 6,  7,  8],
       [10, 11, 12],
       [14, 15, 16]])
```

[ ]:
```
# fetch 1st column , 3rd column , 4th column
arr[:,[0,2,3]]
```

[ ]:
```
array([[ 1,  3,  4],
       [ 5,  7,  8],
       [ 9, 11, 12],
       [13, 15, 16]])
```

[ ]:
```
# fetch 1st column , 2nd column , 3rd column and 5th
arr[:,[0,1,2,4]]
```

[ ]:
```
array([[ 1,  2,  3,  1],
       [ 5,  6,  7,  2],
       [ 9, 10, 11,  3],
       [13, 14, 15,  4]])
```

[ ]:
```
arr[0:2,0:2]
```

[ ]:
```
array([[1, 2],
       [5, 6]])
```

[ ]:
```
# fetch 7 ,8,11,12
arr[1:3,2:4]
```

[ ]:
```
array([[ 7,  8],
       [11, 12]])
```

[ ]:
```
arr[1:4,2:5]
```

[ ]:
```
array([[ 7,  8,  2],
       [11, 12,  3],
       [15, 16,  4]])
```

[ ]:
```
arr[0:3,1:4]
```

[ ]:
```
array([[ 2,  3,  4],
       [ 6,  7,  8],
       [10, 11, 12]])
```

[ ]:
```

```