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 contains various features including these important ones:

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

Besides its obvious scientific uses, NumPy 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.

Installation:

!pip install numpy

1. Arrays in NumPy:

NumPy’s 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’s array class is called ndarray. It is also known by the alias array.

In [2]:

```
# Python program to demonstrate
# basic array characteristics
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))
```

```
Array is of type:  <class 'numpy.ndarray'>
```

In [3]:

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

```
No. of dimensions:  2
```

In [4]:

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

```
Shape of array:  (2, 3)
```

In [5]:

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

```
Size of array:  6
```

In [6]:

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

```
Array stores elements of type:  int32
```

2. Array creation:

```
There are various ways to create arrays in NumPy.
```

1. For example, 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. Often, the elements of an array are 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.
2. For example: np.zeros, np.ones, np.full, np.empty, etc. To create sequences of numbers, NumPy provides a function analogous to range that returns arrays instead of lists.
3. arange: returns evenly spaced values within a given interval. step size is specified.
4. linspace: returns evenly spaced values within a given interval. num no. of elements are returned.
5. Reshaping array: We can use reshape method to reshape an array. Consider an array with shape (a1, a2, a3, …, aN). We can reshape and convert it into another array with shape (b1, b2, b3, …, bM). The only required condition is: a1 x a2 x a3 … x aN = b1 x b2 x b3 … x bM . (i.e original size of array remains unchanged.)
6. Flatten array: We can use flatten method to get a copy of array collapsed into one dimension. It accepts order argument. Default value is ‘C’ (for row-major order). Use ‘F’ for column major order.

In [8]:

```
# Python program to demonstrate
# array creation techniques
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)
```

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

In [9]:

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

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

In [10]:

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

```
An array initialized with all zeros:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

In [11]:

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

```
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]]
```

In [12]:

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

```
A random array:
 [[0.47955142 0.26068698]
 [0.74020941 0.91944416]]
```

In [13]:

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

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

In [14]:

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

```
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.        ]
```

In [15]:

```
# 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 ("\nOriginal array:\n", arr)
print ("Reshaped array:\n", newarr)
```

```
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]]]
```

In [16]:

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

print ("\nOriginal array:\n", arr)
print ("Fattened array:\n", flarr)
```

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

3. Array Indexing:

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

Slicing:

Just like lists in python, NumPy arrays can be sliced. As arrays can be multidim ensional, 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 map ping 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 array which satisfy some condition.

In [21]:

```
# 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)
```

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

In [22]:

```
# 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)
```

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

In [23]:

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

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

4. Basic operations:

```
        Plethora of built-in arithmetic functions are provided in NumPy.
```

Operations on single array:

```
    We can use overloaded arithmetic operators to do element-wise operation on array
    to create a new array. In case of +=, -=, *= operators, the exsisting array is m
    odified.
```

In [24]:

```
# 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)
```

```
Adding 1 to every element: [2 3 6 4]
```

In [25]:

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

```
Subtracting 3 from each element: [-2 -1  2  0]
```

In [26]:

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

```
Multiplying each element by 10: [10 20 50 30]
```

In [27]:

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

```
Squaring each element: [ 1  4 25  9]
```

In [28]:

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

```
Doubled each element of original array: [ 2  4 10  6]
```

In [29]:

```
# 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)
```

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

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.

In [30]:

```
# 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))
```

```
Largest element is: 9
Row-wise maximum elements: [6 7 9]
```

In [31]:

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

```
Column-wise minimum elements: [1 1 2]
```

In [32]:

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

```
Sum of all array elements: 38
```

In [33]:

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

```
Cumulative sum along each row:
 [[ 1  6 12]
 [ 4 11 13]
 [ 3  4 13]]
```

Binary operators:

```
These operations apply on array elementwise and a new array is created. You can
 use all basic arithmetic operators like +, -, /, , etc. In case of +=, -=, = op
erators, the exsisting array is modified
```

In [34]:

```
# 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)
```

```
Array sum:
 [[5 5]
 [5 5]]
```

In [35]:

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

```
Array multiplication:
 [[4 6]
 [6 4]]
```

In [36]:

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

```
Matrix multiplication:
 [[ 8  5]
 [20 13]]
```

Universal functions (ufunc):

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

In [37]:

```
# 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))
```

```
Sine values of array elements: [0.0000000e+00 1.0000000e+00 1.2246468e-16]
```

In [39]:

```
# exponential values
a = np.array([0, 1, 2, 3])
print ("Exponent of array elements:", np.exp(a))
```

```
Exponent of array elements: [ 1.          2.71828183  7.3890561  20.085536
92]
```

In [40]:

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

```
Square root of array elements: [0.         1.         1.41421356 1.7320508
1]
```

4. Sorting array:

```
There is a simple np.sort method for sorting NumPy arrays. Let’s explore it a bi
t.
```

In [41]:

```
# 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))
```

```
Array elements in sorted order:
 [-1  0  1  2  3  4  4  5  6]
```

In [42]:

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

```
Row-wise sorted array:
 [[ 1  2  4]
 [ 3  4  6]
 [-1  0  5]]
```

In [43]:

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

```
Column wise sort by applying merge-sort:
 [[ 0 -1  2]
 [ 1  4  5]
 [ 3  4  6]]
```

In [47]:

```
# 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 = [('Harry', 2009, 8.5), ('Anthony', 2008, 8.7),
                ('Paul', 2008, 7.9), ('Amy', 2009, 9.0)]


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

```
Array sorted by names:
 [(b'Amy', 2009, 9. ) (b'Anthony', 2008, 8.7) (b'Harry', 2009, 8.5)
 (b'Paul', 2008, 7.9)]
```

In [48]:

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

```
Array sorted by grauation year and then cgpa:
 [(b'Paul', 2008, 7.9) (b'Anthony', 2008, 8.7) (b'Harry', 2009, 8.5)
 (b'Amy', 2009, 9. )]
```