BASIC FUNTIONALITIES AND USAGE ON NUMPY PACKAGE

In [1]:
```
import numpy as np
```

Here np enhances code readability and conciseness** Generatiing the random data done by random module on giving row and column size as parameters

In [5]:
```
data=np.random.randn(2,3)
data
```

Out[5]:
```
array([[ 0.26667323, -0.72495686, -0.67127153],
       [-0.96793852, -0.15779634, -0.75248883]])
```

In [18]:
```
data.shape
# shape of an array
```

Out[18]:
```
(2, 3)
```

In [6]:
```
#data type
data.dtype
```

Out[6]:
```
dtype('float64')
```

** CREATING THE NDARRAY **1.creating 1d and multidimensional arrays using list data structure

In [7]:
```
#1 dim.. array
data1=['t',4,6,7.8,0,'r']
arr1=np.array(data1)
data1
```

Out[7]:
```
['t', 4, 6, 7.8, 0, 'r']
```

In [8]:
```
#nested sequence. array
data2=[[2,3,4,5,6],['a','b','c','d','e'],[0.1,0.2,0.3,0.4,0.5]]
arr2=np.array(data2)
data2
```

Out[8]:
```
[[2, 3, 4, 5, 6], ['a', 'b', 'c', 'd', 'e'], [0.1, 0.2, 0.3, 0.4, 0.5]]
```

In [9]:
```
# to know shape
arr2.shape
```

Out[9]:
```
(3, 5)
```

In [10]:
```
# multi dimn.. array
arr3d=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
arr3d
```

Out[10]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

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

properties on arrays

In [11]:
```
arr3d[0]
#  taking the particular data that is from the specified row as it as multi dim.. array
```

Out[11]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [12]:
```
arr3d[1]=20
arr3d
#assigning new values for particular index or row
```

Out[12]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[20, 20, 20],
        [20, 20, 20]]])
```

In [69]:
```
arr3d.shape
```

Out[69]:
```
(2, 2, 3)
```

In [13]:
```
arr3d.ndim
#this function gives the no dimentions in array i.e row,colums,axis
```

Out[13]:
```
3
```

* creating using the dictionary data structure

In [14]:
```
data3={'name':['aa','bb','cc','dd'],
        'age':[20,19,18,17]}
data3
```

Out[14]:
```
{'name': ['aa', 'bb', 'cc', 'dd'], 'age': [20, 19, 18, 17]}
```

OTHER FUNCTIONALITIES AND OTHER OPERATIOS

np.zeros function is used to create arrays with required size by passing the tuple

In [16]:
```
np.zeros(10)
```

Out[16]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [73]:
```
np.zeros((4,8))
```

Out[73]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.]])
```

In [77]:
```
np.zeros((2,3,2))
```

Out[77]:
```
array([[[0., 0.],
        [0., 0.],
        [0., 0.]],

       [[0., 0.],
        [0., 0.],
        [0., 0.]]])
```

*arange function * used tho create an array from starting to n-1 size

In [78]:
```
np.arange(10)
```

Out[78]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [17]:
```
a=np.arange(0,10,3)
print(a)
#step function is used
```

```
[0 3 6 9]
```

In [2]:
```
# maintain the range data in list
inf=list(range(10))
print(inf)
```

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

DATA MANUPULATION USING NUMPY

In [26]:
```
array1=np.array([1,2,4,5,6,7,8,0])
array2=np.array([[1,2,3,4],[5,6,7,8,],[10,11,12,13],[14,15,16,17]])
print('taken 1Darray is',array1)
print('taken 2Darray is',array2)
```

```
taken 1Darray is [1 2 4 5 6 7 8 0]
taken 2Darray is [[ 1  2  3  4]
 [ 5  6  7  8]
 [10 11 12 13]
 [14 15 16 17]]
```

ARRAY CREATION

indexing and slicing and accesing the elements

In [34]:
```
a=array1[2]
print('specific element is',a)
b=array1[:6:2]#using the step function of 1d array
print(" sliced elements is",b)
```

```
specific element is 4
 sliced elements is [1 4 6]
```

In [37]:
```
#2D array
c=array2[1,2]
print(' specific element is',c)
d=array2[:1,2:]
print(d)
```

```
 specific element is 7
[[3 4]]
```

In [32]:
```
array2[::1]
```

Out[32]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [10, 11, 12, 13],
       [14, 15, 16, 17]])
```

In [33]:
```
array1[::-1]
# its prints from backward as step function is -1
```

Out[33]:
```
array([0, 8, 7, 6, 5, 4, 2, 1])
```

In [36]:
```
e=array2[:2,1:]
print('sliced 2Darray',e)
```

```
sliced 2Darray [[2 3 4]
 [6 7 8]]
```

In [39]:
```
array1[3:6]=21
print(array1)
```

```
[ 1  2  4 21 21 21  8  0]
```

Boolean indexing

In [38]:
```
bool_index=array2[array2>10]
print("values greater than 10",bool_index)
```

```
values greater than 10 [11 12 13 14 15 16 17]
```

In [50]:
```
names=np.array(['aa','bb','cc','aa'])
values=np.random.randn(4,5)
print("names are",names)
print("values are",values)
```

```
names are ['aa' 'bb' 'cc' 'aa']
values are [[ 0.66690885  1.80223415  0.87189479 -1.01792285  0.10275137]
 [ 0.59633211 -0.63998964  0.58949842 -0.92224771 -0.01592503]
 [-2.50263041 -0.42931618  0.4613448   0.42799647 -0.33173344]
 [-1.19269583 -0.22513079  1.56808484 -2.42399516  1.79807812]]
```

In [51]:
```
names=='aa'
#condition
```

Out[51]:
```
array([ True, False, False,  True])
```

values[names=='aa']

In [53]:
```
values[names=='aa',2:]
```

Out[53]:
```
array([[ 0.87189479, -1.01792285,  0.10275137],
       [ 1.56808484, -2.42399516,  1.79807812]])
```

In [56]:
```
values[values>0.5]=0
values
```

Out[56]:
```
array([[ 0.        ,  0.        ,  0.        , -1.01792285,  0.10275137],
       [ 0.        , -0.63998964,  0.        , -0.92224771, -0.01592503],
       [-2.50263041, -0.42931618,  0.4613448 ,  0.42799647, -0.33173344],
       [-1.19269583, -0.22513079,  0.        , -2.42399516,  0.        ]])
```

MATHEMATICAL OPERTIONS

In [77]:
```
# ON 1D array
arr01=[2,3,4,6,7,8]
arr02=[4,2,3,6,9,7]
print("sum is",arr01+arr02)
sum=array1 +10
print(" result array is",sum)
multiply= arr01 *2
print("after usung multiplication operator",multiply)
```

```
sum is [2, 3, 4, 6, 7, 8, 4, 2, 3, 6, 9, 7]
 result array is [11 12 14 31 31 31 18 10]
after usung multiplication operator [2, 3, 4, 6, 7, 8, 2, 3, 4, 6, 7, 8]
```

In [78]:
```
# 2D element wise
arr__sum=array2+10
print("addition is",arr__sum)
```

```
addition is [[11 12 13 14]
 [15 16 17 18]
 [20 21 22 23]
 [24 25 26 27]]
```

In [79]:
```
arr_mul=array2*4
print("multipliction  is",arr_mul)
```

```
multipliction is [[ 4  8 12 16]
 [20 24 28 32]
 [40 44 48 52]
 [56 60 64 68]]
```

In [80]:
```
arr_sqr=np.sqrt(array2)
print("square root is",arr_sqr)
```

```
square root is [[1.         1.41421356 1.73205081 2.        ]
 [2.23606798 2.44948974 2.64575131 2.82842712]
 [3.16227766 3.31662479 3.46410162 3.60555128]
 [3.74165739 3.87298335 4.         4.12310563]]
```

DATA AGGREGATION

Numpy provides various aggregate functions that allow you to perform computations across the entire array or along a specified axis. which follows below

In [83]:
```
data=array2
print(np.sum(data))   # sum all the elements in the array
print(np.sum(data,axis=1))  # row wise sum
print(np.sum(data,axis=0))  # column wise sum
```

```
144
[10 26 46 62]
[30 34 38 42]
```

MEAN

In [87]:
```
mean=np.mean(data)
print('mean is', mean)
# normal cal
```

```
mean is 9.0
```

In [88]:
```
print(np.mean(data))   # sum all the elements in the array
print(np.sum(data,axis=1))  # row wise mean
print(np.sum(data,axis=0))  # column wise mean
```

```
9.0
[10 26 46 62]
[30 34 38 42]
```

MEDIAN

In [90]:
```
print(np.median(data))   # sum all the elements in the array
print(np.median(data,axis=1))  # row wise median
print(np.median(data,axis=0))  # column wise median
```

```
9.0
[ 2.5  6.5 11.5 15.5]
[ 7.5  8.5  9.5 10.5]
```

STANDARD DEVIATION

In [91]:
```
print(np.std(data))   # sum all the elements in the array
print(np.std(data,axis=1))  # row wise sum
print(np.std(data,axis=0))  # column wise sum
```

```
5.049752469181039
[1.11803399 1.11803399 1.11803399 1.11803399]
[4.9244289 4.9244289 4.9244289 4.9244289]
```

variance

In [93]:
```
print(np.var(data))   # sum all the elements in the array
print(np.var(data,axis=1))  # row wise sum
print(np.var(data,axis=0))  # column wise sum
```

```
25.5
[1.25 1.25 1.25 1.25]
[24.25 24.25 24.25 24.25]
```

GROUPING AND AGGREGATING THE DATA THE DATA here we group the different data(subjects) and calculating at a time

In [114…
```
scores = np.array([
[85, 90, 78],
[92, 88, 81],
[70, 95, 80],
[88, 76, 85],
[90, 85, 88],
[74, 80, 77]
])

subjects = ['CP', 'DAV', 'DBMS']
print("Scores Data:\n", scores)
print("\n")
# Grouping data by subject and performing aggregations
print("Grouping Data and Aggregations:")
for i, s in enumerate(subjects):
    subject_scores = scores[:, i]
    subject_mean = np.mean(subject_scores)
    subject_median = np.median(subject_scores)
    subject_std = np.std(subject_scores)
    subject_sum = np.sum(subject_scores)
    print(f"Subject: {s}")
    print(f" Mean Score: {subject_mean:}")
    print(f" Median Score: {subject_median:}")
    print(f" Standard Deviation of Scores: {subject_std:}")
    print(f" Total Sum of Scores: {subject_sum:}")
```

```
Scores Data:
 [[85 90 78]
 [92 88 81]
 [70 95 80]
 [88 76 85]
 [90 85 88]
 [74 80 77]]


Grouping Data and Aggregations:
Subject: CP
 Mean Score: 83.16666666666667
 Median Score: 86.5
 Standard Deviation of Scores: 8.254628331359863
 Total Sum of Scores: 499
Subject: DAV
 Mean Score: 85.66666666666667
 Median Score: 86.5
 Standard Deviation of Scores: 6.289320754704403
 Total Sum of Scores: 514
Subject: DBMS
 Mean Score: 81.5
 Median Score: 80.5
 Standard Deviation of Scores: 3.8622100754188224
 Total Sum of Scores: 489
```

DATA ANALYSIS USING NUMPY

In [116…
```
correlation:
```

Out[116…
```
np.float64(1.0)
```

In [115…
```
correlation = np.corrcoef(array2)[0][1]
print("Correlation:", correlation)
```

```
Correlation: 1.0
```

In [118…
```
correlation_matrix = np.corrcoef(array2.T)
print("Correlation Matrix is", correlation_matrix)
```

```
Correlation Matrix is [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

In [120…
```
# or for two arrays
x = np.array([1, 2, 3, 4, 5])
y = np.array([5, 4, 3, 2, 1])

# Calculate correlation coefficient matrix
correlation_matrix = np.corrcoef(x, y)

print(correlation_matrix)
```

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

Calculating Percentiles:

Percentiles are used to understand the distribution of data that is it give data on nth percentile

In [123…
```
percentile_30 = np.percentile(array2, 30)
percentile_50 = np.percentile(array2, 50)  # Median
percentile_90 = np.percentile(array2, 90)

print("30th percentile  is:", percentile_25)
print("50th percentile is :", percentile_50)
print("90th percentile is:", percentile_90)
```

```
30th percentile  is: 4.75
50th percentile is : 9.0
90th percentile is: 15.5
```

conclusion

ADVANTAGES OF NUMPY

SPEED: Compared to traditional python Data structures, Numpy arrays are a lot faster especially for larger datasets. This is because, numpy module is built using the C API which is faster than python. MEMORY EFFICIENCY: NumPy arrays consume less memory compared to Python lists due to their fixed-size data types. Advanced Mathematical and Statistical Functions: Built-in Functions: NumPy provides a comprehensive set of mathematical and statistical functions such as np.mean, np.std, np.corrcoef, and np.percentile. These functions are optimized for performance and can handle large-scale data computations efficientlyStatistical Computations: Numpy provides various builtin statistical functions that allow easy analysis and computation.Broadcasting: While performing operations between two arrays of different shapes, the smaller array is distributed across the larger one making them compatable for the operation.Ease of Use: Conciseness: NumPy simplifies code by allowing complex numerical operations to be expressed in a few lines of code. This results in more readable and maintainable code, especially when dealing with large datasets and complex analyses

USES

NumPy was used to perform various data analysis tasks including aggregation, correlation calculation, outlier detection, and percentile computation. Efficiency in Data Handling and Computation: Vectorized Operations: NumPy enables vectorized operations which apply functions to entire arrays at once. This approach avoids the need for explicit loops in Python, leading to faster execution.

REAL WORLD APLLICATIONS

1. *Data Science*: Used for data manipulation, preprocessing, and supporting machine learning algorithms. 2. *Finance*: Supports time series analysis, risk management, and Monte Carlo simulations. 3. *Engineering*: Facilitates simulations like Finite Element Analysis and signal processing. 4. *Computer Vision*: Used in image processing and object detection tasks. 5. *Scientific Research*: Powers numerical analysis, simulations, and mathematical modeling in various sciences. 6. *Game Development*: Assists in physics simulations

and procedural generation. 7. "Astronomy". Analyzes astronomical data and simulates celestial events. 8. "Robotics". Supports control systems, kinematics, and dynamics calculations. 9. "Healthcare". Processes medical imaging and genomic data. 10. "Meteorology". Models weather and climate patterns