

► Numpy

Phần I: Giới thiệu và Cài đặt

> 1. NumPy là gì?

NumPy (viết tắt của **Numerical Python**) là một thư viện mã nguồn mở dành cho ngôn ngữ lập trình Python. Nó cung cấp một đối tượng mảng đa chiều (multi-dimensional array) hiệu suất cao và các công cụ để làm việc với các mảng này. NumPy là nền tảng cho hầu hết các thư viện khoa học dữ liệu và máy học trong Python (như Pandas, Scikit-learn, TensorFlow).

Tại sao nên dùng NumPy thay vì List của Python?

- **Tốc độ:** Các phép toán trên mảng NumPy nhanh hơn nhiều so với trên List vì NumPy được viết bằng C và tối ưu hóa cho các phép toán số học.
- **Tiết kiệm bộ nhớ:** Mảng NumPy chiếm ít dung lượng bộ nhớ hơn so với List.
- **Tiện lợi:** Cung cấp một lượng lớn các hàm toán học tích hợp sẵn để thao tác trên mảng.

> 2. Cài đặt NumPy

Để sử dụng NumPy, bạn cần cài đặt nó. Nếu bạn đã cài đặt Anaconda, NumPy đã có sẵn. Nếu không, hãy mở Terminal (trên macOS/Linux) hoặc Command Prompt (trên Windows) và chạy lệnh sau:

```
terminal@8syncdev:~$
```

```
pip install numpy
```



Mọi thứ trong NumPy đều xoay quanh đối tượng cốt lõi: ndarray (n-dimensional array), hay còn gọi là mảng NumPy.

> 1. Tạo mảng NumPy

Đầu tiên, hãy import thư viện NumPy. Quy ước chung là import nó với bí danh np.

```
terminal@8syncdev:~$
```

```
import numpy as np
```

a. Tạo mảng từ List của Python

Đây là cách phổ biến nhất để tạo một mảng.

```
terminal@8syncdev:~$
```

```
# Tạo mảng 1 chiều (vector)
```

```
list_1d = [1, 2, 3, 4, 5]
```

```
arr_1d = np.array(list_1d)
```

```
print(arr_1d)
```

```
# Output: [1 2 3 4 5]
```

```
# Giải thích:
```

```
# - np.array(): Hàm này chuyển đổi một cấu trúc dữ liệu (như list, tuple) thành một mảng NumPy.
```

```
# - list_1d: Dữ liệu đầu vào.
```

```
# - arr_1d: Biến lưu trữ mảng NumPy vừa được tạo.
```

```
terminal@8syncdev:~$
```

```

# Tạo mảng 2 chiều (ma trận)
list_2d = [[1, 2, 3], [4, 5, 6]]
arr_2d = np.array(list_2d)

print(arr_2d)
# Output:
# [[1 2 3]
#  [4 5 6]]

# Giải thích:
# - list_2d: Một list chứa các list con. Mỗi list con sẽ trở thành một hàng
trong mảng 2 chiều.

```

b. Sử dụng các hàm tạo mảng đặc biệt

NumPy cung cấp nhiều hàm tiện lợi để tạo các mảng với nội dung cụ thể mà không cần tạo list trước.

- `np.zeros()`: Tạo mảng chứa toàn số 0.

```

terminal@8syncdev:~$ ███
# Tạo mảng 1D có 5 phần tử 0
arr_zeros = np.zeros(5)
print(arr_zeros)
# Output: [0. 0. 0. 0. 0.]

```

```

# Tạo mảng 2D (ma trận 3x4) chứa toàn số 0
arr_zeros_2d = np.zeros((3, 4))
print(arr_zeros_2d)
# Output:
# [[0. 0. 0. 0.]
#  [0. 0. 0. 0.]
#  [0. 0. 0. 0.]]

```

```

# Giải thích:
# - np.zeros(shape): Hàm tạo mảng toàn số 0.
# - shape: Một số nguyên (cho mảng 1D) hoặc một tuple (cho mảng đa chiều) xác
định kích thước của mảng. (3, 4) có nghĩa là 3 hàng, 4 cột.

```

• `np.ones()`: Tương tự `np.zeros()`, nhưng tạo mảng chứa toàn số 1.

terminal@8syncdev:~\$

```
# Tạo mảng 2D (ma trận 2x3) chứa toàn số 1
arr_ones = np.ones((2, 3))
print(arr_ones)
# Output:
# [[1. 1. 1.]
# [1. 1. 1.]]
```

• `np.arange()`: Giống hàm `range()` của Python nhưng trả về một mảng NumPy.

terminal@8syncdev:~\$

```
# Tạo mảng chứa các số từ 0 đến 9
arr_arange = np.arange(10)
print(arr_arange)
# Output: [0 1 2 3 4 5 6 7 8 9]

# Tạo mảng chứa các số từ 2 đến 10, bước nhảy là 2
arr_arange_step = np.arange(2, 11, 2)
print(arr_arange_step)
# Output: [ 2  4  6  8 10]

# Giải thích:
# - np.arange(start, stop, step):
#   - start (tùy chọn): Giá trị bắt đầu (mặc định là 0).
#   - stop: Giá trị kết thúc (không bao gồm).
#   - step (tùy chọn): Khoảng cách giữa các giá trị (mặc định là 1).
```

• `np.linspace()`: Tạo mảng chứa các số cách đều nhau trong một khoảng cho trước.

terminal@8syncdev:~\$

```
# Tạo mảng có 5 phần tử, phân bố đều từ 0 đến 10
arrlinspace = np.linspace(0, 10, 5)
print(arrlinspace)
```

```
# Output: [ 0.  2.5  5.  7.5 10. ]  
  
# Giải thích:  
# - np.linspace(start, stop, num):  
#   - start: Giá trị bắt đầu.  
#   - stop: Giá trị kết thúc (bao gồm).  
#   - num: Số lượng phần tử cần tạo. NumPy sẽ tự động tính khoảng cách.
```

- ▶ **np.random**: Tạo mảng với các giá trị ngẫu nhiên.

```
terminal@8syncdev:~$  
  
# Tạo mảng 2x3 với các số ngẫu nhiên từ 0 đến 1 (phân phối đều)  
arr_rand = np.random.rand(2, 3)  
print(arr_rand)  
# Output (sẽ khác mỗi lần chạy):  
# [[0.8523661  0.99309323  0.17436894]  
#  [0.4496839  0.3582035  0.07728485]]  
  
# Tạo mảng 3x4 với các số nguyên ngẫu nhiên từ 1 đến 100  
arr_randint = np.random.randint(1, 101, (3, 4))  
print(arr_randint)  
# Output (sẽ khác mỗi lần chạy):  
# [[95 13 89 78]  
#  [30 79 64 89]  
#  [25 60 70 45]]
```

> 2. Các thuộc tính quan trọng của mảng

Mỗi mảng NumPy có các thuộc tính cung cấp thông tin về nó.

```
terminal@8syncdev:~$  
  
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])  
  
# shape: Kích thước của mảng (số hàng, số cột, ...)  
print(arr_2d.shape) # Output: (2, 3) -> 2 hàng, 3 cột  
  
# ndim: Số chiều của mảng  
print(arr_2d.ndim) # Output: 2
```

```
# size: Tổng số phần tử trong mảng
print(arr_2d.size)    # Output: 6 (vì  $2 * 3 = 6$ )
# dtype: Kiểu dữ liệu của các phần tử trong mảng
print(arr_2d.dtype)   # Output: int64 (có thể khác tùy hệ điều hành)
```

Phần III: Thao tác cơ bản trên mảng

> 1. Thay đổi hình dạng (Reshaping)

Bạn có thể thay đổi hình dạng của một mảng mà không làm thay đổi dữ liệu của nó, miễn là tổng số phần tử không đổi.

```
terminal@8syncdev:~$ ● ● ●
arr = np.arange(12)  # Mảng 1D từ 0 đến 11
print("Mảng ban đầu:", arr)
# Output: Mảng ban đầu: [ 0  1  2  3  4  5  6  7  8  9 10 11]

# Thay đổi thành mảng 3x4
reshaped_arr = arr.reshape(3, 4)
print("\nMảng sau khi reshape:\n", reshaped_arr)
# Output:
# Mảng sau khi reshape:
# [[ 0  1  2  3]
# [ 4  5  6  7]
# [ 8  9 10 11]]

# Giải thích:
# - arr.reshape(new_shape): Trả về một mảng MỚI với hình dạng đã thay đổi.
# - new_shape: Tuple xác định kích thước mới. Tổng số phần tử phải giữ nguyên
# ( $3 * 4 = 12$ ).
```

> 2. Truy cập phần tử (Indexing và Slicing)

a. Indexing (Truy cập một phần tử)

Giống như List, bạn truy cập phần tử bằng chỉ số (index), bắt đầu từ 0.

```
terminal@8syncdev:~$
```

```
arr1d = np.array([10, 20, 30, 40, 50])  
  
# Lấy phần tử ở vị trí thứ 2 (giá trị 30)  
print(arr1d[2]) # Output: 30  
  
# Đối với mảng 2D, bạn dùng cú pháp `arr[hàng, cột]`  
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
  
# Lấy phần tử ở hàng 1, cột 2 (giá trị 6)  
print(arr2d[1, 2]) # Output: 6  
# Hoặc có thể viết arr2d[1][2], nhưng cách trên hiệu quả hơn
```

b. Slicing (Cắt lấy một đoạn của mảng)

Slicing cho phép bạn lấy một tập con của mảng. Cú pháp là `start:stop:step`.

```
terminal@8syncdev:~$
```

```
arr1d = np.arange(10) # [0 1 2 3 4 5 6 7 8 9]  
  
# Lấy các phần tử từ vị trí 2 đến vị trí 5 (không bao gồm 5)  
print(arr1d[2:5]) # Output: [2 3 4]  
  
# Lấy các phần tử từ đầu đến vị trí 4  
print(arr1d[:4]) # Output: [0 1 2 3]  
  
# Lấy các phần tử từ vị trí 5 đến hết  
print(arr1d[5:]) # Output: [5 6 7 8 9]  
  
# Slicing trên mảng 2D  
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
  
# Lấy 2 hàng đầu tiên và 2 cột đầu tiên  
# :2 ở chiều thứ nhất -> lấy hàng 0, 1  
# :2 ở chiều thứ hai -> lấy cột 0, 1
```

```
sub_arr = arr_2d[ :2, :2]
print(sub_arr)
# Output:
# [[1 2]
#  [4 5]]

# Lấy toàn bộ hàng thứ 1 (hàng có chỉ số 0)
print(arr_2d[0, :]) # Output: [1 2 3]
```

Quan trọng: Lát cắt (slice) của một mảng NumPy là một "view" (khung nhìn) của mảng gốc, không phải một bản sao. Điều này có nghĩa là nếu bạn thay đổi dữ liệu trong lát cắt, dữ liệu trong mảng gốc cũng sẽ thay đổi.

```
terminal@8syncdev:~$ ● ● ●

arr = np.arange(5) # [0 1 2 3 4]
slice_arr = arr[1:3] # [1 2]

# Thay đổi phần tử đầu tiên của lát cắt
slice_arr[0] = 99

print(slice_arr) # Output: [99  2]
print(arr)       # Output: [ 0 99  2  3  4] -> Mảng gốc đã bị thay đổi!

# Để tạo một bản sao, hãy dùng hàm copy()
arr_copy = arr[1:3].copy()
arr_copy[0] = 111
print(arr) # Mảng gốc không đổi
```

> 3. Lọc mảng (Boolean Indexing)

Đây là một tính năng cực kỳ mạnh mẽ, cho phép bạn chọn các phần tử trong mảng dựa trên một điều kiện.

```
terminal@8syncdev:~$ ● ● ●

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

# Tạo một mảng boolean dựa trên điều kiện
```

```

# Mảng này sẽ có cùng kích thước với `arr`, mỗi phần tử là True nếu thỏa mãn
điều kiện, ngược lại là False.
bool_arr = arr > 2
print(bool_arr)
# Output:
# [[False False]
#  [ True  True]
#  [ True  True]]

# Dùng mảng boolean này để lọc các phần tử
# NumPy sẽ chỉ trả về những phần tử ở vị trí có giá trị True
print(arr[bool_arr])
# Output: [3 4 5 6]

# Có thể kết hợp trực tiếp trong một dòng
print(arr[arr > 2]) # Output: [3 4 5 6]

# Lọc các số chẵn
print(arr[arr % 2 == 0]) # Output: [2 4 6]

```

Phần IV: Phép toán trên mảng

Điểm mạnh lớn nhất của NumPy là khả năng thực hiện các phép toán trên toàn bộ mảng một cách nhanh chóng mà không cần vòng lặp. Đây được gọi là **vectorization**.

> 1. Phép toán số học cơ bản

Các phép toán (+, -, *, /, **) được thực hiện trên từng phần tử tương ứng (element-wise).

terminal@8syncdev:~\$



```

arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])

# Phép cộng
print("Cộng:\n", arr1 + arr2)
# Output:

```

```

# [[ 6  8]
#  [10 12]]

# Phép trừ
print("\nTrừ:\n", arr1 - arr2)
# Output:
# [[-4 -4]
#  [-4 -4]]

# Phép nhân (element-wise, không phải nhân ma trận)
print("\nNhân (element-wise):\n", arr1 * arr2)
# Output:
# [[ 5 12]
#  [21 32]]

# Phép chia
print("\nChia:\n", arr1 / arr2)
# Output:
# [[0.2          0.33333333]
#  [0.42857143  0.5         ]]

# Phép toán với một số vô hướng (scalar)
# Số vô hướng sẽ được áp dụng cho mọi phần tử trong mảng
print("\nNhân với vô hướng:\n", arr1 * 10)
# Output:
# [[10 20]
#  [30 40]]

```

> 2. Universal Functions (ufuncs)

NumPy cung cấp các hàm toán học có thể hoạt động trực tiếp trên mảng.

```

terminal@8syncdev:~$ ███

arr = np.arange(1, 6) # [1 2 3 4 5]

# Tính căn bậc hai của mỗi phần tử
print(np.sqrt(arr))
# Output: [1.          1.41421356 1.73205081 2.          2.23606798]

# Tính sin của mỗi phần tử
print(np.sin(arr))

# Tính tổng tất cả các phần tử

```

```
print(np.sum(arr)) # Output: 15  
  
# Tính giá trị trung bình  
print(np.mean(arr)) # Output: 3.0  
  
# Tìm giá trị lớn nhất  
print(np.max(arr)) # Output: 5
```

> 3. Phép toán theo trục (Axis)

Với mảng đa chiều, bạn có thể thực hiện các phép toán (như `sum`, `mean`, `max`) trên toàn bộ mảng hoặc theo một trục cụ thể.

- **axis=0:** Thực hiện phép toán theo **cột**.
- **axis=1:** Thực hiện phép toán theo **hàng**.

```
terminal@8syncdev:~$  
  
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])  
print(arr_2d)  
# [[1 2 3]  
#  [4 5 6]]  
  
# Tính tổng toàn bộ mảng  
print("Tổng toàn bộ:", np.sum(arr_2d)) # Output: 21  
  
# Tính tổng theo cột (axis=0)  
# [1+4, 2+5, 3+6]  
print("Tổng theo cột:", np.sum(arr_2d, axis=0)) # Output: [5 7 9]  
  
# Tính tổng theo hàng (axis=1)  
# [1+2+3, 4+5+6]  
print("Tổng theo hàng:", np.sum(arr_2d, axis=1)) # Output: [ 6 15]
```

Phần V: Các chủ đề nâng cao

> 1. Broadcasting (Phát sóng)

Broadcasting mô tả cách NumPy xử lý các mảng có hình dạng khác nhau trong các phép toán số học. Quy tắc chung là, nếu hai mảng khác hình dạng, NumPy sẽ "kéo dài" mảng nhỏ hơn để nó có cùng hình dạng với mảng lớn hơn, cho phép thực hiện phép toán element-wise.

Ví dụ đơn giản: Phép toán giữa mảng và số vô hướng.

```
terminal@8syncdev:~$
```

```
arr = np.array([1, 2, 3])
scalar = 10
# NumPy "phát sóng" số 10 thành mảng [10, 10, 10] rồi thực hiện phép cộng
result = arr + scalar
print(result) # Output: [11 12 13]
```

Ví dụ phức tạp hơn:

```
terminal@8syncdev:~$
```

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

vector = np.array([10, 20, 30])

# Cộng một vector vào mỗi hàng của ma trận
# NumPy sẽ "sao chép" vector này cho mỗi hàng của ma trận rồi cộng
# matrix (3, 3) + vector (3,) -> vector được broadcast thành (3, 3)
result = matrix + vector
print(result)
# Output:
# [[11 22 33]
#  [14 25 36]
#  [17 28 39]]
```

Broadcasting là một khái niệm mạnh mẽ nhưng có thể khó hiểu lúc đầu. Nó giúp tránh việc tạo ra các bản sao không cần thiết của dữ liệu, làm cho code hiệu quả hơn.

> 2. Đại số tuyến tính (Linear Algebra)

NumPy cung cấp module `np.linalg` cho các phép toán đại số tuyến tính.

```
terminal@8syncdev:~$  
  
A = np.array([[1, 2], [3, 4]])  
B = np.array([[5, 6], [7, 8]])  
  
# Nhân ma trận (khác với phép nhân element-wise *)  
# Dùng toán tử @ (khuyến nghị) hoặc hàm np.dot()  
dot_product = A @ B  
# dot_product = np.dot(A, B)  
print("Tích vô hướng (nhân ma trận):\n", dot_product)  
  
# Tìm ma trận nghịch đảo  
Ainv = np.linalg.inv(A)  
print("\nMa trận nghịch đảo của A:\n", Ainv)  
  
# Tìm định thức (determinant)  
det<sub>A</sub> = np.linalg.det(A)  
print("\nĐịnh thức của A:", det<sub>A</sub>)  
  
# Tìm giá trị riêng (eigenvalues) và vector riêng (eigenvectors)  
eigenvalues, eigenvectors = np.linalg.eig(A)  
print("\nGiá trị riêng:", eigenvalues)  
print("Vector riêng:\n", eigenvectors)
```

> 3. Chuyển vị và sắp xếp

```
terminal@8syncdev:~$  
  
arr = np.array([[1, 5, 3], [4, 2, 6]])  
  
# Chuyển vị ma trận (hàng thành cột, cột thành hàng)  
transposed_arr = arr.T  
print("Ma trận chuyển vị:\n", transposed_arr)
```

```

# Output:
# [[1 4]
# [5 2]
# [3 6]]


# Sắp xếp mảng 1D
arr1d = np.array([3, 1, 4, 1, 5, 9, 2, 6])
sorted_arr = np.sort(arr1d)
print("\nMảng đã sắp xếp:", sorted_arr)
# Output: [1 1 2 3 4 5 6 9]

# Sắp xếp mảng 2D
arr2d = np.array([[3, 2, 1], [6, 5, 4]])


# Sắp xếp theo hàng (axis=1)
sorted_rows = np.sort(arr2d, axis=1)
print("\nSắp xếp theo hàng:\n", sorted_rows)
# Output:
# [[1 2 3]
# [4 5 6]]


# Sắp xếp theo cột (axis=0)
sorted_cols = np.sort(arr2d, axis=0)
print("\nSắp xếp theo cột:\n", sorted_cols)
# Output:
# [[3 2 1] <- So sánh 3 và 6, 2 và 5, 1 và 4
# [6 5 4]]

```

Kết luận

Đây là một hướng dẫn toàn diện bao quát các khía cạnh quan trọng nhất của NumPy. Chìa khóa để thành thạo NumPy là **thực hành**. Hãy thử nghiệm với các loại mảng khác nhau, các hàm khác nhau và cố gắng giải quyết các bài toán nhỏ bằng cách sử dụng các phép toán trên mảng thay vì dùng vòng lặp `for` của Python. Khi bạn đã quen thuộc, bạn sẽ thấy tốc độ và sự tiện lợi mà nó mang lại cho công việc phân tích dữ liệu và tính toán khoa học là vô giá.



Nguyễn Phương Anh Tú (Alex Dev)



Ủng hộ tác giả

Học lập trình hiện đại cùng 8 Sync Dev, tự động hóa công việc, tối ưu hóa thời gian

📍 TP HCM, Việt Nam

✉️ 8sync.dev.1111@gmail.com

📞 0703930513

🔗 Kết nối

🔗 [github](#) [facebook](#) [youtube](#) [tiktok](#)

💬 [zalo](#) [zaloGroup](#)

➡️ Thông tin chuyển khoản:

Vietcombank

0767449819

Momo

0767449819

🛠 Tech Stack

FASTAPI

DJANGO

DJANGO REST FRAMEWORK

NEXTJS

ENCORE

DOCKER

MICROSERVICE ARCHITECTURE

KUBERNETES

⚡ Theme: **Dark Terminal** | 📅 21/8/2025 | 🚀 8 Sync Dev

Nền tảng giáo dục công nghệ trực tuyến tích hợp AI, Game 3D Web và Blockchain. Khám phá cách học lập trình hiện đại với công nghệ đột phá và ứng dụng thực tế.