Numpy Review (Part 1)

Zohreh Karimi

1- NumPy

NumPy is the fundamental package for scientific computing in python. Its stands for 'Numerical Python'.

It provides a high-performance multidimensional array object, and tools for working with these arrays. This tutorial explains the basic of NumPy, array functions, etc. Numeric is the ancestor of NumPy.

2- Operation using NumPy:

- You can perform following operations using NumPy:
- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra.

Install NumPy using popular python package installer, pip.

```
pip install numpy
```

If you use conda:

```
conda install numpy
```

to test whether numpy module is properly installed, try to import from Python Prompt:

3- import numpy

Alternatively, NumPy package is imported using the following syntax:

```
import numpy as np
```

4- Ndarray Object

The most important object defined in NumPy is an N-dimensional array type called ndarray. It describes the collection of items of the same type. Every item in a ndarray takes the same size of block in the memory. Each element in ndarray is an object of data-type object (dtype).

5- Create array

numpy. array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

object: any object exposing the array interface method returns an array, or any (nested) sequence.

dtype: optional, describe data type of array.

copy: optional by default True, the object is copied.

order: C (row major) or F (column major) or Any (default)

subok: By default, returned array forced to be base class array. If true, sub-classes passed through.

ndmin: specifics minimum dimension of resultant array.

Example:

```
import numpy as np
a=np.array([1,9,5])
print(a)
```

Out: [1 9 5]

Example:

```
#minimum dimensions
import numpy as np
a=np.array([1,9,5,5,4,3],ndmin=3)
print(a)
```

Out: `[[[1 9 5 5 4 3]]]`

Example:

```
#dtype parameter
import numpy as np
a=np.array([1,9,5],dtype=complex)
print(a)
```

Out: [1.+0.j 9.+0.j 5.+0.j]

6- Data Types

Different scalar data types defined in NumPy:

Data Types	Description
bool_	Boolean stored as a byte
int_	Default integer type
unit8, unit16, unit32, unit64	Unsigned integer
float_	Shorthand for float 64
complex_	Shorthand for complex128

7- Array attribute of NumPy

7-1: ndarray.shape: return a tuple consisting of array dimensions, and also used to resize the array.

```
import numpy as np
data=np.array([[1,2,3],[4,5,6]])
print(data.shape)
```

Out: (2, 3)

```
import numpy as np
data=np.array([[1,2,3],[4,5,6]])
data.shape=(3,2)
print(data)
```

Out:

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

Also Numpy Provides a reshape function to resize an arr ay.

```
import numpy as np
data=np.array([[1,2,3],[4,5,6]])
new_data=data.reshape(3,2)
print(new_data)
```

Out:

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

7-2: ndarray.ndim: number of array dimension

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

Out:

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

```
import numpy as np
data2=np.zeros((2,4,4))
print(data2)
print(data2.ndim)
```

Numpy Review (Part 1)

Zohreh Karimi

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

 [[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]]
3
```

Out:

7-3: numpy.empty(): it creates an uninitialized array of specified shape and dtype.

numpy.empty(shape, dtype = float, order = 'C')

shape: shape of an empty array in int or tuple of int

dtype: desired output datatype (optional)

order: 'C' for C-style and 'F' for FORTRAN-style

```
import numpy as np
data=np.empty([3,2],dtype=int)
print(data)
```

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

Out:

7-4: numpy.zeros: return a new array of specified size, filled with zeros.

numpy.zeros(shape, dtype = float, order = 'C')

```
import numpy as np
data=np.zeros(4)
print(data)
```

Out: [0. 0. 0. 0.]

```
import numpy as np
data=np.zeros(5,dtype=int)
print(data)
```

Out: [0 0 0 0 0]

7-5: numpy.ones: return a new array of specified size and type, filled with ones.

numpy.ones(shape, dtype = None, order = 'C')

```
import numpy as np
data=np.ones((2,5),dtype=int)
print(data)
```

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

Out:

7-6:numpy.arange(): returns an ndarray object containing evenly spaced values within a given range.

numpy.arange(start, stop, step, dtype)

```
import numpy as np
data=np.arange(0,20,2)
print(data)
```

Out: `[ 0  2  4  6  8 10 12 14 16 18]`

```
import numpy as np
data=np.arange(0,20,2, dtype=float)
print(data)
```

Out:

`[ 0.  2.  4.  6.  8. 10. 12. 14. 16. 18.]`

7-7: numpy.linspace():it is similar to arange() function. in this function, instead of step size, the number of evenl y spaced values between the interval is specified.

```
import numpy as np
data=np.linspace(10,20,5, dtype=int, endpoint= False)
print(data)
```

Out: `[10 12 14 16 18]`

7-8: numpy.logspace(): returns an ndarray object that co ntains the numbers that are evenly spaced on log scale.

numpy.logspace(start, stop, num, endpoint, base, dtype)

```
import numpy as np
data=np.logspace(10,20,5, base=2, endpoint= False)
print(data)
```

Out:

`[  1024.   4096.  16384.  65536. 262144.]`

8-indexing and slicing

Contents of ndarray object can be accessed and modified by indexing or slicing. Three types of indexing methods are available.

8-1: Basic slicing: a python slice object is constructed b y giving start, stop and step parameters.

```
import numpy as np
data=np.arange(10)
data_slice=slice(3,10,2)
print(data[data_slice])
```

Out: [3 5 7 9]

```
import numpy as np
data=np.arange(10)
data_slice=data[2:7:2]
print(data[data_slice])
```

Out: [2 4 6]

8-2: advanced indexing:

There are two types of advanced indexing: **Integer** and **Boolean.**

8-2-1: Integer indexing: helps in selecting any arbitrary item in an array based on its Ndimensional index.

```
import numpy as np
x=np.array([[6,2],[5,9],[3,7]])
y=x[[0,1,2],[0,1,0]]
print(y)
```

Out: [6 9 3]

Advanced and Basic indexing can be combined by using one slice (:) or ellipsis (…) with an index array.

```
import numpy as np
x=np.array([[0,1,2],[3,4,5],[6,7,8],[9,10,11]])
z=x[1:4,1:3]
print(z)
```

```
[[ 4  5]
 [ 7  8]
 [10 11]]
```

Out:

Numpy Review (Part 1)

Zohreh Karimi

8-2-2: Boolean array indexing: it is used when the resultant object is meant to be the result of Boolean operations.

```
import numpy as np
x=np.array([[0,1,2],[3,4,5],[6,7,8],[9,10,11]])
print(x[x>5])
```

Out:
```
[ 6  7  8  9 10 11]
```

```
import numpy as np
a=np.array([np.nan,1,2,np.nan,3,4,])
print(a[~np.isnan(a)])
```

Out: `[1. 2. 3. 4.]`

9- Broadcasting

Broadcasting refers to the ability of NumPy to treat arrays of different shapes during arithmetic operations.

```
import numpy as np
a=np.array([1,2,3,4])
b=np.array([2,1,2,2])
c=a*b
print(c)
```

Out: [2 2 6 8]

If the dimensions of two arrays are dissimilar, element-to-element operations are not possible.

```
import numpy as np
a=np.array([[1,2,3,4,5],[5,6,3,2,1],[2,5,6,2,7]])
b=np.array([2,1,2,2,4])
c=a+b
print('a :\n', a)
print('b:\n', b)
print('a+b:\n ',c)
```

Out:
```
a :
 [[1 2 3 4 5]
 [5 6 3 2 1]
 [2 5 6 2 7]]
b:
 [2 1 2 2 4]
a+b:
  [[ 3  3  5  6  9]
 [ 7  7  5  4  5]
 [ 4  6  8  4 11]]
```

10- Iterating over array

Iterator object in NumPy **is numpy.nditer.** It is a multidimensional iterator object using which it is possible to iterate over an array.

```
import numpy as np
a=np.arange(0,60,5)
b=a.reshape(3,4)
print(b)
for x in np.nditer(b):
    print(x)
```

Out:
```
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]
0
5
10
15
20
25
30
35
40
45
50
55
```

C-style order and F-style order

```
import numpy as np
data=np.arange(0,60,5)
data=data.reshape(3,4)
print('original data are:\n',data,'\n')

transpose_data=data.T
print('Transpose data are:\n',transpose_data,'\n')

c_style_order=transpose_data.copy(order='C')
print('Sorted in C-style order:\n',c_style_order,'\n')
for x in np.nditer(c_style_order):
    print(x,end=' ')
print('\n')

f_style_order=transpose_data.copy(order='F')
print('Sorted in F-style order:\n',f_style_order,'\n')
for x in np.nditer(f_style_order):
    print(x,end=' ')
```

Out:
```
original data are:
 [[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

Transpose data are:
 [[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]

Sorted in C-style order:
 [[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]

0 20 40 5 25 45 10 30 50 15 35 55

Sorted in F-style order:
 [[ 0 20 40]
 [ 5 25 45]
 [10 30 50]
 [15 35 55]]

0 5 10 15 20 25 30 35 40 45 50 55
```

Numpy Review (Part 2)

Zohreh Karimi

11- Manipulation

For manipulation of elements in ndarray there are several methods, Here are some common methods:

11-1: changing shape:

- numpy.reshape(arr, newshape, order')
- numpy.flat[]
- numpy.ravel(a, order)
- ndarray.flatten(order)

```
import numpy as np
data=np.arange(6)
print(data.reshape(2,3))
```

Out:
```
[[0 1 2]
 [3 4 5]]
```

```
import numpy as np
data=np.arange(6)
new_data=data.reshape(2,3)
print(new_data,'\n')
print(new_data.flat[0:6])
```

Out:
```
[[0 1 2]
 [3 4 5]]

[0 1 2 3 4 5]
```

```
import numpy as np
data=np.arange(6)
new_data=data.reshape(2,3)
print(new_data,'\n')
print(new_data.ravel())
```

Out:
```
[[0 1 2]
 [3 4 5]]

[0 1 2 3 4 5]
```

```
import numpy as np
data=np.arange(8)
new_data=data.reshape(2,4)
print(new_data)
print(new_data.flatten(),'\n')
print('F order Type: ',new_data.flatten(order='F'))
print('C order Type: ',new_data.flatten(order='C'))
```

```
[[0 1 2 3]
 [4 5 6 7]]
[0 1 2 3 4 5 6 7]

F order Type:  [0 4 1 5 2 6 3 7]
C order Type:  [0 1 2 3 4 5 6 7]
```

11-2: Transpose Operations:

- numpy.transpose(arr, axes)
- numpy.transpose
- numpy.rollaxis(arr, axis, start)
- numpy.swapaxes(arr, axis1, axis2)

```
import numpy as np
data=np.arange(12)
new_data=data.reshape(3,4)
print('Output:\n','new_data:\n',new_data)
print('Transpose is:\n',np.transpose(new_data))
```

```
Output:
 new_data:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
Transpose is:
 [[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
```

```
import numpy as np
data=np.arange(12)
new_data=data.reshape(3,4)
print('Output:\n','new_data:\n',new_data)
print('Transpose is:\n',new_data.T)
```

```
Output:
 new_data:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
Transpose is:
 [[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
```

```
import numpy as np
data=np.arange(12)
new_data=data.reshape(2,3,2)
print('Output:\n','new_data:\n',new_data,'\n')
print('Transpose in row is:\n',np.rollaxis(new_data,2,0))
print('Transpose in column is:\n',np.rollaxis(new_data,2,1))
```

```
Output:
 new_data:
 [[[ 0  1]
  [ 2  3]
  [ 4  5]]

 [[ 6  7]
  [ 8  9]
  [10 11]]]

Transpose in row is:
 [[[ 0  2  4]
  [ 6  8 10]]

 [[ 1  3  5]
  [ 7  9 11]]]
Transpose in column is:
 [[[ 0  2  4]
  [ 1  3  5]]

 [[ 6  8 10]
  [ 7  9 11]]]
```

```
import numpy as np
data=np.arange(12)
new_data=data.reshape(2,3,2)
print('Output:\n','new_data:\n',new_data,'\n')
print('swap in column is:\n',np.swapaxes(new_data,2,0))
print('swap in row is:\n',np.swapaxes(new_data,2,1))
```

Numpy Review (Part 2)

Zohreh Karimi

```
Output:
 new_data:
 [[[ 0  1]
  [ 2  3]
  [ 4  5]]

 [[ 6  7]
  [ 8  9]
  [10 11]]]

swap in column is:
 [[[ 0  6]
  [ 2  8]
  [ 4 10]]

 [[ 1  7]
  [ 3  9]
  [ 5 11]]]
swap in row is:
 [[[ 0  2  4]
  [ 1  3  5]]

 [[ 6  8 10]
  [ 7  9 11]]]
```

11-3: changing dimension

- numpy.squeeze(arr, axis)
- numpy.expand_dims(arr, axis)
- numpy.broadcast_to(array, shape, subok)

```
import numpy as np
data=np.arange(4).reshape(1,4)
print(f'data is:\n{data}','\n')
print(f'broadcast :\n {np.broadcast_to(data,(4,4))}')
```

```
data is:
[[0 1 2 3]]

broadcast :
 [[0 1 2 3]
 [0 1 2 3]
 [0 1 2 3]
 [0 1 2 3]]
```

Out:

```
import numpy as np
x=np.array(([1,2],[3,4]))
print(f'Array X is:\n {x}')
y=np.expand_dims(x,axis=0)
print(f'Array y with axis=0 is:\n {y}')
print('The Shape of X and Y: ',x.shape,y.shape)
y=np.expand_dims(x,axis=1)
print(f'Array y with axis= 1 is:\n {y}')
print('The Shape of X and Y: ',x.shape,y.shape)
```

Out:

```
Array X is:
 [[1 2]
 [3 4]]
Array y with axis=0 is:
 [[[1 2]
  [3 4]]]
The Shape of X and Y:  (2, 2) (1, 2, 2)
Array y with axis= 1 is:
 [[[1 2]]

 [[3 4]]]
The Shape of X and Y:  (2, 2) (2, 1, 2)
```

```
import numpy as np
x=np.arange(9).reshape(1,3,3)
print(f'Array X is:\n {x}')
y=np.squeeze(x)
print(f'Array Y is:\n {y}')
print('The Shape of X and Y: ',x.shape,y.shape)
```

Out:

```
Array X is:
 [[[0 1 2]
  [3 4 5]
  [6 7 8]]]
Array Y is:
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
The Shape of X and Y:  (1, 3, 3) (3, 3)
```

11-4: Joining Arrays

- numpy.concatenate((a1, a2, ...), axis)
- numpy.stack(arrays, axis)
- numpy.hstack(x,y)
- numpy.vstack(x,y)

```
import numpy as np
x=np.array([[1,2],[3,4]])
print(f'First Array is: \n {x}')
y=np.array([[5,6],[8,9]])
print(f'Second Array is : \n {y}')
print(f' Joining two arrays is: \n {np.concatenate((x,y))}')
```

```
First Array is:
 [[1 2]
 [3 4]]
Second Array is :
 [[5 6]
 [8 9]]
Joining two arrays is:
 [[1 2]
 [3 4]
 [5 6]
 [8 9]]
```

```
import numpy as np
x=np.array([[1,2],[3,4]])
print(f'First Array is: \n {x}')
y=np.array([[5,6],[8,9]])
print(f'Second Array is : \n {y}')
print(f' Stack the two arrays along axis 0 is: \n {np.stack((x,y),0)}')
print(f' Stack the two arrays along axis 1 is: \n {np.stack((x,y),1)}')
```

```
First Array is:
 [[1 2]
 [3 4]]
Second Array is :
 [[5 6]
 [8 9]]
Stack the two arrays along axis 0 is:
 [[[1 2]
  [3 4]]

 [[5 6]
  [8 9]]]
Stack the two arrays along axis 1 is:
 [[[1 2]
  [5 6]]

 [[3 4]
  [8 9]]]
```

Numpy Review (Part 2)

Zohreh Karimi

```
import numpy as np
x=np.array([[1,2],[3,4]])
print(f'First Array is: \n {x}')
y=np.array([[5,6],[7,8]])
print(f'Second Array is : \n {y}')
print(f' Horizontal stacking is: \n {np.hstack((x,y))}')
```

```
First Array is:
 [[1 2]
 [3 4]]
Second Array is :
 [[5 6]
 [7 8]]
 Horizontal stacking is:
 [[1 2 5 6]
 [3 4 7 8]]
```

```
import numpy as np
x=np.array([[1,2],[3,4]])
print(f'First Array is: \n {x}')
y=np.array([[5,6],[7,8]])
print(f'Second Array is : \n {y}')
print(f' Vertical stacking is: \n {np.vstack((x,y))}')
```

```
First Array is:
 [[1 2]
 [3 4]]
Second Array is :
 [[5 6]
 [7 8]]
 Vertical stacking is:
 [[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

11-5: Splitting Arrays

- numpy.split(ary, indices_or_sections, axis)
- numpy.hsplit( )
- numpy.vsplit( )

```
import numpy as np
x=np.arange(9)
print(f'First Array is:\n {x}')
y=np.split(x,3)
print(f'Split the Array in 3 equal size subarrays:\n {y}')
print(f'Split the Array at positions indicated in 1-D Array: {np.split(x,[4,7])}')
```

```
First Array is:
 [0 1 2 3 4 5 6 7 8]
Split the Array in 3 equal size subarrays:
 [array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8])]
Split the Array at positions indicated in 1-D Array: [array([0, 1, 2, 3]), array([4, 5, 6]), array
([7, 8])]
```

```
import numpy as np
data=np.arange(16).reshape(4,4)
print(f'data is: \n {data}')
hsplit=np.hsplit(data,2)
print(f'horizontal spliting data is:\n {hsplit}')
```

Out:

```
data is:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
horizontal spliting data is:
 [array([[ 0,  1],
        [ 4,  5],
        [ 8,  9],
        [12, 13]]), array([[ 2,  3],
        [ 6,  7],
        [10, 11],
        [14, 15]])]
```

```
import numpy as np
data=np.arange(16).reshape(4,4)
print(f'data is: \n {data}')
vsplit=np.vsplit(data,2)
print(f'vertical spliting data is:\n {vsplit}')
```

Out:

```
data is:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
vertical spliting data is:
 [array([[0, 1, 2, 3],
        [4, 5, 6, 7]]), array([[ 8,  9, 10, 11],
        [12, 13, 14, 15]])]
```

11-6: Adding / Removing Elements

- numpy.append(arr, values, axis)
- numpy.insert(arr, obj, values, axis)
- numpy.delete(arr, obj, axis)
- numpy.unique(arr, return_index, return_inverse, return_counts)
- numpy.resize(arr, shape)

```
import numpy as np
data=np.array([[1,2,3],[4,5,6]])
print(f'First Array: \n {data}')
print(f'Append element to array:\n{np.append(data,[[8,6,7]])}')
```

Out:

```
First Array:
 [[1 2 3]
 [4 5 6]]
Append element to array:
[1 2 3 4 5 6 8 6 7]
```

```
import numpy as np
data=np.array([[1,2,3],[4,5,6]])
print(f'First Array: \n {data}')
print(f'Append element to array:\n{np.append(data,[[8,6,7]],axis=0)}')
```

Out:

```
First Array:
 [[1 2 3]
 [4 5 6]]
Append element to array:
[[1 2 3]
 [4 5 6]
 [8 6 7]]
```

```
import numpy as np
data=np.array([[1,2,3],[4,5,6]])
print(f'First Array: \n {data}')
print(f'Append element to array:\n{np.append(data,[[8,6,7],[4,5,6]],axis=1)}')
```

Out:

```
First Array:
 [[1 2 3]
 [4 5 6]]
Append element to array:
[[1 2 3 8 6 7]
 [4 5 6 4 5 6]]
```

Numpy Review (Part 2)

Zohreh Karimi

```
import numpy as np
data=np.array([[1,2],[3,4],[5,6]])
print(f'First Array:\n {data}')
print(f'Array after insertion \n {np.insert(data,2,[10,12])}')
```

```
First Array:
 [[1 2]
 [3 4]
 [5 6]]
Array after insertion
 [ 1  2 10 12  3  4  5  6]
```

Out:

```
import numpy as np
data=np.array([[1,2],[3,4],[5,6]])
print(f'First Array:\n {data}')
print(f'Array after insertion \n {np.insert(data,0,[10],axis=0)}')
```

```
First Array:
 [[1 2]
 [3 4]
 [5 6]]
Array after insertion
 [[10 10]
 [ 1  2]
 [ 3  4]
 [ 5  6]]
```

```
import numpy as np
data=np.array([[1,2],[3,4],[5,6]])
print(f'First Array:\n {data}')
print(f'Array after insertion \n {np.insert(data,0,[10],axis=1)}')
```

```
First Array:
 [[1 2]
 [3 4]
 [5 6]]
Array after insertion
 [[10  1  2]
 [10  3  4]
 [10  5  6]]
```

```
import numpy as np
data=np.array([[1,2,6],[3,4,5]])
print(f'First Array is:\n{data}')
print(f'data size is: {data.shape}')
data_size=np.resize(data,(3,2))
print(f'Array after resize:\n {data_size}')
```

```
First Array is:
[[1 2 6]
 [3 4 5]]
data size is: (2, 3)
Array after resize:
 [[1 2]
 [6 3]
 [4 5]]
```

```
import numpy as np
data=np.array([[1,2,6],[3,4,5]])
print(f'First Array is:\n {data}')
print(f'data size is: {data.shape}')
data_size=np.resize(data,(5,4))
print(f'Array after resize:\n {data_size}')
```

```
First Array is:
 [[1 2 6]
 [3 4 5]]
data size is: (2, 3)
Array after resize:
 [[1 2 6 3]
 [4 5 1 2]
 [6 3 4 5]
 [1 2 6 3]
 [4 5 1 2]]
```

```
import numpy as np
data=np.arange(12).reshape(4,3)
print(f'First Array:\n {data}')
print(f'Array after delete as axis not used: {np.delete(data,5)}')
```

```
First Array:
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
Array after delete as axis not used: [ 0  1  2  3  4  6  7  8  9 10 11]
```

```
import numpy as np
data=np.arange(12).reshape(4,3)
print(f'First Array:\n {data}')
print(f'Array after delete as axis not used: \n {np.delete(data,2,axis=0)}')
```

```
First Array:
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
Array after delete as axis not used:
 [[ 0  1  2]
 [ 3  4  5]
 [ 9 10 11]]
```

```
import numpy as np
data=np.arange(12).reshape(4,3)
print(f'First Array:\n {data}')
print(f'Array after delete as axis not used: \n {np.delete(data,2,axis=1)}')
```

```
First Array:
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
Array after delete as axis not used:
 [[ 0  1]
 [ 3  4]
 [ 6  7]
 [ 9 10]]
```

```
import numpy as np
data=np.array([5,2,6,8,4,3,6,2,7,2,1,6,9])
print(f'First Array is:\n {data}')
print(f'Unique values of First Array is:\n {np.unique(data)}')
```

```
First Array is:
 [5 2 6 8 4 3 6 2 7 2 1 6 9]
Unique values of First Array is:
 [1 2 3 4 5 6 7 8 9]
```

Out:

```
import numpy as np
data=np.array([5,2,6,8,4,3,6,2,7,2,1,6,9])
print(f'First Array is:\n {data}')
print(f'Unique values of First Array is:\n {np.unique(data)}')
u,indices=np.unique(data,return_index=True)
print(f'index of unique value on First Array is:\n {indices}')
u,indices=np.unique(data,return_counts=True)
print(f'Return the count of repetitions of unique value is:\n {indices}')
```

```
First Array is:
 [5 2 6 8 4 3 6 2 7 2 1 6 9]
Unique values of First Array is:
 [1 2 3 4 5 6 7 8 9]
index of unique value on First Array is:
 [10  1  5  4  0  2  8  3 12]
Return the count of repetitions of unique value is:
 [1 3 1 1 1 3 1 1 1]
```

Numpy Review (Part 3)

Zohreh Karimi

12- NumPy Binary Operators

12-1: bitwise AND operation: on the corresponding bits of binary representations of integers in input arrays is computed by np.bitwise_and() function.

```
import numpy as np
print('Binary equivalent of 14 and 10:\n')
x,y=14,10
print(f'bin x is : {bin(x)} and bin y is : {bin(y)}')
print(f'Bitwise AND of 14 and 15: {np.bitwise_and(14,10)}')
```

```
Binary equivalent of 14 and 10:

bin x is : 0b1110 and bin y is : 0b1010
Bitwise AND of 14 and 15: 10
```

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

12-2: bitwise OR operation: on the corresponding bits of binary representations of integers in input array is computed by np.bitwise_or() function.

```
import numpy as np
print('Binary equivalent of 14 and 10:\n')
x,y=14,10
print(f'bin x is : {bin(x)} and bin y is : {bin(y)}')
print(f'Bitwise OR of 14 and 15: {np.bitwise_or(14,15)}')
```

Out:

```
Binary equivalent of 14 and 10:

bin x is : 0b1110 and bin y is : 0b1010
Bitwise OR of 14 and 15: 15
```

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

12-3: bitwise NOT operation: np.invert() result on integers in the input array. For signed integers, two's complement is returned.

```
import numpy as np
print(f'Invert of 13 where dtype of ndarray is uint8:')
print(np.invert(np.array([13],dtype=np.uint8)))
print(f'Binary representaion of 13 is: {np.binary_repr(13,width=8)}')
print(f'Binary representation of 242 is: {np.binary_repr(242,width=8)}')
```

Out:

```
Invert of 13 where dtype of ndarray is uint8:
[242]
Binary representaion of 13 is: 00001101
Binary representation of 242 is: 11110010
```

12-4: numpy.left_shift(): this function shifts the bits in binary representation of an array element to the left. Equal number of 0s are appended from the right.

```
import numpy as np
print(f'Left shift of 10 by two positions : {np.left_shift(10,2)}')
print(f'Binary representation of 10 is : {np.binary_repr(10,width=8)}')
print(f'Binary representation of 40 is : {np.binary_repr(40,width=8)}')
```

Out:

```
Left shift of 10 by two positions : 40
Binary representation of 10 is : 00001010
Binary representation of 40 is : 00101000
```

12-5: numpy.right_shift(): this function shift the bits in the binary representation of an array element to the right by specified positions, and an equal number of 0s are appended from the left.

```
import numpy as np
print(f'Right shift of 10 by two positions : {np.right_shift(10,2)}')
print(f'Binary representation of 10 is : {np.binary_repr(10,width=8)}')
print(f'Binary representation of 2 is : {np.binary_repr(2,width=8)}')
```

Out:

```
Right shift of 10 by two positions : 2
Binary representation of 10 is : 00001010
Binary representation of 2 is : 00000010
```

13- NumPy String Function

13-1: numpy.char.add()

```
import numpy as np
print('concatinate two strings:')
print(np.char.add(['hello '],['every body']))
```

Out:

```
concatinate two strings:
['hello every body']
```

13-2: numpy.char.multiply(): this function performs multiple concatenation.

```
import numpy as np
print(np.char.multiply('hello Freinds ',3))
```

Out:

```
hello Freinds hello Freinds hello Freinds
```

13-3: numpy.char.center(arr,width,fillchar): this function returns an array of the required width so that the input string is centered and padded on the left and right with fillchar.

```
import numpy as np
print(np.char.center('Hello Freinds',20,fillchar='*'))
```

Out:

```
***Hello Freinds****
```

13-4: numpy.char.capitalize(): this function returns the copy of the string with the first letter capitalized.

```
import numpy as np
print(np.char.capitalize('hello, i live here'))
```

Out:

```
Hello, i live here
```

13-5: numpy.char.title(): this function returns a title cased version of input string with the first letter of each word capitalized.

```
import numpy as np
print(np.char.title('hello, i lived here'))
```

```
Hello, I Lived Here
```

13-6: numpy.char.lower(): this function returns an array with elements converted to lower case.

```
import numpy as np
print(np.char.lower('Hello, I Live Here'))
```

Out:

```
hello, i live here
```

13-7: numpy.char.upper(): this function on each element in an array to return the uppercase array elements.

```
import numpy as np
print(np.char.upper('hello, i live here'))
```

Numpy Review (Part 3)

Zohreh Karimi

Out:
```
HELLO, I LIVE HERE
```

13-8: numpy.char.split(): this function return a list of words in the input string.

```
import numpy as np
print(np.char.split('hello ,i live here'))
```

Out:
```
['hello', ',i', 'live', 'here']
```

```
import numpy as np
print(np.char.split('hello ,i live here',sep=','))
```

Out: `['hello ', 'i live here']`

13-9: numpy.char.splitlines(): this function returns a list of elements in the array, breaking at line boundaries.

```
import numpy as np
print(np.char.splitlines('hello\n i live here'))
```

Out: `['hello', ' i live here']`

13-10: numpy.char.strip(): this function returns a copy of array with elements stripped of the specified characters leading and/or trailing in it.

```
import numpy as np
print(np.char.strip(['arrive','agile'],'a'))
```

Out: `['rrive' 'gile']`

13-11: numpy.char.join(): this method returns a string in which the individual characters are joined by separator character specified.

```
import numpy as np
print(np.char.join(': ','dmy'))
```

Out:
```
d: m: y
```

13-12: numpy.char.replace(): this function returns a new copy of the input string in which all occurrences of the sequence of characters is replaced by another given sequence.

```
import numpy as np
print(np.char.replace('she is a good girl','is','was'))
```

Out: `she was a good girl`

13-13: numpy.char.decode(): this function decodes the given string using the specified codec.

```
import numpy as np
a=np.char.encode('hello','cp500')
print(np.char.decode(a,'cp500'))
```

Out: hello

14- Mathematical Function

14-1: Trigonometric Functions:

-numpy.sin()

-numpy.cos()

-numpy.tan()

```
import numpy as np
print(f'sin(90)is :{np.sin(90)}')
print(f'cos(90)is :{np.cos(90)}')
print(f'tan(90)is :{np.tan(90)}')
```

Out:

```
sin(90)is :0.8939966636005579
cos(90)is :-0.4480736161291701
tan(90)is :-1.995200412208242
```

14-2: numpy.around(a,decimals): this function returns the value rounded to the desired precision.

```
import numpy as np
data=np.array([1.05,2.2235,0.6587,1.268])
print(np.around(data,2))
```

Out: `[1.05 2.22 0.66 1.27]`

14-3: numpy.floor(): this function returns the largest integer not greater than the input parameter. The floor of the scaler x is the largest integer I, such that I <=x

```
import numpy as np
data=np.array([-1.6,0.5348,2.3568,7.89,-4.05])
print(np.floor(data))
```

Out:
```
[-2.  0.  2.  7. -5.]
```

14-4: numpy.ceil(): this function returns the ceiling of an input value, i.e. the ceil of the scaler x is the smallest integer I, such that I >=x

```
import numpy as np
data=np.array([-1.6,0.5348,2.3568,7.89,-4.05])
print(np.ceil(data))
```

Out: `[-1.  1.  3.  8. -4.]`

15- Arithmetic operations

- numpy.add(a,b)
- numpy.subtract(a,b)
- numpy.multiply(a,b)
- numpy.divide(a,b)
- numpy.reciprocal()
- numpy.power()
- numpy.mod()

```
import numpy as np
x=np.array([[4,5,6],[3,2,3]])
y=np.array([8,5,4])
print(f'Result of Add two arrays :\n {np.add(x,y)}')
```

Out:
```
Result of Add two arrays :
 [[12 10 10]
 [11  7  7]]
```

```
import numpy as np
x=np.array([[4,5,6],[3,2,3]])
y=np.array([8,5,4])
print(f'Result of Subtract two arrays :\n {np.subtract(x,y)}')
```

Out:
```
Result of Subtract two arrays :
 [[-4  0  2]
 [-5 -3 -1]]
```

```
import numpy as np
x=np.array([[4,5,6],[3,2,3]])
y=np.array([8,5,4])
print(f'Result of Multiply two arrays :\n {np.multiply(x,y)}')
```

Out:
```
Result of Multiply two arrays :
 [[32 25 24]
 [24 10 12]]
```

```
import numpy as np
x=np.array([[4,5,6],[3,2,3]])
y=np.array([8,5,4])
print(f'Result of divide two arrays :\n {np.divide(x,y)}')
```

Numpy Review (Part 3)

Zohreh Karimi

```
Result of divide two arrays :
 [[0.5   1.    1.5  ]
 [0.375 0.4   0.75 ]]
```

Out:

```
import numpy as np
data=np.arange(3)
print(f'output of power() is:\n {np.power(data,3)}')
```

```
output of power() is:
 [0 1 8]
```

Out:

```
import numpy as np
data=np.array([3,9,12])
print(f'output of mod() is:\n {np.mod(data,2)}')
```

```
output of mod() is:
 [1 1 0]
```

Out:

15-1: Arithmetic operations with complex numbers

- numpy.real()
- numpy.imag()
- numpy.conj()
- numpy.angle()

```
import numpy as np
data=np.array([-5.6j,0.2j,15.,2+3j])
print(f'Original array is:\n {data}')
print(f'After applying real() function:\n {np.real(data)}')
print(f'After applying imag( function:\n {np.imag(data)})')
print(f'After applying conj function:\n {np.conj(data)}')
print(f'After applying angle function:\n {np.angle(data)}')
```

Out:

```
Original array is:
 [-0.-5.6j  0.+0.2j 15.+0.j   2.+3.j ]
After applying real() function:
 [-0.  0. 15.  2.]
After applying imag( function:
 [-5.6  0.2  0.   3. ])
After applying conj function:
 [-0.+5.6j  0.-0.2j 15.-0.j   2.-3.j ]
After applying angle function:
 [-1.57079633  1.57079633  0.          0.98279372]
```

16- NumPy Statistical Functions

- numpy.amin()
-numpy.amax()
-numpy.ptp()
-numpy.percentile(a,q,axis)
-numpy.median()
-numpy.mean()
-numpy.average()
- np.std()
-np.var()

```
import numpy as np
data=np.array([[1,5,9],[6,54,25],[9,8,65],[35,24,0]])
print(f'min of data is: {np.amin(data,1)}')
```

Out:

```
min of data is: [1 6 8 0]
```

```
import numpy as np
data=np.array([[1,5,9],[6,54,25],[9,8,65],[35,24,0]])
print(f'max of data is: {np.amax(data,1)}')
```

Out:

```
max of data is: [ 9 54 65 35]
```

The **numpy.ptp()** function returns the range (maximum-minimum) of values along an axis

```
import numpy as np
data=np.array([[1,5,9],[6,54,25],[9,8,65],[35,24,0]])
print(f'data is:\n{data}')
print(f'ptp of data  in axis =1 is: {np.ptp(data,axis=1)}')
print(f'ptp of data  in axis =0 is: {np.ptp(data,axis=0)}')
```

Out:

```
data is:
[[ 1  5  9]
 [ 6 54 25]
 [ 9  8 65]
 [35 24  0]]
ptp of data  in axis =1 is: [ 8 48 57 35]
ptp of data  in axis =0 is: [34 49 65]
```

```
import numpy as np
data=np.array([[1,5,9],[6,54,25],[9,8,65],[35,24,0]])
print(f'data is:\n{data}')
print(f'percentile of data  in axis =1 is: {np.percentile(data,25,axis=1)}')
print(f'percentile of data  in axis =0 is: {np.percentile(data,25,axis=0)}')
```

Out:

```
data is:
[[ 1  5  9]
 [ 6 54 25]
 [ 9  8 65]
 [35 24  0]]
percentile of data  in axis =1 is: [ 3.  15.5  8.5 12. ]
percentile of data  in axis =0 is: [4.75 7.25 6.75]
```

```
import numpy as np
data=np.array([[1,5,9],[6,54,25],[9,8,65],[35,24,0]])
print(f'data is:\n{data}')
print(f'median of total data is : {np.median(data)}')
print(f'median of data  in axis =1 is: {np.median(data,axis=1)}')
print(f'median of data  in axis =0 is: {np.median(data,axis=0)}')
```

Out:

```
data is:
[[ 1  5  9]
 [ 6 54 25]
 [ 9  8 65]
 [35 24  0]]
median of total data is : 9.0
median of data  in axis =1 is: [ 5. 25.  9. 24.]
median of data  in axis =0 is: [ 7.5 16.  17. ]
```

```
import numpy as np
data=np.array([[1,5,9],[6,54,25],[9,8,65],[35,24,0]])
print(f'data is:\n{data}')
print(f'mean of total data is : {np.mean(data)}')
print(f'mean of data  in axis =1 is: {np.mean(data,axis=1)}')
print(f'mean of data  in axis =0 is: {np.mean(data,axis=0)}')
```

Out:

```
data is:
[[ 1  5  9]
 [ 6 54 25]
 [ 9  8 65]
 [35 24  0]]
mean of total data is : 20.083333333333332
mean of data  in axis =1 is: [ 5.         28.33333333 27.33333333 19.66666667]
mean of data  in axis =0 is: [12.75 22.75 24.75]
```

```
import numpy as np
data=np.array([[1,5,9],[6,54,25],[9,8,65],[35,24,0]])
print(f'data is:\n{data}')
print(f'average of total data is : {np.average(data)}')
print(f'average of data  in axis =1 is: {np.average(data,weights=[1,2,2],axis=1)}')
print(f'average of data  in axis =0 is: {np.average(data,weights=[1,2,2,2],axis=0)}')
```

Numpy Review (Part 3)

Zohreh Karimi

Out:

```
data is:
[[ 1  5  9]
 [ 6 54 25]
 [ 9  8 65]
 [35 24  0]]
average of total data is : 20.083333333333332
average of data  in axis =1 is: [ 5.8 32.8 31.  16.6]
average of data  in axis =0 is: [14.42857143 25.28571429 27.        ]
```

```
import numpy as np
data=np.array([[1,5,9],[6,54,25],[9,8,65],[35,24,0]])
print(f'data is:\n{data}')
print(f'standard deviation of total data is : {np.std(data)}')
print(f'variance of total data is : {np.var(data)}')
```

Out:

```
data is:
[[ 1  5  9]
 [ 6 54 25]
 [ 9  8 65]
 [35 24  0]]
standard deviation of total data is : 20.44284036581566
variance of total data is : 417.9097222222222
```

16- NumpPy-Sort, Search&Count Functions

- numpy.sort(a, axis, kind, order):

kind: default is quick sort.

```
import numpy as np
data=np.array([[69,1,21],[32,58,47],[8,9,2]])
print(f'data is :\n {data}')
print(f'Apply sort Function:\n{np.sort(data)}')
print(f'Apply sort Function in axis=0:\n{np.sort(data,axis=0)}')
print(f'Apply sort Function in axis=1:\n{np.sort(data,axis=1)}')
```

```
data is :
 [[69  1 21]
 [32 58 47]
 [ 8  9  2]]
Apply sort Function:
[[ 1 21 69]
 [32 47 58]
 [ 2  8  9]]
Apply sort Function in axis=0:
[[ 8  1  2]
 [32  9 21]
 [69 58 47]]
Apply sort Function in axis=1:
[[ 1 21 69]
 [32 47 58]
 [ 2  8  9]]
```

Out:

- numpy.argsort(a, axis, kind, order): this function performs an indirect sort on input array,along the given axis and a specified kind of sort.

```
import numpy as np
data=np.array([3,2,1])
print(f'data is:\n{data}')
print(f'argsort of data is:\n{np.argsort(data)}')
```

```
data is:
[3 2 1]
argsort of data is:
[2 1 0]
```

Out:

- numpy.lexsort(): function performs an indirect sort using a sequence of keys. the keys can be seen as a column in a spreadsheet. The function returns an array of indices, using which the sorted data can be obtained.

```
import numpy as np
x=np.array([1,2,3,4])
y=np.array([3,2,9,7])
print(f'lexsort(x,y) is:{np.lexsort((x,y))}')
print(f'lexsort(y,x) is:{np.lexsort((y,x))}')
s1=('iran','is','in','tehran')
s2=('you','like','shiraz','ahvaz')
out=np.lexsort((s1,s2))
print(f'lexsort(s1,s2) is: {out}')
print([s1[i]for i in out])
```

```
lexsort(x,y) is:[1 0 3 2]
lexsort(y,x) is:[0 1 2 3]
lexsort(s1,s2) is: [3 1 2 0]
['tehran', 'is', 'in', 'iran']
```

Out:

- numpy.argmin()
- numpy.argmax()

```
import numpy as np
data=np.array([[1,30,25],[98,21,45],[21,3,85]])
print(f'data is: \n {data}')
print(f'argmin of data is:\n{np.argmin(data,axis=0)}')
print(f'argmax of data is:\n {np.argmax(data, axis=1)}')
```

```
data is:
 [[ 1 30 25]
 [98 21 45]
 [21  3 85]]
argmin of data is:
[0 2 0]
argmax of data is:
 [1 0 2]
```

Out:

- numpy.nonzero(): this finction returns the indices of non-zero elements in the input array.

```
import numpy as np
data=np.array([2,1,0])
print(np.nonzero(data))
```

Out: `(array([0, 1], dtype=int64),)`

- numpy.where(): this function returns the indices of elements in an input array where the given condition is satisfied.

```
import numpy as np
data=np.array([[1,2,9,98],[56,4,25,87],[32,21,89,14]])
print(f'data is:\n {data}')
y=np.where(data>10)
print(f'after condition, result is:\n{data[y]}')
```

Out:

```
data is:
 [[ 1  2  9 98]
 [56  4 25 87]
 [32 21 89 14]]
after condition, result is:
[98 56 25 87 32 21 89 14]
```

- numpy.extract(): this function returns the elements satisfying any condition.

```
import numpy as np
data=np.array([[1,2,6],[6,8,0],[2,9,11]])
condition=np.mod(data,2)==0
print(f'data is:\n{data}')
print(f'Extract elements using condition:\n{np.extract(condition,data)}')
```

```
data is:
[[ 1  2  6]
 [ 6  8  0]
 [ 2  9 11]]
Extract elements using condition:
[2 6 6 8 0 2]
```

Out:

Numpy Review (Part 5)

Zohreh Karimi

17- Numpy Byte Swapping

17-1: numpy.ndarray.byteswap(): this function toggles between the two representations.

```
import numpy as np
data=np.array([15, 256, 32],dtype=np.int16)
print(f'data is :\n {data}')
vhex = np.vectorize(hex)
hex_data=vhex(data)
print(f'Representation of data in memory in hexadecimal form:\n {hex_data}')
byteswap=data.byteswap(True)
print(f'Applying byteswap() function:\n{byteswap}')
hex_Data=vhex(byteswap)
print(f'Representation of data in memory in hexadecimal form:\n {hex_Data}')
```

Out:

```
data is :
 [ 15 256  32]
Representation of data in memory in hexadecimal form:
 ['0xf' '0x100' '0x20']
Applying byteswap() function:
[3840    1 8192]
Representation of data in memory in hexadecimal form:
 ['0xf00' '0x1' '0x2000']
```

18- Numpy Copies & Views

18-1: id(): simple assignments do not make the copy of array object. Instead, it uses the same id() of the original array to access it. Id() returns a universal identifier of Python object.

```
import numpy as np
x=np.arange(6)
print(f'origin data is: \n{x}')
y=x
print(f'x is assignment to y:\n{y}')
print('*****************************')
print(f'after Applying id() function id (x) is: {id(x)}')
print(f'after Applying id() function on y id(y) is: {id(y)}')
print('*****************************')
y.shape=3,2
print(f'change the shape of y:\n{y}')
print(f'shape of x also changed:\n {x}')
```

Out:

```
origin data is:
[0 1 2 3 4 5]
x is assignment to y:
[0 1 2 3 4 5]
*****************************
after Applying id() function id (x) is: 98589312
after Applying id() function on y id(y) is: 98589312
*****************************
change the shape of y:
[[0 1]
 [2 3]
 [4 5]]
shape of x also changed:
 [[0 1]
 [2 3]
 [4 5]]
```

18-2: ndarray.view(): Here new array object look at the same data of the original array.Unlike earlier case, change in dimensions of new array doesn't change dimensions of the original.

```
import numpy as np
x=np.arange(6).reshape(3,2)
print(f'origin data is:\n{x}')
y=x.view()
print(f'create view of x:\n{y}')
print(f'id of origin data is: {id(x)}')
print(f'id of new data is: {id(y)}')
y.shape=2,3
print(f'change the shape of y:\n{y}')
print(f'shape of origin data does not not changed:\n {x}')
```

Out:

```
origin data is:
[[0 1]
 [2 3]
 [4 5]]
create view of x:
[[0 1]
 [2 3]
 [4 5]]
id of origin data is: 98636864
id of new data is: 98638704
change the shape of y:
[[0 1 2]
 [3 4 5]]
shape of origin data does not not changed:
 [[0 1]
 [2 3]
 [4 5]]
```

18-3: ndarray.copy(): This function create a deep copy. It is a complete copy of the array and its data, and doesn't share with the origin array.

```
import numpy as np
x=np.array([[1,2],[4,5],[3,2]])
print(f'origin data is:\n{x}')
y=x.copy()
print(f'create a deep copy of x:\n {y}')
y[0,0]=10
print(f'change the content of y :\n{y}')
print(f'x remains unchanged:\n{x}')
```

```
origin data is:
[[1 2]
 [4 5]
 [3 2]]
create a deep copy of x:
 [[1 2]
 [4 5]
 [3 2]]
change the content of y :
[[10  2]
 [ 4  5]
 [ 3  2]]
x remains unchanged:
[[1 2]
 [4 5]
 [3 2]]
```

Out:

19- NumPy Matrix Library

numpy package contains a matrix library numpy.matlib. This module has functions that return matrices instead of ndarray objects.

19-1: numpy.matlib.empty(shape, dtype, order): This function returns a new matrix without initializing the entries.

```
import numpy.matlib
import numpy as np
print(np.matlib.empty((2,3 )))
```

Numpy Review (Part 5)

Zohreh Karimi

```
[[2.204e-321 0.000e+000 0.000e+000]
 [0.000e+000 0.000e+000 0.000e+000]]
```

Out:

19-2: numpy.matlib.zeros(): This function returns the matrix filled with zeros.

```
import numpy.matlib
import numpy as np
print(np.matlib.zeros((3,3)))
```

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

Out:

19-3: numpy.matlib.ones(): This function returns the matrix filled with 1s.

```
import numpy as np
import numpy.matlib
print(np.matlib.ones((3,3)))
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

Out:

19-4: numpy.matlib.eye(n,M,k,dtype):

n:the number of rows
M:the number of columns
k:index of diagonal
dtype:Data type of output
this function returns a matrix with 1 along the diagonal elements and zeros elsewhere.

```
import numpy as np
import numpy.matlib
print(np.matlib.eye(2,2,0,dtype=float))
```

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

Out:

19-5: numpy.matlib.identity(): this function returns the identity matrix of the given size. Identity matix is a square matrix with all diagonal elements as 1.

```
import numpy as np
import numpy.matlib
print(np.matlib.identity(5))
```

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

Out:

19-6: numpy.matlib.rand(): this function returns a matrix of the given size filled with random values.

```
import numpy as np
import numpy.matlib
print(np.matlib.rand(3,2))
```

```
[[0.4075875  0.51520385]
 [0.05916832 0.90018686]
 [0.75834896 0.51907469]]
```

Out:

Note that a matrix is always two-dimensional.

```
import numpy as np
import numpy.matlib
print(np.matrix('1,2;3,4'))
```

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

Out:

20- NumPy-Linear Algebra

nupy package contains numpy.linalg module that provides all functionality required for linear algebra.

20-1:numpy.dot(): this function returns the dot product of two arrays.for 1-D arrays, it is the inner product of the vectors.

```
import numpy as np
import numpy.linalg
x=np.array([[1,2,3],[5,8,2]])
y=np.array([[3,6,2],[6,4,1],[5,4,2]])
print(f'After Applying dot() function result is:\n {np.dot(x,y)}')
```

Out:

```
After Applying dot() function result is:
 [[30 26 10]
 [73 70 22]]
```

20-2: numpy.vdot(): this function returns the dot product of the two vectors.

```
import numpy as np
import numpy.linalg
x=np.array([[1,2,3],[5,8,2]])
y=np.array([[3,6,2],[6,4,1]])
print(f'After Applying vdot() function result is:\n {np.vdot(x,y)}')
```

Out:

```
After Applying vdot() function result is:
 85
```

20-3: numpy.inner(): this function returns the inner product of vectors for 1-D arrays.

```
import numpy as np
import numpy.linalg
x=np.array([[1,2,3],[2,4,5]])
y=np.array([[2,8,3],[4,3,7]])
print(f'After Applying inner() function result is:\n {np.inner(x,y)}')
```

Out:

```
After Applying inner() function result is:
 [[27 31]
 [51 55]]
```

20-4: numpy.matmul(): this function returns the matrix product of two arrays.

```
import numpy.matlib
import numpy as np
x=[[1,1],[2,3]]
y=[[3,2],[6,4]]
print(f'x is :\n {x}\n and y is:\n {y}')
print(f'after Applying matmul() function :\n {np.matmul(x,y)}')
```

```
x is :
 [[1, 1], [2, 3]]
 and y is:
 [[3, 2], [6, 4]]
after Applying matmul() function :
 [[ 9  6]
 [24 16]]
```

Out:

20-5: numpy.linalg.det(): This function calculates the determinant of the input matrix.in other words, for a matrix[[a,b],[c,d]], the determinant is computed as ad-bc.

```
import numpy as np
data=[[3,9],[15,8]]
print(f'determinant of data is :\n {np.linalg.det(data)}')
```

```
determinant of data is :
 -111.00000000000007
```

Out:

Numpy Review (Part 5)

Zohreh Karimi

20-6: numpy.linalg.inv(): This function calculate the inverse of a matrix.

```
import numpy as np
data=[[1,5,8],[2,6,3],[6,9,2]]
print(f'origin data is :\n {data}')
print(f'inverse of data is:\n {np.linalg.inv(data)}')
```

```
origin data is :
 [[1, 5, 8], [2, 6, 3], [6, 9, 2]]
inverse of data is:
 [[ 0.16853933 -0.69662921  0.37078652]
 [-0.15730337  0.51685393 -0.14606742]
 [ 0.20224719 -0.23595506  0.04494382]]
```

Out:

21- Numpy.matplotlib

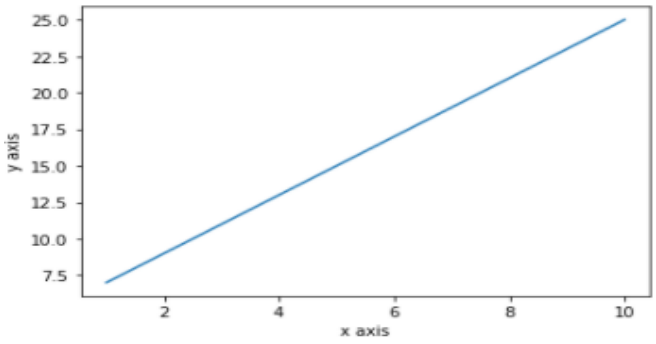
Matplotlib is a plotting library Python. It is used along with NumPy to provide an environment that is an effective open source alternative fot Matlab. It can also be used with graphics toolkits like PyQt and wxPython. The package is imported into the Python script by adding the following statement:

from matplotlib import pyplot as plt

pyplot() is used to plot 2D dta.

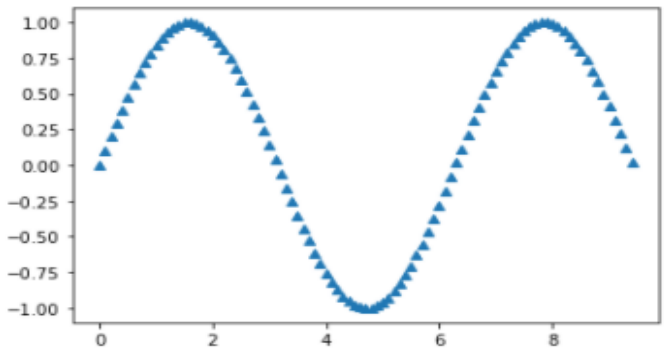
```
import numpy as np
from matplotlib import pyplot as plt
x=np.arange(1,11)
y=2*x+5
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.plot(x,y)
plt.show()
```

Out:




```
import numpy as np
from matplotlib import pyplot as plt
x=np.arange(0,3*np.pi,0.1)
y=np.sin(x)
plt.plot(x,y,'^')
plt.show()
```

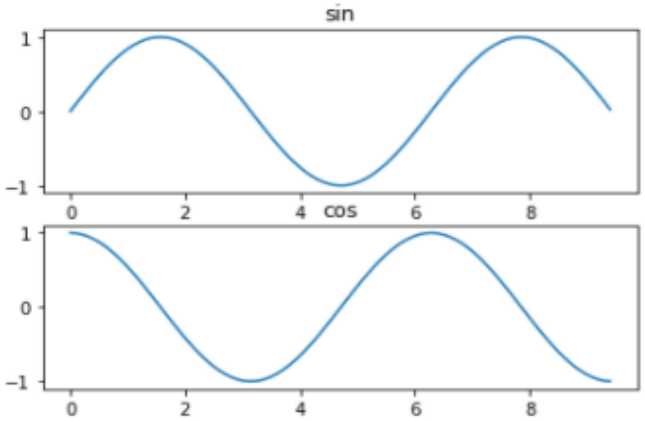
Out:



subplot():this function allows you to plot different things in same figure.

```
import numpy as np
from matplotlib import pyplot as plt
x=np.arange(0,3*np.pi,0.1)
y_sin=np.sin(x)
y_cos=np.cos(x)
plt.subplot(2,1,1)
plt.plot(x,y_sin)
plt.title('sin')
plt.subplot(2,1,2)
plt.plot(x,y_cos)
plt.title('cos')
plt.show()
```

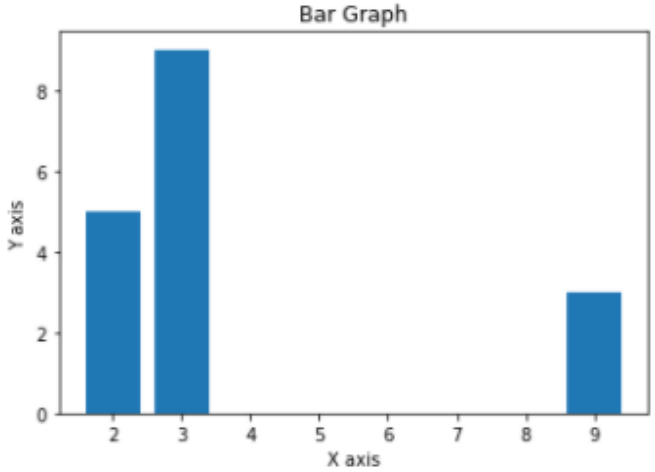
Out:



bar(): the pyplot submodules provides bar() function to generate bar graph.

```
import numpy as np
from matplotlib import pyplot as plt
x=[3,2,9]
y=[9,5,3]
plt.bar(x,y,align='center')
plt.title('Bar Graph')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
```

Out:



Numpy Review (Part 5)

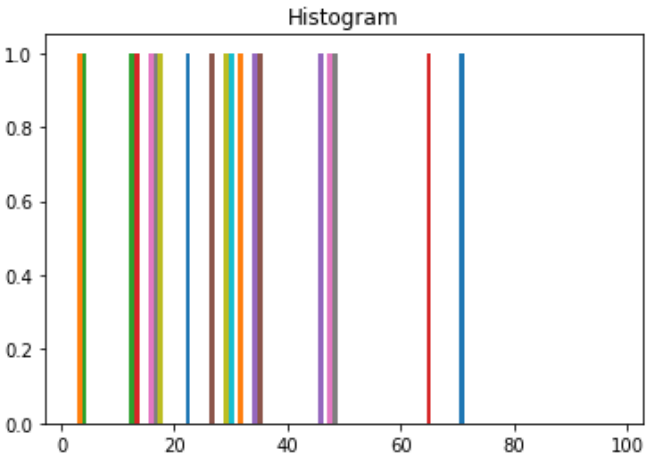
Zohreh Karimi

22- Numpy- Histogram Using Matplotlib

numpy.histogram() function is a graphical representation of the frequency distribution of data. Rectangles of equal horizontal size corresponding to class interval called bin and variable height corresponding to frequency.

```
import numpy as np
from matplotlib import pyplot as plt
x=np.array([[22,8,5,65,45,21,45,47,21,32,74,21,2,12,35,39,12,10,9]])
plt.hist(x,bins=[0,20,40,60,80,100])
plt.title('Histogram')
plt.show()
```

Out:



23- I/O with NumPy

ndarray objects can be saved to and loaded from the disk files. The IO functions available are:

- load() and save() functions handle /numPy binart files.

-loadtxt() and savetxt() functions handle normal text files.

23-1: numpy.save(): this function stores the input array in a disk file with npy extension.

```
import numpy as np
data=np.array([1,2,3,4,5])
np.save('out_data',data)
```

To reconstruct array from outfile.npy, use load() function.

```
import numpy as np
out_data=np.load('out_data.npy')
print(out_data)
```

Out: `[1 2 3 4 5]`

23-2: numpy.savetxt(): this function storage and loadtxt() retrieval of array data in simple text file format is done .

```
import numpy as np
data=np.array([1,2,3,4,5])
np.savetxt('out.txt',data)
output=np.loadtxt('out.txt')
print(output)
```

Out: `[1. 2. 3. 4. 5.]`

Note: The savetxt() and loadtxt() functions accept additional optional parameters such as header, footer and delimiter.

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on array can be performed. This tutorial explains the basics of NumPy such as its architecture and environment. It also discusses the various array functions, types of indexing, etc. All this is explained with the help of examples for better understanding.