Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects, and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

Keypoints

- Third party package for scientific computing
- Operating with numbers and maths
- Use them like the usual C arrays
- Base for most processing elements like Pandas
- Less memory intensive compared to list

```
import numpy as np
```

```
# One Dimensional Array
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(arr.shape)
print(arr.ndim)
```

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

```
# Zero Dimensional Array
arr1 = np.array(42)
print(arr1)
print(arr1.shape)
print(arr1.ndim)
```

```
# Two Dimensional Array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2)
print(arr2.shape)
print(arr2.ndim)
```

```
# Three Dimensional Array
arr3 = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr3)
print(arr3.shape)
print(arr3.ndim)
```

```
#  Creating n-Dimensional Array
arr4 = np.array([1, 2, 3, 4], ndmin=5)

print(arr4)
print('number of dimensions :', arr4.ndim)
print(arr4.shape)
```

```
# Slicing Arrays
print(arr1, arr[0], arr2[0][1], arr3[0][1][2])
```

```
# Slicing Arrays
print(arr1, arr[0], arr2[0, 1], arr3[0, 1, 2])
```

```
# Slicing Arrays - Negative Indexing
print(arr2)
print(arr2[-2, -1])
```

```
# Slicing a 1-D Array
arr4 = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr4[1:5])
print(arr4[-5:-1])
print(arr4[2:])
print(arr4[:4])
print(arr4[0:7:2])
print(arr4[::2])
```

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

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

```
# Slicing a 2-D Array
arr5 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
arr5
```

```
print(arr5[1, 1:4])
print(arr5[0:2, 2])
print(arr5[0:2, 1:4])
```

Data Types in NumPy

NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.

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 for other type (void)

```
print(arr5.dtype)
```

```
arr6 = np.array(['apple', 'banana', 'cherry'])
print(arr6.dtype)
```

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

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

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

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

```
# What if a Value Can Not Be Converted?
arr = np.array(['a', '2', 3], dtype='i')
```

```
# Converting Data Type on Existing Arrays
arr = np.array([3.47, 2.141, 4.15])
newarr = arr.astype('i')
print(newarr)
print(newarr.dtype)
```

```
# Change data type from float to integer by using int as parameter value:
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype(int)
print(newarr)
print(newarr.dtype)
```

The Difference Between Copy and View

The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

```
# COPY

import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)



# VIEW
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42

print(arr)
print(x)

x[0] = 125
print(arr)


# Reshaping arrays (Reshape From 1-D to 2-D)
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
print(newarr.ndim)
print(newarr.shape)


# Reshaping arrays (Reshape From 1-D to 3-D)
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(2, 3, 2)
print(newarr)
print(newarr.ndim)
print(newarr.shape)


# Flattening the arrays (Converting a multidimensional array into a 1D array)
arr = np.array([[1, 2, 3], [4, 5, 6]])
newarr = arr.reshape(-1)
print(newarr)
print(newarr.ndim)
print(newarr.shape)


# Iterating Arrays - 1D
arr = np.array([1, 2, 3])

for x in arr:
  print(x)


# Iterating Arrays - 2D
arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
  print(x)

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



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

for x in arr:
  print(x)

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

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

Joining NumPy Arrays

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

```
# Join two arrays

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



# Join two 2-D arrays along rows (axis=1)
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
print(arr)


# Join two 2-D arrays along columns (axis=0)
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=0)
print(arr)
```

Splitting NumPy Arrays

Splitting is reverse operation of Joining.

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

```
# Splitting a 1D array into 3 parts
arr = np.array([1, 2, 3, 4, 5, 6, 7])
newarr = np.array_split(arr, 3)
print(newarr)

print(newarr[0])
print(newarr[1])
print(newarr[2])


# Splitting 2-D Arrays
arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
newarr = np.array_split(arr, 3)
print(newarr)
print(newarr[0])


# Sorting Arrays
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))

arr1 = np.array(['banana', 'cherry', 'apple'])
print(np.sort(arr1))

arr2 = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr2))


# NumPy Random
from numpy import random

x = random.randint(100)
print(x)

x1 = random.rand()
print(x1)

x2 = random.randint(100, size=(3, 5))
print(x2)

x3 = random.rand(3, 5)
print(x3)



# Array Arithmetic
a = np.array([[3, 2, 4], [5, 0, 1], [2, 5, 8]])
b = np.array([[1, 0, 5], [2, 3, 7], [6, 4, 5]])

print(a+2)
print(a-1)
print(a*5)
print(a/2)

c = a + b
print(c)

d = a - b
print(d)

e = a * b
print(e)

# Transpose of Array
a = np.array([[3, 2, 4], [5, 0, 1], [2, 5, 8]])
print(a)
ta = np.transpose(a)
print(ta)


# Mean, Median and Standard deviation
a = np.arange(5,50,2)
print(a)
print('Mean :',np.mean(a))
print('Standard deviation :',np.std(a))
print('Median :',np.median(a))

# minimum along a column
print('Min :',np.min(a))
# maximum along a row
print('Max :',np.max(a))



# Min and Max with 2D arrays
a = np.array([[3, 2, 4], [10, 0, 1], [7, 5, 8]])
print(a)
print('Min :',np.min(a, axis=0))
print('Max :',np.max(a, axis=1))



# Random Distribution
from numpy import random
x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(100))
print(x)

y = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(3, 5))
print(y)


# Normal Distribution
x = random.normal(size=(2, 3))
print(x)

y = random.normal(loc=1, scale=2, size=(2, 3))
print(y)
```

```
[[-1.6735065  -0.74708579 -0.76708016]
 [ 2.12730796 -0.19966931  1.82941943]]
[[ 2.12726183  0.81576798  3.25374345]
 [-1.30328083  1.20785085  0.80633547]]
```