Hello and welcome, everyone!

"Real-life use case on NumPy; A tool that Every Data Scientist Needs to Master"

Seyyed Ali Mohammadiyeh (Max Base)
Faculty of Mathematics, University of Kashan
Open-Source Maintainer, GitHub
CTO, Asrez
maxbasecode@gmail.com

WHO I AM

ALI, (Aka Max Base)

- GitHub and Open-Source world **(^_^)**
- Open-Source Maintainer
- Software Engineer, Programming more then 10 years
- Mathematics background
- **@basemax** on GitHub

NumPy

NumPy is a library for the **Python** programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

https://numpy.org/

Introduction to NumPy

NumPy is a python package used for numerical and scientific computing. It provides better runtime and space complexity. It provides a wide variety of array operations. If you wish to perform general-purpose operations, use python lists. But, if you care about performance and space complexity, use Numpy.

Benefits of NumPy

Using NumPy, a developer can perform the following operations:

1. Mathematical and logical operations on arrays;
2. Fourier transforms and routines for shape manipulation;
3. Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

Why NumPy

In Python we have lists that serve the purpose of arrays, but they are slow to process.

1 NumPy aims to provide an array object that is up to 50x faster that traditional Python lists.

2 The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

3 Arrays are very frequently used in data science, where speed and resources are very important.

Using NumPy

If you have Python and PIP already installed on a system, then installation of NumPy is very easy.

Install it using this command:

$ pip install numpy

NumPy Array

As we know that NumPy contains a collection of tools and techniques that can be used to solve on a computer mathematical models of problems in Science and Engineering. One of these tools is a high-performance multidimensional array object that is a powerful data structure for efficient computation of arrays and matrices. To work with these arrays, there's a vast amount of high-level mathematical functions operate on these matrices and arrays.

Advantages of NumPy

Below are the points that explain the advantages of NumPy:

- The core of Numpy is its arrays. One of the main advantages of using Numpy arrays is that they take less memory space and provide better runtime speed when compared with similar data structures in python(lists and tuples).
- Numpy support some specific scientific functions such as linear algebra. They help us in solving linear equations.
- Numpy support vectorized operations, like elementwise addition and multiplication, computing Kronecker product, etc. Python lists fail to support these features.
- It is a very good substitute for MATLAB, OCTAVE, etc as it provides similar functionalities and supports with faster development and less mental overhead(as python is easy to write and comprehend)
- NumPy is very good for data analysis.

Disadvantages of NumPy

Below are the points that explain the disadvantages of NumPy:

- Using "nan" in Numpy: "Nan" stands for "not a number". It was designed to address the problem of missing values. NumPy itself supports "nan" but lack of cross-platform support within Python makes it difficult for the user. That's why we may face problems when comparing values within the Python interpreter.
- Require a contiguous allocation of memory: Insertion and deletion operations become costly as data is stored in contiguous memory locations as shifting it requires shifting.

Numpy Tutorial

```
import numpy as np
```

Arrays in Python

```
my_list = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
print(type(my_list))
```

> list

This is a normal array in Python.

Arrays in NumPy

```
import numpy as np

my_list = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

arr1 = np.array(my_list)
print(arr1)
```

> array([10, 20, 30, 40, 50, 60])

This is a array in NumPy.

data

Memory address of an array object:

print(arr1.data)

> <memory at 0x000001C2B747E348>

type()

Display type of an object:

print(type(arr1))

> numpy.ndarray

dtype

Datatype of array:

print(arr1.dtype)

> dtype('int32')

astype

Convert Integer Array to FLOAT:

print(arr1.astype(float))

> array([10., 20., 30., 40., 50., 60., 70., 80., 90., 100.])

np.arange

Generate evenly spaced numbers (space =1) between 0 to 10:

print(np.arange(0,10))

> array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

np.arange

Generate numbers between 0 to 100 with a space of 10:

print(np.arange(0,100,10))

> array([0, 10, 20, 30, 40, 50, 60, 70, 80, 90])

np.arange

Generate numbers between 10 to 100 with a space of 10 in descending order:

print(np.arange(100, 10, -10))

> array([100, 90, 80, 70, 60, 50, 40, 30, 20])

shape

Shape of Array:

arr = np.arange(0,10)

print(arr.shape)

> (10,)

print(arr)

> array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

size

Size of Array:

print(arr3.size)

> 10

Length of array

```
print(len(arr3))
```

ndim

Dimension of Array:

print(arr.ndim)

> 1

itemsize

Bytes consumed by one element of an array object:

print(arr3.itemsize)

> 4

nbytes

Bytes consumed by an array object:

print(arr3.nbytes)

> 40

np.zeros

Generate an array of zeros:

print(np.zeros(10))

> array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

np.ones

Generate an array of ones with given shape:

print(np.ones(10))

> array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])

np.repeat

Repeat 10 five times in an array:

print(np.repeat(10,5))

> array([10, 10, 10, 10, 10])

np.repeat

Repeat each element in array 'a' thrice:

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

print(np.repeat(a, 3))

> array([10, 10, 10, 20, 20, 20, 30, 30, 30])

np.full

Array of 10's:

print(np.full(5,10))

> array([10, 10, 10, 10, 10])

np.arange

Generate array of Odd numbers:

ar1 = np.arange(1,20)

print(ar1[ar1%2 ==1])

> array([1, 3, 5, 7, 9, 11, 13, 15, 17, 19])

np.arange

Generate array of even numbers:

```
ar1 = np.arange(1,20)
print(ar1[ar1%2 == 0])
```

> array([2, 4, 6, 8, 10, 12, 14, 16, 18])

np.linspace

Generate evenly spaced 4 numbers between 10 to 20:

print(np.linspace(10,20,4))

> array([10. , 13.33333333, 16.66666667, 20.])

np.linspace

Generate evenly spaced 11 numbers between 10 to 20:

print(np.linspace(10,20,11))

> array([10., 11., 12., 13., 14., 15., 16., 17., 18., 19., 20.])

np.random.random

Create an array of random values:

print(np.random.random(4))

> array([0.61387161, 0.7734601 , 0.48868515, 0.05535259])

np.random.randint

Generate an array of Random Integer numbers:

print(np.random.randint(0,500,5))

> array([359, 3, 200, 437, 400])

np.random.randint

Generate an array of Random Integer numbers:

print(np.random.randint(0,500,10))

> array([402, 196, 481, 426, 245, 19, 292, 233, 399, 175])

np.random.seed

Using random.seed we can generate same number of Random numbers:

np.random.seed(123)

print(np.random.randint(0,100,10))

> array([66, 92, 98, 17, 83, 57, 86, 97, 96, 47])

np.random.seed(123)

print(np.random.randint(0,100,10))

> array([66, 92, 98, 17, 83, 57, 86, 97, 96, 47])

np.random.seed

Using random.seed we can generate same number of Random numbers:

np.random.seed(101)

print(np.random.randint(0,100,10))

> array([95, 11, 81, 70, 63, 87, 75, 9, 77, 40])

np.random.seed

Generate array of Random float numbers:

f1 = np.random.uniform(5,10, size=(10))

print(f1)

> array([6.5348311 , 9.4680654 , 8.60771931, 5.94969477, 7.77113796,

6.76065977, 5.90946201, 8.92800881, 9.82741611, 6.16176831])

np.floor

Extract Integer part:

print(np.floor(f1))

> array([6., 9., 8., 5., 7., 6., 5., 8., 9., 6.])

np.ceil

Extract Integer part:

```
a = np.array([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0])
print(np.ceil(a))
```

> array([-1., -1., -0., 1., 2., 2., 2.])

astype

Convert Float Array to Integer array:

print(f1.astype(int))

> array([6, 9, 8, 5, 7, 6, 5, 8, 9, 6])

np.trunc

Truncate decimal part:

print(np.trunc(f1))

>array([6., 9., 8., 5., 7., 6., 5., 8., 9., 6.])

Convert Float Array to Integer array:

print(f1.astype(int))

> array([6, 9, 8, 5, 7, 6, 5, 8, 9, 6])

np.random.randn

Normal distribution (mean=0 and variance=1):

b2 = np.random.randn(10)

print(b2)

> array([0.18869531, -0.75887206, -0.93323722, 0.95505651, 0.19079432,

1.97875732, 2.60596728, 0.68350889, 0.30266545, 1.69372293])

np.ndenumerate

print(arr1)
> array([10, 20, 30, 40, 50, 60])

Enumerate for Numpy Arrays:

```
for index, value in np.ndenumerate(arr1):
    print(index, value)
```
> (0,) 10
> (1,) 20
> (2,) 30
> (3,) 40
> (4,) 50
> (5,) 60

sum

Sum of all elements in an array:

print(arr2.sum())

> 190

cumsum

Cumulative Sum:

print(np.cumsum(arr2))

> array([1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91,
105, 120, 136, 153, 171, 190], dtype=int32)

min

Find Minimum number in an array:

print(arr2.min())

> 1

max

Find MAX number in an array:

print(arr2.max())

> 19

argmin

Find INDEX of Minimum number in an array:

print(arr2.argmin())

> 0

argmax

Find INDEX of MAX number in an array:

print(arr2.argmax())

> 18

mean

Find mean of all numbers in an array:

print(arr2.mean())

> 10.0

median

Find median of all numbers present in arr2:

print(np.median(arr2))

> 10.0

var

Variance:

print(np.var(arr2))

> 30.0

np.std

Standard deviation:

print(np.std(arr2))

> 5.477225575051661

np.percentile

Calculating percentiles:

print(np.percentile(arr2,70))

> 13.6

np.percentile

10th & 70th percentile:

print(np.percentile(arr2, [10, 70]))

> array([2.8, 13.6])

Two-dimensional array

```
A = np.array([[1,2,3,0] , [5,6,7,22] , [10 , 11 , 1 ,13] , [14,15,16,3]])
print(A)
> array([[ 1, 2, 3, 0],
 [ 5, 6, 7, 22],
 [10, 11, 1, 13],
 [14, 15, 16, 3]])
```

sum

SUM of all numbers in a 2D array:

print(A.sum())

> 129

max

MAX number in a 2D array:

print(A.max())

> 22

min

Minimum number in a 2D array:

print(A.min())

> 0

amin

Column wise mimimum value:

print(np.amin(A, axis=0))

> array([1, 2, 1, 0])

amin

Row wise mimimum value:

print(np.amin(A, axis=1))

> array([0, 5, 1, 3])

mean

Mean of all numbers in a 2D array:

print(A.mean())

> 8.0625

median

Median:

print(np.median(A))

> 6.5

[index]

Access first element of the array:

```
a = np.array([7,5,3,9,0,2])
print(a[0])
> 7
```

[index:]

Access all elements of Array except first one:

a = np.array([7,5,3,9,0,2])

print(a[1:])

> array([5, 3, 9, 0, 2])

[index1:index2]

Fetch 2nd , 3rd & 4th value from the Array:

a = np.array([7,5,3,9,0,2])

print(a[1:4])

> array([5, 3, 9])

[- index1]

Get last element of the array:

```
a = np.array([7,5,3,9,0,2])
print(a[-1])
> 2
print(a[-3])
> 9
print(a[-6])
> 7
```

[- index1 : - index2]

```
a = np.array([7,5,3,9,0,2])
print(a[-3:-1])
```

> array([9, 0])

np.where

ar = np.arange(1,20)

print(ar)

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

rep1 = np.where(ar % 2 == 0, 0 , ar)

print(rep1)

> [1 0 3 0 5 0 7 0 9 0 11 0 13 0 15 0 17 0 19]

np.where

ar2 = np.array([10, 20 , 30 , 10 ,10 ,20, 20])

print(ar2)

> array([10, 20, 30, 10, 10, 20, 20])

Replace 10 with value 99:

rep2 = np.where(ar2 == 10, 99 , ar2)

print(rep2)

> [99 20 30 99 99 20 20]

np.put

p2 = np.arange(0,100,10)

print(p2)

> array([0, 10, 20, 30, 40, 50, 60, 70, 80, 90])

Replace values at INDEX loc 0,3,5 with 33,55,99:

np.put(p2, [0, 3 , 5], [33, 55, 99])

print(p2)

> array([33, 10, 20, 55, 40, 99, 60, 70, 80, 90])

np.nan

Missing Values in an array:

```
a = np.array([10 ,np.nan,20,30,60,np.nan,90,np.inf])
print(a)
```

> array([10., nan, 20., 30., 60., nan, 90., inf])

np.isnan

Search for missing values and return as a boolean array:

print(np.isnan(a))

> array([False, True, False, False, False, True, False, False])

np.isnan

Index of missing values in an array:

```
print(
    np.where(np.isnan(a))
)
```

(array([1, 5], dtype=int64),)

np.isnan

Replace all missing values with 99:

a[np.isnan(a)] = 99

print(a)

> array([10., 99., 20., 30., 60., 99., 90., inf])

np.isnan, any

Check if array has any NULL value:

print(np.isnan(a).any())

> False

np.nan

A = np.array([[1,2,np.nan,4] , [np.nan,6,7,8] , [10 , np.nan , 12 ,13] , [14,15,……)

print(A)

> array([[1., 2., nan, 4.],

[nan, 6., 7., 8.],

[10., nan, 12., 13.],

[14., 15., 16., 17.]])

np.isnan

Search for missing values and return as a boolean array:

np.isnan(A)

```
> array([[False, False, True, False],
 [ True, False, False, False],
 [False, True, False, False],
 [False, False, False, False]])
```

np.isnan

Index of missing values in an array:

np.where(np.isnan(A))

(array([0, 1, 2], dtype=int64), array([2, 0, 1], dtype=int64))

reshape

a = np.arange(6).reshape((3, 2))

print(a)

> array([[0, 1], [2, 3], [4, 5]])

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

> array([[0, 1, 2], [3, 4, 5]])

print(np.reshape(np.ravel(a), (2, 3)))

> array([[0, 1, 2], [3, 4, 5]])

reshape

Stack Arrays Vertically

The unspecified value is inferred to be 2:

```
a = np.zeros(20).reshape(2,-1)
b = np.repeat(1, 20).reshape(2,-1)
print(a)
> array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
print(b)
> array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

np.vstack

np.vstack([a,b])

```
> array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
 [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
 [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])
```

a1 = np.array([[1], [2], [3]])
b1 = np.array([[4], [5], [6]])
print(a1)

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

print(b1)

```
> array([[4],
 [5],
 [6]])
```

np.hstack

Stack Arrays Horizontally:

print(np.hstack([a,b]))

```
> array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1.,
 1., 1., 1., 1.],
 [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1.,
 1., 1., 1., 1.]])
```

print(np.hstack([a1,b1]))

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

np.intersect1d

Common items between two Arrays:

```
c1 = np.array([10,20,30,40,50,60])
c2 = np.array([12,20,33,40,55,60])
print( np.intersect1d(c1,c2) )
```

> array([20, 40, 60])

np.setdiff1d

Remove Common Elements

E.g: Remove common elements of C1 & C2 array from C1

print(np.setdiff1d(c1,c2))

> array([10, 30, 50])

np. where

Process Elements on Conditions:

a = np.array([1,2,3,6,8])

b = np.array([10,2,30,60,8])

Returns the indices of elements in an input array where a is equal to b:

print(np.where(a == b))

> (array([1, 4], dtype=int64),)

np.where

Return an array where condition is satisfied:

print(a[np.where(a == b)])

> array([2, 8])

np.where

Return all numbers betweeen 20 & 35:

a1 = np.arange(0,60)

print(a1[np.where ((a1>20) & (a1<35))])

> array([21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34])

np.where

Return all numbers betweeen 20 & 35 OR numbers divisible by 10:

```
a1 = np.arange(0,60)
print(
    a1[np.where (((a1>20) & (a1<35)) | (a1 % 10 ==0)) ]
)
```

> array([0, 10, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
 40, 50])

np.logical_and

Return all numbers betweeen 20 & 35 using np.logical_and:

```
print(
   a1[np.where(np.logical_and(a1>20, a1<35))]
)
```

> array([21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34])

np.isin

a = np.array([10,20,30,40,50,60,70])
print(a)
> array([10, 20, 30, 40, 50, 60, 70])

Check whether number 11 & 20 are present in an array:

print(np.isin(a, [11,20]))
> array([False, True, False, False, False, False, False])

Display the matching numbers:
print(a[np.isin(a,20)])
> array([20])

np.isin

a = np.array([10,20,30,40,50,60,70])
print(a)
> array([10, 20, 30, 40, 50, 60, 70])

Display the matching numbers:
print(a[np.isin(a,20)])
> array([20])

Check whether number 33 is present in an array:
np.isin(a, 33)
> array([False, False, False, False, False, False, False])
print(a[np.isin(a, 33)])
> array([], dtype=int32)

np.isin

b = np.array([10,20,30,40,10,10,70,80,70,90])
print(b)
> array([10, 20, 30, 40, 10, 10, 70, 80, 70, 90])

Check whether number 10 & 70 are present in an array:
np.isin(b, [10,70])
> array([True, False, False, False, True, True, True, False, True, False])

Display the indices where match occurred:
np.where(np.isin(b, [10,70]))
> (array([0, 4, 5, 6, 8], dtype=int64),)

np.isin

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

print(b)

> array([10, 20, 30, 40, 10, 10, 70, 80, 70, 90])

Display the matching values:

print(b[np.where(np.isin(b, [10,70]))])

> array([10, 10, 10, 70, 70])

Display the matching values:

print(b[np.isin(b, [10,70])])

> array([10, 10, 10, 70, 70])

Reverse the list

a4 = np.arange(10,30)

print(a4)

> array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])

Reverse the array:

print(a4[::-1])

> array([29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10])

np.flip

Reverse the array:

print(np.flip(a4))

> array([29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10])

Reverse row-based or column-based

a3 = np.array([[3,2,8,1] , [70,50,10,67] , [45,25,75,15] , [12,9,77,4]])
print(a3)
> array([[3, 2, 8, 1],
[70, 50, 10, 67],
[45, 25, 75, 15],
[12, 9, 77, 4]])

Reverse ROW positions:
print(a3[::-1,])
> array([[12, 9, 77, 4],
[45, 25, 75, 15],
[70, 50, 10, 67],
[3, 2, 8, 1]])

Reverse COLUMN positions:
print(a3[:,::-1])
> array([[1, 8, 2, 3],
[67, 10, 50, 70],
[15, 75, 25, 45],
[4, 77, 9, 12]])

Reverse both row-based and column-based

a3 = np.array([[3,2,8,1] , [70,50,10,67] , [45,25,75,15] , [12,9,77,4]])
print(a3)
> array([[3, 2, 8, 1],
[70, 50, 10, 67],
[45, 25, 75, 15],
[12, 9, 77, 4]])

Reverse both ROW & COLUMN positions:
print(a3[::-1,::-1])
> array([[4, 77, 9, 12],
[15, 75, 25, 45],
[67, 10, 50, 70],
[1, 8, 2, 3]])

np.sort

Sort array in ascending order:

a = np.array([10,5,2,22,12,92,17,33])

print(np.sort(a))

> array([2, 5, 10, 12, 17, 22, 33, 92])

np.sort

a3 = np.array([[3,2,8,1] , [70,50,10,67] , [45,25,75,15]])

print(a3)

> array([[3, 2, 8, 1],
[70, 50, 10, 67],
[45, 25, 75, 15]])

Sort along rows:

print(np.sort(a3))

> array([[1, 2, 3, 8],
[10, 50, 67, 70],
[15, 25, 45, 75]])

np.sort, row axis

a3 = np.array([[3,2,8,1] , [70,50,10,67] , [45,25,75,15]])

print(a3)

> array([[3, 2, 8, 1],
[70, 50, 10, 67],
[45, 25, 75, 15]])

Sort along rows:

print(np.sort(a3,axis =1))

> array([[1, 2, 3, 8],
[10, 50, 67, 70],
[15, 25, 45, 75]])

np.sort, column axis

a3 = np.array([[3,2,8,1] , [70,50,10,67] , [45,25,75,15]])

print(a3)

> array([[3, 2, 8, 1],
[70, 50, 10, 67],
[45, 25, 75, 15]])

Sort along columns:

print(np.sort(a3,axis =0))

> array([[3, 2, 8, 1],
[45, 25, 10, 15],
[70, 50, 75, 67]])

np.sort, DESC

Sort in descending order:

```
b = np.sort(a)
b = b[::-1]
print(b)
```

> array([92, 33, 22, 17, 12, 10, 5, 2])

np.sort, DESC

Sort in descending order:

c = np.sort(a)

print(np.flip(c))

> array([92, 33, 22, 17, 12, 10, 5, 2])

sort

Sort in descending order:

a[::-1].sort()

print(a)

> array([92, 33, 22, 17, 12, 10, 5, 2])

np.random.shuffle

p = np.arange(0,50)

print(p)

> array([0, 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, 28, 29, 30, 31, 32, 33,
34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])

np.random.shuffle(p)

print(p)

> array([33, 48, 14, 20, 44, 29, 4, 46, 18, 45, 21, 2, 7, 30, 17, 40,
37, 42, 34, 25, 35, 38, 43, 8, 24, 32, 10, 36, 0, 26, 12, 9, 3, 39,
6, 49, 23, 13, 1, 5, 19, 27, 47, 15, 22, 11, 41, 31, 16, 28])

np.argsort

Return "n" largest numbers in an Array:

n = 4

print(p[np.argsort(p)[-nth:]])

> array([46, 47, 48, 49])

np.argpartition

Return "n" largest numbers in an Array:

print(p[np.argpartition(-p,n)[:n]])

> array([48, 47, 49, 46])

np.argsort

Return "n" smallest numbers in an Array:

print(p[np.argsort(-p)[-n:]])

> array([3, 2, 1, 0])

=

np.argpartition

Return "n" smallest numbers in an Array:

print(p[np.argpartition(p,n)[:n]])

> array([1, 0, 2, 3])

np.tile

a5 = [10,20,30]

print(a5)

> [10, 20, 30]

Repeat whole array twice:

print(np.tile(a5, 2))

> array([10, 20, 30, 10, 20, 30])

np.repeat

a5 = [10,20,30]

print(a5)

> [10, 20, 30]

Repeat each element in an array thrice:

print(np.repeat(a5, 3))

> array([10, 10, 10, 20, 20, 20, 30, 30, 30])

np.allclose

Compare arrays using "allclose" function:

d1 = np.arange(0,10)
print(d1)
> array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
d2 = np.arange(0,10)
print(d2)
> array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
d3 = np.arange(10,20)
print(d3)
> array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
d4 = d1[::-1]
print(d4)
> array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])

res1 = np.allclose(d1,d2)
print(res1)
> true

res2 = np.allclose(d1,d3)
print(res2)
> false

res3 = np.allclose(d1,d4)
print(res3)
> false

np.unique

Unique numbers in an array:

```
b = np.array([10,10,10,20,30,20,30,30,20,10,10,30,10])
print( np.unique(b) )
```

> array([10, 20, 30])

Unique numbers in an array along with the count E.g value 10 occurred maximum:

```
val , count = np.unique(b,return_counts=True)
print(val,count)
```

> (array([10, 20, 30]), array([6, 3, 4], dtype=int64))

np.bincount

Unique numbers in an array:

b = np.array([10,10,10,20,30,20,30,30,20,10,10,30,10])

print(np.bincount(b))

> array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0,
0, 0, 0, 0, 0, 0, 0, 4], dtype=int64)

argmax()

Unique numbers in an array:

b = np.array([10,10,10,20,30,20,30,30,20,10,10,30,10])

print(np.bincount(b))

> array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0,
0, 0, 0, 0, 0, 0, 0, 4], dtype=int64)

10 is the most frequent value:

print(np.bincount(b).argmax())

> 10

flags.writeable

d5 = np.arange(10,100,10)

print(d5)

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

Make arrays immutable:

d5.flags.writeable = False

d5[0] = 99

> ValueError: assignment destination is read-only

d5[2] = 11

> ValueError: assignment destination is read-only

np.loadtxt

Load data from a text file using loadtext:

```
p4 = np.loadtxt('sample.txt',
    dtype = np.integer # Decides the datatype of resulting array
)
print(p4)
> array([[24, 29, 88],
 [ 1, 0, 8],
 [33, 7, 99],
 [39, 11, 98],
 [22, 76, 87]])
```

np.genfromtxt

Load data from a text file using genfromtxt:

p5 = np.genfromtxt('sample0.txt',dtype='str')
print(p5)
> array([['Asif', 'India', 'Cricket'],
['John', 'USA', 'Hockey'],
['Ramiro', 'Canada', 'Football']], dtype='<U8')

Accessing specific rows:
print(p5[0])
> array(['Asif', 'India', 'Cricket'], dtype='<U8')

Accessing specific columns:
print(p5[:,0])
> array(['Asif', 'John', 'Ramiro'], dtype='<U8')

np.genfromtxt

```
p6 = np.genfromtxt('sample2.txt',
  delimiter=' ',
  dtype=None,
  names=('Name', 'ID', 'Age')
)
print(p6)
```

```
> array([(b'Name', b'ID', b'Age'), (b'Asif', b'22', b'29'),
 (b'John', b'45', b'33'), (b'Ramiro', b'55', b'67'),
 (b'Michael', b'67', b'55'), (b'Klaus', b'44', b'32'),
 (b'Sajad', b'23', b'53')],
 dtype=[('Name', 'S7'), ('ID', 'S2'), ('Age', 'S3')])
```

References

- https://www.educba.com/introduction-to-numpy/
- https://medium.com/analytics-vidhya/introduction-to-numpy-279bbc88c615
- https://github.com/basemax/DataScienceGlobalSummit-NumPy

Thank you for your attention.

Tabatabaei Historical House,
Kashan, I.R.

For any questions in the future:
maxbasecode@gmail.com