LẬP TRÌNH PYTHON CHO PHÂN TÍCH DỮ LIỆU VÀ HỌC MÁY

Bài 5: Xử lý dữ liệu số với NumPy - 02

AI Academy Vietnam

Nội dung bài 5

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

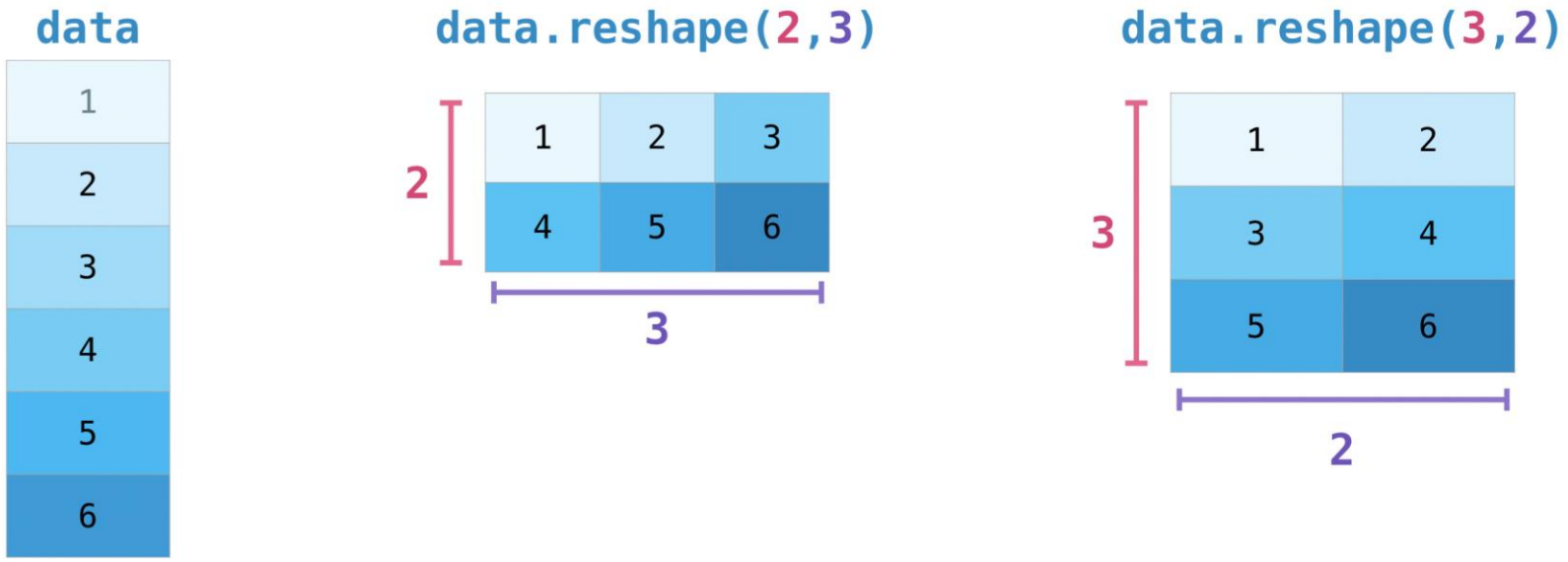
6. Phép toán trên 2 ma trận

- nq.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)



name of the new array

name of the original NumPy array

a tuple of values specifying the new shape

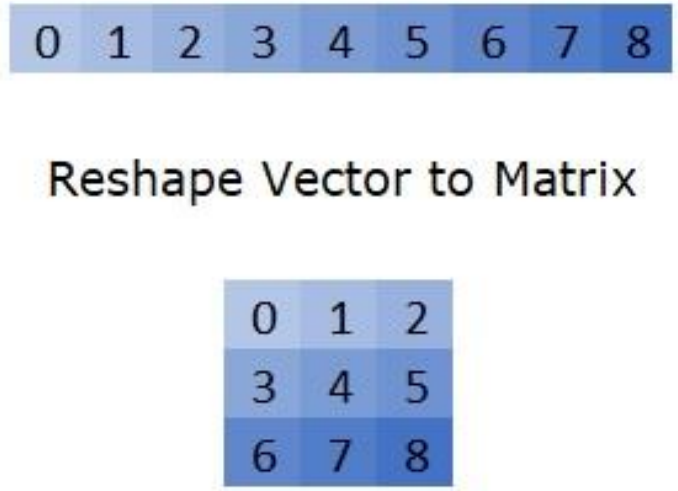
```
new_array = old_array.reshape((2,6))
```

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

1.1 Reshape Arrays (2)

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

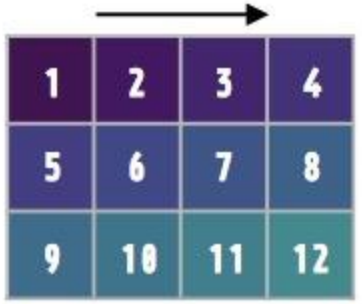


1.1 Reshape Arrays (3)

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

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

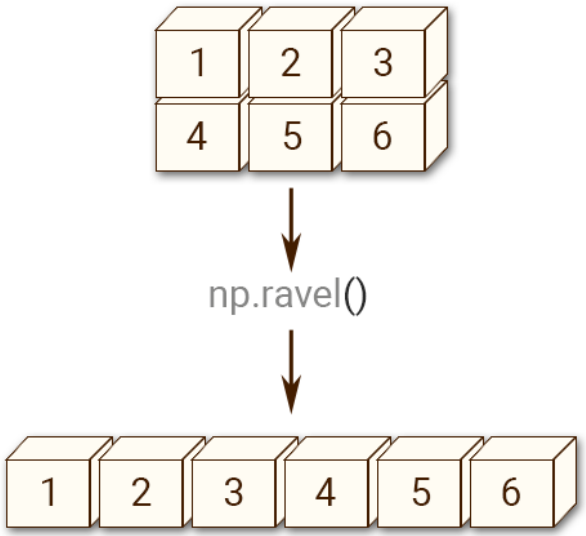
order =‘C’ | ‘F’




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


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

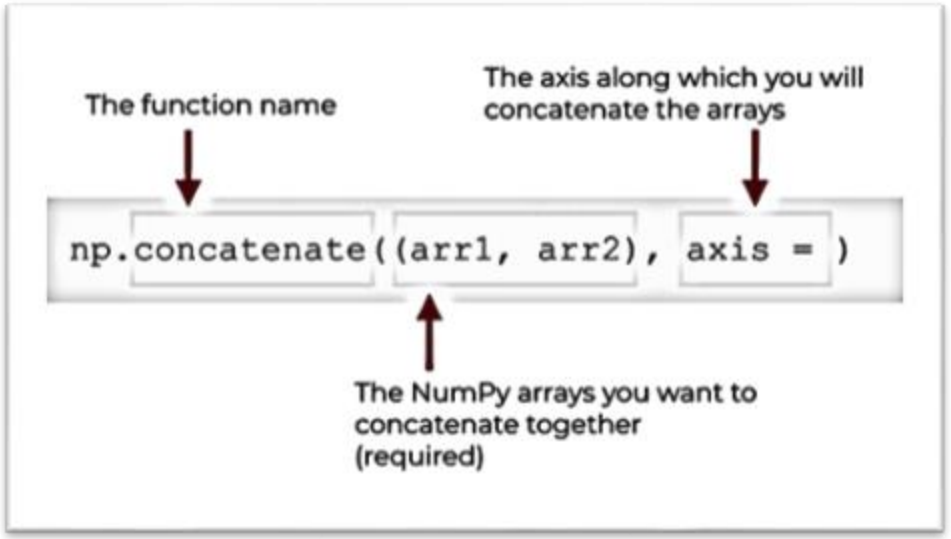
1.2 Flatten|ravel Arrays



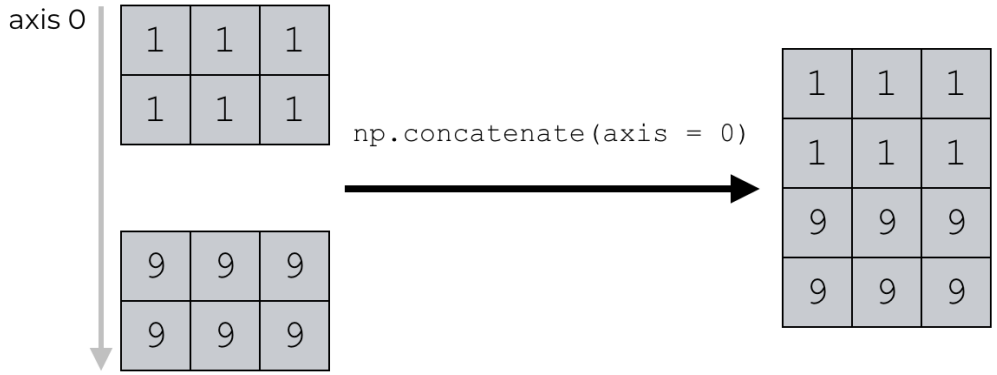

```python
#Chuyển đổi từ Matrix --> Vector

a1_2d = np.array([(1,2,3,4),(5,6,7,8),(9,10,11,12)])
print('Matrix: \n', a1_2d)

print('----------------------------------------')
print('a) ravel by row (default order=\'C\')')
print(a1_2d.ravel())

print('\n b) ravel by column (order=\'F\')')
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]
```

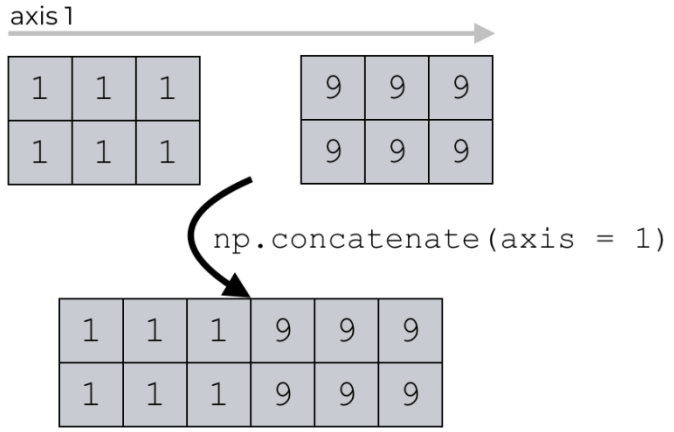
1.3 Concatenate Arrays (1)



Setting `axis=0` concatenates along the row axis

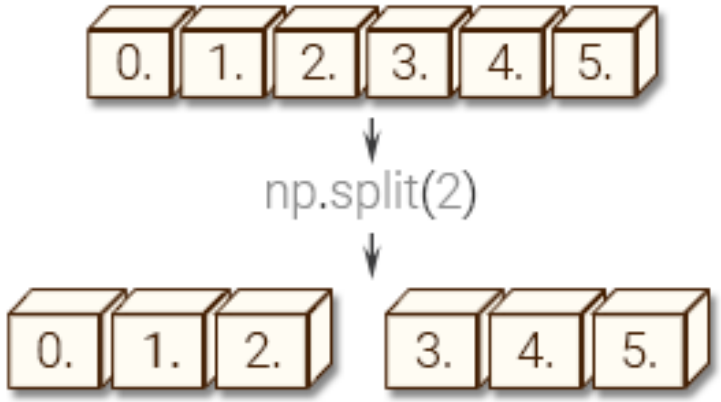


Setting `axis=1` concatenates along the column axis



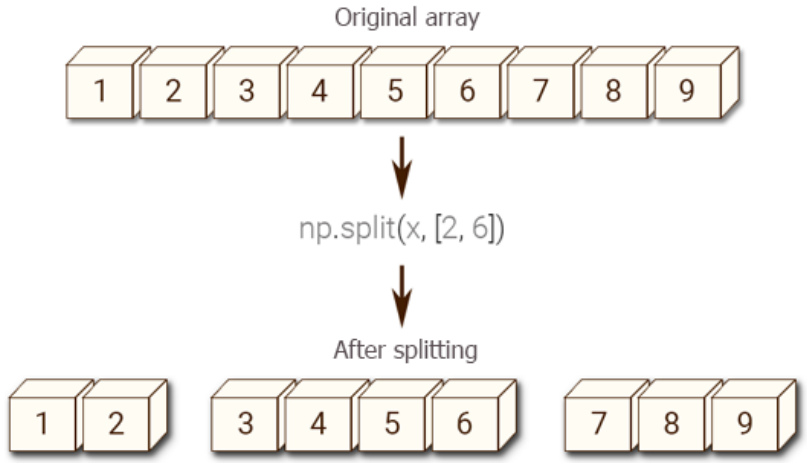
1.4 Split Arrays (1)

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




```python
import numpy as np
x = np.arange(0,6)
print(x)

#Tách vector x thành 2 vector
#có số phần tử bằng nhau
x1, x2 = np.split(x, 2)
print(x1, x2)
```

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

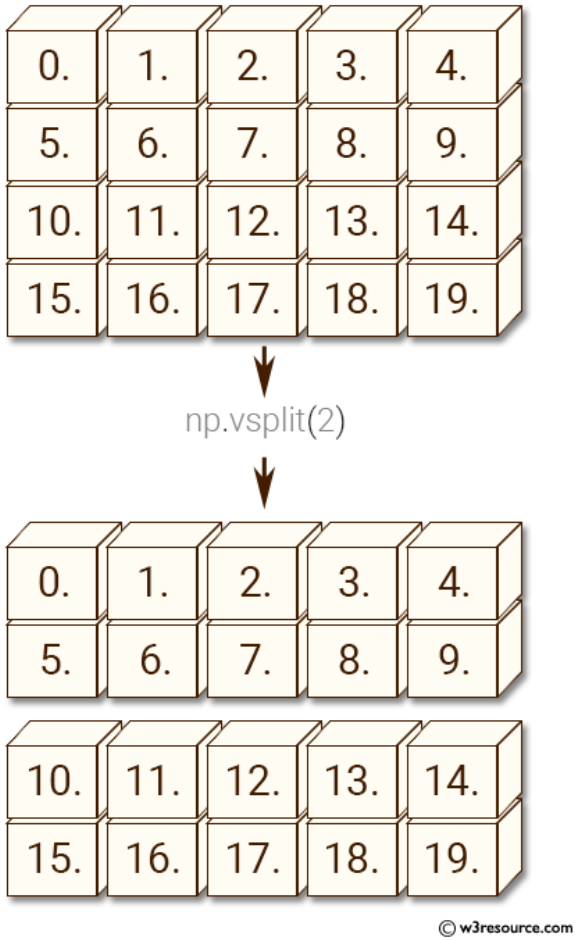


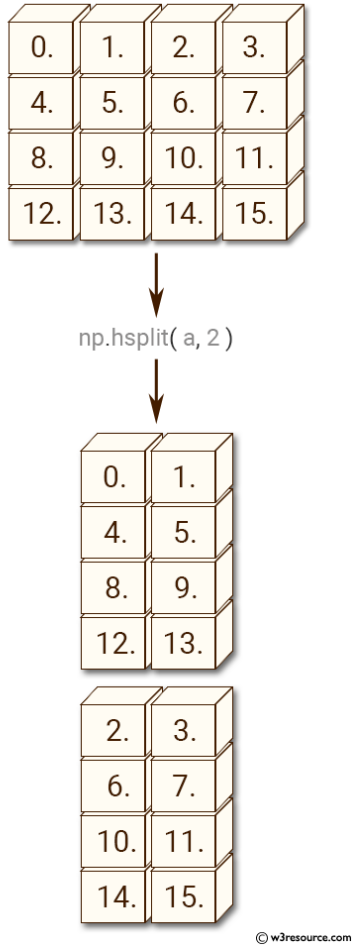

```python
import numpy as np
x = np.arange(1,10)
print(x)

#Tách vector x thành 3 vector
#tại các vị trí 2 và 6
x1, x2, x3 = np.split(x, [2,6])
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





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.fipud(A): Lật ngược ma trận A theo hàng.**

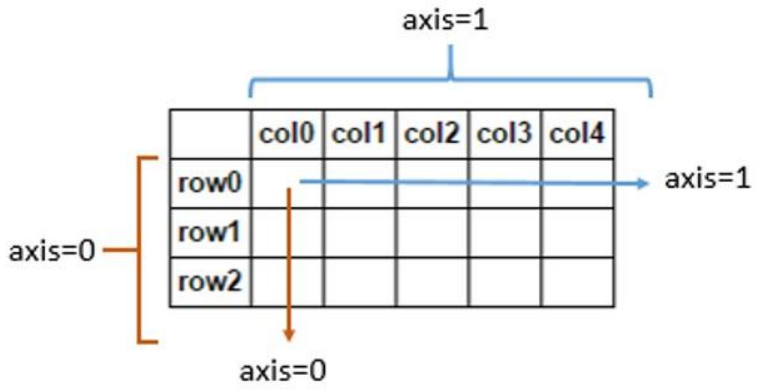
```
1  #Lật ma trận theo hàng
2  A2 = np.flip(A,0)
3  #Tương đương với
4  A2 = np.flipud(A)
5  print('Lật ma trận theo hàng: \n',A2)
```

```
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

Thực hành 1

Yêu cầu: Tạo một vector gồm 30 phần tử, có giá trị tăng dần từ 1 đến 30.

```
Vector a:
 [ 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 26 27 28 29 30]
```

Chỉ sử dụng các phương thức reshape, ravel, split…tách Vector a ở trên thành 3 vector con bao gồm:

- **a_le: chứa các phần tử là số lẻ;**
- **a_chan: chứa các phần tử là số chẵn.**
- **a_3: chứa các phần tử chia hết cho 3**

```
Vector a_le  : [ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29]
Vector a_chan: [ 2  4  6  8 10 12 14 16 18 20 22 24 26 28 30]
Vector a_3   : [ 3  6  9 12 15 18 21 24 27 30]
```

2. Tính toán với NumPy

2.1 Arithmetic operators.

The following table lists the arithmetic operators implemented in NumPy:

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

2.1 Arithmetic operators(2).

```python
import numpy as np
x = np.arange(8)
print("x     =", x)
print('-----------------------------------')
#Các phép toán đã giới thiệu trong buổi 01
print("x + 5  =", x + 5)
print("x - 5  =", x - 5)
print(" -x    =", -x)
print("x * 2  =", x * 2)
print("x / 2  =", x / 2)
print("x // 2 =", x // 2)
print("x % 2  =", x % 2)
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]
```

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

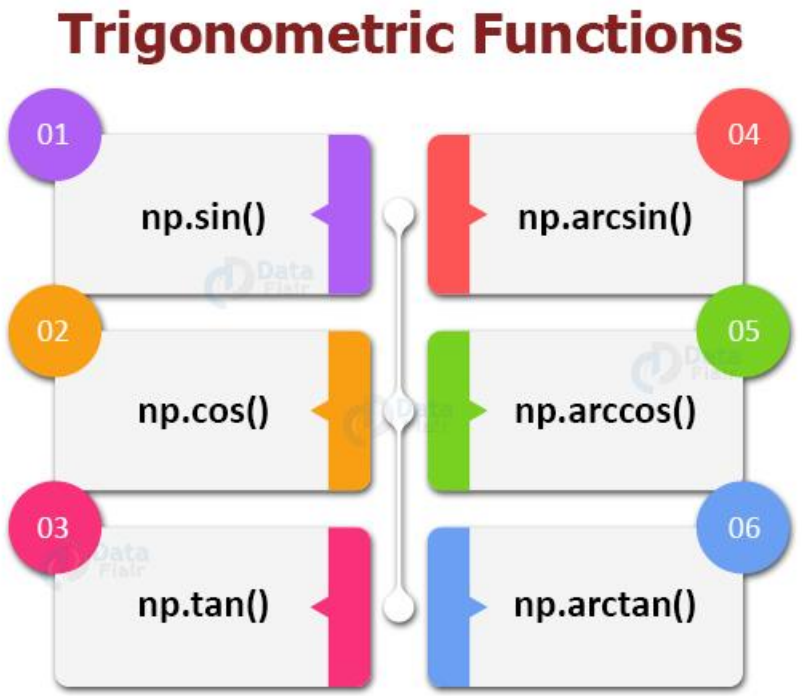
```
x      = [0 1 2 3 4 5 6 7]
--------------------------------
x + 5  = [ 5  6  7  8  9 10 11 12]
x - 5  = [-5 -4 -3 -2 -1  0  1  2]
-x     = [ 0 -1 -2 -3 -4 -5 -6 -7]
x * 2  = [ 0  2  4  6  8 10 12 14]
x / 2  = [0.  0.5 1.  1.5 2.  2.5 3.  3.5]
x // 2 = [0 0 1 1 2 2 3 3]
x % 2  = [0 1 0 1 0 1 0 1]
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]
```




```
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) =  [0.0000000e+00 1.0000000e+00 1.2246468e-16]
cos(theta) =  [ 1.000000e+00  6.123234e-17 -1.000000e+00]
tan(theta) =  [ 0.00000000e+00  1.63312394e+16 -1.22464680e-16]
```

2.3 Exponents and logarithms

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

2.3 Exponents and logarithms (2)

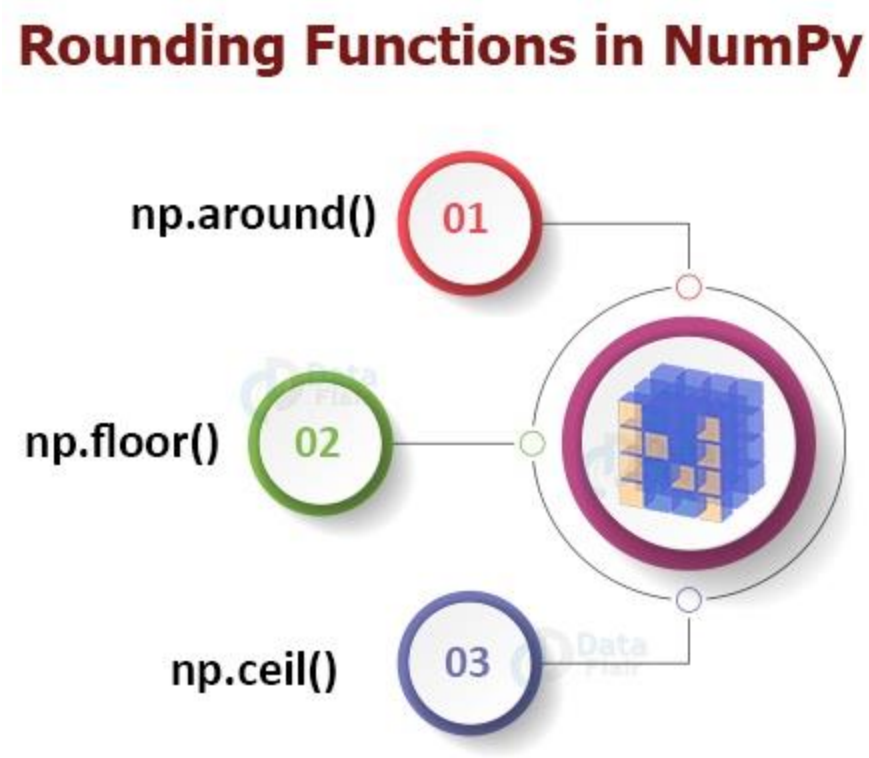
```
1  x = np.array([1, 2, 3])
2
3  print("x     =", x)
4  print('---------------------------')
5  print("e^x   =", np.exp(x))
6  print("2^x   =", np.exp2(x))
7  print("3^x   =", np.power(3, x))
```

```
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

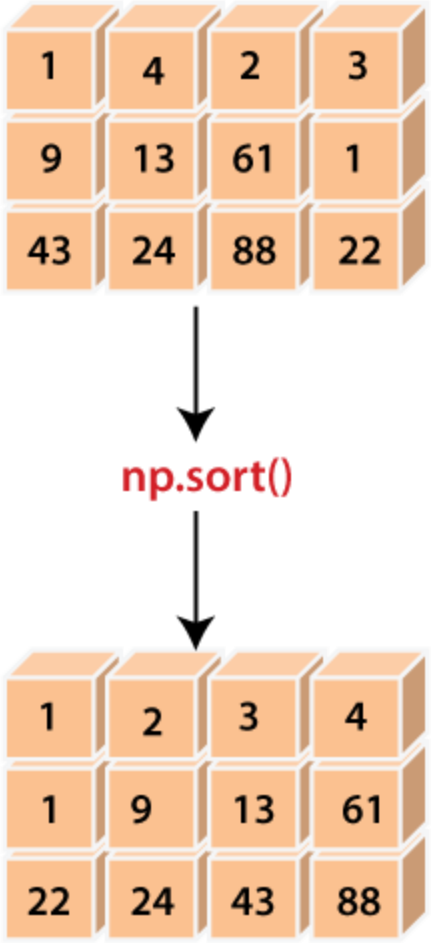


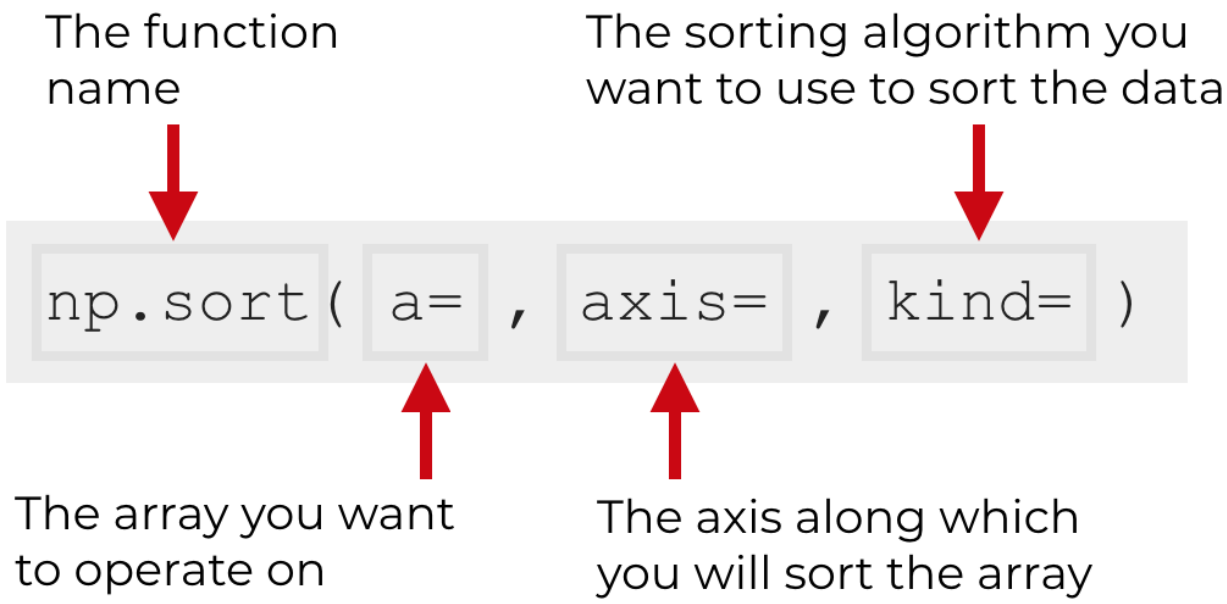

```python
import numpy as np
arr = np.array([20.8999,67.89899,54.43409])

print(arr)
print('-----------------------------------')
#1) Làm tròn tới 1 số sau dấu ,
print(np.around(arr,1))

#2)Làm tròn tới 2 số sau dấu ,
print(np.around(arr,2))

#3)Làm tròn xuống số nguyên gần nhất
print(np.floor(arr))

#4)Làm tròn lên số nguyên gần nhất
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)

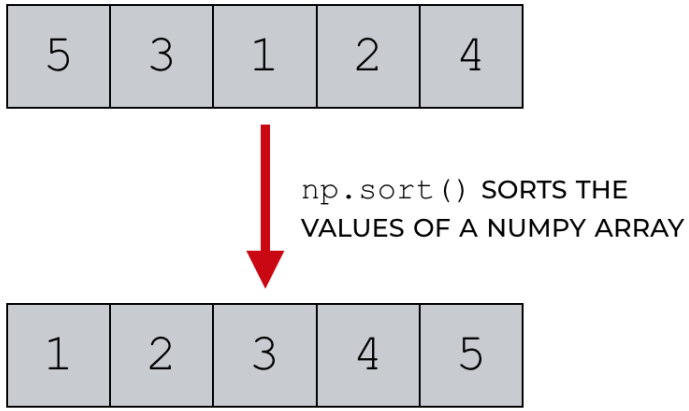




* kind='quicksort'- Default, 'mergesort', 'heapsort', 'stable'

3. Sắp xếp các phần tử trong mảng (2)

Sắp xếp các phần tử trong một Vector

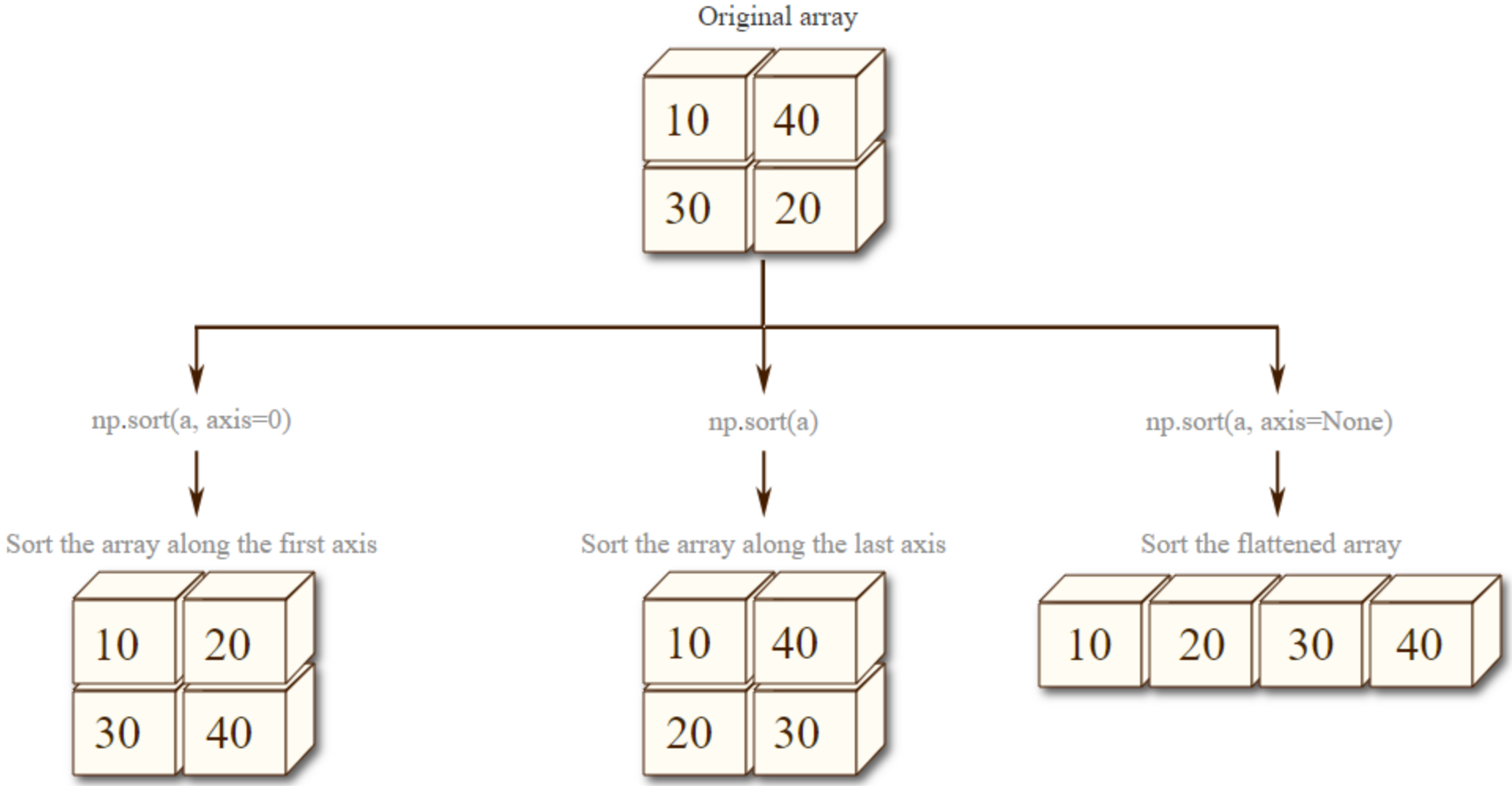



```python
#Sắp xếp các phần tử trong một vector
a = np.random.randint(1,33,15)

print('Vector ban đầu:\n', a)
print('--------------------------------')
#Sắp xếp vector a tăng dần
a_sort = np.sort(a)

#Sắp xếp vector a giảm dần:
#1) Lật vector a_sort để sắp xếp giảm dần
b_sort = np.flip(a_sort)
#2) sử dụng -np.sort(-x)
b_sort = -np.sort(-a)
print('Vector sắp xếp tăng dần: \n',a_sort)

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]
```

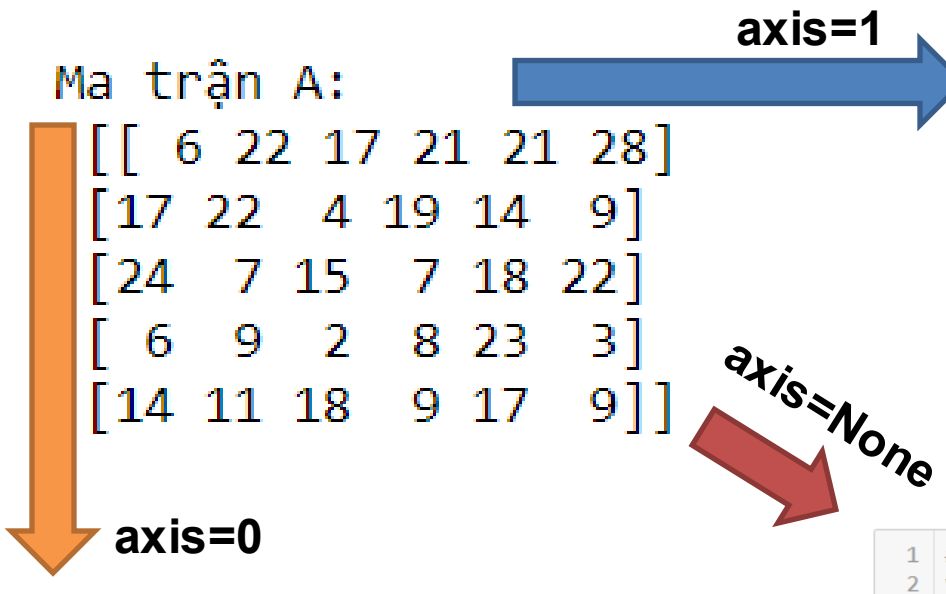
3. Sắp xếp các phần tử trong mảng (3)

Sắp xếp các phần tử trong một Ma trận:



3. Sắp xếp các phần tử trong mảng (4)

Sắp xếp các phần tử trong một Ma trận:




```
#Sắp xếp các phần tử của ma trận A
#2) Sắp xếp theo hàng axis=1 | Default
a_sort2 = np.sort(A,axis=1)
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) Sắp xếp theo cột axis=0
a_sort1 = np.sort(A,axis=0)
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]]
```

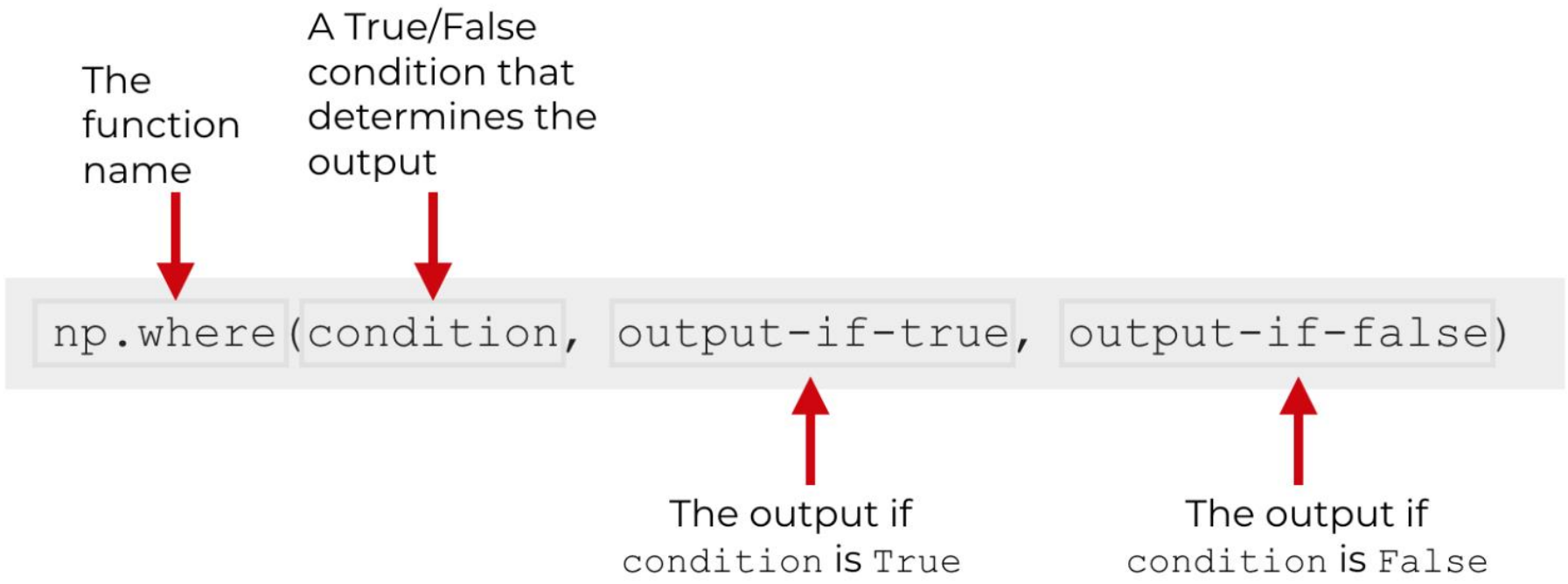
```
#3) Chuyển thành vector và sắp xếp các phần tử tăng dần theo hàng
v_sort = np.sort(A,axis=None)
print('Vector: \n', v_sort)

#Sắp xếp tất cả các phần tử theo thứ tự tăng dần theo hàng
a_sort3 = np.sort(A, axis=None).reshape(A.shape[0],A.shape[1])
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 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)

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



4. Tìm kiếm trong mảng: np.where (2)

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)

Vd1: Tìm kiếm trên ma trận:

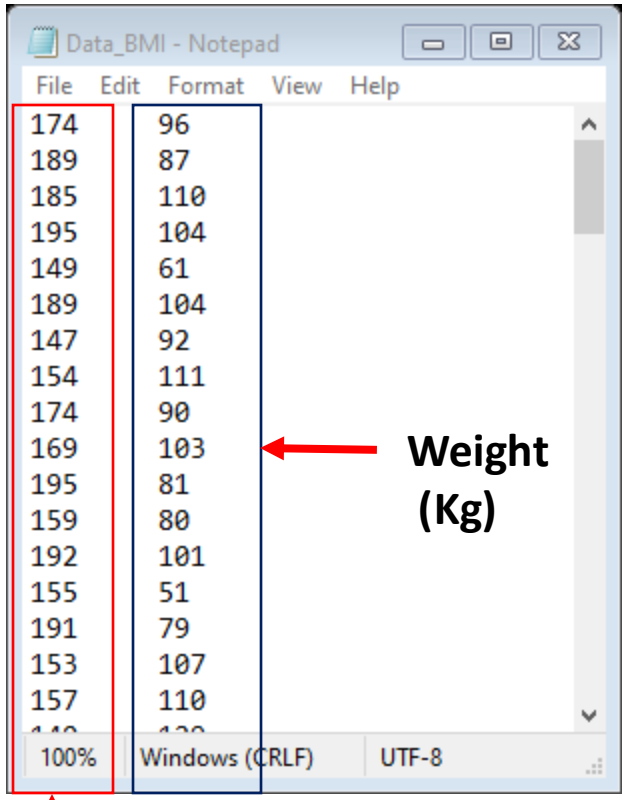
```
1  import numpy as np
2  #Tìm kiếm trên ma trận
3  arr = np.array([(1, 2, 3, 4, 5, 4, 4),
4                  (7, 3, 4, 8, 9, 6, 7)])
5  #Tìm kiếm phần tử > 4
6  x = np.where(arr >4)
7
8  print('Ma trận A: \n',arr)
9  print('----------------------------------------')
10 print(x)
11 print('Số phần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

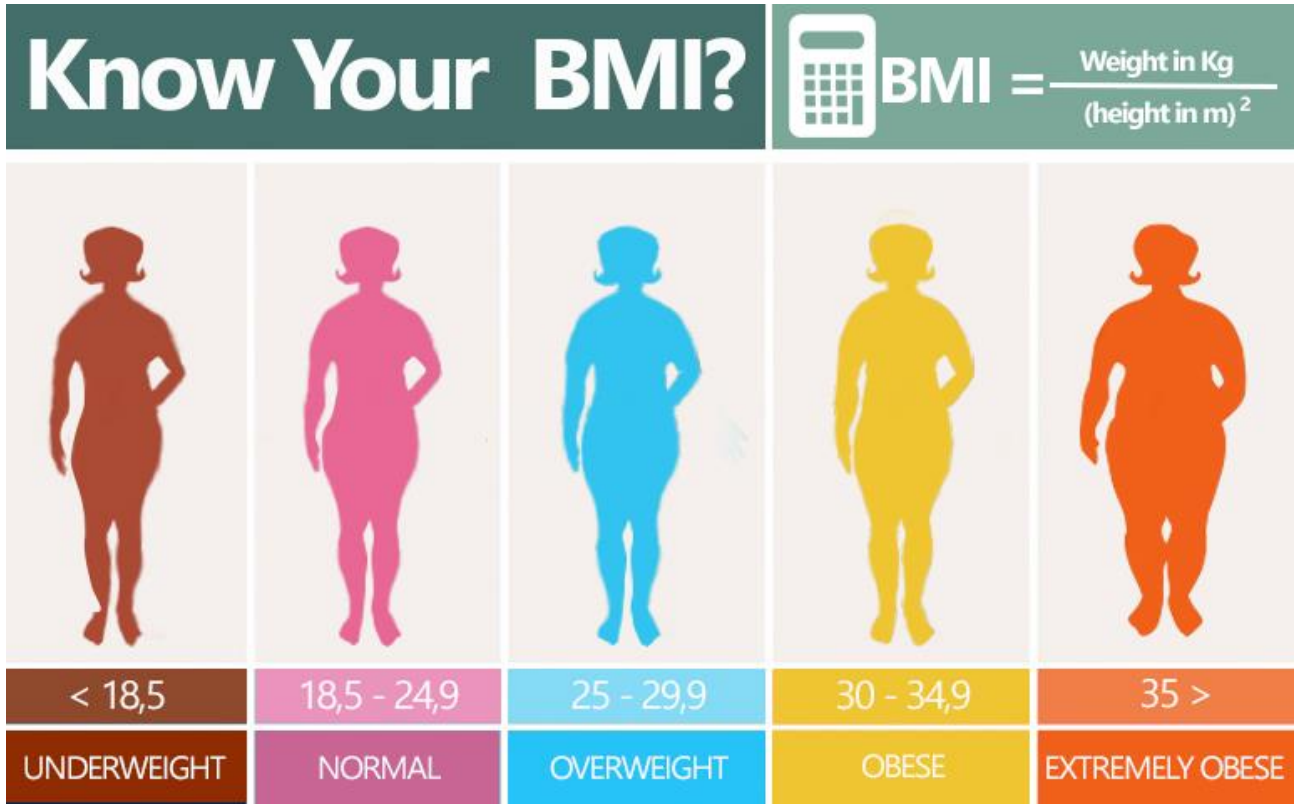
```
Ma trận A:
 [[1 2 3 4 5 4 4]
 [7 3 4 8 9 6 7]]
----------------------------------------
(array([0, 1, 1, 1, 1, 1], dtype=int64), array([4, 0, 3, 4, 5, 6], dtype=int64))
Số phần tử thỏa mãn điều kiện > 4: 6
```

Thực hành 2

Thực hành 2

File dữ liệu Data_BMI.txt lưu trữ thông tin chiều cao, cân nặng của 100 người.





Thực hành 2

Yêu cầu 2.1: Đọc dữ liệu từ file Data_BMI.txt vào 2 vector tương ứng.

- **v_height: chứa dữ liệu chiều cao**
- **v_weight: Chứa dữ liệu cân nặng**

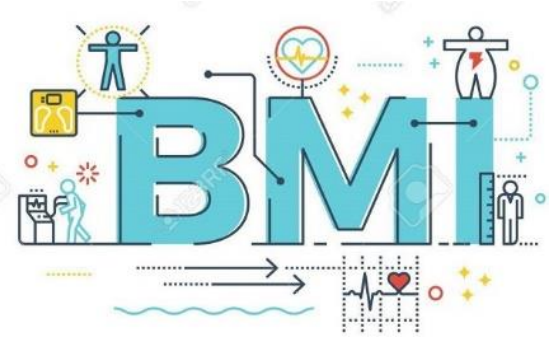
```
Vector chiều cao:
 [174 189 185 195 149 189 147 154 174 169 195 159 192 155 191 153 157 140
 144 172 157 153 169 185 172 151 190 187 163 179 153 178 195 160 157 189
 197 144 171 185 175 149 157 161 182 185 188 181 161 140 168 176 163 172
 196 187 172 178 164 143 191 141 193 190 175 179 172 168 164 194 153 178
 141 180 185 197 165 168 176 181 164 166 190 186 168 198 175 145 159 185
 178 183 194 177 197 170 142 160 195 190]
Vector cân nặng:
 [ 96  87 110 104  61 104  92 111  90 103  81  80 101  51  79 107 110 129
 145 139 110 149  97 139  67  64  95  62 159 152 121  52  65 131 153 132
 114  80 152  81 120 108  56 118 126  76 122 111  72 152 135  54 110 105
 116  89  92 127  70  88  54 143  54  83 135 158  96  59  82 136  51 117
  80  75 100 154 104  90 122  51  75 140 105 118 123  50 141 117 104 140
 154  96 111  61 119 156  69 139  69  50]
```

Thực hành 2

Yêu cầu 2.2: Tạo vector v_height_m2 từ vector v_height theo yêu cầu sau:

- **Chuyển đổi đơn vị của các phần tử từ cm sang m**
- **Tính bình phương giá trị các phần tử**

```
Vector v_height_m2:
 [3.0276 3.5721 3.4225 3.8025 2.2201 3.5721 2.1609 2.3716 3.0276 2.8561
 3.8025 2.5281 3.6864 2.4025 3.6481 2.3409 2.4649 1.96   2.0736 2.9584
 2.4649 2.3409 2.8561 3.4225 2.9584 2.2801 3.61   3.4969 2.6569 3.2041
 2.3409 3.1684 3.8025 2.56   2.4649 3.5721 3.8809 2.0736 2.9241 3.4225
 3.0625 2.2201 2.4649 2.5921 3.3124 3.4225 3.5344 3.2761 2.5921 1.96
 2.8224 3.0976 2.6569 2.9584 3.8416 3.4969 2.9584 3.1684 2.6896 2.0449
 3.6481 1.9881 3.7249 3.61   3.0625 3.2041 2.9584 2.8224 2.6896 3.7636
 2.3409 3.1684 1.9881 3.24   3.4225 3.8809 2.7225 2.8224 3.0976 3.2761
 2.6896 2.7556 3.61   3.4596 2.8224 3.9204 3.0625 2.1025 2.5281 3.4225
 3.1684 3.3489 3.7636 3.1329 3.8809 2.89   2.0164 2.56   3.8025 3.61   ]
```



Thực hành 2

Yêu cầu 2.3: **Tính chỉ số BMI của 100 người ngày theo công thức bên dưới, chỉ số BMI được làm tròn tới 1 số sau dấu “,”. Lưu kết quả vào vector v_bmi.**

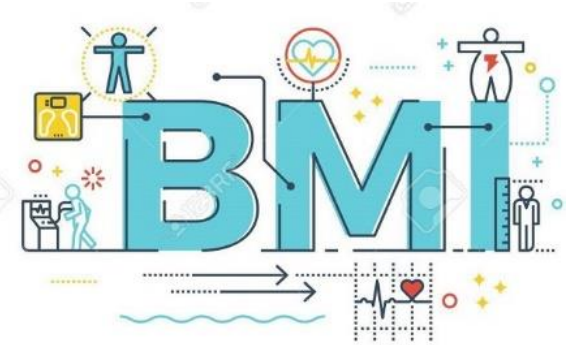
$$\text{BMI} = \frac{\text{Cân nặng (kg)}}{\text{Chiều cao x chiều cao (m)}}$$

```
Chỉ số BMI:
 [31.7 24.4 32.1 27.4 27.5 29.1 42.6 46.8 29.7 36.1 21.3 31.6 27.4 21.2
 21.7 45.7 44.6 65.8 69.9 47.  44.6 63.7 34.  40.6 22.6 28.1 26.3 17.7
 59.8 47.4 51.7 16.4 17.1 51.2 62.1 37.  29.4 38.6 52.  23.7 39.2 48.6
 22.7 45.5 38.  22.2 34.5 33.9 27.8 77.6 47.8 17.4 41.4 35.5 30.2 25.5
 31.1 40.1 26.  43.  14.8 71.9 14.5 23.  44.1 49.3 32.4 20.9 30.5 36.1
 21.8 36.9 40.2 23.1 29.2 39.7 38.2 31.9 39.4 15.6 27.9 50.8 29.1 34.1
 43.6 12.8 46.  55.6 41.1 40.9 48.6 28.7 29.5 19.5 30.7 54.  34.2 54.3
 18.1 13.9]
```

Thực hành 2

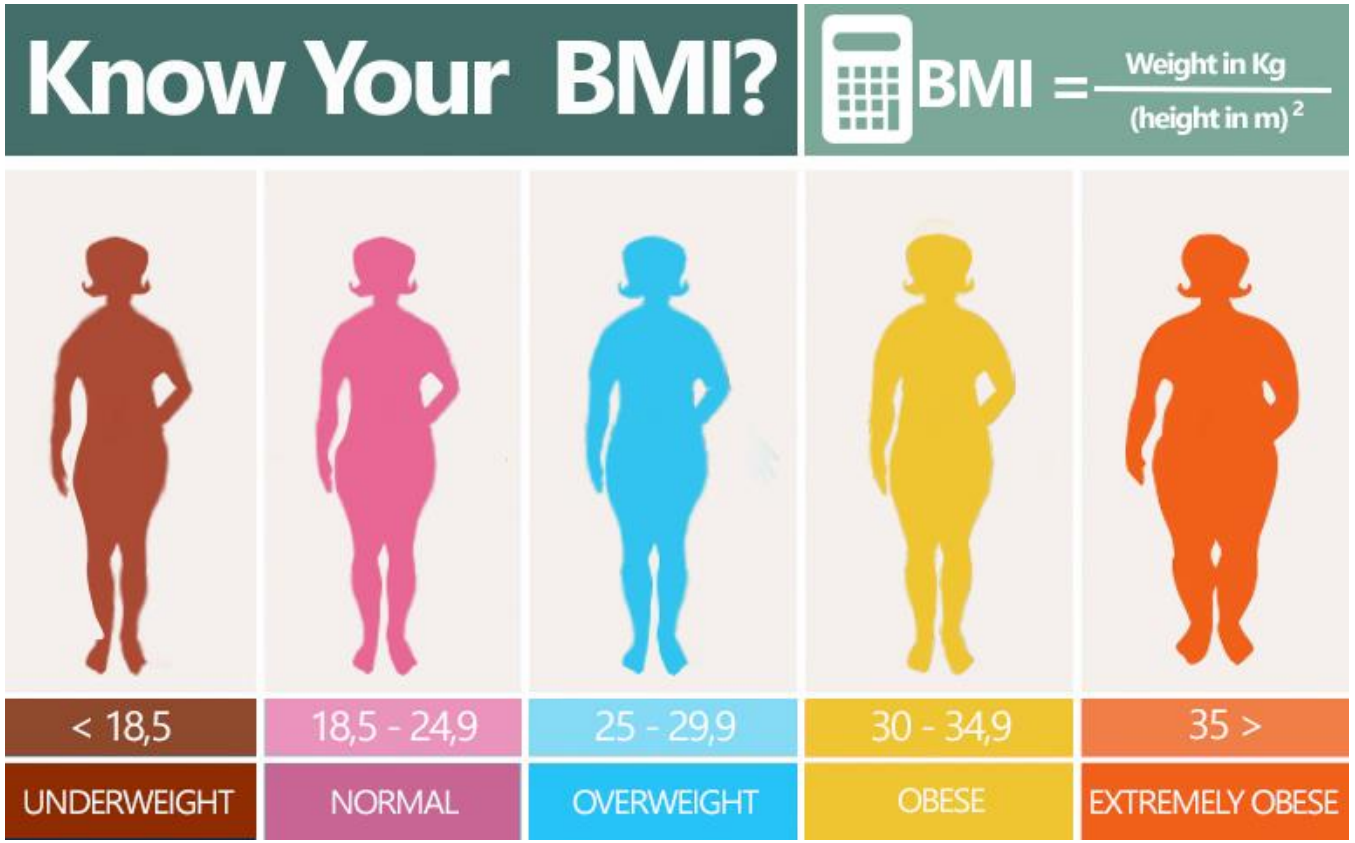
Yêu cầu 2.4: Sắp xếp thứ tự các phần tử trong vector v_bmi theo chiều tăng dần, giảm dần.

```
1. Sắp xếp theo thứ tự tăng dần:
 [12.8 13.9 14.5 14.8 15.6 16.4 17.1 17.4 17.7 18.1 19.5 20.9 21.2 21.3
 21.7 21.8 22.2 22.6 22.7 23.  23.1 23.7 24.4 25.5 26.  26.3 27.4 27.4
 27.5 27.8 27.9 28.1 28.7 29.1 29.1 29.2 29.4 29.5 29.7 30.2 30.5 30.7
 31.1 31.6 31.7 31.9 32.1 32.4 33.9 34.  34.1 34.2 34.5 35.5 36.1 36.1
 36.9 37.  38.  38.2 38.6 39.2 39.4 39.7 40.1 40.2 40.6 40.9 41.1 41.4
 42.6 43.  43.6 44.1 44.6 44.6 45.5 45.7 46.  46.8 47.  47.4 47.8 48.6
 48.6 49.3 50.8 51.2 51.7 52.  54.  54.3 55.6 59.8 62.1 63.7 65.8 69.9
 71.9 77.6]
2. Sắp xếp theo thứ tự giảm dần:
 [77.6 71.9 69.9 65.8 63.7 62.1 59.8 55.6 54.3 54.  52.  51.7 51.2 50.8
 49.3 48.6 48.6 47.8 47.4 47.  46.8 46.  45.7 45.5 44.6 44.6 44.1 43.6
 43.  42.6 41.4 41.1 40.9 40.6 40.2 40.1 39.7 39.4 39.2 38.6 38.2 38.
 37.  36.9 36.1 36.1 35.5 34.5 34.2 34.1 34.  33.9 32.4 32.1 31.9 31.7
 31.6 31.1 30.7 30.5 30.2 29.7 29.5 29.4 29.2 29.1 29.1 28.7 28.1 27.9
 27.8 27.5 27.4 27.4 26.3 26.  25.5 24.4 23.7 23.1 23.  22.7 22.6 22.2
 21.8 21.7 21.3 21.2 20.9 19.5 18.1 17.7 17.4 17.1 16.4 15.6 14.8 14.5
 13.9 12.8]
```



Thực hành 2

Yêu cầu 2.5: Thống kê số lượng người theo từng mức dựa vào phân loại theo hình.




```
Tổng số:  100
-----------------------------
1. Underweight    :  10
2. Normal         :  13
3. Overweight     :  16
4. Obese          :  14
5. Extremely Bbese:  47
```

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

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

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

$$detA = \begin{vmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{m,n} \end{vmatrix}$$

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

$$\det\begin{bmatrix} a \end{bmatrix} = a$$

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

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

5.1 Định thức det (A)

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) =  -3.999999999999999
```

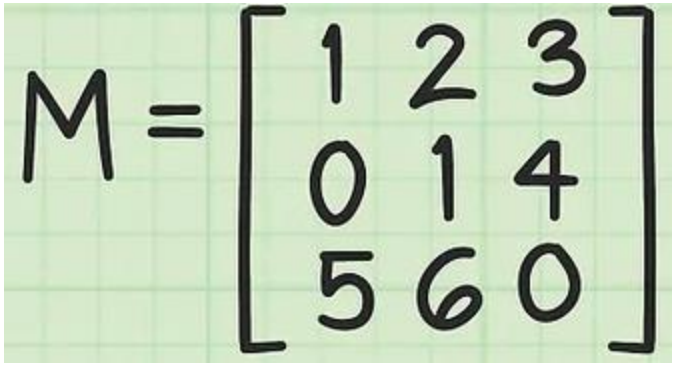
```
1  import numpy as np
2  b =np.array([( 1, 2, 3, 4),
3               (-2,-1, 4, 1),
4               ( 3,-4,-5, 6),
5               ( 1, 2, 3, 4)])
6  print('Ma trận b:\n',b)
7  det_b =  np.linalg.det(b)
8  print('det(b) = ', det_b)
```

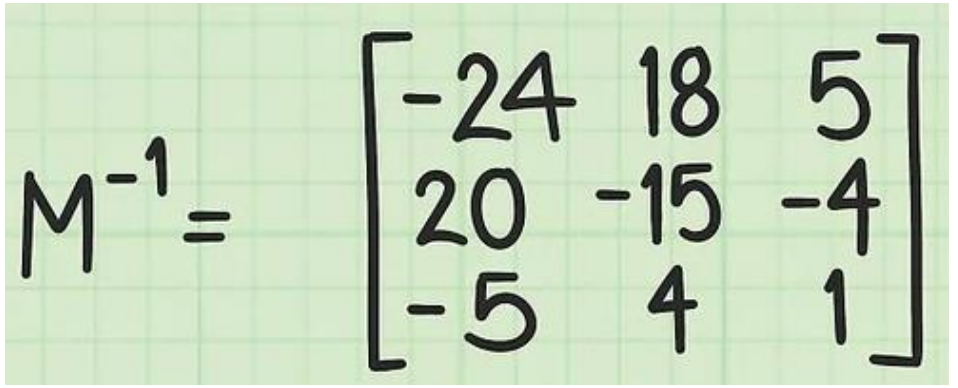
```
Ma trận b:
 [[ 1  2  3  4]
 [-2 -1  4  1]
 [ 3 -4 -5  6]
 [ 1  2  3  4]]
det(b) =  0.0
```

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)





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

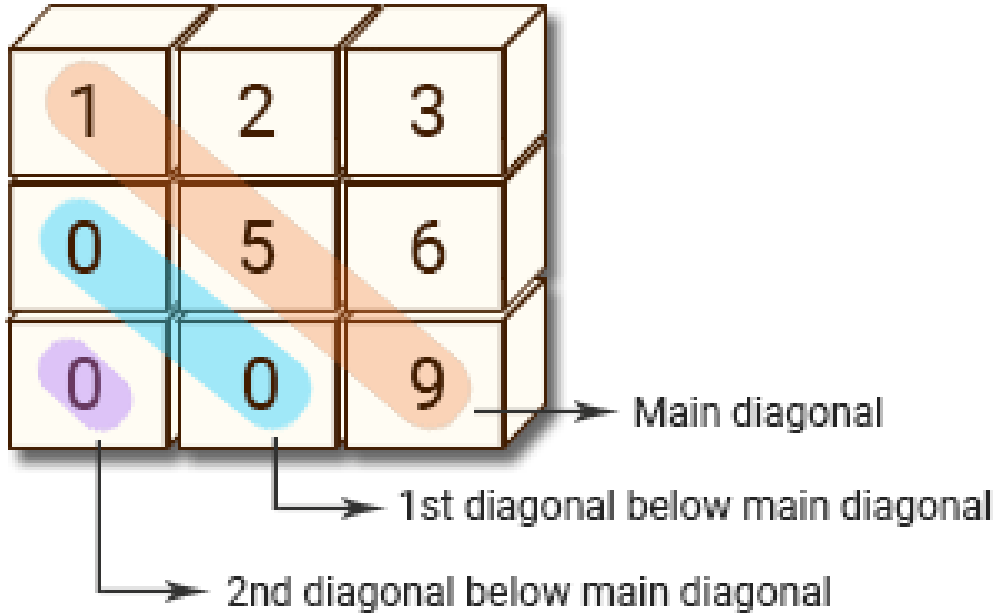
det(m) = 0: Không tồn tại ma trận nghịch đảo

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]
```

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

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

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

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

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

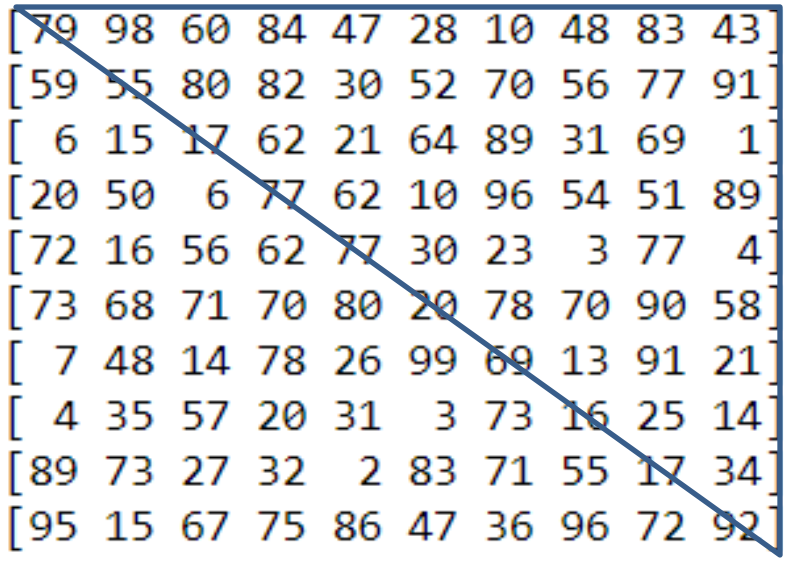
```
[72 68 14 20  2 47]
```

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




```
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)

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

```python
#Tạo ma trận là các phần tử trên đường chéo chính
#của ma trận vuông A cách đường chéo chính
#về phía dưới 3 đường
d_A1 = np.triu(A,-3)
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 $\operatorname{trace} T$ equals the sum of the diagonal entries of that matrix.

Definition: *trace of a matrix*

The *trace* of a square matrix A, denoted $\operatorname{trace} A$, 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

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

Trace of AB equals trace of BA

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

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

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

```
Trace of Matrix A:  519
```

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

```
Trace of Matrix A:  519
```

Thực hành 3

Thực hành 3

Yêu cầu 3.1: Sử dụng vector_weigth trong bài thực hành 2 chuyển về ma trận kích thước 10 x 10.

```
Vector Weight:
 [ 96  87 110 104  61 104  92 111  90 103  81  80 101  51  79 107 110 129
 145 139 110 149  97 139  67  64  95  62 159 152 121  52  65 131 153 132
 114  80 152  81 120 108  56 118 126  76 122 111  72 152 135  54 110 105
 116  89  92 127  70  88  54 143  54  83 135 158  96  59  82 136  51 117
  80  75 100 154 104  90 122  51  75 140 105 118 123  50 141 117 104 140
 154  96 111  61 119 156  69 139  69  50]
----------------------------------------------------------------
Ma trận Weight:
 [[ 96  87 110 104  61 104  92 111  90 103]
 [ 81  80 101  51  79 107 110 129 145 139]
 [110 149  97 139  67  64  95  62 159 152]
 [121  52  65 131 153 132 114  80 152  81]
 [120 108  56 118 126  76 122 111  72 152]
 [135  54 110 105 116  89  92 127  70  88]
 [ 54 143  54  83 135 158  96  59  82 136]
 [ 51 117  80  75 100 154 104  90 122  51]
 [ 75 140 105 118 123  50 141 117 104 140]
 [154  96 111  61 119 156  69 139  69  50]]
```

Thực hành 3

Yêu cầu 3.2: Cho biết ma trận weight có tồn tại ma trân nghịch đảo không? Nếu có hãy xác định ma trận weight^{-1} ?

Yêu cầu 3.3: Tạo vector_diagonal chứa các phần tử trên đường chéo chính của ma trận weight, tính trace của ma trận weight.

```
a) Đường chéo chính của ma trận weight:
 [ 96  80  97 131 126  89  96  90 104  50]

b)Trace của ma trận weight: 959
```

Thực hành 3

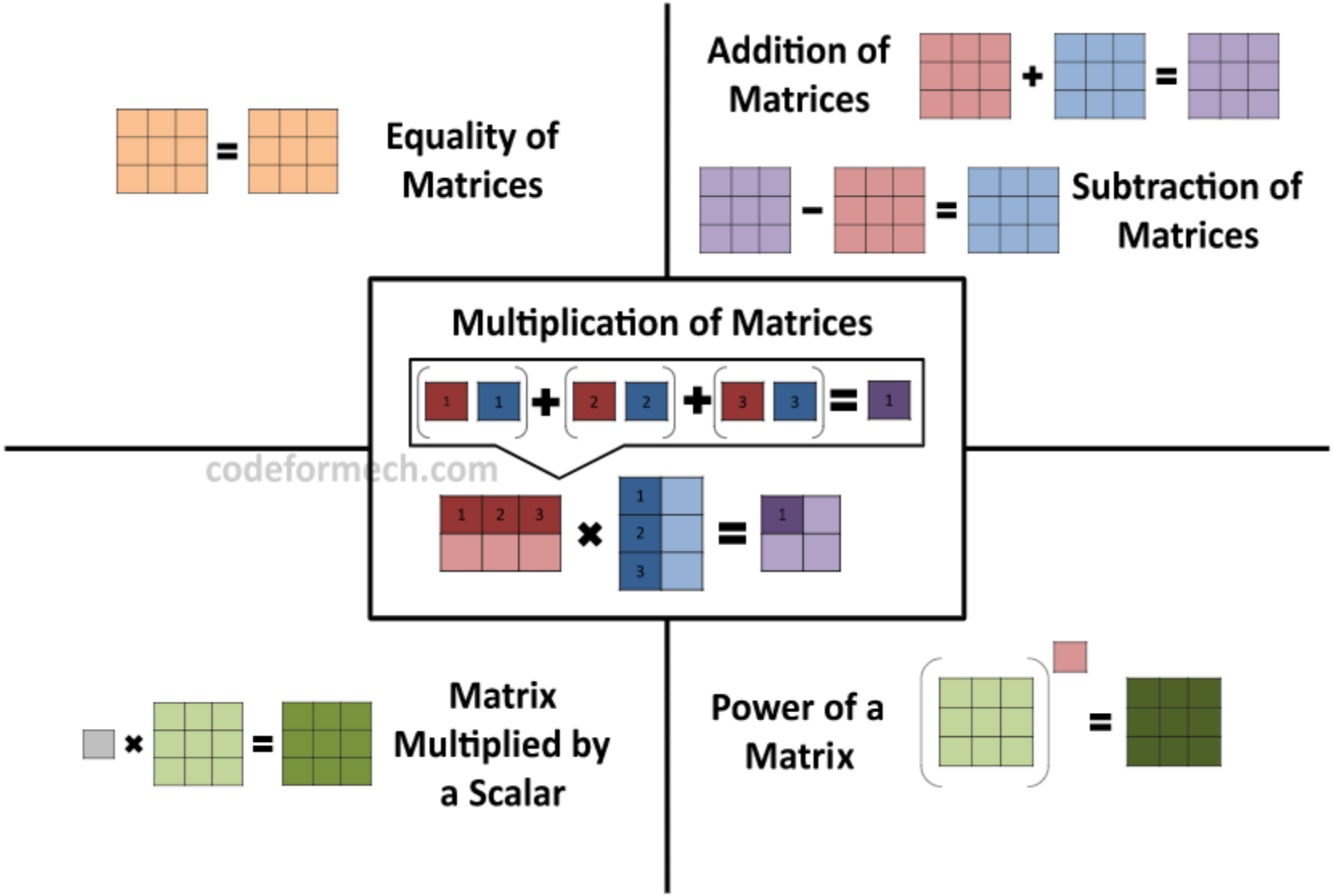
Yêu cầu 3.4: Tìm giá trị lớn nhất, nhỏ nhất của ma trận đường chéo trên và ma trận đường chéo dưới không bao gồm các phần tử nằm trên đường chéo chính của ma trận weight?

```
[[  0  87 110 104  61 104  92 111  90 103]
 [  0   0 101  51  79 107 110 129 145 139]
 [  0   0   0 139  67  64  95  62 159 152]
 [  0   0   0   0 153 132 114  80 152  81]
 [  0   0   0   0   0  76 122 111  72 152]
 [  0   0   0   0   0   0  92 127  70  88]
 [  0   0   0   0   0   0   0  59  82 136]
 [  0   0   0   0   0   0   0   0 122  51]
 [  0   0   0   0   0   0   0   0   0 140]
 [  0   0   0   0   0   0   0   0   0   0]]
Phần tử max:  159
```

```
[[  0   0   0   0   0   0   0   0   0   0]
 [ 81   0   0   0   0   0   0   0   0   0]
 [110 149   0   0   0   0   0   0   0   0]
 [121  52  65   0   0   0   0   0   0   0]
 [120 108  56 118   0   0   0   0   0   0]
 [135  54 110 105 116   0   0   0   0   0]
 [ 54 143  54  83 135 158   0   0   0   0]
 [ 51 117  80  75 100 154 104   0   0   0]
 [ 75 140 105 118 123  50 141 117   0   0]
 [154  96 111  61 119 156  69 139  69   0]]
Phần tử max:  158
```

6.Phép toán với 2 ma trận

6. Phép toán trên 2 ma trận



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.suptract(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)
```

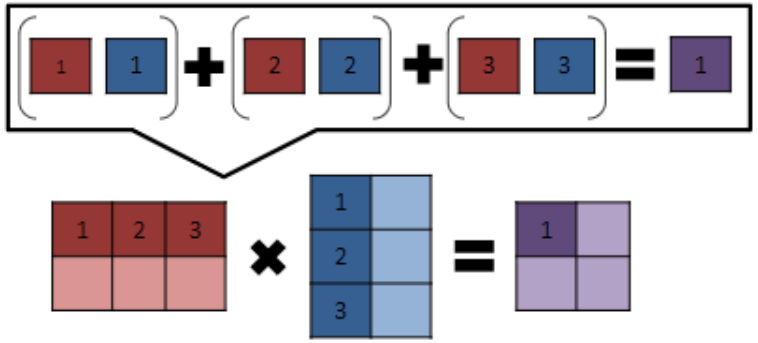
```
[[  3   0  10 -11  14]
 [ 12   5 -10  -5   4]
 [ 16   2  -1  -2  11]]
```

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)



Matrix Multiplication

```
#Tích của 2 vector:
vector_a = np.random.randint(1,20,10)
vector_b = np.random.randint(1,20,10)
#Thực hiện tính tích của 2 vector
#Kết quả trả về một số
vector_ab = vector_a @ vector_b

print('Vector a:\n',vector_a)
print('Vector b:\n',vector_b)
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]]
```

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

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

Thực hành 4

Thực hành 4

Yêu cầu: Chuyển vector_height, vector_weigth trong bài thực hành 1 về ma trận height, weight kích thước 10 x 10.

Thực hiện các phép toán: So sánh; Cộng; Trừ; Nhân hai ma trận height và weight

```
Ma trận height:
 [[174 189 185 195 149 189 147 154 174 169]
 [195 159 192 155 191 153 157 140 144 172]
 [157 153 169 185 172 151 190 187 163 179]
 [153 178 195 160 157 189 197 144 171 185]
 [175 149 157 161 182 185 188 181 161 140]
 [168 176 163 172 196 187 172 178 164 143]
 [191 141 193 190 175 179 172 168 164 194]
 [153 178 141 180 185 197 165 168 176 181]
 [164 166 190 186 168 198 175 145 159 185]
 [178 183 194 177 197 170 142 160 195 190]]
```

```
Ma trận weight:
 [[ 96  87 110 104  61 104  92 111  90 103]
 [ 81  80 101  51  79 107 110 129 145 139]
 [110 149  97 139  67  64  95  62 159 152]
 [121  52  65 131 153 132 114  80 152  81]
 [120 108  56 118 126  76 122 111  72 152]
 [135  54 110 105 116  89  92 127  70  88]
 [ 54 143  54  83 135 158  96  59  82 136]
 [ 51 117  80  75 100 154 104  90 122  51]
 [ 75 140 105 118 123  50 141 117 104 140]
 [154  96 111  61 119 156  69 139  69  50]]
```

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 rank(A) hoặc r(A)

+ Ma trận 0 có hạng bằng 0

+ Ma trận A cấp m x 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 rank(A)=2.

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

```python
#Hạng của ma trận 0
A_0 = np.zeros((4,5))
print(A_0)
rank = np.linalg.matrix_rank(A_0)
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
```

```
Ma trận B1:
 [[ 1  2  3  4]
 [-2 -1  4  1]
 [ 3 -4 -5  6]
 [ 1  2  3  4]]
det(B1) =  0.0
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
Rank(B2) =  4
```

7.2 Ma trận chuyển vị

Chuyển vị của ma trận *m*x*n* **A** là ma trận *n*x*m* $\mathbf{A}^T$ tạo ra bằng cách chuyển hàng thành cột và cột thành hàng:

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

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

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

Thực hành 5

Yêu cầu 5.1: Học viên tìm ma trận nghịch đảo và hạng của ma trận Height, Weight trong bài thực hành số 3:

```
Ma trận nghịch đảo height.T:
 [[174 195 157 153 175 168 191 153 164 178]
 [189 159 153 178 149 176 141 178 166 183]
 [185 192 169 195 157 163 193 141 190 194]
 [195 155 185 160 161 172 190 180 186 177]
 [149 191 172 157 182 196 175 185 168 197]
 [189 153 151 189 185 187 179 197 198 170]
 [147 157 190 197 188 172 172 165 175 142]
 [154 140 187 144 181 178 168 168 145 160]
 [174 144 163 171 161 164 164 176 159 195]
 [169 172 179 185 140 143 194 181 185 190]]
Hạng của ma trận height:  10
```

```
Ma trận nghịch đảo weight.T:
 [[ 96  81 110 121 120 135  54  51  75 154]
 [ 87  80 149  52 108  54 143 117 140  96]
 [110 101  97  65  56 110  54  80 105 111]
 [104  51 139 131 118 105  83  75 118  61]
 [ 61  79  67 153 126 116 135 100 123 119]
 [104 107  64 132  76  89 158 154  50 156]
 [ 92 110  95 114 122  92  96 104 141  69]
 [111 129  62  80 111 127  59  90 117 139]
 [ 90 145 159 152  72  70  82 122 104  69]
 [103 139 152  81 152  88 136  51 140  50]]
Hạng của ma trận weight:  10
```

Thực hành 5

Yêu cầu 5.2: Học viên thực hành và trả lời các câu hỏi sau:

1. The rank of matrix $A=\begin{bmatrix}1 & 2 & 3\\ 2 & 2 & 2\\ 3 & 3 & 3\end{bmatrix}$ is equal to

   A) 4 B) 3 C) 2 D) 1

2. If $A=\begin{bmatrix}1 & 2 & 3\\ 3 & 4 & 5\\ 4 & 5 & 6\end{bmatrix}$ and det(A)=0 then rank of a matrix A is

   A) Greater than or equal to 3
   B) Strictly less than 3
   C) Less than or equal to 3
   D) Strictly greater than 3 .

3. The rank of matrix $\begin{bmatrix}10 & 10 & 10 & 10 & 10\\ 10 & 10 & 10 & 10 & 10\\ 10 & 10 & 10 & 10 & 10\end{bmatrix}$ is

   A) 10 B) 5 C)2 D)1.

Thực hành 5

4 For matrix A of order mxn, the rank r of matrix A is

A) $r \geq \min\{m,n\}$ B) $r \geq \max\{m,n\}$

C) $r \leq \min\{m,n\}$ D) $r \leq \max\{m,n\}$

5 A 5×7 matrix has all its entries equal to -1 , then rank of matrix is

A)7 B) 5 C) 1 D) zero

6 The rank of the following matrix by determinant method $\begin{bmatrix} 2 & 3 & 4 \\ 4 & 3 & 1 \\ 1 & 2 & 4 \end{bmatrix}$ is

A)2 B) 3 C) 1 D) 0

Q & A
Thank you!