AI VIET NAM

Exercise 6

Contents

1. Giới thiệu các hàm trong numpy để giải Excercise 6

Problem 1

- Kiểm tra version numpy
  - __version__ : hầu hết các modules đều có hỗ trợ method này
  - importlib_metadata

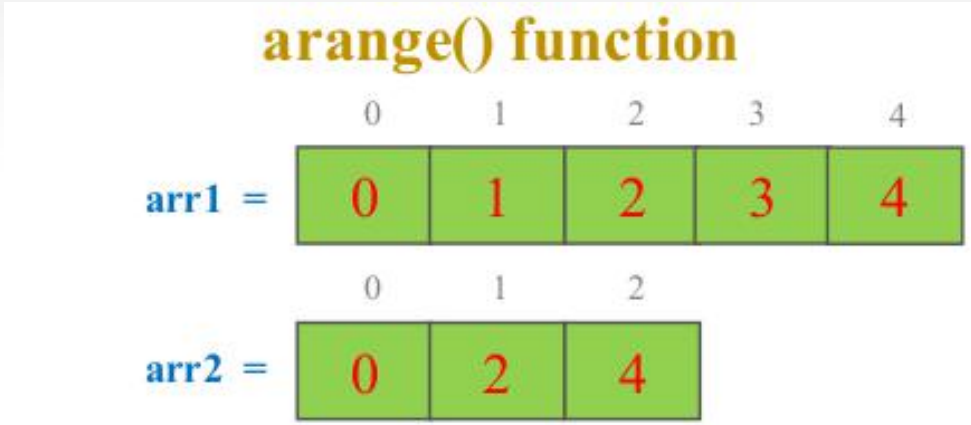
```
# python 3.8+
from importlib_metadata import version
print(version('numpy'))
```

  - pkg_resources

```
#python below 3.8
import pkg_resources
print(pkg_resources.get_distribution('numpy').version)
```

Problem 2

- Tạo mảng một chiều 0-9
  - np.arange(start, stop, step)
  - range [start, stop)
  - start: mặc định là 0
  - step: mặc định là 1
  - **np.arange(0, 10)**

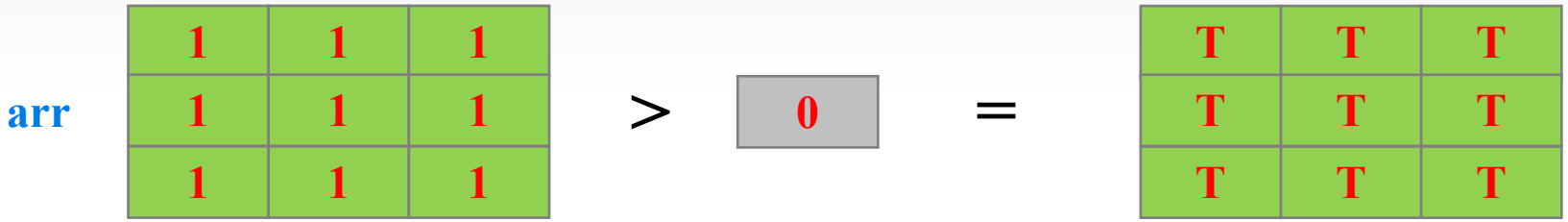



```python
# aivietnam.ai
import numpy as np

# np.arange(start=0, stop, step=1)
arr1 = np.arange(5)
print(arr1)

arr2 = np.arange(0, 5, 2)
print(arr2)
```

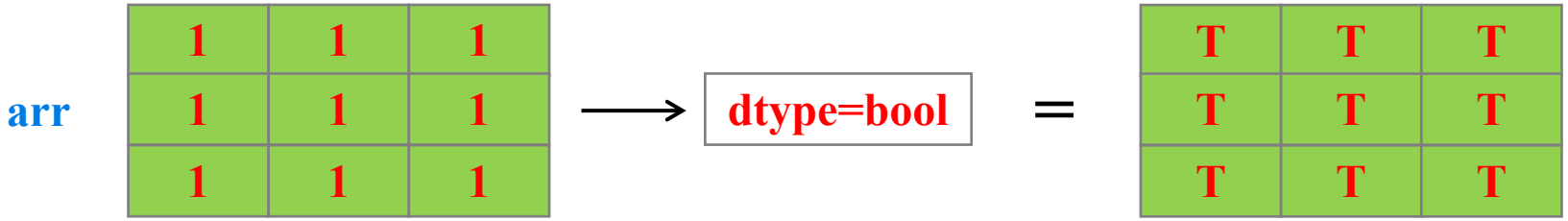
```
[0 1 2 3 4]
[0 2 4]
```

Problem 3

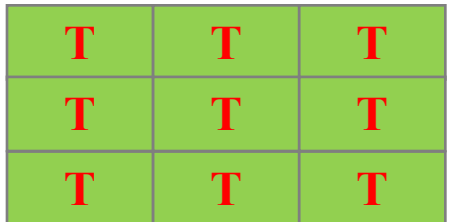
- Tạo mảng boolean **3x3** với giá trị là **True**
  - Cách 1: Các phép toán tử trên array sẽ được sử dụng theo **elmentwise**



  - Cách 2: Dùng array ones và dtype

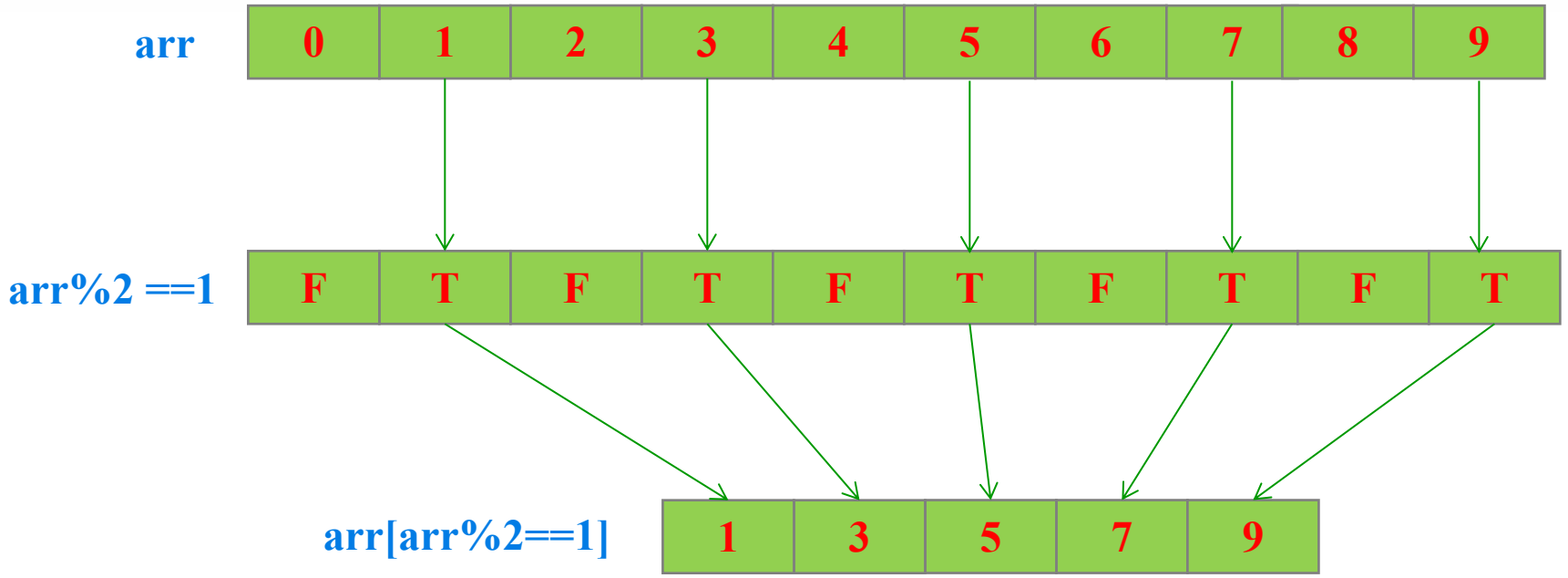


  - Cách 3: Sử dụng numpy.full(shape=(3,3), fill_value=True, dtype=bool)



Problem 4

- Lấy các giá trị của mảng là số lẻ



– Những vị trí có giá trị True sẽ được trả về

Problem 5

- Thay thế elements có các giá trị của mảng là số lẻ bằng -1

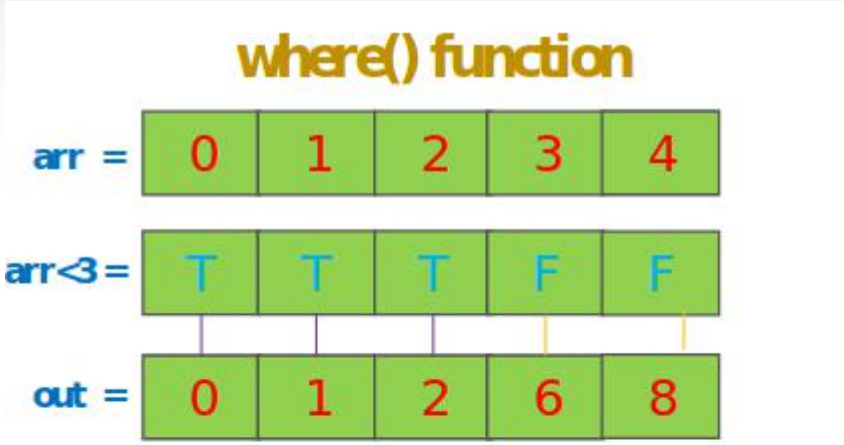
arr	0	1	2	3	4	5	6	7	8	9
arr%2 ==1	F	T	F	T	F	T	F	T	F	T
arr[arr%2==1] = -1	0	-1	2	-1	4	-1	6	-1	8	-1

  - Những vị trí có giá trị True sẽ được thay thế bằng -1

Problem 6

- Thay thế elements có các giá trị của mảng là số lẻ bằng -1, nhưng **không thay đổi mảng ban đầu**
  - Cách 1: thực hiện copy array trước khi thay giá trị
  - Cách 2: np.where(condition, x, y)
  - condition đúng trả về x
  - condition sai trả về y
  - np.where(arr%2==1, -1, arr)

```
# nếu array là 1D
def np_where(condition, x, y):
    new_list = []
    for c, xv, yv in zip(condition, x, y):
        if c:
            new_list.append(xv)
        else:
            new_list.append(yv)
    return np.array(new_list)
```

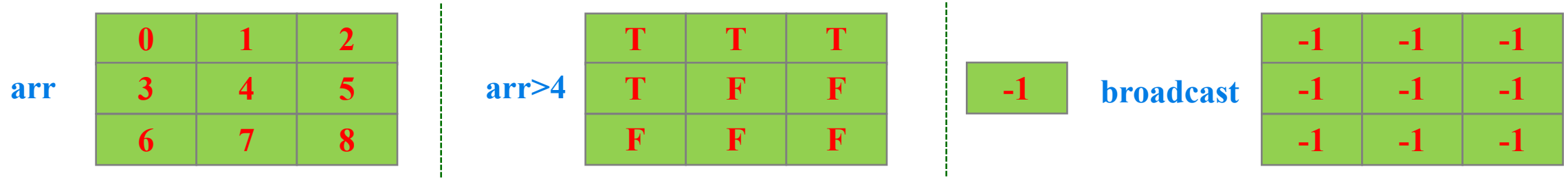



```
1  # aivietnam.ai
2  import numpy as np
3
4  # create an array
5  arr = np.arange(5)
6  print(arr)
7
8  # condition
9  condition = arr < 3
10 out = np.where(condition, arr, arr*2)
11
12 print(condition)
13 print(out)
```

```
[0 1 2 3 4]
[ True  True  True False False]
[0 1 2 6 8]
```

Problem 6

- Thay thế elements có các giá trị của mảng là số lẻ bằng -1, nhưng **không thay đổi mảng ban đầu**
  - Cách 1: thực hiện copy array trước khi thay giá trị
  - Cách 2: np.where(condition, x, y)




```
1 arr = np.array([[0,1,2],
2                 [3,4,5],
3                 [6,7,8]])
4 # vị trí nào có giá trị < 4 thay bằng -1
5 out = np.where(arr < 4, -1, arr)
6 print(out)
```

```
[[-1 -1 -1]
 [-1  4  5]
 [ 6  7  8]]
```

Problem 6

- Thay thế elements có các giá trị của mảng là số lẻ bằng -1, nhưng **không thay đổi mảng ban đầu**
  - Cách 1: thực hiện copy array trước khi thay giá trị
  - Cách 2: np.where(condition) trả về index

arr

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

arr>4

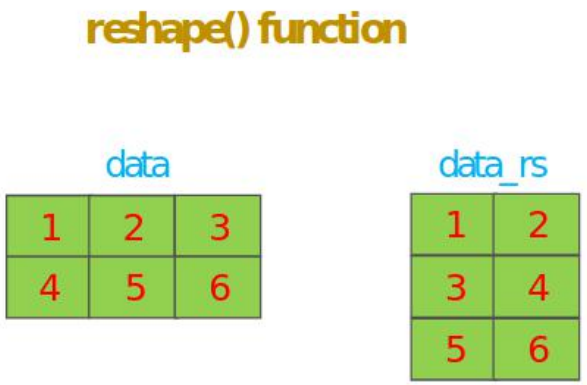
	0	1	2
0	T	T	T
1	T	F	F
2	F	F	F

```
1 arr = np.array([[0,1,2],
2                 [3,4,5],
3                 [6,7,8]])
4 # Trả về vị trí nào có giá trị < 4 theo Dim của input
5 out = np.where(arr < 4)
6 print(out)
```

```
(array([0, 0, 0, 1]), array([0, 1, 2, 0]))
```

Problem 7

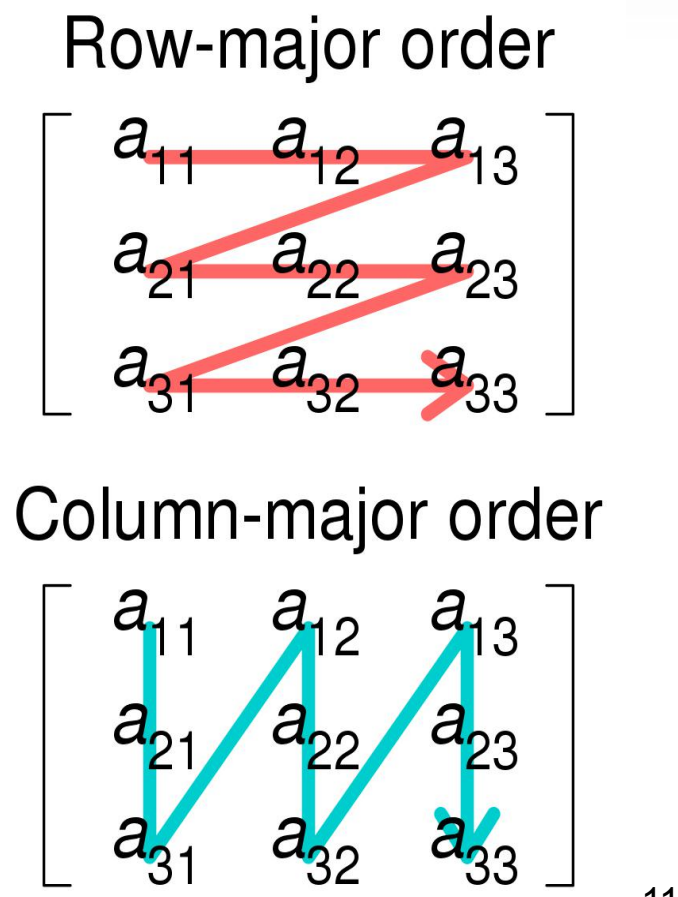
- Chuyển mảng 1 chiều thành mảng **2 chiều có 2 hàng** (dòng)
  - Sử dụng np.reshape(a, newshape)
  - **np.rshape((2,-1))**




```python
# aivietnam.ai
import numpy as np

# tạo list
l = [[1,2,3],
     [4,5,6]]

# tạo ndarray
data = np.array(l)
print('data\n', data)
print('data shape\n', data.shape)

# reshape
data_rs = np.reshape(data, (3,2))
print('data_rs\n', data_rs)
print('data_rs shape\n', data_rs.shape)
```

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

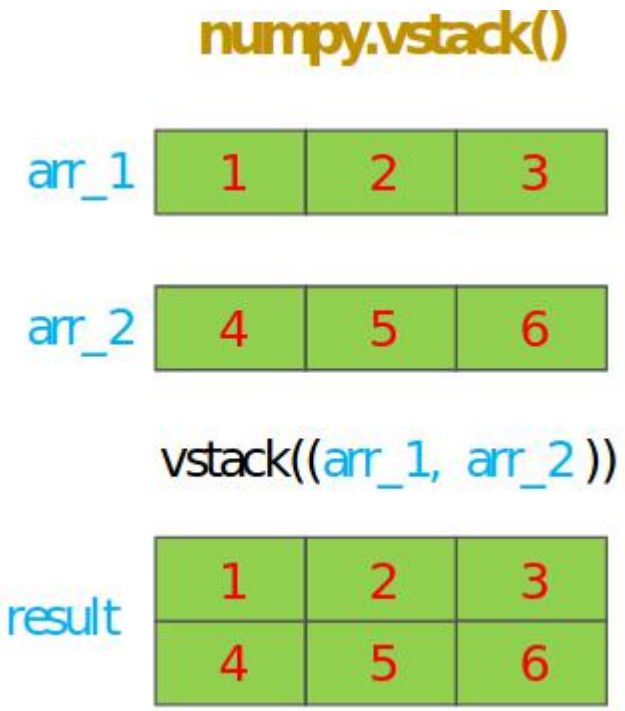
(*)



(*) https://en.wikipedia.org/wiki/Row-_and_column-major_order#/media/File:Row_and_column_major_order.svg

Problem 8

- Xếp chồng 2 mảng theo chiều dọc
  - np.vstack()
  - np.concatenate()
  - np.r_[]: concatenate theo axis thứ nhất, thuần Python code không nhanh như **concatenate**



Problem 8

- np.r_[]: thuần Python code không nhanh như **concatenate**
  - Có thể merge scalar và vector trực tiếp

```
1  np.r_[0.0, np.array([1,2,3,4]), 0.0]
```

```
array([0., 1., 2., 3., 4., 0.])
```

```
1 np.concatenate([[0.0], np.array([1,2,3,4]), [0.0]])
```

```
array([0., 1., 2., 3., 4., 0.])
```

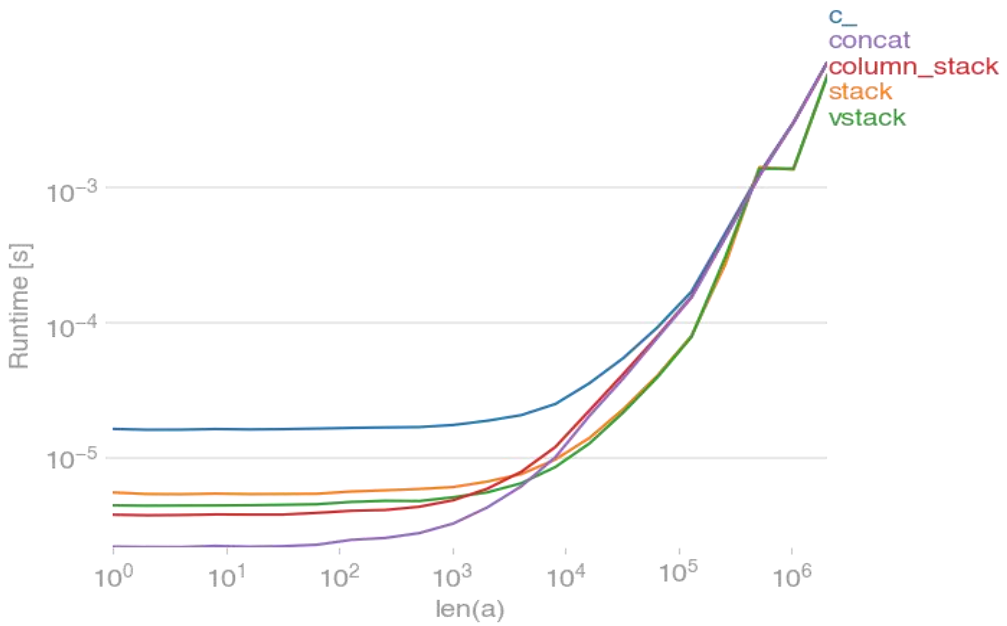
  - Khai báo range nhanh

```
1 np.r_[0.0, 1:5, 0.0]
```

```
array([0., 1., 2., 3., 4., 0.])
```

```
1 np.concatenate([0.0, np.array([1,2,3,4]), 0.0])
```

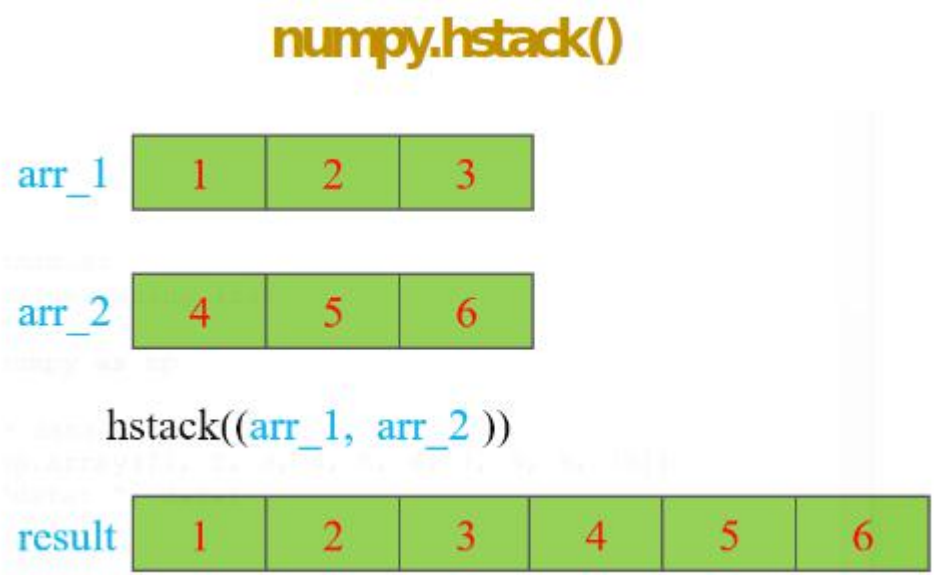
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call
<ipython-input-73-93cb22dfd5b8> in <module>()
----> 1 np.concatenate([0.0, np.array([1,2,3,4]), 0.0])

<__array_function__ internals> in concatenate(*args, **kwargs)

ValueError: zero-dimensional arrays cannot be concatenated
```



https://stackoverflow.com/questions/37743843/python-why-use-numpy-r-instead-of-concatenate/37751518#37751518

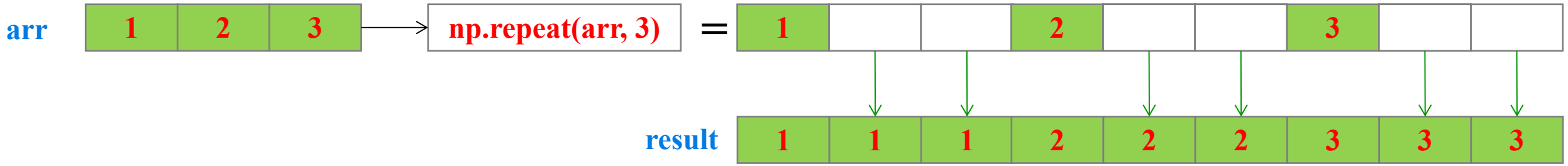
Problem 9

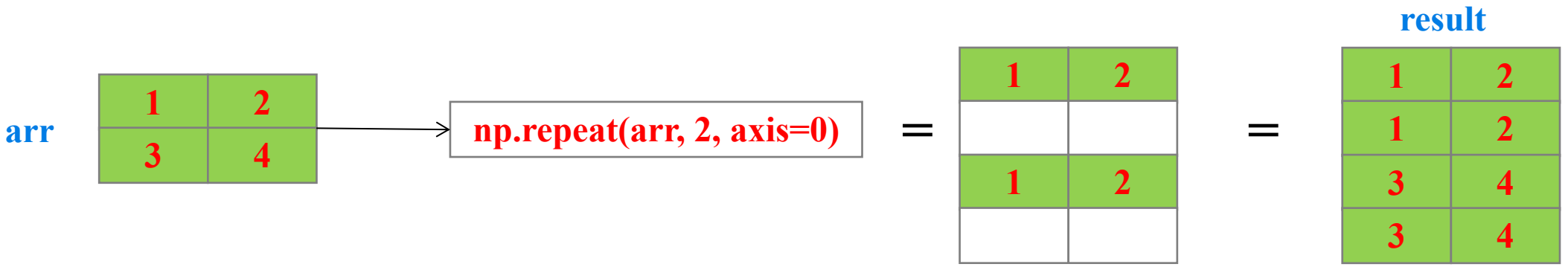
- Xếp chồng 2 mảng theo chiều ngang
  - np.hstack()
  - np.concatenate()
  - np.c_[]: concatenate theo axis thứ hai, thuần Python code không nhanh như **concatenate**

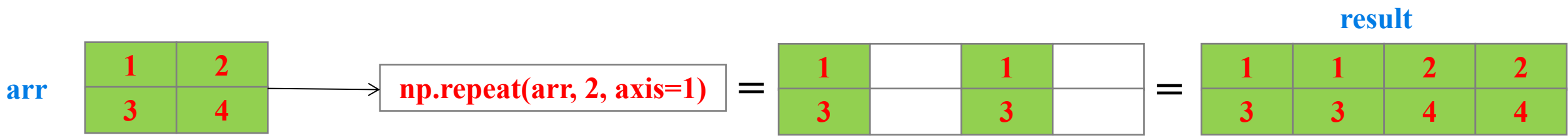


Problem 10

- Tạo mảng mỗi phần tử lập lại 3 lần
  - np.repeat(a, repeats, axis)

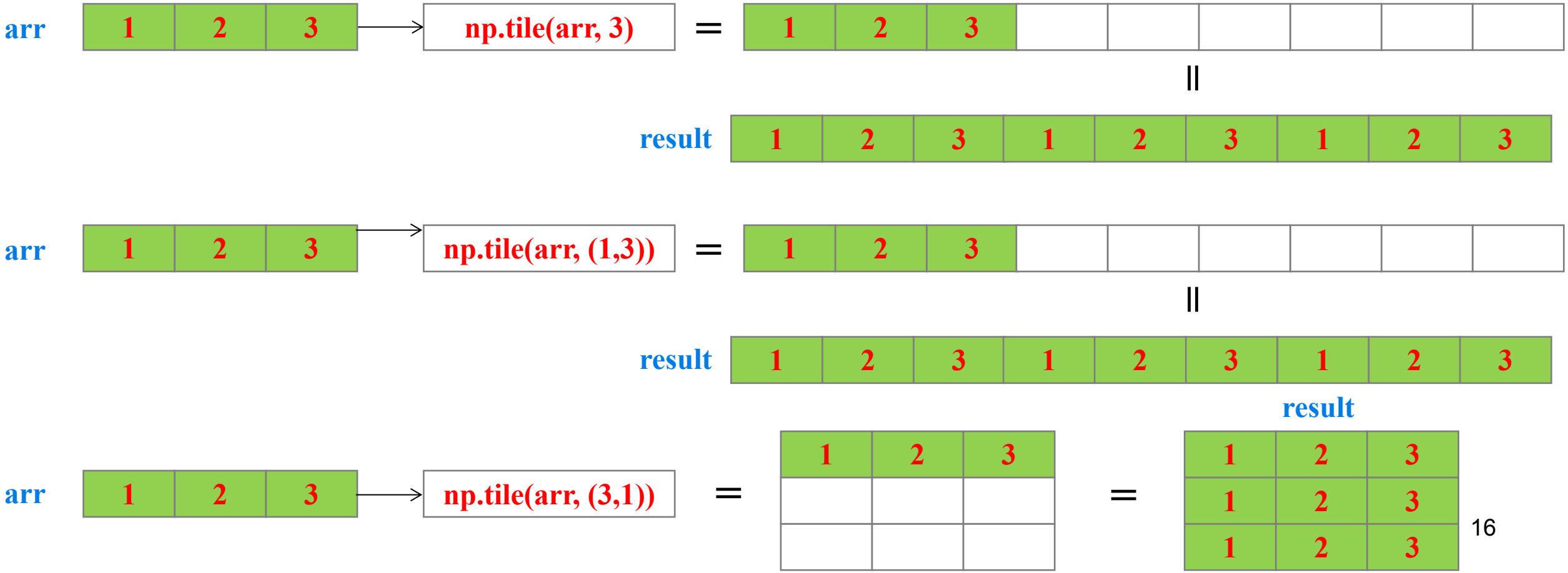






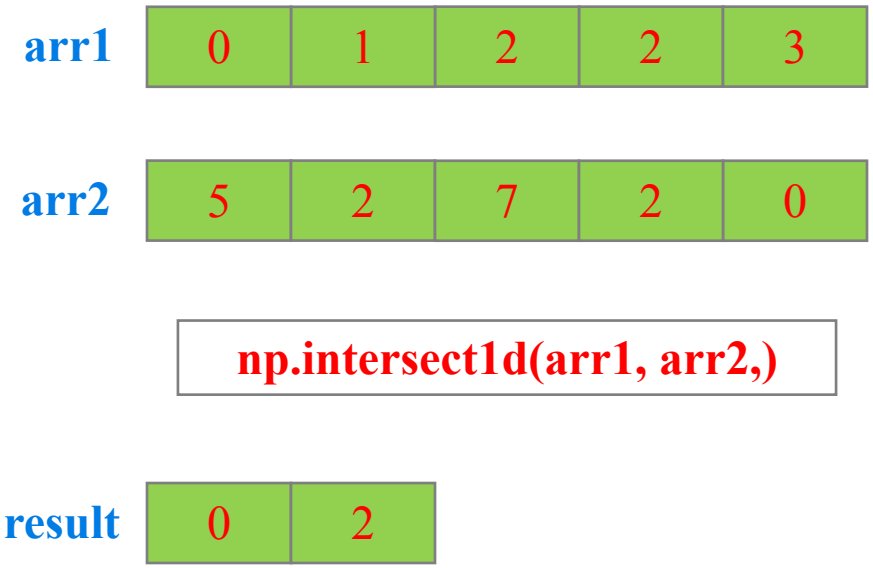
Problem 10

- Tạo mảng lập tất cả phần tử lên 3 lần
  - np.tile(A, reps)
  - reps: là số lần lặp lại hoặc 1 arr theo các axis



Problem 11

- Lấy phần tử chung của 2 mảng và trả về kết quả duy nhất xuất hiện ở 2 mảng
  - np.intersect1d(ar1, ar2)




```
1 arr1 = np.array([0,1,2,2,3])
2 arr2 = np.array([5,2,7,2,0])
3 # Lấy phần tử chung của 2 mảng
4 np.intersect1d(arr1,arr2)
```

```
array([0, 2])
```

Problem 12

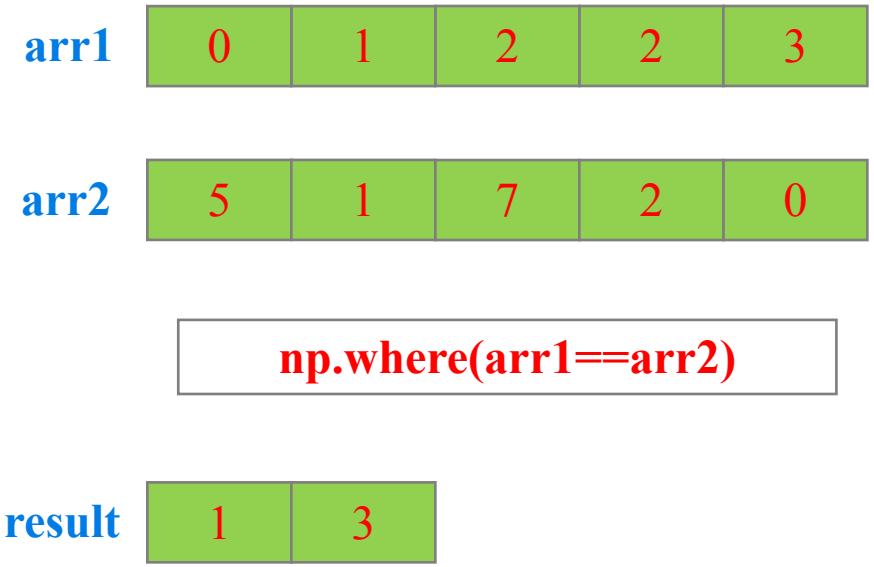
- Lấy phần tử trong mảng **a** khác phần tử trong mảng **b**
  - np.setdiff1d(ar1, ar2)


```
1 arr1 = np.array([0,1,2,2,3])
2 arr2 = np.array([5,2,7,2,0])
3 # Lấy phần tử có trong arr1 mà không có trong arr2
4 np.setdiff1d(arr1,arr2)
```

```
array([1, 3])
```

Problem 13

- Lấy tất cả vị trí nơi giá trị các phần tử của 2 mảng a,b giống nhau (**cùng vị trí và value giống nhau**)
  - np.where(condtion)

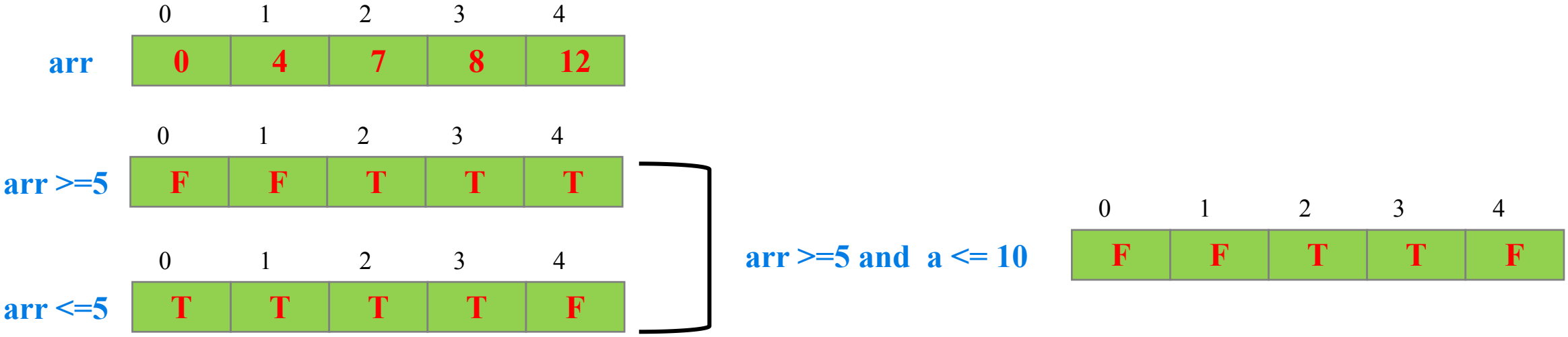



```
1 arr1 = np.array([0,1,2,2,3])
2 arr2 = np.array([5,1,7,2,0])
3 print(np.where(arr1==arr2))
```

```
(array([1, 3]),)
```

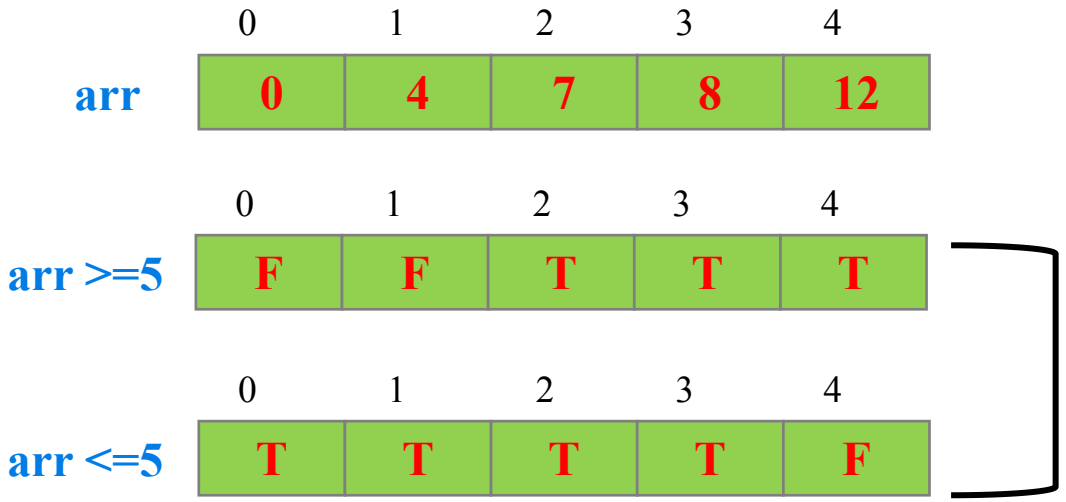
Problem 14

- Tìm tất cả vị trí của các phần tử có giá trị trong phạm vi [5,10]
  - Tìm index thỏa điều kiện trên
  - Không thể dùng 5 <= arr <= 10.
  - Tìm index1 arr >=5, ibdex2 a <= 10
  - index result = vị trí mà cả index1 và index2 có giá trị True

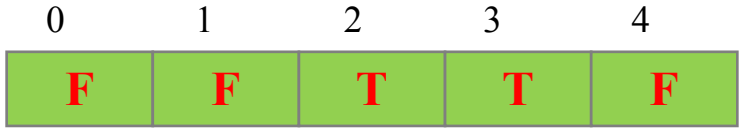


Problem 14

- Tìm tất cả vị trí của các phần tử có giá trị trong phạm vi [5,10]
  - Cách 1: dùng toán tử & và dùng np.where()
  - Cách 2: dùng np.logical_and() với np.where
  - Cách 3: dùng toán tử & và aray indexing

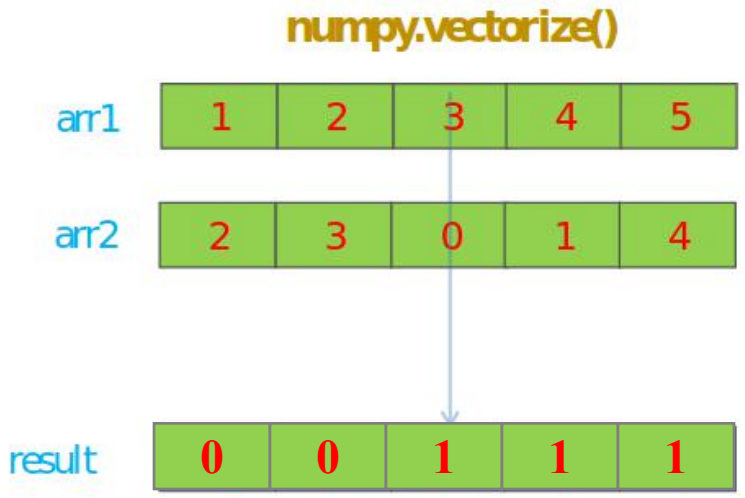


arr >=5 and a <= 10



Problem 15

- Tạo hàm xử lý trên mảng numpy dùng vectorize để lấy số lớn nhất trong 2 mảng
  - vect_fnc = vectorize(pyfunc)
  - vect_fnc(ar1, ar2)

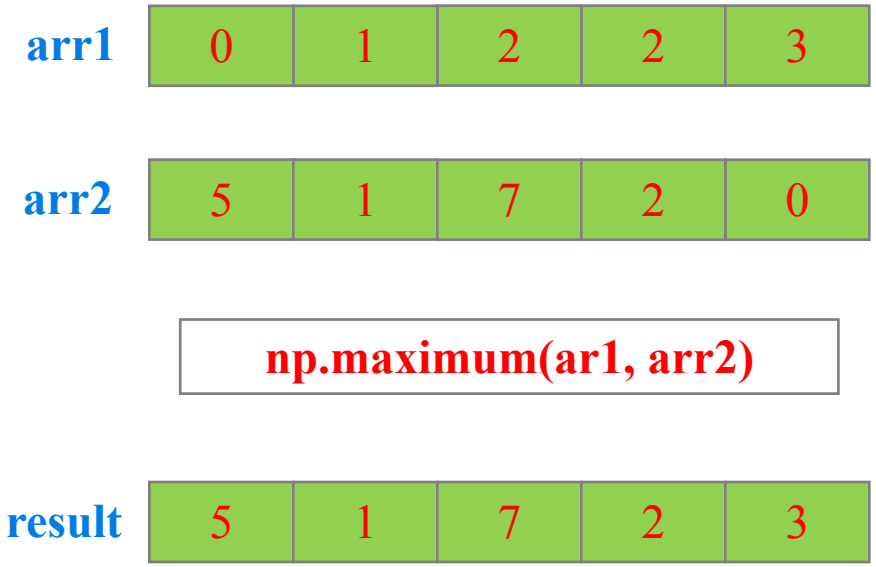



```python
# aivietnam.ai
import numpy as np

# a normal python function
def compare(a, b):
    if a >= b:
        return 0
    else:
        return 1

# vectorize the function
compare_vec = np.vectorize(compare)

# create two arrays
arr1 = np.array([1, 2, 3, 4, 5])
print("arr1: ", arr1)

arr2 = np.array([2, 3, 0, 1, 4])
print("arr2: ", arr2)

# compute result
result = compare_vec(arr1, arr2)
print("result: ", result)
```

```
arr1:  [1 2 3 4 5]
arr2:  [2 3 0 1 4]
result:  [1 1 0 0 0]
```

Problem 15

- Lấy số lớn nhất trong 2 mảng dùng np.maximum
  - np.maximum(x1, x2) (find max element wise)

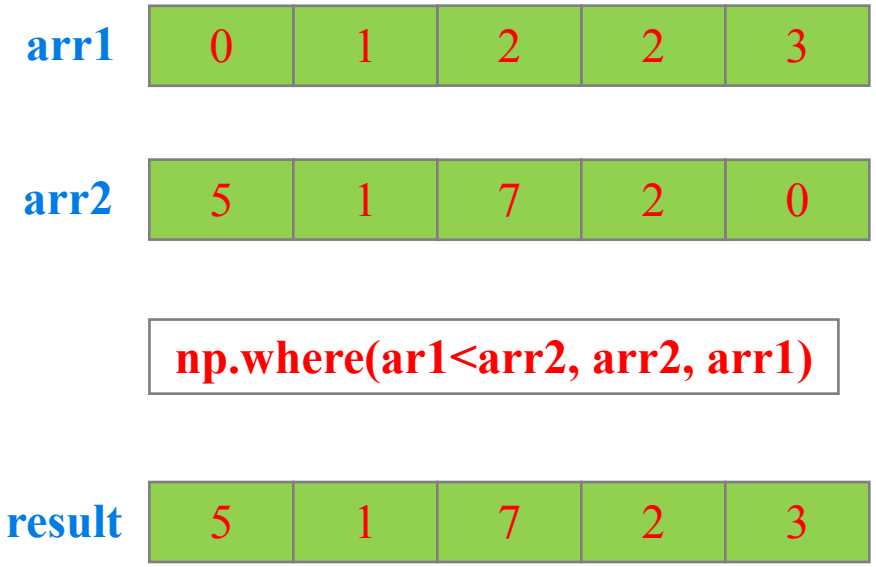



```
1 arr1 = np.array([0,1,2,2,3])
2 arr2 = np.array([5,1,7,2,0])
3 # max giữa 2 array
4 np.maximum(arr1,arr2)
```

```
array([5, 1, 7, 2, 3])
```

Problem 15

- Lấy số lớn nhất trong 2 mảng dùng np.where
  - np.where(condition,x,y)




```
1 arr1 = np.array([0,1,2,2,3])
2 arr2 = np.array([5,1,7,2,0])
3 # max giữa 2 array
4 np.where(arr1<arr2, arr2, arr1)
```

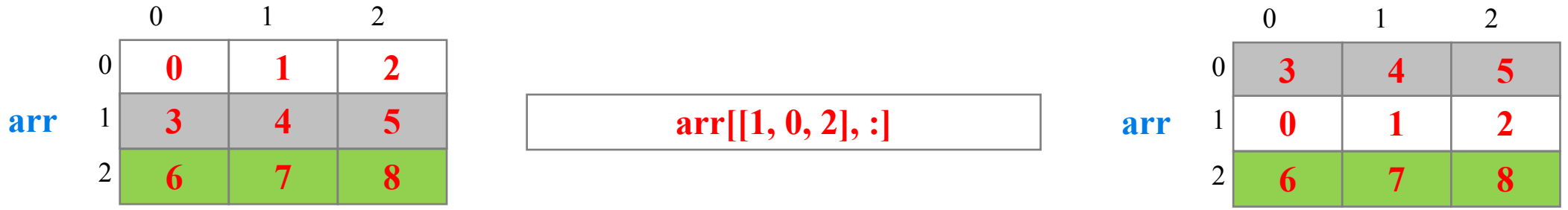
```
array([5, 1, 7, 2, 3])
```

Problem 16

- Hoán đổi các cột trong mảng 2 chiều
  - Dùng aray indexing
  - arr[:, [1, 0, 2]] “:” lấy hết các hàng, [1 0 2], lấy column 1, 0, 2

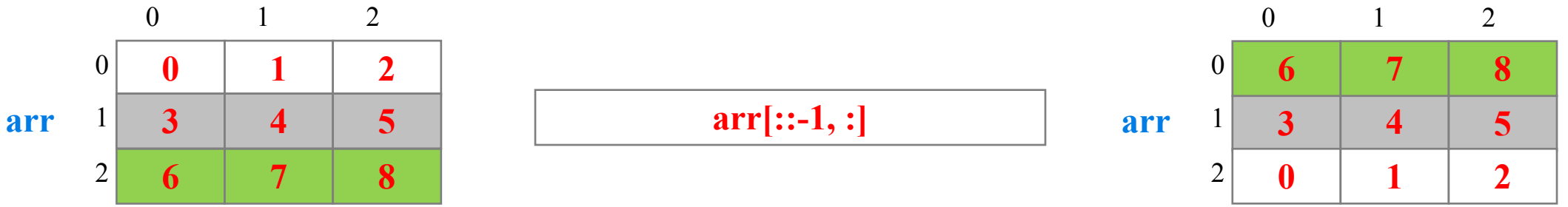
Problem 17

- Hoán đổi các hàng trong mảng 2 chiều
  - Dùng aray indexing
  - arr[[1, 0, 2], :] “:” lấy hết các cột, [1 0 2], lấy hàng 1, 0, 2



Problem 18

- Đảo ngược hàng trong mảng 2 chiều
  - Dùng aray indexing
  - arr[::-1, :]
  - "::-1": bắt đầu từ hàng cuối cùng, lùi dần lên vị trí đầu, mỗi lần di chuyển 1 đơn vị
  - ":" lấy hết các cột
  - VD: [0, 1, 2] => [::-1] => [2, 1, 0]



Problem 19

- Đảo ngược cột trong mảng 2 chiều
  - Dùng aray indexing
  - arr[:, ::-1]
  - ":" lấy hết các cột
  - "::-1": bắt đầu từ cột cuối cùng, lùi dần lên vị trí đầu, mỗi lần di chuyển 1 đơn vị.
  - VD: [0, 1, 2] => [::-1] => [2, 1, 0]

arr

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

arr[:, ::-1]

arr

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

Problem 20

- Tạo mảng 2 chiều chứa số random kiểu float từ [5, 10)
  - Dùng np.random.uniform(low, high, size)

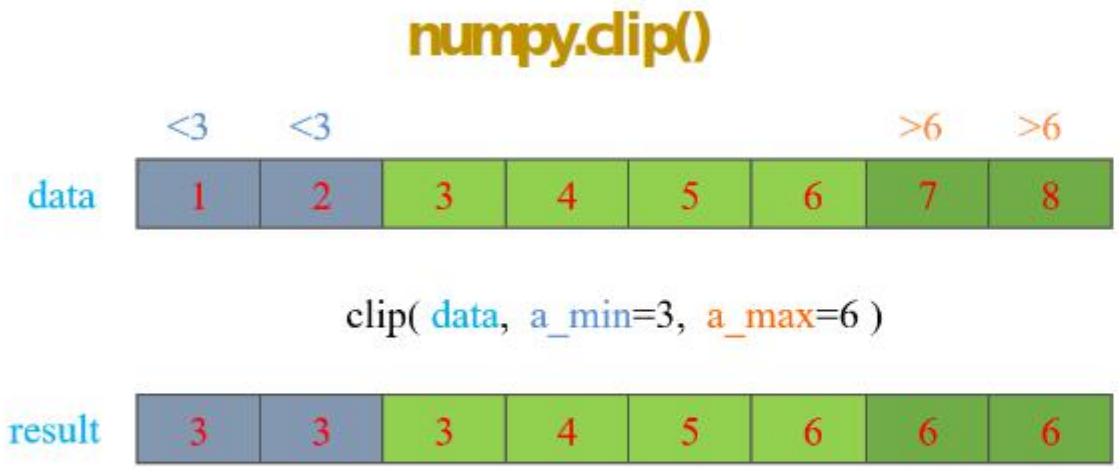
Uniform distribution [a, b)

```
1  import numpy as np
2
3  # uniform distribution in [a, b)
4  data = np.random.uniform(1, 9, 3)
5  print(data)
```

```
[1.03405352 5.71464479 6.09586162]
```

Problem 21

- Thay thế tất cả các giá trị lớn hơn 30 thành 30 và dưới 10 thành 10
  - Dùng np.clip(a, a_min, a_max)

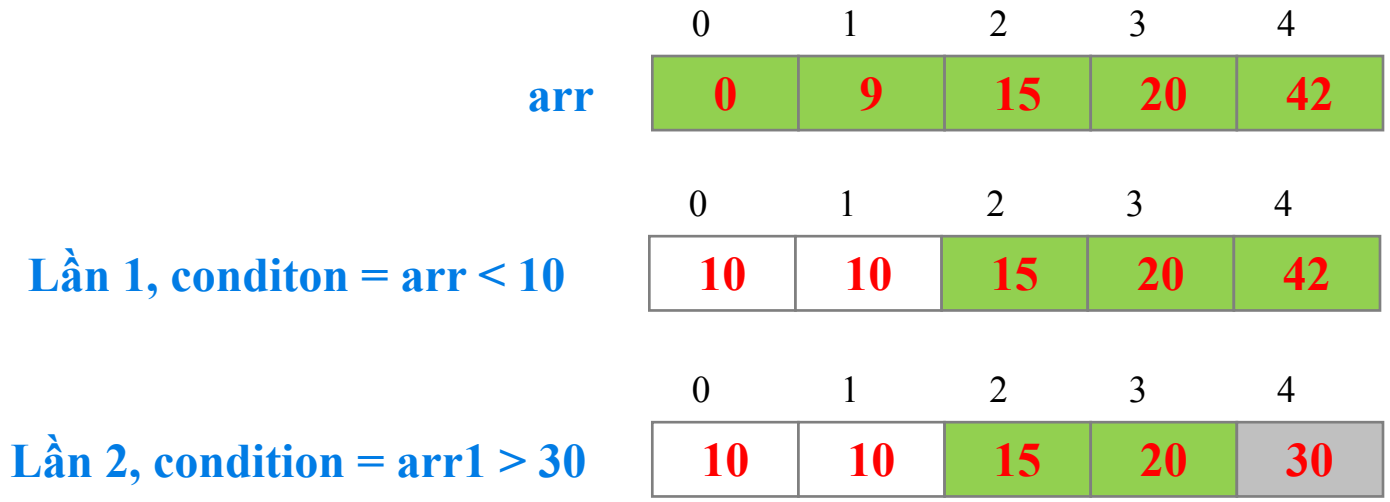



```python
# aivietnam.ai
# clip values

import numpy as np

data = np.array([1, 2, 3, 4, 5, 6, 7, 8])
print("data: ", data)

# element < 3 sẽ gán bằng 3
# element > 6 sẽ gán bằng 6
result = np.clip(data, a_min=3, a_max=6)
print("result: ", result)
```

```
data:  [1 2 3 4 5 6 7 8]
result:  [3 3 3 4 5 6 6 6]
```

Problem 21

- Thay thế tất cả các giá trị lớn hơn 30 thành 30 và dưới 10 thành 10
  - Dùng np.where(condition, x, y)
  - Dùng 2 lần np.where
  - Lần 1 (arr1) condition= arr < 10, x = 10, y = arr
  - Lần 2 (arr2) condition = arr1 > 30, x = 30, y=arr1

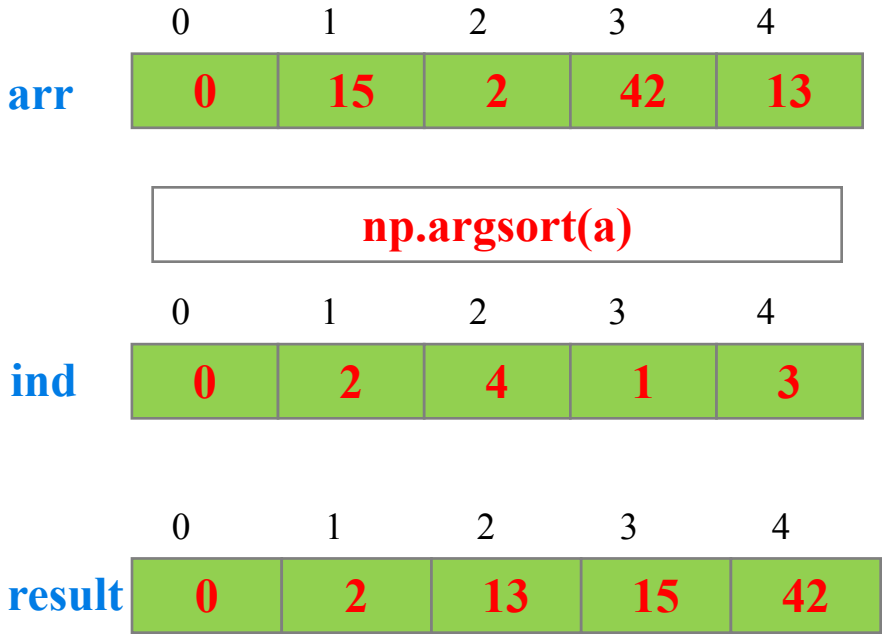


Problem 22

- Lấy vị trí của 5 giá trị lớn nhất trong một mảng a
  - Dùng np.argsort(a): trả về indices các element trong a sắp xếp theo từ bé đến lớn

```
1 arr = np.array([0, 15, 2, 42, 13])
2 ind = np.argsort(arr)
3 print(ind)
4 result = arr[ind]
5 print(result)
```

```
[0 2 4 1 3]
[ 0  2 13 15 42]
```



Problem 22

- Lấy vị trí của 5 giá trị lớn nhất trong một mảng **a**
  - Dùng np.argpartition(a, kth): vị trí thứ kth là kết quả vị trí sau khi **a** đã được sắp xếp, phần tử nhỏ hơn a nằm trái, phần tử lớn hơn a nằm bên phải. Bên trái và bên phải là không theo thứ tự . Nhưng kết quả trả về là index của **a**
  - VD: khi kth = 2

	0	1	2	3	4
sorted	0	2	13	15	42

```
1 arr = np.array([0, 15, 2, 42, 13])
2 ind = np.argpartition(arr, 2)
3 print(ind)
4 result = arr[ind]
5 print(result)
```

```
[0 2 4 3 1]
[ 0  2 13 42 15]
```

	0	1	2	3	4
arr	0	15	2	42	13

np.argpartition(a, 2)

	0	1	2	3	4
ind	0	2	4	3	1

arr[ind]

	0	1	2	3	4
result	0	2	13	42	15

Problem 22

- Lấy vị trí của 5 giá trị lớn nhất trong một mảng **a**
  - Dùng np.argpartition(a, kth): vị trí thứ kth là kết quả vị trí sau khi **a** đã được sắp xếp, phần tử nhỏ hơn a nằm trái, phần tử lớn hơn a nằm bên phải. Bên trái và bên phải là không theo thứ tự . Nhưng kết quả trả về là index của **a**
  - VD: khi kth = -3

	0	1	2	3	4
sorted	0	2	13	15	42

```
1 arr = np.array([0, 15, 2, 42, 13])
2 ind = np.argpartition(arr, -3)
3 print(ind)
4 result = arr[ind]
5 print(result)
```

```
[0 2 4 3 1]
[ 0  2 13 42 15]
```

	0	1	2	3	4
arr	0	15	2	42	13

np.argpartition(a, 2)

	0	1	2	3	4
ind	0	2	4	3	1

arr[ind]

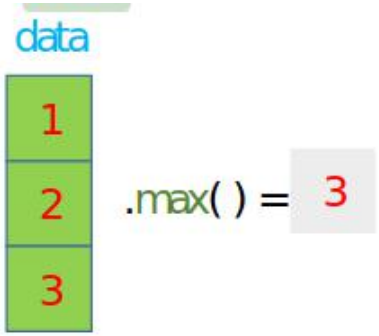
	0	1	2	3	4
result	0	2	13	42	15

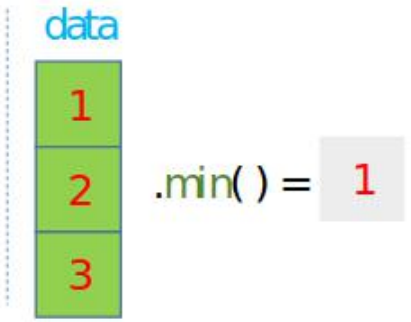
Problem 23

- Chuyển đổi một mảng chứa nhiều mảng thành một mảng 1d phẳng
  - Cách 1: Dùng 2 vong lặp for, vòng lặp thứ nhất để lấy từng array, vòng lặp thứ 2 lấy từng element. Sau đó append các element vào list và convert sang array
  - Cách 2: Dùng np.concatenate

Problem 24

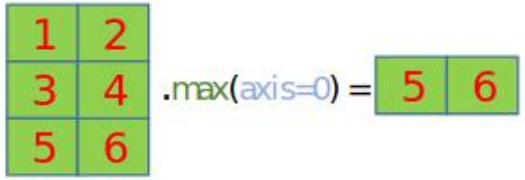
- Tìm giá trị lớn nhất trong mỗi hàng của mảng 2d
  - Dùng np.amax(a, axis) **(np.max là alias của np.amax)**

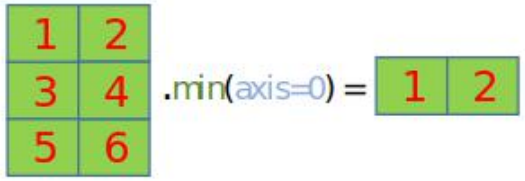


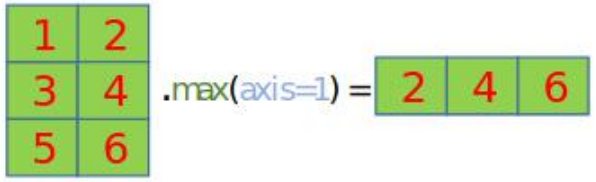


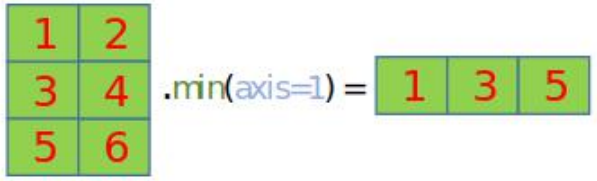

```
# aivietnam.ai
import numpy as np

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

print(data.max())
print(data.min())
```

```
3
1
```



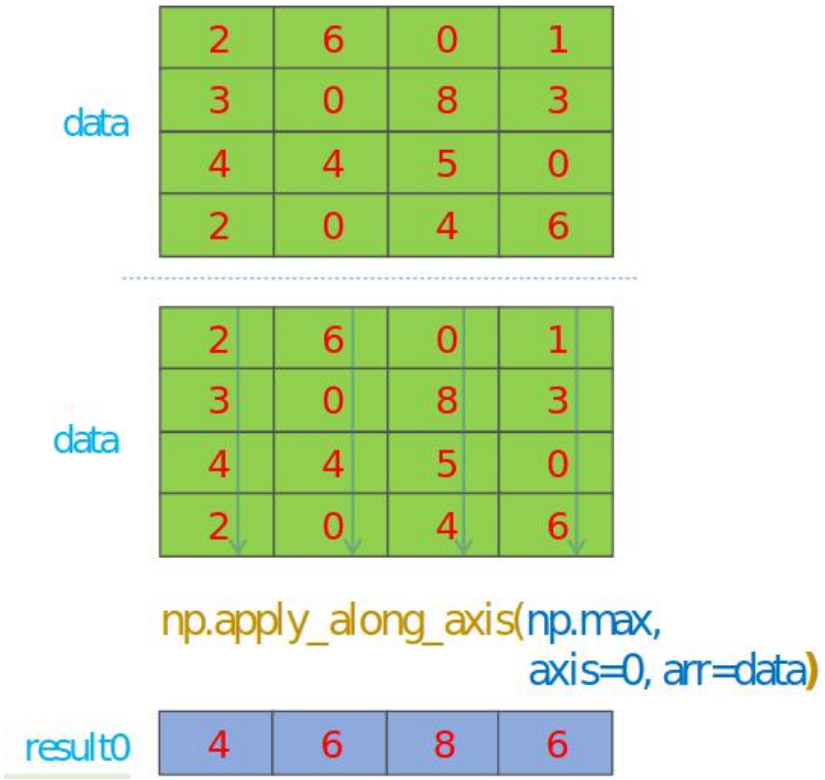






Problem 24

- Tìm giá trị lớn nhất trong mỗi hàng của mảng 2d
  - Dùng np.apply_along_axis(func1d, axis, args)
  - func1d: nên nhận 1d array



Problem 25

- Tính tỉ số min/max cho mỗi hàng trong mảng **a** (2d)
  - Dùng np.apply_along_axis(func1d, axis, args)
  - func1d: lambda x: np.min(x)/np.max(x)

Problem 26

- Tìm các vị trí có giá trị trùng lặp trong một mảng numpy. Lần đầu xất hiện là fasle, những lần sau (>=2) là True
  - np.unique(ar, return_index, return_inverse, return_counts)

	0	1	2	3	4	5
arr	0	2	0	15	42	2

```
1 arr = np.array([0, 2, 0, 15, 42, 2])
2 unique_a, return_index, return_inverse, return_counts = \
3     np.unique(arr, return_index=True, return_inverse=True, return_counts=True)
4
5 print("arr = ",arr)
6 print("unique_a = ", unique_a)
7 print("return_index = ", return_index)
8 print("return_inverse = ", return_inverse)
9 print("return_counts = ", return_counts)
```

```
arr =  [ 0  2  0 15 42  2]
unique_a =  [ 0  2 15 42]
return_index =  [0 1 3 4]
return_inverse =  [0 1 0 2 3 1]
return_counts =  [2 2 1 1]
```

	0	1	2	3
unique_a	0	2	15	42

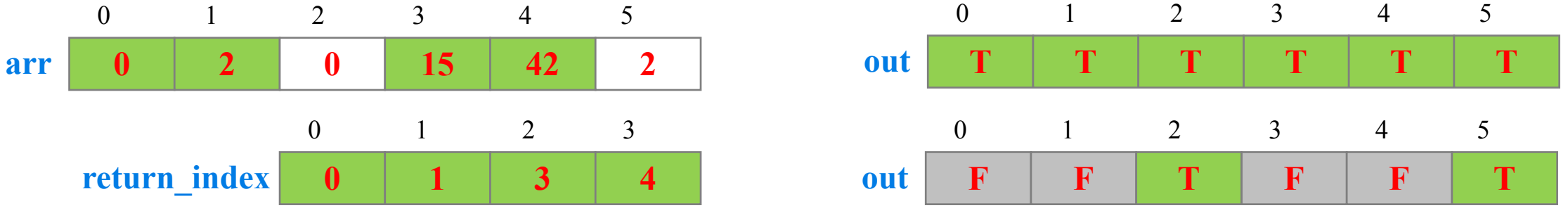
	0	1	2	3
return_index	0	1	3	4

	0	1	2	3	4	5
return_inverse	0	1	0	2	3	1

	0	1	2	3
return_counts	2	2	1	1

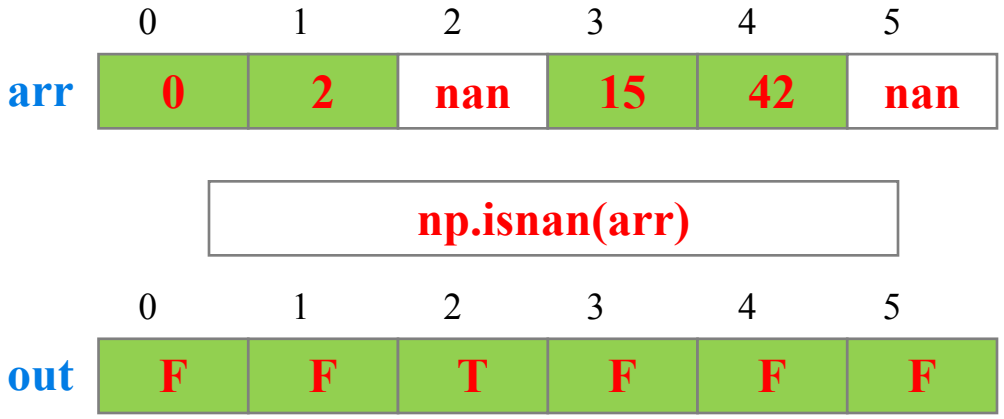
Problem 26

- Tìm các vị trí có giá trị trùng lặp trong một mảng numpy. Lần đầu xất hiện là fasle, những lần sau (>=2) là True
  - Dùng np.full để tạo array output có shape như input và các element là True
  - Dùng np.unique để lấy unique indices (vị trí số xuất hiện đầu tiên)
  - Tại output thay thế các element có index nằm trong indices bằng False (vị trí số đó xuất hiện đầu tiên luôn False)



Problem 27

- Loại bỏ tất cả các giá trị còn thiếu (np.nan) trong một mảng
  - np.nan == np.nan (sẽ bằng False) (theo định nghĩa của IEEE nan (not a number) không bằng nan và cũng ko phải là infinity)
  - Dùng hàm np.isnan(): trả về array với element có giá trị True khi đó là nan và ngược lại là False
  - Dùng array indexing để lấy các giá trị trong arr không phải là nan

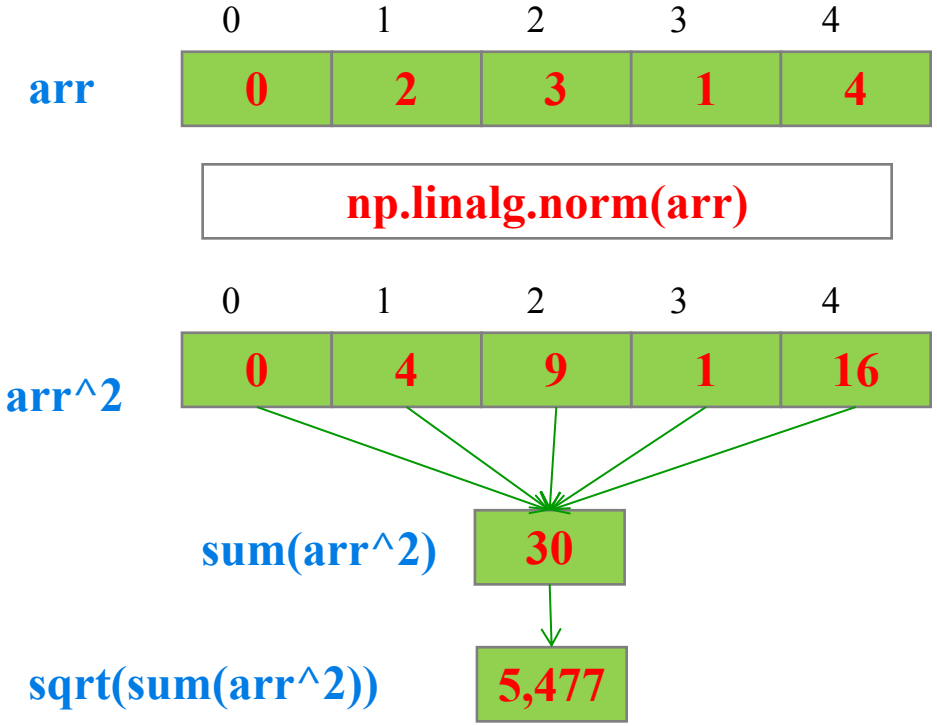



```
1 arr = np.array([0, 2, np.nan, 15, 42, np.nan])
2 out = np.isnan(arr)
3 print(arr)
4 print(out)
```

```
[ 0.  2. nan 15. 42. nan]
[False False  True False False  True]
```

Problem 28

- Tính khoảng cách giữa hai mảng a và b, dùng np.linalg.norm
  - np.linalg.norm(x, ord)
  - ord: là bậc của căn



- The 1-norm (aka taxicab norm) $||\mathbf{x}||_1 = \sum_i |x_i|$
- The 2-norm (aka Euclidean norm) $||\mathbf{x}||_2 = \sqrt{\sum_i |x_i|^2}$
- The more general p-norm $||\mathbf{x}||_p = \left(\sum_i |x_i|^p\right)^{1/p}$
- The infinity norm $||\mathbf{x}||_\infty = \max\{|x_i|\}$

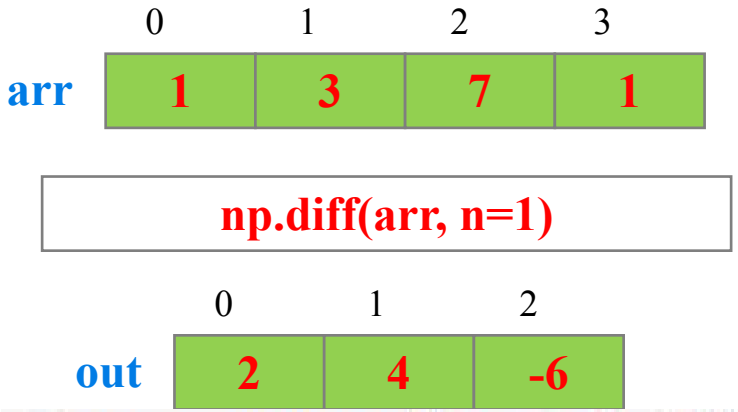
```
1 arr = np.array([0, 2, 3, 1, 4])
2 # norm(arr)
3 np.linalg.norm(arr)
```

```
5.477225575051661
```

Problem 29

- Tìm tất cả các cực đại cục bộ (hay còn gọi các đỉnh) trong một mảng 1d
  - np.diff(a, n)
  - np.sign(a)

out[i] = arr[i+1] - arr[i]




```
1 arr = np.array([1, 3, 7, 1])
2 out = np.diff(arr, n=1)
3 print(out)
```

```
[ 2  4 -6]
```

-1 if x < 0, 0 if x==0, 1 if x > 0

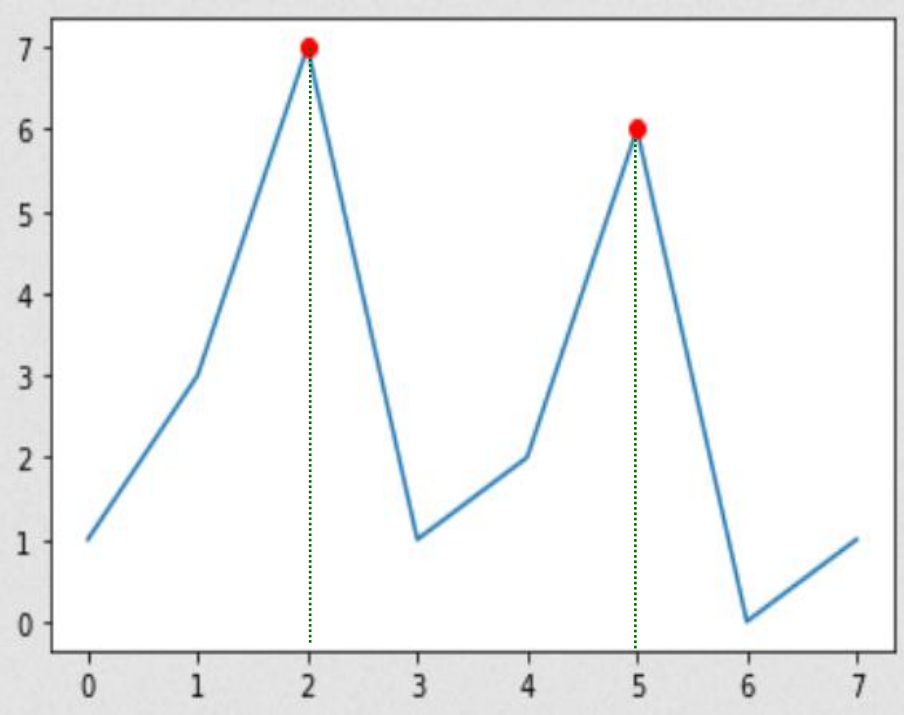

```
1 arr = np.array([1, 0, -7, 10])
2 out = np.sign(a)
3 print(out)
```

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

Problem 29

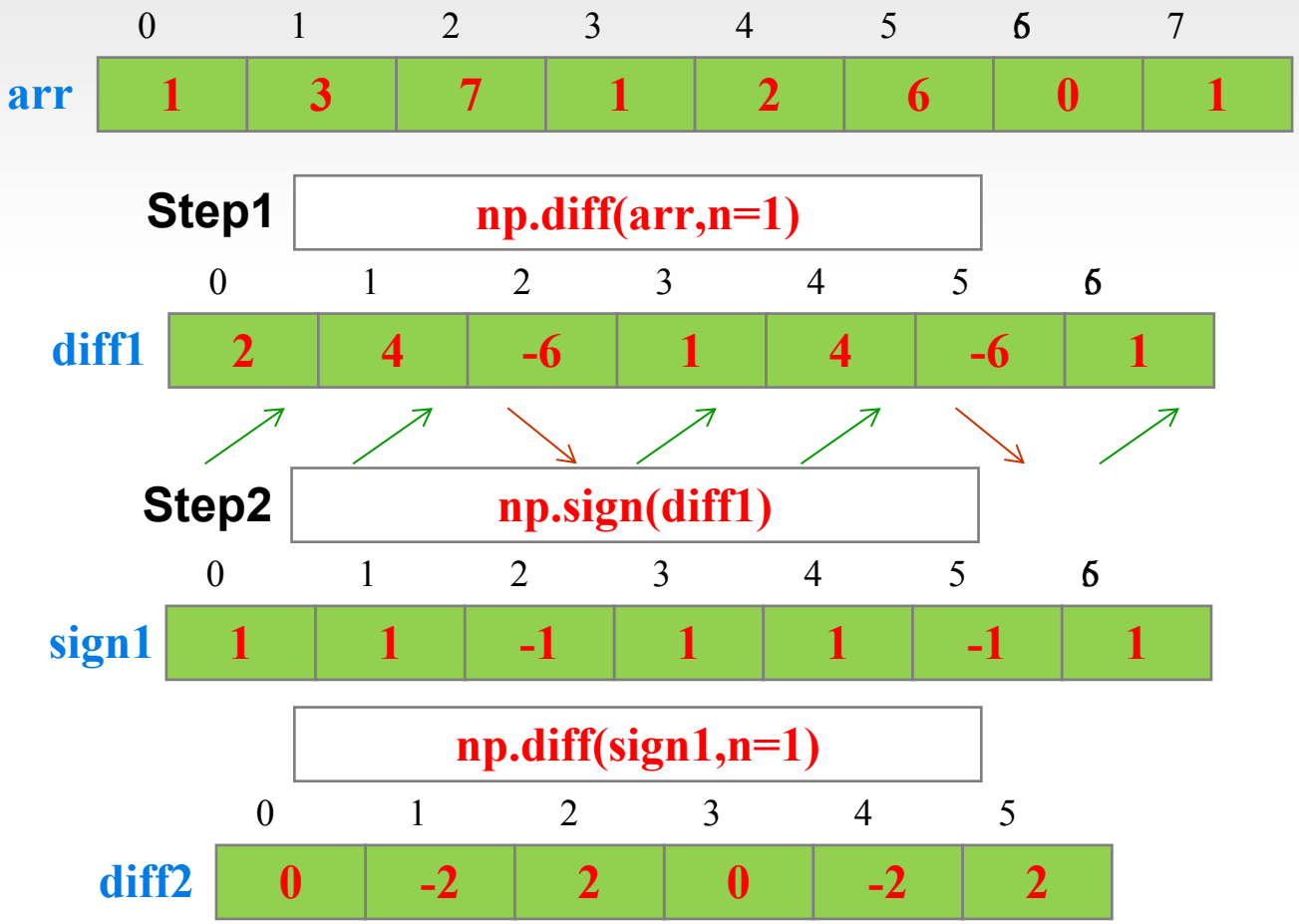
- Tìm tất cả các cực đại cục bộ (hay còn gọi các đỉnh) trong một mảng 1d

a = np.array([1, 3, 7, 1, 2, 6, 0, 1]) (trục y)



- Vị trí hiện tại (curr), trước đó 1 đơn vị (prev), sau đó 1 đơn vị (next)
- Để xác định được đỉnh cần điều kiện sau. prev<curr và curr > next
- prev<curr, xét tại prev là hàm có xu hướng tăng, curr>next, xét tại curr là hàm có xu hướng giảm
- Điều kiện thỏa mãn, thì **curr** chính là **đỉnh**
- prev < curr => curr-prev > 0 (np.sign = 1)
- curr > next => next-curr < 0 (np.sign = -1)

Problem 29



- Vị trí hiện tại (curr), trước đó 1 đơn vị (prev), sau đó 1 đơn vị (next)
- Để xác định được đỉnh cần điều kiện sau. prev < curr và curr > next
- **Step1:** dùng hàm np.diff để tìm hiệu giữa 2 điểm liên tiếp
- **Step2:** dùng hàm np.sign để lấy dấu giữa 2 điểm (-1: next<curr, hàm giảm) và (1: next>curr, hàm tăng). Để tạo thành một đỉnh thì hàm tăng sau đó sẽ giảm để thỏa mãn điều kiện trên. Vậy kết quả của np.sign mà 2 vị trí liên tiếp nhau có giá trị 1, -1 thì được coi là đỉnh
- **Step3:** để xác định 2 điểm liên tiếp từ kết quả của np.sign là 1, -1. Sử dụng hàm np.diff thêm lần nữa để tìm giá trị -2 chỉ có trường hợp này thì mới có 1, -1 liến tiếp
- **Step4:** Tìm các vị trí = -2 để lấy vị trí các đỉnh

Step4: xét trên arr.

- Nếu diff1[i]: dùng arr[i] và arr[i+1]
- Do đó diff2[i]: dùng diff[i], diff[i+1] <=> arr[i], arr[i+1], arr[i+2]
- Khi tìm value -2 trên diff2 thì vị trí tại điểm **[i]** trên diff2 chính là đỉnh tương ứng với **[i+1]**
- diff2 có 2 vị trí =-2 (i = 1, 4), vậy đỉnh sẽ là vị trí i=2,5 trên arr tương ứng với giá trị 7 và 6


Thank
you!