Contents

..Thuật toán
Python
Framework
Machine learning
Deep learning
Mấy cái sota
Linear algebra
Statistic & probability
Pytorch, sklearn, caffe, tensorflow, keras

Python NumPy

NumPy Getting Started

```
import numpy
arr = numpy.array([1, 2, 3, 4, 5])
print(arr)
```

```
[1 2 3 4 5]
```

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

```
[1 2 3 4 5]
```

NumPy Creating Arrays

```
import numpy as np
arr = np.array(42)
print(arr)
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

```
42
[1 2 3 4 5]
[[1 2 3]
 [4 5 6]]
```

NumPy Array Indexing

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[2] + arr[3])

import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st dim: ', arr[0, 1])
print('Last element from 2nd dim: ', arr[1, -1])
```

```
7
2nd element on 1st dim:  2
Last element from 2nd dim:  10
```

NumPy Array Slicing

Cắt mảng:

[*start*:*end*:*step*].

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
print(arr[::2])
print()
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
print(arr[0:2, 2])
print(arr[0:2, 1:4])
```

```
[2 4]
[1 3 5 7]

[7 8 9]
[3 8]
[[2 3 4]
 [7 8 9]]
```

NumPy Data Types

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

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
print()
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='S')
print(arr)
print(arr.dtype)
```

```
int64

[b'1' b'2' b'3' b'4']
|S1
```

NumPy Array Copy vs View

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42
print(arr)
print(x)
print()
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42
print(arr)
print(x)
print()
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
x[0] = 31
print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]

[42  2  3  4  5]
[42  2  3  4  5]

[31  2  3  4  5]
[31  2  3  4  5]
```

NumPy Array Shape

#The example above returns (2, 4), which means that the array has 2 dimensions, and each dimension has 4 elements.

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

```
(2, 4)
```

NumPy Array Reshaping :định hình

#Convert the following 1-D array with 12 elements into a 2-D array.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
print()
#You are allowed to have one "unknown" dimension.
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
newarr = arr.reshape(2, 2, -1)
print(newarr)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```


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

 [[5 6]
  [7 8]]]
```

NumPy Array Iterating :lặp lại

```
import numpy as np
arr = np.array([1, 2, 3])
for x in arr:
  print(x,end=' ')
print()
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
for x in arr:
  print(x)
for x in arr:
  print()
  for y in x:
    print(y,end=' ')
```

```
1 2 3
[1 2 3]
[4 5 6]

1 2 3
4 5 6
```

```
import numpy as np
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
for x in np.nditer(arr):
  print(x,end=' ')
print()
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
for x in np.nditer(arr[:, ::2]):
  print(x,end=' ')
```

```
1 2 3 4 5 6 7 8
1 3 5 7
```

NumPy Joining Array

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))
print(arr)
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
print(arr)
arr = np.concatenate((arr1, arr2), axis=0)
print(arr)
```

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

```
#Nối các mảng bằng cách sử dụng các hàm ngăn xếp
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
#nối hai mảng 1-D dọc theo trục , điều này sẽ dẫn đến việc xếp chúng chồng lên nhau
arr = np.stack((arr1, arr2), axis=0)
print(arr)
arr = np.stack((arr1, arr2), axis=1)
print(arr)
#hstack() xếp chồng dọc theo hàng.
arr = np.hstack((arr1, arr2))
print(arr)
#vstack() xếp chồng dọc theo các cột
arr = np.vstack((arr1, arr2))
print(arr)
#dstack() xếp chồng theo chiều cao, bằng với chiều sâu.
arr = np.dstack((arr1, arr2))
print(arr)
```

```
[[1 2 3]
 [4 5 6]]
[[1 4]
 [2 5]
 [3 6]]
[1 2 3 4 5 6]
[[1 2 3]
 [4 5 6]]
[[[1 4]
  [2 5]
  [3 6]]]
```

NumPy Splitting Array

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)#dùng  array_split,tách thành 3 lần
print(newarr)
print(newarr[0])
print(newarr[1])
print(newarr[2])
```

```
newarr = np.array_split(arr, 4)
print(newarr)
```
```
[array([1, 2]), array([3, 4]), array([5, 6])]
[1 2]
[3 4]
[5 6]
[array([1, 2]), array([3, 4]), array([5]), array([6])]
```
Tách mảng 2 chiều
```
#Chia mảng 2-D thành ba mảng 2-D.
import numpy as np
arr = np.array([[1, 2], [3, 4],
                                [5, 6], [7, 8],
                            [9, 10], [11, 12]])
newarr = np.array_split(arr, 3)
print(newarr)
#Tách mảng 2-D thành 1 mảng 2-D dọc theo các hàng/cột.
newarr=np.array_split(arr,1,axis=1)
print(newarr)
#hsplit() ,vsplit()và dsplit() có thể thay thế cho hstack(),vstack()và dstack()
newarr = np.hsplit(arr,2)
print(newarr)
```
```
[array([[1, 2],
       [3, 4]]), array([[5, 6],
       [7, 8]]), array([[ 9, 10],
       [11, 12]])]
[array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12]])]
[array([[ 1],
       [ 3],
       [ 5],
       [ 7],
       [ 9],
       [11]]), array([[ 2],
       [ 4],
       [ 6],
       [ 8],
       [10],
       [12]])]
```

NumPy Searching Arrays
```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)
x = np.where(arr%2 == 0)
print(x)
#searchsorted(): tìm kiếm trong mảng và trả về chỉ mục nơi giá trị được chỉ định sẽ được chèn vào để duy trì thứ tự tìm kiếm.import numpy as np
#[1 2 3]chứa ba chỉ mục trong đó 2, 4, 6 sẽ được chèn vào mảng ban đầu để duy trì thứ tự.
x = np.searchsorted(arr, [2, 4, 6])#tìm kiếm nhiều
print(x)
arr = np.array([6, 7, 8, 9])
#Số 7 nên được chèn vào chỉ mục 1 để giữ nguyên thứ tự sắp xếp.
x = np.searchsorted(arr, 7)
print(x)
#Số 7 nên được chèn vào chỉ mục 2 để giữ nguyên thứ tự sắp xếp.
x = np.searchsorted(arr, 7, side='right')
print(x)
```
```
(array([3, 5, 6]),)
(array([1, 3, 5, 6]),)
[1 3 7]
1
2
```

NumPy Sorting Arrays
```
import numpy as np
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
arr = np.array(['banana', 'cherry', 'apple'])
print(np.sort(arr))
arr = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr))
```
```
[0 1 2 3]
['apple' 'banana' 'cherry']
[[2 3 4]
 [0 1 5]]
```

NumPy Filter Array Mảng lọc
```
#Getting some elements out of an existing array and creating a new array out of them is called filtering.
import numpy as np
arr = np.array([41, 42, 43, 44])
x = arr[[True, False, True, False]]
print(x)

# Create an empty list
filter_arr = []
# go through each element in arr
for element in arr:
  # if the element is higher than 42, set the value to True, otherwise False:
  if element > 42:
    filter_arr.append(True)
  else:
```

```
    filter_arr.append(False)

newarr = arr[filter_arr]
print(filter_arr)
print(newarr)

filter_arr = arr > 42
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])
filter_arr = arr % 2 == 0
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

```
[41 43]
[False, False, True, True]
[43 44]
[False False  True  True]
[43 44]
[False  True False  True False  True False]
[2 4 6]
```

NumPy ufuncs
ufuncs là viết tắt của "Universal Functions" và chúng là các hàm NumPy hoạt động trên ndarrayđối tượng. ufuncs được sử dụng để triển khai vectơ *hóa* trong NumPy, cách này nhanh hơn so với việc lặp qua các phần tử.

Create Your Own ufunc

Để tạo ufunc của riêng bạn, bạn phải xác định một hàm, giống như bạn làm với các hàm bình thường trong Python, sau đó bạn thêm nó vào thư viện ufunc NumPy của mình bằng frompyfunc()phương thức.
Các frompyfunc()phương pháp có những lập luận như sau:

1. *function* - tên của hàm.
2. *inputs* - số lượng đối số đầu vào (mảng).
3. *outputs* - số lượng mảng đầu ra.

```
import numpy as np
def myadd(x, y):
  return x+y
myadd = np.frompyfunc(myadd, 2, 1)
print(myadd([1, 2, 3, 4], [5, 6, 7, 8]))
#Kiểm tra xem một Hàm có phải là một chức năng không
print(type(np.add))
print(type(np.concatenate))
#print(type(np.blahblah)) => lỗi
```

```
[6 8 10 12]
<class 'numpy.ufunc'>
<class 'builtin_function_or_method'>
```

Simple Arithmetic : Số học đơn giản

Bạn có thể sử dụng các toán tử số học + - * / trực tiếp giữa các mảng NumPy, nhưng phần này thảo luận về một phần mở rộng tương tự, trong đó chúng ta có các hàm có thể lấy bất kỳ đối tượng giống mảng nào, ví dụ danh sách, bộ giá trị, v.v. và thực hiện số học có *điều kiện* .

Arithmetic Conditionally: có nghĩa là chúng ta có thể xác định các điều kiện mà phép toán số học sẽ xảy ra.

```
import numpy as np
arr1 = np.array([10, 11, 12, 13, 14, 15])
arr2 = np.array([20, 21, 22, 23, 24, 25])
newarr = np.add(arr1, arr2)
print(newarr)
newarr = np.subtract(arr1, arr2)
print(newarr)
newarr = np.multiply(arr1, arr2) #1D
print(newarr)
newarr = np.divide(arr1, arr2)
print(newarr)
newarr = np.power(arr1, arr2)
print(newarr)
```

```
[30 32 34 36 38 40]
[-10 -10 -10 -10 -10 -10]
[200 231 264 299 336 375]
[0.5        0.52380952 0.54545455 0.56521739
0.58333333 0.6        ]
[ 7766279631452241920  3105570700629903195
5729018530666381312
 -4649523274362944347 -1849127232522420224
1824414961309619599]
```

```
import numpy as np
arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 7, 9, 8, 2, 33])
newarr = np.mod(arr1, arr2)#phần dư
print(newarr)
#Bạn nhận được kết quả tương tự khi sử dụng remainder()hàm:
newarr = np.remainder(arr1, arr2)
print(newarr)
newarr = np.divmod(arr1, arr2)
print(newarr)
newarr = np.absolute(arr1)#giá trị tuyệt đối
print(newarr)
```

#Ví dụ trên sẽ trả về [1 6 3 0 0 27] là phần còn lại khi bạn chia (10% 3), (20% 7) (30% 9), v.v.
#Ví dụ dưới sẽ trả về: (array ([3, 2, 3, 5, 25, 1]), array ([1, 6, 3, 0, 0, 27]))Mảng đầu tiên đại diện cho các thương số .Mảng thứ hai đại diện cho phần dư

```
[ 1  6  3  0  0 27]
[ 1  6  3  0  0 27]
(array([ 3,  2,  3,  5, 25,  1]), array([ 1,  6,  3,  0,  0, 27]))
[10 20 30 40 50 60]
```

Rounding Decimals

Chủ yếu có năm cách làm tròn số thập phân trong NumPy: truncation, fix ,rounding ,floor, ceil
#Loại bỏ các số thập phân và trả về số thực gần nhất với số 0. Sử dụng các chức năng trunc()và fix().
#around()gia số chức năng làm tròn n chữ số thập phân
#Hàm floor () làm tròn số thập phân thành số nguyên thấp hơn gần nhất.
#Hàm ceil () làm tròn số thập phân thành số nguyên trên gần nhất.

```
import numpy as np
arr = np.trunc([-3.1666, 3.6667])
print(arr)
arr = np.fix([-3.1666, 3.6667])
print(arr)
arr = np.around(3.1666, 2)
print(arr)
arr = np.floor([-3.1666, 3.6667])
print(arr)
arr = np.ceil([-3.1666, 3.6667])
print(arr)
```

```
[-3.  3.]
[-3.  3.]
3.17
[-4.  3.]
[-3.  4.]
```

NumPy Logs : hàm logrit

có cơ số 2, e và 10.
Tất cả các hàm log sẽ đặt -inf hoặc inf trong các phần tử nếu không thể được tính toán.

```
import numpy as np
arr = np.arange(1, 10)
#arange(1, 10)hàm trả về một mảng với số nguyên bắt đầu từ 1  đến 9
print(np.log2(arr))
print(np.log10(arr))
print(np.log(arr)) # cơ số e
```

```
[0.         1.         1.5849625  2.         2.32192809 2.5849625
 2.80735492 3.         3.169925  ]
[0.         0.30103    0.47712125 0.60205999 0.69897    0.77815125
 0.84509804 0.90308999 0.95424251]
[0.         0.69314718 1.09861229 1.38629436 1.60943791 1.79175947
 1.94591015 2.07944154 2.19722458]
```

NumPy Summations : tổng

Addition is done between two arguments whereas summation happens over n elements.
Phép cộng được thực hiện giữa hai đối số trong khi tính tổng xảy ra trên n phần tử.

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([1, 2, 3])
newarr = np.add(arr1, arr2)
print(newarr)
newarr = np.sum([arr1, arr2])
print(newarr)
#Nếu bạn chỉ định axis=1, NumPy sẽ tính tổng các số trong mỗi mảng.
newarr = np.sum([arr1, arr2], axis=1)
print(newarr)
#Thực hiện tính tổng tích lũy :cursum
newarr = np.cumsum(arr1)
print(newarr)
```

```
[2 4 6]
12
[6 6]
[1 3 6]
```

NumPy Products

#Để tìm tích của các phần tử trong một mảng, hãy sử dụng prod()hàm.

```
import numpy as np
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([5, 6, 7, 8])
x=np.prod(arr1)
print(x)
x = np.prod([arr1, arr2])
print(x)
#trên một trục : axis=1 (hàng), 0(cột)
newarr = np.prod([arr1, arr2], axis=0)
print(newarr)
#kết quả tích lũy:
newarr = np.cumprod(arr1)
print(newarr)
```

```
24
40320
[ 5 12 21 32]
[ 1  2  6 24]
```

NumPy Differences

```
import numpy as np
arr = np.array([10, 15, 25, 5])
#Một sự khác biệt rời rạc có nghĩa là trừ hai phần tử liên tiếp.(sau trừ trước)
newarr = np.diff(arr)
print(newarr)
#Tính toán chênh lệch rời rạc của mảng sau hai lần:
newarr = np.diff(arr, n=2)
print(newarr)
```

```
[  5  10 -20]
[  5 -30]
```

NumPy LCM Lowest Common Multiple bcnn

```
import numpy as np
#giữa 2 số
num1 = 4
num2 = 6
x = np.lcm(num1, num2)
print(x)
#Tìm LCM trong Mảng
arr = np.array([3, 6, 9])
x = np.lcm.reduce(arr)
print(x)
```

```
12
18
```

NumPy LCM Lowest Common Multiple UCnL

```
import numpy as np
# UCnL của hai số
num1 = 6
num2=  9
x = np.gcd(num1, num2)
print(x)
#Để tìm  UCnL của tất cả các giá trị trong một mảng, bạn có thể sử dụng reduce()phương pháp này:
arr = np.array([20, 8, 32, 36, 16])
x = np.gcd.reduce(arr)
print(x)
```

```
3
4
```

NumPy Trigonometric Functions: Hàm lượng giác

```
import numpy as np
#NumPy cung cấp ufuncs sin(), cos()và tan()đó sẽ có giá trị trong radian
x = np.sin(np.pi/2)
print(x)
arr = np.array([np.pi/2, np.pi/3, np.pi/4, np.pi/5])
x = np.sin(arr)
print(x)
#Chuyển đổi độ sang Radian
arr = np.array([90, 180, 270, 360])
x = np.deg2rad(arr)
print(x)
#=>Radian sang Độ: x = np.rad2deg(arr)
#Tìm góc
x = np.arcsin(1.0)
print(x)
arr = np.array([1, -1, 0.1])
x = np.arcsin(arr)
print(x)
#cạnh huyền bằng cách sử dụng định lý pythagoras trong NumPy.
base = 3
perp = 4
x = np.hypot(base, perp)
print(x)
```

```
1.0
[1.         0.8660254  0.70710678 0.58778525]
[1.57079633 3.14159265 4.71238898 6.28318531]
1.5707963267948966
[ 1.57079633 -1.57079633  0.10016742]
5.0
```

NumPy Hyperbolic Functions:

```
import numpy as np
#NumPy cung cấp ufuncs sinh(), cosh()và tanh()đó sẽ có giá trị trong radian
x = np.sinh(np.pi/2)
print(x)
arr = np.array([np.pi/2, np.pi/3, np.pi/4, np.pi/5])
x = np.cosh(arr)
print(x)
#Tìm góc:Tìm các góc từ các giá trị của hypebolic sin, cos, tan. Vd: sinh, cosh và tanh nghịch đảo (arcsinh, arccosh, arctanh).
#NumPy cung cấp ufuncs arcsinh(), arccosh()và arctanh()sản xuất giá trị radian cho tương ứng sinh, cây vồ và tanh giá trị nhất định.
x = np.arcsinh(1.0)
print(x)
arr = np.array([0.1, 0.2, 0.5])
x = np.arctanh(arr)
print(x)
```

```
2.3012989023072947
[2.50917848 1.60028686 1.32460909 1.20397209]
0.881373587019543
[0.10033535 0.20273255 0.54930614]
```

NumPy Set Operations

Một tập hợp trong toán học là một tập hợp các phần tử duy nhất.

Các tập hợp được sử dụng cho các phép toán liên quan đến các phép toán giao nhau, liên hợp và chênh lệch thường xuyên.

```
#Tạo set x trong NumPy
import numpy as np
arr = np.array([1, 1, 1, 2, 3, 4, 5, 5, 6, 7])
#sử dụng unique()phương pháp của NumPy để tìm các phần tử duy nhất từ bất kỳ mảng nào
x = np.unique(arr)
print(x)
#Để tìm các giá trị duy nhất của hai mảng của hai mảng, hãy sử dụng union1d()phương pháp này.
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([3, 4, 5, 6])
newarr = np.union1d(arr1, arr2)
print(newarr)
#Để chỉ tìm các giá trị có trong cả hai mảng, hãy sử dụng intersect1d()
newarr = np.intersect1d(arr1, arr2, assume_unique=True)
print(newarr)
#Để chỉ tìm các giá trị trong tập đầu tiên KHÔNG có trong tập 2, hãy sử dụng setdiff1d()
newarr = np.setdiff1d(arr1, arr2, assume_unique=True)
print(newarr)
#Để chỉ tìm các giá trị KHÔNG có trong CẢ HAI tập hợp, hãy sử dụng setxor1d()
newarr = np.setxor1d(arr1, arr2, assume_unique=True)
print(newarr)
```

```
[1 2 3 4 5 6 7]
[1 2 3 4 5 6]
[3 4]
[1 2]
[1 2 5 6]
```