Chapter 5
Numeric Computing with Numpy

Chapter Content

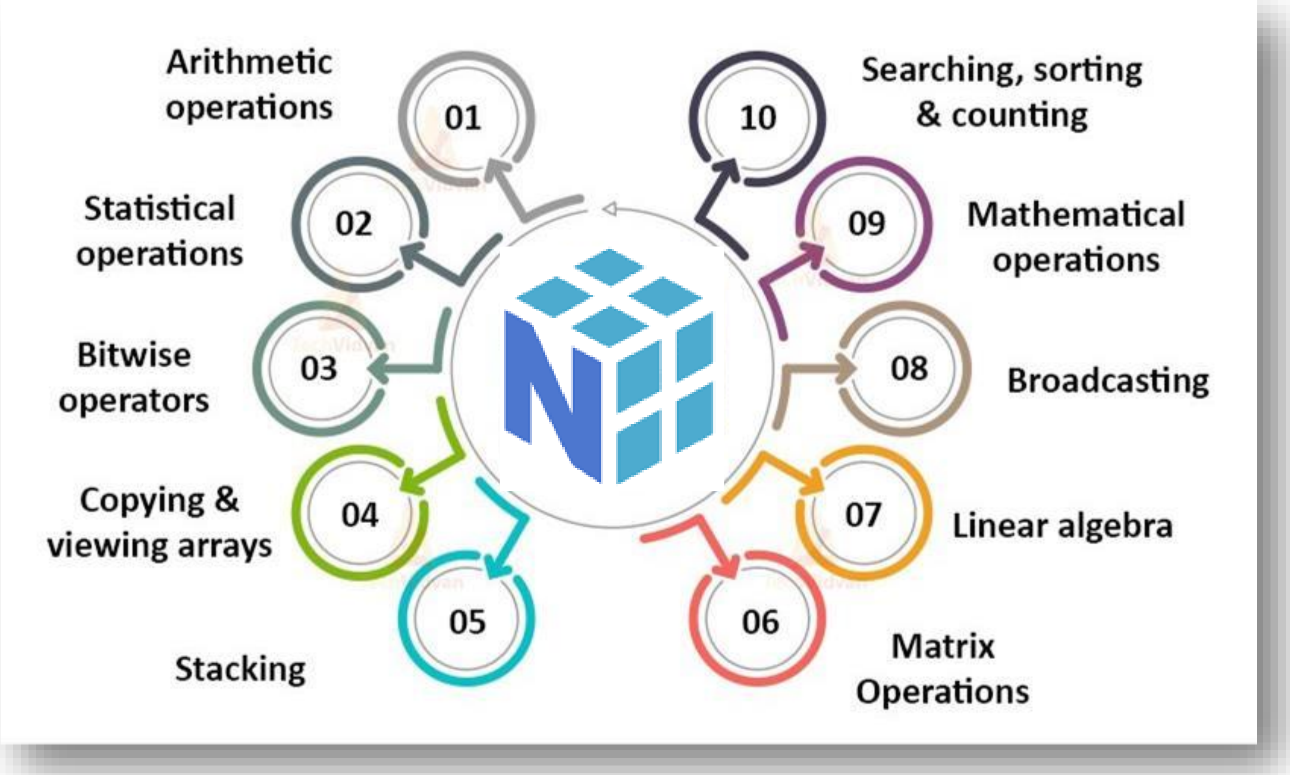
- 1. Introduction to NumPy
- 2. Numpy Data Types
- 3. Numpy Getting Started
- 4. Numpy Array
- 5. Numpy Linear Algebra
- 6. Numpy Matrix Library - **Matlib**
- 7. Input/Output with Numpy

1. Introduction to NumPy

- In 2005, Travis Oliphant created NumPy
- Travis Oliphant is an American data scientist and businessman
- NumPy (Numerical Python) is an open source library in Python used to work with array data structures and related fast numerical routines
- Array oriented computing, Efficiently implemented multi-dimensional arrays

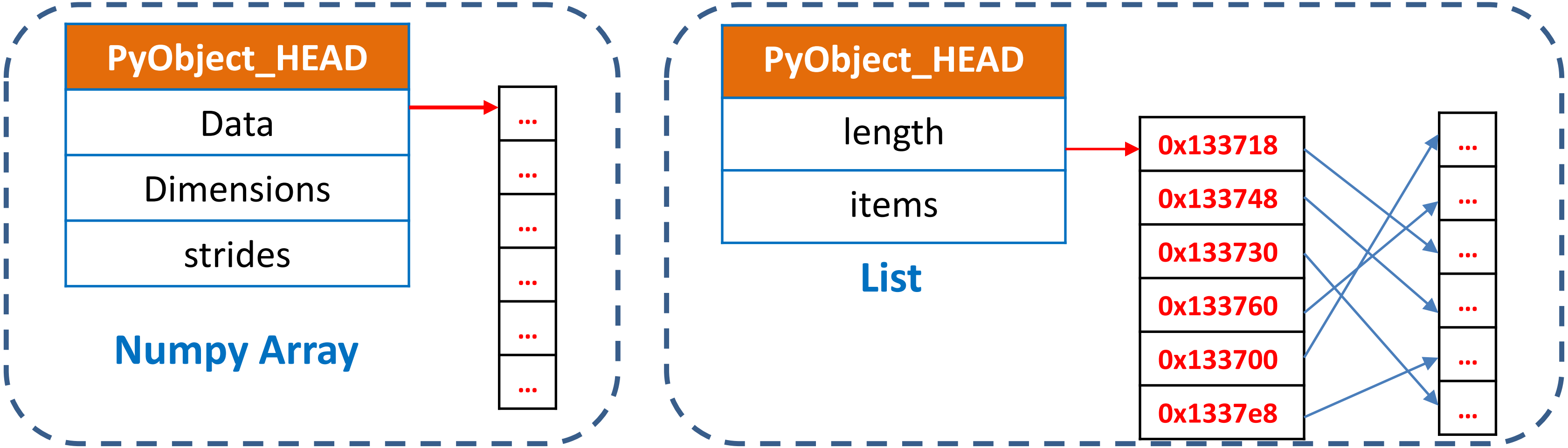
1. Introduction to NumPy

- NumPy is relied upon by scientists, engineers, and many other professionals around the world



1. Introduction to NumPy

- Convenient interface for working with multi-dimensional array data structures efficiently (**ndarray** object).
- Using fixed memory to store data (same data type) and less memory than Lists (different data type) .
- Allocate contiguous memory
- High computational efficiency



2. Numpy Data Types

- Supports a much greater variety of numerical types than Python does

Data Types	Keywords/Sub Data Types
Boolean	`bool_`
Integer	`int_, intc, intp, int8, int16, int32, int64`
Unsigned Integer	`uint8, uint16, uint32, uint64`
Float	`float_, float 16, float32, float64`
Complex	`complex_, complex64, complex128`

3. Numpy Getting Started

- Installing Numpy: `pip install numpy`
- Import Numpy: `import numpy`
- Alias of Numpy: `import numpy as np`
- Check Numpy version: `np.__version__`

4. Numpy Array

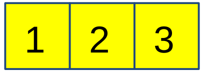
- **ndarray** object (N-Dimensional array)

0-D Array

1

```
np.array(1)
```

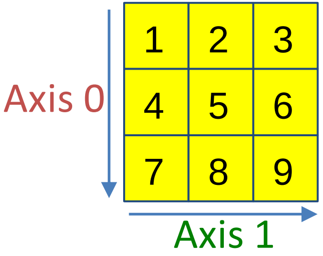
1-D Array → Vector



1	2	3

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

2-D Array → Matrix



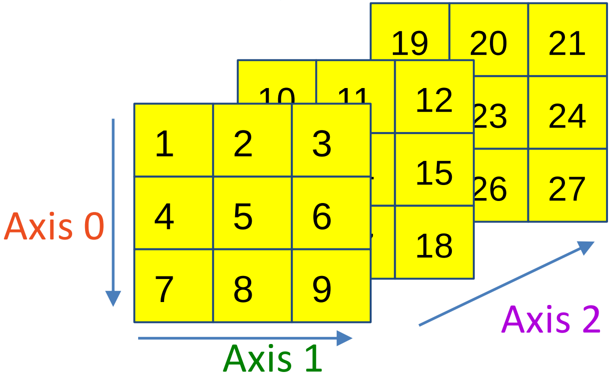
1	2	3
4	5	6
7	8	9

Axis 0

Axis 1

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

3-D Array → Tensor



Axis 0

Axis 1

Axis 2

```
np.array([[[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]],
          [[10,11,12],
           [13,14,15],
           [16,17,18]],
          [[19,20,21],
           [22,23,24],
           [25,26,27]]])
```

4. Numpy Array

- Numpy Array Creation
- Numpy Array Indexing
- Numpy Array Slicing
- Numpy Arithmetic Operations
- Numpy Arithmetic Functions
- Numpy Array Manipulation Functions
- Numpy Broadcasting
- Numpy Statistical Operations

4. Numpy Array: Numpy Array Creation

- Using the `array()` method

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

- Converting from lists, tuples → Create array from available data.
- Syntax:

```
np.array(Data)
```

Data: Input data (list, tuple, or other data types).

```
list1 = [1, 2, 3, 4, 5]
arr = np.array(list1)
```

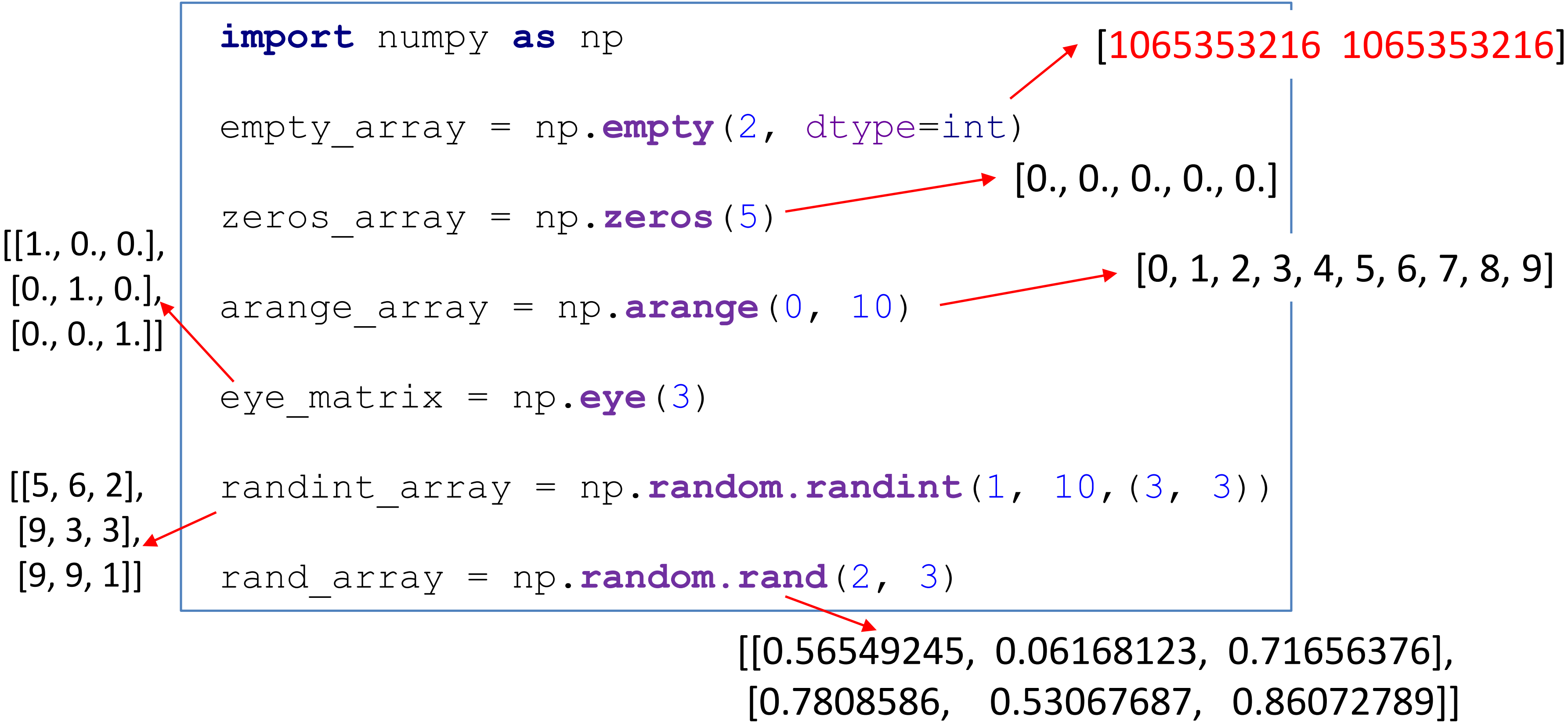
```
tuple1 = (1, 2, 3, 4, 5)
arr = np.array(tuple1)
```

4. Numpy Array: Numpy Array Creation

- Using special methods:
  - `randint`(start, end): return a random integer in range [start, end] including the end points
  - `numpy.random.rand()`: return a array of defined shape, filled with random values.
  - `numpy.eye()`: returns a 2-D array with 1's as the diagonal and 0's elsewhere
  - `numpy.full(shape,fill_value,dtype=None,order = 'C')`: return a new array with the same shape and type as a given array filled with a fill_value
  - `numpy.random.random()`: return a array of random floats in the interval [0.0, 1.0).
  - `arange([start,] stop[,step,][,dtype])`: returns an array with evenly spaced elements as per the interval. The interval mentioned is half-opened i.e. [Start, Stop)
  - `numpy.zeros()`: returns a new array of given shape and type, with zeros.
  - `numpy.ones()`: returns a new array of given shape and type, with ones.
  - `numpy.empty(shape, dtype = float, order = 'C')`: return a new array of given shape and type, with random values.

4. Numpy Array: Numpy Array Creation

- Example 1:



```
import numpy as np

empty_array = np.empty(2, dtype=int)

zeros_array = np.zeros(5)

arange_array = np.arange(0, 10)

eye_matrix = np.eye(3)

randint_array = np.random.randint(1, 10,(3, 3))

rand_array = np.random.rand(2, 3)
```

```
[1065353216  1065353216]
```

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

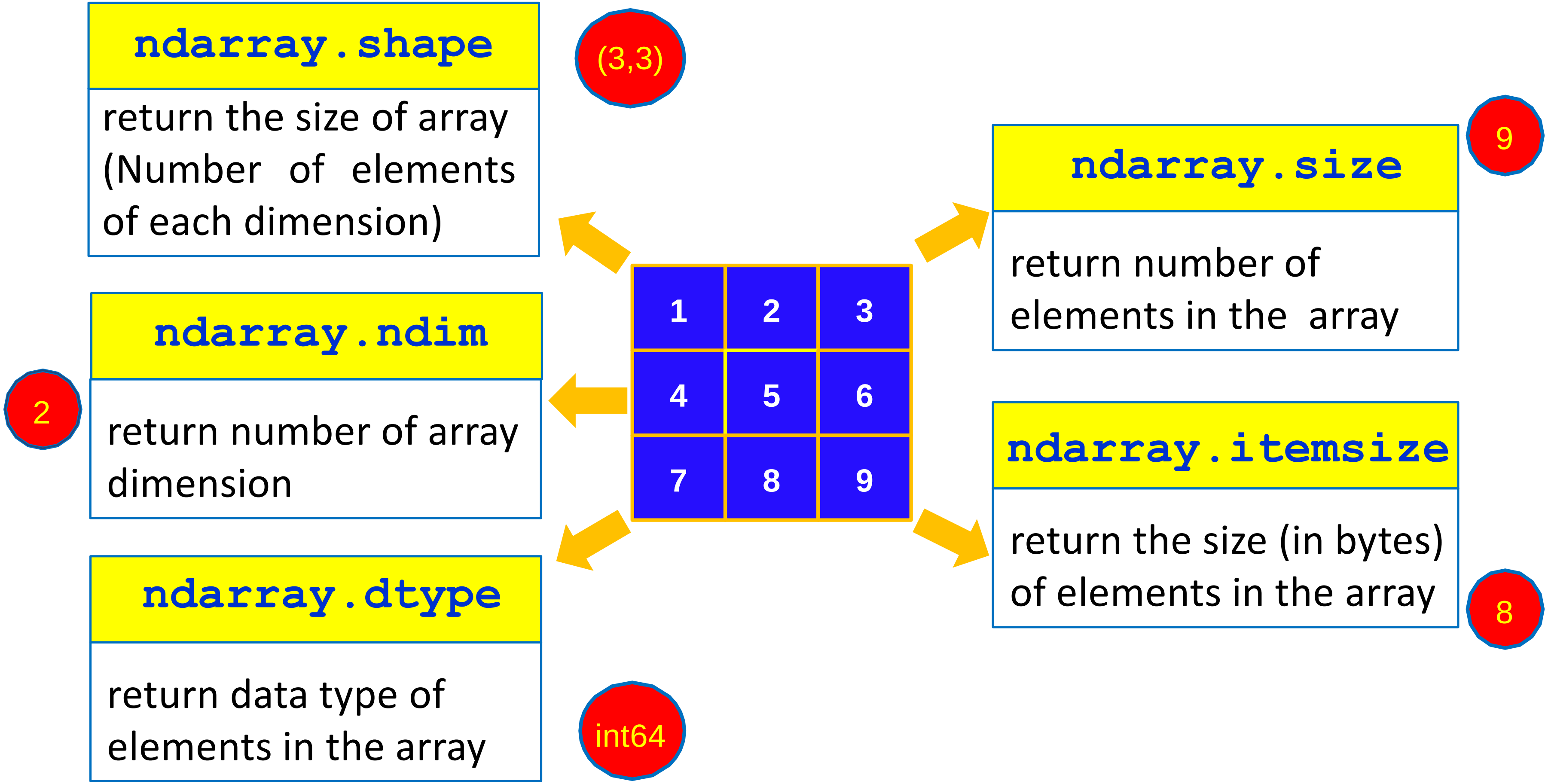
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[[1., 0., 0.],
 [0., 1., 0.],
 [0., 0., 1.]]
```

```
[[5, 6, 2],
 [9, 3, 3],
 [9, 9, 1]]
```

```
[[0.56549245,  0.06168123,  0.71656376],
 [0.7808586,   0.53067687,   0.86072789]]
```

4. Numpy Array: Numpy Array Attributes



1	2	3
4	5	6
7	8	9

ndarray.shape

return the size of array (Number of elements of each dimension)

(3,3)

ndarray.ndim

return number of array dimension

2

ndarray.dtype

return data type of elements in the array

int64

ndarray.size

return number of elements in the array

9

ndarray.itemsize

return the size (in bytes) of elements in the array

8

4. Numpy Array: Numpy Array Attributes

- Example 2:

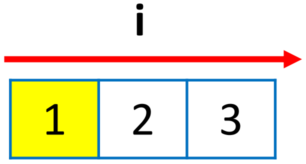
```
import numpy as np

arr = np.array([1 , 2, 3, 4, 5])
print("arr :", arr)
print("Shape of arr:", arr.shape)
print("Size of arr:", arr.size)
print("Data type of arr :", arr.dtype)
print("Number of dimensions of arr:", arr.ndim)
```

```
arr : [1,  2,  3,  4,  5]
Shape of arr: (5 ,)
Size of arr: 5
Data type of arr: int64
Number of dimensions of
arr: 1
```

4. Numpy Array: Numpy Array Indexing

- NumPy allows to access each element of an array using its index

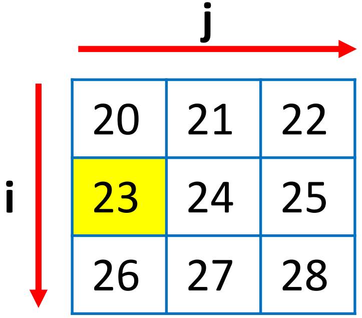
1-D Array: → ArrayName[i]



1	2	3

arr[0]

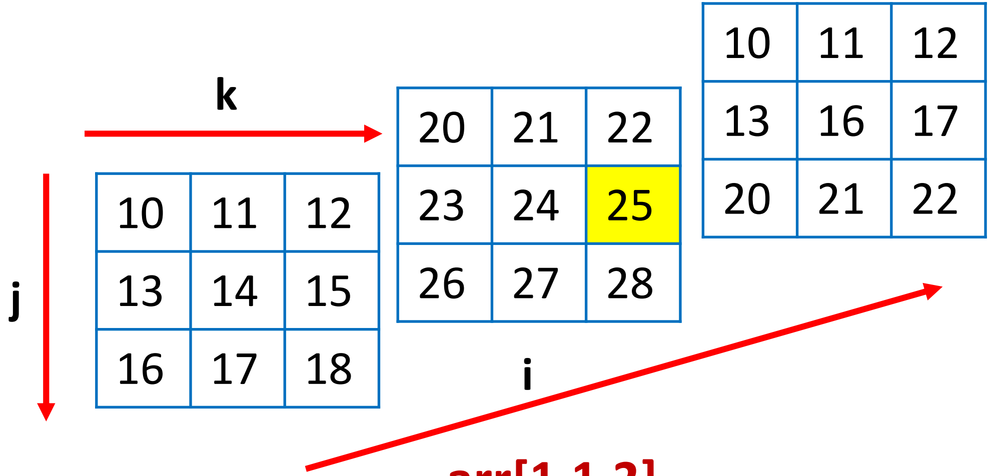
2-D Array: → ArrayName[i,j]



20	21	22
23	24	25
26	27	28

arr[1,0]

3-D Array: → ArrayName[i,j,k]



10	11	12
13	14	15
16	17	18

20	21	22
23	24	25
26	27	28

10	11	12
13	16	17
20	21	22

arr[1,1,2]

Note: The Index of first element is a 0

The Index of last element is a -1

4. Numpy Array: Numpy Array Indexing

- Example 3:

```
import numpy as np
arr1d = np.array([1 , 2, 3, 4, 5])
first_element = arr1d[0]
print("The first element of arr1d: ", first_element )
last_element = arr1d[-1]
print("The last element of arr1d: ",last_element )
```

```
The first element of arr1d:1
The last element of arr1d:5
```

- Example 4:

```
import numpy as np

arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2d)
element_1_2 = arr2d[0, 1]
print(element_1_2 )
```

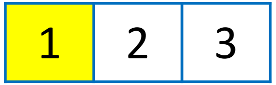
```
[[1, 2, 3],
 [4, 5, 6]]

2
```

4. Numpy Array: Numpy Array Slicing

- Slicing in NumPy to extract a part of an array using its the index range

```
ArrayName_1D[start_Index:end_Index]
```

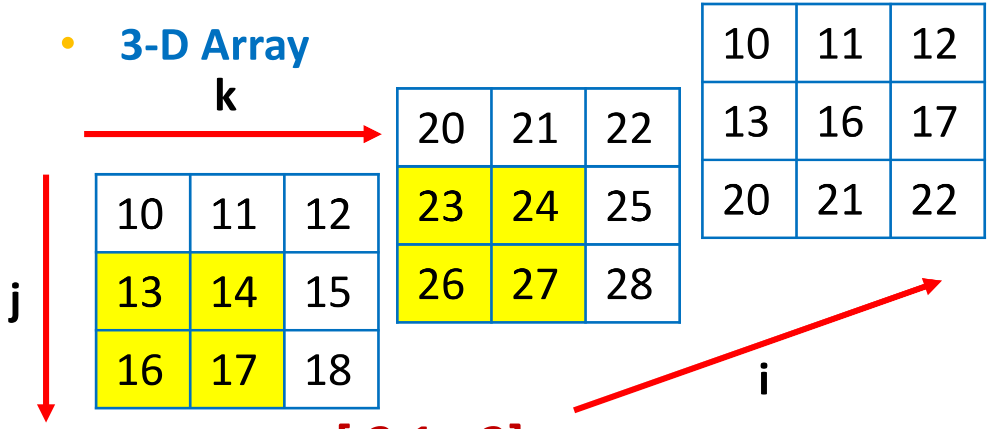
- **1-D Array**



1	2	3

arr[:1]

- **3-D Array**



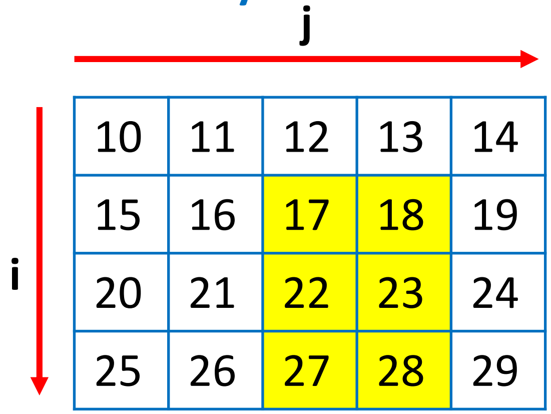
10	11	12
13	14	15
16	17	18

20	21	22
23	24	25
26	27	28

10	11	12
13	16	17
20	21	22

arr[:2,1:,:2]

- **2-D Array**

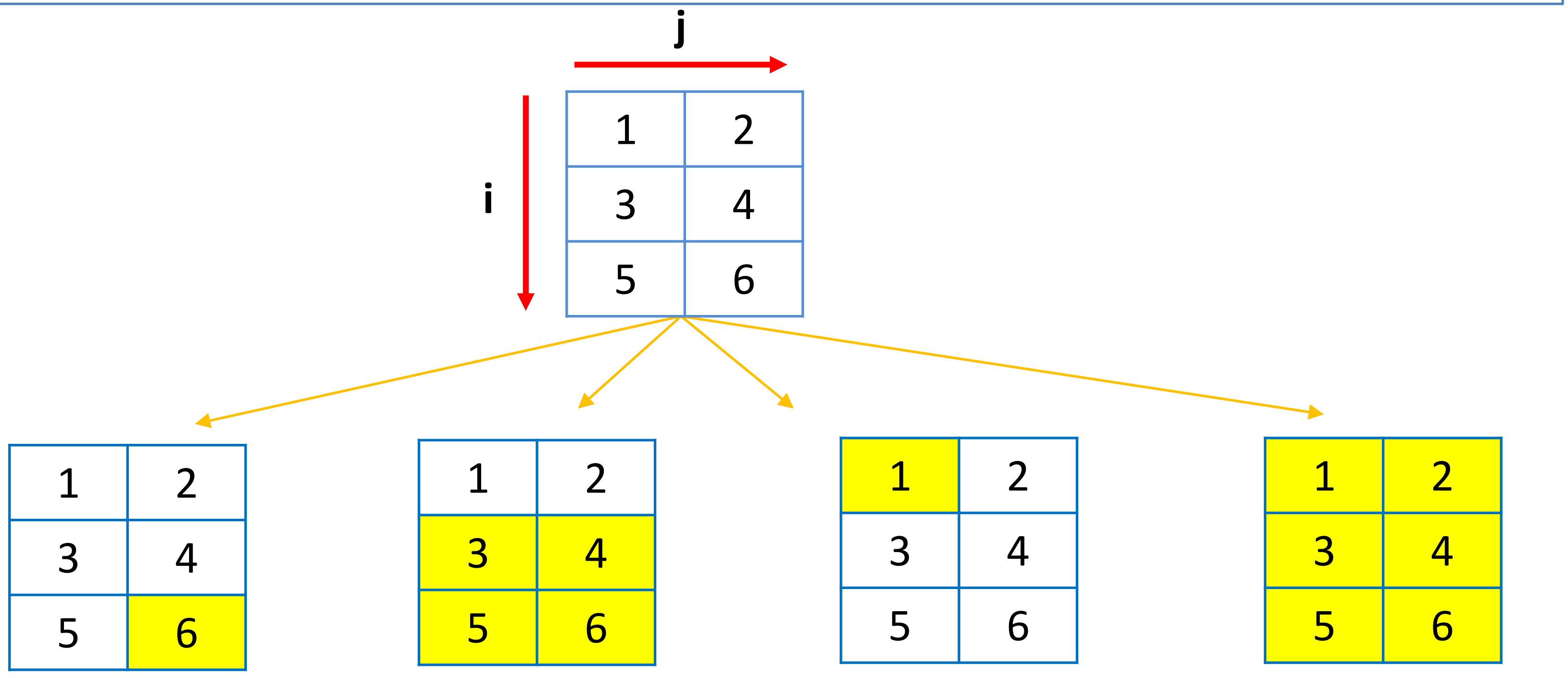


10	11	12	13	14
15	16	17	18	19
20	21	22	23	24
25	26	27	28	29

arr[1:,2:4]

4. Numpy Array: Numpy Array Slicing

```
ArrayName_2D[start_Indexi:end_Indexi, start_Indexj:end_Indexj]
```



Arr[2:, 1:]

Arr[1:, :]

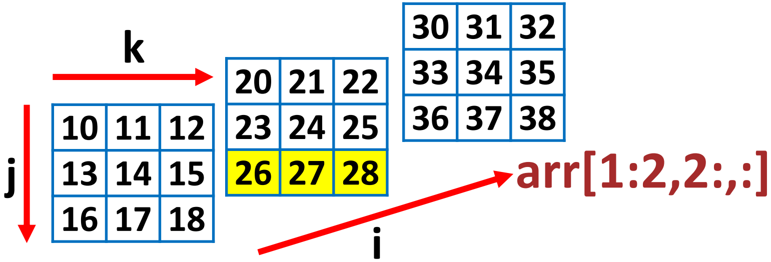
Arr[:1,:1]

Arr[:,:]

4. Numpy Array: Numpy Array Slicing

```
ArrayName_3D[start_Indexi:end_Indexi, start_Indexj:end_Indexj,
                                     start_Indexk:end_Indexk ]
```

4. Numpy Array: Numpy Array Slicing

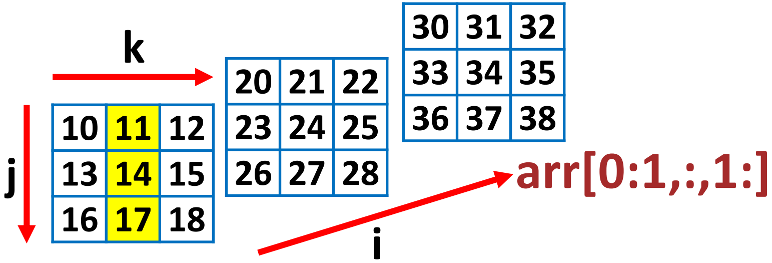


arr[1:2,2:,:]

10	11	12
13	14	15
16	17	18

20	21	22
23	24	25
26	**27**	**28**

30	31	32
33	34	35
36	37	38

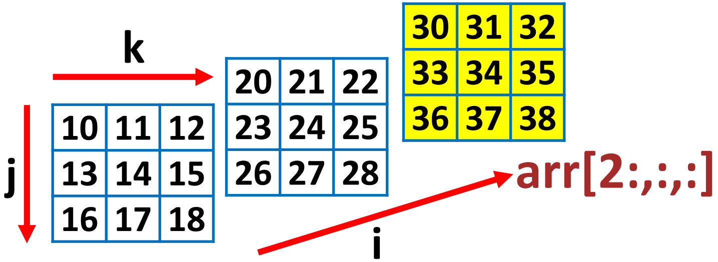


arr[0:1,:,1:]

10	**11**	12
13	**14**	15
16	**17**	18

20	21	22
23	24	25
26	27	28

30	31	32
33	34	35
36	37	38

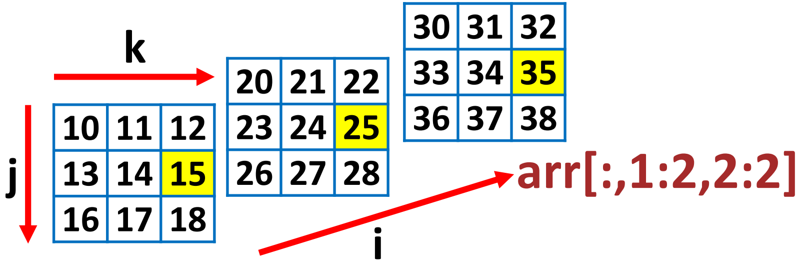


arr[2:,:,:]

10	11	12
13	14	15
16	17	18

20	21	22
23	24	25
26	27	28

30	**31**	**32**
33	**34**	**35**
36	**37**	**38**

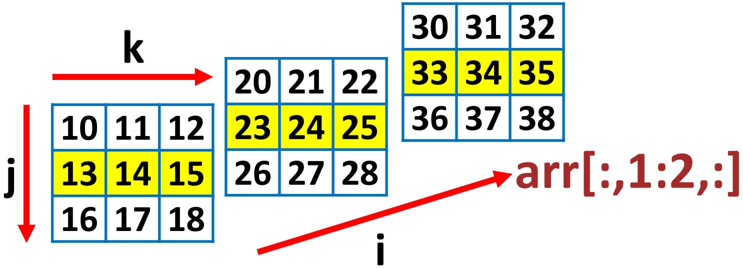


arr[:,1:2,2:2]

10	11	12
13	14	**15**
16	17	18

20	21	22
23	24	**25**
26	27	28

30	31	32
33	34	**35**
36	37	38

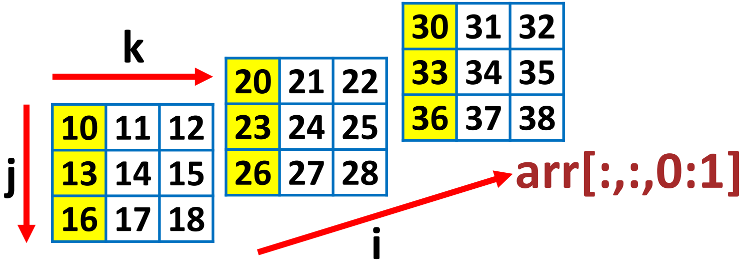


arr[:,1:2,:]

10	11	12
13	**14**	**15**
16	17	18

20	21	22
23	**24**	**25**
26	27	28

30	31	32
33	**34**	**35**
36	37	38



arr[:,:,0:1]

10	11	12
13	14	15
16	17	18

20	21	22
23	24	25
26	27	28

30	31	32
33	34	35
36	37	38

4. Numpy Array: Numpy Array Slicing

- Example 5:

```
import numpy as np
arr1d = np.array ([1 , 2, 3, 4, 5])
slice_arr1d = arr1d[1:4]
print("arr1d:", arr1d )
print("Sclicing arr1d frome 1 to 3:",slice_arr1d)
```

```
arr1d: [1, 2, 3, 4, 5]
Sclicing arr1d from 1 to 3: [2,
3,  4]
```

- Example 6:

```
import numpy as np
arr2d = np.array(
                 [[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])
slice_arr2d = arr2d[1:3 , 1:]
print("arr2d:\n", arr2d )
print("Slicing arr2d from row 2 to row 3 and
         column 2 to column end:\n", slice_arr2d )
```

```
arr2d:
[[1,  2,  3],
 [4,  5,  6],
 [7,  8,  9]]
Slicing arr2d from row 2 to row
3 and column 2 to column end:
[[5,  6],
 [8,  9]]
```

4. Numpy Array: Manipulation Functions

- **transpose**() method:
  - Create a new array by transposing a 2-dimensional numpy array.
  - Syntax:

```
new_array = original_array.transpose()
```

- Example 7:

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

new_array = original_array.transpose()
print(new_array)
```

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

4. Numpy Array: Manipulation Functions

- **reshape**() method:
  - Create a new array by changing the shape from another array
  - Syntax: new_array = np.reshape(original_array, shape))

- Example 8:

```
import numpy as np
original_array = np.array([[0, 1],
                           [2, 3],
                           [4, 5]])

new_array=np.reshape(original_array, (2,3))
print(new_array)
```

```
[[0, 2, 4],
 [1, 3, 5]]
```

4. Numpy Array: Manipulation Functions

- **reshape()** method
- Example 9:

```
import numpy as np

arr_2D = np.array([[1 , 2, 3], [4, 5, 6]])
# (2, 3) ->(3, 2)
new_arr_2D = np.reshape(arr_2D , (3, 2))
# 2D -> 1D
arr_1D = np.reshape(arr_2D , newshape =(6 , ))
print("array 2D:\n", arr_2D , arr_2D.shape )
print("new array 2D :\n", new_arr_2D, new_arr_2D.shape)
print("array 1D:\n", arr_1D, arr_1D.shape)
```

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

4. Numpy Array: Manipulation Functions

- resize() method:
  - Change the size of an array
  - Syntax: original_array.resize(row_number, column_number)
- Example 10:

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

original_array.resize(3,4)
print(original_array)
```

```
[[1, 2, 3, 4],
 [5, 6, 7, 8],
 [9, 0, 0, 0]]
```

4. Numpy Array: Manipulation Functions

- **insert**() method:
  - Create a new array by inserting values from another array
  - Syntax: `new_array =np.insert(original_array, index, value,axis)`
- Example 11:

```
import numpy as np
original_array = np.array([[1,  2,  3],
                            [4,  5,  6],
                axis =0     [7,  8,  9]])
                                 axis =1
new_array1 = np.insert(original_array,0,[10,10,10],axis=0)
new_array2 = np.insert(original_array,3,[10,10,10],axis=0)
new_array3 = np.insert(original_array,0,[10,10,10],axis=1)
new_array4 = np.insert(original_array,3,[10,10,10],axis=1)
```

```
[[10, 10, 10],
 [ 1,  2,  3],
 [ 4 , 5,  6],
 [ 7,  8,  9]]
```

```
[[ 1,  2,  3],
 [ 4,  5,  6],
 [ 7,  8,  9],
 [10, 10, 10]]
```

```
[[10,  1,  2,  3],
 [10,  4,  5,  6],
 [10,  7,  8,  9]]
```

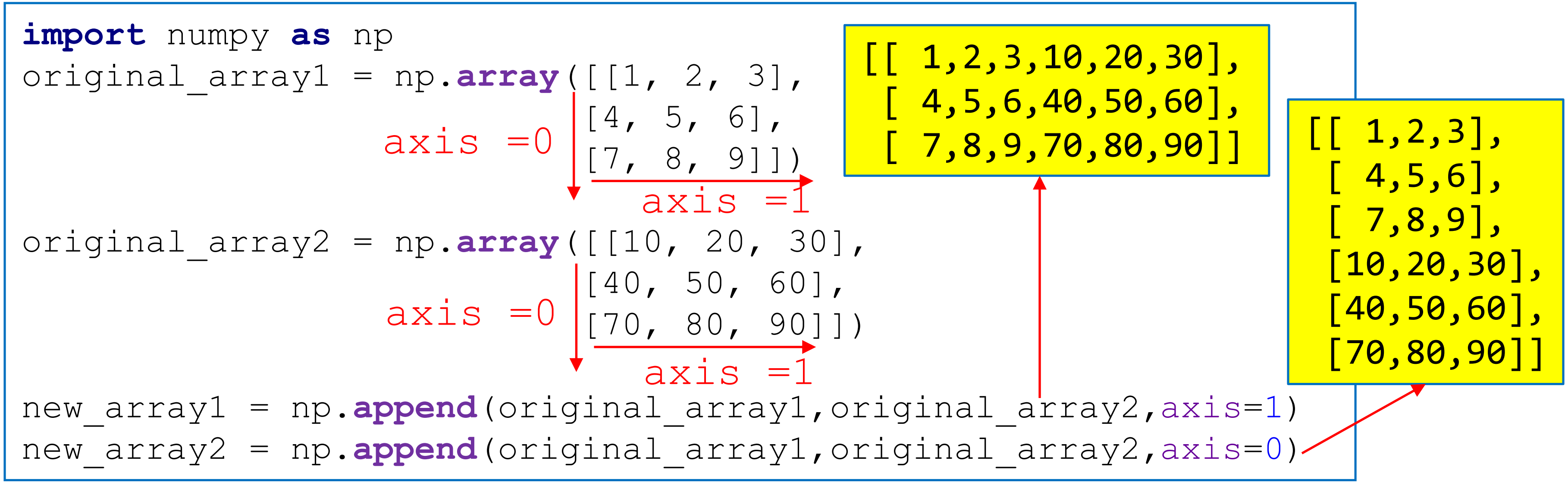
```
[[ 1,  2,  3, 10],
 [ 4,  5,  6, 10],
 [ 7,  8,  9, 10]]
```

4. Numpy Array: Manipulation Functions

- **append()** method:
  - Create a new array by appending values along the mentioned axis at the end of the array
  - Syntax:

```
new_array =np.append(original_array, value, axis)
```

- Example 12:



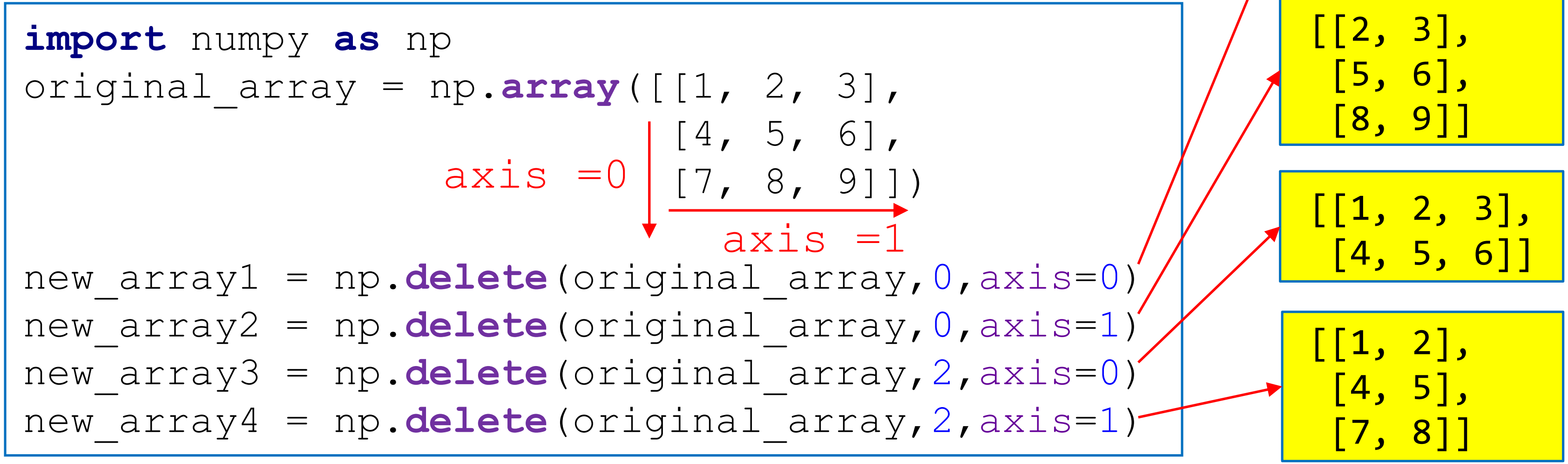
```
import numpy as np
original_array1 = np.array([[1, 2, 3],
                            [4, 5, 6],
                            [7, 8, 9]])

original_array2 = np.array([[10, 20, 30],
                            [40, 50, 60],
                            [70, 80, 90]])

new_array1 = np.append(original_array1,original_array2,axis=1)
new_array2 = np.append(original_array1,original_array2,axis=0)
```

```
[[ 1,2,3,10,20,30],
 [ 4,5,6,40,50,60],
 [ 7,8,9,70,80,90]]
```

```
[[ 1,2,3],
 [ 4,5,6],
 [ 7,8,9],
 [10,20,30],
 [40,50,60],
 [70,80,90]]
```

4. Numpy Array: Manipulation Functions

- **delete()** method:
  - Returns a new array with the deletion of sub-arrays along with the mentioned axis
  - Syntax: `new_array =np.delete(original_array, index, axis)`
- Example 13:



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

new_array1 = np.delete(original_array,0,axis=0)
new_array2 = np.delete(original_array,0,axis=1)
new_array3 = np.delete(original_array,2,axis=0)
new_array4 = np.delete(original_array,2,axis=1)
```

```
[[4, 5, 6],
 [7, 8, 9]]
```

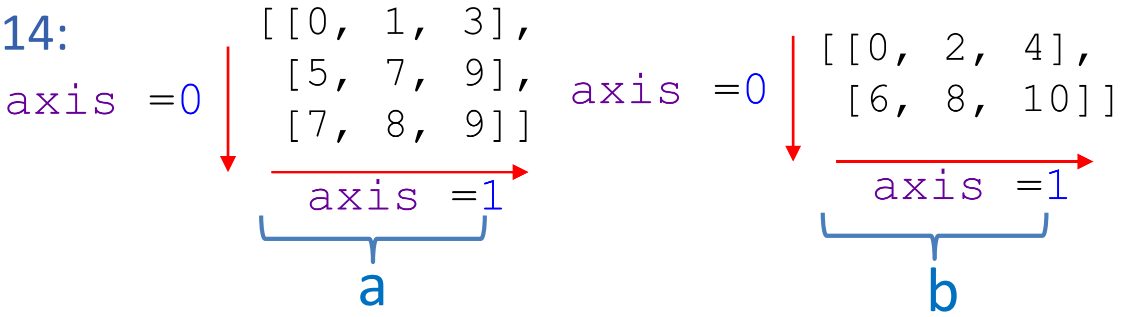
```
[[2, 3],
 [5, 6],
 [8, 9]]
```

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

```
[[1, 2],
 [4, 5],
 [7, 8]]
```

4. Numpy Array: Manipulation Functions

- Example 14:



axis =0

```
[[0, 1, 3],
 [5, 7, 9],
 [7, 8, 9]]
```

axis =1

a

axis =0

```
[[0, 2, 4],
 [6, 8, 10]]
```

axis =1

b

```
[[0, 1, 3, 0, 2, 4],
 [5, 7, 9, 6, 8, 10]]
```

np.concatenate((a,b),axis=1)

```
[[0, 1, 3, 0, 2, 4],
 [5, 7, 9, 6, 8, 10]]
```

np. hstack ((a , b))

```
[[0, 1, 3,]
 [0, 2, 4],
 [5, 7, 9],
 [6, 8, 10]]
```

np. vstack ((a , b))

4. Numpy Array: Manipulation Functions

- Example 15:

```
import numpy as np
arr_1 = np.array([1 , 2, 3])
arr_2 = np.array([4 , 5, 6])
arr_3 = np.hstack((arr_1 , arr_2))
print("arr 1:\n", arr_1)
print("arr 2:\n", arr_2)
print("Result_arr:\n", arr_3)
```

```
arr 1: [1,  2,  3]
arr 2: [4,  5,  6]
Result_arr:
 [1, 2, 3, 4, 5, 6]
```

- Example 16:

```
import numpy as np
arr_1 = np.array([1 , 2, 3])
arr_2 = np.array([4 , 5, 6])
arr_3 = np.vstack((arr_1 , arr_2))
print("arr 1:", arr_1)
print("arr 2:", arr_2)
print("Result_arr:\n", arr_3)
```

```
arr 1:[1,  2,  3]
arr 2:[4,  5,  6]
Result_arr:
  [[1,  2,  3],
   [4,  5,  6]]
```

4. Numpy Array: Manipulation Functions

- Example 17:

```
import numpy as np

arr_1 = np.array([[1 , 2, 3],
                   [4, 5, 6]])
arr_2 = np.array([[7 , 8, 9],
                   [10 , 11, 12]])

arr_3 = np.concatenate((arr_1, arr_2),axis =0)
arr_4 = np.concatenate((arr_1, arr_2),axis =1)

print("Array 1:\n", arr_1)
print("Array 2:\n", arr_2)
print("Result( axis =0) :\n",arr_3)
print("Result( axis =1) :\n",arr_4)
```

```
Array 1:
 [[1,2,3],
  [4,5,6]]

Array 2:
 [[7,8,9],
  [10,11,12]]

Result(axis =0):
[[1,2,3],
 [4,5,6],
 [7,8,9],
 [10,11,12]]

Result(axis =1):
[[1,2,3,7,8,9],
 [4,5,6,10,11,12]]
```

4. Numpy Array: Manipulation Functions

- Add a New Axis to a Numpy Array: There are various ways to add a new axis to a NumPy array
  - Using
    - `newaxis()` method
    - `expand_dims()` method
    - `reshape()` method

4. Numpy Array: Manipulation Functions

- **newaxis()** method
- Example 18:

```
import numpy as np

# array 1D
arr_1 = np.array([1 , 2, 3])

# 1D -> 2D
arr_2 = arr_1[np.newaxis, :]

# 2D -> 5D
arr_3 = arr_2[np.newaxis, :, np.newaxis ,:, np.newaxis]
print("array 1: ", arr_1 , arr_1.shape)
print("array 2: ", arr_2 , arr_2.shape)
print("array 3:\n ", arr_3 , arr_3.shape)
```

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

4. Numpy Array: Manipulation Functions

- **expand_dims()** method
- Example 19:

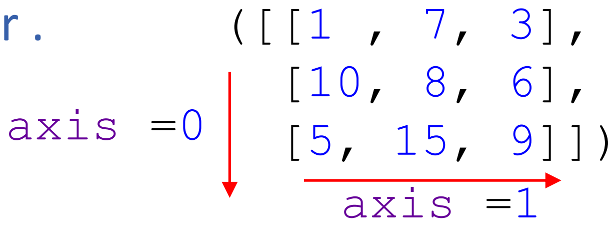
```
import numpy as np

arr_1 = np.array([1 , 2, 3])

# 1D -> 2D
arr_2 = np.expand_dims(arr_1 , axis =0)

# 2D -> 5D
arr_3 = np.expand_dims(arr_2 ,axis =(0 , 2, 4))

print("array 1:", arr_1 , arr_1.shape)
print("array 2:", arr_2 , arr_2.shape)
print("array 3:\n", arr_3 , arr_3.shape)
```

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

4. Numpy Array: Manipulation Functions

- Sort the elements of the array in ascending order .
- Syntax:

```
np.sort(arr , [axis] )
```



```
([[1 , 7, 3],
  [10, 8, 6],
  [5, 15, 9]])
```

axis =0

axis =1

  - arr: input array
  - axis: sort direction

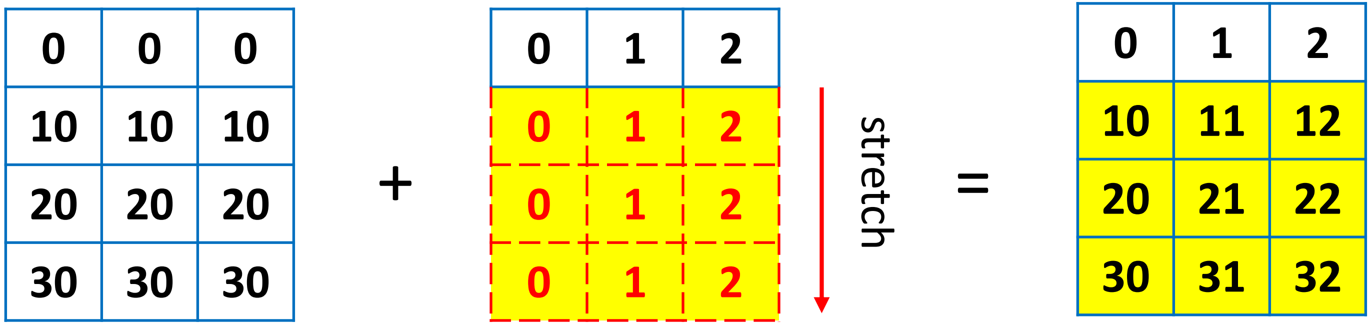
- Example 20:

```
import numpy as np
arr2d = np. array ([[1 , 7, 3],
                    [10, 8, 6],
                    [5, 15, 9]])
print("Original array\n",arr2d)
sort_in_row = np.sort(arr2d , axis =1)
sort_in_col = np.sort(arr2d , axis =0)
print("Result by row:\n", sort_in_row)
print("Result by col:\n", sort_in_col)
```

```
Original array:
 [[1,7,3],
  [10,8,6],
  [5,15,9]]
Result by row:
 [[1,3,7],
  [6,8,10],
  [5,9,15]]
Result by col:
 [[1,7,3],
  [5,8,6],
  [10,15,9]]
```

4. Numpy Array: Numpy Broadcasting

- Broadcasting:
  - Allows performing mathematical operations on arrays of different sizes
  - When done operations, NumPy automatically expands smaller arrays so that they are the same size as the larger array
  - ➔ Thereby, saving memory and optimizing performance



0	0	0
10	10	10
20	20	20
30	30	30

A (4 x 3)

+

0	1	2
0	1	2
0	1	2
0	1	2

B (3)

=

0	1	2
10	11	12
20	21	22
30	31	32

Result (4 x 3)

4. Numpy Array: Numpy Broadcasting

- Broadcasting Rules (Simplified):
  - If array shapes differ in rank, prepend 1s to the smaller one until they match in length.
  - Two dimensions are compatible if they are equal or one of them is 1.
  - Arrays can be broadcast only if they are compatible in all dimensions.
  - After broadcasting, both arrays behave as if they had the shape of the element-wise maximum.
  - Any array with dimension 1 behaves as if it's copied to match the other array's size in that dimension.

4. Numpy Array: Numpy Arithmetic Operations

- Example 21:

```
[[0,  3,  1],
 [3,  7,  5],
 [6,  4,  8]]
```

a

```
[10,  5,  11]
```

b

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

```
[[10,8,12],
 13,12,16],
 [16,9,19]]
```

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

```
[[-10,-2,-10],
 [-7,2,-6],
 [-4,-1,-3]]
```

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

```
[[0,15,11],
[30,35,55],
[60,20,88]]
```

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

```
[[0.,  0.6, 0.09090909],
 [0.3, 1.4, 0.45454545],
 [0.6, 0.8, 0.72727273]]
```

4. Numpy Array: Numpy Arithmetic Operations

- Example 22:

```python
import numpy as np
a = np.array ([[0, 3, 1],
               [3, 7, 5],
               [6, 4, 8]])

b = [10, 5, 11]

print("add_array =\n",np.add(a,b))
print("subtract_array =\n",np.subtract(a,b))
print("multiply_array =\n",np.multiply(a,b))
print("divide_array =\n",np.divide(a,b))
```

```
add_array =
[[10,  8, 12],
 [13, 12, 16],
 [16,  9, 19]]
subtract_array =
 [[-10,  -2, -10],
  [ -7,   2,  -6],
  [ -4,  -1,  -3]]
multiply_array =
 [[ 0, 15, 11],
  [30, 35, 55],
  [60, 20, 88]]
divide_array =
 [[0. , 0.6, 0.09090909],
  [0.3, 1.4, 0.45454545],
  [0.6, 0.8, 0.72727273]]
```

4. Numpy Array: Numpy Arithmetic Operations

- Example 23:

$\underbrace{[7,3,4,5,1]}_{a}$ $\underbrace{[3,4,5,6,7]}_{b}$

```
np.remainder(a,b)
→ [1,  3,  4,  5,  1]
```

```
np.power(a,b)
→ [343,  81,1024,15625,  1]
```

```
np.mod(a,b)
→ [1,  3,  4,  5,  1]
```

```
np.reciprocal(a)
→ [0,  0,  0,  0,  1]
```

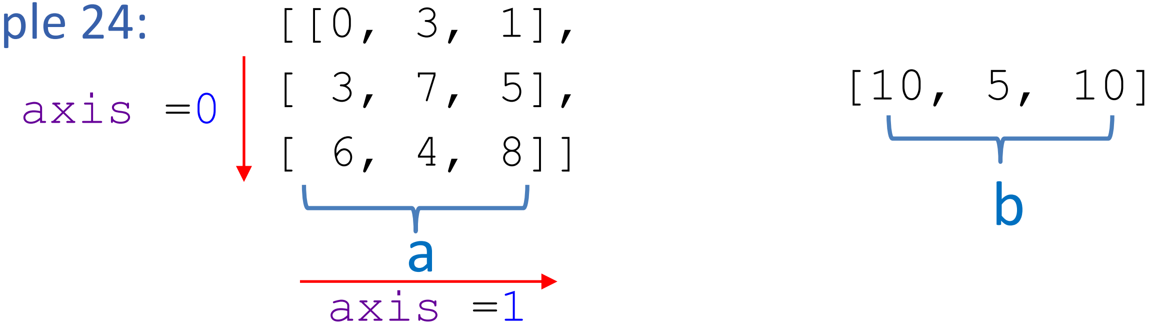
4. Numpy Array: Numpy Arithmetic Operations

- Returm sum value in the array.
- Syntax:

```
np.sum(array,[axis])
```

- array: input array
- axis: search direction

- Example 24:



```
[[0, 3, 1],
[ 3, 7, 5],
[ 6, 4, 8]]
```

```
[10, 5, 10]
```

```
np.sum(a)              37
np.sum(b)              25
np.sum(a , axis =0)    [9, 14, 14]
np.sum(a , axis =1)    [4, 15, 18]
```

4. Numpy Array: Numpy Arithmetic Operations

- Example 25:

```python
import numpy as np
a = np.array ([[0, 3, 1],
               [3, 7, 5],
               [6, 4, 8]])

b = [10, 5, 11]

print("Sum_a =", np.sum(a))
print("Sum_b =", np.sum(b))
print("Sum_a_axis_0 =", np.sum(a,axis=0))
print("Sum_a_axis_1 =", np.sum(a,axis = 1))
```

→

```
Sum_a = 37
Sum_b = 26
Sum_a_axis_0 = [ 9, 14, 14]
Sum_a_axis_1 = [ 4, 15, 18]
```

4. Numpy Array: Numpy Statistical Operations

- Returm the largest/smallest value in the array.
- Syntax:

```
np.max(array , [axis] )
np.min(array , [axis] )
```

- array: input array
- axis: search direction

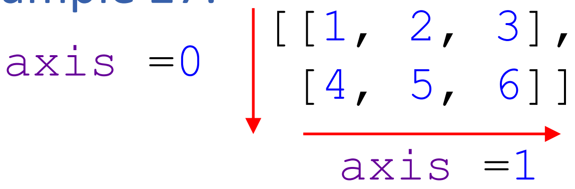
- Example 26:

```
import numpy as np
arr = np. array ([1 , 2, 3, 4, 5])
max_value = np.max(arr)
print("largest value :", max_value)
min_value = np.min(arr)
print("Smallest value:", min_value)
```

```
largest value : 5
Smallest value: 1
```

4. Numpy Array: Numpy Statistical Operations

- Example 27:


axis =0

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

axis =1

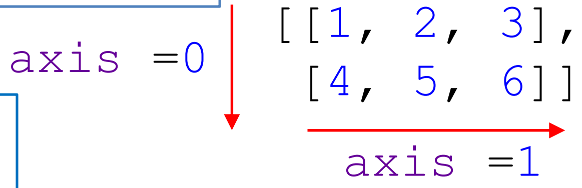
```
import numpy as np
arr2d = np.array ([[1 , 2, 3], [4, 5, 6]])
max_value_row = np.max(arr2d , axis =1)
max_value_col = np.max(arr2d , axis =0)
print("Maximum value in row:", max_value_row)
print("Maximum value in col:", max_value_col)

min_value_row = np.min(arr2d , axis =1)
min_value_col = np.min(arr2d , axis =0)
print("Minimum value in row:", min_value_row)
print("Minimum value in col:", min_value_col)
```

```
Maximum value in row: [3,  6]
Maximum value in col: [4, 5, 6]
Minimum value in row: [1,  4]
Minimum value in col: [1 2, 3]
```

4. Numpy Array: Numpy Statistical Operations

- Return the Index of the element that has the largest/smallest value.
- Syntax:
  - arr: input array
  - axis: search direction
- Example 28:

```
np. argmax(arr , [axis] )
np. argmin(arr , [axis] )
```



```
import numpy as np
arr2d = np. array ([[1 , 2, 3], [4, 5, 6]])
max_value_row = np.max(arr2d , axis =1)
max_index_row = np.argmax(arr2d , axis =1)
print("Maximum value in row:",max_value_row)
print("Index:", max_index_row)

max_value_col = np.max(arr2d , axis =0)
max_index_col = np.argmax(arr2d , axis =0)
print("Maximum value in col:", max_value_col)
print("Index:", max_index_col)
```

```
Maximum value in row:[3,6]
Index: [2,2]
Maximum value in col:[4,5,6]
Index: [1,1,1]
```

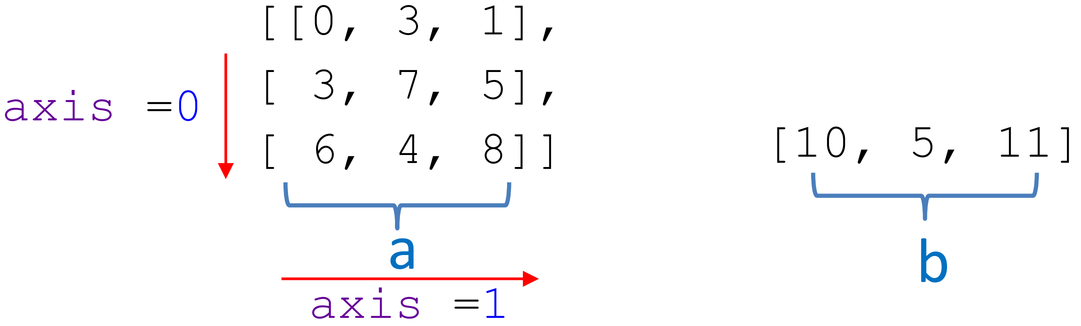
4. Numpy Array: Numpy Statistical Operations

- Returm median value in the array.
- Syntax:

```
np.median(array,[axis])
```

- `array`: input array
- `axis`: search direction

- Example 29:



```
        [[0, 3, 1],
axis =0  [ 3, 7, 5],
         [ 6, 4, 8]]        [10, 5, 11]
```

a

axis =1

b

```
np.median(a)          4.0
np.median(b)          10.0
np.median(a, axis =0)        [3.0, 4.0, 5.0]
np.median(a, axis =1)        [1.0, 5.0, 6.0]
```

4. Numpy Array: Numpy Statistical Operations

- Example 30:

```
import numpy as np
a = np.array ([[0, 3, 1],
               [3, 7, 5],
               [6, 4, 8]])
b = [10, 5, 11]

print("Median value in row:", np.median(a , axis =1))
print("Median value in col:", np.median(a , axis =0))
print("Median array A:", np.median(a))
print("Median array B:", np.median(b))
```

```
Median value in row: [1.,  5.,  6.]
Median value in col: [3.,  4.,  5.]
Median array A: 4.0
Median array B: 10.0
```

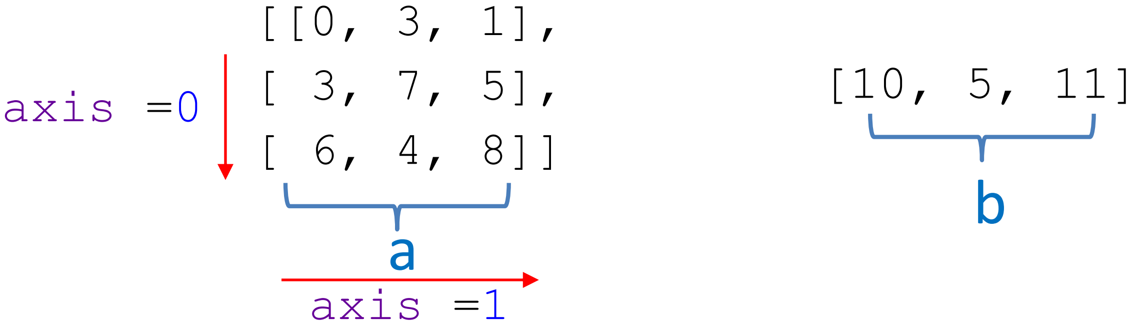
4. Numpy Array: Numpy Statistical Operations

- Returm mean value in the array.
- Syntax:

```
np.mean(array, [axis])
```

- `array`: input array
- `axis`: search direction

- Example 31:



```
[[0, 3, 1],
 [ 3, 7, 5],
 [ 6, 4, 8]]
```

```
[10, 5, 11]
```

```
np.mean(a)                  4.11
np.mean(b)                  8.66
np.mean(a , axis =0)          [3.0,    4.66,    4.66]
np.mean(a , axis =1)          [1.33,   5.0,     6.0]
```

4. Numpy Array: Numpy Statistical Operations

- Example 32:

```
import numpy as np
a = np.array ([[0, 3, 1],
               [3, 7, 5],
               [6, 4, 8]])

b = [10, 5, 11]

print("Mean_a =", np.mean(a))
print("Mean_b =", np.mean(b))
print("Mean_a_axis_0 =", np.mean(a,axis = 0))
print("Mean_a_axis_1 =", np.mean(a,axis = 1))
```

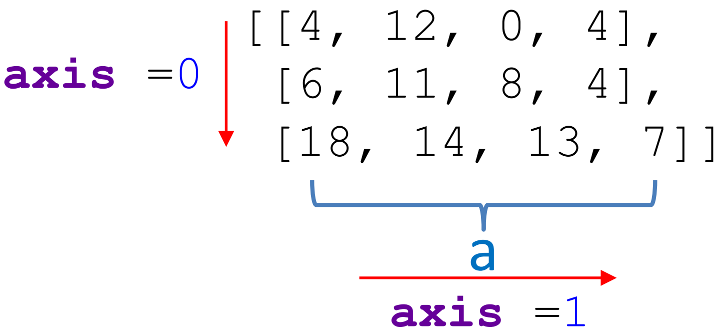
```
Mean_a = 4.11
Mean_b = 8.66
Mean_a_axis_0 = [3., 4.66, 4.66]
Mean_a_axis_1 = [1.33, 5., 6.  ]
```

4. Numpy Array: Numpy Statistical Operations

- Return standard deviation:
- Return variance :

```
np.std(array,[axis])
np.var(array, [axis])
```

- Example 33:



axis =0

```
[[4, 12, 0, 4],
 [6, 11, 8, 4],
 [18, 14, 13, 7]]
```

a

axis =1

- `array`: input array
- `axis`: search direction

```
np.std(a,axis=0)        [6.18, 1.24, 5.35, 1.41]
np.std(a,axis=1)        [4.35, 2.58, 3.93]
np.var(a,axis=0)        [38.22, 1.55, 28.66, 2.   ]
np.var(a,axis=1)        [19.,    6.68, 15.5    ]
```

4. Numpy Array: Numpy Statistical Operations

- Example 34:

```
import numpy as np
a = np.array ([[4, 12, 0, 4],
               [6, 11, 8, 4],
               [18, 14, 13, 7]])

print("Standard Deviation_a_axis_0 =", np.std(a,axis=0))
print("Standard Deviation_a_axis_1 =", np.std(a,axis=1))
print("Variance_a_axis_0 =", np.var(a,axis=0))
print("Variance_a_axis_1 =", np.var(a,axis=1))
```

```
Standard Deviation_a_axis_0 = [6.18, 1.24, 5.35, 1.41]
Standard Deviation_a_axis_1 = [4.35, 2.58, 3.93]
Variance_a_axis_0 = [38.22, 1.55, 28.66,  2.    ]
Variance_a_axis_1 = [19.  ,  6.68, 15.5   ]
```

4. Numpy Array: Numpy Statistical Operations

- Return the elements with conditions **where()**
- Syntax:

```
np.where (condition)
```

- Example 35:

arr =	1	2	3	4	5
arr >2	F	F	T	T	T
result =	3	4	5		

```
import numpy as np

arr = np.array([1 , 2, 3, 4, 5])
result = np.where(arr > 2)
print("Position of elements greater than 2:",result)
print("The value of the elements at the found position:",arr[result])
```

```
Position of elements greater than 2: (array([2, 3, 4], dtype=int64),)
The value of the elements at the found position: [3, 4, 5]
```

5. Numpy Linear Algebra

- Linalg: the package in NumPy for Linear Algebra
  - `dot()` : Dot product of two arrays
  - `vdot()`: Return the dot product of two vectors.
  - `inner()`: Inner product of two arrays
  - `outer()`: Compute the outer product of two vectors.
  - `matmul()`: Matrix product of two arrays.
  - https://numpy.org/doc/stable/reference/routines.linalg.html

5. Numpy Linear Algebra

- Example 36:

```
import numpy as np
a = np.array([[1, 0],
              [0, 1]])

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

print("Dot_Result =\n", np.dot(a, b))
print("Vdot_Result =\n", np.vdot(a, b))
print("Inner_Result =\n", np.inner(a, b))
print("Outer_Result =\n", np.outer(a, b))
```

→

```
Dot_Result =
[[4, 1]
 [2, 2]]

Vdot_Result =
6

Inner_Result =
[[4, 2]
 [1, 2]]

Outer_Result =
[[4, 1, 2, 2]
 [0, 0, 0, 0]
 [0, 0, 0, 0]
 [4, 1, 2, 2]]
```

6. Numpy Linear Algebra

- Example 37:

```
import numpy as np

arr_1 = np.array([[1 , 2], [3, 4]])
arr_2 = np.array([[5 , 6], [7, 8]])
print("arr_1 :\n", arr_1 )
print("arr_2 :\n", arr_2 )
result_matmul = np.matmul(arr_1,arr_2)
print("Result:\n",result_matmul)
```

```
arr_1:
[[1,   2],
 [3,  4]]

arr_2:
[[5,  6],
 [7,  8]]

Result:
[[19,  22],
 [43,  50]]
```

- Example 38:

Euclicdean Distance Using Python

```
import math
def Euclicdean_distance(point1, point2):
   return math.sqrt((point1[0]-point2[0])**2+ point1[1]-point2[1])**2)
point1 = (1, 2)
point2 = (4, 6)
distance = Euclicdean_distance(point1, point2)
print("Euclicdean distance:", distance)
# Result: Euclicdean distance: 5.0
```

$$d(p,q) = \sqrt{\sum_{i=1}^{N} (q_i - p_i)^2}$$

```
import numpy as np
def Euclicdean_distance_np(point1, point2):
    return np.linalg.norm(np.array(point1)-np.array(point2))
point1 = (1, 2)
point2 = (4, 6)
distance = Euclicdean_distance_np(point1, point2)
print("Euclicdean distance (NumPy):", distance)
# Result: Euclicdean distance (NumPy): 5.0
```

Euclicdean Distance Using Numpy

- Example 39:

Manhattan Distance Using Python

```python
def manhattan_distance(point1, point2):
    return sum(abs(a - b) for a, b in zip(point1, point2))
point1 = (1, 2)
point2 = (4, 6)
distance = manhattan_distance(point1, point2)
print("Manhattan distance:", distance)
#Result: Manhattan distance: 7
```

$$d(p,q) = \sum_{i=1}^{N} |q_i - p_i|$$

```python
import numpy as np
def manhattan_distance_np(point1, point2):
    return np.sum(np.abs(np.array(point1) - np.array(point2)))
point1 = (1, 2)
point2 = (4, 6)
distance = manhattan_distance_np(point1, point2)
print("Manhattan distance (NumPy):", distance)
# Result: Manhattan distance (NumPy): 7
```

Manhattan Distance Using NumPy

6. Numpy Matrix Library Matlib

- Has functions that return matrices instead of ndarray objects
- Example 40:

```
import numpy as np
import numpy.matlib
#with the specified shape and type without initializing entries
mat_e = np.matlib.empty((3, 2), dtype=int)
#filled with 0
mat_zeros =np.matlib.zeros(5, 3)
#filled with 1
mat_ones = np.matlib.ones(4,3)
#diagonal elements filled with 1, others with 0
mat_ones = np.matlib.eye(3,5)
#create square matrix with 0, diagonal filled with 1, others with 0
mat_zeros = np.matlib.identity(5)
#filled with random data
mat_e = np.matlib.empty(3, 2)
```

7. Input/Output with Numpy

- What are the I/O functions of NumPy?
- The I/O functions provided by NumPy are
  - `Load()` and `save()` methods handle numPy binary files (with npy extension)
  - `loadtxt()` and `savetxt()` methods handle normal text files

7. Input/Output with Numpy

- Example 41: The input array is stored in a disk file with the **save()** method, file and is loaded by **load()** method

```
import numpy as np
a = np.array([1,2,3,4,5])
np.save("out.npy", a)
b = np.load("out.npy")
print(b)                  →    [1, 2, 3, 4, 5]
```

- Example 42: The input array is stored in a disk file with the **savetxt()** method, file and is loaded by **loadtxt()** method

```
import numpy as np
a = np.array([1,2,3,4,5])
np.savetxt("out.txt", a)
b = np.loadtxt("out.txt")
print(b)               →    [1., 2., 3., 4., 5.]
```

Practice and exercises

- 1. **Practice**: Practice all the examples of Chapter 5
- 2**. Exercise**: in the next slide below

Exercises

1. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

a = np.array([1, 2, 3])   # …
print(type(a))            # …
print(a.shape)            # …
print(a[0], a[1], a[2])   # …
a[0] = 5                  # …
print(a)                  # …

b = np.array([[1,2,3],[4,5,6]])    # …
print(b.shape)                     # …
print(b[0, 0], b[0, 1], b[1, 0])   # …
```

Exercises

2. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

a = np.zeros((2,2))   # …
b = np.ones((1,2))    # …
c = np.full((2,2), 7)  # …
d = np.eye(2)         # …
e = np.random.random((2,2))  # …
```

Exercises

3. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

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

b = a[:2, 1:3]

print(a[0, 1])     # …

b[0, 0] = 77       # …

print(a[0, 1])     # …
```

Exercises

4. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

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

row_r1 = a[1, :]    # …
row_r2 = a[1:2, :]  # …
print(row_r1, row_r1.shape)  # …
print(row_r2, row_r2.shape)  # …

col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)  # …
print(col_r2, col_r2.shape)  # …
```

Exercises

5. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

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

print(a[[0, 1, 2], [0, 1, 0]])   # …

print(np.array([a[0, 0], a[1, 1], a[2, 0]]))   # …

print(a[[0, 0], [1, 1]])   # …

print(np.array([a[0, 1], a[0, 1]]))   # …
```

Exercises

6. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

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

print(a)  #...

b = np.array([0, 2, 0, 1]) #...

print(a[np.arange(4), b])  # …

a[np.arange(4), b] += 10 #...

print(a) #...
```

Exercises

7. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

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

bool_idx = (a > 2)    # …

print(bool_idx)       # …

print(a[bool_idx])   # …

print(a[a > 2])      # …
```

Exercises

8. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

x = np.array([1, 2])   # …
print(x.dtype)         # …

x = np.array([1.0, 2.0])   # …
print(x.dtype)             # …

x = np.array([1, 2], dtype=np.int64)   # …
print(x.dtype) # …
```

Exercises

9. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)  #...
y = np.array([[5,6],[7,8]], dtype=np.float64)  #...

print(x + y) #...
print(np.add(x, y)) #...
print(x - y) #...
print(np.subtract(x, y)) #...
print(x * y) #...
print(np.multiply(x, y)) #...
print(x / y) #...
print(np.divide(x, y)) #...
print(np.sqrt(x)) #...
```

Exercises

9. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

x = np.array([[1,2],[3,4]]) #...
y = np.array([[5,6],[7,8]]) #...
v = np.array([9,10]) #...
w = np.array([11, 12]) #...

print(v.dot(w)) #...
print(np.dot(v, w)) #...
print(x.dot(v)) #...
print(np.dot(x, v)) #...
print(x.dot(y)) #...
print(np.dot(x, y)) #...
```

Exercises

10. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

x = np.array([[1,2],[3,4]]) # …

print(np.sum(x))  # …
print(np.sum(x, axis=0))  # …
print(np.sum(x, axis=1))  # …
x = np.array([[1,2], [3,4]])
print(x)    # …
print(x.T)  # …
v = np.array([1,2,3])
print(v)    # …
print(v.T)  # …
```

Exercises

11. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)    # …

for i in range(4):
    y[i, :] = x[i, :] + v

print(y) #...
```

Exercises

12. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1))    # …
print(vv)                  # …
y = x + vv   # …
print(y)   # …
```

13. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v   # …
print(y)   # …
```

Exercises

14. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

# Compute outer product of vectors
v = np.array([1,2,3])  # v has shape (3,)
w = np.array([4,5])    # w has shape (2,)
print(np.reshape(v, (3, 1)) * w) #...

x = np.array([[1,2,3], [4,5,6]]) #...
print(x + v) #...

print((x.T + w).T)#...
print(x + np.reshape(w, (2, 1)))#...
```

Exercises

15. Explain the meaning of each command line and show the result of the following code:

```
import numpy as np

v = np.array([1,2,3])  # …
w = np.array([4,5])    # …
print(np.reshape(v, (3, 1)) * w) # …

x = np.array([[1,2,3], [4,5,6]]) # …
print(x + v) # …

print((x.T + w).T) # …
print(x + np.reshape(w, (2, 1))) # …

print(x * 2) # …
```

The end of Chapter