NumPy

What is NumPy?

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software.

Features of NumPy

NumPy has various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy in Python can also be used as an efficient multi-dimensional container of generic data. Arbitrary data types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Install Python NumPy

Numpy can be installed for **Mac** and **Linux** users via the following pip command:

```
pip install numpy
```

Windows does not have any package manager analogous to that in Linux or Mac. Please download the pre-built Windows installer for NumPy from here (according to your system configuration and Python version). And then install the packages manually.

Note: All the examples discussed below will not run on an **online IDE.**

Arrays in NumPy

NumPy&#x2019s main object is the homogeneous multidimensional array.

- It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
- In NumPy, dimensions are called *axes*. The number of axes is *rank*.
- NumPy&#x2019s array class is called **ndarray**. It is also known by the alias **array**.

Example:

In this example, we are creating a two-dimensional array that has the **rank** of 2 as it has 2 **axes**. The first axis(dimension) is of length 2, i.e., the number of

rows, and the second axis(dimension) is of length 3, i.e., the number of columns. The overall shape of the array can be represented as (2, 3)

- Python3

```
import numpy as np

# Creating array object
arr = np.array( [[ 1, 2, 3],
                 [ 4, 2, 5]] )

# Printing type of arr object
print("Array is of type: ", type(arr))

# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)

# Printing shape of array
print("Shape of array: ", arr.shape)

# Printing size (total number of elements) of array
print("Size of array: ", arr.size)

# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```

Output:

```
Array is of type:  <class 'numpy.ndarray'>
No. of dimensions:  2
Shape of array:  (2, 3)
Size of array:  6
Array stores elements of type:  int64
```

NumPy Array Creation

There are various ways of Numpy array creation in Python. They are as follows:

1. You can create an array from a regular Python list or tuple using the array() function. The type of the resulting array is deduced from the type of the elements in the sequences. Let's see this implementation:

- Python3

```
import numpy as np

# Creating array from list with type float
a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')
```

```
print ("Array created using passed list:\n", a)

# Creating array from tuple
b = np.array((1 , 3, 2))
print ("\nArray created using passed tuple:\n", b)
```

Output:

```
Array created using passed list:
 [[1. 2. 4.]
 [5. 8. 7.]]

Array created using passed tuple:
 [1 3 2]
```

2. Often, the element is of an array is originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with **initial placeholder content**. These minimize the necessity of growing arrays, an expensive operation. **For example:** np.zeros, np.ones, np.full, np.empty, etc.
To create sequences of numbers, NumPy provides a function analogous to the range that returns arrays instead of lists.

- Python3

```
# Creating a 3X4 array with all zeros
c = np.zeros((3, 4))
print ("An array initialized with all zeros:\n", c)

# Create a constant value array of complex type
d = np.full((3, 3), 6, dtype = 'complex')
print ("An array initialized with all 6s."
            "Array type is complex:\n", d)

# Create an array with random values
e = np.random.random((2, 2))
print ("A random array:\n", e)
```

Output:

```
An array initialized with all zeros:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
An array initialized with all 6s.Array type is complex:
 [[6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]]
A random array:
 [[0.15471821 0.47506745]
 [0.03637972 0.15772238]]
```

3. **arange:** This function returns evenly spaced values within a given interval. **Step** size is specified.

- Python3

```
# Create a sequence of integers
# from 0 to 30 with steps of 5
f = np.arange(0, 30, 5)
print ("A sequential array with steps of 5:\n", f)
```

Output:

```
A sequential array with steps of 5:
[ 0  5 10 15 20 25]
```

4. **linspace:** It returns evenly spaced values within a given interval.

- Python3

```
# Create a sequence of 10 values in range 0 to 5
g = np.linspace(0, 5, 10)
print ("A sequential array with 10 values between"
                                  "0 and 5:\n", g)
```

Output:

```
A sequential array with 10 values between0 and 5:
[0.         0.55555556 1.11111111 1.66666667 2.22222222 2.77777778
 3.33333333 3.88888889 4.44444444 5.        ]
```

5. **Reshaping array:** We can use **reshape** method to reshape an array. Consider an array with shape (a1, a2, a3, &#x2026, aN). We can reshape and convert it into another array with shape (b1, b2, b3, &#x2026, bM). The only required condition is a1 x a2 x a3 &#x2026 x aN = b1 x b2 x b3 &#x2026 x bM. (i.e. the original size of the array remains unchanged.)

- Python3

```
# Reshaping 3X4 array to 2X2X3 array
arr = np.array([[1, 2, 3, 4],
                [5, 2, 4, 2],
                [1, 2, 0, 1]])

newarr = arr.reshape(2, 2, 3)

print ("Original array:\n", arr)
print("---------------")
print ("Reshaped array:\n", newarr)
```

Output:

```
Original array:
 [[1 2 3 4]
```

```
 [5 2 4 2]
 [1 2 0 1]]
---------------
Reshaped array:
 [[[1 2 3]
  [4 5 2]]
 [[4 2 1]
  [2 0 1]]]
```

6. Flatten array: We can use **flatten** method to get a copy of the array collapsed into **one dimension**. It accepts *order* argument. The default value is &#x2018C&#x2019 (for row-major order). Use &#x2018F&#x2019 for column-major order.

- Python3

```
# Flatten array
arr = np.array([[1, 2, 3], [4, 5, 6]])
flat_arr = arr.flatten()

print ("Original array:\n", arr)
print ("Fattened array:\n", flat_arr)
```

Output:

```
Original array:
 [[1 2 3]
 [4 5 6]]
Fattened array:
 [1 2 3 4 5 6]
```

Note: *The type of array can be explicitly defined while creating the array.*

NumPy Array Indexing

Knowing the basics of NumPy array indexing is important for analyzing and manipulating the array object. NumPy in Python offers many ways to do array indexing.

- **Slicing:** Just like lists in Python, NumPy arrays can be sliced. As arrays can be multidimensional, you need to specify a slice for each dimension of the array.
- **Integer array indexing:** In this method, lists are passed for indexing for each dimension. One-to-one mapping of corresponding elements is done to construct a new arbitrary array.
- **Boolean array indexing:** This method is used when we want to pick elements from the array which satisfy some condition.

- Python3

```
# Python program to demonstrate
```

```
# indexing in numpy
import numpy as np

# An exemplar array
arr = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])

# Slicing array
temp = arr[:2, ::2]
print ("Array with first 2 rows and alternate"
                    "columns(0 and 2):\n", temp)

# Integer array indexing example
temp = arr[[0, 1, 2, 3], [3, 2, 1, 0]]
print ("\nElements at indices (0, 3), (1, 2), (2, 1),"
                                    "(3, 0):\n", temp)

# boolean array indexing example
cond = arr > 0 # cond is a boolean array
temp = arr[cond]
print ("\nElements greater than 0:\n", temp)
```

Output:

```
Array with first 2 rows and alternatecolumns(0 and 2):
 [[-1.  0.]
 [ 4.  6.]]

Elements at indices (0, 3), (1, 2), (2, 1),(3, 0):
 [ 4.  6.  0.  3.]

Elements greater than 0:
 [ 2.   4.   4.   6.   2.6  7.   8.   3.   4.   2. ]
```

NumPy Basic Operations

The Plethora of built-in arithmetic functions is provided in Python NumPy.

Operations on a single NumPy array

We can use overloaded arithmetic operators to do element-wise operations on the array to create a new array. In the case of +=, -=, *= operators, the existing array is modified.

- Python3

```
# Python program to demonstrate
# basic operations on single array
```

```
import numpy as np

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

# add 1 to every element
print ("Adding 1 to every element:", a+1)

# subtract 3 from each element
print ("Subtracting 3 from each element:", a-3)

# multiply each element by 10
print ("Multiplying each element by 10:", a*10)

# square each element
print ("Squaring each element:", a**2)

# modify existing array
a *= 2
print ("Doubled each element of original array:", a)

# transpose of array
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])

print ("\nOriginal array:\n", a)
print ("Transpose of array:\n", a.T)
```

Output:

```
Adding 1 to every element: [2 3 6 4]
Subtracting 3 from each element: [-2 -1  2  0]
Multiplying each element by 10: [10 20 50 30]
Squaring each element: [ 1  4 25  9]
Doubled each element of original array: [ 2  4 10  6]

Original array:
 [[1 2 3]
 [3 4 5]
 [9 6 0]]
Transpose of array:
 [[1 3 9]
 [2 4 6]
 [3 5 0]]
```

NumPy – Unary Operators

Many unary operations are provided as a method of **ndarray** class. This includes sum, min, max, etc. These functions can also be applied row-wise or column-wise by setting an axis parameter.

- Python3

```
# Python program to demonstrate
# unary operators in numpy
import numpy as np

arr = np.array([[1, 5, 6],
                [4, 7, 2],
                [3, 1, 9]])

# maximum element of array
print ("Largest element is:", arr.max())
print ("Row-wise maximum elements:",
                    arr.max(axis = 1))

# minimum element of array
print ("Column-wise minimum elements:",
                        arr.min(axis = 0))

# sum of array elements
print ("Sum of all array elements:",
                            arr.sum())

# cumulative sum along each row
print ("Cumulative sum along each row:\n",
                        arr.cumsum(axis = 1))
```

Output:

```
Largest element is: 9
Row-wise maximum elements: [6 7 9]
Column-wise minimum elements: [1 1 2]
Sum of all array elements: 38
Cumulative sum along each row:
[[ 1  6 12]
 [ 4 11 13]
 [ 3  4 13]]
```

NumPy – Binary Operators

These operations apply to the array elementwise and a new array is created. You can use all basic arithmetic operators like +, -, /, etc. In the case of *+=, -=,* = operators, the existing array is modified.

- Python3

```
# Python program to demonstrate
# binary operators in Numpy
```

```
import numpy as np

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

# add arrays
print ("Array sum:\n", a + b)

# multiply arrays (elementwise multiplication)
print ("Array multiplication:\n", a*b)

# matrix multiplication
print ("Matrix multiplication:\n", a.dot(b))
```

Output:

```
Array sum:
[[5 5]
 [5 5]]
Array multiplication:
[[4 6]
 [6 4]]
Matrix multiplication:
[[ 8  5]
 [20 13]]
```

Introduction to NymPy's ufuncs

NumPy provides familiar mathematical functions such as sin, cos, exp, etc. These functions also operate elementwise on an array, producing an array as output.

Note: All the operations we did above using overloaded operators can be done using ufuncs like np.add, np.subtract, np.multiply, np.divide, np.sum, etc.

- Python3

```
# Python program to demonstrate
# universal functions in numpy
import numpy as np

# create an array of sine values
a = np.array([0, np.pi/2, np.pi])
print ("Sine values of array elements:", np.sin(a))

# exponential values
a = np.array([0, 1, 2, 3])
```

```
print ("Exponent of array elements:", np.exp(a))

# square root of array values
print ("Square root of array elements:", np.sqrt(a))
```

Output:

```
Sine values of array elements: [  0.00000000e+00   1.00000000e+00
1.22464680e-16]
Exponent of array elements: [  1.           2.71828183   7.3890561
20.08553692]
Square root of array elements: [ 0.          1.          1.41421356
1.73205081]
```

NumPy Sorting Arrays

There is a simple **np.sort()** method for sorting Python NumPy arrays. Let's explore it a bit.

- Python3

```
# Python program to demonstrate sorting in numpy
import numpy as np

a = np.array([[1, 4, 2],
                [3, 4, 6],
              [0, -1, 5]])

# sorted array
print ("Array elements in sorted order:\n",
                    np.sort(a, axis = None))

# sort array row-wise
print ("Row-wise sorted array:\n",
                np.sort(a, axis = 1))

# specify sort algorithm
print ("Column wise sort by applying merge-sort:\n",
            np.sort(a, axis = 0, kind = 'mergesort'))

# Example to show sorting of structured array
# set alias names for dtypes
dtypes = [('name', 'S10'), ('grad_year', int), ('cgpa', float)]

# Values to be put in array
values = [('Hrithik', 2009, 8.5), ('Ajay', 2008, 8.7),
          ('Pankaj', 2008, 7.9), ('Aakash', 2009, 9.0)]
```

```
# Creating array
arr = np.array(values, dtype = dtypes)
print ("\nArray sorted by names:\n",
            np.sort(arr, order = 'name'))

print ("Array sorted by graduation year and then cgpa:\n",
                np.sort(arr, order = ['grad_year', 'cgpa']))
```

Output:

```
Array elements in sorted order:
[-1  0  1  2  3  4  4  5  6]
Row-wise sorted array:
[[ 1  2  4]
 [ 3  4  6]
 [-1  0  5]]
Column wise sort by applying merge-sort:
[[ 0 -1  2]
 [ 1  4  5]
 [ 3  4  6]]

Array sorted by names:
[('Aakash', 2009, 9.0) ('Ajay', 2008, 8.7) ('Hrithik', 2009, 8.5)
 ('Pankaj', 2008, 7.9)]
Array sorted by graduation year and then cgpa:
[('Pankaj', 2008, 7.9) ('Ajay', 2008, 8.7) ('Hrithik', 2009, 8.5)
 ('Aakash', 2009, 9.0)]
```

FAQs

1. Why NumPy arrays are better than Python lists?

- NumPy is not only better in performance but it also provides us with several functionalities like broadcasting, indexing, slicing.
- NumPy arrays are memory efficient as it stores the homogenous data in contiguous blocks of memory, where lists can store heterogenous data.
- NumPy supports multidimensional arrays which allows easy representation.

2. What are ndarrays in NumPy?

ndarrays or n-dimensional arrays are capable of storing homogenous elements. They have a fixed size which is defined at the time of creation.

3. What is Broadcasting?

Broadcasting allows us to perform operations between arrays of different shapes. NumPy arrays can perform element-wise operations. Example:

arr1 = np.array([1, 2, 3])
arr2 = np.array([[5], [10], [15]])
result = arr1 + arr2 #[[6, 7, 8], [11, 12, 13], [16, 17, 18]]