**Numpy**

Index -

# SEQUENCE => ARRAY()

# 1D, 2D, 3D array

# ACCESS THE ELEMENTS : INDEX AND SLICING

# ATTRIBUTES : NDIM, SHAPE, SIZE ETC

# FUNCTIONS : SUM() MIN() MAX() : ROWS : 1 COLUMNS : 0

# FUNCTION : RESHAPE()

# range() arange() linspace()

# Random Number Generator

# Trigonometric Functions: sin(), cos(), tan()

# Statistical Function : mean(), median()

# decimal function : ceil(), floor(), around()

# matrix functions : empty(), zeros(), ones(), identity(), matmul(), multiply(),

Import the library

# import libraries

import numpy as nm

Convert the list to the array -

Using (np.array( x )) we convert any datatype into array.

# convert sequence into array

# array()

a = [10, 20, 30, 40, 50]

arr1 = nm.array(a)

print(arr1, type(arr1))

Output - [10, 20, 30, 40, 50] <class ‘nump.ndarray’>

Arithmetic Operations -

Addition (+)

Substraction (-)

Multiplication (\*)

Division (/)

Floor Division (//)

Mod (% remainder)

Exponential (\*\*)

Indexing & Slicing -

To access particular element from an array we use indexing.

# Indexing

a = [10, 20, 30, 40, 50, 0.1, 0.2, 0.3, 0.4, 0.5] # List

# convert list into array

arr = nm.array(a)

arr

# To ascess elements : Indexing : 1. Positive 2. Negative

# Display : 20, 40

print(arr3[1],arr3[-9] )

print(arr3[3],arr3[-7] )

# Slicing : array\_name[start=0 : end=-1 : steps=1]

arr = [10, 20, 30, 40, 50, 0.1, 0.2, 0.3, 0.4, 0.5]

# Display :

# 1. [10, 20, 30, 40, 50]

# 2. [0.3, 0.4, 0.5]

# 3. [10, 20, 30, 40, 50, 0.1, 0.2, 0.3, 0.4]

# 4. [10, , 40, 0.2, 0.5]

print(arr3[ : 5])

print(arr3[-3 :])

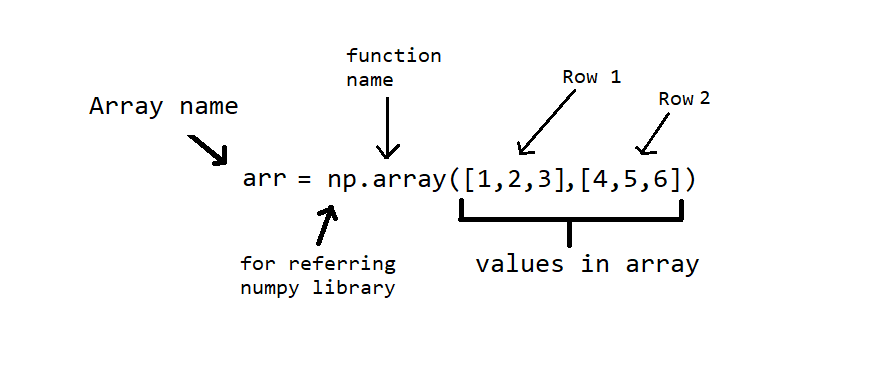
print(arr3[0 :9 ])

print(arr3[0 : : 3])

2-D array -

2-D array is basically a matrix.

2-D array contains rows and columns.



a = [10,20,30]

b = [40,50,60]

c = [70,80,90]

arr = nm.array([a,b,c])

print(arr4, type(arr4))

Output - <class ‘numpy.ndarray’>

# dimensions : dim

print(arr.ndim)

O/P - 2

# size

print(arr.size)

O/P - 12

# shape

print(arr.shape) # O/P : (Rows, Columns ) => (3, 3)

O/P - (3,3)

Accessing Elements in 2-D array -

# Accessing elements :

print(arr4[0])

print(arr4[0,0])

print(arr4[0,1])

O/P -

[10, 20, 30]

10

20

Exercise -

# create a 5, 7 matrix

arr=np.array([range(1,8),range(11,18),range(21,28),range(31,38),range(41,48) ])

arr

O/P -

array([

[1, 2, 3, 4, 5, 6, 7],

[11, 12, 13, 14, 15, 16, 17],

[21, 22, 23, 24, 25, 26, 27],

[31, 32, 33, 34, 35, 36, 37,]

[41,42, 43, 44, 45, 46, 47]

])

# attributes :

print(arr.ndim)

print(arr.size)

print(arr.shape)

O/P -

2

35

(5, 7)

arr = ([

[1, 2, 3, 4, 5, 6, 7],

[11, 12, 13, 14, 15, 16, 17],

[21, 22, 23, 24, 25, 26, 27],

[31, 32, 33, 34, 35, 36, 37,]

[41,42, 43, 44, 45, 46, 47]

])

**# To select columns we did slicing**

# Display :

[13, 14, 15, 16, 17]

print(arr[1, 2:])

# Display Column Values : 1, 11, 21

print(arr[ 0:3 , 0]) # here, (0:3) is used to select the rows and (, 0) is used to select the columns.

O/P - [1, 11, 21]

# Display

# 13, 14, 15,

# 23, 24, 25,

# 33, 34, 35,

print(arr[1:4, 2:5]) # here, (1:4) is used to select the rows and (1:5) is used to select the columns.

'''

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

2 [21, 22, 23, 24, 25, 26, 27],

4 [41, 42, 43, 44, 45, 46, 47]

'''

# here, [0:5:2] = [start, stop, step], : = columns

print(arr5[0 : 5 : 2 , : ])

3-D array -

A three dimensional means we can use nested levels of array for each dimension.

# create a 3D array :

arr = nm.array([[[12, 15, 17], [21,34,12], [23, 33,13]],

[[45,56, 67], [44,76,89], [54,67,78]],

[[100,102,103], [121,133,145], [143,134,122]]])

# attributes :

print(arr.ndim)

print(arr.size)

print(arr.shape)

O/P -

3

27

(3, 3, 3)

# Indexing :

# display : 100 , 76, 134, 33

print(arr6[2,0,0])

print(arr6[1,1,1])

print(arr6[2,2,1])

print(arr6[0,2,1])

Arithmetic Functions -

min()

max()

sum()

# Note : Columns : axis = 0 Rows : axis = 1

arr = nm.array([[11,22,33,44], [-10, -50, 60, 70], [15, 67, -89, 54]])

arr

# Display row wise min values

print(arr.min(axis = 1))

# Display row wise max values

print(arr.max(axis = 1))

# Display row wise sum

print(arr.sum(axis = 1))

# Display column wise min values

print(arr7.min(axis = 0)

# min() max() sum() + rows : '1', columns : '0' + index/slicing

# Find min value in 3rd row.

print(arr7.min(axis = 1)[2])

# Find max value in 2nd column

print(arr7.max(0)[1])

# Find sum of all elements of 1st row.

print(arr7.sum(1)[0])

# reshape()

arr = nm.array([[11,22,33,44], [-10, -50, 60, 70], [15, 67, -89, 54]])

print(arr)

print(" Dimensions : {} \t Size : {} \t Shape : {}".format(arr.ndim, arr.size, arr.shape ))

O/P - Dimensions: 2 Size:12 Shape:(3, 4)

# row = 3 columns = 4 => row = 2 column = 6

arr1 = arr.reshape(2,6)

print(arr1)

print(" Dimensions : {} \t Size : {} \t Shape : {}".format(arr1.ndim, arr1.size, arr1.shape ))

O/P - Dimensions: 2 Size:12 Shape:(2, 6)

Sequence Generator -

arrange()

linespace()

# 1. arange(start , end, steps )

b = nm.arange(1,101)

print(b)

print(type(b))

O/P -

[1, 2, 3, 4, 5, …………………………………… 100]

<class ‘numpy.ndarray’>

# 2. linspace(start , end, interval)

c = nm.linspace(1,10, 5)

print(c)

O/P - [1. 3.75 5.5 7.5 10.]

9/4 = 2.5

1 + 2.5 + 2.5 + 2.5 + 2.5 = 10.0

Trigonometric Functions -

Sin, cos, Tan

# sin, cos, tan

angles\_degree = nm.array([0 , 30, 45, 60, 90, 180])

angles\_radians = (nm.pi / 180) \* angles\_degree

# Trigonometric sin

a\_sin = nm.sin(angles\_radians)

print(a\_sin)

# Trigonometric cos

a\_cos = nm.cos(angles\_radians)

print(a\_cos)

# Trigonometric tan

a\_tan = nm.tan(angles\_radians)

print(a\_tan)

Decimal Functions -

floor() - Removes all the decimals

ceil() - Returns nearest integer

around(2) - Returns 2 decimal numbers after point eg.(10.00)

arr1 = nm.linspace(1,5, 7)

print(arr1)

print("Output of Floor : ", nm.floor(arr1))

print("Output of ceil : ", nm.ceil(arr1))

print("Output of Around : ", nm.around(arr1, 3))

Statistical Functions -

mean() -

median() -

# mean() , median()

arr2 = nm.array([2,4,10,46,54,5,7,643,24,11,76,54,43,4,57,65,3,6,76,543,4,57,65,44,6,5,4,4,6,56,4,6,65,4,57,6,76,5,45,4], order = 'F')

print(" Mean of arr2 : ", nm.floor(nm.mean(arr2)))

print("Median of arr2 : ", nm.median(arr2))

Generate random numbers -

nm.random.randint(1, 10, 5) # start value, stop value, total random numbers required

Matrix functions -

empty() -

zeros() -

ones() -

identity() -

matmul() -

multiply () -

det() -

inv() -

dot() -

linalg() -

# Matrix :

# empty() : Creating empty matrix

# Create a 5X5 empty matrix

a = numpy.matlib.empty((5,5))

print(a)

# zeros() : create a matrix of zeros.

# create a matrix of 5X5.

a = numpy.matlib.zeros((5,5))

print(a)

# ones() : create a matrix of ones.

# create a matrix of 5X3.

b = numpy.matlib.ones((5,3))

print(b)

# identity() : create an Identity matrix.

c = numpy.matlib.identity(6)

print(c)

# Matrix Multiplication :

# matmul() :

a = nm.array([[1,2,3], [4,5,6], [7,8,9]])

b = nm.array([[10,20,30], [6,5,8], [1,2,4]])

print(a)

print(b)

c = nm.matmul(a,b)

print(c)

# multiply() : Element wise multiplication

d = nm.multiply(a,b)

print(d)

# Determinant of Matrix : det()

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

# 1 2

# 3 4 = (1 \* 4) - (2 \* 3) = 4-6 = -2

det = nm.linalg.det(a)

print("Determinant of Matrix : ",det)

# Inverse of a Matrix : inv()

# 1. determinant 2. co-factors 3. Inverse

# A \* [INV] = [1]

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

inv = nm.linalg.inv(a)

print(inv)

# vector :

# 1. dot() : dot product

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

b = nm.array([10,20,30,40,50])

#dot\_product = a1\* b1 + a2+b2 + a3\*b3 + ... = 1\*10 + 2\*20 + 3\*30 + 4\*40 + 5 \* 50 = 10+40 + 90+160+250 = 550

c = nm.dot(a,b)

print(c)

If you have an array of shape (2,4) then reshaping it with (-1, 1), then the array will get reshaped in such a way that the resulting array has only 1 column and this is only possible by having 8 rows, hence, (8,1).