

Numpy

1 / 1

Install Numpy :

Install a pip package in the current Jupyter kernel.
! pip install numpy.

Install a pip package by command line

pip install numpy.

python list :

lst = [1, 2, 3, 4, 5, 6]

- Import numpy : (import numpy as np.)

Create Numpy array :

arr = np.array(lst)

print(type(arr))

<class 'numpy.ndarray'>

print(arr)

[1 2 3 4 5 6]

Why Numpy ?

Most powerful numerical processing library in python.

Array oriented computing.

Provides extension package to python for multi dimensional array.

Very efficient.

Scientific computation.

- properties of numpy array :

- array : arr
- shape : arr.shape
- Data Type : arr.dtype
- Itemsize : arr.itemsize
- Dimensionality : arr.ndim

#

Functions for creating numpy array :

→ arr = np.arange(start, end, step)

→ arr = np.ones((3, 3)) #
 ↑ ↑
 Rows columns.

1.	1.	1.
1.	1.	1.
1.	1.	1.

→ arr = np.zeros((3, 3)) #
 [0. 0. 0.
 0. 0. 0.
 0. 0. 0.]

→ arr = np.eye(3) #
 ②
 [[1. 0. 0.
 0. 1. 0.
 0. 0. 1.]]

arr = np.eye(3, 2) #
 [[1. 0.
 0. 1.
 0. 0.]]

Datatypes in Numpy :-

Below is a list of all data types in Numpy and the characters used to represent them.

- i - integers
- b - boolean
- str - string
- f - float
- m - timedelta
- M - datetime
- O - Object
- u - unsigned integers
- c - complex float
- U - unicode string
- V - fixed chunk of memory for other type (void)

Example :

```
import numpy as np
lst = [1, 2, 3]
```

```
arr = np.array(lst, dtype='f') # [1. 2. 3.]
arr = np.array(lst, dtype='O') # [1 2 3]
arr = np.array(lst, dtype='str') # ['1' '2' '3']
```

#

Numpy Random Numbers :

①

`np.random.rand` - Generates an array with random numbers that are uniformly distributed between 0 and 1

②

`np.random.randn` - Generates an array with random numbers that are normally distributed, mean = 0 and std = 1

③

`np.random.randint` - Generates an array with random numbers (integers) that are uniformly distributed 0 and given numbers.

④

`np.random.uniform` - Generates an array with random numbers (float) between given numbers.

Examples :

①

$$\text{arr} = \text{np.random.rand}(\text{rows}, \text{columns})$$

$$\text{arr} = \text{np.random.rand}(10, 2)$$

②

$$\text{arr} = \text{np.random.randn}(\text{rows}, \text{columns})$$

$$\text{arr} = \text{np.random.randn}(5, 4)$$

③

$$\text{arr} = \text{np.random.randint}(\text{start}, \text{stop}, \text{size}=(\text{rows}, \text{columns}))$$

$$\text{arr} = \text{np.random.randint}(10, 40, \text{size}=(5, 4))$$

④

$$\text{arr} = \text{np.random.uniform}(\text{start}, \text{stop}, \text{size}=(\text{rows}, \text{columns}))$$

$$\text{arr} = \text{np.random.uniform}(10, 40, \text{size}=(5, 7))$$

Numpy Array - Indexing, Slicing and updating :-

- * Since data in numpy array is stored sequentially, It is possible to access the data with the help of Indexing and Slicing operation.
- * Numpy array's are Mutable. i.e; Data stored in numpy array can be updated/changed.

Indexing :

Randomly generating 1-D array :

```
arr = np.random.randint(100, size=(5,))
```

```
arr # [1 98 59 28 12]
```

Accessing single Index :

```
⇒ arr[2] # 59
```

Accessing Multiple Indexes :

```
⇒ arr[[2, 4]] # [59, 12]
```

Randomly generated 2-D array :

```
arr = np.random.randint(100, size=(5, 4))
```

```
arr
```

# {	[43 32 9 42]
	[39 51 78 75]
	[49 59 87 69]
	[11 78 46 44]
	[40 43 20 42]

Accessing 2nd index :

`arr[2]` # [49 59 87 69]

Accessing (2,1) index :

Syntax-1

`arr[2][1]` # [59]

Syntax-2

`arr[2, 1]` # [59]
Row Index Column Index

Accessing Multiple values :

`arr[[2, 1], [1, 1]]` # [59 51]
Row Indexes Column Indexes

`arr[[1, 3], [1, 2]]` # [51, 46]
(1, 1) (3, 2)
51 46

Indexing with Boolean Arrays :

`arr = np.random.randint(100, size=(6,))`

`arr` # [1 80 68 22 54 68]

`idx = [True, False, True, False, False, True]`

`arr[idx]`
[1 68 68]

Slicing:

Eg: `a88 = np.random.randint(100, size=(10,))`

`a88` # [6 40 86 3 50 33 80 18 84 47]

`a88[1:4]` # [40 86 3]

`a88[0:-4]` # [6 40 86 3 50 33]

`a88[::-2]` # [6 86 50 80 84]

Eg: `a88 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])`

`a88` # [[1 2 3],
[4 5 6],
[7 8 9]]

Remember Syntax-2 for accessing the data in Numpy array:

`a88[::2, 1:3]` # [[2, 3],
[5, 6]]

Row Indexes Column Indexes
from 0 to 2. from 1 to 3
(2 not included (upto 2 only)
upto 1 only) 3 not included)

`a88[1:2, :]` # [[4, 5, 6]]

Updating Data in Numpy Array:

```
arr = np.random.randint(100, size=(5,4))
```

```
print('Original Array:\n', arr)
```

```
arr[1,1] = 99
```

```
print('Updated Array:\n', arr)
```

Original Array:

```
[94 7 65 88]
```

```
[23 72 58 57]
```

```
[48 20 33 52]
```

```
[2 90 5 63]]
```

updated Array:

```
[94 7 65 88]
```

```
[23 99 58 57]
```

```
[48 20 30 52]
```

```
[2 90 5 63]]
```

Numpy Flatten and Ravel :

```
arr = np.random.randint(10,40, size=(2,2))
```

```
arr
```

```
# [[12 19]
 [34 27]]
```

```
flatten_array = arr.flatten() # [12 19 34 27]
```

```
ravel_array = arr.ravel() # [12 19 34 27]
```

Note: `flatten()` and `ravel()` look's like both are performing flatten operation on the array data.

→ Is there any difference b/w `flatten()` & `ravel()`:

update value in the array.

`arr[1,1] = 15`.

Original Array $\rightarrow \begin{bmatrix} 12 & 19 \\ 34 & 27 \end{bmatrix}$

updated Array $\rightarrow \begin{bmatrix} 12 & 19 \\ 34 & 15 \end{bmatrix}$

`flatten_array` $\rightarrow [12 19 34 27]$

`ravel_array` $\rightarrow [12 19 34 15]$

Note: * The main difference b/w `flatten` and `ravel` is the `flatten` creates the copy of the original array and flattens to 1-D array, whereas, `ravel` creates a view of the Original array and changes to 1-D array.

* Whenever we perform update on original Array `ravel` will take care of updates, whereas `flatten` doesn't.

Numpy Reshape :

```
arr = np.random.randint(10,40, size=(3,2))
```

arr

```
# [[15 21]
 [11 39]
 [27 32]]
```

```
arr_reshaped = arr.reshape(2,3)
```

arr_reshaped

```
# [[15 21 11]
 [39 27 32]]
```

Iterating over Numpy Array :

Iterating over a 1-D Array :

```
arr = np.random.randint(10,40, size=(3,))
```

arr : # [15, 27 35]

Looping over items in the array.

```
for item in arr :
```

```
print(item, end=' ')
```

15 27 35 .

Iterating over a 2-D array:

```
arr = np.random.randint(10, 70, size=(3, 2))
```

arr

```
# [[12 19]
 [65 70]
 [51 43]]
```

Looping over rows in the 2-D array:

```
for row in arr:
```

```
print(row) # [12 19]
            [65 70]
            [51 43]
```

Looping over items in the 2-D array:

```
for item in arr.ravel():
```

```
print(item, end=' ') # 12 19 65 70 51 43
```

Iterating using np.nditer():

```
for item in np.nditer(arr):
```

```
print(item, end=' ')
```

```
# 12 19 65 70 51 43.
```

using Ellipsis:

```
for item in np.nditer(arr, op_flags=['readwrite']):
```

```
if item > 20:
```

```
item[...] = (item * 0)
```

arr

```
# [[12, 19],
 [0, 0],
 [0, 0]]
```

Python Operators on Numpy array :

$x = np.array([[1, 2, 3], [4, 5, 6]])$

$x \# [[1 2 3] \\ [4 5 6]]$

$x + 5 \# [[6 7 8] \\ [9 10 11]]$

$x \% 2 \# [[1 0 1] \\ [0 1 0]]$

$x \% 2 == 0 \# [[False True False] \\ [True False True]]$

Numpy Maths :

- Square Root $np.sqrt(4)$

- Exponent $np.exp()$

- Trigonometric sin $np.sin()$

- Trigonometric cos $np.cos()$

- Elementwise Operations :

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

$y = np.array([[5, 6], [7, 8]])$

- Addition : $np.add(x, y) \# [[6 8] \\ [10 12]]$

- Subtraction : $np.subtract(x, y) \# [[-4 -4] \\ [-9 -4]]$

- Multiply : `np.multiply(x, y)` # $\begin{bmatrix} 5 & 12 \\ 21 & 32 \end{bmatrix}$
- Division : `np.divide(x, y)` # $\begin{bmatrix} 0.2 & 0.33333333 \\ 0.42857143 & 0.5 \end{bmatrix}$

Matrix Multiplication :

way-1 : `np.matmul(x, y)`

way-2 : `np.dot(x, y)`

way-3 : $x @ y$

$\begin{bmatrix} 19 & 20 \\ 43 & 50 \end{bmatrix}$

Diagonal Elements :

$x = np.array([[[1, 2, 3], [4, 5, 6]]])$

x # $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

`np.diag(x)` # [1 5]

Transpose of an array :

$x = np.array([[[1, 2, 3], [4, 5, 6]]])$

x # $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

- Transpose : $x.T$ # $\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$

Numpy Statistics :

Example →

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

x # [[1 2]
[3 4]]

np.sum(x) # 10

np.sum(x, axis=0) # columnwise : [4 6]

np.sum(x, axis=1) # Rowwise : [3 7]

Example → x = np.array ([[1, 2, 3], [4, 5, 6]])

x # [[1 2 3]
[4 5 6]]

np.min(x) # 1

np.min(x, axis=0) # columnwise : [1 2 3]

np.max(x, axis=1) # Rowwise : [1 4]

- Minimum : np.min()
- Maximum : np.max()
- Mean : np.mean()
- Median : np.median()
- Variance : np.var()
- Std Dev : np.std()

Miscellaneous Topics :

- Linspace :

syntax → np.linspace (begin, end, No.of.elements)

Example : np.linspace(1, 5, 9)

[1. 1.5 2. 2.5 3. 3.5 4. 4.5 5.]

- sorting :

syntax → np.sort()

```
x = np.array([ [1, 2, 3], [4, 5, 6], [7, 8, 9] ])
```

```
y = np.array([[[7,4,3],[1,9,8],[2,5,6]]])
```

$$y \quad \# \begin{bmatrix} 7 & 4 & 3 \\ 1 & 9 & 8 \\ 2 & 5 & 6 \end{bmatrix}$$

```
np.sort(y) # [3 4 7]
```

08

np.sort(y, axis=1)

[3 4 7]

$$\begin{bmatrix} 1 & 8 & 9 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 5 & 6 \end{bmatrix}$$

Note : By default sorting in ascending order and row wise

`np.sort(y, axis=0)` # [[1 4 3]
[2 5 6]
[7 9 8]]

Stacking :

Example →

`arr_1 = np.arange(5, 15).reshape(2, 5)`

`arr_1`

$\begin{bmatrix} 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \end{bmatrix}$

`arr_2 = np.arange(25, 35).reshape(2, 5)`

`arr_2`

$\begin{bmatrix} 25 & 26 & 27 & 28 & 29 \\ 30 & 31 & 32 & 33 & 34 \end{bmatrix}$

`np.vstack([arr_1, arr_2])`

$\begin{bmatrix} [5, 6, 7, 8, 9], \\ [10, 11, 12, 11, 12], \\ [25, 26, 27, 28, 29], \\ [30, 31, 32, 33, 34] \end{bmatrix}$

`np.hstack([arr_1, arr_2])`

$\begin{bmatrix} [5, 6, 7, 8, 9, 25, 26, 27, 28, 29], \\ [10, 11, 12, 13, 14, 30, 31, 32, 33, 34] \end{bmatrix}$

Concatenate :

`np.concatenate([arr_1, arr_2], axis=0)`

concatenating
(vertical stacking)

$\begin{bmatrix} [5, 6, 7, 8, 9], \\ [10, 11, 12, 13, 14], \\ [25, 26, 27, 28, 29], \\ [30, 31, 32, 33, 34] \end{bmatrix}$

`np.concatenate([arr_1, arr_2], axis=1)`

$\begin{bmatrix} [5, 6, 7, 8, 9, 25, 26, 27, 28, 29], \\ [10, 11, 12, 13, 14, 30, 31, 32, 33, 34] \end{bmatrix}$

Append :

np.append (arr-1, arr-2, axis=0) # Appending - vertical stacking

[[5, 6, 7, 8, 9],

[10, 11, 12, 13, 14],

[25, 26, 27, 28, 29],

[30, 31, 32, 33, 34]]

np.append (arr-1, arr-2, axis=1) # Appending - horizontal stacking.

[[5, 6, 7, 8, 9, 25, 26, 27, 28, 29],

[10, 11, 13, 14, 30, 31, 32, 33, 34]]

Where - process Array Elements conditionally :-

numpy.where (condition, # where True, yield x, otherwise y
[x, y]) # values to choose from.

Eg:

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

arr

[[1 2 3]

[4 5 6]]

np.where (arr % 2 == 0, 'Even', 'odd')

[[odd Even odd]]

[Even odd Even]]

ArgSort :

$arr = np.array([10, -5, 6, -1])$

Indexes : $arr.argsort() \# [1 3 2 0]$

Sorted Array : $arr[arr.argsort()] \# [-5 -1 6 10]$.

Numpy Broadcasting :

- * Start matching the dimensions backward (Right to Left)
 - Compatible - If same numbers appears \otimes
If one of them is 1
 - Incompatible - Otherwise.

Examples :

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline (3,3) & & \end{array} * \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline (3,) \otimes (1,3) & & \end{array} = \begin{array}{|c|c|c|} \hline -1 & 0 & 3 \\ \hline -4 & 0 & 6 \\ \hline -7 & 0 & 9 \\ \hline (3,3) & & \end{array}$$

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline (3,3) & & \end{array} / \begin{array}{|c|c|c|} \hline 3 & 3 & 3 \\ \hline 6 & 6 & 6 \\ \hline 9 & 9 & 9 \\ \hline (3,1) & & \end{array} = \begin{array}{|c|c|c|} \hline .3 & .7 & 1. \\ \hline .6 & .8 & 1. \\ \hline .8 & .9 & 1. \\ \hline (3,3) & & \end{array}$$

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 1 & 2 & 3 \\ \hline 1 & 2 & 3 \\ \hline (3,) & (1,3) & \end{array} * \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 2 & 2 & 2 \\ \hline 3 & 3 & 3 \\ \hline (3,1) & (3,) & \end{array} = \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 2 & 4 & 6 \\ \hline 3 & 6 & 9 \\ \hline (3,3) & & \end{array}$$

Reading CSV into Numpy Array :

import numpy as np

using loadtxt()

arr = np.loadtxt('data/nyc-weather.csv', delimiter=',',
dtype='str')

print(arr)

[['EST' 'Temperature' 'DewPoint' 'Humidity']]

['1/1/2016' '38' '23' '52']

['1/2/2016' '36' '18' '46']

['1/3/2016' '40' '21' '47']

['1/4/2016' '25' '9' '44']

['1/5/2016' '20' '-3' '41']]