Numpy

NumPy is a fundamental library for scientific computing in Python. It provides support for arrays and matrices, along with a collection of mathematical functions to operate on these data structures. Here we will cover the complete NumPy, focusing on arrays and vectorized operations.

In [2]:
```
"""Import the numpy library"""
import numpy as np
```

In [3]:
```
## create array using numpy, create a 1D array
arr1=np.array([1,2,3,4,5])
print(arr1)
print(type(arr1)) #type of array
print(arr1.shape) #shape of the array
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
(5,)
```

In [4]:
```
## 1 d array
arr2=np.array([1,2,3,4,5])
arr2.reshape(1,5)  ##1 row and 5 columns (conversion)
```

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

In [5]:
```
arr2=np.array([[1,2,3,4,5]])
arr2.shape
```

Out[5]:
```
(1, 5)
```

In [6]:
```
## 2d array
arr2=np.array([[1,2,3,4,5],[2,3,4,5,6]])
print(arr2)
print(arr2.shape)
```

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

In [7]:
```
np.arange(0,10,2).reshape(5,1)
```

Out[7]:
```
array([[0],
       [2],
       [4],
       [6],
       [8]])
```

In [8]:
```
np.ones((3,4))
```

Out[8]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [9]:
```
## identity matrix
np.eye(3)
```

Out[9]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [10]:
```
# Attributes of Numpy Array
arr = np.array([[1, 2, 3], [4, 5, 6]])

print("Array:\n", arr)
print("Shape:", arr.shape)  # Output: (2, 3)
print("Number of dimensions:", arr.ndim)  # Output: 2
print("Size (number of elements):", arr.size)  # Output: 6
print("Data type:", arr.dtype)  # Output: int64 (may vary based on platform 32 bit & 64 bit systems)
print("Item size (in bytes):", arr.itemsize)  # Output: 8 (may vary based on platform)
```

```
Array:
 [[1 2 3]
 [4 5 6]]
Shape: (2, 3)
Number of dimensions: 2
Size (number of elements): 6
Data type: int64
Item size (in bytes): 8
```

In [11]:
```
### Numpy Vectorized Operation
arr1=np.array([1,2,3,4,5])
arr2=np.array([10,20,30,40,50])
```

In [12]:
```
### Element Wise addition
print("Addition:", arr1+arr2)
```

```
Addition: [11 22 33 44 55]
```

In [13]:
```
## Element Wise Substraction
print("Substraction:", arr1-arr2)
```

```
Substraction: [ -9 -18 -27 -36 -45]
```

In [14]:
```
# Element-wise multiplication
print("Multiplication:", arr1 * arr2)
```

```
Multiplication: [ 10  40  90 160 250]
```

In [15]:
```
# Element-wise division
print("Division:", arr1 / arr2)
```

```
Division: [0.1 0.1 0.1 0.1 0.1]
```

Universal Function

In [16]:
```
arr=np.array([2,3,4,5,6])
## square root
print(np.sqrt(arr))
```

```
[1.41421356 1.73205081 2.         2.23606798 2.44948974]
```

In [17]:
```
## Exponential
print(np.exp(arr))
```

```
[  7.3890561   20.08553692  54.59815003 148.4131591  403.42879349]
```

In [18]:
```
## Sine
print(np.sin(arr))
```

```
[ 0.90929743  0.14112001 -0.7568025  -0.95892427 -0.2794155 ]
```

In [19]:
```
## natural log
print(np.log(arr))
```

```
[0.69314718 1.09861229 1.38629436 1.60943791 1.79175947]
```

Array slicing and Indexing

In [20]:
```
arr=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
print("Array : \n", arr)
```

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

In [21]:
```
print(arr[1:,1:3])
```

```
[[ 6  7]
 [10 11]]
```

In [22]:
```
print(arr[0][0])
print(arr[0:2,2:])
```

```
1
[[3 4]
 [7 8]]
```

In [23]:
```
arr[1:,2:]
```

Out[23]:
```
array([[ 7,  8],
       [11, 12]])
```

In [24]:
```
## Modify array elements
arr[0,0]=100
print(arr)
```

```
[[100   2   3   4]
 [  5   6   7   8]
 [  9  10  11  12]]
```

In [25]:
```
arr[1:]=100
print(arr)
```

```
[[100   2   3   4]
 [100 100 100 100]
 [100 100 100 100]]
```

Statistical Concepts--Normalization

In [26]:
```
## To have a mean of 0 and standard deviation of 1
data = np.array([1, 2, 3, 4, 5])
```

In [27]:
```
# Calculate the mean and standard deviation
mean = np.mean(data)
std_dev = np.std(data)
```

In [28]:
```
# Normalize the data
normalized_data = (data - mean) / std_dev
print("Normalized data:", normalized_data)
```

```
Normalized data: [-1.41421356 -0.70710678  0.          0.70710678  1.41421356]
```

In [29]:
```
data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

In [30]:
```
# Mean
mean = np.mean(data)
print("Mean:", mean)
```

```
Mean: 5.5
```

In [31]:
```
# Median
median = np.median(data)
print("Median:", median)
```

```
Median: 5.5
```

In [32]:
```
# Standard deviation
std_dev = np.std(data)
print("Standard Deviation:", std_dev)
```

```
Standard Deviation: 2.8722813232690143
```

In [33]:
```
# Variance
variance = np.var(data)
print("Variance:", variance)
```

```
Variance: 8.25
```

In [34]:
```
## Logical operation
data=np.array([1,2,3,4,5,6,7,8,9,10])

data[(data>=5) & (data<=8)]
```

Out[34]:
```
array([5, 6, 7, 8])
```

NumPy Arithmetic Array Operations

In [35]:
```
import numpy as np
a1=np.array([[2,2,2],[3,3,3]])
```

In [36]:
```
print(a1+2) # adding 2 in each row and column
```

```
[[4 4 4]
 [5 5 5]]
```

In [37]:
```
a1+a1 # adding a1 matrix twice
```

Out[37]:
```
array([[4, 4, 4],
       [6, 6, 6]])
```

In [39]:
```
a2=a1*2
print(a2)
```

```
[[4 4 4]
 [6 6 6]]
```

In [40]:
```
a2//2
print(a2)
```

```
[[4 4 4]
 [6 6 6]]
```

In [42]:
```
a3=a2//a1
print(a3)
```

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

In [43]:
```
6/a1
```

Out[43]:
```
array([[3., 3., 3.],
       [2., 2., 2.]])
```

In [44]:
```
a1**2 # Power **
```

Out[44]:
```
array([[4, 4, 4],
       [9, 9, 9]])
```

In [45]:
```
k=5/2
print(k)
```

```
2.5
```

In [47]:
```
a1>a2   # Comparison operations
```

Out[47]:
```
array([[False, False, False],
       [False, False, False]])
```

In [48]:
```
a1<a2 # Comparison operations
```

Out[48]:
```
array([[ True,  True,  True],
       [ True,  True,  True]])
```

Indexing and Slicing in Numpy Array

In [49]:
```
a4=np.array([0,1,2,3,4,5,6,7,8,9])
```

In [50]:
```
z=np.arange(10)
z
```

Out[50]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [52]:
```
z[5] # Indexing
```

Out[52]:
```
5
```

In [53]:
```
z[5:8] # Slicing
```

Out[53]:
```
array([5, 6, 7])
```

In [54]:
```
z[::-1]
```

Out[54]:
```
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

In [55]:
```
z[::2]
```

Out[55]:
```
array([0, 2, 4, 6, 8])
```

Boolean Indexing / Masking

In [56]:
```
a=np.arange(10)
mask=a%3==0
print(mask)
```

```
[ True False False  True False False  True False False  True]
```

In [57]:
```
a[mask]
```

Out[57]:
```
array([0, 3, 6, 9])
```

Reshaping, Transposing, Swaping

In [58]:
```
a=np.arange(16)
print(a1)
```

```
[[2 2 2]
 [3 3 3]]
```

In [60]:
```
a1=a.reshape((4,4), order='F') #converting from 1d to nd matrix using reshaping or reshape function
print(a1)
```

```
[[ 0  4  8 12]
 [ 1  5  9 13]
 [ 2  6 10 14]
 [ 3  7 11 15]]
```

In [61]:
```
a1.T  # Transpose
```

Out[61]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [63]:
```
b=a.reshape((2,2,4)) # Reshape
print(b)
```

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

 [[ 8  9 10 11]
  [12 13 14 15]]]
```

In [64]:
```
b=a.reshape((1,2,8))
print(b)
```

```
[[[ 0  1  2  3  4  5  6  7]
  [ 8  9 10 11 12 13 14 15]]]
```

In [65]:
```
print(b.shape)
print(b.size)
print(b.ndim)
```

```
(1, 2, 8)
16
3
```

Dot Product

In [66]:
```
print(a1)
```

```
[[ 0  4  8 12]
 [ 1  5  9 13]
 [ 2  6 10 14]
 [ 3  7 11 15]]
```

In [67]:
```
print(a1.T)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```

In [68]:
```
np.dot(a1,a1.T)
```

Out[68]:
```
array([[224, 248, 272, 296],
       [248, 276, 304, 332],
       [272, 304, 336, 368],
       [296, 332, 368, 404]])
```

In [69]:
```
a1.swapaxes(1,0)
```

Out[69]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Functions in Array

In [70]:
```
a=np.array([3,4,2,1,9,8,6,7,0,5])
a.sort() #shorting the data
print(a)
```

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

In [71]:
```
np.square(a) #for sqauare of any numbeer
```

Out[71]:
```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

In [72]:
```
np.sqrt([30,265,9000]) # Square root
```

Out[72]:
```
array([ 5.47722558, 16.2788206 , 94.86832981])
```

In [73]:
```
np.abs([-1.5,-9.5,-999.0]) # Always give +ve result
```

Out[73]:
```
array([  1.5,   9.5, 999. ])
```

In [85]:
```
k=np.array([4,6,8,9])
k2=np.array([2,3,4,3])
np.add(k,k2) # addig two matrix
```

Out[85]:
```
array([ 6,  9, 12, 12])
```

In [86]:
```
np.subtract(k,k2) #Substract two matrix
```

Out[86]:
```
array([2, 3, 4, 6])
```

In [87]:
```
np.multiply(k,k2) #Multiply two matrix
```

Out[87]:
```
array([ 8, 18, 32, 27])
```

In [88]:
```
np.divide(k,k2) # Divide two matrix
```

Out[88]:
```
array([2., 2., 2., 3.])
```

In [89]:
```
np.maximum(k,k2)
```

Out[89]:
```
array([4, 6, 8, 9])
```

In [90]:
```
np.minimum(k,k2)
```

Out[90]:
```
array([2, 3, 4, 3])
```

In [91]:
```
np.power(k,k2) # Gives the power k**k2
```

Out[91]:
```
array([  16,  216, 4096,  729])
```

In [92]:
```
np.greater(k,k2) # Result as in boolean
```

Out[92]:
```
array([ True,  True,  True,  True])
```

In [93]:
```
np.less(k,k2) # Result as in boolean
```

Out[93]:
```
array([False, False, False, False])
```

In [94]:
```
np.concatenate((k,k2)) # Concate two matrix to create one matrix as result
```

Out[94]:
```
array([4, 6, 8, 9, 2, 3, 4, 3])
```

In [95]:
```
a1=np.array([5,7,8]) #1D
a2=np.array([[10,50,30],[20,40,60]]) #2d list with list
a3=np.array((23,84,28)) #creating an array with a tupple
print(a1)
print()
print(a2)
print()
print(a3)
print()
print(a1.shape)
```

```
print(a2.shape)
print(a3.shape)
print(a1.dtype)
print(a2.sum(axis=0))
print(a2.sum(axis=1))
print(a1.ndim)
print(a2.ndim)
print(a3.ndim)
```

```
[5 7 8]

[[10 50 30]
 [20 40 60]]

[23 84 28]

(3,)
(2, 3)
(3,)
int64
[30 90 90]
[ 90 120]
1
2
1
```

In [96]:
```
print(a1.max())
print(a2.max())
print(a3.max())
```

```
8
60
84
```

In [97]:
```
a4=np.array([15,55,34,45,90]) #1d data
print(a4)
```

```
[15 55 34 45 90]
```

In [98]:
```
a4=np.array([15,55,34,45,90],ndmin=2) #1d data to 2d data
print(a4)
```

```
[[15 55 34 45 90]]
```

In [99]:
```
print(a4.size)
print(a4.shape)
```

```
5
(1, 5)
```

In [100…
```
a5=np.array([25,32,87],dtype=complex) #array using d type parameter which implies the creation of an array with the desired data
print(a5)
```

```
[25.+0.j 32.+0.j 87.+0.j]
```

In [101…
```
print(a5.shape)
print(a5.size)
```

```
(3,)
3
```

In [102…
```
a5=np.array([25,32,87],dtype=float)
print(a5)
```

```
[25. 32. 87.]
```

In [104…
```
a2=np.array([[10,50,30],[20,40,60]],dtype="int32")  #2d data 2rows 3 columns
print(a2)
print(a2.shape)
```

```
[[10 50 30]
 [20 40 60]]
(2, 3)
```

In [106…
```
a2.shape=(3,2) #converting 2rows 3 colums to 3 rows to 2 columns
print(a2.shape)
```

```
(3, 2)
```

In [107…
```
a2.shape=(6,1)
print(a2)
```

```
[[10]
 [50]
 [30]
 [20]
 [40]
 [60]]
```

In [108…
```
a5=np.ones(10) #  numpy.ones() function returns a new array of given shape and type, with ones.
print(a5)
```

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

In [109…
```
a5=np.zeros(10) # numpy.zeros() function returns a new array of given shape and type, with zeros.
print(a5)
```

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

In [110…
```
a2=np.array([[10,50,30],[20,40,60]]) #2d list with list
print(a2)
```

```
[[10 50 30]
 [20 40 60]]
```

In [111…
```
type(a2)
```

Out[111]:
```
numpy.ndarray
```

In [115…
```
print(a2.ndim)
print(a2.shape)
```

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

In [116…
```
a6=np.array([1.1,3.2,4.5])
print(a6)
print(type(a6))
print(a6.ndim)
print(a6.shape)
print(a6.size)
print(a6.dtype)
```

```
[1.1 3.2 4.5]
<class 'numpy.ndarray'>
1
(3,)
3
float64
```

In [117…
```
a7=np.array([1,1.6,"sanad"])
print(a7)
print(type(a7))
```

```
print(a7.ndim)
print(a7.shape)
print(a7.size)
print(a7.dtype)
```

```
['1' '1.6' 'sanad']
<class 'numpy.ndarray'>
1
(3,)
3
<U32
```

In [118…

```
a8=np.array('d',[5,6.9,"sanad"]) #only homogenous characters
print(a8) # result error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-118-d5fd82be4a23> in <cell line: 1>()
----> 1 a8=np.array('d',[5,6.9,"sanad"]) #homogenous characters
      2 print(a8)

TypeError: Field elements must be 2- or 3-tuples, got '5'
```

In [119…

```
a8=np.array('f',[5,6.9,"sanad"])
print(a8) # result error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-119-8363deecdf31> in <cell line: 1>()
----> 1 a8=np.array('f',[5,6.9,"sanad"])
      2 print(a8)

TypeError: Field elements must be 2- or 3-tuples, got '5'
```

In [120…

```
a8=np.array([5,6.9,"sanad"],dtype="int64")
print(a8) # result error
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-120-b4feb1367ba1> in <cell line: 1>()
----> 1 a8=np.array([5,6.9,"sanad"],dtype="int64")
      2 print(a8) # result error

ValueError: invalid literal for int() with base 10: 'sanad'
```

In [121…
```
a8=np.array([5,6.9,5+6j],dtype="int64")
print(a8) # result error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-121-b99e728875be> in <cell line: 1>()
----> 1 a8=np.array([5,6.9,5+6j],dtype="int64")
      2 print(a8) # result error

TypeError: int() argument must be a string, a bytes-like object or a real number, not 'complex'
```

In [122…
```
a8=np.array([5,6.9,9.8],dtype="int64")
print(a8)
```

```
[5 6 9]
```

In [123…
```
b1=np.array([5.5,7.8,8.9])
print(b1)
b1[0] # Indexing
```

```
[5.5 7.8 8.9]
```
Out[123]:
```
5.5
```

In [124…
```
b1=np.array([[5.5,7.8],[9.9,5.6]])
print(b1)
print()
print()
print()
print()
b1=np.zeros((2,2))
print(b1)
```

```
[[5.5 7.8]
 [9.9 5.6]]




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

In [125…
```
b1=np.ones((1,2))
print(b1)
```

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

In [126…
```
b2=np.array([1.1,2.4,3.6,])
for item in range(0,3):
  print(b2[item])
```

```
1.1
2.4
3.6
```

In [127…
```
k=np.full((2,2),15) #if we need to add particular data in a predifine matrix.
print(k)
```

```
[[15 15]
 [15 15]]
```

In [128…
```
k2=np.eye(3) #predefine function in numpy
print(k2) # Return a 2-D array with ones on the diagonal and zeros elsewhere.
```

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

In [129…
```
k2=np.random.random((3,3)) #use of random function
print(k2)
```

```
[[0.54315534 0.11012298 0.45823678]
 [0.1036917  0.66571015 0.72377049]
 [0.26934651 0.68306865 0.35557191]]
```

In [131…
```
k3=np.array([1.1,2.2,3.3])
k3[0]=7.5 # Replace the value with index number 0
print(k3)
```

```
[7.5 2.2 3.3]
```

In [132…
```
np.insert(k3,2,1.1)
print(k3)
```

```
[7.5 2.2 3.3]
```

In [133…
```
import array as arr #library of python.
```

In [134…
```
k4=arr.array("d",[1.1,2.3,5.6]) #adding more data in list using array library
k4.insert(1,7.7)
```

```
print(k4)
```

```
array('d', [1.1, 7.7, 2.3, 5.6])
```

In [135…
```
k4.pop(2)
print(k4)
```

```
array('d', [1.1, 7.7, 5.6])
```

In [136…
```
k4.remove(7.7) #remove
```

In [137…
```
print(k4)
```

```
array('d', [1.1, 5.6])
```

In [138…
```
k4.append(9.8) #append work in array but not in numpy
k4
```

Out[138]:
```
array('d', [1.1, 5.6, 9.8])
```

In [139…
```
k4[2]=10.5 #updating the list
print(k4)
```

```
array('d', [1.1, 5.6, 10.5])
```

In [140…
```
k4.index(1.1) #searching the data from list
```

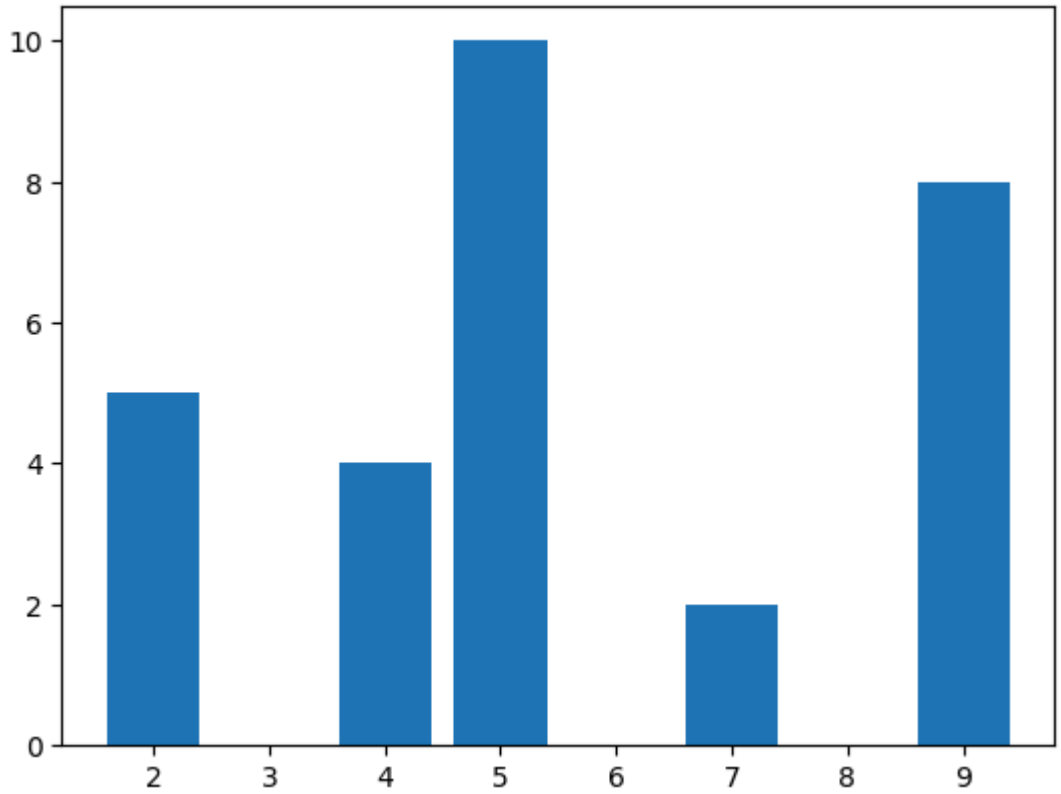
Out[140]:
```
0
```

In [141…
```
# Check the NumPy Version

import numpy as np
np.__version__
```

Out[141]:
```
'1.26.4'
```

In [142…
```
from matplotlib import pyplot as plt
x=[5,2,9,4,7]
y=[10,5,8,4,2]
plt.bar(x,y)
plt.show() # Representing the values in graph view
```



In [143…
```
l=list(range(10))
print(l)
type(l[0])
```

Out[143]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
int
```

In [144…
```
string=([str(item) for item in l]) #converting list to str
print(string)
type(string[0])
```

Out[144]:
```
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
str
```

In [145…
```
l2=[False,100,57.5,5+4j,"dataloves"]
datatype=([type(item) for item in l2])
```

```
print((datatype))
```

```
[<class 'bool'>, <class 'int'>, <class 'float'>, <class 'complex'>, <class 'str'>]
```

In [146…
```
a0=np.zeros(10)
np.dtype=int
print(a0)
```

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

In [147…
```
a0=np.zeros(10,dtype=int)
print(a0)
```

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

In [148…
```
a0=int(np.zeros(10)) #not able to do the type cating in numpy array list can perform.
print(a0)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-148-0348fe3cf087> in <cell line: 1>()
----> 1 a0=int(np.zeros(10)) #not able to do the type cating in numpy array list can perform.
      2 print(a0)

TypeError: only length-1 arrays can be converted to Python scalars
```

In [150…
```
s=np.random.random(10)
print(s)
```

```
[0.25080748 0.97039983 0.37234952 0.23963749 0.23585357 0.31762815
 0.16198949 0.71683042 0.08610264 0.98098685]
```

In [151…
```
import numpy as np
np.random.normal(0,1,(3,3)) # Error as Result
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/IPython/core/formatters.py in __call__(self, obj)
    700                 type_printers=self.type_printers,
    701                 deferred_pprinters=self.deferred_printers)
--> 702             printer.pretty(obj)
    703             printer.flush()
    704             return stream.getvalue()

/usr/local/lib/python3.10/dist-packages/IPython/lib/pretty.py in pretty(self, obj)
    392                         if cls is not object \
    393                                 and callable(cls.__dict__.get('__repr__')):
--> 394                             return _repr_pprint(obj, self, cycle)
    395
    396             return _default_pprint(obj, self, cycle)

/usr/local/lib/python3.10/dist-packages/IPython/lib/pretty.py in _repr_pprint(obj, p, cycle)
    698     """A pprint that just redirects to the normal repr function."""
    699     # Find newlines and replace them with p.break_()
--> 700     output = repr(obj)
    701     lines = output.splitlines()
    702     with p.group():

/usr/local/lib/python3.10/dist-packages/numpy/core/arrayprint.py in _array_repr_implementation(arr, max_line_width, precision, s
uppress_small, array2string)
   1497         class_name = "array"
   1498
-> 1499     skipdtype = dtype_is_implied(arr.dtype) and arr.size > 0
   1500
   1501     prefix = class_name + "("

/usr/local/lib/python3.10/dist-packages/numpy/core/arrayprint.py in dtype_is_implied(dtype)
   1438     array([1, 2, 3], dtype=int8)
   1439     """
-> 1440     dtype = np.dtype(dtype)
   1441     if _format_options['legacy'] <= 113 and dtype.type == bool_:
   1442         return False

TypeError: int() argument must be a string, a bytes-like object or a real number, not 'numpy.dtypes.Float64DType'
```

In [152…

```
np.random.normal(0,1,(3,3)) # Error as Result
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/IPython/core/formatters.py in __call__(self, obj)
    700                 type_printers=self.type_printers,
    701                 deferred_printers=self.deferred_printers)
--> 702             printer.pretty(obj)
    703             printer.flush()
    704             return stream.getvalue()

/usr/local/lib/python3.10/dist-packages/IPython/lib/pretty.py in pretty(self, obj)
    392                         if cls is not object \
    393                                 and callable(cls.__dict__.get('__repr__')):
--> 394                             return _repr_pprint(obj, self, cycle)
    395
    396             return _default_pprint(obj, self, cycle)

/usr/local/lib/python3.10/dist-packages/IPython/lib/pretty.py in _repr_pprint(obj, p, cycle)
    698     """A pprint that just redirects to the normal repr function."""
    699     # Find newlines and replace them with p.break_()
--> 700     output = repr(obj)
    701     lines = output.splitlines()
    702     with p.group():

/usr/local/lib/python3.10/dist-packages/numpy/core/arrayprint.py in _array_repr_implementation(arr, max_line_width, precision, s
uppress_small, array2string)
   1497         class_name = "array"
   1498
-> 1499     skipdtype = dtype_is_implied(arr.dtype) and arr.size > 0
   1500
   1501     prefix = class_name + "("

/usr/local/lib/python3.10/dist-packages/numpy/core/arrayprint.py in dtype_is_implied(dtype)
   1438     array([1, 2, 3], dtype=int8)
   1439     """
-> 1440     dtype = np.dtype(dtype)
   1441     if _format_options['legacy'] <= 113 and dtype.type == bool_:
   1442         return False

TypeError: int() argument must be a string, a bytes-like object or a real number, not 'numpy.dtypes.Float64DType'
```

How numpy is faster then list?

In [153…
```
a=np.arange(1000000) # Define a numpy array
l=list(range(1000000)) # Define a list with range of 0 to 1000000
```

In [154…
```
%timeit for _ in range(200):a2=a*2 #Use the timeit function, Loop time is very less as compare to list
```

```
446 ms ± 61.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [155…
```
%timeit for _ in range(200): l2=l*2 # Loop process time iis very huge as compare to array
```

```
5.41 s ± 596 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [156…
```
#creating n-d arrays

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

d2=[[1,2,3],[4,5,6]]
a2=np.array(d2)
print(d2)
```

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

In [157…
```
d3=np.array(d,dtype=np.float64)
print(d3)
```

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

In [158…
```
print(a1.dtype,a1.shape,a1.ndim)
print(a2.dtype,a2.shape,a2.ndim)
```

```
int64 (5,) 1
int64 (2, 3) 2
```

In [160…
```
print(np.arange(10))
print(np.ones((2,3)))
```

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

In [161…
```
print(np.full((2,3),fill_value=5)) #filling the matrix with 5
```

```
[[5 5 5]
 [5 5 5]]
```

In [162…
```
print(np.empty(5))
```

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

In [164…
```
print(np.zeros((3,3)))
```

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

astype type casting

In [168…
```
import numpy as np
```

In [169…
```
a=np.array([1.2,2.5,600])
b=a.astype(np.int32)
print(b)
```

```
[  1   2 600]
```

In [170…
```
a1=np.array([[2,2,2],[3,3,3]])
a2=a1+2
print(a2)
```

```
[[4 4 4]
 [5 5 5]]
```

In [171…
```
a3=a1+a1
print(a3)
```

```
[[4 4 4]
 [6 6 6]]
```

In [172…
```
a2=a1*2
```

In [173…
```
print(a2)
```

```
[[4 4 4]
 [6 6 6]]
```

In [174…
```
a4=a1//2
print(a4)
```

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

Basic Mathematical and statistical functions

In [175…
```
import numpy as np
a=np.array([3,4,5,6,7,8]) #finding mean (Average  of all number)
a.mean()
```

Out[175]:
```
5.5
```

In [176…
```
np.mean(a) # Mean of the array
```

Out[176]:
```
5.5
```

In [177…
```
a.sum() # Sum of the array
```

Out[177]:
```
33
```

In [178…
```
a.min() # Minimum Value within the array
```

Out[178]:
```
3
```

In [179…
```
a.max()  #Maximum Value within the array
```

Out[179]:
```
8
```

In [180…
```
import statistics as st
import numpy as np
b=np.array([1,1,1,2,3,4,5,6,4,5,6,8])
print(st.mode(b)) # Mode of the array
```

```
1
```

In [181…
```
np.median(a) #middle data of list
```

Out[181]: 5.5

In [182…
```
data=np.array([2,4,6,8,10])
np.median(data) # Median value of the array
```

Out[182]: 6.0

How to get the median (middle) of any data

- short the data
- find the middle value
- if their are two middle value, the we will find the avergae of those two value that will call as the median of the data

In [183…
```
data=[4,5,6,7,2,3]
np.median(data)
```

Out[183]: 4.5

In [184…
```
data=[2,4,6,8,10,1000] #outlaier
np.mean(data)
```

Out[184]: 171.66666666666666

How to find the Variance of any Data

- Calculate the mean of the data
- Calculate the distance from the points to the mean
- Calculate the squared distance
- Calculate the sum of the squared distance
- Divide by the total number of values

In [185…
```
sata=[2,4,6,8,10]
np.var(sata)
```

Out[185]: 8.0

Seed Function

Seed function in numpy, fixed the data while using random in numpy (any positive integer will call as seed value)

In [186...
```
np.random.seed(0)
```

In [187...
```
x1=np.random.randint(10,size=6)
print(x1)
```

```
[5 0 3 3 7 9]
```

In [188...
```
x2=np.random.randint(10,size=(3,4))
print(x2)
```

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

In [189...
```
x3=np.random.randint(10,size=(3,4,5))
print(x3)
```

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

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

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

In [191...
```
print(x3.ndim, x3.shape, x3.size, x3.dtype, x3.itemsize, x3.nbytes)
```

```
3 (3, 4, 5) 60 int64 8 480
```

In [193…
```
print(x1)
print(x1[0])
print(x1[4])
print(x1[-1])
print(x1[-2])
```

```
[5 0 3 3 7 9]
5
7
9
7
```

In [194…
```
print(x2)
```

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

In [195…
```
print(x2[0,0])
print(x2[2,0])
print(x2[2,-1])
```

```
3
1
7
```

In [196…
```
x2[0,0]=100 # Replacing the Value
print(x2)
```

```
[[100   5   2   4]
 [  7   6   8   8]
 [  1   6   7   7]]
```

In [198…
```
x2[0,0]=1.1 #truncate the value floor data can't be insterted in matrix but we can add floor however we can get the integer value
print(x2)
```

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

In [199…
```
x=np.arange(10)
print(x)
```

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

In [200…
```
print(x[0:5])
print(x[5:])
print(x[4:7])
print(x[::2])
print(x[1::2])
print(x[::-1])
print(x[5::-2])
```

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

In [201…
```
print(x2)
```

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

In [203…
```
print(x2[0:2])
```

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

In [204…
```
print(x2[:2,:3]) #use of sclicing (before:-row after :-column)
```

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

In [205…
```
print(x2[:3,:2])
print(x2[0:3:2])
print(x2[:3,::2])
print(x2[::-1])
print(x2[::-1,::-1])
print(x2[:,0])
print(x2[0,:])
print(x2[0])
print(x2[:,1:2])
```

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

In [207…
```
k=x2[:2,:2]
print(k)
```

```
[[1 5]
 [7 6]]
```

In [208…
```
np.random.seed(0)
x2=np.random.randint(10,size=(3,4))
print(x2)
```

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

In [209…
```
k=x2[:2,:2]
print(k)
```

```
[[5 0]
 [7 9]]
```

In [210…
```
k[0,0]=777
print(k)
```

```
[[777   0]
 [  7   9]]
```

In [211…
```
print(x2)
```

```
[[777   0   3   3]
 [  7   9   3   5]
 [  2   4   7   6]]
```

In [212…
```
k3=x2[:2,:2].copy()
print(k3)
```

```
[[777   0]
 [  7   9]]
```

In [213…
```
k3[0,0]=444
print(k3)
```

```
[[444   0]
 [  7   9]]
```

In [214…
```
print(x2)
```

```
[[777   0   3   3]
 [  7   9   3   5]
 [  2   4   7   6]]
```

Reshaping

In [215…
```
g=np.arange(1,10).reshape((3,3))
print(g)
```

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

In [217…
```
x=np.array([1,2,3]).reshape((1,3))
print(x)
```

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

In [218…
```
x=np.array([1,2,3]).reshape((3,1))
print(x)
```

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

- Concatination and Splitting
- vstack
- hstack

In [219…
```
x=np.array([1,2,3])
y=np.array([3,2,1])
k=np.concatenate([x,y])
print(k)
```

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

In [220…
```
z=[999,999,999]
l=np.concatenate([k,z])
print(l)
```

```
[  1   2   3   3   2   1 999 999 999]
```

In [221…
```
print(np.concatenate([x,y,z]))
```

```
[  1   2   3   3   2   1 999 999 999]
```

In [222…
```
h=np.array([[1,2,3],[4,5,6]])
print(h)
```

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

In [224…
```
print(np.concatenate([h,h]))
```

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

In [225…
```
#axis=1 convert to horizontal length # axis=0 convert to vertical
print(np.concatenate([h,h],axis=1))
```

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

In [227…
```
print(x)
y=np.array([[9,8,7],[6,5,4]])
print(np.vstack([x,y])) #vertical stacking directely add variables
```

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

In [228…
```
print(y)
```

```
[[9 8 7]
 [6 5 4]]
```

In [229…
```
z=np.array([[888],[888]])
print(np.hstack([y,z])) # Horizontal Stack
```

```
[[  9   8   7 888]
 [  6   5   4 888]]
```

Splitting

In [230…
```
x1=[1,2,3,99,99,3,2,1]
y1,y2,y3=np.split(x1,[3,5])
print(y1,y2,y3)
```

```
[1 2 3] [99 99] [3 2 1]
```

In [231…
```
x1=[1,2,3,99,99,3,2,1]
y1,y2,y3=np.split(x1,[2,5])
print(y1,y2,y3)
```

```
[1 2] [ 3 99 99] [3 2 1]
```

In [232…
```
x1=[1,2,3,99,99,3,2,1]
y1,y2,y3=np.split(x1,[2,6])
print(y1,y2,y3)
```

```
[1 2] [ 3 99 99  3] [2 1]
```

In [233…
```
x1=[1,2,3,99,99,3,2,1]
y1,y2,y3=np.split(x1,[2,4])
print(y1,y2,y3)
```

```
[1 2] [ 3 99] [99  3  2  1]
```

In [234...
```
x1=[1,2,3,99,99,3,2,1]
y1,y2,y3=np.split(x1,[4,4])
print(y1,y2,y3)
```

```
[ 1  2  3 99] [] [99  3  2  1]
```

- n-split points lead to n+1 sub array.
- np.hsplit and np.vsplit both are function on n-split

In [235...
```
l1=np.arange(16).reshape((4,4))
print(l1)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```

In [236...
```
upper,lower=np.vsplit(l1,[2])
print(upper)
print(lower)
```

```
[[0 1 2 3]
 [4 5 6 7]]
[[ 8  9 10 11]
 [12 13 14 15]]
```

In [237...
```
left,right=np.hsplit(l1,[2])
print(left)
print(right)
```

```
[[ 0  1]
 [ 4  5]
 [ 8  9]
 [12 13]]
[[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
```

In [238...
```
left,right=np.hsplit(l1,[3])
print(left)
```

```
print(right)
```

```
[[ 0  1  2]
 [ 4  5  6]
 [ 8  9 10]
 [12 13 14]]
[[ 3]
 [ 7]
 [11]
 [15]]
```

Numpy (universal function)

Universal function in arithematic operations, all these functions works with np.add, np.substract, np.mod etc

- add
- multiply
- substract
- negative
- divide
- mod
- power
- floor_divide

In [239…

```
import numpy as np
x=np.arange(4)
print(x)
print(x+5)
print(x-5)
print(x*5)
print(x/2)
print(x//2)
print(-x)
print(x**2)
```

```
print(x%2)
print(-(0.5*x+1)**2)
```

```
[0 1 2 3]
[5 6 7 8]
[-5 -4 -3 -2]
[ 0  5 10 15]
[0.  0.5 1.  1.5]
[0 0 1 1]
[ 0 -1 -2 -3]
[0 1 4 9]
[0 1 0 1]
[-1.   -2.25 -4.   -6.25]
```

In [240…
```
print(np.add(x,2))
```

```
[2 3 4 5]
```

In [241…
```
abs(-6)
```

Out[241]:
```
6
```

In [243…
```
a=np.array([-1,-2,-3,-4,-5]) #abs will work with numpy
print(abs(a))
```

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

In [247…
```
a=[-1,-2,-3,-4,-5] #abs will not work in list
print(np.abs(a))
```

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

In [248…
```
print(np.absolute(a))
```

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

In [249…
```
x=np.arange(1,6)
print(np.add.reduce(x))
```

```
15
```

In [250…
```
print(np.multiply.reduce(x))
```

```
120
```

In [251…
```
print(np.add.accumulate(x))
```

```
[ 1  3  6 10 15]
```

In [252…
```
print(np.multiply.accumulate(x))
```

```
[  1   2   6  24 120]
```

In [253…
```
n=np.random.random(100)
sum(n) #sum of all random values in array
```

Out[253]:
```
52.77225132363466
```

In [254…
```
print(np.sum(n))
```

```
52.77225132363466
```

In [ ]: