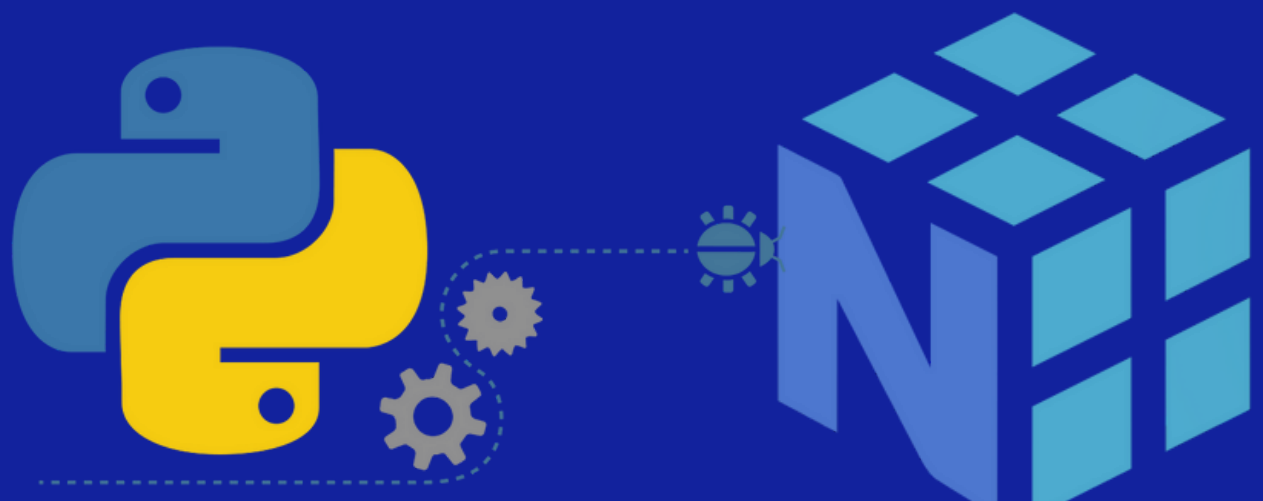
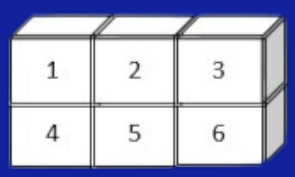


Numpy

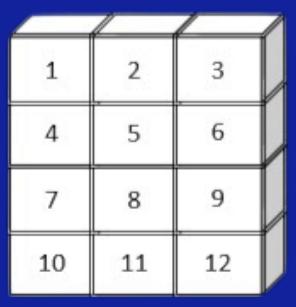
IN PYTHON

Beginner's Code Guide

1-D NumPy Arrays



2-D NumPy Arrays



3-D NumPy Arrays

ABHISHEK MISHRA
@abhishekmishra3

Numpy

Numerical Python is library used for working with arrays

50x faster than traditional Python lists

array object in NumPy is called ndarray

import Numpy in Anaconda, Spyder etc

```
import numpy as np
```

Create a NumPy ndarray Object

```
# using the array() function

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

```
print(arr)
```

```
[1 2 3]
```

```
print(type(arr))
```

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

even using tuple it will make numpy array

```
# using tuple ( )

arr1 = np.array((4,5,6))
```

```
print(arr1)
```

```
[4 5 6]
```

Dimensions in array

0 D array, or Scalars

Each value in an array is a 0-D array

```
arr = np.array(98)
```

```
print(arr)
```

```
98
```

```
print(type(arr))
```

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

```
# check dimension use ndim
```

```
print(arr.ndim)
```

```
0
```

1 d array, uni-dimensional

An array that has 0-D arrays as its elements

```
# Use [ ]

arr2 = np.array([2, 3, 5, 6])
```

```
print(arr)
```

```
2
```

```
print(type(arr))
```

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

```
print(arr.ndim)
```

```
0
```

2 d array, matrix or 2nd order tensors

An array that has 1-D arrays as its elements

```
# nested bracket [ [], [] ]
# float value
```

```
arr = np.array([[1,2.66],[7,4]])


print(arr)
```

```
[[1.   2.66]
 [7.   4.  ]]
```

```
# integer value


arr1 = np.array([[1,2],[3,4]])


print(arr1)
```

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

```
# check dimension


print(arr.ndim)
```

```
2
```

3 d array. used to represent a 3rd order tensor

an array that has 2D arrays as its element is called 3d array

```
# [   [ [],[] ],  [ [],[] ]   ]
```

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

```
print(arr)
```

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

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

```
# check dimension


print(arr.ndim)
```

```
3
```

Higher Dimensional Arrays

using the ndmin argument

```
arr = np.array([1, 2], ndmin=5)
```

```
print(arr)
```

```
[[[[[1 2]]]]]
```

```
# check number of dimensions


print(arr.ndim)
```

```
5
```

Indexing

Access array element

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

print(arr[0])
```

```
1
```

```
print(arr[2] + arr[3])  # 3 + 4 = 7
```

```
7
```

Access 2-D array

Access the element on the first row, second column

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

print(arr[0,1])
```

```
2
```

Access the element on the 2nd row, 3rd column

```
print(arr[1,2])
```

```
6
```

add 1st element of row1 and last element

```
# 1st elment of row1 = 1
# last element = 6. 1+6=7

print(arr[0,0] + arr[1,2])
```

```
7
```

Access 3-D array

Access the 2nd element of the 2nd array of the 1st array

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

```
print(a[0,1,1])
```

```
6
```

Slicing in Numpy array

```
# [start:end], [start:end:step]

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

```
print(arr[1:4])
```

```
[2 3 4]
```

```
# index 2 to the end of the array:

print(arr[2:])
```

```
[3 4 5]
```

negative slicing

```
#start with -4 index i.e element 2

print(arr[-4:-1])
```

```
[2 3 4]
```

```
#retun reverse order

print(arr[::-1])
```

```
# return alternate value

print(arr[::2])
```

Slicing 2-D Arrays

```
arr = np.array([[1,2,3,4,5],
                [6,7,8,9,10]])
```

From the second element, slice elements from index 1 to index 4 (not included):

```
print(arr[1, 1:4])
```

```
[7 8 9]
```

```
#From both elements, return index 2

print(arr[0:2, 2])
```

```
[3 8]
```

From both elements, slice index 1 to index 4 (not included)

this will return a 2-D array

```
print(arr[0:3, 1:4])
```

```
[[[ 3  4]]

 [[60 70]]]
```

Slicing 3-D index

```
new_arr3 = np.array([
    [[1,2,3], [4,5,6], [7,8,9]],
    [[9,8,7], [6,5,4], [3,2,1]]
])
```

```
#START, STOP, END

print(new_arr3[:2, :2, :2])
```

```
[[[1 2]
  [4 5]]

 [[9 8]
  [6 5]]]
```

```
print(new_arr3[:2, 1:, 1:])
```

```
[[[5 6]
  [8 9]]

 [[5 4]
  [2 1]]]
```

Numpy Datatype

refer to data types with one character

below is a list of all data types in NumPy and the characters used to represent them.

```
i - integer
b - boolean
u - unsigned integer
f - float
c - complex float
m - timedelta
M - datetime
O - object
S - string
U - unicode string
V - fixed chunk of memory|other type( void )
```

Checking the Datatype of numpy array

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


print(arr.dtype)
```

```
int32
```

```
arr = np.array(["ab", "cd"])
print(arr.dtype)
```

```
<U2
```

Creating numpy datatype

```
# convert integer to string


a= np.array([1,2],dtype='S')
# capital S
```

```
print(a)
```

```
[b'1' b'2']
```

```
print(a.dtype)
```

```
|S1
```

create an array with data types 4 bytes integer

```
# 4byte integer = int32

a= np.array([1,2,3,4],dtype='i4')
```

```
print(a)
```

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

```
print(a.dtype)
```

```
int32
```

Numpy array shape

it is number of elements in each dimension

```
#print the shape of 2D array

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

print(a.shape)
```

```
(2, 3)
```

```
# shape of 3D array
```

```
# element2, array 3, matrix 1
arr = np.array([[[1,2],
                 [4,5],
                 [7,8]]])

print(arr.shape)
```

```
(1, 3, 2)
```

3-D element =3, 2-D element = 2, 1-D element = 1

```
a= np.array([[[1],[4]],
             [[7],[3]],
             [[2],[9]]])

print(a.shape)
```

```
(3, 2, 1)
```

Joining Numpy arrays

```
arr1 = np.array([1,2,3])
arr2 = np.array([4,5,6])

arr = np.concatenate((arr1, arr2))

print(arr)
```

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

join 2-D array

```
arr1 = np.array([ [1,2], [4,5] ])
arr2 = np.array([ [7,8], [3,6] ])

print(arr1)
print(arr2)
```

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

```
#axis 1 is ROW wise join

a= np.concatenate((arr1,arr2),
                  axis=1)1

print(a)
```

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

```
#axis 0 is Column wise join

b= np.concatenate((arr1,arr2),
                  axis=0)
```

```
print(b)
```

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

Splitting Numpy Array

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

b = np.array_split(a, 3)

print(b)
```

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

Ravel & Flatten

it convert multi dimension array into 1-D array

```
# create 3d array
m = np.array([[[1,2],[3,4],[6,7]]])

print(m)
```

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

```
# convert to 1d array

#n = m.ravel()
n = m.flatten()

print(n)
```

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

Unique Function

```
k = np.array([1,1,2,2,3,3,4,4])

print(k)
```

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

```
x = np.unique(k)

print(x)
```

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

```
#returns the index of unique value

y = np.unique(k,
              return_index = True)
```

```
print(y)
```

```
(array([1, 2, 3, 4]), array([0, 2, 4, 6], dtype=int64))
```

```
#returns the count of duplicate valu

z = np.unique(k,
              return_counts = True)

print(z)
```

```
(array([1, 2, 3, 4]), array([2, 2, 2, 2], dtype=int64))
```

Delete

```
arr = np.array([10,20,30,40])

# delete 20
d = np.delete(arr,[1])

print(d)
```

```
[10 30 40]
```

```
# 2D array
x = np.array([[1,2],
              [3,4],
```

```
                    [5,6]])

print(x)
```

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

```
# delete [5,6] | 2nd row

y = np.delete(x,1,axis=0)

print(y)
```

```
[[1 2]
 [5 6]]
```