Numpy

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Arrays

1. A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers.
2. The number of dimensions is the rank of the array;
3. the shape of an array is a tuple of integers giving the size of the array along each dimension.
4. We can initialize numpy arrays from nested Python lists, and access elements using square brackets:

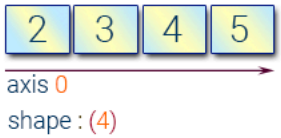
NumPy Basics

Operator	Description
np.array([1,2,3])	1d array
np.array([(1,2,3),(4,5,6)])	2d array
np.arange(start,stop,step)	range array

1D Array

```
>>> import numpy as np
>>> x =  np.arange(2, 6).reshape(4)
>>> x
array( [ 2, 3, 4, 5 ] )

>>>
```



shape : (4)

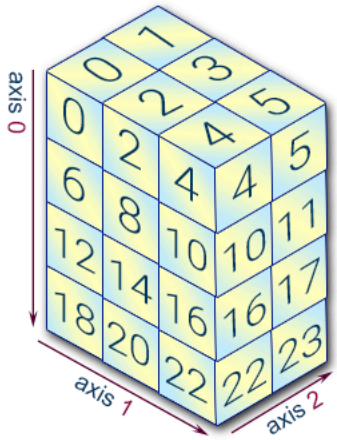
2D Array

```
>>> import numpy as np
>>> x =  np.arange(2, 10).reshape(2, 4)
>>> x
array( [ [ 2, 3, 4, 5 ] )
         [ 6, 7, 8, 9 ] ] )
>>>
```

shape : (2, 4)

3D Array

```
>>> import numpy as np
>>> x =  np.arange(24).reshape(4, 3, 2)
>>> x
array( [ [ [ 0, 1 ] ),      [ [ 6, 7] ),     [ [ 12, 13] ),    [ [ 18, 19] ),
           [ 2, 3] ),        [ 8, 9] ),       [ 14, 15] ),      [ 20, 21] ),
           [ 4, 5] ),        [ 10, 11] ),     [ 16, 17] ),      [ 22, 23] ] ] )
>>>
```



shape : (4, 3, 2)

Placeholders

Operator	Description
np.linspace(0,2,9)	Add evenly spaced values btw interval to array of length
np.zeros((1,2))	Create and array filled with zeros
np.ones((1,2))	Creates an array filled with ones
np.random.random((5,5))	Creates random array
np.empty((2,2))	Creates an empty array

Array

Syntax	Description
array.shape	Dimensions (Rows,Columns)
len(array)	Length of Array
array.ndim	Number of Array Dimensions

Syntax	Description
array.dtype	Data Type
array.astype(type)	Converts to Data Type
type(array)	Type of Array

Copying/Sorting

Operators	Description
np.copy(array)	Creates copy of array
other = array.copy()	Creates deep copy of array
array.sort()	Sorts an array
array.sort(axis=0)	Sorts axis of array

Array Manipulation

Adding or Removing Elements

Operator	Description
np.append(a,b)	Append items to array
np.insert(array, 1, 2, axis)	Insert items into array at axis 0 or 1
np.resize((2,4))	Resize array to shape(2,4)
np.delete(array,1,axis)	Deletes items from array

Combining Arrays

Operator	Description
np.concatenate((a,b),axis=0)	Concatenates 2 arrays, adds to end
np.vstack((a,b))	Stack array row-wise
np.hstack((a,b))	Stack array column wise

Splitting Arrays

Operator	Description
numpy.split()	Split an array into multiple sub-arrays.
np.array_split(array, 3)	Split an array in sub-arrays of (nearly) identical size
numpy.hsplit(array, 3)	Split the array horizontally at 3rd index

More

Operator	Description
other = ndarray.flatten()	Flattens a 2d array to 1d
array = np.transpose(other)	
array.T Transpose array	
inverse = np.linalg.inv(matrix)	Inverse of a given matrix
Mathematics	

Operations

Operator	Description
np.add(x,y)	
x + y	Addition
np.substract(x,y)	
x - y	Subtraction
np.divide(x,y)	
x / y	Division
np.multiply(x,y)	
x @ y	Multiplication
np.sqrt(x)	Square Root
np.sin(x)	Element-wise sine
np.cos(x)	Element-wise cosine
np.log(x)	Element-wise natural log

Operator	Description
np.dot(x,y)	Dot product
np.roots([1,0,-4])	Roots of a given polynomial coefficients

Comparison

Operator	Description
==	Equal
!=	Not equal
<	Smaller than
>	Greater than
<=	Smaller than or equal
>=	Greater than or equal
np.array_equal(x,y)	Array-wise comparison

Basic Statistics

Operator	Description
np.mean(array)	Mean
np.median(array)	Median
array.corrcoef()	Correlation Coefficient
np.std(array)	Standard Deviation

More

Operator	Description
array.sum()	Array-wise sum
array.min()	Array-wise minimum value
array.max(axis=0)	Maximum value of specified axis
array.cumsum(axis=0)	Cumulative sum of specified axis

Slicing and Subsetting

Operator	Description
array[i]	1d array at index i
array[i,j]	2d array at index[i][j]
array[i<4]	Boolean Indexing, see Tricks
array[0:3]	Select items of index 0, 1 and 2
array[0:2,1]	Select items of rows 0 and 1 at column 1
array[:1]	Select items of row 0 (equals array[0:1, :])
array[1:2, :]	Select items of row 1
[comment]: <>	(array[1,...]
array[: :-1]	Reverses array

1. Installation

pip install numpy

2. Motivation

Numpy provides extension package to python for multi dimensional arrays.
Also known as array oriented computing.

creating array in numpy

```
import numpy as np

a = np.arange(10) #using arange function
print(a)

b = np.array([1,2,3,4,5]) #creating array from list
print(b)
```

```
[0 1 2 3 4 5 6 7 8 9]
[1 2 3 4 5]
```

why to use numpy ?

```
L = range(1000)
%timeit [i**2 for i in L]
```

```
299 µs ± 4.36 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
A = np.arange(1000)
%timeit A**2
```

```
1.54 µs ± 367 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

```
print("Numpy implementation is {0} times faster than lists implementation".format(272/1.25))
```

```
Numpy implementation is 217.6 times faster than lists implementation
```

3. Creating arrays

3.1. Manual construction of the arrays

```
#create 1 - D array #vector
a = np.array([1,2,3,4,5])

a
```

```
array([1, 2, 3, 4, 5])
```

```
#print dimensions
a.ndim
```

```
2
```

```
#print shape
a.shape
```

```
(5,)
```

```
len(a)
```

```
4
```

```
#create 2-D array #matrix

b = np.array([[1,2,3],[4,5,6]]) #list of lists

b
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
b.ndim
```

```
2
```

```
b.shape
```

```
(2, 3)
```

```
len(b) #size of first dimension or number of rows
```

```
2
```

```
#create 3 - D array #tensors

c = np.array([[[0,1],[2,3]],[[4,5],[6,7]]])

c
```

```
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
```

```
c.ndim
```

```
3
```

```
c.shape
```

```
(2, 2, 2)
```

```
len(c)
```

```
2
```

3.2. Functions for creating arrays

```
#using arange function

a = np.arange(10)

a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
b = np.arange(1,10,2) #start, end, step

b
```

```
array([1, 3, 5, 7, 9])
```

```
#using linspace #linearspace

c = np.linspace(5, 10, 5) #start, end, number of points

c
```

```
array([ 5.  ,  6.25,  7.5 ,  8.75, 10.  ])
```

```
#common arrays

d = np.ones((3,3))

d
```

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

```
x = np.zeros((3,3))

x
```

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

```
       [0., 0., 0.]])
```

```
y = np.eye(3) #creates a matrix with 1 as the diagonals and 0 as non-diagonal elements
```

```
y
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
z = np.eye(3,2)
```

```
z
```

```
array([[1., 0.],
       [0., 1.],
       [0., 0.]])
```

```
a = np.diag([1,2,3,4]) #construct a diagonal array
```

```
print(a)
```

```
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
```

```
np.diag(a) #extracts the diagonal elements of matrix a
```

```
array([1, 2, 3, 4])
```

```
#creates a array using random
a = np.random.rand(4)
```

```
print(a)
```

```
[0.65322898 0.73547486 0.74396941 0.39548332]
```

4. Datatypes

```
#we can explicitly specify the required data type
a = np.arange(10, dtype='float')
```

```
print(a)
```

```
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

```
b = np.array([1+2j, 5+1j])
```

```
print(b.dtype)
```

```
complex128
```

```
c = np.array([True, False, True])
```

```
print(c.dtype)
```

```
bool
```

5. Indexing and slicing

5.1. Indexing

```
a = np.arange(10)

print(a)
print(a[5])
print(a[-1])
```

```
[0 1 2 3 4 5 6 7 8 9]
5
9
```

```
b = np.diag([1,2,3])

print(b)
print(b[2,2])
```

```
[[1 0 0]
 [0 2 0]
 [0 0 3]]
3
```

```
b[2,1] = 10 #asigning value to a index

print(b)
```

```
[[ 1  0  0]
 [ 0  2  0]
 [ 0 10  3]]
```

5.2. Slicing

```
a = np.arange(10)

print(a[1:10:2]) #[start_value: end_value(exclusive): step]
```

```
[1 3 5 7 9]
```

```
b = np.arange(10)

b[5:] = 10 #assign 10 from index 5 to end

print(b)
```

```
[ 0  1  2  3  4 10 10 10 10 10]
```

```
c = np.arange(5)

b[5:] = c[::-1] #assign in reverse order

print(b)
```

```
[0 1 2 3 4 4 3 2 1 0]
```

6. Copies and views

Slicing operation creates a view on the original array which is just a way of accessing the data. Thus, the original array is not copied in the memory. You can use **np.may_share_memory()** to check whether two arrays share the same memory block.

```
a = np.arange(10)

b = a[::2]

np.shares_memory(a,b)
```

```
True
```

```
b[0] = 10

print(b)
print(a) #a is also updated, since is shares the same location in memory
```

```
[10  2  4  6  8]
[10  1  2  3  4  5  6  7  8  9]
```

```
c = a[::2].copy() #force the copy

np.shares_memory(a,c)
```

```
False
```

```
c[0] = 5

print(c)
print(a)
```

```
[5 2 4 6 8]
[10  1  2  3  4  5  6  7  8  9]
```

7. Fancy Indexing

7.1 Using Boolean mask

```
a = np.random.randint(0,20,15)

print(a)
```

```
[ 3  7  3 16  6  0  7  8  4 12  6 11 12 16  1]
```

```
mask = (a % 2 == 0)
print(mask)
```

```
[False False False False False False False False False False False False
 False False False]
```

```
even_numbers = a[mask]

print(even_numbers)
```

```
[]
```

```
a[mask]=-1 #it can be very useful to assign a new value to sub array

print(a)
```

```
[ 3  7  3 -1 -1 -1  7 -1 -1 -1 -1 11 -1 -1  1]
```

7.2. Using Integer Array

```
import numpy as np
a = np.arange(0,100,10)

print(a)
```

```
[ 0 10 20 30 40 50 60 70 80 90]
```

```
b = a[[2,3,5,2,4]]

print(b)
```

```
[20 30 50 20 40]
```

```
a[[9,7]] = -200

print(a)
print(b)
```

```
[   0   10   20   30   40   50   60 -200   80 -200]
[20 30 50 20 40]
```

8. Numerical Operations on numpy

8.1. Element wise operations

```
a = np.arange(10)

print(a+1)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

```
print(a ** 2)
```

```
[ 0  1  4  9 16 25 36 49 64 81]
```

```
b = np.ones(10) + 1
print("b = ", b)

print("a - b = ",a-b)
```

```
b =  [2. 2. 2. 2. 2. 2. 2. 2. 2. 2.]
a - b =  [-2. -1.  0.  1.  2.  3.  4.  5.  6.  7.]
```

```
print(a*b)
```

```
[ 0.  2.  4.  6.  8. 10. 12. 14. 16. 18.]
```

```
#Matrix Multiplication

c = np.diag([1,2,3,4])

print(c)
print("*"*100)
print(c*c)
print("*"*100)
print(c.dot(c))
```

```
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
****************************************************************************************************
[[ 1  0  0  0]
 [ 0  4  0  0]
 [ 0  0  9  0]
 [ 0  0  0 16]]
****************************************************************************************************
[[ 1  0  0  0]
 [ 0  4  0  0]
 [ 0  0  9  0]
 [ 0  0  0 16]]
```

```
# element comparisions

a = np.array([1,2,5,4])
b = np.array([6,2,9,4])

print(a==b)
```

```
[False  True False  True]
```

```
print(a>b)
```

```
[False False False False]
```

```
print(a<b)
```

```
[ True False  True False]
```

```
print(a<=b)
```

```
[ True  True  True  True]
```

```
#array wise comparision

print(np.array_equal(a,b))
```

```
False
```

```
c = np.array([1,2,5,4])

print(np.array_equal(a,c))
```

```
True
```

Logical Operations

```
a = np.array([1,0,0,1],dtype='bool')
b = np.array([0,1,0,1],dtype='bool')

print(np.logical_or(a,b))
```

```
[ True  True False  True]
```

```
print(np.logical_and(a,b))
```

```
[False False False  True]
```

```
print(np.logical_not(a))
```

```
[False  True  True False]
```

Transcendental Functions:

```
a = np.arange(5)+1

print(np.sin(a))
```

```
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427]
```

```
print(np.log(a))
```

```
[0.         0.69314718 1.09861229 1.38629436 1.60943791]
```

```
print(np.exp(a))
```

```
[  2.71828183   7.3890561   20.08553692  54.59815003 148.4131591 ]
```

Shape Mismatch:

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

b = np.array([5, 10,8,9])

print(a+b)
```

```
[ 6 12 11 13]
```

8.2. Basic Reductions

```
x = np.array([1,2,3,4])

print(np.sum(x))
```

```
10
```

```
y = np.array([[1,2],[3,4]])

print(y)
print("*"*100)
print(y.T)
```

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

```
print(y.sum(axis=0)) #column wise sum
```

```
[4 6]
```

```
print(y.sum(axis=1)) #row wise sum
```

```
[3 7]
```

other reductions

```
print(y.min())
```

```
1
```

```
print(y.max())
```

```
4
```

```
print(y.argmin()) #index of minimum element
```

```
0
```

```
print(y.argmax()) #index of maximum element
```

```
3
```

Logical reductions

```
print(np.all([True, False, False])) #logical and
```

```
False
```

```
print(np.any([True, False, False])) #logical or
```

```
True
```

```
a = np.zeros((50,50))

print(np.any(a!=0)) #checks whether any element in a is not equal to zero
```

```
False
```

Statistics

```
x = np.arange(1,10)

print(np.mean(x))
```

```
5.0
```

```
print(np.median(x))
```

```
5.0
```

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

```
print(np.mean(y,axis=0)) #column wise mean
print(np.mean(y,axis=1)) #row wise mean
```

```
[2.5 3.5 4.5]
[2. 5.]
```

```
print(np.std(x))
```

```
2.581988897471611
```

Example:

```
data = np.loadtxt("dataset/populations.txt")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-8-fe63850b1259> in <cell line: 1>()
----> 1 data = np.loadtxt("dataset/populations.txt")

 3 frames
/usr/local/lib/python3.10/dist-packages/numpy/lib/_datasource.py in open(self, path, mode, encoding, newline)
    531                                      encoding=encoding, newline=newline)
    532         else:
--> 533             raise FileNotFoundError(f"{path} not found.")
    534
    535

FileNotFoundError: dataset/populations.txt not found.
```

```
print(data)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-40-dbd883db58b7> in <cell line: 1>()
----> 1 print(data)

NameError: name 'data' is not defined
```

```
data.shape
```

```
year, hare, lynx, carrot = data.T
```

```
print(year)
```

```
populations = data[:,1:]
```

```
print(populations)
```

```
populations.std(axis=0)
```

```
populations.mean(axis=0)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-f9c90115a8bf> in <cell line: 1>()
----> 1 populations.mean(axis=0)

NameError: name 'populations' is not defined
```

```
#which species has the highest population each year

print(np.argmax(populations, axis=1))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-6fb936716219> in <cell line: 3>()
      1 #which species has the highest population each year
      2
----> 3 print(np.argmax(populations, axis=1))

NameError: name 'populations' is not defined
```

9. Broadcasting

```
from IPython.display import Image

Image("photos/broadcasting.PNG")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/IPython/core/display.py in _data_and_metadata(self, always_both)
   1299         try:
-> 1300             b64_data = b2a_base64(self.data).decode('ascii')
   1301         except TypeError:

TypeError: a bytes-like object is required, not 'str'

During handling of the above exception, another exception occurred:

FileNotFoundError                         Traceback (most recent call last)
                                          2 frames
/usr/local/lib/python3.10/dist-packages/IPython/core/display.py in _data_and_metadata(self, always_both)
   1300             b64_data = b2a_base64(self.data).decode('ascii')
   1301         except TypeError:
-> 1302             raise FileNotFoundError(
   1303                 "No such file or directory: '%s'" % (self.data))
   1304         md = {}

FileNotFoundError: No such file or directory: 'photos/broadcasting.PNG'

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/IPython/core/display.py in _data_and_metadata(self, always_both)
   1299         try:
-> 1300             b64_data = b2a_base64(self.data).decode('ascii')
   1301         except TypeError:

TypeError: a bytes-like object is required, not 'str'

During handling of the above exception, another exception occurred:

FileNotFoundError                         Traceback (most recent call last)
                                          2 frames
/usr/local/lib/python3.10/dist-packages/IPython/core/display.py in _data_and_metadata(self, always_both)
   1300             b64_data = b2a_base64(self.data).decode('ascii')
   1301         except TypeError:
-> 1302             raise FileNotFoundError(
   1303                 "No such file or directory: '%s'" % (self.data))
   1304         md = {}

FileNotFoundError: No such file or directory: 'photos/broadcasting.PNG'

<IPython.core.display.Image object>
```

```
a = np.tile(np.arange(0, 40, 10),(3,1)) #replicate rows 3 times and columns 1 time # change columns
print(a)
print("*"*100)

a = a.T
print(a)
```

```
[[ 0 10 20 30]
 [ 0 10 20 30]
 [ 0 10 20 30]]
****************************************************************************************************
[[ 0  0  0]
 [10 10 10]
 [20 20 20]
 [30 30 30]]
```

```
b = np.array([0,1,2])

print(b)
```

```
c = a + b

print(c)
```

```
a = np.arange(0,40,10)
print(a.shape)
```

```
a = a[:, np.newaxis] #adds a new axis
print(a.shape)
```

```
print(a)
```

```
print(a+b)
```

10. Array Manipulation

10.1. Flattening

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

```
print(a.ravel())
```

```
print(a.T)
```

```
print(a.T.ravel())
```

10.2. Reshaping

```
print(a.shape)
```

```
b = a.ravel()
print(b.shape)
```

```
b = b.reshape(2,3)
print(b.shape)
```

10.3. Adding a new dimension

```
z = np.arange(1,10,2)
print(z.shape)
```

```
z = z[:,np.newaxis]
print(z.shape)
```

```
z = z[:,np.newaxis]
print(z.shape)
```

10.4. Dimension Shuffling

```
a = np.arange(4*3*2)
print(a)
```

```
a = a.reshape(4,3,2) # 4 matrices of 3 rows and 2 columns each
print(a)
```

```
a[3,1,1]
```

10.5. Resizing

```
a = np.arange(5)
a.resize((10,))
```

10.6. Sorting

```
a = np.array([[5,4,3],[2,1,9]])
print(a)
```

```
b = np.sort(a, axis=1)
print(b)
```

```
c = np.sort(a, axis=0)
print(c)
```

```
d = a.ravel()
print(d)
```

```
print(np.sort(d))
```

11. Exercises

(Type you code in the below cell)

1. **Write a NumPy program to convert a list of numeric value into a one-dimensional NumPy array.**

Expected Output:

Original List: [12.23, 13.32, 100, 36.32]

One-dimensional NumPy array: [12.23 13.32 100. 36.32]

```
Start coding or generate with AI.
```

2. **Write a NumPy program to create a 3x3 matrix with values ranging from 2 to 10.**

Expected Output:

[[2 3 4] [5 6 7] [8 9 10]]

```
Start coding or generate with AI.
```

3. *Write a NumPy program to sort an along the first, last axis of an array.**

Sample array: [[2,5],[4,4]]

Expected Output:

Original array: [[4 6] [2 1]]

Sort along the first axis: [[2 1] [4 6]]

Sort along the last axis: [[1 2] [4 6]]

Start coding or generate with AI.

4. **Write a NumPy program to create a contiguous flattened array.**

Expected Output:

Original array: [[10 20 30] [20 40 50]]

New flattened array: [10 20 30 20 40 50]

Start coding or generate with AI.

5. **Write a NumPy program to display all the dates for the month of March, 2017.**

Expected Output:

March, 2017

['2017-03-01' '2017-03-02' '2017-03-03' '2017-03-04' '2017-03-05' '2017-03-06' '2017-03-07' '2017-03-08' '2017-03-09' '2017-03-10' '2017-03-11' '2017-03-12' '2017-03-13' '2017-03-14' '2017-03-15' '2017-03-16' '2017-03-17' '2017-03-18' '2017-03-19' '2017-03-20' '2017-03-21' '2017-03-22' '2017-03-23' '2017-03-24' '2017-03-25' '2017-03-26' '2017-03-27' '2017-03-28' '2017-03-29' '2017-03-30' '2017-03-31']

Start coding or generate with AI.

6. **Write a NumPy program to generate six random integers between 10 and 30.**

Expected Output:

[20 28 27 17 28 29]

Start coding or generate with AI.

7. **Write a NumPy program to create a 5x5 array with random values and find the minimum and maximum values.**

Expected Output

Original Array:

[[0.96336355 0.12339131 0.20295196 0.37243578 0.88105252] [0.93228246 0.67470158 0.38103235 0.32242645 0.40610231] [0.3113495 0.31688 0.79189089 0.08676434 0.60829874] [0.30360149 0.94316317 0.98142491 0.77222542 0.51532195] [0.97392305 0.16669609 0.81377917 0.2165645 0.00121611]]

Minimum and Maximum Values