

# Bài 5:

# LẬP TRÌNH PYTHON CƠ BẢN

## (Sử dụng thư viện NumPy làm việc với ma trận - 02)

AI Academy Vietnam

## 1. Chuyển đổi Vector, Array

- Reshape, Ravel, Concatenate, Split, Hsplit, Vsplit, Flip

## 2. Tính toán trên mảng

- Arithmetic operators | Abs | Trigonometric | Exponents | logarithms | Round

## 3. Sắp xếp các phần tử trong mảng (sort)

- Sắp xếp Vector | Sắp xếp Matrix

## 4. Tìm kiếm các phần tử trong mảng (where)

## 5. Ma trận vuông

- $\det(A)$  |  $A^{-1}$  | diagonal | Matrix below diagonal | trace of matrix | Eigen

## 6. Phép toán trên 2 ma trận

- `np.equal` | `np.add` | `np.subtract` | `np.dot` (@)

## 7. Rank(A), A.T

# 1. Kết hợp, chuyển đổi Vector, Ma trận

# 1.1 Reshape Arrays (1)

data

1
2
3
4
5
6

data.reshape(2,3)

1	2	3
4	5	6

data.reshape(3,2)

1	2
3	4
5	6

name of the new array

name of the original NumPy array

a tuple of values specifying the new shape

```
new_array = old_array.reshape((2, 6))
```

the reshape() method, called with "dot" notation

# 1.1 Reshape Arrays (2)

```
1 # Phương thức a.reshape(m,n)
2 vector_a = np.array([5,7,2,9,10,15,2,9,2,17,28,16],dtype=np.int16)
3 print(vector_a)
4 print('Số phần tử của vector:', vector_a.size)
5 print('-----')
6 #Chuyển đổi vector về matrix (n x m)
7 #Lưu ý: matrix.size =vector.size
8 matrix_a = vector_a.reshape((3,4))
9 print('Reshape về matrix: 3 x 4')
10 print(matrix_a)
11 print('Số phần tử của matrix_a:',matrix_a.size)
12 print('-----')
13 print('Reshape về matrix: 2 x 6')
14 matrix_b = vector_a.reshape((2,6))
15 print(matrix_b)
16 print('Số phần tử của matrix_b:',matrix_b.size)
```

[ 5 7 2 9 10 15 2 9 2 17 28 16]

Số phần tử của vector: 12

-----  
Reshape về matrix: 3 x 4

[[ 5 7 2 9]  
 [10 15 2 9]  
 [ 2 17 28 16]]

Số phần tử của matrix\_a: 12

-----  
Reshape về matrix: 2 x 6

[[ 5 7 2 9 10 15]  
 [ 2 9 2 17 28 16]]

0	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---

Reshape Vector to Matrix

0	1	2
3	4	5
6	7	8

# 1.1 Reshape Arrays (3)

```
a1 = np.arange(1, 13)
```

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

Sắp xếp thứ tự các phần tử khi reshape array sử dụng thuộc tính

**order='C' | 'F'**

→

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

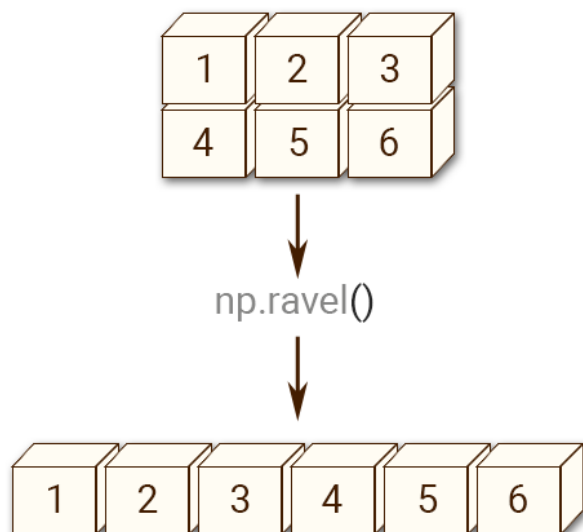
```
a1.reshape(3, 4) # reshapes or 'fills in' row by row  
a1.reshape(3, 4, order='C') # same results as above
```

↓

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

```
a1.reshape(3, 4, order='F') # reshapes column by column
```

# 1.2 Flatten | ravel Arrays



```
1 #Chuyển đổi từ Matrix --> Vector
2
3 a1_2d = np.array([(1,2,3,4),(5,6,7,8),(9,10,11,12)])
4 print('Matrix: \n', a1_2d)
5
6 print('-----')
7 print('a) ravel by row (default order=\'C\')')
8 print(a1_2d.ravel())
9
10 print('\n b) ravel by column (order=\'F\')')
11 print(a1_2d.ravel(order='F'))
```

Matrix:

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

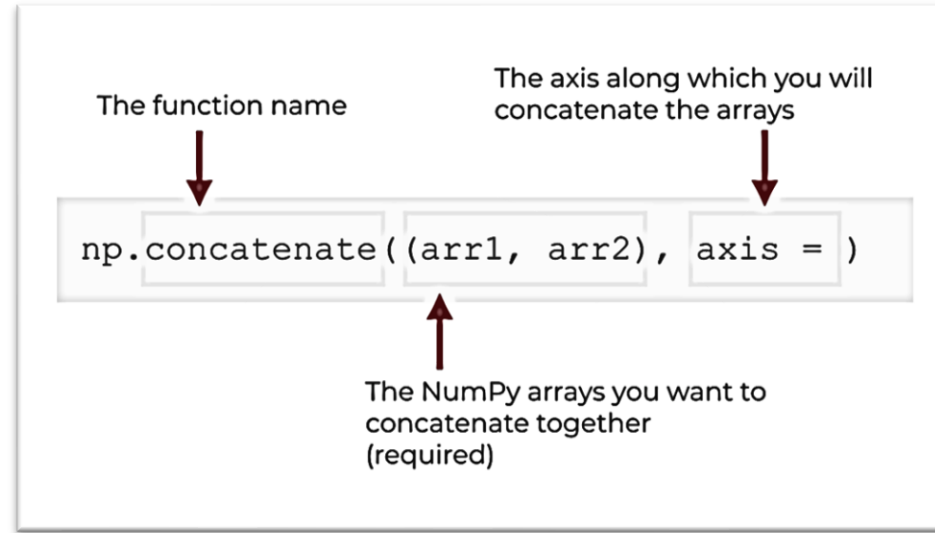
-----  
a) ravel by row (default order='C')

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

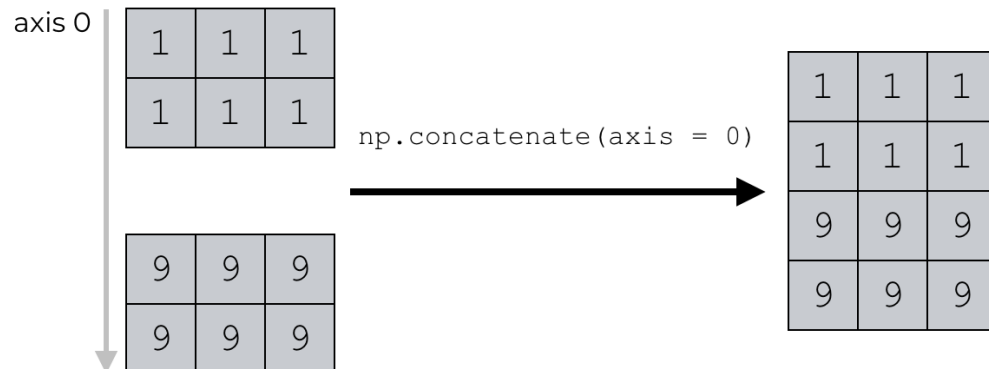
b) ravel by column (order='F')

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

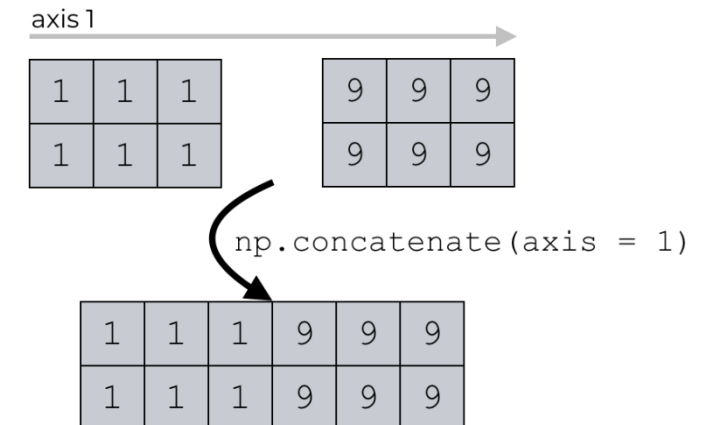
# 1.3 Concatenate Arrays (1)



Setting `axis=0` concatenates along the row axis



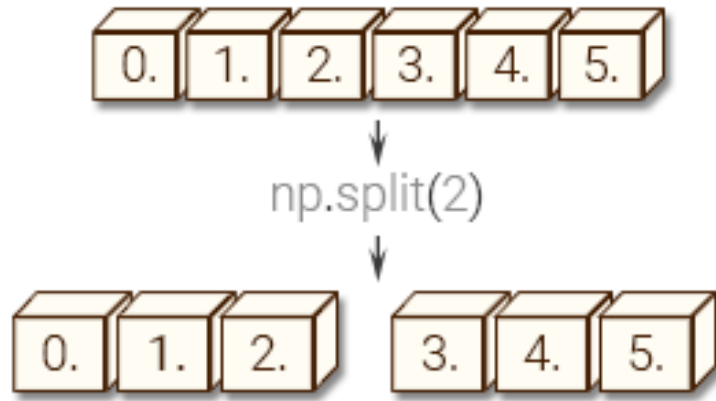
Setting `axis=1` concatenates along the column axis





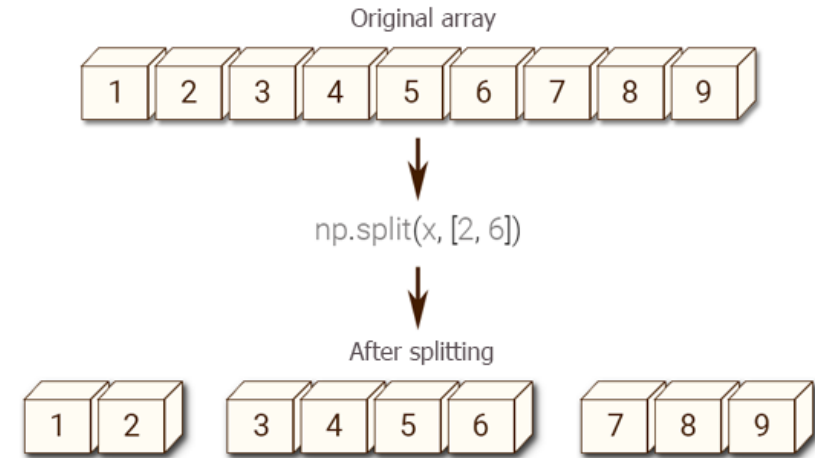
# 1.4 Split Arrays (1)

**Split:** Tách một vector, ma trận thành các vector, ma trận con



```
1 import numpy as np
2 x = np.arange(0,6)
3 print(x)
4
5 #Tách vector x thành 2 vector
6 #có số phần tử bằng nhau
7 x1, x2 = np.split(x, 2)
8 print(x1, x2)
```

```
[0 1 2 3 4 5]
[0 1 2] [3 4 5]
```



```
1 import numpy as np
2 x = np.arange(1,10)
3 print(x)
4
5 #Tách vector x thành 3 vector
6 #tại các vị trí 2 và 6
7 x1, x2, x3 = np.split(x, [2,6])
8 print(x1, x2, x3)
```

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

# 1.4 Split Arrays (2)

Vsplit, hsplit: Tách một ma trận thành các ma trận con theo hàng, cột

0.	1.	2.	3.	4.
5.	6.	7.	8.	9.
10.	11.	12.	13.	14.
15.	16.	17.	18.	19.

`np.vsplit(2)`

0.	1.	2.	3.	4.
5.	6.	7.	8.	9.

10.	11.	12.	13.	14.
15.	16.	17.	18.	19.

© w3resource.com

0.	1.	2.	3.
4.	5.	6.	7.
8.	9.	10.	11.
12.	13.	14.	15.

`np.hsplit(a, 2)`

0.	1.
4.	5.
8.	9.
12.	13.

2.	3.
6.	7.
10.	11.
14.	15.

© w3resource.com

# 1.5 Flip

Ma trận ban đầu:

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]
```

- `np.flip(A,0) ~ np.flipud(A)`: Lật ngược ma trận A theo hàng.

```
1 #Lật ma trận theo hàng
2 A2 = np.flip(A,0)
3 #Tương đương với
4 A2 = np.flipud(A)
5 print('Lật ma trận theo hàng: \n',A2)
```

Lật ma trận theo hàng:

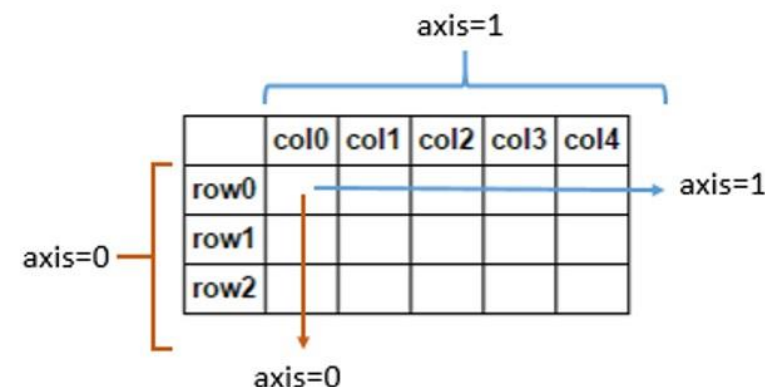
```
[[21 22 23 24 25]
 [16 17 18 19 20]
 [11 12 13 14 15]
 [ 6  7  8  9 10]
 [ 1  2  3  4  5]]
```

- `np.flip(A,1) ~ np.fliplr(A)`: Lật ngược ma trận A theo cột.

```
1 #Lật ma trận theo cột
2 A1 = np.flip(A,1)
3 #Tương đương với
4 A1 = np.fliplr(A)
5 print('Lật ma trận theo cột: \n',A1)
```

Lật ma trận theo cột:

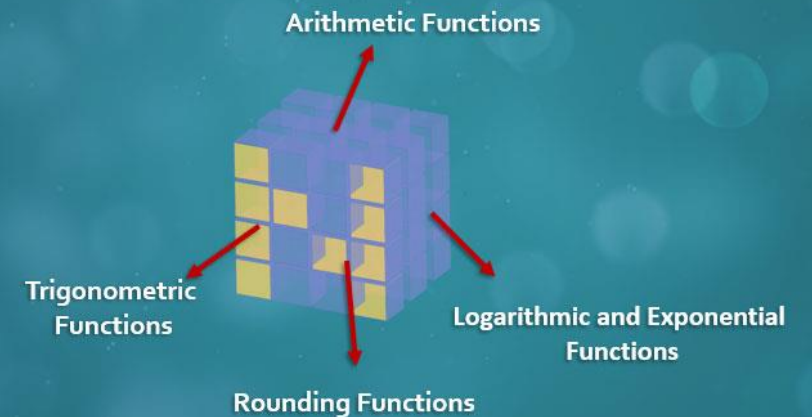
```
[[ 5  4  3  2  1]
 [10  9  8  7  6]
 [15 14 13 12 11]
 [20 19 18 17 16]
 [25 24 23 22 21]]
```



# Thực hành 1

## 2. Tính toán với NumPy

### NumPy Functions



## 2.1 Arithmetic operators.

The following table lists the arithmetic operators implemented in NumPy:

Operator	Equivalent ufunc	Description
<code>+</code>	<code>np.add</code>	Addition (e.g., <code>1 + 1 = 2</code> )
<code>-</code>	<code>np.subtract</code>	Subtraction (e.g., <code>3 - 2 = 1</code> )
<code>-</code>	<code>np.negative</code>	Unary negation (e.g., <code>-2</code> )
<code>*</code>	<code>np.multiply</code>	Multiplication (e.g., <code>2 * 3 = 6</code> )
<code>/</code>	<code>np.divide</code>	Division (e.g., <code>3 / 2 = 1.5</code> )
<code>//</code>	<code>np.floor_divide</code>	Floor division (e.g., <code>3 // 2 = 1</code> )
<code>**</code>	<code>np.power</code>	Exponentiation (e.g., <code>2 ** 3 = 8</code> )
<code>%</code>	<code>np.mod</code>	Modulus/remainder (e.g., <code>9 % 4 = 1</code> )

## 2.1 Arithmetic operators(2).



```
1 import numpy as np
2 x = np.arange(8)
3 print("x      =", x)
4 print('-----')
5 #Các phép toán đã giới thiệu trong buổi 01
6 print("x + 5 =", x + 5)
7 print("x - 5 =", x - 5)
8 print("-x    =", -x)
9 print("x * 2 =", x * 2)
10 print("x / 2 =", x / 2)
11 print("x // 2 =", x // 2)
12 print("x % 2 =", x % 2)
13 print("x ^ 3 =", x**3)
```

x = [0 1 2 3 4 5 6 7]

-----

x + 5 = [ 5 6 7 8 9 10 11 12]

x - 5 = [-5 -4 -3 -2 -1 0 1 2]

-x = [ 0 -1 -2 -3 -4 -5 -6 -7]

x \* 2 = [ 0 2 4 6 8 10 12 14]

x / 2 = [0. 0.5 1. 1.5 2. 2.5 3. 3.5]

x // 2 = [0 0 1 1 2 2 3 3]

x % 2 = [0 1 0 1 0 1 0 1]

x ^ 3 = [ 0 1 8 27 64 125 216 343]

```
1 print("x      =", x)
2 print('-----')
3 #Sử dụng các phương thức của NumPy
4 print("x + 5 =", np.add(x,5))
5 print("x - 5 =", np.subtract(x, 5))
6 print("-x    =", np.negative(x))
7 print("x * 2 =", np.multiply(x,2))
8 print("x / 2 =", np.divide(x,2))
9 print("x // 2 =", np.floor_divide(x, 2))
10 print("x % 2 =", np.mod(x,2))
11 print("x ^ 3 =", np.power(x,3))
```

x = [0 1 2 3 4 5 6 7]

-----

x + 5 = [ 5 6 7 8 9 10 11 12]

x - 5 = [-5 -4 -3 -2 -1 0 1 2]

-x = [ 0 -1 -2 -3 -4 -5 -6 -7]

x \* 2 = [ 0 2 4 6 8 10 12 14]

x / 2 = [0. 0.5 1. 1.5 2. 2.5 3. 3.5]

x // 2 = [0 0 1 1 2 2 3 3]

x % 2 = [0 1 0 1 0 1 0 1]

x ^ 3 = [ 0 1 8 27 64 125 216 343]

## 2.2 Abs | Trigonometric functions

- np.abs() | np.absolute():** để tính giá trị tuyệt đối của các phần tử.

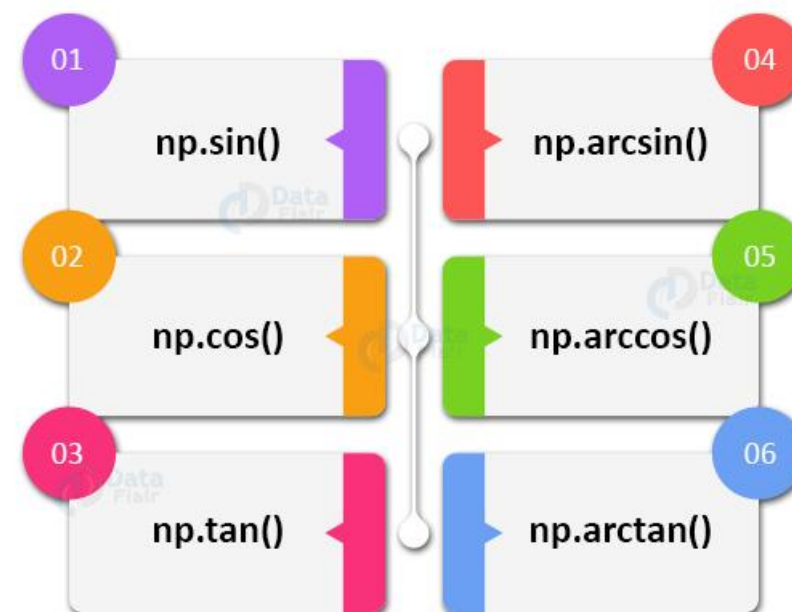
```
1 x = np.array([-2, -1, 0, 1, 2])
2 print(x)
3 print('-----')
4 print(np.abs(x))
5 print(np.absolute(x))
6
```

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

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

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

### Trigonometric Functions



```
1 theta = np.linspace(0, np.pi, 3)
2 print("theta = ", theta)
3 print('-----')
4 print("sin(theta) = ", np.sin(theta))
5 print("cos(theta) = ", np.cos(theta))
6 print("tan(theta) = ", np.tan(theta))
7
```

```
theta = [0. 1.57079633 3.14159265]
```

```
-----
sin(theta) = [0.00000000e+00 1.00000000e+00 1.2246468e-16]
```

```
cos(theta) = [ 1.0000000e+00  6.123234e-17 -1.0000000e+00]
```

```
tan(theta) = [ 0.00000000e+00  1.63312394e+16 -1.22464680e-16]
```



## 2.3 Exponents and logarithms

Function	Description
<code>exp(arr)</code>	Returns exponential of an input array element wise
<code>expm1(arr)</code>	Returns exponential $\exp(x)-1$ of an input array element wise
<code>exp2(arr)</code>	Returns exponential $2^{**}x$ of all elements in an array
<code>log(arr)</code>	Returns natural log of an input array element wise
<code>log10(arr)</code>	Returns log base 10 of an input array element wise
<code>log2(arr)</code>	Returns log base 2 of an input array element wise
<code>logaddexp(arr)</code>	Returns logarithm of the sum of exponentiations of all inputs
<code>logaddexp2(arr)</code>	Returns logarithm of the sum of exponentiations of the inputs in base 2

## 2.3 Exponents and logarithms (2)



VINBIGDATA



VINGROUP



AI Academy  
Vietnam

```
1 x = np.array([1, 2, 3])
2
3 print("x      =", x)
4 print('-----')
5 print("e^x    =", np.exp(x))
6 print("2^x    =", np.exp2(x))
7 print("3^x    =", np.power(3, x))
```

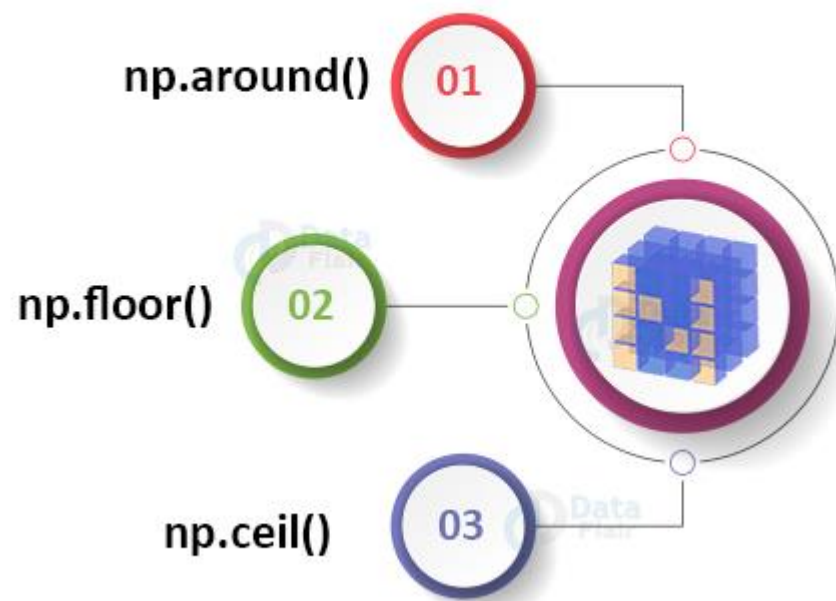
```
x      = [1 2 3]
-----
e^x    = [ 2.71828183  7.3890561 20.08553692]
2^x    = [2.  4.  8.]
3^x    = [ 3  9 27]
```

```
1 x = np.array([1, 2, 4, 100])
2
3 print("x      =", x)
4 print('-----')
5 print("ln(x)   =", np.log(x))
6 print("log2(x)  =", np.log2(x))
7 print("log10(x) =", np.log10(x))
```

```
x      = [ 1  2  4 100]
-----
ln(x)   = [0.          0.69314718 1.38629436 4.60517019]
log2(x) = [0.          1.          2.          6.64385619]
log10(x) = [0.          0.30103   0.60205999 2.          ]
```

## 2.4 Rounding Functions

### Rounding Functions in NumPy



```
1 import numpy as np
2 arr = np.array([20.8999, 67.89899, 54.43409])
3
4 print(arr)
5 print('-----')
6 #1) Làm tròn tới 1 số sau dấu ,
7 print(np.around(arr, 1))
8
9 #2) Làm tròn tới 2 số sau dấu ,
10 print(np.around(arr, 2))
11
12 #3) Làm tròn xuống số nguyên gần nhất
13 print(np.floor(arr))
14
15 #4) Làm tròn lên số nguyên gần nhất
16 print(np.ceil(arr))
```

```
[20.8999  67.89899 54.43409]
```

```
-----
[20.9 67.9 54.4]
```

```
[20.9 67.9 54.43]
```

```
[20. 67. 54.]
```

```
[21. 68. 55.]
```

### 3. Sắp xếp mảng (np.sort)

### 3. Sắp xếp các phần tử trong mảng (1)

1	4	2	3
9	13	61	1
43	24	88	22

`np.sort()`

1	2	3	4
1	9	13	61
22	24	43	88

The function name

The sorting algorithm you want to use to sort the data

```
np.sort( a= , axis= , kind= )
```

The array you want to operate on

The axis along which you will sort the array

\* kind='quicksort'- Default, 'mergesort', 'heapsort', 'stable'

### 3. Sắp xếp các phần tử trong mảng (2)



VINBIGDATA



#### Sắp xếp các phần tử trong một Vector

5	3	1	2	4
---	---	---	---	---



`np.sort()` SORTS THE  
VALUES OF A NUMPY ARRAY

1	2	3	4	5
---	---	---	---	---

```
1 #Sắp xếp các phần tử trong một vector
2 a = np.random.randint(1,33,15)
3
4 print('Vector ban đầu:\n', a)
5 print('-----')
6 #Sắp xếp vector a tăng dần
7 a_sort = np.sort(a)
8
9 #Sắp xếp vector a giảm dần:
10 #1) Lật vector a_sort để sắp xếp giảm dần
11 b_sort = np.flip(a_sort)
12 #2) sử dụng -np.sort(-x)
13 b_sort = -np.sort(-a)
14 print('Vector sắp xếp tăng dần: \n',a_sort)
15
16 print('Vector sắp xếp giảm dần: \n',b_sort)
```

Vector ban đầu:

```
[ 1 17 13 15  9  9 23 30 32 10 30 16  4 16 24]
```

Vector sắp xếp tăng dần:

```
[ 1  4  9  9 10 13 15 16 16 17 23 24 30 30 32]
```

Vector sắp xếp giảm dần:

```
[32 30 30 24 23 17 16 16 15 13 10  9  9  4  1]
```

### 3. Sắp xếp các phần tử trong mảng (3)



VINBIGDATA

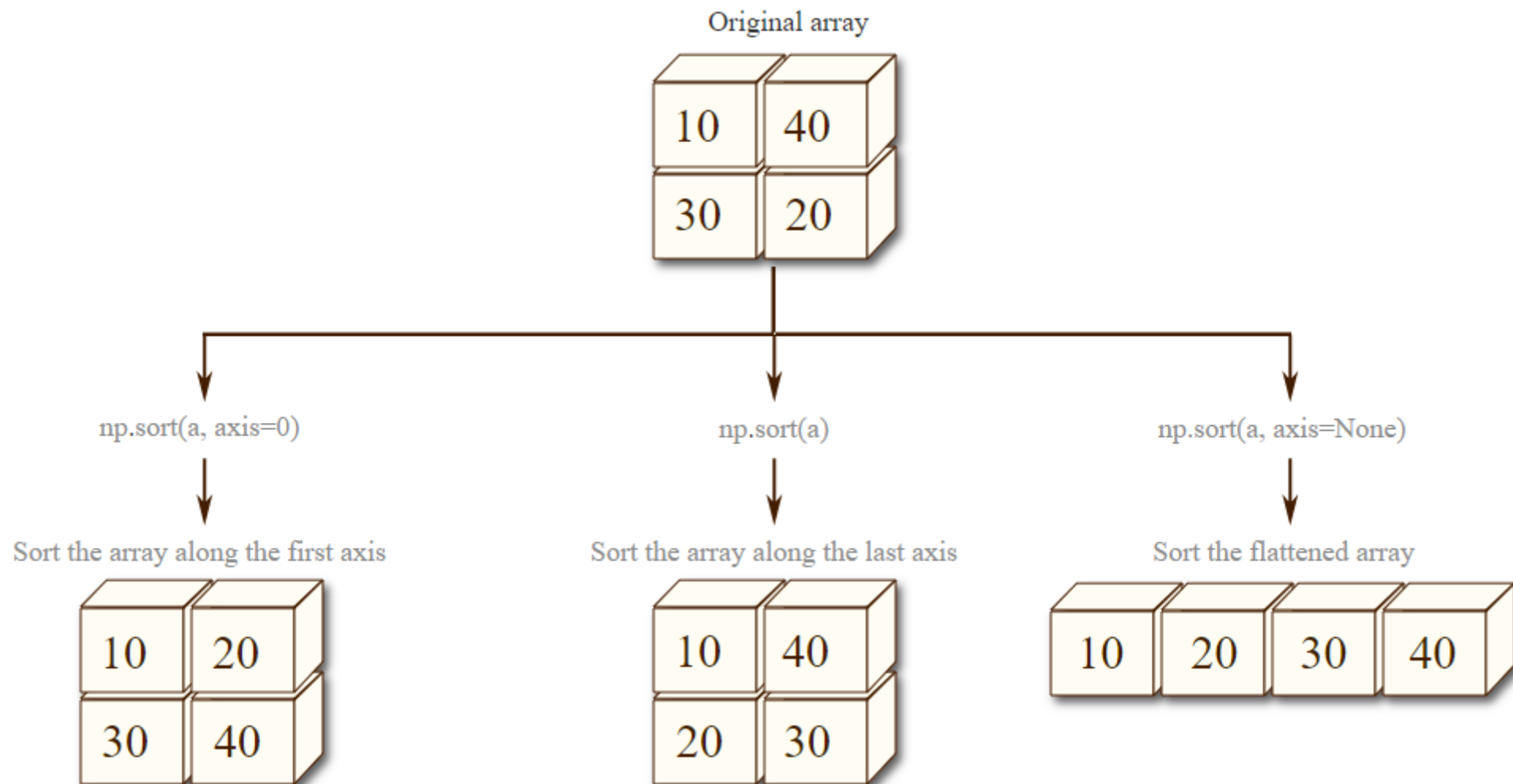


VINGROUP



AI Academy  
Vietnam

#### Sắp xếp các phần tử trong một Ma trận:



### 3. Sắp xếp các phần tử trong mảng (4)



VINBIGDATA



VINGROUP



AI Academy  
Vietnam

#### Sắp xếp các phần tử trong một Ma trận:

Ma trận A:

```
[[ 6 22 17 21 21 28]
 [17 22  4 19 14  9]
 [24  7 15  7 18 22]
 [ 6  9  2  8 23  3]
 [14 11 18  9 17  9]]
```

axis=1

axis=0

axis=None

```
1 #1) Sắp xếp theo cột axis=0
2 a_sort1 = np.sort(A,axis=0)
3 print('Ma trận 1:\n', a_sort1)
```

Ma trận 1:

```
[[ 6  7  2  7 14  3]
 [ 6  9  4  8 17  9]
 [14 11 15  9 18  9]
 [17 22 17 19 21 22]
 [24 22 18 21 23 28]]
```

```
1 #Sắp xếp các phần tử của ma trận A
2 #2) Sắp xếp theo hàng axis=1 | Default
3 a_sort2 = np.sort(A,axis=1)
4 print('Ma trận 2:\n', a_sort2)
```

Ma trận 2:

```
[[ 6 17 21 21 22 28]
 [ 4  9 14 17 19 22]
 [ 7  7 15 18 22 24]
 [ 2  3  6  8  9 23]
 [ 9  9 11 14 17 18]]
```

```
1 #3) Chuyển thành vector và sắp xếp các phần tử tăng dần theo hàng
2 v_sort = np.sort(A,axis=None)
3 print('Vector: \n', v_sort)
4
5 #Sắp xếp tất cả các phần tử theo thứ tự tăng dần theo hàng
6 a_sort3 = np.sort(A, axis=None).reshape(A.shape[0],A.shape[1])
7 print('Ma trận 3:\n', a_sort3)
```

Vector:

```
[ 2  3  4  6  6  7  7  8  9  9  9  9 11 14 14 15 17 17 18 18 19 21 21
 22 22 22 23 24 28]
```

Ma trận 3:

```
[[ 2  3  4  6  6  7]
 [ 7  8  9  9  9  9]
 [11 14 14 15 17 17]
 [17 18 18 19 21 21]
 [22 22 22 23 24 28]]
```



## 4. Tìm kiếm (np.where)

## 4. Tìm kiếm trong mảng: np.where (1)



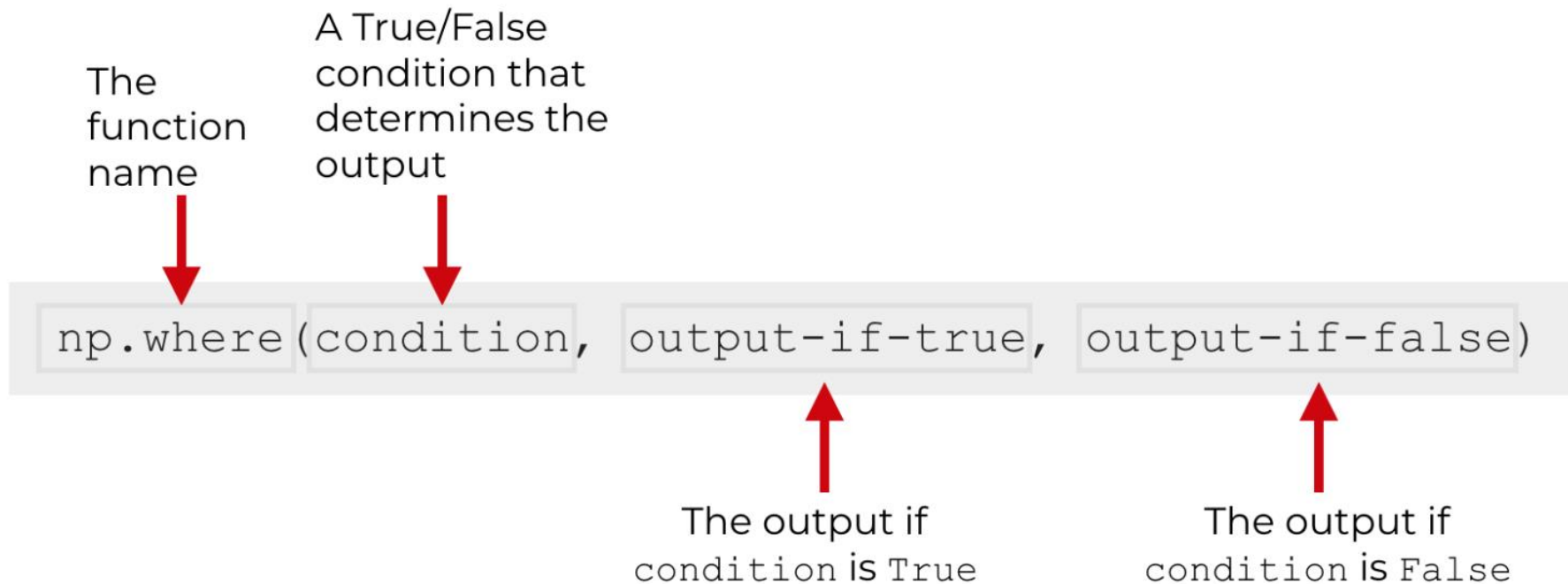
VINBIGDATA



VINGROUP



**np.where(): Tìm kiếm một phần tử trong mảng theo điều kiện**



## 4. Tìm kiếm trong mảng: np.where (2)



VINBIGDATA



### Vd1: Tìm kiếm trên vector

```
1 import numpy as np
2 x = np.array([17, 2, 11, 1, 9, 15, 1, 3, 8, 1, 12, 13, 5])
3 #1) Tìm kiếm các phần tử có giá trị ==1
4 t1 = np.where(x==1)
5 print(t1)
6 print('1. Số phần tử thỏa mãn điều kiện = 1: ', t1[0].size)
7 print('-----')
8 #2) Tìm kiếm các phần tử có giá trị >10
9 t2 = np.where(x>10)
10 print(t2)
11 print('2. Số phần tử thỏa mãn điều kiện>10: ', t2[0].size)
12 print('-----')
13 #Tìm kiếm các phần tử có giá trị [5,12)
14 t3 = np.where((x>=5) & (x<12))
15 print(t3)
16 print('3.Số phần tử thỏa mãn điều kiện [5,10): ', t3[0].size)
17
```

(array([3, 6, 9], dtype=int64),)

1. Số phần tử thỏa mãn điều kiện = 1: 3

(array([ 0, 2, 5, 10, 11], dtype=int64),)

2. Số phần tử thỏa mãn điều kiện>10: 5

(array([ 2, 4, 8, 12], dtype=int64),)

3.Số phần tử thỏa mãn điều kiện [5,10): 4

## 4. Tìm kiếm trong mảng: np.where (2)



VINBIGDATA



### Vd1: Tìm kiếm trên ma trận:

```
1 import numpy as np
2 #Tìm kiếm trên ma trận
3 arr = np.array([(1, 2, 3, 4, 5, 4, 4),
4                 (7, 3, 4, 8, 9, 6, 7)])
5 #Tìm kiếm phần tử > 4
6 x = np.where(arr > 4)
7
8 print('Ma trận A: \n',arr)
9 print('-----')
10 print(x)
11 print('Số phần tử thỏa mãn điều kiện > 4:',x[0].size)
```

Ma trận A:

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

-----

```
(array([0, 1, 1, 1, 1, 1], dtype=int64), array([4, 0, 3, 4, 5, 6], dtype=int64))
```

Số phần tử thỏa mãn điều kiện > 4: 6

## Thực hành 2

## 5. Ma trận vuông

## 5.1 Định thức det (A)

Định thức của ma trận vuông cấp  $n$  là tổng đại số của  $n!$  ( $n$  giai thừa) số hạng, mỗi số hạng là tích của  $n$  phần tử lấy trên các hàng và các cột khác nhau của ma trận  $A$ , mỗi tích được nhân với phần tử dấu là  $+1$  hoặc  $-1$  theo phép thế tạo bởi các chỉ số hàng và chỉ số cột của các phần tử trong tích

$$\det(A) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^n a_{i,\sigma(i)}$$

Định thức của một ma trận vuông còn được viết như sau

$$\det A = \begin{vmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} \end{vmatrix}$$

Áp dụng với các ma trận vuông cấp 1,2,3 ta có:

$$\det [a] = a$$

$$\det \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} a_{22} - a_{12} a_{21}$$

$$\det \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} a_{22} a_{33} + a_{12} a_{23} a_{31} + a_{13} a_{21} a_{32} - a_{13} a_{22} a_{31} - a_{12} a_{21} a_{33} - a_{11} a_{23} a_{32}$$

## 5.1 Định thức $\det(A)$



VINBIGDATA



`np.linalg.det(a)`: tính định thức của ma trận vuông `a`

```
1 import numpy as np
2 a = np.array([( 1, 3, 1, 4),
3               ( 3, 9, 5, 15),
4               ( 0, 2, 1, 1),
5               ( 0, 4, 2, 3)])
6 print('Ma trận a:\n',a)
7 det_a = np.linalg.det(a)
8 print('det(a) = ', det_a)
```

Ma trận a:

```
[[ 1  3  1  4]
 [ 3  9  5 15]
 [ 0  2  1  1]
 [ 0  4  2  3]]
det(a) = -3.9999999999999999
```

```
1 import numpy as np
2 b = np.array([( 1, 2, 3, 4),
3               (-2, -1, 4, 1),
4               ( 3, -4, -5, 6),
5               ( 1, 2, 3, 4)])
6 print('Ma trận b:\n',b)
7 det_b = np.linalg.det(b)
8 print('det(b) = ', det_b)
```

Ma trận b:

```
[[ 1  2  3  4]
 [-2 -1  4  1]
 [ 3 -4 -5  6]
 [ 1  2  3  4]]
det(b) = 0.0
```



## 5.2 Ma trận nghịch đảo

Ma trận nghịch đảo của ma trận vuông  $M$  ký hiệu  $M^{-1}$

$M * M^{-1} = I$  (ma trận đơn vị)

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{bmatrix}$$

$$M^{-1} = \begin{bmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{bmatrix}$$

**np.linalg.inv(m)**: Tìm ma trận nghịch đảo của ma trận  $m$

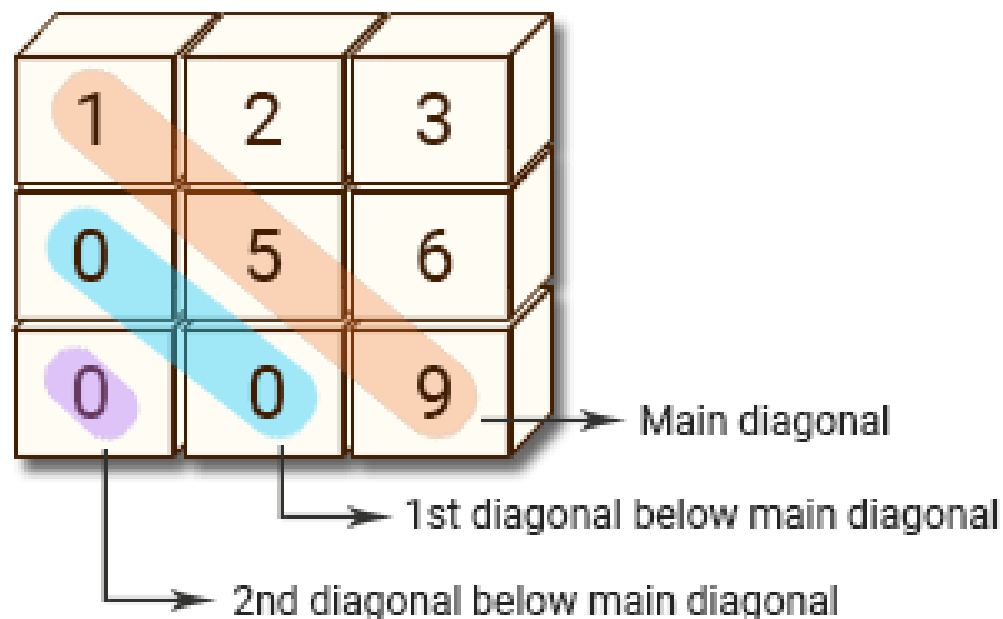
$\det(m) = 0$ : Không tồn tại ma trận nghịch đảo

## 5.3 Đường chéo (1)

### a) Lấy phần tử trên đường chéo

**a.diagonal():** trả về vector chứa các phần tử nằm trên đường chéo chính của ma trận a.

**a.diagonal(k):** trả về vector chứa các phần tử nằm cách đường chéo chính của ma trận a, k phần tử ( $k > 0$  trên đường chéo chính,  $k < 0$  dưới đường chéo chính)



## 5.3 Đường chéo (2)

79	98	60	84	47	28	10	48	83	43
59	55	80	82	30	52	70	56	77	91
6	15	17	62	21	64	89	31	69	1
20	50	6	77	62	10	96	54	51	89
72	16	56	62	77	30	23	3	77	4
73	68	71	70	80	20	78	70	90	58
7	48	14	78	26	99	69	13	91	21
4	35	57	20	31	3	73	16	25	14
89	73	27	32	2	83	71	55	17	34
95	15	67	75	86	47	36	96	72	92

```
1 #Lấy các phần tử nằm trên đường chéo chính
2 #của ma trận a 1 phần tử
3 d_A1 = A.diagonal(1)
4 print(d_A1)
```

```
[98 80 62 62 30 78 13 25 34]
```

```
1 #Lấy các phần tử trên đường chéo chính của
2 #ma trận vuông A
3 d_A = A.diagonal()
4 print(d_A)
```

```
[79 55 17 77 77 20 69 16 17 92]
```

```
1 #Lấy các phần tử nằm dưới đường chéo chính
2 #của ma trận a 4 phần tử
3 d_A1 = A.diagonal(-4)
4 print(d_A1)
```

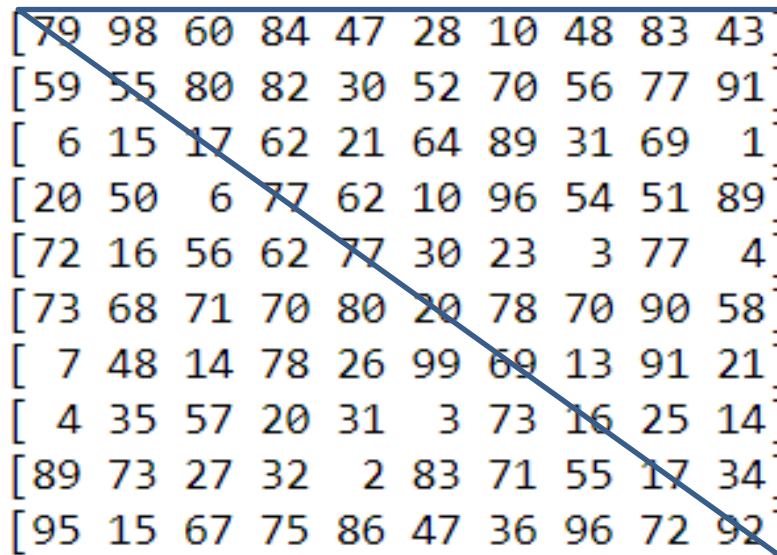
```
[72 68 14 20 2 47]
```

## 5.3 Đường chéo (3)

### b) Ma trận tam giác (triu | tril)

**np.triu(m) | np.tril(m):** trả về ma trận tam giác trên | dưới của ma trận m.

**np.triu(m, k):** trả về ma trận trên của ma trận m cách đường chéo chính k phần tử (  $k = 0$  (default),  $k < 0$  bên dưới đường chéo chính,  $k > 0$  bên trên đường chéo chính



```
[79 98 60 84 47 28 10 48 83 43]
[59 55 80 82 30 52 70 56 77 91]
[ 6 15 17 62 21 64 89 31 69  1]
[20 50  6 77 62 10 96 54 51 89]
[72 16 56 62 77 30 23  3 77  4]
[73 68 71 70 80 20 78 70 90 58]
[ 7 48 14 78 26 99 69 13 91 21]
[ 4 35 57 20 31  3 73 16 25 14]
[89 73 27 32  2 83 71 55 17 34]
[95 15 67 75 86 47 36 96 72 92]
```

```
1  #Ma trận tam giác trên của
2  #ma trận A
3  d_A1 = np.triu(A)
4  print(d_A1)
```

```
[[79 98 60 84 47 28 10 48 83 43]
 [ 0 55 80 82 30 52 70 56 77 91]
 [ 0  0 17 62 21 64 89 31 69  1]
 [ 0  0  0 77 62 10 96 54 51 89]
 [ 0  0  0  0 77 30 23  3 77  4]
 [ 0  0  0  0  0 20 78 70 90 58]
 [ 0  0  0  0  0  0 69 13 91 21]
 [ 0  0  0  0  0  0  0 16 25 14]
 [ 0  0  0  0  0  0  0  0 17 34]
 [ 0  0  0  0  0  0  0  0  0 92]]
```

## 5.3 Đường chéo (4)

```
1 #Tạo ma trận là các phần tử trên đường chéo chính
2 #của ma trận vuông A cách đường chéo chính
3 #về phía trên 2 đường
4 d_A1 = np.triu(A,2)
5 print(d_A1)
```

```
[[ 0  0 60 84 47 28 10 48 83 43]
 [ 0  0  0 82 30 52 70 56 77 91]
 [ 0  0  0  0 21 64 89 31 69  1]
 [ 0  0  0  0  0 10 96 54 51 89]
 [ 0  0  0  0  0  0 23  3 77  4]
 [ 0  0  0  0  0  0  0 70 90 58]
 [ 0  0  0  0  0  0  0  0 91 21]
 [ 0  0  0  0  0  0  0  0  0 14]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]]
```

```
1 #Tạo ma trận là các phần tử trên đường chéo chính
2 #của ma trận vuông A cách đường chéo chính
3 #về phía dưới 3 đường
4 d_A1 = np.triu(A,-3)
5 print(d_A1)
```

```
[[79 98 60 84 47 28 10 48 83 43]
 [59 55 80 82 30 52 70 56 77 91]
 [ 6 15 17 62 21 64 89 31 69  1]
 [20 50  6 77 62 10 96 54 51 89]
 [ 0 16 56 62 77 30 23  3 77  4]
 [ 0  0 71 70 80 20 78 70 90 58]
 [ 0  0  0 78 26 99 69 13 91 21]
 [ 0  0  0  0 31  3 73 16 25 14]
 [ 0  0  0  0  0 83 71 55 17 34]
 [ 0  0  0  0  0  0 36 96 72 92]]
```

## 5.3 Đường chéo (5)

### c) Vết của ma trận

#### Trace of a Matrix

Suppose  $T \in \mathcal{L}(V)$ ,  $\mathbf{F} = \mathbf{C}$ , and we choose a basis of  $V$  corresponding to the Decomposition Theorem. Then  $\text{trace } T$  equals the sum of the diagonal entries of that matrix.

**Definition: trace of a matrix**

The *trace* of a square matrix  $A$ , denoted  $\text{trace } A$ , is defined to be the sum of the diagonal entries of  $A$ .

**Example:** Suppose

$$A = \begin{pmatrix} 3 & -1 & -2 \\ 3 & 2 & -3 \\ 1 & 2 & 0 \end{pmatrix}.$$

Then

$$\begin{aligned} \text{trace } A &= 3 + 2 + 0 \\ &= 5. \end{aligned}$$

**Trace of  $AB$  equals trace of  $BA$**

If  $A$  and  $B$  are square matrices of the same size, then

$$\text{trace}(AB) = \text{trace}(BA).$$

```
1 #Tính trace của ma trận vuông A
2 trace_A = A.trace()
3 print('Trace of Matrix A: ',trace_A)
```

Trace of Matrix A: 519

```
1 #Cách 2: Tính trace của ma trận vuông A
2 trace_A = A.diagonal().sum()
3 print('Trace of Matrix A: ',trace_A)
```

Trace of Matrix A: 519



### d) Vector riêng(Eigenvector), giá trị riêng (Eigenvalue)

Một số  $\lambda$  và một vector khác 0  $\mathbf{x}$  thỏa mãn

$$\mathbf{Ax} = \lambda \mathbf{x}$$

Eigenvector of Matrix A

Eigenvalue of Matrix A

được gọi lần lượt là *giá trị riêng* và *vector riêng* của  $\mathbf{A}$

```
1 #Tìm giá trị riêng, vector riêng của vector A
2 A = np.array([(3, 4, -2),
3               (1, 4, -1),
4               (2, 6, -1)])
5
6 eigenvalues, eigenvectors = np.linalg.eig(A)
7 print('Eigenvalues: ', eigenvalues)
8 print('Eigenvectors: \n', eigenvectors)
```

Eigenvalues: [3. 2. 1.]

Eigenvectors:

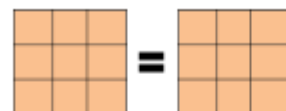
```
[[-4.08248290e-01  2.67561446e-15  7.07106781e-01]
 [-4.08248290e-01  4.47213595e-01 -3.33066907e-16]
 [-8.16496581e-01  8.94427191e-01  7.07106781e-01]]
```

# Thực hành 3



## 6. Phép toán với 2 ma trận

# 10. Phép toán trên 2 ma trận



Equality of  
Matrices

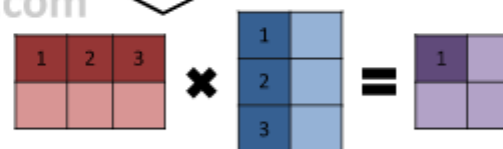
Addition of  
Matrices



Subtraction of  
Matrices

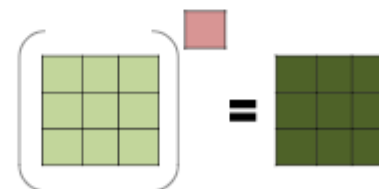
Multiplication of Matrices

$$\begin{pmatrix} 1 & 1 \end{pmatrix} + \begin{pmatrix} 2 & 2 \end{pmatrix} + \begin{pmatrix} 3 & 3 \end{pmatrix} = \begin{pmatrix} 1 \end{pmatrix}$$



Matrix  
Multiplied by  
a Scalar

Power of a  
Matrix



codeformech.com

## 6. Phép toán trên 2 ma trận

### a) So sánh 2 ma trận

`np.equal(a,b) | ==`: trả về ma trận (T|F) so sánh từng phần tử của ma trận a và ma trận b theo vị trí.

Lưu ý: ma trận a, b có cùng kích thước (m,n)

Matrix a:

```
[[ 9  4 19  1 18]
 [15 11  1  9 14]
 [17  8  4 10 13]]
```

Matrix b:

```
[[ 6  4  9 12  4]
 [ 3  6 11 14 10]
 [ 1  6  5 12  2]]
```

```
1 #1) So sánh 2 ma trận
2 equal_ab = np.equal(a,b)
3 #hoặc equal_ab = a==b
4
5 print(equal_ab)
```

```
[[False True False False False]
 [False False False False False]
 [False False False False False]]
```

## 6. Phép toán trên 2 ma trận

### b) Cộng, trừ 2 ma trận

**np.add(a,b) | +:** trả về ma trận có các phần tử là tổng của phần tử của ma trận a và ma trận b.

**np.subtract(a,b) | - :** trả về ma trận có các phần tử là hiệu của phần tử ma trận a và ma trận b

Lưu ý: ma trận a, b có cùng kích thước (m,n)

```
1 #Phép cộng 2 ma trận
2 sum_ab = np.add(a,b)
3 #hoặc sum_ab = a + b
4 print (sum_ab)
```

```
[[15  8 28 13 22]
 [18 17 12 23 24]
 [18 14  9 22 15]]
```

```
1 #Phép trừ 2 ma trận
2 sub_ab = np.subtract(a,b)
3 #hoặc sub_ab = a - b
4 print (sub_ab)
```

```
[[ 3  0 10 -11 14]
 [12  5 -10 -5  4]
 [16  2 -1  -2 11]]
```

## 6. Phép toán trên 2 ma trận

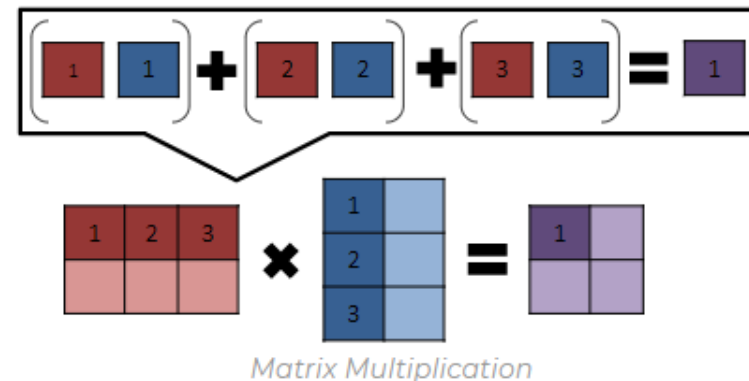
### c) Nhân 2 ma trận

`np.dot(a,b)` | `@`: trả về ma trận kết quả là tích của 2 ma trận a,b

Lưu ý: ma trận a có kích thước (m,n)  
ma trận c có kích thước (n,k)  
ma trận ac có kích thước (m,k)

```
1 #Tích của 2 vector:  
2 vector_a = np.random.randint(1,20,10)  
3 vector_b = np.random.randint(1,20,10)  
4 #Thực hiện tính tích của 2 vector  
5 #Kết quả trả về một số  
6 vector_ab = vector_a @ vector_b  
7  
8 print('Vector a:\n',vector_a)  
9 print('Vector b:\n',vector_b)  
10 print('Tích của hai vector:\n',vector_ab)
```

Vector a:  
[ 7 15 18 6 3 13 8 11 2 4]  
Vector b:  
[ 4 1 15 12 13 16 17 6 9 14]  
Tích của hai vector:  
908



Matrix a:

```
[[ 9  4 19  1 18]  
 [15 11  1  9 14]  
 [17  8  4 10 13]]
```

Matrix c:

```
[[13  8  9 13]  
 [17 11  4  3]  
 [13  7  2 18]  
 [12  1  7  2]  
 [ 1 13  6  4]]
```

```
1 #3) Tích của 2 ma trận:  
2 multi_ac = np.dot(a,c)  
3 #hoặc multi_ac1 = a@c  
4 print(multi_ac)
```

```
[[462 484 250 545]  
 [517 439 328 320]  
 [542 431 341 389]]
```

# Thực hành 4

## 7. Hạng của ma trận, ma trận chuyển vị

## 7.1 Hạng của ma trận A

Hạng của ma trận là cấp cao nhất của định thức con khác 0 của ma trận đó.

Hạng của ma trận A kí hiệu  $\text{rank}(A)$  hoặc  $r(A)$

+ Ma trận 0 có hạng bằng 0

+ Ma trận A cấp  $m \times n$  thì  $0 \leq r(A) \leq \min(m, n)$

+ Ma trận A vuông cấp  $n$ :

- Nếu  $\det(A) \neq 0$  thì  $r(A) = n$

- Nếu  $\det(A) = 0$  thì  $r(A) < n$

**`np.linalg.matrix_rank(A)`**: Tính hạng của ma trận A



# 7.1 Hạng của ma trận A

- Find the rank and nullity of the matrix

$$A = \begin{bmatrix} -1 & 2 & 0 & 4 & 5 & -3 \\ 3 & -7 & 2 & 0 & 1 & 4 \\ 2 & -5 & 2 & 4 & 6 & 1 \\ 4 & -9 & 2 & -4 & -4 & 7 \end{bmatrix}$$

Solution.

The reduced row-echelon form of A is

$$\begin{bmatrix} 1 & 0 & -4 & -28 & -37 & 13 \\ 0 & 1 & -2 & -12 & -16 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Since there are two nonzero rows, the row space and column space are both two-dimensional, so  $\text{rank}(A)=2$ .

```
1 #Hạng của ma trận 0
2 A_0 = np.zeros((4,5))
3 print(A_0)
4 rank = np.linalg.matrix_rank(A_0)
5 print('Rank(A_0) = ', rank)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
Rank(A_0) = 0
```

```
1 A=np.array([(-1, 2, 0, 4, 5, -3),
2             ( 3,-7, 2, 0, 1, 4),
3             ( 2,-5, 2, 4, 6, 1),
4             ( 4,-9, 2,-4,-4, 7)])
5 #Tìm hạng của ma trận A
6 rank_a = np.linalg.matrix_rank(A)
7 print(A)
8 print('Rank(A) = ', rank_a)
```

```
[[ -1  2  0  4  5 -3]
 [  3 -7  2  0  1  4]
 [  2 -5  2  4  6  1]
 [  4 -9  2 -4 -4  7]]
Rank(A) = 2
```

Ma trận B1:

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

$\det(B1) = 0.0$

$\text{Rank}(B1) = 3$

Ma trận B2:

```
[[ 1  3  1  4]
 [ 3  9  5 15]
 [ 0  2  1  1]
 [ 0  4  2  3]]
```

$\det(B2) = -4.0$

$\text{Rank}(B2) = 4$

## 7.2 Ma trận chuyển vị

*Chuyển vị* của ma trận  $m \times n$  **A** là ma trận  $n \times m$  **A<sup>T</sup>** tạo ra bằng cách chuyển hàng thành cột và cột thành hàng:

**A.T:** Tìm ma trận chuyển vị của ma trận A

```
1 A=np.array([(-1, 2, 0, 4, 5,-3),
2             ( 3,-7, 2, 0, 1, 4),
3             ( 2,-5, 2, 4, 6, 1),
4             ( 4,-9, 2,-4,-4, 7)])
5 #Tìm ma trận chuyển vị của A
6 A_T = A.T
7 print('Ma trận A:\n',A)
8 print('Ma trận chuyển vị của A:\n',A_T)
```

Ma trận A:

```
[[-1  2  0  4  5 -3]
 [ 3 -7  2  0  1  4]
 [ 2 -5  2  4  6  1]
 [ 4 -9  2 -4 -4  7]]
```

Ma trận chuyển vị của A:

```
[[-1  3  2  4]
 [ 2 -7 -5 -9]
 [ 0  2  2  2]
 [ 4  0  4 -4]
 [ 5  1  6 -4]
 [-3  4  1  7]]
```

```
1 B =np.array([( 1, 3, 1, 4),
2              ( 3, 9, 5,15),
3              ( 0, 2, 1, 1),
4              ( 0, 4, 2, 3)])
5 #Tìm ma trận chuyển vị của B
6 B_T = B.T
7 print('Ma trận B:\n',B)
8 print('Ma trận chuyển vị của B:\n',B_T)
```

Ma trận B:

```
[[ 1  3  1  4]
 [ 3  9  5 15]
 [ 0  2  1  1]
 [ 0  4  2  3]]
```

Ma trận chuyển vị của B:

```
[[ 1  3  0  0]
 [ 3  9  2  4]
 [ 1  5  1  2]
 [ 4 15  1  3]]
```

# Thực hành 5

Q & A  
Thank you!