Ngôn Ngữ Lập Trình Python
Các thư viện phổ biến

Nội Dung

- Numpy
- Matplotlib

NumPy

Nội Dung

- Mảng và Ma Trận trong Python
- Giới thiệu về NumPy
- Các phép toán trên mảng
- Một số thao tác thông dụng

Mảng và Ma Trận trong Python

- **Ma trận** là cấu trúc dữ liệu hai chiều, trong đó các số được sắp xếp thành các hàng và cột.
- **Nested list** thường được dùng để trình bày ma trận trong Python. Biểu diễn như sau

```
A = [[1, 4, 5],
     [-5, 8, 9]]
```

Mảng và Ma Trận trong Python

```
A = [[1, 4, 5, 12],
[-5, 8, 9, 0],
[-6, 7, 11, 19]]
print("A =", A)
print("A[1] =", A[1]) # [-5, 8, 9, 0]
print("A[1][2] =", A[1][2]) # 9
print("A[0][-1] =", A[0][-1]) # 12
column = [];
for row in A:
        column.append(row[2])
print("Cột thứ 3 =", column)
```

Giới thiệu về NumPy

- Ngôn ngữ python có hệ thống các gói rất phong phú, hỗ trợ nhiều lĩnh vực khác nhau, từ xử lý ảnh/video/audio, xử lý văn bản, thống kê, máy học, đồ họa, web,…
- Sử dụng pip để tải các gói mới về từ internet
  - Ví dụ: open Command Prompt (cmd.exe)

```
pip install numpy
pip install pandas
pip install matplotlib
……..
```

Giới thiệu về NumPy

- NumPy (Numerical Python): là gói chuyên về xử lý dữ liệu số (nhiều chiều); gói cũng chứa các hàm đại số tuyến tính cơ bản, biến đổi fourier, sinh số ngẫu nhiên nâng cao,…
- NumPy là thư viện bổ sung của python, do không có sẵn, ta phải cài đặt:

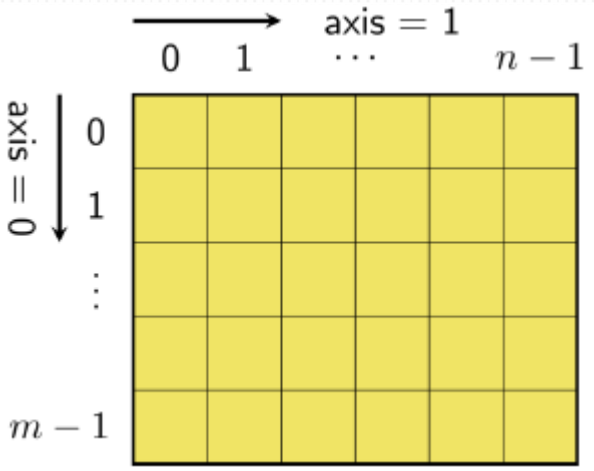
open Command Prompt(cmd.exe)

pip install numpy

- Anaconda đã tích hợp sẵn numpy khi cài đặt.
- Cách đơn giản nhất để kiểm tra xem hệ thống đã cài numpy hay không là thử import gói xem có bị báo lỗi hay không: **import numpy as np**

Giới thiệu về NumPy

- Đối tượng chính của NumPy là các mảng đa chiều đồng nhất (homogeneous multidimention array)
  - Kiểu dữ liệu phần tử con trong mảng phải giống nhau
  - Mảng có thể một chiều hoặc nhiều chiều
  - Các chiều (axis) được đánh thứ tự từ 0 trở đi
  - Kiểu **ndarray** là lớp chính xử lý dữ liệu mảng nhiều chiều
  - Rất nhiều hàm và phươngthức xử lý ma trận



Tạo mảng và truy cập

```
import numpy as np
a = np.array([1, 2, 3]) # tạo mảng 1 chiều
print(type(a)) # in "<class 'numpy.ndarray'>"
print(a.shape) # in "(3,)"
print(a[0], a[1], a[2]) # in "1 2 3"
a[0] = 5
print(a) # in "[5, 2, 3]"
b = np.array([[1, 2, 3],[4, 5, 6]]) # tạo mảng 2 chiều
print(b.shape) # in "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0]) # in "1 2 4"
print(np.diag([1, 3, 4])) #
```

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

Tạo mảng và truy cập

```
import numpy as np
x = np.arange(3) # mảng [0 1 2]
print(x)
a = np.zeros((2, 2)) # mảng 2x2 toàn số 0
print(a)
b = np.ones((1, 2)) # mảng 1x2 toàn số 1
print(b)
c = np.full((3, 2, 2), 9) # mảng 3x2x2 toàn số 9
print(c)
d = np.eye(2) # ma trận đơn vị 2x2
print(d)
e = np.random.rand(3, 2) # mảng 3x2 ngẫu nhiên [0,1)
print(e)
# mảng 2x3 điền các số từ 1 đến 6, kiểu số nguyên 32 bit
x = np.array([[1, 2, 3], [4, 5, 6]], np.int32)
print(x.ndim, x.size)
print(x.shape) # in "(2, 3)"
print(x.dtype) # in "dtype('int32')"
```

```
[0 1 2]
[[0. 0.]
 [0. 0.]]
[[1. 1.]]
[[[9 9]
  [9 9]]

 [[9 9]
  [9 9]]

 [[9 9]
  [9 9]]]
[[1. 0.]
 [0. 1.]]
[[0.7332557  0.64747164]
 [0.77964332 0.27683164]
 [0.55152677 0.84255606]]
2 6
(2, 3)
int32
```

Tạo mảng và truy cập

```
import numpy as np
A = np.arange(4)
print('A =', A)
B = np.arange(12).reshape(2, 6)
print('B =', B)
# Output:
#A = [0 1 2 3]
#B = [[ 0 1 2 3 4 5]
#      [ 6 7 8 9 10 11]]
```

Truy cập theo chỉ số (slicing)

```
import numpy as np
# mảng 3x4
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(a)
# mảng 2x2 trích xuất từ a, dòng 0+1, cột 1+2
b = a[:2, 1:3]
print(b)
# chú ý: mảng của numpy tham chiếu chứ không copy dữ liệu
print(a[0, 1]) # in "2"
b[0, 0] = 77 # b[0, 0] cũng là a[0, 1]  !!! alias
print(a[0, 1]) # in "77"

a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
c = a[:2, 1:3].copy()
c[0, 0] = 77 # b[0, 0] cũng là a[0, 1]
print(a[0, 1]) # in "2"   !!! copy
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[[2 3]
 [6 7]]
2
77
2
```

Truy cập theo chỉ số (slicing)

```
row_r1 = a[1, :] # mảng 1 chiều độ dài 4
row_r2 = a[1:2, :] # mảng 2 chiều 1x4
print(row_r1, row_r1.shape) # in ra "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape) # in ra "[[5 6 7 8]] (1, 4)"
col_r1 = a[:, 1] # mảng 1 chiều độ dài 3
col_r2 = a[:, 1:2] # mảng 2 chiều 3x1
print(col_r1, col_r1.shape) # in ra "[ 2 6 10] (3,)"
print(col_r2, col_r2.shape) # in ra "[[ 2]
                           #        [ 6]
                           #        [10]] (3, 1)"
```

```
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
[ 2  6 10] (3,)
[[ 2]
 [ 6]
 [10]] (3, 1)
```

Các phép toán trên mảng

```
import numpy as np
x = np.array([[1, 2], [3, 4]], dtype=np.float64)
y = np.array([[5, 6], [7, 8]], dtype=np.float64)
print(x + y) # print(np.add(x, y)), xử lý khác list
print(x -y) # print(np.subtract(x, y))
print(x * y) # print(np.multiply(x, y))
print(x / y) # print(np.divide(x, y))
print(np.sqrt(x)) # khai căn tất cả các phần tử
print(2**x) # tính 2 mũ các phần tử trong x
# chú ý: phép nhân/chia thực hiện theo cặp phần tử của x và y
```

```
[[ 6.  8.]
 [10. 12.]]
[[-4. -4.]
 [-4. -4.]]
[[ 5. 12.]
 [21. 32.]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[1.         1.41421356]
 [1.73205081 2.        ]]
[[ 2.  4.]
 [ 8. 16.]]
```

Các phép toán trên mảng

- Nhân ma trận (dot) và nghịch đảo

```
import numpy as np
x = np.array([[1, 2],[3, 4]])
y = np.array([[5, 6],[7, 8]])
v = np.array([1, 0])
w = np.array([0, 1])
print(v.dot(w)) # tương tự print(np.dot(v, w))
print(x.dot(v)) # tương tự print(np.dot(x, v))
print(y.dot(w)) # tương tự print(np.dot(y, w))
print(x.dot(y)) # tương tự print(np.dot(x, y))
print(np.linalg.inv(x)) # tính và in nghịch đảo của x
```

```
0
[1 3]
[6 8]
[[19 22]
 [43 50]]
[[-2.   1. ]
 [ 1.5 -0.5]]
```

Các phép toán trên mảng

- Ma trận chuyển vị: x**.T** hoặc x**.transpose()**

```
import numpy as np
x = np.array([[1, 2], [3, 4]])
print(x) # in ra "[[1 2]
         #          [3 4]]"
print(x.T) # in ra "[[1 3]
           #          [2 4]]"
# chú ý: mảng 1 chiều không có chuyển vị
y = np.array([1, 2, 3])
print(y) # in ra "[1 2 3]"
print(y.T) # in ra "[1 2 3]"
z = np.array([[1, 2, 3]])
print(z.T)
```

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

Một số thao tác thông dụng

- Cơ chế broadcasting
  - Khi +, -, , / matrix với vector thì python sẽ tự động biến đổi vector thành một matrix cùng kích thước với matrix kia, đó gọi là broadcasting. Tổng quát (m, n) + - * / (1, n) -> (m, n) (m, 1) -> (m, n)

```
import numpy as np
x = np.array([1, 2, 4])
a = x + 1
# tự động broadcast 1 thành np.array([1, 1, 1])
print(a)
x = np.array([[1, 2, 4], [4, 2, 1]])
b = x + np.array([0, 1, 4])
#tự động broadcast np.array([0, 1, 4]) thành np.array([[0, 1, 4], [0, 1, 4]])
print(b)
x = np.array([[1, 2, 4], [4, 2, 1]])
c = x+ np.array([[0], [1]])
#tự động broadcast np.array([[0], [1]]) thành np.array([[0, 0, 0], [1, 1, 1]])
print(c)
```

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

Một số thao tác thông dụng

- Tính tổng theo các trục

```
import numpy as np
x = np.array([[1, 2], [3, 4]])
print(np.sum(x)) # tính tổng toàn bộ x, in "10"
print(np.sum(x, axis=0)) # tính tổng mỗi cột, in "[4 6]"
print(np.sum(x, axis=1)) # tính tổng mỗi hàng, in "[3 7]"
```

Một số thao tác thông dụng

- Trích xuất dữ liệu theo dãy

```
import numpy as np
a = np.array([[1,2], [3, 4], [5, 6]])
print(a)
# Prints "[1 4 5]"
print(a[[0, 1, 2], [0, 1, 0]])
# Prints "[1 4 5]"
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))
# Prints "[2 2]"
print(a[[0, 0], [1, 1]])
# Prints "[2 2]"
print(np.array([a[0, 1], a[0, 1]]))
```

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

Một số thao tác thông dụng

- Lọc phần tử theo chỉ số

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
b = np.array([0, 2, 0, 1]) # b là mảng các chỉ số
print(a[np.arange(4), b]) # in ra "[1 6 7 11]"
# cộng tất cả các phần tử được lọc thêm 10
a[np.arange(4), b] += 10
print(a)
```

```
[ 1  6  7 11]
[[11  2  3]
 [ 4  5 16]
 [17  8  9]
 [10 21 12]]
```

Một số thao tác thông dụng

- Lọc dữ liệu theo điều kiện

```
import numpy as np
a = np.array([[1, 2], [3, 4], [5, 6]])
bool_idx = (a > 2)
print(bool_idx)
# lọc dữ liệu trong a, trả về một dãy
print(a[bool_idx]) # Prints "[3 4 5 6]"
print(type(a[bool_idx]),a[bool_idx].shape)
# có thể viết trực tiếp điều kiện (ngắn gọn hơn)
print(a[a > 2]) # Prints "[3 4 5 6]"
print(type(a[a > 2]),a[a > 2].shape)
```

```
[[False False]
 [ True  True]
 [ True  True]]
[3 4 5 6]
<class 'numpy.ndarray'> (4,)
[3 4 5 6]
<class 'numpy.ndarray'> (4,)
```

Một số thao tác thông dụng

- Điều chỉnh cỡ ma trận

```
import numpy as np
x = np.array([[1, 3], [4, 4], [4, 2]])
print(x)
print(x.shape) # (3, 2)
x = np.array([[1, 3], [4, 4], [4, 2]])
a = x.reshape(2, 3) # chỉnh thành 2x3
print(a) # array([[1, 3, 4], [4, 4, 2]])
x = np.array([[1, 3], [4, 4], [4, 2]])
b = x.reshape(2, -1) # tự tính chiều còn lại
print(b) # array([[1, 3, 4], [4, 4, 2]])
```

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

Một số thao tác thông dụng

- Elementwise operation

```
import numpy as np
x = np.array([1, 2, 3])
print(np.log(x)) # lấy log cơ số e từngphần tử
print(np.abs(x)) #lấy trị tuyệt đối từng phần tử
print(np.maximum(x, 2)) # so sánh từng phần tử với 2 và lấy max
print(np.minimum(x, 2)) # so sánh từngphần tử với 2 và lấy min
print(x**2) # lũy thừa 2 từng phần tử
```

```
[0.         0.69314718 1.09861229]
[1 2 3]
[2 2 3]
[1 2 2]
[1 4 9]
```

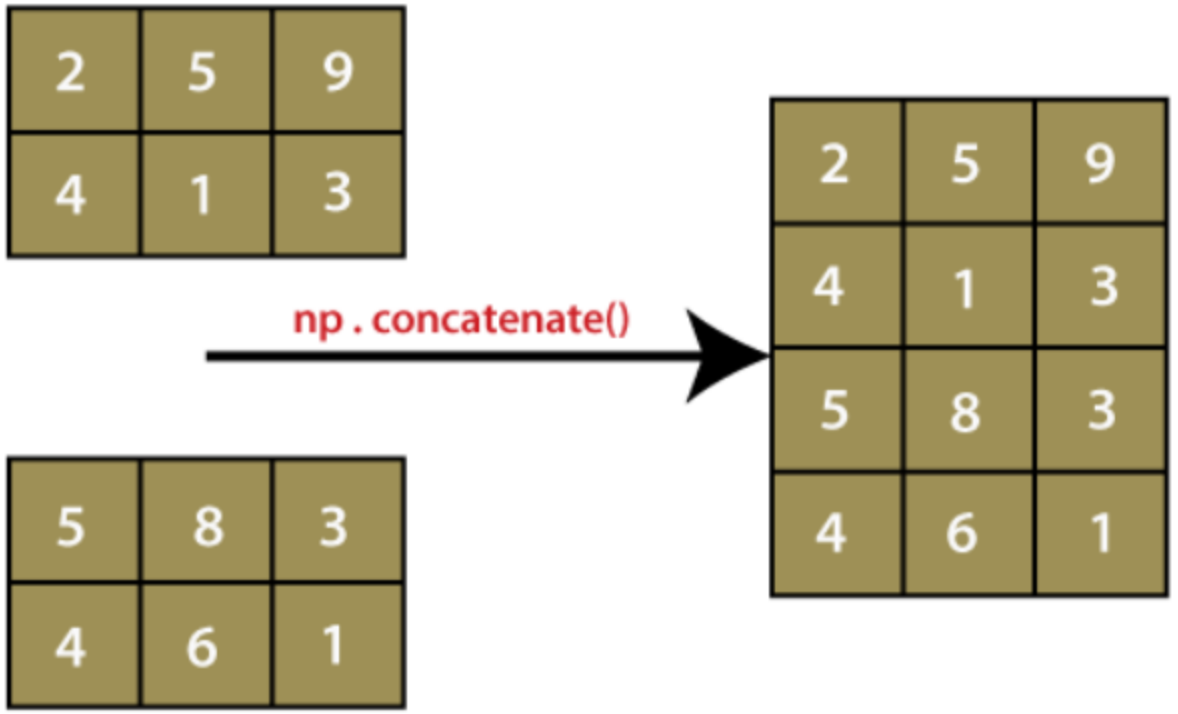
Một số thao tác thông dụng

- Nối mảng (concatenation): nối các mảng với nhau theo chiều ngang hoặc chiều dọc

Cú pháp:
numpy.concatenate ((a1, a2, ...), axis)

Trong đó:

- a1, a2, a3 … là các mảng có hình dạng giống nhau, ngoại trừ chiều tương ứng với trục.
- axis: Tham số này xác định trục mà mảng sẽ được nối với nhau. Theo mặc định, giá trị của nó là 0.



Ví dụ

In [1]:
```
import numpy as np
x=np.array([[1,2],[3,4]])
y=np.array([[12,30]])
z=np.concatenate((x,y))
z
```

Out[1]:
```
array([[ 1,  2],
       [ 3,  4],
       [12, 30]])
```

In [2]:
```
import numpy as np
x=np.array([[1,2],[3,4]])
y=np.array([[12,30]])
z=np.concatenate((x,y), axis=0)
z
```

Out[2]:
```
array([[ 1,  2],
       [ 3,  4],
       [12, 30]])
```

In [3]:
```
import numpy as np
x=np.array([[1,2],[3,4]])
y=np.array([[12,30]])
z=np.concatenate((x,y.T), axis=1)
z
```

Out[3]:
```
array([[ 1,  2, 12],
       [ 3,  4, 30]])
```

Ví dụ

- Ví dụ : numpy.concatenate () với axis = None

```
In [4]: import numpy as np
        x=np.array([[1,2],[3,4]])
        y=np.array([[12,30]])
        z=np.concatenate((x,y), axis=None)
        z

Out[4]: array([ 1,  2,  3,  4, 12, 30])
```

Một số thao tác thông dụng

- **Stacking: numpy.stack ()**
- **Vertical & Horizontal Stacking:**
  - hstack() to stack along rows.
  - vstack() to stack along columns.

In [10]:
```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.stack((arr1, arr2), axis=1)
print(arr)
```

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

In [11]:
```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.hstack((arr1, arr2))
print(arr)
```

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

In [12]:
```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.dstack((arr1, arr2))
print(arr)
```

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

Tham khảo

- **https://numpy.org/learn/**
- https://phamdinhkhanh.github.io/deepai-book/ch_appendix/appendix_numpy.html

Thực Hành

1. Tạo một ma trận 4x4 toàn các giá trị False (hint: np.full())
2. Cho một dãy số nguyên 10 phần tử, hãy tách lấy tất cả những phần tử lẻ cho vào một mảng (**hint**: lọc dữ liệu theo điều kiện)
3. Cho một dãy số tự nhiên 10 phần tử, hãy thay thế tất cả những phần tử lẻ bằng số -1 (**hint**: lọc dữ liệu theo điều kiện)
4. Hai mảng a và b có cùng số dòng, hãy ghép chúng theo các dòng thành mảng c, các cột của a rồi đến các cột của b (**hint**: dung numpy.concatenate(,))
5. Mảng a và b có cùng số cột, hãy ghép chúng theo các cột thành mảng c, các dòng của a rồi đến của b (**hint**: dung numpy.concatenate(,))