NumPy Library

By

Rajesh Verma

Rajesh Verma, MRA DAV Public School,
Solan

What is NumPy?

- NumPy is the fundamental package for scientific computing with Python.
- It is also known as Numerical Python.
- It is part of SciPy (pronounced "Sigh Pie") stack.
- It is a powerful N-dimensional array object.
- Internally uses C/C++ and Fortran code.
- NumPy's main object is the homogeneous multidimensional array.
- It is a table of elements (usually numbers), all of the same type, indexed by positive integers.
- In NumPy, dimensions are called *axes.*
- Provides both speed and higher productivity.
- Compared with list in Python, an Array in NumPy occupies less space, performs better and provides more functionality.
- Provides linear algebra, and random number capabilities.

Array

- An array is a data structure that stores values of same data types.
- Lists can contain values corresponding to different data types.
- Arrays in python can only contain values corresponding to same data types.

Numpy Array

- A numpy array is a grid of values all of the same type and is indexed by a tuple of non negative integers.
- The numbers of dimension is the rank of arrays.
- The shape of an array is a tuple of integers giving the size of the arrays along each dimensions.
- The dtype , which tells about the type of data stored in the ndarray.

Why use Numpy?

- Numpy provides efficient storage.
- It also provides better ways of handling data for processing.
- It is fast.
- It is easy to learn.
- Numpy uses relatively less memory to store data

How to get NumPy?

- The best way to get it is to install **Anaconda** distribution
- Install using pip : **pip install numpy**

Jupyter Notebook

•The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

•In addition to running your code, Notebook stores code and output, together with markdown notes, in an editable document called a notebook.

•When you save it, this is sent from your browser to the notebook server, which saves it on disk as a JSON file with **.ipynb extension.**

How to install Jupyter Notebook?

Install using pip: pip install jupyter

Arrays creation Methods

There are 6 general mechanisms for creating arrays:

1. Conversion from other Python structures (i.e. lists and tuples)
2. Intrinsic NumPy array creation functions (e.g. arange, ones, zeros, etc.)
3. Replicating, joining, or mutating existing arrays
4. Reading arrays from disk, either from standard or custom formats
5. Creating arrays from raw bytes through the use of strings or buffers
6. Use of special library functions (e.g., random)

1) Converting Python sequences to NumPy Arrays

NumPy arrays can be defined using Python sequences such as lists and tuples. Lists and tuples are defined using [...] and (...), respectively. Lists and tuples can define ndarray creation:

1. a list of numbers will create a 1D array,
2. a list of lists will create a 2D array,
3. further nested lists will create higher-dimensional arrays. In general, any array object is called an **ndarray** in NumPy.

```
>>> a1D = np.array([1, 2, 3, 4])
>>> a2D = np.array([(1, 2), (3, 4)])
>>> a3D = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
```

Class ndarray

- Class **ndarray** represents an array.
- It is also known by the alias **array**.

Attribute	Description
ndim	The number of axes (dimensions) of the array.
shape	The dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension.
size	The total number of elements of the array. This is equal to the product of the elements of shape.
dtype	An object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally, NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.
data	The buffer containing the actual elements of the array.

Find the dimension of Array

ndim = This operation or statement allows us to find or calculate the dimension or no of axes of an array

Syntax :

arrayname.ndim

For Example

```
import numpy as np
A = np.array([(1,2,3,4),(5,6,7,8)])
print(A.ndim)
```

Output : 2

```
A=np.array([1,2,3,4])
print(A.ndim)
```

Output : 1

Find the byte size of each of the element

itemsize = This operation or statement allow user to find the size of the element.

Syntax :

arrayname.itemsize

For Example

```
import numpy as np
A = np.array([(1,2,3,4),(5,6,7,8)])
print(A.itemsize)
```

Output : 4

```
A1=np.array([1,2,3,4],np.int16)
print(A1.ndim)
```

Output : 2

Find the data type of the elements

dtype = This operation or statement allow user to find the data type or dtype of the elements.

Syntax :

arrayname.dtype

For Example

```
import numpy as np
A = np.array([(1,2,3,4),(5,6,7,8)])
print(A.dtype)
```

Output : int32

```
A1=np.array([1,2,3,4],np.int16)
print(A1.dtype)
```

Output : int16

Find the size of the Arrays

size = This operation or statement allow user to find the size of the arrays. It shows total no of elements present in the arrays.

Syntax :

arrayname.size

For Example

```
import numpy as np
A = np.array([(1,2,3,4),(5,6,7,8)])
print(A.size)
```

Output : 8

```
A1=np.array([1,2,3,4],np.int16)
print(A1.size)
```

Output : 4

Find the shape of the Arrays

shape = This operation or statement shows total no of rows and columns in a array.

Syntax :

arrayname.shape

For Example

```
import numpy as np
A = np.array([(1,2,3,4),(5,6,7,8)])
print(A.dtype)
```

Output : int32

```
A1=np.array([1,2,3,4],np.int16)
print(A1.dtype)
```

Output : int16

Methods to create an array

- **zeros(shape)**: Produce an arrays of all zeros with the given shape and dtype
- **ones(shape)** : Produce an arrays of all 1s with given shape and dtype
- **empty(shape)**: Create a new arrays by allocating new memory, but do not populate any values like ones and zeros.
- **full(shape, fill_value)**: Produce an array of the given shape and dtype with all the values set to the indicated fill value

```
>>>np.zeros(5)
array([0., 0., 0., 0.,0.])
 >>>np.ones(5)
 array([1, 1, 1, 1,1])
 >>>np.full(5,2)
 array([2, 2, 2, 2, 2])
 >>> np.full((3,3),5)
             array([[5,   5,   5],
                    [5,   5,   5],
                    [5,   5,   5]])
```

Methods to create an array

- **arange(size)**: It is one of the array creation routines based on numerical ranges. It creates an instance of ndarray with *evenly spaced values* and returns the reference to it.

Syntax :

numpy.arange([start,]stop, [step,], dtype=None)

The first three parameters determine the range of the values, while the fourth specifies the type of the elements:

1. **start** is the number (integer or decimal) that defines the first value in the array.
2. **stop** is the number that defines the end of the array and isn't included in the array.
3. **step** is the number that defines the spacing (difference) between each two consecutive values in the array and defaults to 1.
4. **dtype** is the type of the elements of the output array and defaults to None.

arange ()Methods to create an array

```
# For Example
import numpy as np
Ar= np.arange(10)
Print(Ar)
# Output is : [0 1 2 3 4 5 6 7 8 9]
ar2 = np.arange(1,10,2)
print(ar2)
```

How to create an Array with random numbers

- numpy.random.rand(d0,d1,…,dn) →Random numbers from 0 to 1
- numpy.random.randint(low,high,size)→Random numbers from low to high

```
>>> numpp.random.rand(5)
array([0.37337661, 0.3623917 , 0.69519743, 0.82115195, 0.56427905])
>>> np.random.randint(1,20,10)
array([19, 14, 19, 11, 15])
```

linspace()

```
numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
```

- Returns an array with evenly spaced numbers over a specified interval.
- The num parameter indicates number of values to generate.
- The endpoint specifies whether to include stop in samples.
- The retstep returns step value calculated by linspace().
- The dtype is datatype of values generated.

```
>>> np.linspace(1,10,5, dtype=np.int32)      # 5 numbers of int32 from 1 to 10
array([ 1,  3,  5,  7, 10])
>>> np.linspace(1,5,10)
array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])
```

Indexing

- Indexing and slicing is similar to Python sequences.
- Multidimensional arrays can have one index per axis. These indices are given in a tuple separated by commas.
- You can access an array element by referring to index number
- The indexes in numpy array start with 0 meaning that the first element has index 0 and second has index 1.
- Use negative indexing to access an array from the end.

```
>>> arr = np.array([1,2,3,4])       # Accessing 1-d Array
>>> print(arr[0])       # gets first elements from the array
>>>arr2 = np.array([[1,2,3,4,5],[6,7,8,9,10]]) #accessing 2-d array
>>>print(arr2[0,2])    # gets the 3rd element on 1st dimension
>>> arr3 = np.array([[[1,2,3],[4,5,6]],[[7,8,9],10,11,12]]])
>>>print(arr3=[0,1,2])
```

Indexing by Array

- An array of integers can be used as index to get elements from indexes provided in the given array.

```
a = np.array([10,20,30,40,50])
ia = [0,3,4]
print (a [ia])
```

Slicing

Slicing in python means taking elements from one given index to another given index.

- We pass slice instead of index like this: [*start*:*end*].
- We can also define the step, like this: [*start*:*end*:*step*].
- If we don't pass start its considered 0
- If we don't pass end its considered length of array in that dimension
- If we don't pass step its considered 1

```
>>> a = np.arange(10)
>>> a[0:4]
array([0, 1, 2, 3])
>>> a[:6:2]
array([0, 2, 4])
>>> a[-1]          # last value
9
>>>a[::-1]         # Reverse Array
array([9,8,.. ,0])
>>>a[5:]          # From 5th Element
array([5, 6, 7, 8, 9])
```

```
>>> a = np.arange(9).reshape(3,3)
>>> a
array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
>>> a[:,1]     # Only 1st col in all rows
array([1, 4, 7])
>>> a[:,1:]    # 1st col onwards in all rows
array([[1, 2],[4, 5],[7, 8]])
>>> a[1]       # Only 1st row
array([3, 4, 5])
>>> a[2,...]   # All cols in 2nd row
array([6, 7, 8])
```

Array Operations

- Arithmetic operations are applied element-wise.
- A new array with result is created.
- Some operations can be performed inline.
- When operating with arrays of different types, the type of the resulting array corresponds to the more general one (a behavior known as upcasting).

```
>>> a = np.array([1,2,3,4,5])
>>> a ** 2                              # Raise each element in array by 2
array([ 1,  4,  9, 16, 25], dtype=int32)
>>> a + 10
array([11, 12, 13, 14, 15])
>>> b = np.array((10,20,30,40,50))     # create from tuple
>>> a * b
array([ 10,  40,  90, 160, 250])
>>> a += 5                              # Array a is updated (inline operation)
>>> a
array([ 6,  7,  8,  9, 10])
```

Universal Functions

- NumPy provides familiar mathematical functions such as sin, cos, and exp.
- Within NumPy, these functions operate elementwise on an array, producing another array.

add(x1, x2)	Add arguments element-wise.
subtract(x1, x2)	Subtract arguments, element-wise.
multiply(x1, x2	Multiply arguments element-wise.
divide(x1, x2)	Return a true division of the inputs, element-wise.
floor_divide(x1, x2)	Return the largest integer smaller or equal to the division of the inputs.
negative(x)	Numerical negative, element-wise.
positive(x)	Numerical positive, element-wise.
power(x1, x2)	First array elements raised to powers from second array, element-wise.
remainder(x1, x2)	Return element-wise remainder of division.
mod(x1, x2)	Return element-wise remainder of division.
fmod(x1, x2)	Return the element-wise remainder of division.
divmod(x1, x2)	Return element-wise quotient and remainder simultaneously.
absolute(x)	Calculate the absolute value element-wise.

Universal Functions

fabs(x)	Compute the absolute values element-wise.
rint(x)	Round elements of the array to the nearest integer.
sign(x)	Return an element-wise indication of the sign of a number.
exp(x)	Calculate the exponential of all elements in the input array.
log(x)	Natural logarithm, element-wise.
log2(x)	Base-2 logarithm of *x*.
log10(x)	Return the base 10 logarithm of the input array, element-wise.
sqrt(x)	Return the non-negative square-root of an array, element-wise.
square(x)	Return the element-wise square of the input.
cbrt(x)	Return the cube-root of an array, element-wise.
reciprocal(x)	Return the reciprocal of the argument, element-wise.

Universal Functions - Examples

```
>>> import numpy as np
>>> a = np.array([1,2,3,4,5])
>>> b = np.arange(1,10,2)
>>> b
array([1, 3, 5, 7, 9])

>>> np.add(a,b)
array([ 2,  5,  8, 11, 14])

c = np.array([-1,-2,3,4,-5])
>>> np.absolute(c)
array([1, 2, 3, 4, 5])

>>> np.square(a)
array([ 1,  4,  9, 16, 25], dtype=int32)

>>> np.sign(c)
array([-1, -1,  1,  1, -1])
```

Methods of ndarray

- Methods of **ndarray** are used to perform operations on array as a whole by taking all elements.
- By default, these operations apply to the array as though it were a list of numbers, regardless of its shape.
- However, by specifying the **axis** parameter you can apply an operation along the specified axis of an array.

Methods of ndarray

max()	Return the maximum along a given axis.
mean()	Return the average of the array elements along given axis.
min()	Return the minimum along a given axis.
reshape(shape)	Return an array containing the same data with a new shape.
round()	Return each element rounded to the given number of decimals.
sort()	Sort an array, in-place.
std()	Return the standard deviation of the array elements along given axis.
sum()	Return the sum of the array elements over the given axis.
transpose(*axes)	Return a view of the array with axes transposed.
argmax()	Return indices of the maximum values along the given axis.
argmin([axis, out])	Return indices of the minimum values along the given axis.

Methods of ndarray

```
>>> a = np.arange(1,11).reshape(2,5)
>>> a
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
>>> a.sum()
55
>>> a.sum(axis=0)                          # Column
array([ 7,  9, 11, 13, 15])
>>> a.sum(axis=1)                          # Row
array([15, 40])
>>>
```

Reshaping Array

- Arrays can be reshaped and/or resized.
- Some return new array and some reshape the array itself.

reshape()	Give a new shape to an array without changing its data. Use **-1** to ignore a dimension.
ravel()	Return a contiguous flattened array.(to print array in single column)
resize()	Change shape and size of array in-place.
transpose()	Return a view of the array with axes transposed.
T	Attribute T also returns transposed array.

```
a = np.arange(1,10).reshape(3,3)
>>> a
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> a.T              # a.transpose()
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

Stacking Arrays

- Method vstack() and hstack() of NumPy stack array vertically and horizontally respectively.
- Both methods return a new array.

```
>>> a
array([[0, 1],
       [2, 3]])
>>> b
array([[10, 11],
       [12, 13]])
>>> np.vstack((a,b))
array([[ 0,  1],
       [ 2,  3],
       [10, 11],
       [12, 13]])
>>> np.hstack((a,b))
array([[ 0,  1, 10, 11],
       [ 2,  3, 12, 13]])
>>>
```

Concatenate Arrays

- Method concatenate() is used to concatenate given arrays.

```
>>> a = np.arange(5)
>>> a
array([0, 1, 2, 3, 4])
>>> b = np.arange(5,10)
>>> b
array([5, 6, 7, 8, 9])
>>> np.concatenate( (a,b))
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>>
```

Splitting Arrays

- Method hsplit() can split an array along its horizontal axis, either by specifying the number of equally shaped arrays to return, or by specifying the columns after which the division should occur.
- Method vsplit() splits vertically.
- Method split() allows you to specify indices where array is to be split.

```
>>> a = np.arange(1,9).reshape(2,-1)
>>> a
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>> p1,p2 = np.hsplit(a,2)
>>> p1
array([[1, 2],
       [5, 6]])
>>> p2
array([[3, 4],
       [7, 8]])
```

```
>>> np.hsplit(a,(1,3))
[array([[1],  [5]]),
 array([[2, 3], [6, 7]]),
 array([[4], [8]])]
>>> np.vsplit(a,2)
[array([[1, 2, 3, 4]]),
 array([[5, 6, 7, 8]])]
>>> np.split(a,[3,6])
[array([0, 1, 2]), array([3, 4, 5]),
array([6, 7, 8, 9])]
```

Matrix Product (Dot product)

```
>>> a1 = np.array([[1,2,3],[2,2,2]])
>>> a2 = np.array([[1,2],[1,2],[1,1]])
>>> a1 @ a2  array([[ 6,
                   9],
           [ 6, 10]])
>>> a1.dot(a2)  array([[ 6,
                   9],
           [ 6, 10]])
```

How to load data from a text file?

loadtxt(): This function is used to load data from a text file. Each rows in the text file must have the same number of values.

Syntax :

Numpy.loadtxt(filename, dtye = <class 'float'> deltimiter = " ")

Example :

```
import numpy as np
ax = np.loadtxt("test.txt", delimiter=",")
```

How to load data from a text file?

genfromtxt(): This function is used to load data from a text file, with missing values handled as specified.

Syntax :

numpy.genfromtxt(fname, dtype= <class 'float'> ,delimiter = None, skip_header = number, skip_footer = number)

Where

fname = File, filename, list or generator to read.

dtype = Data types of resulting arrays.

delimiter = the string used to separate value.

skip_header = the numbers of lines to kip at the beginning of the file.

skip_footer = The number of lines to skip at the end of the file.

Example :

```
import numpy as np
ax = np.genfromtxt("test.txt", skip_header = 1,  delimiter=",", dtype = int)
```

How to save data to a text file?

savetxt(): This function is used to save an array to a text file.

Syntax :

numpy.savetxt(fname,arrayname, delimiter=‘,’)

Where

fname = File, filename, list or generator to save

delimiter = the string used to separate value.

Example :

```
import numpy as np
a = np.arange(0,12).reshape(4,3)
np.savetxt("test3.txt", a, delimiter = ",")
```

Any Question??..

THANK YOU

Rajesh Verma, MRA DAV Public School,
Solan