PYTHON INTERNSHIP MANUAL

Batch A - 2025 | Week#10 (Numpy Library - Python)

Important Instructions:

- Read the manual carefully and understand every topic before starting the assignment.
- Follow the deadlines properly; marks will be deducted for late submissions.
- Avoid to use AI Tools and make your own logic.
- Completing all tasks in the assignment is compulsory to receive full marks.

- **Manual Posted Date:**

  10th May, 2025 (Saturday)

- **Assignment-6 Deadline:**

  15th May, 2025 (Thursday 11:59pm)

NumPy (Numerical Python)

NumPy is a Python package that stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object.

Where is NumPy used?

Python NumPy arrays provide tools for integrating C, C++, etc. It is also useful in linear algebra, random number capability etc. NumPy array can also be used as an efficient multi-dimensional container for generic data. Now, let me tell you what exactly is a Python NumPy array.

Python NumPy Array

Numpy array is a powerful N-dimensional array object which is in the form of rows and columns. We can initialize NumPy arrays from nested Python lists and access it elements. In order to perform these NumPy operations, the next question which will come in your mind is:

How to install NumPy?

To install Python NumPy, go to your command prompt and type "pip install numpy". Once the installation is completed, go to your IDE (For example: Jupyter Notebook) and simply import it by typing: "import numpy as np"

Python NumPy Array v/s List

Why NumPy over List?

We use python NumPy array instead of a list because of the below three reasons:

1) Less Memory
2) Fast
3) Convinient

1) Less Memory

```
# this code is just to show that python numpy arrays take less memory as compared to python lists
import numpy as np

import time
import sys
S= range(1000)
print("Memory allocated by List = ",sys.getsizeof()*len(S))

D= np.arange(1000)
print("Memory allocated by Numpy array = ",D.size*D.itemsize)
```

```
Memory allocated by List =  28000
Memory allocated by Numpy array =  4000
```

2) Fast

```
# this code is just to show that python numpy arrays take less time as cocmpared to python lists
import numpy as np
import time
import sys

SIZE = 1000000

L1= range(SIZE)
L2= range(SIZE)
A1= np.arange(SIZE)
A2=np.arange(SIZE)

start= time.time()
result=[(x+y) for x,y in zip(L1,L2)]
print("List time in milli_second = ",(time.time()-start)*100)

start=time.time()
result= A1+A2
print(result)
print("Numpy time in milli_second = ",(time.time()-start)*100)
```

```
List time in milli_second =  12.966203689575195
[      0       2       4 ... 1999994 1999996 1999998]
Numpy time in milli_second =  1.4004945755004883
```

Making a NumPy Array

1) Single Dimensional Array

```
# Single dimensional numpy array
import numpy as np
a=np.array([1,2,3])
print("One dimensional",a)
```

```
One dimensional [1 2 3]
```

2) Multi-Dimensional Array

```
#two-dimensional numpy array
a=np.array([[1,2,3],[4,5,6]])
print("two dimensional\n\n",a)

#three-dimensional numpy array
a = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print("\n\nthree dimensional\n\n",a)
```

```
two dimensional

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


three dimensional

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

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

3) Checking Dimensions of the Array

```
# how to check dimensions using ndim
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

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

```
0
1
2
3
```

```
#create an array with 5 dimensions and verify using ndim that it has 5 dimensions

import numpy as np

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

print(arr)
print('number of dimensions :', arr.ndim)
```

```
[[[[[1 2 3 4]]]]]
number of dimensions : 5
```

Converting a List to NumPy Array

```
#Converting list to numpy array

import numpy as np
l = [12.23, 13.32, 100, 36.32]
print("Original List:",l)
a = np.array(l)
print("One-dimensional NumPy array: ",a)
```

```
Original List: [12.23, 13.32, 100, 36.32]
One-dimensional NumPy array:  [ 12.23  13.32 100.    36.32]
```

Append Values to NumPy Array

```
# how to append values to numpy array

import numpy as np
x = [10, 20, 30]
print("Original array:")
print(x)
x = np.append(x, [[40, 50, 60], [70, 80, 90]])
print("After append values to the end of the array:")
print(x)
```

```
Original array:
[10, 20, 30]
After append values to the end of the array:
[10 20 30 40 50 60 70 80 90]
```

Find Common Values in NumPy Arrays

```
# to find common values between two numpy arrays

import numpy as np
array1 = np.array([0, 10, 20, 40, 60])
print("Array1: ",array1)
array2 = [10, 30, 40]
print("Array2: ",array2)
print("Common values between two arrays:")
print(np.intersect1d(array1, array2))
```

```
Array1:  [ 0 10 20 40 60]
Array2:  [10, 30, 40]
Common values between two arrays:
[10 40]
```

Creating NumPy Arrays with zero(s) and one(s)

```
#if you want to create numpy array of zeros then :

# Python Program illustrating
# numpy.zeros method

import numpy as np

b = np.zeros(2, dtype = int)
print("Matrix b : \n", b)

a = np.zeros([2, 2], dtype = int)
print("\nMatrix a : \n", a)

c = np.zeros([3, 3])
print("\nMatrix c : \n", c)
```

```
Matrix b :
 [0 0]

Matrix a :
 [[0 0]
 [0 0]]

Matrix c :
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

```
#if you want to create numpy array of ones then :

# Python Program illustrating
# numpy.ones method

import numpy as np

b = np.ones(2, dtype = int)
print("Matrix b : \n", b)

a = np.ones([2, 2], dtype = int)
print("\nMatrix a : \n", a)

c = np.ones([3, 3])
print("\nMatrix c : \n", c)
```

```
Matrix b :
 [1 1]

Matrix a :
 [[1 1]
 [1 1]]

Matrix c :
 [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

Creating 3x3 Identity Matrix in NumPy Arrays

```
# creating identity matrix with diagonals elements 1 and all other are zero

import numpy as np
x = np.eye(3)
print(x)

y = np.eye(3,4)
print(y)
```

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

```
# program to create a 5x5 zero matrix with elements on the main diagonal equal to 1, 2, 3, 4, 5.

import numpy as np
x = np.diag([1, 2, 3, 4, 5])
print(x)
```

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

Checking Null Values in NumPy Arrays

```
# to check if any value is nan (null) in numpy array using np.isnan()

import numpy as np
a = np.array([1, 0, np.nan, 3])
print("Original array")
print(a)
print("Test element-wise for NaN:")
print(np.isnan(a))
```

```
Original array
[ 1.  0. nan  3.]
Test element-wise for NaN:
[False False  True False]
```

Creating NumPy Arrays Using ‘arange’ Function

```
# "arange of numpy array" is similar to "range of list"

# np.arange(starting,ending,step)

print(np.arange(3))
print(np.arange(3.0))
print(np.arange(3,7))
print(np.arange(3,7,2))

print(np.arange(1,4,2))
```

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

```
#how to create a multi-dimensional array using arange
#it will create array having 3 rows and 4 columns with elements from 10 to 21

a = np.arange(10,22).reshape((3, 4))
print(a)
```

```
[[10 11 12 13]
 [14 15 16 17]
 [18 19 20 21]]
```

Accessing Elements in NumPy Arrays

```
#get first element

import numpy as np

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

print(arr[0])
```

```
1
```

```
#get third and forth and add them

import numpy as np

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

print(arr[2] + arr[3])
```

```
7
```

```
#get element on first row, second column

import numpy as np

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

print('2nd element on 1st row: ', arr[0, 1])
```

```
2nd element on 1st row:  2
```

```
# access third element of second array of first array

import numpy as np

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

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

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

 [[ 7  8  9]
  [10 11 12]]]
6
```

Explaination for Above Example

arr[0, 1, 2] prints the value 6.
And this is why:
The first number represents the first dimension, which contains two arrays: [[1, 2, 3], [4, 5, 6]] and: [[7, 8, 9], [10, 11, 12]] Since we selected 0, we are left with the first array: [[1, 2, 3], [4, 5, 6]]
The second number represents the second dimension, which also contains two arrays: [1, 2, 3] and: [4, 5, 6] Since we selected 1, we are left with the second array: [4, 5, 6]
The third number represents the third dimension, which contains three values: 4 5 6 Since we selected 2, we end up with the third value: 6

NumPy Array Slicing

```
#Numpy Array Slicing is same as that of list slicing that we studied earlier

import numpy as np

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

print(arr[1:5])



import numpy as np

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

print(arr[4:])


import numpy as np

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

print(arr[1:5:2])
```

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

Checking Data Type of NumPy Array

```
# For this, we have a command dtype

import numpy as np

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

print(arr.dtype)
```

```
int32
```

```
#creating an array with dataType 8 byte integer

import numpy as np

arr = np.array([1, 2, 3, 4], dtype='i8')

print(arr)
print(arr.dtype)
```

```
[1 2 3 4]
int64
```

NumPy Array Shape and Reshape

The Shape of an array is the number of elements in each dimension.
Reshape means changing shape of array.

```
# get shape of array

import numpy as np

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

print(arr.shape)
```

```
(2, 4)
```

The example above returns (2, 4), which means that the array has 2 dimensions, and each dimension has 4 elements.

```python
# reshaping 1D to 2D array

import numpy as np

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

newarr = arr.reshape(4, 3)

print(newarr)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

```python
# reshaping from 1D to 3D

import numpy as np

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

newarr = arr.reshape(2, 3, 2)

print(newarr)
```

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

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

Iterating NumPy Arrays

```
#itrating 1D array

import numpy as np

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

for x in arr:
    print(x)
```

```
1
2
3
```

```
#itrating 2D array

import numpy as np

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

for x in arr:
    print(x)
```

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

```
#in above example, it is not iterating each individual element. For that, we need to use nested for loops
import numpy as np

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

for x in arr:
    for y in x:
        print(y)
```

```
1
2
3
4
5
6
```

Iterating NumPy Arrays using nditer()

In above example, iterating through each number of an array we need to use n for loops which can be difficult to write for arrays with very high dimensionality. nditor() solves this problem:

```
import numpy as np

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

for x in np.nditer(arr):
    print(x)
```

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

NumPy Array Join

Joining means putting contents of two or more arrays in a single array.

```
import numpy as np

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]
```

```
# joining with axis=1

import numpy as np

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

arr2 = np.array([[5, 6], [7, 8]])

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

print(arr)
```

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

NumPy Array Join using Stacking

Stacking is same as concatenation, the only difference is that stacking is done along a new axis.
We can concatenate two 1-D arrays along the second axis which would result in putting them one over the other, ie. stacking.
We pass a sequence of arrays that we want to join to the stack() method along with the axis. If axis is not explicitly passed it is taken as 0.

```
# stacking along rows using hstack

import numpy as np

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

arr2 = np.array([4, 5, 6])

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

print(arr)
```

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

```
#stacking along rows using vstack

import numpy as np

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

arr2 = np.array([4, 5, 6])

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

print(arr)
```

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

```
## stack along the height using dstack

import numpy as np

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

arr2 = np.array([4, 5, 6])

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

print(arr)
```

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

NumPy Array Splitting

Splitting is reverse operation of Joining.

Joining merges multiple arrays into one and Splitting breaks one array into multiple.

We use array_split() for splitting arrays, we pass it the array we want to split and the number of splits.

```
import numpy as np

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

newarr = np.array_split(arr, 3)

print(newarr)
```

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

NumPy Array Searching

You can search an array for a certain value, and return the indexes that get a match.

To search an array, use the where() method:

```
# find index where value=4

import numpy as np

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

x = np.where(arr == 4)

print(x)
```

```
(array([3, 5, 6], dtype=int64),)
```

```
import numpy as np

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

x = np.where(arr%2 == 0)

print(x)
```

```
(array([1, 3, 5, 7], dtype=int64),)
```

There is a method called **searchsorted()** which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.

```
# find index where value 10 should be inserted

import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 10)

print(x)
```

```
4
```

NumPy Array Sorting

The NumPy ndarray object has a function called sort(), that will sort a specified array.

```
import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

```
[0 1 2 3]
```

```
#sorts alphabetically
import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr))
```

```
['apple' 'banana' 'cherry']
```

```
#sorting 2D array

import numpy as np

arr = np.array([[3, 2, 4], [5, 0, 1]])

print(np.sort(arr))
```

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

```
# creating a structured array and sorting based on height

import numpy as np
data_type = [('name', 'S15'), ('class', int), ('height', float)]   # where S15 is actually string data type
students_details = [('James', 5, 48.5), ('Nail', 6, 52.5),('Paul', 5, 42.10), ('Pit', 5, 40.11)]
# create a structured array
students = np.array(students_details, dtype=data_type)
print("Original array:")
print(students)
print("Sort by height")
print(np.sort(students, order='height'))
```

```
Original array:
[(b'James', 5, 48.5 ) (b'Nail', 6, 52.5 ) (b'Paul', 5, 42.1 )
 (b'Pit', 5, 40.11)]
Sort by height
[(b'Pit', 5, 40.11) (b'Paul', 5, 42.1 ) (b'James', 5, 48.5 )
 (b'Nail', 6, 52.5 )]
```

NumPy Operations

1) Basic Arithmetic

```
a = np.array([10,32,12])
b = np.array([3,4,5])


print(a+b)
# or
print(np.add(a,b))


print(a-b)
# or
print(np.subtract(a,b))


print(a*b)
# or
print(np.multiply(a,b))


print(a/b)
# or
print(np.divide(a,b))


print(np.mod(a,b)) # it returns remainder after dividing a by b
# or
print(np.remainder(a, b))


print(np.divmod(a, b))  #it returns quotient and remainder both
```

- **Summation**

```
import numpy as np

arr1 = np.array([1, 1, 2])
arr2 = np.array([1, 3, 2])

total_sum = np.sum([arr1,arr2])             # overall sum of all elements in array1 and array2
print(total_sum)

first_arr = np.sum([arr1, arr2], axis=1)    # if axis = 0 means sum across each indidual array
second_arr = np.sum([arr1, arr2], axis=0)   # if axis =1 means adding first element of both array then second and so on
print(first_arr)
print(second_arr)
```

```
10
[4 6]
[2 4 4]
```

- **Product**

```
import numpy as np

arr1 = np.array([1, 1, 2])
arr2 = np.array([1, 3, 1])

total_prod = np.product([arr1,arr2])            # overall product of all elements in array1 and array2
print(total_prod)

first_arr = np.product([arr1, arr2], axis=1)    # if axis = 0 means product across each indidual array
second_arr = np.product([arr1, arr2], axis=0)   # if axis =1 means multiply first element of both array then second and so on
print(first_arr)
print(second_arr)
```

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

- **LCM (Least Common Multiple)**

```
import numpy as np

num1 = 4
num2 = 6

x = np.lcm(num1, num2)

print(x)

print("------------------------------")

# LCM of array
import numpy as np

arr = np.array([3, 6, 9])

x = np.lcm.reduce(arr)

print(x)
```

```
12
------------------------------
18
```

- **GCD (Greatest Common Divisor) OR HCF (Highest Common Factor)**

```
import numpy as np

num1 = 6
num2 = 9

x = np.gcd(num1, num2)

print(x)

print("------------------------------")

import numpy as np

arr = np.array([20, 8, 32, 36, 16])

x = np.gcd.reduce(arr)

print(x)
```

```
3
------------------------------
4
```

- **Cross Product & Dot Product of Matrices**

```
import numpy as np
p = [[1, 0], [0, 1]]
q = [[1, 2], [3, 4]]
print("original matrix:")
print(p)
print(q)
result1 = np.cross(p, q)
result2 = np.cross(q, p)
print("cross product of the said two vectors(p, q):")
print(result1)
print("cross product of the said two vectors(q, p):")
print(result2)
```

```
original matrix:
[[1, 0], [0, 1]]
[[1, 2], [3, 4]]
cross product of the said two vectors(p, q):
[ 2 -3]
cross product of the said two vectors(q, p):
[-2  3]
```

```
import numpy as np
p = [[1, 0], [0, 1]]
q = [[1, 2], [3, 4]]
print("original matrix:")
print(p)
print(q)
result1 = np.dot(p, q)
print("dot product\n",result1)
```

```
original matrix:
[[1, 0], [0, 1]]
[[1, 2], [3, 4]]
dot product
 [[1 2]
 [3 4]]
```

- **Transpose of a Matrix**

```
import numpy as np

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

print(arr1)

arr1_transpose = arr1.transpose()

print(arr1_transpose)
```

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

- **Determinant of a Square Matrix**

```
import numpy as np
from numpy import linalg as LA
a = np.array([[1, 1], [1, 2]])
print("Original 2-d array")
print(a)
print("Determinant of the said 2-D array:")
print(np.linalg.det(a))
```

```
Original 2-d array
[[1 1]
 [1 2]]
Determinant of the said 2-D array:
1.0
```

- **Eigen Values and Eigen Vectors**

```
import numpy as np
m = np.mat("3 -2;1 0")
print("Original matrix:")
print("a\n", m)
w, v = np.linalg.eig(m)
print( "Eigenvalues of the said matrix",w)
print( "Eigenvectors of the said matrix",v)
```

```
Original matrix:
a
 [[ 3 -2]
 [ 1  0]]
Eigenvalues of the said matrix [2. 1.]
Eigenvectors of the said matrix [[0.89442719 0.70710678]
 [0.4472136  0.70710678]]
```

- **Inverse of Matrix**

```
# calculating inverse of a matrix


# Python program to inverse
# a matrix using numpy

# Import required package
import numpy as np

# Taking a 3 * 3 matrix
A = np.array([[6, 1, 1],
              [4, -2, 5],
              [2, 8, 7]])

# Calculating the inverse of the matrix
print(np.linalg.inv(A))
```

```
[[ 0.17647059 -0.00326797 -0.02287582]
 [ 0.05882353 -0.13071895  0.08496732]
 [-0.11764706  0.1503268   0.05228758]]
```

2) Statistics

```
# this will print mean, median, standard deviation, variance, min, max and average
a = np.array([[7, 2], [5, 4]])

print(np.mean(a))
print(np.median(a))
print(np.std(a))
print(np.var(a))
print(np.min(a))
print(np.max(a))
print(np.average(a))
```

```
4.5
4.5
1.8027756377319946
3.25
2
7
4.5
```

```
# to cumpute covariance and correlation of two matrices

import numpy as np
x = np.array([0, 1, 2])
y = np.array([2, 1, 0])
print("\nOriginal array1:")
print(x)
print("\nOriginal array1:")
print(y)
print("\nCovariance matrix of the said arrays:\n",np.cov(x, y))

print("\nCorrelation of the said arrays:\n",np.correlate(x,y))
```

```
Original array1:
[0 1 2]

Original array1:
[2 1 0]

Covariance matrix of the said arrays:
 [[ 1. -1.]
 [-1.  1.]]

Correlation of the said arrays:
 [1]
```

3) Random Numbers

```
#it will generate random number from zero to 100

from numpy import random

x = random.randint(100)

print(x)
```

```
39
```

```
# rand returns random numbers between 0 and 1

from numpy import random

x = random.rand()

print(x)
```

```
0.41265787353277905
```

```
# generate 5 random numbers using rand between 0 and 1

from numpy import random

x = random.rand(5)

print(x)
```

```
[0.31406115 0.04945838 0.00748686 0.63218236 0.49340008]
```

```
#generate random array of size 5 with elements between 0-100

from numpy import random

x=random.randint(100, size=(5))

print(x)
```

```
[87 43 45 20 57]
```

```python
# generate random numbers in 3 rows and 5 columns. Each number between 0-100

from numpy import random

x = random.randint(100, size=(3, 5))

print(x)
```

```
[[24 53 19  3 93]
 [43 31 65 42 95]
 [96 87 15 52 28]]
```

```python
#it will pick any number randomnly from this array

from numpy import random

x = random.choice([3, 5, 7, 9])

print(x)
```

```
3
```

```python
#it will generate random numbers in 3 rows and 5 columns. Each random number will be one of the given numbers in array

from numpy import random

x = random.choice([3, 5, 7, 9], size=(3, 5))

print(x)
```

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