Numpy

import numpy as np.

1D Array

arr = np.array([1,2,3]) → [1,2,3]

arr2 = np.zeros(3) → [0,0,0]

np.ones(3) → [1,1,1]

arr3 = np.arange(5) → [0,1,2,3,4]

last not included

arr4 = np.arrange(1,9,2) → [1, 3, 5, 7]

not included

step(2)

np. random . choice ([1,2,3]) → 2

arr5 = np.random. rand (3) → [0.00122.. , 0.12 , 0.34]

. randint(0, 10, 5) → [1, 2, 3, 1, 2]

arr6 = np. empty (3) → [1e-... , — , —]

very arbitrary no.

arr7 = np. full (2, 10) → [10, 10]

no. of elements

value

2D

arr1 = np. array ([[. .], [ ]])

arr2 = np.zeros ((2, 3))

(2×3)

0	0
0	0
0	0

arr3
arr4

arr5 = np. random . rand (2, 3) → [[—,—,—],
[—,—,—]]

directly pass

arr6 = np. empty ((2,3))

arr6 = np. full ((2,2), 3)

array size

value

datatype

A) Integer

(signed) (unsigned)

⇒ int8 , int32 ⇒ uint8 , uint32

int16 , $\boxed{\text{int64}}$ uint16 , uint64

↑ default

B) Float

⇒ float32, $\boxed{\text{float64}}$

$\boxed{\text{print (arr.dtype)}}$

→ np.array([0.1, 0.2])

C) complex no

⇒ $\boxed{\text{complex128}}$, complex64

→ np.array([1+2j , 3+4j])

D) String

Note

① np.array([4,2,3], dtype = 'int32') 2nd

np.array([1,2,3], dtype = np.int32)

② print(np.array([1, 2.3, 4.9]).astype('uint8'))

[1, 2.3, 4.9]
dtype = float64

[1, 2, 4]
dtype = uint8

Attributes

① arr.ndim → 'x'D-Array

② arr.shape → (AxB)

③ arr.size → no. of elements

④ arr.dtype → data type

⑤ arr.itemsize → (data types bytes/8)

⑥ ~~dat~~ arr.data → pointer / returns memory address.

Input /output

('npy' format)

('txt' format)

arr1 = np.array([1,2,3])

⇒ np. save('myfile.npy', arr1)

arr1_load = np.load ('myfile.npy')

⇒ np.savetxt ('myfile.txt', arr1)

arr1_load = np.loadtxt ('myfile.txt')

Indexing

1	2	3	4

index → 0 1 2 3

(-ve)index → -4 -3 -2 -1

arr[[1, 3]] ⟶ [2, 4]

(index)

(2D) arr[[1,2],[3,4]] ⟶ arr(..)

(1,3),(1,4)
(2,3),(2,4)

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

arr[1] → 2

arr[-2] → 2

arr[:] → [1,2,3]

arr [[True, True, False, True]] → [1, 2, 4]

arr[arr>2] → [3,4]

Slicing

array[start : end : step]

index

included

not-included

default (1)

eg. [step=2
leave (1) and select 2nd .

[step=3
leave (2) and select 3rd one.

or

Jump

default ⇒ (+1)

↳ jump to next (right side)

if (step ⇒ -1)

↳ jump to prev one (left side)

arr = np.array([0,1,2,3,4,5])

arr [2:5] → [2, 3, 4]

arr [-3:] → [3, 4,5]

arr [-5::2] → [1, 3, 5]

default

depend on(step)

similar logic for start

if (step=+ve)
'end' will be last (element)'s index +1)

if (step = -ve)
'end' will be first (element)'s index -1).

eg arr = np.array([0, 1, 2, 3, 4])

0 1 2 3 4

-5 -4 -3 -2 -1

① arr[::-1] → [4, 3, 2, 1, 0]

(-ve)

start : (-1)

end : (-6)

③ arr[-2::-2] → [3, 1]

(-ve)

start : (-2)

end : (-6)

② arr[-2::-1] → [3, 2, 1, 0]

(-ve)

start : (-2)

end : (-6)

Reshaping

result = np.reshape(array2nd, newshape, order = 'C')

default = 'C'
'F'
'A'

2D → (2,3)

3D → (2,2,2)

(Back to) 1D → -1

Arithmatic operations

condn : arr1.shape = arr2.shape

① ADD) ⇒ arr1 + arr2, np.add(arr1, arr2)

② sub) ⇒ ar1 - ar2 , np.subtract(ar1, ar2)

③ divide ⇒ ar1/ar2 , np.divide(ar1, ar2)

④ multiply ⇒ ar1 × ar2 , np.multiply(ar1, ar2)

⑤ power ⇒ ar1 ** 2 , np.power(ar1, 2).
np.sqrt(ar1)

⑥ mod ⇒ ar1 % ar2 , np.mod(ar1, ar2).

⑦ logical ⇒ ar1 > ar2 → [True, False, True]

(>, <, >=, <=, ==, !=)

np.less() , np.greater() , np.equal()

np.less_equal() , np.greater_equal() , np.not_equal()

⑧ np.logical_and() , np.logical_or() , np.logical_not()

Note

① arr2 = np.transpose(arr1)

↓

transpore matrix(arr1)

1	4
2	5
3	6

(arr1) →

1	2	3
4	5	6

(arr2)

② np.mean(),
 .median()
 .min( )
 .max( )
 .std()
 ↳ standard deviation

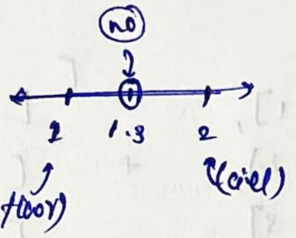
→ (arr, axis)
(for 'N'D Array)
 ⇒ 0 (vertical)
 → 1 (horz)

③ trigon
np. sin()
 . cos()
 . tan()

, np.arcsin()
 , arccos()
 , arctan()

np. radian()
 . degree()

④ np. round(arr1 , precision)
 . ceil()
 . floor()



⑤ np.pi → 3.14158...
np.e → 2.71...
np.exp() → e^x

⑧ np. linspace (1, 10, 10)
↳ generate array from 1 to 10 having total 10 elements equal spaces apart.

→ [1, 2, 3, 4, ... 9, 10]

⑥ np. percentile(arr, x)

value of 'x' percentile

⑦ np. argsort(np.array([8, 3, 0, 8])) $\xrightarrow[\text{sorted index}]{\text{return}}$ [2, 1, 3, 0]

0 1 2 3 (index)

strings

ar = np. array(['A' , 'BCD'])

np. char. add( )
 . multiply (arr, 2) → ['AA' , 'BCD BCD']
 . Captalize () → ['A' , 'Bcd']
 . upper ()
 . lower ()
 . join('-', ar) → ['A' , 'B-C-D']
 . equal( ) → [True, False, True]

Array Broadcasting

eg

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

① $\begin{bmatrix} 0 & 1 & 4 \\ 2 & 3 & 5 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 3 & 7 \\ 3 & 5 & 8 \end{bmatrix}$

$+ \begin{bmatrix} 1 \\ 2 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & 5 \\ 4 & 5 & 7 \end{bmatrix}$

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix}$$

②

```
[
   [1],
   [2],         +   [1 2 3]   →
   [3],
]
```

```
[ [1 1 1],         [ [1 2 3],         [ [2 3 4],
  [2 2 2],     +     [1 2 3],    →      [3 4 5],
  [3 3 3]            [1 2 3]            [4 5 6]
]                  ]                  ]
```

Matrix operation

→ np. dot (ar1 , ar2)
- · transpose (ar1)
- · linalg. inv() → inverse
- · linalg. det() → determinant
- · flatten() → convert into 1D

  np . reshape (ar, -1) same

- · solve (A, b) → return 'x' Array

  Ax = b

- · trace ()

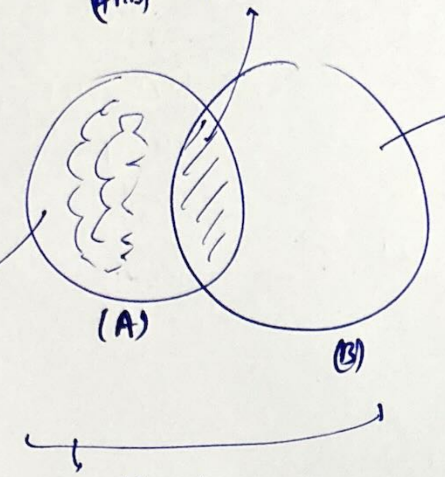
- · inner (ar1 , ar2) → calculate inner product ar1 & ar2
- · outer (ar1 , ar2) → — " — outer — " —

~~union~~ set operation

(A∩B) np.intersect1d(A,B)

(AUB - A∩B)

np.setxor1d(A,B)



(A-B)

np.setdiff1d(A,B)

(AUB) np.union1d(A,B)

unique element

np.unique(arr)

vectorization

```
arr = np.array([-1, 0, 1, 2, 3])
```

if to perform some kind of operation to each element

```
for i in [0,5]:
    arr[i] = my-logic(arr[i])
```

Better to do this way

```
*  my-logic
   mylogic_vector = np.vectorize( )
result_arr = my_logic_vector(arr)
```

Interpolation → find data for missing input

```
x = np.array([1 2, 3, 5])
y = np.array([10, 20, 30, 50])

result = np.interp(4, x, y) → 40
```

get value for 'y' at (x = 4)