Python NumPy

- Tạo mảng NumPy đơn và đa chiều
- Sử dụng hiệu quả các hàm và phương thức tích hợp của NumPy
- Thực hiện các phép toán trên mảng
- Trích xuất các phần tử từ mảng bằng cách cắt và lập chỉ mục
- Chọn các phần tử của mảng có điều kiện.

Giới thiệu về NumPy

- Viết tắt của Numerical Python
- Một thư viện Python thực hiện các phép tính số
- Rất nhanh
- Liên quan đến ma trận và vectơ

NumPy dùng để làm gì

- NumPy là một thư viện quan trọng cho:
  - Khoa học dữ liệu
  - Học máy
  - Xử lý tín hiệu và hình ảnh
  - Máy tính khoa học và kỹ thuật

Cài đặt NumPy

- Cài đặt: pip install numpy
- Nhập NumPy: import numpy
- Nhập NumPy dưới dạng np: import numpy as np

Tạo mảng NumPy

- **Tạo mảng 1 chiều**

```
import numpy as np

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

print(type(a))
print(a)
```

```
<class 'numpy.ndarray'>
[1 2 3]
```

Tạo mảng NumPy

- **Lấy kích thước của một mảng**

```
import numpy as np

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

print(a.ndim)
```

```
1
```

Tạo mảng NumPy

- **Lấy kiểu dữ liệu của các phần tử mảng**

```
import numpy as np

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

print(a.dtype)
```

```
int32
```

Tạo mảng NumPy

- **Tạo mảng 2 chiều**

```
import numpy as np

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

print(b)
print(b.ndim)
```

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

2
```

Tạo mảng NumPy

- **Tạo mảng 3 chiều**

```
import numpy as np

c = np.array(
    [
        [
            [1, 2, 3],
            [4, 5, 6]
        ],
        [
            [7, 8, 9],
            [10, 11, 12]
        ],
    ]
)

print(c.ndim)
```

```
3
```

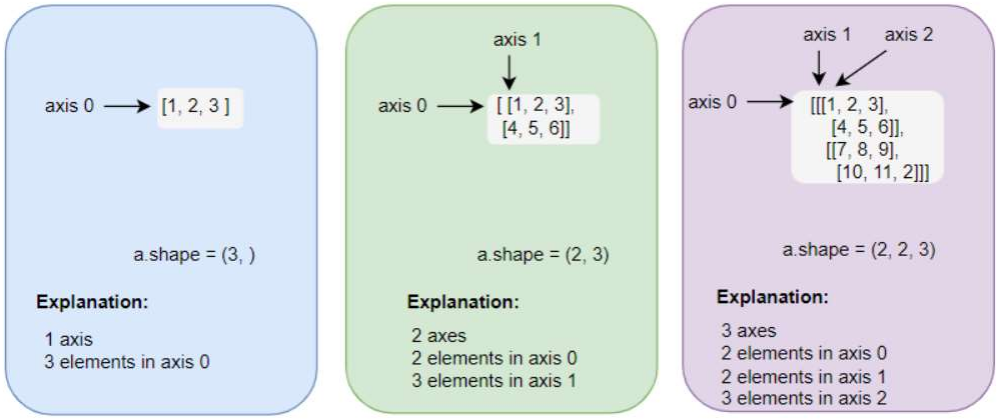
Nhận hình dạng của mảng

```
import numpy as np


a = np.array([1, 2, 3])
print(a.shape)  # (3,)

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

```
c = np.array(
    [
        [
            [1, 2, 3],
            [4, 5, 6]
        ],
        [
            [7, 8, 9],
            [10, 11, 12]
        ],
    ]
)
print(c.shape)  # (2, 2, 3)
```

```
(3,)
(2, 3)
(2, 2, 3)
```

Nhận hình dạng của mảng



NumPy zeros()

```
import numpy as np

a = np.zeros((2, 3))
print(a)
```

Đầu ra:

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

NumPy zeros()

```
import numpy as np

a = np.zeros((2, 3))
print(a.dtype)
```

Đầu ra:

```
float64
```

```
import numpy as np

a = np.zeros((2, 3), dtype=np.int32)
print(a)
print(a.dtype)
```

Đầu ra:

```
[[0 0 0]
 [0 0 0]]
int32
```

NumPy ones()

```
import numpy as np

a = np.ones((2, 3, 2))
print(a)
```

Đầu ra:

```
[[[1. 1.]
  [1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]
  [1. 1.]]]
```

NumPy ones()

```
import numpy as np

a = np.ones((2, 3, 2))
print(a.dtype)
```

Đầu ra:

```
float64
```

```
import numpy as np

a = np.ones((2, 3, 4), dtype=np.int32)
print(a)
print(a.dtype)
```

Đầu ra:

```
[[[1 1]
  [1 1]
  [1 1]]

 [[1 1]
  [1 1]
  [1 1]]]
int32
```

NumPy arange()

- Tạo một mảng numpy mới với các số cách đều nhau giữa start (bao gồm) và stop (không bao gồm) với điều kiện step:

```
import numpy as np

a = np.arange(1, 10, 2)

print(a)
```

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

NumPy arange()

- Tạo một mảng các số float:

```
import numpy as np

a = np.arange(1.0, 10.0, 2)

print(a)
```

Đầu ra:

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

```
import numpy as np

a = np.arange(1, 10, 2, dtype=np.float64)

print(a)
```

Đầu ra:

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

NumPy linspace()

- Tạo một mảng numpy mới với các số cách đều nhau theo một khoảng xác định

```
import numpy as np

a = np.linspace(1, 2, 5)

print(a)
```

Đầu ra:

```
[1.   1.25 1.5  1.75 2.  ]
```

NumPy linspace()

- Không muốn bao gồm giá trị stop

```
import numpy as np

a = np.linspace(1, 2, 5, endpoint=False)

print(a)
```

Đầu ra:

```
[1.  1.2 1.4 1.6 1.8]
```

Lập chỉ mục mảng NumPy

Lập chỉ mục mảng NumPy trên mảng 1 chiều

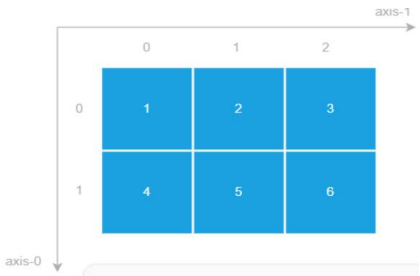

```
import numpy as np

a = np.arange(0, 5)
print(a)

print(a[0])
print(a[1])
print(a[-1])
```

Đầu ra:

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

Trong ví dụ này:
- a[0] trả về phần tử đầu tiên (0)
- a[1] trả về phần tử thứ hai (1)
- a[-1] trả về phần tử cuối cùng (4)
- a[-2] trả về phần tử cuối cùng thứ hai (3)

Lập chỉ mục mảng NumPy trên mảng 2 chiều



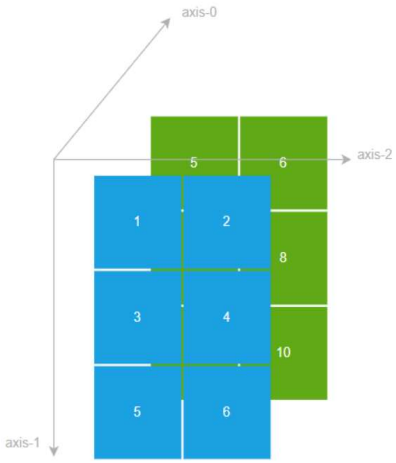

```
import numpy as np

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

print(a.shape)
```

```
print(a[0])  # [1 2 3]
print(a[1])  # [4 5 6]

print(a[0, 0])  # 1
print(a[1, 0])  # 4
print(a[0, 2])  # 3
print(a[1, 2])  # 6
print(a[0, -1])  # 3
print(a[1, -1])  # 6
```

Đầu ra:

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

Lập chỉ mục mảng NumPy trên mảng 3 chiều




```
import numpy as np

a = np.array([
    [[1, 2], [3, 4], [5, 6]],
    [[5, 6], [7, 8], [9, 10]],
])

print(a.shape)
```

Đầu ra:

```
(2, 3, 2)
```

Lập chỉ mục mảng NumPy trên mảng 3 chiều

```
import numpy as np

a = np.array([
    [[1, 2], [3, 4], [5, 6]],
    [[5, 6], [7, 8], [9, 10]],
])

print(a[0, 0, 1])  # 2
```

Biểu thức sau trả về 2:

```
a[0,0,1]
```

Số 0 đầu tiên chọn phần tử đầu tiên của trục đầu tiên nên nó trả về:

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

Số 0 thứ hai chọn phần tử đầu tiên của trục thứ hai nên nó trả về:

```
[1, 2]
```

Số thứ ba (1) chọn phần tử thứ hai của trục thứ ba trả về 2.

Phân chia mảng Numpy

Phân chia mảng Numpy trên mảng một chiều

Biểu thức cắt lát	Nghĩa
`a[m:n]`	Chọn các phần tử có chỉ số bắt đầu từ m và kết thúc tại n-1.
`a[:]`hoặc`a[0:-1]`	Chọn tất cả các phần tử trong một trục nhất định
`a[:n]`	Chọn các phần tử bắt đầu bằng chỉ số 0 và lên đến phần tử có chỉ số n-1
`a[m:]`	Chọn các phần tử bắt đầu bằng chỉ số m và lên đến phần tử cuối cùng
`a[m:-1]`	Chọn các phần tử bắt đầu bằng chỉ số m và lên đến phần tử cuối cùng
`a[m:n:k]`	Chọn các phần tử có chỉ số từ m đến n (không bao gồm), với gia số k
`a[::-1]`	Chọn tất cả các phần tử theo thứ tự ngược lại

Phân chia mảng Numpy trên mảng một chiều

```
import numpy as np

a = np.arange(0, 10)

print('a=', a)
print('a[2:5]=', a[2:5])
print('a[:]=', a[:])
print('a[0:-1]=', a[0:-1])
print('a[0:6]=', a[0:6])
print('a[7:]=', a[7:])
print('a[5:-1]=', a[5:-1])
print('a[0:5:2]=', a[0:5:2])
print('a[::-1]=', a[::-1])
```

Phân chia mảng Numpy trên mảng một chiều

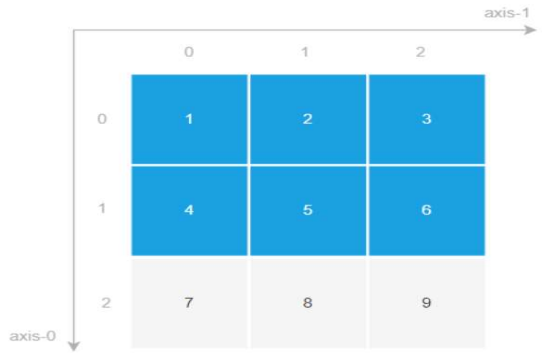
```
import numpy as np

a = np.arange(0, 10)

print('a=', a)
print('a[2:5]=', a[2:5])
print('a[:]=', a[:])
print('a[0:-1]=', a[0:-1])
print('a[0:6]=', a[0:6])
print('a[7:]=', a[7:])
print('a[5:-1]=', a[5:-1])
print('a[0:5:2]=', a[0:5:2])
print('a[::-1]=', a[::-1])
```

Đầu ra:

```
a= [0 1 2 3 4 5 6 7 8 9]
a[2:5]= [2 3 4]
a[:]= [0 1 2 3 4 5 6 7 8 9]
a[0:-1]= [0 1 2 3 4 5 6 7 8]
a[0:6]= [0 1 2 3 4 5]
a[7:]= [7 8 9]
a[5:-1]= [5 6 7 8]
a[0:5:2]= [0 2 4]
a[::-1]= [9 8 7 6 5 4 3 2 1 0]
```

Phân chia mảng Numpy trên mảng đa chiều

```
import numpy as np

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


print(a[0:2, :])
```



Đầu ra:

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

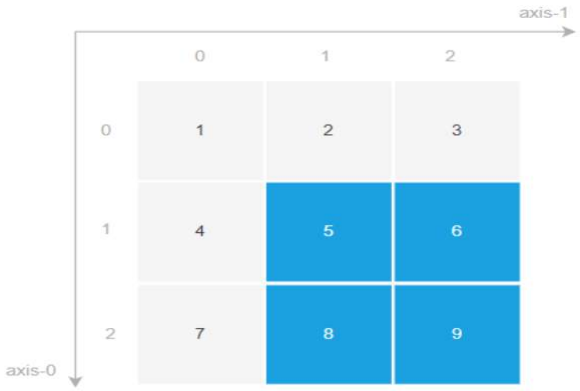
Phân chia mảng Numpy trên mảng đa chiều

```
import numpy as np

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

print(a[1:, 1:])
```

Đầu ra: ?????

Phân chia mảng Numpy trên mảng đa chiều

```
import numpy as np

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

print(a[1:, 1:])
```



Đầu ra:

```
[[5 6]
 [8 9]]
```

Phân chia mảng Numpy trên mảng đa chiều

Đầu tiên, 1:chọn các phần tử bắt đầu từ chỉ số 1 đến phần tử cuối cùng của trục đầu tiên (hoặc hàng), trả về:

```
[[4 5 6]
[7 8 9]]
```

Thứ hai, `1:` chọn các phần tử bắt đầu từ chỉ số 1 đến các phần tử cuối cùng của trục thứ hai (hoặc cột), trả về:

```
[[5 6]
 [8 9]]
```

Lập chỉ mục Boolean

- Numpy cho phép bạn sử dụng một mảng các giá trị boolean làm chỉ mục của một mảng khác.

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([True, True, False])
c = a[b]
print(c)
```

Đầu ra:

```
[1 2]
```

- Nó hoạt động như thế nào?

Lập chỉ mục Boolean

- Đầu tiên, tạo một mảng numpy mới bao gồm ba số từ 1 đến 3:
  - a = np.array([1, 2, 3])
- Thứ hai, tạo một mảng numpy khác với ba giá trị boolean True, True, và False:
  - b = np.array([True, True, False])
- Thứ ba, sử dụng mảng boolean b làm chỉ mục của mảng a và gán các phần tử đã chọn cho biến c:
  - c = a[b]

Lập chỉ mục Boolean

- Thông thường, bạn sẽ sử dụng chỉ mục boolean để lọc một mảng.
- Ví dụ:

```
import numpy as np

a = np.arange(1, 10)
b = a > 5
print(b)

c = a[b]
print(c)
```

Đầu ra: ?????

Lập chỉ mục Boolean

- Thông thường, bạn sẽ sử dụng chỉ mục boolean để lọc một mảng.
- Ví dụ:

```
import numpy as np

a = np.arange(1, 10)
b = a > 5
print(b)

c = a[b]
print(c)
```

Đầu ra:

```
[False False False False False  True  True  True  True]
[6 7 8 9]
```

Nó hoạt động như thế nào?????

Các hàm tổng hợp

NumPy sum()

- Hàm tổng hợp lấy một mảng và trả về tổng của tất cả các phần tử?

NumPy sum()

- Ví dụ sau đây sử dụng hàm sum() để tính tổng tất cả các phần tử của mảng 1 chiều:

```
import numpy as np

a = np.array([1, 2, 3])
total = np.sum(a)
print(total)
```

Đầu ra:

```
6
```

NumPy sum()

- Ví dụ sau đây sử dụng hàm sum() để tính tổng tất cả các phần tử của mảng 2 chiều:

```
import numpy as np

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

total = np.sum(a)
print(total)
```

Đầu ra:

```
21
```

NumPy sum()

- Hàm sum() cũng chấp nhận đối số trục cho phép bạn trả về tổng các phần tử của một trục. Ví dụ::

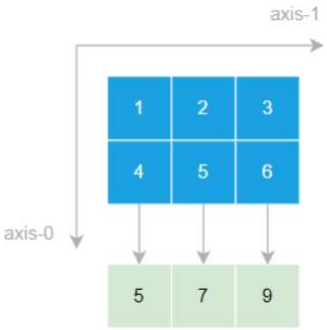
```
import numpy as np

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

total = np.sum(a, axis=0)
print(total)
```

Đầu ra?????

NumPy sum()

- Hàm sum() cũng chấp nhận đối số trục cho phép bạn trả về tổng các phần tử của một trục. Ví dụ:

```
import numpy as np

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

total = np.sum(a, axis=0)
print(total)
```



Đầu ra:

```
[5 7 9]
```

NumPy sum()

- Tương tự như vậy, bạn có thể tính tổng các phần tử trên trục-1 như thế này:

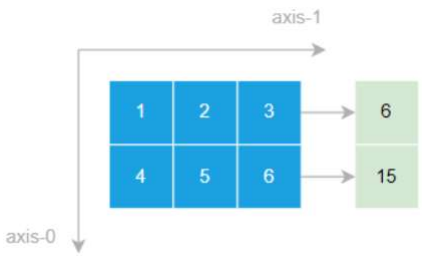
```
import numpy as np

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

total = np.sum(a, axis=1)
print(total)
```

Đầu ra?????

NumPy sum()

- Tương tự như vậy, bạn có thể tính tổng các phần tử trên trục-1 như thế này:

```
import numpy as np

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

total = np.sum(a, axis=1)
print(total)
```




```
[ 6 15]
```

NumPy mean()

- Hàm này mean()trả về giá trị trung bình của các phần tử trong một mảng.

NumPy mean()

- Hàm này mean() trả về giá trị trung bình của các phần tử trong một mảng.

NumPy mean()

- Sử dụng hàm NumPy mean() trên ví dụ mảng 1 chiều

```
import numpy as np

a = np.array([1, 2, 3])
average = np.mean(a)
print(average)
```

Đầu ra ????

NumPy mean()

- Sử dụng hàm NumPy mean() trên ví dụ mảng 1 chiều

```
import numpy as np

a = np.array([1, 2, 3])
average = np.mean(a)
print(average)
```

Đầu ra:

```
2.0
```

NumPy mean()

- Sử dụng hàm NumPy mean() trên ví dụ mảng 2 chiều

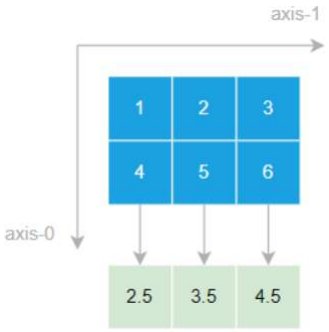
```
import numpy as np

a = np.array([
    [1, 2, 3],
    [4, 5, 6]
])
average = np.mean(a, axis=0)
print(average)
```

Đầu ra ????

NumPy mean()

- Sử dụng hàm NumPy mean() trên ví dụ mảng 2 chiều

```
import numpy as np

a = np.array([
    [1, 2, 3],
    [4, 5, 6]
])
average = np.mean(a, axis=0)
print(average)
```



Đầu ra:

```
[2.5 3.5 4.5]
```

NumPy var()

- Ví dụ: Hãy tính tính phương sai của ba số 1, 2 và 3.

NumPy var()

- Ví dụ: Hãy tính tính phương sai của ba số 1, 2 và 3.
- Các bước:
  - Đầu tiên, tính giá trị trung bình (hoặc giá trị trung bình cộng): (1+2+3) / 3 = 2,0
  - Thứ hai, tính bình phương hiệu số của mỗi số với giá trị trung bình: (1-2) 2 + (2-2) 2 + (3-2) 2 = 2
  - Thứ ba, tính trung bình cộng của các bình phương chênh lệch này: 2/3=0.667

NumPy var()

- Sử dụng hàm var() để tính toán phương sai của các phần tử trong một mảng.

NumPy var()

```
import numpy as np

a = np.array([1, 2, 3])
result = np.var(a)
print(round(result,3))
```

Đầu ra:

```
0.667
```

NumPy std()

- Sử dụng hàm numpy std() để tính độ lệch chuẩn

NumPy std()

- Ví dụ: Giả sử bạn có danh sách số đo chiều cao của các cây. Cột đầu tiên hiển thị tên cây và cột thứ hai hiển thị số đo chiều cao tương ứng tính bằng cm:

Tên cây	C.cao	Tên cây	C.cao
Lim	591	Sầu riêng	176
Táu	239	Mắc ca	175
Chà là	210	Ca cao	170
Sến	207	Đa	170
Mít	201	Măng cầu	169
Nhãn	182	Măng cụt	168
Cao su	176		

NumPy std()

- Sử dụng hàm std() để tính độ lệch chuẩn của số đo chiều cao các cây:

```
import numpy as np


diameters = np.array([591, 239, 210, 207, 201, 182,
                      176, 176, 175, 170, 170, 169, 168, ])
result = np.std(diameters)
print(round(result, 1))
```

Đầu ra:

```
109.6
```

NumPy prod()

- Sử dụng hàm prod() để tính tích của các số trong mảng:

NumPy prod()

- Sử dụng hàm numpy prod() với ví dụ mảng 1 chiều

```
import numpy as np

a = np.arange(1, 5)
result = np.prod(a)

print(a)
print(f'result={result}')
```

Đầu ra: ?????

NumPy prod()

- Sử dụng hàm numpy prod() với ví dụ mảng 1 chiều

```
import numpy as np

a = np.arange(1, 5)
result = np.prod(a)

print(a)
print(f'result={result}')
```

Đầu ra:

```
[1 2 3 4]
result=24
```

NumPy prod()

- Sử dụng hàm numpy prod() với các ví dụ về mảng đa chiều

```
import numpy as np

result = np.prod([
    [1, 2],
    [3, 4]
])

print(f'result={result}')
```

Đầu ra: ?????

NumPy prod()

- Sử dụng hàm numpy prod() với các ví dụ về mảng đa chiều

```
import numpy as np

result = np.prod([
    [1, 2],
    [3, 4]
])

print(f'result={result}')
```

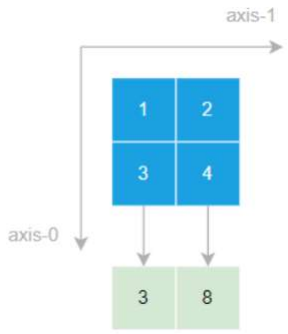
Đầu ra:

```
result=24
```

NumPy prod()

- Để tính tích các số của một trục, bạn có thể chỉ định đối số trục

```
import numpy as np

result = np.prod([
    [1, 2],
    [3, 4]
], axis=0)

print(f'result={result}')
```



Đầu ra:

```
result=[3 8]
```

NumPy prod()

- Để chọn số cụ thể để đưa vào hàm prod(), bạn sử dụng đối số where.

```
import numpy as np

a = np.array([np.nan, 3, 4])
result = np.prod(a, where=[False, True, True])
print(result)
```

Đầu ra:

```
12.0
```

NumPy amin()

- Hàm numpy amin() để tìm phần tử nhỏ nhất trong một mảng.

NumPy amin()

- Sử dụng hàm numpy amin() trên ví dụ mảng 1 chiều

```
import numpy as np


a = np.array([1, 2, 3])
min = np.amin(a)
print(min)
```

Đầu ra:

```
1
```

NumPy amin()

- Sử dụng hàm numpy amin() trên các ví dụ về mảng đa chiều

```
import numpy as np

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

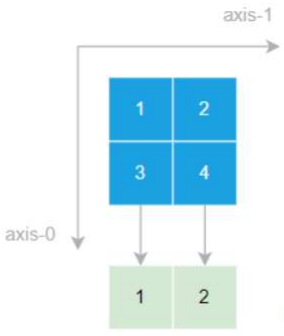
Đầu ra:

```
1
```

NumPy amin()

- Muốn tìm giá trị nhỏ nhất trên mỗi trục, bạn có thể sử dụng đối số trục.

```
import numpy as np

a = np.array([
    [1, 2],
    [3, 4]]
)
min = np.amin(a, axis=0)
print(min)
```



Đầu ra:

```
[1 2]
```

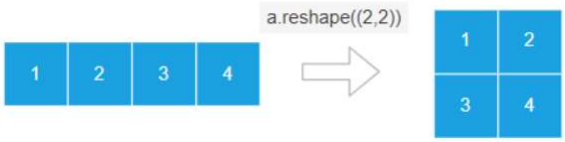
Các phép toán mảng

- reshape() – cung cấp cho mảng một hình dạng mới trong khi vẫn giữ nguyên các phần tử.
- transpose() – trả về dạng xem của một mảng với các trục được chuyển vị.
- sort() – trả về một bản sao đã được sắp xếp của một mảng.
- flatten() – trả về một bản sao của một mảng được thu gọn thành một chiều.
- ravel() – trả về một mảng phẳng liền kề.

NumPy reshape()

- Sử dụng hàm numpy reshape() với ví dụ mảng 1 chiều

```
import numpy as np

a = np.arange(1, 5)
print(a)

b = np.reshape(a, (2, 2))
print(b)
```



Đầu ra:

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

NumPy transpose()

- Sử dụng hàm numpy transpose() để đảo ngược trục của một mảng

NumPy transpose()

- Sử dụng hàm transpose() với ví dụ mảng 1 chiều

```
import numpy as np

a = np.array([1, 2, 3])
b = np.transpose(a)
print(b)
```

Đầu ra: ?????

NumPy transpose()

- Sử dụng hàm transpose() với ví dụ mảng 1 chiều

```
import numpy as np

a = np.array([1, 2, 3])
b = np.transpose(a)
print(b)
```

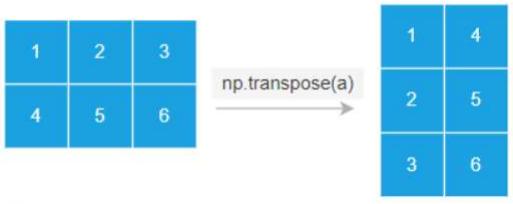
Đầu ra:

```
[1 2 3]
```

NumPy transpose()

- Sử dụng hàm transpose() với ví dụ mảng 2 chiều

```
import numpy as np

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

b = np.transpose(a)
print(b)
```



Đầu ra:

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

NumPy sort()

- Sử dụng hàm numpy sort()để sắp xếp các phần tử của một mảng.
- Cú pháp: np.sort(a, axis=- 1, kind=None, order=None)
  - a là một mảng numpy cần được sắp xếp
  - axis chỉ định trục mà các phần tử sẽ được sắp xếp
    - None: hàm sẽ làm phẳng mảng trước khi sắp xếp
    - -1: sắp xếp các phần tử theo trục cuối cùng
  - kind chỉ định thuật toán sắp xếp có thể là 'quicksort', 'mergesort', 'heapsort' và 'stable‘
  - order chỉ định trường nào sẽ so sánh trước, sau

NumPy sort()

- Sử dụng hàm sort() để sắp xếp mảng 1 chiều

```
import numpy as np

a = np.array([2, 3, 1])
b = np.sort(a)
print(b)
```

Đầu ra:

```
[1 2 3]
```

```
import numpy as np

a = np.array([2, 3, 1])
b = np.sort(a)[::-1]
print(b)
```

Đầu ra:

```
[3 2 1]
```

NumPy sort()

- Sử dụng hàm numpy sort() để sắp xếp một ví dụ về mảng 2 chiều

```
import numpy as np

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

b = np.sort(a)
print(b)
```

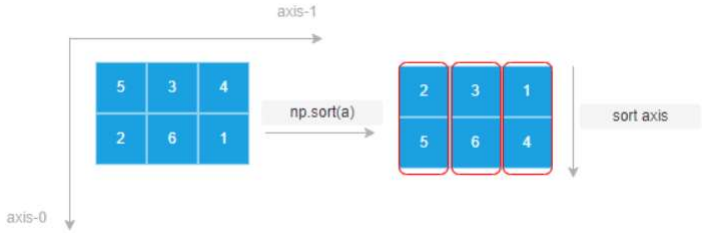
Đầu ra:

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

NumPy sort()

- Sử dụng sort()hàm để sắp xếp các phần tử trên trục 0:

```
import numpy as np

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

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



Đầu ra:

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

NumPy flatten()

- Sử dụng hàm flatten() với mảng đa chiều

```
import numpy as np

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

Đầu ra:

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

NumPy flatten()

- Sử dụng flatten() để làm phẳng một mảng bằng cách sử dụng thứ tự cột chính

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = a.flatten(order='F')

print(b)
```

Đầu ra:

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

NumPy ravel()

- Sử dụng hàm NumPy ravel() để làm phẳng một mảng

```
import numpy as np

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

print(b)
```

Đầu ra:

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

Các phép toán số học

- add() – trả về tổng của hai mảng có kích thước bằng nhau.
- minus() – trả về sự khác biệt giữa hai mảng có kích thước bằng nhau
- multiply() – trả về tích của hai mảng có kích thước bằng nha
- divide() – trả về thương của hai mảng có kích thước bằng nh

NumPy add()

- Hàm numpy add()hoặc toán tử + để cộng hai mảng có kích thước bằng nhau

```
import numpy as np

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

c = np.add(a, b)

print(c)
```

Đầu ra:

```
[3 5]
```

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])

c = a + b
print(c)
```

Đầu ra:

```
[[ 6  8]
 [10 12]]
```

Nối và tách mảng

NumPy concatenate()

- Nối hai hoặc nhiều mảng thành một mảng duy nhất.
- Cú pháp: np.concatenate((a1,a2,...),axis=0)

NumPy concatenate()

- Sử dụng hàm concatenate() để nối hai mảng 1 chiều.

```
import numpy as np

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

c = np.concatenate((a, b))
print(c)
```

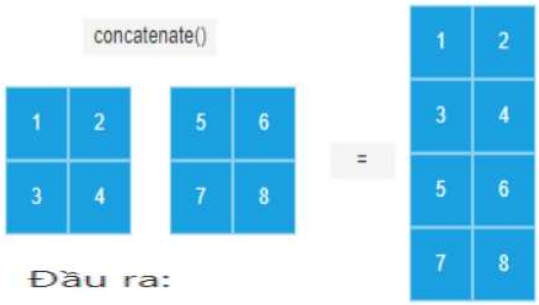
Đầu ra:

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

NumPy concatenate()

- Sử dụng hàm concatenate() để nối hai mảng 2 chiều

```
import numpy as np

a = np.array([
    [1, 2],
    [3, 4]
])
b = np.array([
    [5, 6],
    [7, 8]
])

c = np.concatenate((a, b))
print(c)
```



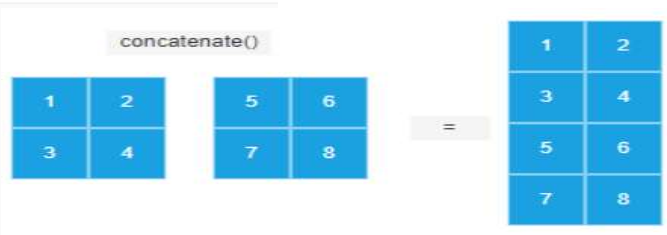
Đầu ra:

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

NumPy concatenate()

- Sử dụng hàm concatenate() để nối hai mảng 2 chiều

```
import numpy as np

a = np.array([
    [1, 2],
    [3, 4]
])
b = np.array([
    [5, 6],
    [7, 8]
])

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



Đầu ra:

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

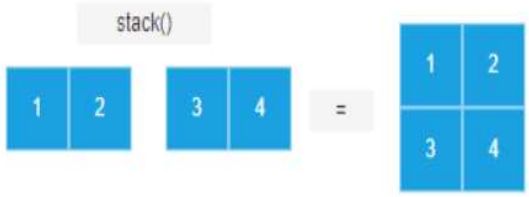
NumPy stack()

- Nối hai hoặc nhiều mảng thành một mảng duy nhất
- Cú pháp: numpy.stack((a1,a2,...),axis=0)

NumPy stack()

- Sử dụng hàm stack() để nối các mảng 1 chiều

```
import numpy as np

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

c = np.stack((a, b))
print(c)
```



Đầu ra:

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

NumPy stack()

- Sử dụng hàm stack() để nối hai mảng 1 chiều theo chiều ngang bằng cách sử dụng trục 1

```
import numpy as np

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

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

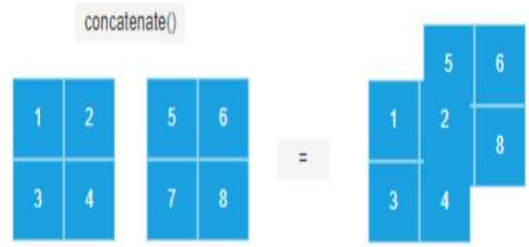
Đầu ra:

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

NumPy stack()

- Sử dụng hàm numpy stack() để nối các mảng 2 chiều

```
import numpy as np

a = np.array([
    [1, 2],
    [3, 4]
])
b = np.array([
    [5, 6],
    [7, 8]
])

c = np.stack((a, b))
print(c)
print(c.shape)
```



Đầu ra:

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

NumPy stack()

- **NumPy stack() so với concatenate()**

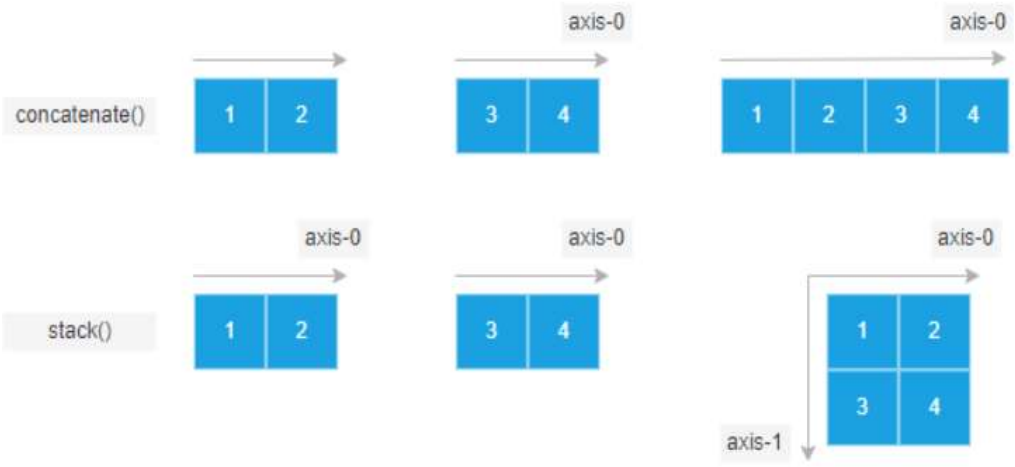
```
a = np.array([1,2])
b = np.array([3,4])

c = np.concatenate((a,b)) # return 1-D array
d = np.stack((a,b)) # return 2-D array
print(c)
print(d)
```

Đầu ra:

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

NumPy stack()

- **NumPy stack() so với concatenate()**



NumPy vstack()

- Nối các phần tử của hai hoặc nhiều mảng thành một mảng duy nhất theo chiều dọc (theo hàng)
- Cú pháp: numpy.vstack((a1,a2,...))

NumPy vstack()

- Nối các phần tử của mảng 1 chiều

```
import numpy as np

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

c = np.vstack((a, b))
print(c)
```

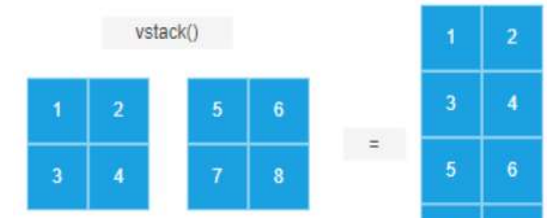
Đầu ra:

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

NumPy vstack()

- Nối các phần tử của mảng 2D

```
import numpy as np

a = np.array([
    [1, 2],
    [3, 4]
])
b = np.array([
    [5, 6],
    [7, 8]
])

c = np.vstack((a, b))
print(c)
```



Đầu ra:

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

NumPy hstack()

- Nối các phần tử của hai hoặc nhiều mảng thành một mảng duy nhất theo chiều ngang (theo cột)
- Cú pháp: numpy.hstack((a1,a2,...))

NumPy hstack()

- Nối các phần tử của mảng 1 chiều

```
import numpy as np

a = np.array([1, 2])
b = np.array([3, 4, 5])

c = np.hstack((a, b))
print(c)
```

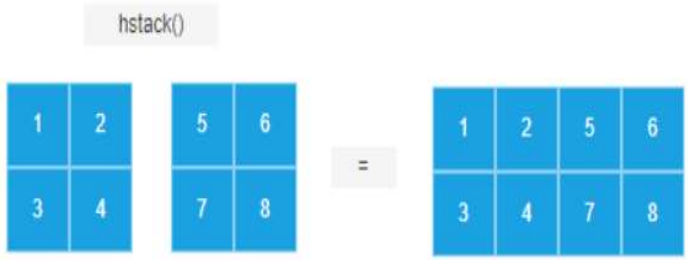
Đầu ra:

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

NumPy hstack()

- Nối các phần tử của mảng 2D

```
import numpy as np

a = np.array([
    [1, 2],
    [3, 4]
])
b = np.array([
    [5, 6],
    [7, 8]
])

c = np.hstack((a, b))
print(c)
```



Đầu ra:

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

Đại số tuyến tính Numpy

Các hàm đại số tuyến tính NumPy

`dot()`	tính tích của hai mảng
`inner()`	tính tích vô hướng của mảng
`outer()`	tính toán tích ngoài của mảng
`det()`	tính toán định thức của một ma trận
`solve()`	giải phương trình ma trận tuyến tính
`inv()`	tính toán nghịch đảo nhân của ma trận
`trace()`	tính tổng các phần tử đường chéo

Hàm NumPy dot()

```
import numpy as np

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

# sử dụng .dot() để thực hiện phép nhân mảng
multiply = np.dot(a,b)

print(multiply)
```

```
44
```

Hàm NumPy inner()

```
import numpy as np

a =  np.array([[1,3],[5,7]])
b =  np.array([[2,4],[6,8]])

# sử dụng .inner() để tính tổng các tích của các mục tương ứng trong hai mảng.
inner = np.inner(a,b)

print(inner)
```

```
[[14 30]
 [38 86]]
```

Hàm NumPy outer()

```
import numpy as np

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

# sử dụng .outer() để tính tích của tất cả các cặp phần tử có thể có của hai mảng.
outer = np.outer(a,b)

print(outer)
```

```
[[ 2  4  6]
 [ 6 12 18]
 [10 20 30]]
```

Hàm NumPy det()

```
import numpy as np

# Định nghĩa một ma trận vuông
a =  np.array([[1,3], [5,7]])

# sử dụng .linalg.det() để tính định thức của ma trận a.
det = np.linalg.det(a)

print(det)
```

Hàm NumPy solve()

- Ví dụ: Giải hệ phương trình tuyến tính
  - 2x+4y=5 và 6x+8y=6

Hàm NumPy solve()

```
import numpy as np

# Định nghĩa the coefficient matrix A
A =  np.array([[2,4], [6,8]])

# Định nghĩa the constant vector b

b = np.array([5,6])

# sử dụng .linalg.solve() để giải phương trình Ax=b.
linear = np.linalg.solve(A,b)

print(linear)
```

```
[-2.    2.25]
```

Hàm NumPy inv()

```
import numpy as np

# Định nghĩa ma trận vuông
a =  np.array([[2,4], [6,8]])

# sử dụng .linalg.inv() để tìm nghịch đảo của ma trận vuông.
inv = np.linalg.inv(a)

print(inv)
```

```
[[-1.    0.5 ]
 [ 0.75 -0.25]]
```

Hàm NumPy trace()

```
import numpy as np

# Định nghĩa ma trận vuông
a =  np.array([[2,4,3], [4,6,8], [3,5,7]])

# sử dụng .trace() để tính tổng các phần tử đường chéo chính.
trace = np.trace(a)

print(trace)
```

```
15
```