# Numerical Python

Tushar B. Kute,
http://tusharkute.com

## What is numpy?

- NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.
- Numeric, the ancestor of NumPy, was developed by Jim Hugunin.
- Another package Numarray was also developed, having some additional functionalities.
- In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package.

## Operations using numpy

- Using NumPy, a developer can perform the following operations –
  - Mathematical and logical operations on arrays.
  - Fourier transforms and routines for shape manipulation.
  - Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

## Replacement to MatLab

- NumPy is often used along with packages like SciPy (Scientific Python) and Matplotlib (plotting library).
- This combination is widely used as a replacement for MatLab, a popular platform for technical computing.
- However, Python alternative to MatLab is now seen as a more modern and complete programming language.
- It is open source, which is an added advantage of NumPy.

## Installing numpy

- Standard Python distribution doesn't come bundled with NumPy module. A lightweight alternative is to install NumPy using popular Python package installer, pip.

  ```
  sudo pip3 install numpy
  sudo apt install python3-numpy
  ```

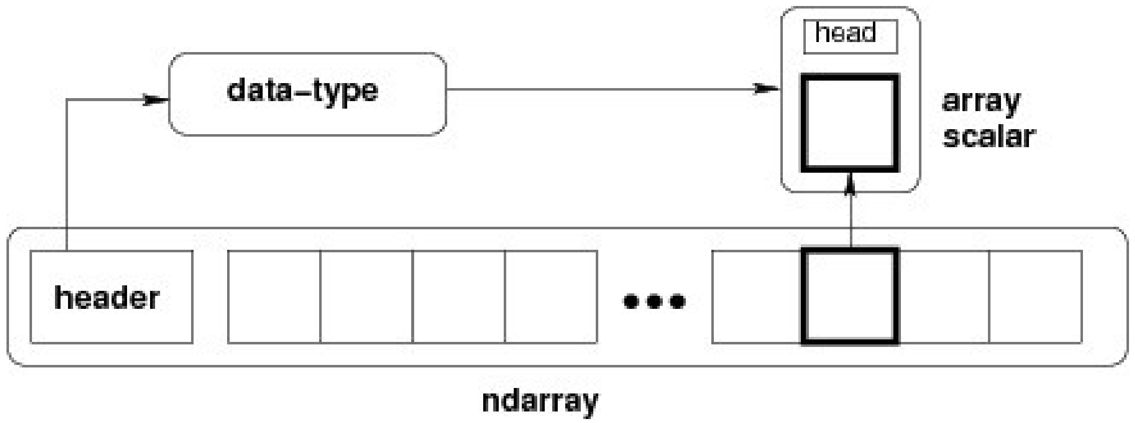
- The best way to enable NumPy is to use an installable binary package specific to your operating system.
  - These binaries contain full SciPy stack (inclusive of NumPy, SciPy, matplotlib, IPython, SymPy and nose packages along with core Python).

## Ndarray object

- The most important object defined in NumPy is an N-dimensional array type called ndarray. It describes the collection of items of the same type.
- Items in the collection can be accessed using a zero-based index.
- Every item in an ndarray takes the same size of block in the memory.
- Each element in ndarray is an object of data-type object (called dtype).

## Ndarray object

- Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types.
- The following diagram shows a relationship between ndarray, data type object (dtype) and array scalar type –



# Ndarray object

- An instance of ndarray class can be constructed by different array creation routines described later in the tutorial.
- The basic ndarray is created using an array function in NumPy as follows –

  **`numpy.array`**
- It creates an ndarray from any object exposing array interface, or from any method that returns an array.

```
numpy.array(object, dtype = None, copy =
True, order = None, subok = False, ndmin = 0)
```

## Creating ndarray

```
import numpy as np
a = np.array([1,2,3])
print (a)
# more than one dimensions
import numpy as np
a = np.array([[1, 2], [3, 4]])
print (a)
# minimum dimensions
import numpy as np
a = np.array([1, 2, 3,4,5], ndmin = 2)
print (a)
```

## Creating ndarray

```
# dtype parameter
import numpy as np
a = np.array([1, 2, 3], dtype = complex)
print (a)
```

## Data types

- bool_
  - Boolean (True or False) stored as a byte
- int_
  - Default integer type (same as C long; normally either int64 or int32)
- intc
  - Identical to C int (normally int32 or int64)
- intp
  - Integer used for indexing (same as C ssize_t; normally either int32 or int64)
- int8
  - Byte (-128 to 127)
- int16
  - Integer (-32768 to 32767)

## Data types

- int32
  - Integer (-2147483648 to 2147483647)
- int64
  - Integer (-9223372036854775808 to 9223372036854775807)
- uint8
  - Unsigned integer (0 to 255)
- uint16
  - Unsigned integer (0 to 65535)
- uint32
  - Unsigned integer (0 to 4294967295)
- uint64
  - Unsigned integer (0 to 18446744073709551615)

## Data types

- float_
  - Shorthand for float64
- float16
  - Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
- float32
  - Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
- float64
  - Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
- complex_
  - Shorthand for complex128
- complex64
  - Complex number, represented by two 32-bit floats (real and imaginary components)
- complex128
  - Complex number, represented by two 64-bit floats (real and imaginary components)

## Data type objects: dtype

- A data type object describes interpretation of fixed block of memory corresponding to an array, depending on the following aspects –
  - Type of data (integer, float or Python object)
  - Size of data
  - Byte order (little-endian or big-endian)
  - In case of structured type, the names of fields, data type of each field and part of the memory block taken by each field.
  - If data type is a subarray, its shape and data type

## Data type objects: dtype

- The byte order is decided by prefixing '<' or '>' to data type. '<' means that encoding is little-endian (least significant is stored in smallest address). '>' means that encoding is big-endian (most significant byte is stored in smallest address).
- A dtype object is constructed using the following syntax –

  `numpy.dtype(object, align, copy)`
- The parameters are –
  - Object – To be converted to data type object
  - Align – If true, adds padding to the field to make it similar to C-struct
  - Copy – Makes a new copy of dtype object. If false, the result is reference to builtin data type object

## Data types: example

```
# using array-scalar type
import numpy as np
dt = np.dtype(np.int32)
print (dt)

#int8, int16, int32, int64 can be replaced by
equivalent string 'i1', 'i2','i4', etc.
dt = np.dtype('i4')
print (dt)

# using endian notation
dt = np.dtype('>i4')
print (dt)
```

## Data types: example

```
# first create structured data type
import numpy as np
dt = np.dtype([('age',np.int8)])
print (dt)

# now apply it to ndarray object
dt = np.dtype([('age',np.int8)])
a = np.array([(10,),(20,),(30,)], dtype = dt)
print (dt)

# file name can be used to access content of age column
dt = np.dtype([('age',np.int8)])
a = np.array([(10,),(20,),(30,)], dtype = dt)
print (a['age'])
```

## Data types: example

```
import numpy as np
student = np.dtype([('name','S20'), ('age',
'i1'), ('marks', 'f4')])
print (student)
```

```
import numpy as np
student = np.dtype([('name','S20'),
('age','i1'), ('marks', 'f4')])
a = np.array([('abc', 21, 50),('xyz', 18,
75)], dtype = student)
print (a)
```

## Unique character codes

- 'b' – boolean
- 'i' – (signed) integer
- 'u' – unsigned integer
- 'f' – floating-point
- 'c' – complex-floating point
- 'm' – timedelta
- 'M' – datetime
- 'O' – (Python) objects
- 'S', 'a' – (byte-)string
- 'U' – Unicode
- 'V' – raw data (void)

## ndarray.shape

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

# this resizes the ndarray
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
a.shape = (3,2)
print (a)

import numpy as np
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print (b)
```

## ndarray.ndim

```
# an array of evenly spaced numbers
import numpy as np
a = np.arange(24)
print (a)

# this is one dimensional array
import numpy as np
a = np.arange(24)
print(a.ndim)

# now reshape it
b = a.reshape(2,4,3)
print (b)
# b is having three dimensions
```

## ndarray.itemsize

```
# dtype of array is int8 (1 byte)
import numpy as np
x = np.array([1,2,3,4,5], dtype = np.int8)
print (x.itemsize)

# dtype of array is now float32 (4 bytes)
import numpy as np
x = np.array([1,2,3,4,5], dtype = np.float32)
print (x.itemsize)
```

## Array creation routines

- A new ndarray object can be constructed by any of the following array creation routines or using a low-level ndarray constructor.
- numpy.empty
  - It creates an uninitialized array of specified shape and dtype. It uses the following constructor –

    numpy.empty(shape, dtype = float, order = 'C')
  - The constructor takes the following parameters.
    - Shape
      - Shape of an empty array in int or tuple of int
    - Dtype
      - Desired output data type. Optional
    - Order
      - 'C' for C-style row-major array, 'F' for FORTRAN style column-major array

## Example:

- The following code shows an example of an empty array.

```
import numpy as np
x = np.empty([3,2], dtype = int)
print (x)
```

## Numpy zeros

```
# array of five zeros. Default dtype is float
import numpy as np
x = np.zeros(5)
print (x)

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

# custom type
import numpy as np
x = np.zeros((2,2), dtype = [('x', 'i4'), ('y', 'i4')])
print (x)
```

## Numpy ones

```
# array of five ones. Default dtype is float
import numpy as np
x = np.ones(5)
print (x)

import numpy as np
x = np.ones([2,2], dtype = int)
print (x)
```

## Example:

```
import numpy as np
x = np.arange(5)
print (x)

# dtype set
x = np.arange(5, dtype = float)
print (x)

# start and stop parameters set
import numpy as np
x = np.arange(10,20,2)
print (x)
```

## linspace

numpy.linspace

- This function is similar to arange() function. In this function, instead of step size, the number of evenly spaced values between the interval is specified. The usage of this function is as follows –

**`numpy.linspace(start, stop, num, endpoint, retstep, dtype)`**

  - Start: The starting value of the sequence
  - Stop: The end value of the sequence, included in the sequence if endpoint set to true
  - Num: The number of evenly spaced samples to be generated. Default is 50
  - Endpoint: True by default, hence the stop value is included in the sequence. If false, it is not included
  - Retstep: If true, returns samples and step between the consecutive numbers
  - Dtype: Data type of output ndarray

## Example:

```
import numpy as np
x = np.linspace(10,20,5)
print (x)

# endpoint set to false
x = np.linspace(10,20, 5, endpoint = False)
print (x)

# find retstep value
x = np.linspace(1,2,5, retstep = True)
print (x)
# retstep here is 0.25
```

## Indexing and slicing

- Contents of ndarray object can be accessed and modified by indexing or slicing, just like Python's in-built container objects.
- Basic slicing is an extension of Python's basic concept of slicing to n dimensions.
- A Python slice object is constructed by giving start, stop, and step parameters to the built-in slice function.
- This slice object is passed to the array to extract a part of array.

## Example:

```
import numpy as np
a = np.arange(10)
s = slice(2,7,2)
print (a[s])

a = np.arange(10)
b = a[2:7:2]
print (b)

a = np.arange(10)
b = a[5]
print (b)
```

## Example:

```
# slice items starting from index
import numpy as np
a = np.arange(10)
print (a[2:])

# slice items between indexes
a = np.arange(10)
print (a[2:5])

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

# slice items starting from index
print 'Now we will slice the array from the index a[1:]'
print (a[1:])
```

## Example:

```
# array to begin with
a = np.array([[1,2,3],[3,4,5],[4,5,6]])

print 'Our array is:'
print (a)

# this returns array of items in the second column
print 'The items in the second column are:'
print (a[...,1] )

# Now we will slice all items from the second row
print 'The items in the second row are:'
print (a[1,...])

# Now we will slice all items from column 1 onwards
print 'The items column 1 onwards are:'
print (a[...,1:])
```

## Integer Indexing

- This mechanism helps in selecting any arbitrary item in an array based on its N-dimensional index.
- Each integer array represents the number of indexes into that dimension.
- When the index consists of as many integer arrays as the dimensions of the target ndarray, it becomes straightforward.

## Example:

```
import numpy as np
x = np.array([[1,2],[3,4],[5,6]])
y = x[[0,1,2],[0,1,0]]
print (y)
```

- One element of specified column from each row of ndarray object is selected. Hence, the row index contains all row numbers, and the column index specifies the element to be selected.
- The selection includes elements at (0,0), (1,1) and (2,0) from the first array.

## Example:

```
import numpy as np
x = np.array([[0,1,2],[3,4,5],[6,7,8],
                   [9,10,11]])
rows = np.array([[0,0],[3,3]])
cols = np.array([[0,2],[0,2]])
y = x[rows,cols]
print('The corner elements are:')
print(y)
```

- The elements placed at corners of a 4X3 array are selected. The row indices of selection are [0, 0] and [3,3] whereas the column indices are [0,2] and [0,2].

# Advanced Slicing: Example

```
import numpy as np
x = np.array([[0,1,2],[3,4,5],[6,7,8],[9,10,11]])
print('Our array is:\n',x)

# With slicing
z = x[1:4,1:3]
print('After slicing:\n',z)

# Using advanced index for column
y = x[1:4,[1,2]]
print('Slicing advanced index:\n',y)
```

## Broadcasting

- The term broadcasting refers to the ability of NumPy to treat arrays of different shapes during arithmetic operations.
- Arithmetic operations on arrays are usually done on corresponding elements.
- If two arrays are of exactly the same shape, then these operations are smoothly performed.

## Lets take an example

```
import numpy as np

a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
print(c)
```

## Broadcasting

- If the dimensions of two arrays are dissimilar, element-to-element operations are not possible. However, operations on arrays of non-similar shapes is still possible in NumPy, because of the broadcasting capability. The smaller array is broadcast to the size of the larger array so that they have compatible shapes.
- Broadcasting is possible if the following rules are satisfied –
  - Array with smaller ndim than the other is prepended with '1' in its shape.
  - Size in each dimension of the output shape is maximum of the input sizes in that dimension.
  - An input can be used in calculation, if its size in a particular dimension matches the output size or its value is exactly 1.
  - If an input has a dimension size of 1, the first data entry in that dimension is used for all calculations along that dimension.

## Broadcasting

- A set of arrays is said to be broadcastable if the above rules produce a valid result and one of the following is true –
  - Arrays have exactly the same shape.
  - Arrays have the same number of dimensions and the length of each dimension is either a common length or 1.
  - Array having too few dimensions can have its shape prepended with a dimension of length 1, so that the above stated property is true.

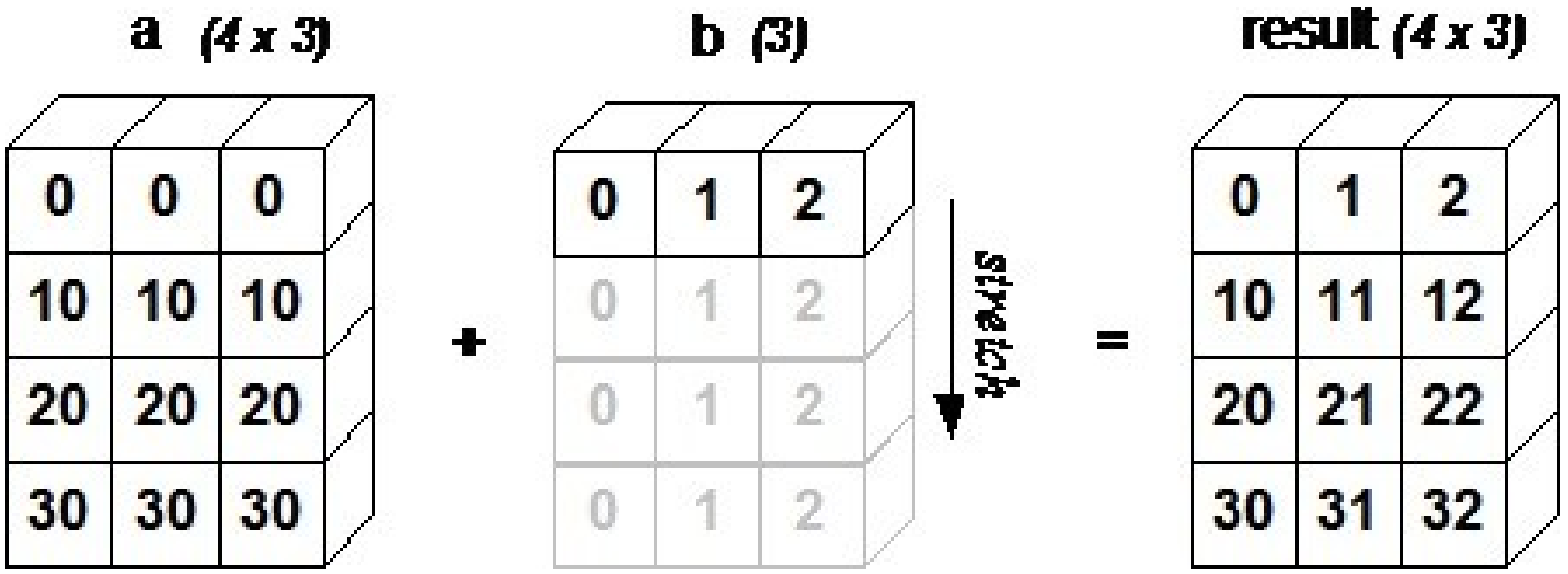
## Example: addition

```
import numpy as np
a = np.array([[0.0,0.0,0.0],[10.0,10.0,10.0],
          [20.0,20.0,20.0],[30.0,30.0,30.0]])
b = np.array([1.0,2.0,3.0])

print('First array:\n',a)
print('Second array:\b',b)

print('First Array + Second Array')
print(a + b)
```

## How it does it?



## Iterating over the array

- NumPy package contains an iterator object numpy.nditer.
- It is an efficient multidimensional iterator object using which it is possible to iterate over an array.
- Each element of an array is visited using Python’s standard Iterator interface.

## Example:

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)

print('Original array is:',a)

print('Modified array is:')
for x in np.nditer(a):
   print(x)
```

# Thank you

@mitu_skillologies @mITuSkillologies @mitu_group @mitu-skillologies @MITUSkillologies

kaggle
@mituskillologies

**Web Resources**
https://mitu.co.in
http://tusharkute.com

@mituskillologies

**contact@mitu.co.in**
**tushar@tusharkute.com**