Topics to be covered

- Creating Arrays
- List Vs ndarray
- ndarray Attributes
- Indexing and Slicing

Introduction to Numpy

- NumPy is one of the most important foundational packages for numerical computing in Python.
- It provides a high-performance multidimensional array object, and tools for working with these arrays.

Import NumPy

In [1]:

```
import numpy

Myarray = numpy.array([1,2,3,4])

print(Myarray)
```

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

NumPy as np

- NumPy is usually imported under the np alias.

> alias: In Python alias are an alternate name for referring to the same thing.

In [1]:

```
import numpy as np

Myarray = np.array(['black','blue','green'])

print(Myarray)
```

```
['black' 'blue' 'green']
```

Checking NumPy Version

In [3]:

```
np.__version__
```

Out[3]:

```
'1.15.4'
```

Creating Arrays

- The array object in NumPy is called ndarray.
- We can create a NumPy ndarray object by using the array() function.

Dimensions in Arrays

- A dimension in arrays is one level of array depth

0-D Arrays

In [4]:

```
arr = np.array(100)
print(arr)
print(type(arr))
```

```
100
<class 'numpy.ndarray'>
```

Creating Single-dimensional ndarray

In [5]:

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

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

In [6]:

```
element = [12,23,34,45,56]
oneD = np.array(element)
print(oneD)
```

```
[12 23 34 45 56]
```

Creating Multi-dimensional ndarray

In [7]:

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

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

In [8]:

```
elem = [[1,2,3,4],[9,8,7,6]]
twoD = np.array(elem)
print(twoD)
```

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

Check Dimensions of ndarray

In [9]:

```
twoD.ndim
```

Out[9]:

```
2
```

In [10]:

```
oneD.ndim
```

Out[10]:

```
1
```

In [11]:

```
threeD = np.array([[[1,2,3],[4,5,6]],[[1,1,1],[2,2,2]]])
threeD.ndim
```

Out[11]:

```
3
```

Creating ndarray with Zeros

In [12]:

```
x=np.zeros(5,dtype=int)
x
```

Out[12]:

```
array([0, 0, 0, 0, 0])
```

In [13]:

```
x= np.zeros((2,3),dtype=float)
x
```

Out[13]:

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

Creating ndarray with Ones

In [14]:

```
x = np.ones(5,dtype=int)
x
```

Out[14]:

```
array([1, 1, 1, 1, 1])
```

In [15]:

```
x = np.ones((5,5),dtype=int)
x
```

Out[15]:

```
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

Why do we use python numpy if we already have python list?

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

> - Less Memory
> - Fast
> - ConvenientTo understand it

lets see Example

In [16]:

```
# Less Memory

import numpy as np

import sys

l = [1,2,3,4,5,6,7,8,9]
print(sys.getsizeof(1)*len(l))

arr = np.array(l)
print(arr.size*arr.itemsize)
```

```
252
36
```

In [17]:

```
# Fast

import numpy as np
import time

l1 = range(1,100000)
l2 = range(1,100000)
A1 = np.array(l1)
A2 = np.array(l2)
start = time.time()
result1 = [x+y for x,y in zip(l1,l2)]
print((time.time()- start)*1000)

start =time.time()
result2 = A1+A2
print((time.time()- start)*1000)
```

```
7.978200912475586
2.079486846923828
```

NumPy - Array Attributes (Array Description)

> **Each array has features like shape, size and number of dimensions.**

ndarray.shape

> **It returns a tuple consisting of array dimensions**

In [18]:

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

```
(2, 3)
```

numpy.itemsize

> **Itemsize represents the number of bytes in each element of an array.**

In [19]:

```
x = np.array([1,2,3,4,5], dtype = np.int8)
print(x.itemsize)
```

```
1
```

In [20]:

```
x = np.array([1,2,3,4,5], dtype = np.float16)
print(x.itemsize)
```

```
2
```

How to use Python Numpy to generate Random Numbers?

The random module in Numpy package contains many functions for generation of random numbers

1 . numpy.random.rand() −

Generate Random Float from 0 to 1 of a given size

In [14]:

```
# Generate 10 random float
x = np.random.rand(10)
x
```

Out[14]:

```
array([0.67175037, 0.87236186, 0.7238487 , 0.85128526, 0.00794901,
       0.15716684, 0.76464399, 0.8956791 , 0.10577944, 0.25462601])
```

In [15]:

```
# Generate (3X4) vector of random float
x = np.random.rand(3,4)
x
```

Out[15]:

```
array([[0.0863407 , 0.19299886, 0.96202233, 0.46391629],
       [0.65381446, 0.61132897, 0.98939792, 0.08756883],
       [0.63788797, 0.60562214, 0.58570249, 0.24267445]])
```

2. numpy.random.randint() –

Generate a random integer from given range (start to end-1)

In [12]:

```
#Generate 10 random integer in between 5 to 10.
x = np.random.randint(5,10,size=10)
x
```

Out[12]:

```
array([9, 5, 9, 7, 7, 7, 9, 5, 6, 6])
```

In [13]:

```
#Generate (3X2)vector of random integer in between 5 to 10.
x = np.random.randint(5,10,size=(3,2))
x
```

Out[13]:

```
array([[6, 5],
       [9, 6],
       [9, 5]])
```

3. The choice() method

Generate Random Number From Array

The choice() method allows you to generate a random value based on an array of values.

In [26]:

```
# Generate a randome number from a list of element [1,2,3,4,5,6,7,8]
x = np.random.choice([1,2,3,4,5,6,7,8])
x
```

Out[26]:

```
8
```

In [27]:

```
# The choice() method also allows you to return an array of values.
# Add a size parameter to specify the shape of the array.

x = np.random.choice([1,2,3,4,5,6,7,8],size=2)
x
```

Out[27]:

```
array([3, 8])
```

In [28]:

```
x = np.random.choice([1,2,3,4,5,6,7,8],size=(2,3))
x
```

Out[28]:

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

Accessing elements from an array

> **Indexing in Numpy is quite similar to Python's list standard indexing.**

In [22]:

```
# In 1-D we can access the nth element by stating the index in square brackets
x = np.array([10,23,45,67,89])
x[2]
```

Out[22]:

```
45
```

In [26]:

```
# Method-1
# In multi dimensional arrays, you can specify a comma separated tuple of indices to ac
cess the desired elements.
y = np.array([[12,23,34],[2,4,67]])
y[0,2]
```

Out[26]:

```
34
```

In [27]:

```
# Method-2
y = np.array([[12,23,34],[2,4,67]])
y[0][2]
```

Out[27]:

```
34
```

Indexing & Slicing

> **Contents of ndarray object can be accessed and modified by indexing or slicin**

basic slicing

In [28]:

```
x = np.array([10,23,45,67,89])
x[1:4]
```

Out[28]:

```
array([23, 45, 67])
```

In [33]:

```
y = np.array([[12,23,34],[2,4,67],[1,1,1],[2,2,2]])
y[1:]
```

Out[33]:

```
array([[ 2,  4, 67],
       [ 1,  1,  1],
       [ 2,  2,  2]])
```

In [35]:

```
y[-1:]
```

Out[35]:

```
array([[2, 2, 2]])
```

In [37]:

```
y[-3:-1]
```

Out[37]:

```
array([[ 2,  4, 67],
       [ 1,  1,  1]])
```

In [38]:

```
y[0:2]
```

Out[38]:

```
array([[12, 23, 34],
       [ 2,  4, 67]])
```

In [42]:

```
# All item from column 1 onward
y[:,1:]
```

Out[42]:

```
array([[23, 34],
       [ 4, 67],
       [ 1,  1],
       [ 2,  2]])
```

In [43]:

```
y[0:2,1:]
```

Out[43]:

```
array([[23, 34],
       [ 4, 67]])
```

Boolean Array Indexing

> **This type of advanced indexing is used when the resultant object is meant to be the result of Boolean operations, such as comparison operators.**

In [45]:

```
x = np.array([10,23,45,67,89])
x[x%5==0]
```

Out[45]:

```
array([10, 45])
```

In [49]:

```
y = np.array([[12,23,34],[2,4,67],[1,1,1],[2,2,2]])
y[y%2==0]
```

Out[49]:

```
1
```

Fancy Indexing

Fancy indexing is like the simple indexing we've already seen, but we pass arrays of indices in place of single scalars.

In [35]:

```
x = np.random.randint(100,size=(10,6))
x
```

Out[35]:

```
array([[44, 16, 41,  4, 87, 37],
       [28, 76,  2, 66, 96, 56],
       [92, 83, 13, 75, 52, 93],
       [27, 93, 10, 90, 83, 73],
       [ 8, 44, 35, 25, 29, 82],
       [11, 85, 74, 22, 23, 52],
       [79,  8, 26, 91, 88, 55],
       [35, 54, 43, 60, 37, 19],
       [27, 15, 68, 87, 85, 37],
       [98, 83, 66, 34, 80, 81]])
```

In [40]:

```
# print second,forth and sixth column
x[:,[1,3,5]]
```

Out[40]:

```
array([[16,  4, 37],
       [76, 66, 56],
       [83, 75, 93],
       [93, 90, 73],
       [44, 25, 82],
       [85, 22, 52],
       [ 8, 91, 55],
       [54, 60, 19],
       [15, 87, 37],
       [83, 34, 81]])
```

In [42]:

```
#print from row= 2 to 6
x[1:7,]
```

Out[42]:

```
array([[28, 76,  2, 66, 96, 56],
       [92, 83, 13, 75, 52, 93],
       [27, 93, 10, 90, 83, 73],
       [ 8, 44, 35, 25, 29, 82],
       [11, 85, 74, 22, 23, 52],
       [79,  8, 26, 91, 88, 55]])
```

In [44]:

```
# print from row= 2 to 6 and only column 4 and 6
x[1:7,[3,5]]
```

Out[44]:

```
array([[66, 56],
       [75, 93],
       [90, 73],
       [25, 82],
       [22, 52],
       [91, 55]])
```

In [64]:

```
# More about Indexing
print(x)
x[2:5]
```

```
[[44 16 41  4 87 37]
 [28 76  2 66 96 56]
 [92 83 13 75 52 93]
 [27 93 10 90 83 73]
 [ 8 44 35 25 29 82]
 [11 85 74 22 23 52]
 [79  8 26 91 88 55]
 [35 54 43 60 37 19]
 [27 15 68 87 85 37]
 [98 83 66 34 80 81]]
```

Out[64]:

```
array([[92, 83, 13, 75, 52, 93],
       [27, 93, 10, 90, 83, 73],
       [ 8, 44, 35, 25, 29, 82]])
```

In [65]:

```
x[2:5][0] # It gives first row
```

Out[65]:

```
array([92, 83, 13, 75, 52, 93])
```

In [67]:

```
x[2:5][:,3:] # it gives col 3 to end
```

Out[67]:

```
array([[75, 52, 93],
       [90, 83, 73],
       [25, 29, 82]])
```

In [ ]:

```

```