Xử lý dữ liệu mảng với Numpy

Giảng viên: TS**. Nguyễn Văn Quyết**

Email: quyetict@gmail.com

Nguyễn Văn Quyết

- **Đào tạo:**
  - 2015-2019: Tiến sĩ KHMT, Đại học Quốc gia Chonnam, Hàn Quốc
  - 2011-2013: Thạc sĩ CNTT, Đại học Bách Khoa Hà Nội
- **Lĩnh vực nghiên cứu:**
  - Phân tích dữ liệu lớn (Big Data Analytics)
  - Xử lý đồ thị lớn (Big Graph Processing)
  - Tính toán song song (Parallel Computing)
  - Các hệ thống phân tán (Distributed Systems)
  - Kết nối vạn vật (Internet of Things)

Nội dung

- Giới thiệu về NumPy
- Kiểu dữ liệu trong NumPy
- Các thao tác cơ bản trong NumPy
- Các hàm tính toán trong NumPy
- Sắp xếp mảng trong NumPy
- Mảng cấu trúc trong NumPy
- Các hàm tính toán nâng cao với SciPy

NumPy là gì?

- NumPy (Numerical Python) là một thư viện Python được sử dụng trong tính toán khoa học và phân tích dữ liệu.
- NumPy cung cấp những tính năng sau:
  - **ndarray**, một kiểu mảng nhiều chiều nhanh và hiệu quả về lưu trữ cung cấp khả năng tính toán vector.
  - Các hàm toán học tiêu chuẩn cho việc tính toán nhanh trên toàn bộ mảng dữ liệu mà không cần phải thông qua vòng lặp.
  - Công cụ cho đọc/ ghi dữ liệu mảng trên đĩa và làm việc với các tập tin ánh xạ dữ liệu (memory-mapped files).
  - Khả năng làm việc với đại số tuyến tính, tạo số ngẫu nhiên, và phép biến đổi Fourier.
  - Công cụ để nhúng code được viết bằng C, C++, và Fortran.

Cài đặt NumPy

- Để cài đặt NumPy, ta tiến hành kích hoạt môi trường thực thi dòng lệnh:
  - conda install numpy
  - Hoặc conda install numpy= (số phiên bản) để cài phiên bản numpy mong muốn.
- Để sử dụng NumPy trong Python ta khai báo:

```
import numpy as np
```

Mảng trong NumPy

- **ndarray,** viết tắt của N-dimensional array**,** là một đối tượng trong thư viện NumPy cung cấp khả năng tương tác, lưu trữ và làm việc với các tập hợp mảng dữ liệu nhiều chiều.
- Tất cả các phần tử trong mảng phải cùng một kiểu dữ liệu, cấu trúc của mảng không thể thay đổi như việc thêm hay gỡ một phần tử khỏi mảng vì vậy các hàm trong NumPy thường sẽ trả về mảng mới hoặc một mảng tham chiếu đến mảng ban đầu.

Mảng trong NumPy

- Các mảng ndarray đều có các thuộc tính sau:
  - shape: là một tuple đại diện cho kích thước của mỗi chiều.
  - dtype: là một đối tượng miêu tả kiểu dữ liệu của mảng.

```
In [69]: arr=np.array([[0,3,4,61,2],[33,43,34,5,3]])
         print(arr.shape)
         print(arr.dtype)

         (2, 5)
         int32
```

Kiểu dữ liệu được sử dụng trong NumPy

- Mảng của NumPy không phải là một mảng thông thường có thể chứa bất kỳ đối tượng nào. Những kiểu dữ liệu mà NumPy có thể lưu trữ phải được miêu tả bởi một đối tượng dtype.
- dtype, viết tắt của data type, là các đối tượng chứa thông tin miêu tả cách mà dữ liệu được ghi và xử lý trên bộ nhớ.
  - dtype là một phần quan trọng giúp cho NumPy trở nên mềm dẻo và mãnh mẽ.
  - Trong nhiều trường hợp, nó ánh xạ trực tiếp dữ liệu với các kiểu dữ liệu bậc thấp làm cho nó dễ dàng xử lý với C hoặc Fortran.

```
In [27]: arr1 = np.array([1, 2, 3], dtype=np.float64)

In [28]: arr2 = np.array([1, 2, 3], dtype=np.int32)

In [29]: arr1.dtype              In [30]: arr2.dtype
Out[29]: dtype('float64')        Out[30]: dtype('int32')
```

Các kiểu dữ liệu dtype được hỗ trợ bởi NumPy

Kiểu dữ liệu	Kiểu Code
int8, uint8	i1, u1
int16, uint16	i2, u2
int32, uint32	i4, u4
int64, uint64	i8, u8
float16	f2
float32	f4 or f
float64, float128	f8 or d
float128	f16 or g
complex64, complex128,	c8, c16,
complex256	c32
bool	?
object	O
string_	S
unicode_	U

Ép kiểu dữ liệu trong NumPy

- Sử dụng hàm astype để tạo ra một mảng mới từ mảng cũ với kiểu dữ liệu mong muốn.

```
In [66]: string_arr=np.array(["34","45","46","54"])
         print(string_arr)
         int_arr=string_arr.astype(np.int32)
         int_arr

         ['34' '45' '46' '54']

Out[66]: array([34, 45, 46, 54])
```

Tạo mảng với NumPy

- Dùng hàm array để tạo mảng NumPy từ các đối tượng có tính chất tập hợp (sequence-like object) như là tuple, list,…

```
In [2]: data1=[1,2,3,4,5]
        arr1=np.array(data1)
        arr1

Out[2]: array([1, 2, 3, 4, 5])

In [5]: data2=("Khoa","Học","Máy","Tính")
        arr2=np.array(data2)
        arr2

Out[5]: array(['Khoa', 'Học', 'Máy', 'Tính'], dtype='<U4')
```

Tạo mảng với NumPy

- Đối với các tập hợp lồng (nested sequence) có các tập hợp con có độ dài giống nhau sẽ được hàm array chuyển đổi thành mảng đa chiều.

```
In [8]: data3=[[1,2,3,4],[5,6,7,8]]
        arr3=np.array(data3)
        arr3

Out[8]: array([[1, 2, 3, 4],
               [5, 6, 7, 8]])
```

Tạo mảng với NumPy

- Nếu không được người dùng định nghĩa, NumPy sẽ dự đoán kiểu dữ liệu phù hợp cho mảng được tạo.

```
In [9]: print(arr1.dtype)
        int32
```

Tạo mảng với NumPy

- Người dùng cũng có thể tự định nghĩa kiểu dữ liệu mong muốn cho mảng bằng cách thiết đặt với tham số dtype trong hàm array.

```
In [14]: data4=[1,2,3,4]
         arr4=np.array(data4, dtype=np.float64)
         print(arr4)
         print(arr4.dtype)

         [1. 2. 3. 4.]
         float64
```

Tạo các mảng đặc biệt với NumPy

- NumPy cung cấp các hàm đặc biệt để tạo các mảng đặc biệt phục vụ cho mục đích tính toán khoa học.
  - Hàm zeros cho phép tạo mảng có các phần tử bằng 0 với kích thước mong muốn.

```
In [18]: arr5=np.zeros(5, dtype=np.int32)
         arr5
Out[18]: array([0, 0, 0, 0, 0])
```

Tạo các mảng đặc biệt với NumPy

- Hàm ones cho phép tạo mảng với các phần tử bằng 1.

```
In [20]: arr6=np.ones(6, dtype=np.float64)
         arr6
Out[20]: array([1., 1., 1., 1., 1., 1.])
```

Tạo các mảng đặc biệt với NumPy

- Tạo mảng với các phần tử trong khoảng mong muốn với hàm arange.

```
In [22]: #Tạo một mảng có giá trị bắt đầu từ 0, kết thuc ở 20, số bước nhảy bằng 2
         arr7=np.arange(0,20,2)
         arr7

Out[22]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

Tạo các mảng đặc biệt với NumPy

- Tạo mảng với số phần tử cách đều nhau trong một khoảng.

```
In [23]: #Tạo mảng có 5 phần tử chia đều khoảng từ 0 đến 1
         arr8=np.linspace(0,1,5)
         arr8

Out[23]: array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

Truy xuất các phần tử của mảng NumPy

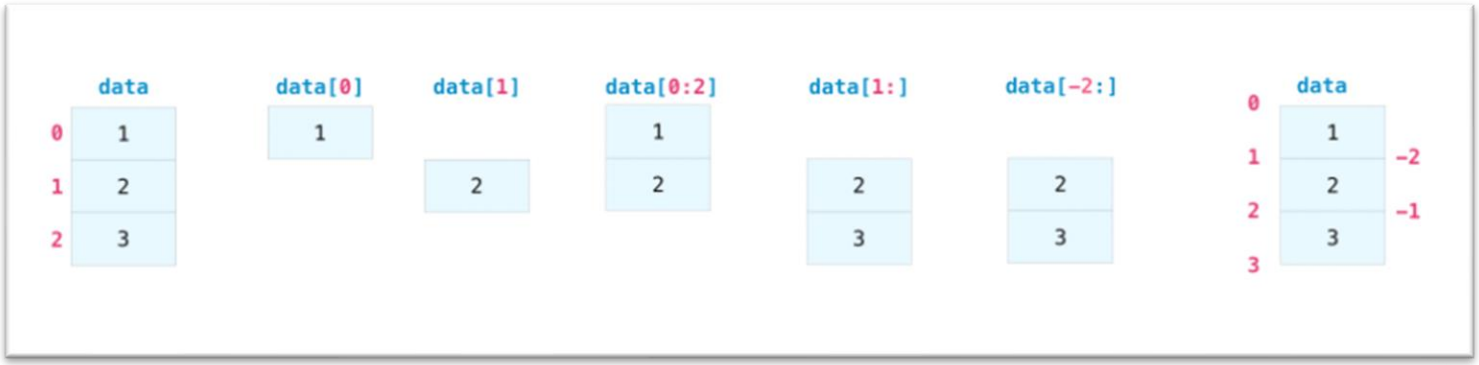
- Các phần tử trong mảng NumPy được truy xuất như với list trong Python.

```
In [25]: #Truy vấn đến hàng 1 của mảng và truy vấn đến phần tử tại vị trí 0, 1
         data3=[[1,2,3,4],[5,6,7,8]]
         arr3=np.array(data3)
         print(arr3[0])
         print(arr3[0][1])

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

Cắt (Slicing) các mảng NumPy

- Slicing các mảng là “cắt” mảng ra và lấy phần mong muốn
  - Cú pháp: array[start:stop:step] (step mặc định là 1)

```
>>> data = np.array([1, 2, 3])

>>> data[1]
2
>>> data[0:2]
array([1, 2])
>>> data[1:]
array([2, 3])
>>> data[-2:]
array([2, 3])
```



Cắt (Slicing) các mảng NumPy

- Việc slice tạo ra mảng mới với các phần tử được tham chiếu đến mảng cũ.

```
In [26]: arr9=np.array([13,4,5,6,6,5,1,10,9,10])
         arr9
Out[26]: array([13,  4,  5,  6,  6,  5,  1, 10,  9, 10])

In [28]: arr10=arr9[4:7]
         arr10
Out[28]: array([6, 5, 1])

In [29]: arr10[0] #Phần tử 0 của mảng arr10 chính là phần tử 4 của arr9
Out[29]: 6

In [30]: arr10[0]=3
         arr9[4]
Out[30]: 3
```

Duyệt mảng NumPy

- Dùng vòng for để duyệt toàn bộ mảng một chiều, đối với mảng đa chiều vòng for sẽ duyệt theo chiều đầu tiên của mảng.

```
In [32]: arr9=np.array([13,4,5,6,6,5,1,10,9,10])
         for i in arr9:
             print(i)

         13
         4
         5
         6
         6
         5
         1
         10
         9
         10
```

```
In [33]: data3=[[1,2,3,4],[5,6,7,8]]
         arr3=np.array(data3)
         for i in arr3:
             print(i)

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

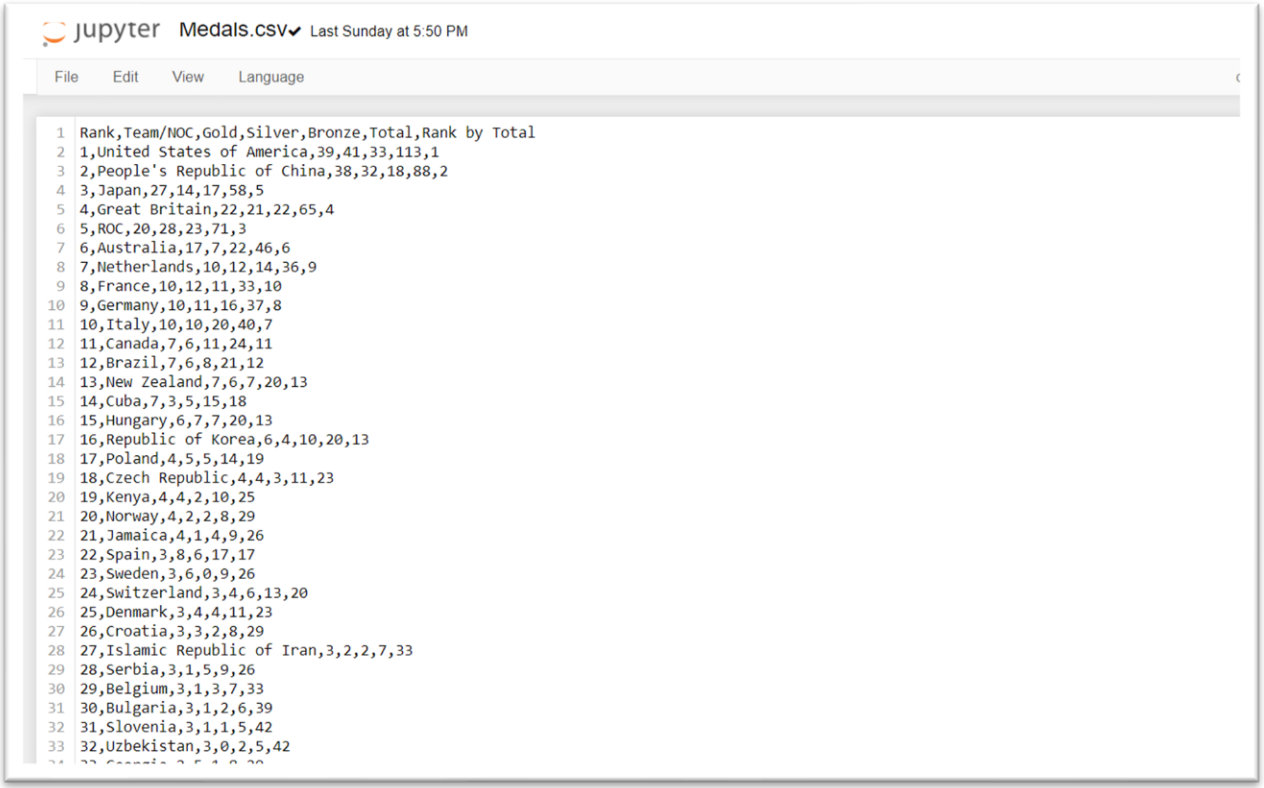
Duyệt mảng NumPy

- Đối với các mảng nhiều chiều để có thể duyệt từng phần tử của mảng, ta có thể dùng hàm for để duyệt thông qua hàm flatten có tác dụng tạo ra mảng một chiều từ mảng đa chiều đã cho.

```
In [37]: data3=[[1,2,3,4],[5,6,7,8]]
         arr3=np.array(data3)
         for i in arr3.flatten():
             print(i)

1
2
3
4
5
6
7
8
```

Tạo mảng NumPy từ file csv

- Tạo mảng NumPy lấy dữ liệu về số huy chương giành được của các đội trong đại hội Olympic 2020 từ file csv, txt bằng phương thức numpy.genfromtxt
  - Tải tệp tại đây. Sau đó đưa tệp vào folder mong muốn.



Tạo mảng NumPy từ file csv

- Sử dụng phương thức numpy.genfromtxt để tạo mảng numpy mong muốn. Những phần tử không phải là số sẽ được NumPy tự động đánh dấu bằng nan (Not a number).

```
In [7]: import numpy as np

In [8]: arr=np.genfromtxt("Medals.csv", delimiter=",",dtype=np.float64)
        arr

Out[8]: array([[ nan,  nan,  nan,  nan,  nan,  nan,  nan],
               [  1.,  nan,  39.,  41.,  33., 113.,   1.],
               [  2.,  nan,  38.,  32.,  18.,  88.,   2.],
               [  3.,  nan,  27.,  14.,  17.,  58.,   5.],
               [  4.,  nan,  22.,  21.,  22.,  65.,   4.],
               [  5.,  nan,  20.,  28.,  23.,  71.,   3.],
               [  6.,  nan,  17.,   7.,  22.,  46.,   6.],
               [  7.,  nan,  10.,  12.,  14.,  36.,   9.],
               [  8.,  nan,  10.,  12.,  11.,  33.,  10.],
               [  9.,  nan,  10.,  11.,  16.,  37.,   8.],
               [ 10.,  nan,  10.,  10.,  20.,  40.,   7.],
               [ 11.,  nan,   7.,   6.,  11.,  24.,  11.],
```

4. Các hàm tính toán trong NumPy

- Tính toán mảng trong NumPy có thể trở nên rất nhanh nhờ vào các phép toán vector hóa (vectorized operations) được thêm vào Python qua các universal functions (ufuncs).
- Việc tính toán trong NumPy có thể trở lên tiện dụng nhờ vào việc chúng ta có thể thực hiện tính toán trên các mảng y như biến trong python.

```
In [3]: x=np.array([1,2,3,4,5])
        print(x+2)
        print(x-2)
        print(x*2)
        print(x/2)

        [3 4 5 6 7]
        [-1  0  1  2  3]
        [ 2  4  6  8 10]
        [0.5 1.  1.5 2.  2.5]
```

Hàm lấy giá trị tuyệt đối

- Hàm abs() trả về mảng NumPy là giá trị tuyệt đối của các phần tử tương ứng ở mảng ban đầu. Tương ứng với hàm abs() là các hàm numpy.absolute() và numpy.abs().

```
In [5]: x=np.array([-2,-1,0,1,2])
        print(abs(x))
        print(np.absolute(x))
        print(np.abs(x))

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

Hàm lượng giác

- NumPy cung cấp các hàm lượng giác như sin(), cos(), tan(),…

```
In [2]: theta=np.array([np.pi/2,2*np.pi/2,3*np.pi/2,4*np.pi/2])
        print(np.sin(theta)) #Tính sin của các góc theta
        print(np.cos(theta)) #Tính cos của các góc theta
        print(np.tan(theta)) #Tính tan của các góc theta

        [ 1.0000000e+00  1.2246468e-16 -1.0000000e+00 -2.4492936e-16]
        [ 6.1232340e-17 -1.0000000e+00 -1.8369702e-16  1.0000000e+00]
        [ 1.63312394e+16 -1.22464680e-16  5.44374645e+15 -2.44929360e-16]
```

Hàm mũ và hàm logarithms

- NumPy cung cấp các hàm mũ và logarithm.

```
In [8]: x=[1,2,3]
        print("x= ",x)
        print("e^x= " ,np.exp(x))
        print("2^x= ", np.exp2(x))
        print("3^x= ", np.power(3,x))

        x=  [1, 2, 3]
        e^x=  [ 2.71828183  7.3890561  20.08553692]
        2^x=  [2. 4. 8.]
        3^x=  [ 3  9 27]
```

Hiệu suất của NumPy

- Với kiểu dữ liệu được xác định trong bởi đối tượng dtype NumPy có thể chuyển đổi kiểu dữ liệu nhằm tương thích với bộ nhớ nhằm tăng tốc độ tính toán lên nhiều lần.
  - So sánh tốc độ xử lý của hàm sum() của python và hàm numpy.sum() của NumPy bằng lệnh %timeit, ta thấy tốc độ của NumPy được đẩy nhanh lên nhiều lần.

```
In [9]: x=np.arange(1000)
        %timeit sum(x)
        %timeit np.sum(x)

        76.3 µs ± 1.77 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
        4.29 µs ± 131 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Một số hàm tính toán với NumPy

Function Name	NaN-safe Version	Description
np.sum	np.nansum	Compute sum of elements
np.prod	np.nanprod	Compute product of elements
np.mean	np.nanmean	Compute median of elements
np.std	np.nanstd	Compute standard deviation
np.var	np.nanvar	Compute variance
np.min	np.nanmin	Find minimum value
np.max	np.nanmax	Find maximum value
np.argmin	np.nanargmin	Find index of minimum value
np.argmax	np.nanargmax	Find index of maximum value
np.median	np.nanmedian	Compute median of elements
np.percentile	np.nanpercentile	Compute rank-based statistics of elements
np.any	N/A	Evaluate whether any elements are true
np.all	N/A	Evaluate whether all elements are true

Ví dụ

- Tính tổng số huy chương vàng mà đội của tất cả các nước nhận được trong tệp Medals.csv.
  - Bước 1: Tạo mảng NumPy từ tệp Medals.csv

```
In [21]: medals=np.genfromtxt("Medals.csv", delimiter=",",dtype=np.float64)
         medals

Out[21]: array([[ nan,  nan,  nan,  nan,  nan,  nan,  nan],
                [  1.,  nan,  39.,  41.,  33., 113.,   1.],
                [  2.,  nan,  38.,  32.,  18.,  88.,   2.],
                [  3.,  nan,  27.,  14.,  17.,  58.,   5.],
                [  4.,  nan,  22.,  21.,  22.,  65.,   4.],
                [  5.,  nan,  20.,  28.,  23.,  71.,   3.],
                [  6.,  nan,  17.,   7.,  22.,  46.,   6.],
                [  7.,  nan,  10.,  12.,  14.,  36.,   9.],
                [  8.,  nan,  10.,  12.,  11.,  33.,  10.],
                [  9.,  nan,  10.,  11.,  16.,  37.,   8.],
                [ 10.,  nan,  10.,  10.,  20.,  40.,   7.],
                [ 11.,  nan,   7.,   6.,  11.,  24.,  11.],
                [ 12.,  nan,   7.,   6.,   8.,  21.,  12.],
                [ 13.,  nan,   7.,   6.,   7.,  20.,  13.],
```

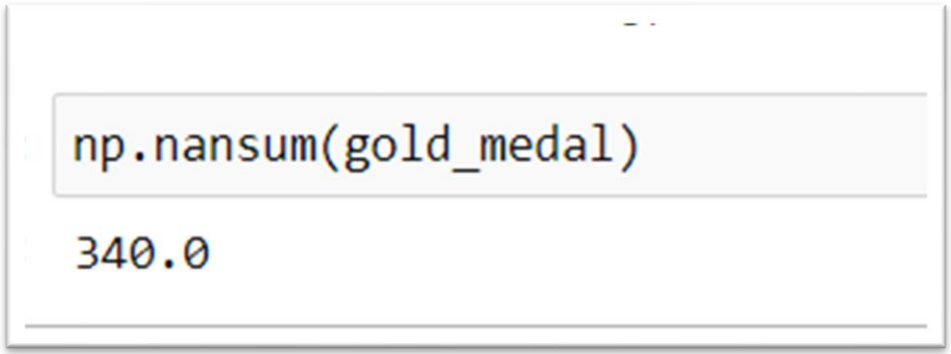
Ví dụ

- Bước 2: Lấy danh sách huy chương vàng mà các đội tuyển các nước nhận được. (cột 2)

```
In [22]: gold_medal=medals[:,2] #Lấy danh sách huy chương vàng của các nước.
         gold_medal

Out[22]: array([nan, 39., 38., 27., 22., 20., 17., 10., 10., 10., 10.,  7.,  7.,
                 7.,  7.,  6.,  6.,  4.,  4.,  4.,  4.,  4.,  3.,  3.,  3.,  3.,
                 3.,  3.,  3.,  3.,  3.,  3.,  3.,  2.,  2.,  2.,  2.,  2.,  2.,
                 2.,  2.,  2.,  2.,  2.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
                 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
                 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                 0.,  0.,  0.])
```

Ví dụ

- Bước 3: Dùng hàm numpy.nansum (tương tự như hàm numpy.sum() nhưng bỏ qua giá trị không phải là số) để tính tổng



5. Sắp xếp mảng trong NumPy (Sorting)

- Hàm sort() trong NumPy được dùng để sắp xếp các phần tử trong NumPy với tốc độ nhanh hơn rất nhiều so với các hàm sort có sẵn trong Python.

```
In [41]: #Tạo mảng ngẫu nhiên có 5 phần tử rồi sắp xếp chúng
         x=np.random.rand(5)
         print(x)
         x.sort()
         print(x)

         [0.9260806  0.68023844 0.95741601 0.28369639 0.96083881]
         [0.28369639 0.68023844 0.9260806  0.95741601 0.96083881]
```

Hàm numpy.sort()

- Dùng hàm numpy.sort() để lấy một mảng đã được sắp xếp mà không làm thay đổi mảng ban đầu.

```
In [44]: arr=np.array([1,26,85,34,23,50,45])
         sorted_arr=np.sort(arr)
         print(arr)
         print(sorted_arr)

         [ 1 26 85 34 23 50 45]
         [ 1 23 26 34 45 50 85]
```

Sắp xếp mảng với mảng nhiều chiều

- Điều chỉnh tham số axis trong hàm sort của mảng nhiều chiều để hàm sort có thể sắp xếp theo chiều đó.

6. Mảng cấu trúc trong NumPy

- Mảng cấu trúc trong NumPy là mảng ndarray kết hợp của những kiểu dữ liệu dtypes đơn giản được tổ chức dưới dạng các trường được đặt tên. (Giống như các bảng)

```
>>> x = np.array([('Rex', 9, 81.0), ('Fido', 3, 27.0)],
...              dtype=[('name', 'U10'), ('age', 'i4'), ('weight', 'f4')])
>>> x
array([('Rex', 9, 81.), ('Fido', 3, 27.)],
      dtype=[('name', 'U10'), ('age', '<i4'), ('weight', '<f4')])
```

Cách tạo mảng cấu trúc

- Ta xác định dtype bằng cách thiết đặt tên trường với kiểu dữ liệu tương ứng.

```
In [59]: medal=np.array([("Mỹ",39),("Trung Quốc",38)],dtype=[("Quốc gia",np.unicode_),("Số huy chương vàng",np.int64)])
         print(medal)

         [('', 39) ('', 38)]
```

7. Tính toán nâng cao với Scipy

- SciPy, viết tắt của scientific python, là một thư viện bao gồm tập hợp thuật toán toán học và phương thức tiện dụng phục vụ trong nghiên cứu khoa học được xây dựng trên NumPy.

Giải hệ phương trình tuyến tính bằng SciPy

- Giải phương trình tuyến tính sau:

$$
\begin{aligned}
x + 3y + 5z &= 10 \\
2x + 5y + z &= 8 \\
2x + 3y + 8z &= 3
\end{aligned}
$$

Giải hệ phương trình tuyến tính bằng SciPy

- Chúng ta có thể giải phương trình trên bằng cách tìm ma trận nghịch đảo.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 10 \\ 8 \\ 3 \end{bmatrix} = \frac{1}{25} \begin{bmatrix} -232 \\ 129 \\ 19 \end{bmatrix} = \begin{bmatrix} -9.28 \\ 5.16 \\ 0.76 \end{bmatrix}.$$

Giải hệ phương trình tuyến tính bằng SciPy

- Bước 1: Chuyển các ma trận về dạng mảng array.

```
In [46]: A=np.array([[1,3,5],[2,5,1],[2,3,8]])
         b=np.array([10,8,3])
         print(A)
         print(b)

         [[1 3 5]
          [2 5 1]
          [2 3 8]]
         [10  8  3]
```

Giải hệ phương trình tuyến tính bằng SciPy

- Bước 2: import thư viện linalg (linear algebra) của scipy

```
In [53]: from scipy import linalg
```

Giải hệ phương trình tuyến tính bằng SciPy

- Bước 3: Sử dụng hàm linalg.solve() để giải phương trình trên.

```
In [55]: x=linalg.solve(A,b)
         x
Out[55]: array([-9.28,  5.16,  0.76])
```

Hỏi đáp

Q&A