Numpy Practice Tutorial

Introduction to NumPy

NumPy stands for Numerical Python.

It provides lot of functions to work in a domain of Linear Algebra,Fourier Transform and Matrices.

It provudes functions related to Arrays.

It creates an array called ndarray by using array().

Working of ndarray is faster than Lists. In array function we can pass list,tuple,or any sequencial datatype.

To get the version of NumPy:

```
print(numpy*__version__)

Most of the part of numpy library is designed by C & C++.
```

** Python provides set of modules.

** Python is a general purpose programming language.

** Due to NumPy, Scipy, Pandas,Python becomes more required programming language for data analysis.

** We are processing the data by storing in different formats.i.e., one dimensional, 2-dimensional, multidimensional.

** We can store the data in one dimensional, 2-dimensional, multidimensional format using arrays concepts. But, arrays are static i.e., the size of the array is fixed. and Python is a dynamic programming language.

** Python supports only dynamic collection data types.like lists, tuple

** Because of NumPy module only. We can achieve array concept in Python.

```
 which means we can process and store the data of 1-D, 2-D, 3-D and multidimensional data effectively.
```

** NumPy is a low level library written in C (and FORTRAN) for high level mathematical functions.

** NumPy cleverly overcomes the problem of running slower algorithms on python by using multidimensional arrays and functions that operate on arrays.

** Any algorithm can be expressed as a function of arrays, allowing the algorithm to be run quickly.

What is NumPy?

** NumPy is the fundamental package for scientific computing in Python.

** NumPy is a python library that provides a multidimensional array object, various derived objects.

What is NumPy Array?

** An Array is a grid of values and it contains information about the raw data how to locate an element, & how to interprete an element.

** Wide variety of mathematical operations on arrays.

** It supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

** Mathematical, LOgical, Shape manipulation, sorting, selecting, I/O, discrete fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

Advantages of using NumPy Array over Python Lists:

1. consumes less memory
2. fast as compared to the python list
3. convenient to use

Difference between NumPy Array and List in Python:

1.Data types storage

2.Importing Module

3.Numerical Operation

4.Modification Capabilities

5.Consumes less memory

6.Fast as compared tothe Python list

7.Convenient to use

```
List = [1,2,3,4]

Array = [1 2 3 4]
```

```
%timeit [j**4 for j in range(1,9)]

2.55 s+ 61.1 ns per loop(mean+std.dev. of 7 runs, 100000 loops each)

import numpy as np

%timeit np.arange(1,9)**4

2.02 s+ 142 ns per loop(mean+std.dev. of 7 runs,100000 loops each)
```

Applications of NumPy:

1. A powerful N-dimensional array object.
2. Sophisticated(broadcasting)functions.
3. Tools for integrating C/C++ Fortran code
4. Useful linear algebra, Fourier transform, and random number capabilities.

Properties of NumPy:

1. Homogenous
2. Only be numbers(Number, Float, Complex)
3. Fixed item size
4. Actually, It can hold strings but we never use numpy that way.

Installation Of NumPy

```
pip install numpy
```

Import NumPy Library

import numpy as np

Syntax for Creating a NumPy Array by using array():

In [1]:

```
import numpy
a= numpy.array((1,2,3,4,5,6))
print(a)
print(type(a))
print(a.itemsize)
```

```
[1 2 3 4 5 6]
<class 'numpy.ndarray'>
4
```

or

In [2]:

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

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

or

In [6]:

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

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

or

In [4]:

```
import numpy as np
l = []
for i in range(1,5):
    a = input("enter:")
    l.append(a)
print(np.array(l))
```

```
enter:1
enter:2
enter:3
enter:4
['1' '2' '3' '4']
```

In [5]:

```
import numpy as np
l1 = []
for i in range(1,5):
    b = int(input("enter:"))
    l1.append(b)
print(np.array(l1))
```

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

Types of Arrays:

** 0-D Array => np.array(90)

** 1-D Array => np.array((3,4,5)) => Array of 0-D Array elements.

** 2-D Array => np.array([[1,3,6],[4,8,9]]) => Array of 1-D Array elements.

** 3-D Array => np.array([[[1,2,3],[4,5,6]],[[7,8,9],[3,4,5]]]) => Array of 2-D Array elements.

** Higher Dimensional Arrays

How to know the dimention of Arrays?

Dimention can be calculated by using ndim attribute. Dimention means Level of depth of an array.

type() describes the type of object assigned to a variable.It is used to tell which type of data the variable consists of.

There are some Attributes: ndim shape

Code:

0-D array:

In [5]:

```python
import numpy as np
a = np.array(90)
print(a)
```

```
90
```

1-D array:

In [7]:

```python
import numpy as np
b = np.array([3,4,5])
print(b)
```

```
[3 4 5]
```

2-D array:

In [3]:

```python
import numpy as np
c= np.array([[4,6,8],[5,9,2]])
print(c)
print(c.itemsize)
```

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

3-D array:

In [6]:

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

```
[[[3 5 7]
  [8 9 2]]

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

In [12]:

```
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```

In [13]:

```
print(type(a))
print(type(b))
print(type(c))
print(type(d))
```

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

Higher Dimensional Array:

In [7]:

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

```
[[[[[[[[[[1 2 3 4]]]]]]]]]]
10
4
```

In [ ]:

```
```

Properties or attributes of NumPy arrays:

1. shape
2. ndim
3. size
4. Itemsize
5. dtype

1-D Array With Properties or attributes:

In [2]:

```
import numpy as np
l = [20,25,30,35,40,45,50,55]
arr = np.array(l)
print("Array:", arr)
print("Size:", arr.size)
print("Datatype:", arr.dtype)
print("Dimension:", arr.ndim)
print("Shape:", arr.shape)
```

```
Array: [20 25 30 35 40 45 50 55]
Size: 8
Datatype: int32
Dimension: 1
Shape: (8,)
```

2-D Array With Properties or attributes:

In [4]:

```
import numpy as np
l1 = [[20,25,30],[35,40,45],[50,55,60]]
arr_1 = np.array(l1)
print("Array:", arr_1)
print("Size:", arr_1.size)
print("Datatype:", arr_1.dtype)
print("Dimension:", arr_1.ndim)
print("Shape:", arr_1.shape)
```

```
Array: [[20 25 30]
 [35 40 45]
 [50 55 60]]
Size: 9
Datatype: int32
Dimension: 2
Shape: (3, 3)
```

3-D Array With Properties or attributes:

In [5]:

```
import numpy as np
l2 = [[[20,25,30],[35,40,45]],[[50,55,60],[65,70,75]]]
arr_2 = np.array(l2)
print("Array:", arr_2)
print("Size:", arr_2.size)
print("Datatype:", arr_2.dtype)
print("Dimension:", arr_2.ndim)
print("Shape:", arr_2.shape)
```

```
Array: [[[20 25 30]
  [35 40 45]]

 [[50 55 60]
  [65 70 75]]]
Size: 12
Datatype: int32
Dimension: 3
Shape: (2, 2, 3)
```

Creating 1-D Array in NumPy

In [9]:

```
import numpy as np
n = int(input("Enter Size:"))
arr_3 = np.ndarray(shape=(n),dtype=int)
print("Enter %d elements:" %n)
for i in range(n):
    arr_3[i] = int(input())
print("Array Elements:", arr_3)
```

```
Enter Size:4
Enter 4 elements:
4678
679
543
89
Array Elements: [4678  679  543   89]
```

In [10]:

```
import numpy as np
n = int(input("Enter Size:"))
arr_3 = np.ndarray(shape=(n),dtype=int)
print("Enter %d elements:" %n)
for i in range(n):
    arr_3[i] = input()
print("Array Elements:", arr_3)
```

```
Enter Size:6
Enter 6 elements:
5
6
8
9
2
1
Array Elements: [5 6 8 9 2 1]
```

In [19]:

```
import numpy as np
matrix = np.ndarray(shape=(4,),dtype=int)
print(matrix)
print("Size:", matrix.size)
print("Shape:", matrix.shape)
print("Dimensions:", matrix.ndim)
print("Datatype:", matrix.dtype)
```

```
[4678  679  543   89]
Size: 4
Shape: (4,)
Dimensions: 1
Datatype: int32
```

In [3]:

```
import numpy as np
arr_4 =np.ndarray(shape=(5),dtype=int)
n = arr_4.size
print("Enter %d elements:" %n)
for i in range(n):
    arr_4[i] = int(input())
print("Elements:", arr_4)
print(arr_4)
print("Size:", arr_4.size)
print("Shape:", arr_4.shape)
print("Dimensions:", arr_4.ndim)
print("Datatype:", arr_4.dtype)
```

```
Enter 5 elements:
18
79
54
93
20
Elements: [18 79 54 93 20]
[18 79 54 93 20]
Size: 5
Shape: (5,)
Dimensions: 1
Datatype: int32
```

How to construct two dimensional array and how to check the properties?

In [15]:

```
import numpy as np
x = int(input("Enter row size:"))
y = int(input("Enter column size:"))
matrix = np.ndarray(shape=(x,y),dtype=int)
print(matrix)
print("Size:", matrix.size)
print("Shape:", matrix.shape)
print("Dimensions:", matrix.ndim)
print("Datatype:", matrix.dtype)
```

```
Enter row size:4
Enter column size:3
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
Size: 12
Shape: (4, 3)
Dimensions: 2
Datatype: int32
```

In [16]:

```
import numpy as np
matrix = np.ndarray(shape=(4,3),dtype=int)
print(matrix)
print("Size:", matrix.size)
print("Shape:", matrix.shape)
print("Dimensions:", matrix.ndim)
print("Datatype:", matrix.dtype)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
Size: 12
Shape: (4, 3)
Dimensions: 2
Datatype: int32
```

Numpy Reading Elements into Matrix

In [3]:

```
import numpy as np
a = int(input("Enter row size:"))
b = int(input("Enter column size:"))
matrix = np.ndarray(shape=(a,b),dtype=int)
print("Enter %d elements of %dx%d matrix:"%(a*b,a,b))
for i in range(a):
    for j in range(b):
        matrix[i][j]=int(input())
print("%dx%d matrix is:"%(a,b))
print(matrix)
```

```
Enter row size:4
Enter column size:3
Enter 12 elements of 4x3 matrix:
1
3
6
4
7
9
1
8
45
78
93
45
4x3 matrix is:
[[ 1  3  6]
 [ 4  7  9]
 [ 1  8 45]
 [78 93 45]]
```

Multi Dimentional Arrays

In [8]:

```
import numpy as np
arr = np.ndarray(shape=(3,3,3),dtype=int)
print("size:",arr.size)
print("Datatype:",arr.dtype)
print("Shape:",arr.shape)
print("Dimentions:",arr.ndim)
print(arr.itemsize)
```

```
size: 27
Datatype: int32
Shape: (3, 3, 3)
Dimentions: 3
4
```

In [8]:

```
import numpy
arr = numpy.ndarray(shape=(3,3,2),dtype=int)
print("Enter %d elements:" %arr.size)
```

```
Enter 18 elements:
```

In [9]:

```
import numpy as np
arr = np.ndarray(shape=(3,3,3),dtype=int)
print("Enter %d element:" %arr.size)
```

```
Enter 27 element:
```

In [11]:

```
import numpy as np
arr = np.ndarray(shape=(3,3,3),dtype=int)
val=1
a = arr.shape[0]
b = arr.shape[1]
c = arr.shape[2]
for i in range(a):
    for j in range(b):
        for k in range(c):
            arr[i][j][k]=val
            val = val+1
print("Array elements:")
print(arr)
```

```
Array elements:
[[[ 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]]]
```

Arrays with Existing Data

1. asarray()
2. frombuffer()
3. fromiter()

Syntax for asarray():

syntax: np.asarray(input, dtype, order)

It has Three attributes or parameters.

input could be a List, Tuple, any combination.

dtype could be an int,float,s1.

order could be a Row Major("C") or Column Major("F")

By Default: dtype = float, order = "C" except 1-d array

Syntax for frombuffer():

syntax: np.frombuffer(buffer, dtype, count, offset)

It has Four parameters.

By Default: count=-1 offset=0

input could be a s.

dtype could be a s1.

count means the length of the resultant array.

offset means from which position or index the data should be printed.

Syntax for fromiter:

syntax: np.fromiter(iterable, dtype, count)

It has Three parameters.

iterable could a List, Tuple.

dtype could be an int, float, s1.

count could be a lenght of returned array or resultant array.

By Default: count=-1

Code: Arrays with Existing data

In [15]:

```
import numpy as np
a = np.array([20,40,60,80,100])
print(a)
```

```
[ 20  40  60  80 100]
```

1. asarray()

In [17]:

```
import numpy as np
b = np.asarray(a, dtype="float", order = "C")
print(b)
```

```
[ 20.  40.  60.  80. 100.]
```

In [20]:

```
b = np.asarray(a, dtype="int", order = "C")
print(b)
```

```
[ 20  40  60  80 100]
```

In [22]:

```
b = np.asarray(a,dtype = "int")
print(b)
```

```
[ 20  40  60  80 100]
```

In [23]:

```
b = np.asarray(a)
print(b)
```

```
[ 20  40  60  80 100]
```

In [24]:

```
import numpy as np
b = np.asarray(a, dtype="float", order = "F")
print(b)
```

```
[ 20.  40.  60.  80. 100.]
```

In [25]:

```
b = np.asarray(a, dtype="int", order = "F")
print(b)
```

```
[ 20  40  60  80 100]
```

In [26]:

```
import numpy as np
c = np.array([[20, 40, 60],[10,30,90]])
print(c)
```

```
[[20 40 60]
 [10 30 90]]
```

In [27]:

```
d = np.asarray([[20, 40, 60],[10,30,90]])
print(d)
```

```
[[20 40 60]
 [10 30 90]]
```

In [34]:

```
e = np.asarray([[20, 40, 60],[10,30,90]], dtype = "int",order = "C")
print(e)
```

```
[[20 40 60]
 [10 30 90]]
```

In [29]:

```
f = np.asarray([[20, 40, 60],[10,30,90]], dtype = "int",order="F")
print(f)
```

```
[[20 40 60]
 [10 30 90]]
```

In [30]:

```
g = np.asarray([[20, 40, 60],[10,30,90]], dtype = "float",order="F")
print(g)
```

```
[[20. 40. 60.]
 [10. 30. 90.]]
```

In [31]:

```
h = np.asarray([[20, 40, 60],[10,30,90]], dtype = "float",order="C")
print(h)
```

```
[[20. 40. 60.]
 [10. 30. 90.]]
```

In [39]:

```
for i in np.nditer(d):
    print(i)
```

```
20
40
60
10
30
90
```

In [40]:

```
for i in np.nditer(e):
    print(i)
```

```
20
40
60
10
30
90
```

In [41]:

```
for i in np.nditer(f):
    print(i)
```

```
20
10
40
30
60
90
```

In [42]:

```
for i in np.nditer(g):
    print(i)
```

```
20.0
10.0
40.0
30.0
60.0
90.0
```

In [43]:

```
for i in np.nditer(h):
    print(i)
```

```
20.0
40.0
60.0
10.0
30.0
90.0
```

2.frombuffer:

In [66]:

```
import numpy as np
s = b"hellow welcome to the world"
j = np.frombuffer(s, dtype = "S1", count = -1, offset = 0)
print(j)
```

```
[b'h' b'e' b'l' b'l' b'o' b'w' b' ' b'w' b'e' b'l' b'c' b'o' b'm' b'e'
 b' ' b't' b'o' b' ' b't' b'h' b'e' b' ' b'w' b'o' b'r' b'l' b'd']
```

In [50]:

```
s = b"hellow welcome to the world"
k = np.frombuffer(s, dtype = "S1", count = 10, offset = 7)
print(k)
```

```
[b'w' b'e' b'l' b'c' b'o' b'm' b'e' b' ' b't' b'o']
```

In [52]:

```
s = b"hellow welcome to the world"
l = np.frombuffer(s, dtype = "S1", count = 15, offset = 10)
print(l)
```

```
[b'c' b'o' b'm' b'e' b' ' b't' b'o' b' ' b't' b'h' b'e' b' ' b'w' b'o'
 b'r']
```

3.fromiter():

In [65]:

```
import numpy as np
list = [20,30,60,80,90]
m = np.fromiter(list, dtype = "float", count = -1)
print(m)
```

```
[20. 30. 60. 80. 90.]
```

In [55]:

```
list = [20,30,60,80,90]
n = np.fromiter(list, dtype = "float", count = 3)
print(n)
```

```
[20. 30. 60.]
```

In [56]:

```
list = [20,30,60,80,90]
o = np.fromiter(list, dtype = "int", count = 3)
print(o)
```

```
[20 30 60]
```

In [57]:

```
list = [20,30,60,80,90]
p = np.fromiter(list, dtype = "int", count = -1)
print(p)
```

```
[20 30 60 80 90]
```

Arrays with Numerical Ranges

1. arange()
2. linspace()

3. logspace()

start, stop, dtype parameters are the default common parameters to pass into the function.

Syantax for arange():

syntax: np.arange(start index, stop index, step size, dtype)

It has Four parametes:

dtype could be a int, float.

Syntax for linspace:

syntax: np.linspace(start index, stop index, num, endpoint, retstep, dtype)

It has Six parameters.

i. start index is a lower boundary.

ii. stop index is a upper boundary.

iii. num = number of required values in array.

num is an optional.

num By Default: 50

iv. end point: True - stop index will be included. False - stop index will be not included.

end point is optional.

end point By Default: True

v. retstep: True - visible

```
            False - not visible
```

retstep means difference between values.

retstep is an optional

retstep: False

vi. dtype - float, int.

dtype is an optional.

Syntax for logspace():

syntax: np.logspace(start index, stop index, num, endpoint, base, dtype)

It has Six parameters.

i. start index is a lower boundary.

ii. stop index is a upper boundary.

iii. num = number of required values in array.

num is an optional.

num By Default: 50

iv. end point: True - stop index will be included. False - stop index will be not included.

end point is optional.

end point By Default: True

v. base - base of log value.

base is an optional.

base By Default: 10

vi. dtype - float, int.

dtype is an optional.

Code: Arrays with numerical ranges

1. arange():

In [58]:

```
import numpy as np
q = np.arange(0,14,2,dtype = "int")
print(q)
```

```
[ 0  2  4  6  8 10 12]
```

In [61]:

```
import numpy as np
r = np.arange(0,10,1,dtype = "float")
print(r)
```

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

In [63]:

```
import numpy as np
s = np.arange(0,10)
print(s)
```

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

In [64]:

```
import numpy as np
t = np.arange(0,10,dtype = "float")
print(t)
```

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

2. linspace():

In [67]:

```
import numpy as np
u = np.linspace(0,10)
print(u)
```

```
[ 0.          0.20408163  0.40816327  0.6122449   0.81632653  1.02040816
  1.2244898   1.42857143  1.63265306  1.83673469  2.04081633  2.24489796
  2.44897959  2.65306122  2.85714286  3.06122449  3.26530612  3.46938776
  3.67346939  3.87755102  4.08163265  4.28571429  4.48979592  4.69387755
  4.89795918  5.10204082  5.30612245  5.51020408  5.71428571  5.91836735
  6.12244898  6.32653061  6.53061224  6.73469388  6.93877551  7.14285714
  7.34693878  7.55102041  7.75510204  7.95918367  8.16326531  8.36734694
  8.57142857  8.7755102   8.97959184  9.18367347  9.3877551   9.59183673
  9.79591837 10.        ]
```

In [68]:

```
import numpy as np
v = np.linspace(0,10,20)
print(v)
```

```
[ 0.          0.52631579  1.05263158  1.57894737  2.10526316  2.63157895
  3.15789474  3.68421053  4.21052632  4.73684211  5.26315789  5.78947368
  6.31578947  6.84210526  7.36842105  7.89473684  8.42105263  8.94736842
  9.47368421 10.        ]
```

In [69]:
```
import numpy as np
w = np.linspace(0,10,20,endpoint = False)
print(w)
```

```
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5
 9.  9.5]
```

In [72]:
```
import numpy as np
x = np.linspace(0,21,10,endpoint = True,retstep = True, dtype = "int")
print(x)
```

```
(array([ 0,  2,  4,  7,  9, 11, 14, 16, 18, 21]), 2.3333333333333335)
```

In [76]:
```
import numpy as np
y= np.linspace(0,21,10,endpoint = True,retstep = False, dtype = "int")
print(y)
```

```
[ 0  2  4  7  9 11 14 16 18 21]
```

In [75]:
```
import numpy as np
z = np.linspace(0,21,10,endpoint = True,retstep = False)
print(z)
```

```
[ 0.          2.33333333  4.66666667  7.          9.33333333 11.66666667
 14.         16.33333333 18.66666667 21.        ]
```

3.logspace():

In [78]:
```
import numpy as np
arr1 = np.logspace(0,10)
arr1
```

Out[78]:
```
array([1.00000000e+00, 1.59985872e+00, 2.55954792e+00, 4.09491506e+00,
       6.55128557e+00, 1.04811313e+01, 1.67683294e+01, 2.68269580e+01,
       4.29193426e+01, 6.86648845e+01, 1.09854114e+02, 1.75751062e+02,
       2.81176870e+02, 4.49843267e+02, 7.19685673e+02, 1.15139540e+03,
       1.84206997e+03, 2.94705170e+03, 4.71486636e+03, 7.54312006e+03,
       1.20679264e+04, 1.93069773e+04, 3.08884360e+04, 4.94171336e+04,
       7.90604321e+04, 1.26485522e+05, 2.02358965e+05, 3.23745754e+05,
       5.17947468e+05, 8.28642773e+05, 1.32571137e+06, 2.12095089e+06,
       3.39322177e+06, 5.42867544e+06, 8.68511374e+06, 1.38949549e+07,
       2.22299648e+07, 3.55648031e+07, 5.68986603e+07, 9.10298178e+07,
       1.45634848e+08, 2.32995181e+08, 3.72759372e+08, 5.96362332e+08,
       9.54095476e+08, 1.52641797e+09, 2.44205309e+09, 3.90693994e+09,
       6.25055193e+09, 1.00000000e+10])
```

In [79]:
```
import numpy as np
arr2 = np.logspace(0,24,10,endpoint = True, dtype = "float")
arr2
```

Out[79]:
```
array([1.00000000e+00, 4.64158883e+02, 2.15443469e+05, 1.00000000e+08,
       4.64158883e+10, 2.15443469e+13, 1.00000000e+16, 4.64158883e+18,
       2.15443469e+21, 1.00000000e+24])
```

In [82]:
```
import numpy as np
arr3 = np.logspace(0,24,10,endpoint = True, base = 2, dtype = "float")
arr3
```

Out[82]:
```
array([1.00000000e+00, 6.34960421e+00, 4.03174736e+01, 2.56000000e+02,
       1.62549868e+03, 1.03212732e+04, 6.55360000e+04, 4.16127661e+05,
       2.64224595e+06, 1.67772160e+07])
```

In [81]:

```
import numpy as np
arr4 = np.logspace(0,24,10,endpoint = False, base = 2, dtype = "float")
arr4
```

Out[81]:

```
array([1.00000000e+00, 5.27803164e+00, 2.78576180e+01, 1.47033389e+02,
       7.76046882e+02, 4.09600000e+03, 2.16188176e+04, 1.14104803e+05,
       6.02248763e+05, 3.17868803e+06])
```

Initializing of Arrays in NumPy

1. zeros() - Array filled with 0's
2. ones() - Array filled with 1's
3. full() - Array filled with required elements
4. eye() - Array diagonal elements filled with 1's
5. empty() - creates an empty array

Syntax for zeros():

syntax: np.zeros(shape,dtype)

dtype = int or float

float is a default dtype

Syntax for ones():

syntax: np.ones(shape, dtype)

dtype = int or float

float is a default dtype

Syntax for full():

full() initializes the elements with the given value.

syntax: np.full(shape, default value)

Syntax for eye():

eye() will take square matrix and it will initialize 1's on all diadonal positions and initializes 0's on all Non-Diagonal positions.

syntax: np.eye(Rows/Columns,dtype)

dtype = int/float

float is a default dtype

Code:Initializing of Arrays in numpy

1.zeros():

In [8]:

```python
import numpy as np
x = np.zeros(6)
print(x)
```

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

In [9]:

```python
import numpy as np
y = np.zeros((4,2))
print(y)
```

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

In [83]:

```python
import numpy as np
arr = np.zeros((3,3))
print(arr)
```

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

In [85]:

```python
arr_1 = np.zeros((3,3),dtype="int")
print(arr_1)
```

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

2.ones():

In [87]:

```python
import numpy as np
arr_2 =np.ones((2,4),dtype="int")
arr_2
```

Out[87]:

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

In [88]:

```python
import numpy as np
arr_3 =np.ones((4,3),dtype="float")
arr_3
```

Out[88]:

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

In [89]:

```python
import numpy as np
arr_3 =np.ones((4,3))
arr_3
```

Out[89]:

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

In [11]:

```python
import numpy as np
z =np.ones((3))
print(z)
```

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

3.full():

```
In [94]: import numpy as np
         arr_4 =np.full((4,3),6)
         arr_4

Out[94]: array([[6, 6, 6],
                [6, 6, 6],
                [6, 6, 6],
                [6, 6, 6]])
```

```
In [95]: import numpy as np
         arr_5 =np.full((4,3),6,dtype="float")
         arr_5

Out[95]: array([[6., 6., 6.],
                [6., 6., 6.],
                [6., 6., 6.],
                [6., 6., 6.]])
```

4.eye():

```
In [97]: import numpy as np
         arr_6 =np.eye(2,dtype="float")
         arr_6

Out[97]: array([[1., 0.],
                [0., 1.]])
```

```
In [98]: import numpy as np
         arr_7 =np.eye(4)
         arr_7

Out[98]: array([[1., 0., 0., 0.],
                [0., 1., 0., 0.],
                [0., 0., 1., 0.],
                [0., 0., 0., 1.]])
```

```
In [100]: import numpy as np
          arr_8 =np.eye(4,dtype = "int")
          arr_8

Out[100]: array([[1, 0, 0, 0],
                 [0, 1, 0, 0],
                 [0, 0, 1, 0],
                 [0, 0, 0, 1]])
```

```
In [19]: import numpy as np
         ar =np.eye(3,5)
         print(ar)

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

```
In [21]: import numpy as np
         arrr =np.eye(5,3)
         print(arrr)

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

5.empty():

In [12]:

```
import numpy as np
arr_9 = np.empty(4)
print(arr_9)
```

```
[2.12199579e-314 8.99726215e-312 8.20148972e-321 3.79442416e-321]
```

In [13]:

```
import numpy as np
arr_9 = np.empty((4,3))
print(arr_9)
```

```
[[9.01183978e-312 2.47032823e-322 0.00000000e+000]
 [0.00000000e+000 1.18831764e-312 2.46567317e+179]
 [5.44928098e-090 4.00769608e+174 1.28272390e+160]
 [1.16006373e-046 3.99910963e+252 4.42625653e-062]]
```

In [15]:

```
import numpy as np
arr_10 = np.empty((2,3))
print(arr_10)
```

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

In [16]:

```
import numpy as np
arr_11 = np.empty((3,2))
print(arr_11)
```

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

In [17]:

```
import numpy as np
arr_12 = np.empty((3,4))
print(arr_12)
```

```
[[9.01183978e-312 2.47032823e-322 0.00000000e+000 0.00000000e+000]
 [1.18831764e-312 2.46567317e+179 5.44928098e-090 4.00769608e+174]
 [1.28272390e+160 1.16006373e-046 3.99910963e+252 4.42625653e-062]]
```

Create NumPy Arrays with Random Numbers

1.rand():

```
The function is used to generate a random value between 0 to 1.
```

2.randn():

```
The function is used to generate a random value close to zero. This may return positive or n
egative numbers as well.
```

3.ranf():

```
The function is used to generate a random value close to zero.

The function for doing random sampling in numpy.

It returns an array of specified shape and fills it with random floats in the half open inte
rval[0.0,1.0].
```

4.randint():

```
The function is used to generate a random number between a given range.
```

1.rand():

In [29]:

```
import numpy as np
array1=np.random.rand(3)
print(array1)
```

```
[0.23733819 0.00032765 0.30554692]
```

In [28]:

```
import numpy as np
array2=np.random.rand(2,5)
print(array2)
```

```
[[0.91841762 0.17478362 0.27413305 0.35954776 0.74250397]
 [0.87368294 0.73851841 0.28560795 0.8033986  0.05023747]]
```

In [27]:

```
import numpy as np
array3=np.random.rand(3,4)
print(array3)
```

```
[[0.82982455 0.84961316 0.00722902 0.83240701]
 [0.45460637 0.01860966 0.9173192  0.0669857 ]
 [0.05150768 0.24446183 0.8453103  0.19683412]]
```

2.randn():

In [31]:

```
import numpy as np
array4=np.random.randn(4)
print(array4)
```

```
[ 1.12882903  1.24703418  0.64960884 -0.33064272]
```

In [33]:

```
import numpy as np
array5=np.random.randn(4,3)
print(array5)
```

```
[[-0.70245921 -2.06635964  0.8308698 ]
 [ 1.0361534   0.63984489  0.67205609]
 [ 1.93107238 -1.06425487 -1.32524385]
 [ 0.13079974 -0.46417547 -0.35565341]]
```

In [35]:

```
import numpy as np
array6=np.random.randn(3,3)
print(array6)
```

```
[[ 1.06994815 -1.12821625 -0.81215523]
 [ 0.21255246  1.08782763  0.52418612]
 [-0.99351535 -0.00888063  0.79051441]]
```

3.randf():

In [37]:

```
import numpy as np
array7=np.random.ranf(5)
print(array7)
```

```
[0.118567   0.35730937 0.77745414 0.88768969 0.69870724]
```

4.randint():

In [40]:

```
import numpy as np
array9=np.random.randint(4,25,6)
print(array9)
```

```
[12 20 11 21  9  5]
```

In [41]:
```
import numpy as np
array10=np.random.randint(2,30,9)
print(array10)
```

```
[28 22  8 22  2 14 11 17  8]
```

In [42]:
```
import numpy as np
array11=np.random.randint(6,35,12)
print(array10)
```

```
[28 22  8 22  2 14 11 17  8]
```

shape() and reshape()

Syntax for shape():

shape() will return tuple specifying indices and no:of elements

syntax: np.shape(variable name of the array)

Syntax for reshape():

reshape() will accept Array_Name and shape dimensions.

syntax: np.reshape(variable name of the array,shape)

Code: shape() and reshape()

1.shape():

In [107]:
```
import numpy as np
a=np.array([1,3,5,7,9])
print(a)
np.shape(a)
```

```
[1 3 5 7 9]
```

Out[107]:
```
(5,)
```

In [106]:
```
import numpy as np
b=np.array([[1,3,5],[2,4,6]])
print(b)
np.shape(b)
```

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

Out[106]:
```
(2, 3)
```

In [108]:
```
import numpy as np
c=np.array([[[1,3,5,9],[2,4,6,8]],[[6,4,2,9],[3,8,5,1]]])
print(c)
np.shape(c)
```

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

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

Out[108]:
```
(2, 2, 4)
```

In [110]:

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

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

Out[110]:

```
(1, 4, 4)
```

In [111]:

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

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

Out[111]:

```
(4, 4)
```

In [115]:

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

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

Out[115]:

```
(3, 4)
```

In [116]:

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

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

Out[116]:

```
(3, 3)
```

In [117]:

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

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

Out[117]:

```
(3, 2)
```

In [49]:

```
import numpy as np
j=np.array([[1,3,5,7],[2,4,6,8]])
print(j)
print(j.ndim)
np.shape(j)
```

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

Out[49]:

```
(2, 4)
```

In [50]:

```
import numpy as np
k = np.array([5,6,7,8],ndmin=5)
print(k)
print(k.ndim)
np.shape(k)
```

```
[[[[[5 6 7 8]]]]]
5
```

Out[50]:

```
(1, 1, 1, 1, 4)
```

In [52]:

```
import numpy as np
l = np.array([[5,6,7,8],[12,34,56,79]],ndmin=6)
print(l)
print(l.ndim)
np.shape(l)
```

```
[[[[[[ 5  6  7  8]
     [12 34 56 79]]]]]]
6
```

Out[52]:

```
(1, 1, 1, 1, 2, 4)
```

In [54]:

```
import numpy as np
m = np.array([[5,6,7,8],[12,34,56,79],[12,34,65,86]],ndmin=4)
print(m)
print(m.ndim)
np.shape(m)
```

```
[[[[ 5  6  7  8]
   [12 34 56 79]
   [12 34 65 86]]]]
4
```

Out[54]:

```
(1, 1, 3, 4)
```

In [56]:

```
import numpy as np
n=np.array([[[1,3,5,9],[2,4,6,8]],[[6,4,2,9],[3,8,5,1]],[[32,34,67,93],[12,46,83,45]]])
print(n)
np.shape(n)
```

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

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

 [[32 34 67 93]
  [12 46 83 45]]]
```

Out[56]:

```
(3, 2, 4)
```

2.reshape():

In [127]:

```
import numpy as np
print(e)
h = np.reshape(e,(6,2))
print(h)
```

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

In [128]:

```
import numpy as np
print(e)
i = np.reshape(e,(3,4))
print(i)
```

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

In [123]:

```
import numpy as np
print(e)
np.reshape(e,(2,6))
```

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

Out[123]:

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

In [57]:

```
import numpy as np
p = np.array([12,14,16,18,20,22])
print(p)
print(p.ndim)
print()
q=p.reshape(2,3)
print(q)
print(q.ndim)
```

```
[12 14 16 18 20 22]
1

[[12 14 16]
 [18 20 22]]
2
```

In [59]:

```
import numpy as np
r = np.array([12,14,16,18,20,22])
print(r)
print(r.ndim)
print()
s=p.reshape(3,2)
print(s)
print(s.ndim)
```

```
[12 14 16 18 20 22]
1

[[12 14]
 [16 18]
 [20 22]]
2
```

In [62]:

```
import numpy as np
t = np.array([12,14,16,18,20,22])
print(t)
print(t.ndim)
print()
u=p.reshape(6,1)
print(u)
print(u.ndim)
```

```
[12 14 16 18 20 22]
1

[[12]
 [14]
 [16]
 [18]
 [20]
 [22]]
2
```

In [63]:

```
import numpy as np
v = np.array([12,14,16,18,20,22])
print(v)
print(v.ndim)
print()
w=v.reshape(1,6)
print(w)
print(w.ndim)
```

```
[12 14 16 18 20 22]
1

[[12 14 16 18 20 22]]
2
```

In [67]:

```
import numpy as np
x = np.array(([23,45,64,86,69,95,34,25,41]))
print(x)
print(x.ndim)
print()
y=x.reshape(3,3)
print(y)
print(y.ndim)
```

```
[23 45 64 86 69 95 34 25 41]
1

[[23 45 64]
 [86 69 95]
 [34 25 41]]
2
```

In [68]:

```
import numpy as np
Z = np.array(([23,45,64,86,69,95,34,25,41,52,93,63]))
print(Z)
print(Z.ndim)
print()
z=Z.reshape(2,3,2)
print(z)
print(z.ndim)
```

```
[23 45 64 86 69 95 34 25 41 52 93 63]
1

[[[23 45]
  [64 86]
  [69 95]]

 [[34 25]
  [41 52]
  [93 63]]]
3
```

In [69]:

```
import numpy as np
A = np.array(([23,45,64,86,69,95,34,25,41,52,93,63]))
print(A)
print(A.ndim)
print()
a=A.reshape(2,3,2)
print(a)
print(a.ndim)
print()
B=a.reshape(-1)
print(B)
print(B.ndim)
```

```
[23 45 64 86 69 95 34 25 41 52 93 63]
1

[[[23 45]
  [64 86]
  [69 95]]

 [[34 25]
  [41 52]
  [93 63]]]
3

[23 45 64 86 69 95 34 25 41 52 93 63]
1
```

In [22]:

```
import numpy as np
l1 = [22,33,44,55,66,77,88,99]
arr_1 = np.array(l1)
print("Array:",arr_1)
print("Size:",arr_1.size)
print("Dimention:",arr_1.ndim)
a = arr_1.reshape(4,2)
print(a)
print("Dimensions:",a.ndim)
```

```
Array: [22 33 44 55 66 77 88 99]
Size: 8
Dimention: 1
[[22 33]
 [44 55]
 [66 77]
 [88 99]]
Dimensions: 2
```

In [23]:

```
import numpy as np
l2 = [22,33,44,55,66,77]
arr_2 = np.array(l2)
print("Array:",arr_2)
print("Size:",arr_2.size)
print("Dimention:",arr_2.ndim)
c = arr_2.reshape(3,2)
print(c)
print("Dimensions:",c.ndim)
d = arr_2.reshape(2,3)
print(d)
```

```
Array: [22 33 44 55 66 77]
Size: 6
Dimention: 1
[[22 33]
 [44 55]
 [66 77]]
Dimensions: 2
[[22 33 44]
 [55 66 77]]
```

In [24]:

```python
import numpy as np
l3 = [12,13,14,15,16,17,18,19]
arr_3 = np.array(l3)
print("Array:",arr_3)
print("Size:",arr_3.size)
print("Dimensions:",arr_3.ndim)
x = arr_3.reshape(2,2,2)
print(x)
print("Dimensions:",x.ndim)
```

```
Array: [12 13 14 15 16 17 18 19]
Size: 8
Dimensions: 1
[[[12 13]
  [14 15]]

 [[16 17]
  [18 19]]]
Dimensions: 3
```

In [ ]:

```

```