3iL INGENIEURS
</les sciences informatiques>

Data Science Lecture

By Raymond Houé Ngouna
(raymond.houe-ngouna@enit.fr)

Python and scientific computation: navigating in a NumPy array

Lecture objectives

Learn to **navigate** in a NumPy array which is a powerful data structure and particularly efficient for **scientific computing**.

Main concepts covered:

1. **Indexing:** navigating an array from indexes.
2. **Slicing/Subsetting:** extraction of a portion of an array.
3. **Broadcasting:** broadcasting operation between arrays.
4. **Selection/Masking:** verification of a condition for each item of an array.
5. **Boolean indexing:** conditional navigation in an array (selection of items that satisfy a condition).

[Outline]

Indexing | Slicing | Broadcasting | Masking et Boolean indexing

```python
# importation de NumPy
import numpy as np
```

Indexing

This is an array browsing mechanism, based on the use of indexes to search for elements (in an array).
The general principle is to specify in parameters the respective values of the dimensions corresponding to the required element.

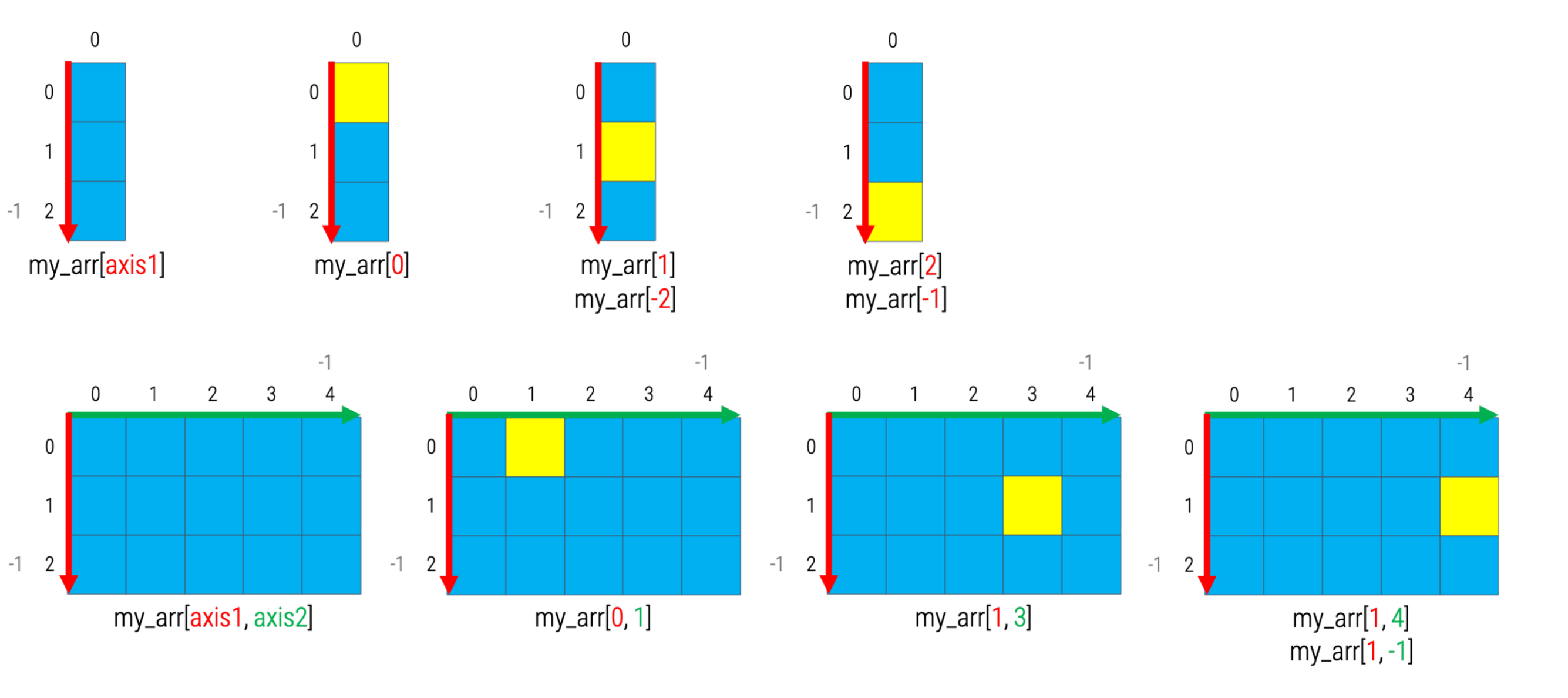
Syntax:

- For a **1 dimension** array (column vector): `my_arr[index]`
- For other dimensions: `my_arr[index_dim1, index_dim2, ...]`

Note: For a multi-dimension array, when specifying the value of a single dimension, the system returns all the elements of the 1st dimension of the table.

Hint:
Negative indexes can be used to specify navigation from the end:

- index `-1` corresponds to the last element of the corresponding dimension (i.e. the last row or last column in a 2-dimension array),
- index `-2` corresponds to the second last one,
- ...



Indexing on a 1-dimension array

```python
# creation of a 1-dimension array
# (with random numbers between 0 and 1)
my_arr_1d = np.random.rand(3)
```

```python
# check the shape of the created array
# which corresponds to a column vector
print(my_arr_1d.shape)
```

```python
# display the content of the array
print(my_arr_1d)
```

```python
# retrieve the first item (index 0 !!!)
print(my_arr_1d[0])
```

```python
# the second one (index 1 !!!)
print(my_arr_1d[1])
```

```python
# but the item is also the second two last (index -2)
print(my_arr_1d[-2])
```

```python
# the 3rd item (index 2 !!!)
print(my_arr_1d[2])
```

```python
# the 3rd item is the last one (index -1)
print(my_arr_1d[-1])
```

Indexing on a 2-dimension array

```python
# creation of a 2-d array
# with random numbers between 0 and 1
my_arr_2d = np.random.rand(3,5)
```

```python
# check the dimension of the array
print(my_arr_2d.shape)
```

```python
# display its content
print(my_arr_2d)
```

```python
# retrieve the items of the 1st row
# (no need to specify the indexes of the column)
print(my_arr_2d[0])
```

```python
# 2nd row
print(my_arr_2d[1])
```

```python
# 1st row and 2nd column
print(my_arr_2d[0,1])
```

```python
# last column of the the 1st row
print(my_arr_2d[0,-1])
```

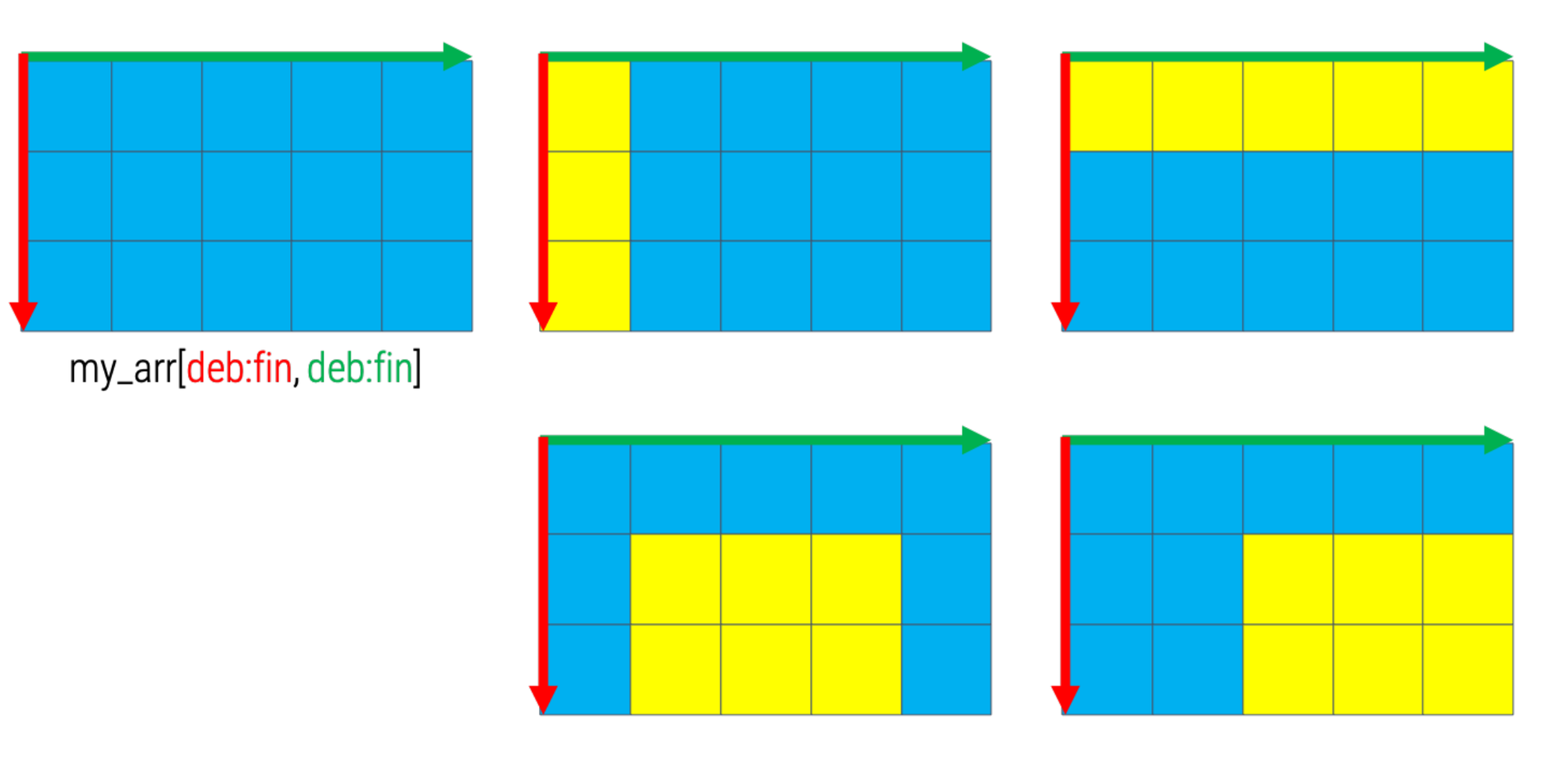
```python
# last row of the last column
print(my_arr_2d[-1,-1])
```

[Outline]

Slicing/Subsetting

This is the mechanism for extracting a sub-part of an array.
The principle consists in recovering the items from a starting index to an ending one.
By default, the path is made by steps of `1`, otherwise the step must be explicitly specified as the last parameter.
Syntax:

- For a **1 dimension** array (column vector): `my_arr[start:end:step]`
- For other dimensions: `my_arr[start:end,start:end,...]`



Array of the illustration

```python
# creation of a 1-d array composed of
# composed of the first 15 consecutive integers (0 excluded)
# and reformatting into a 2-d array
my_arr = np.arange(1,16).reshape(3,5)
print(my_arr)
```

Examples of slicing

```python
# retrieve the first row
print(my_arr[0,:])
```

```python
# the 1st column
print(my_arr[:,0])
```

```python
# from the 2nd to the 4th column, for the 2nd to the 3rd row
print(my_arr[1:3,1:4])
```

```python
# same result
print(my_arr[1:,1:4])
```

```python
# same result
print(my_arr[-2:,1:4])
```

```python
# same result
print(my_arr[-2:,-4:-1])
```

[Outline]

Broadcasting

Without the **vectorization** mechanism, to modify a sub-part of an array, a loop would have been used to navigate through the array.
In NumPy, one can, in a very simplified way, manipulate the values of a sub-part of an array by **broadcasting** the desired manipulation (for example, for a value modification).

Note: Slicing can then be used to specify the broadcasting area for complex usage.

Array of the illustration

```python
# recall the previously 2-d created array
print(my_arr)
```

```python
# let's create a new one by copy of the original one
# (which means modification on the copy will not affect the original)
my_arr_copy = my_arr.copy()
print(my_arr_copy)
```

```python
# and another array, by creating a view of the original
# (modifications from one array will create cchanges on the other)
my_arr_view = my_arr
print(my_arr_view)
```

Broadcasting of an assignment

```python
# let's modfiy the 1st row, replacing all the values by 9999
# (no need to loop, NumPy will broadcast the assignment)
my_arr[0,:] = 999
print(my_arr)
```

```python
# modification of the last column of the original array
# replacing the corresponding values by the number 555
my_arr[:,-1] = 555
print(my_arr)
```

```python
# the two modifications on the original affected the view
print(my_arr_view)
```

```python
# but not the copy
print(my_arr_copy)
```

```python
# modification of the last column of the copy of the original array
# with the number 777
my_arr_copy[:, -1] = 777
print(my_arr_copy)
```

```python
# modification on the copy did not affect the original array
print(my_arr)
```

```python
# modification of the last column of the view of the original array
# with the number 333
my_arr_view[:, -1] = 333
print(my_arr_view)
```

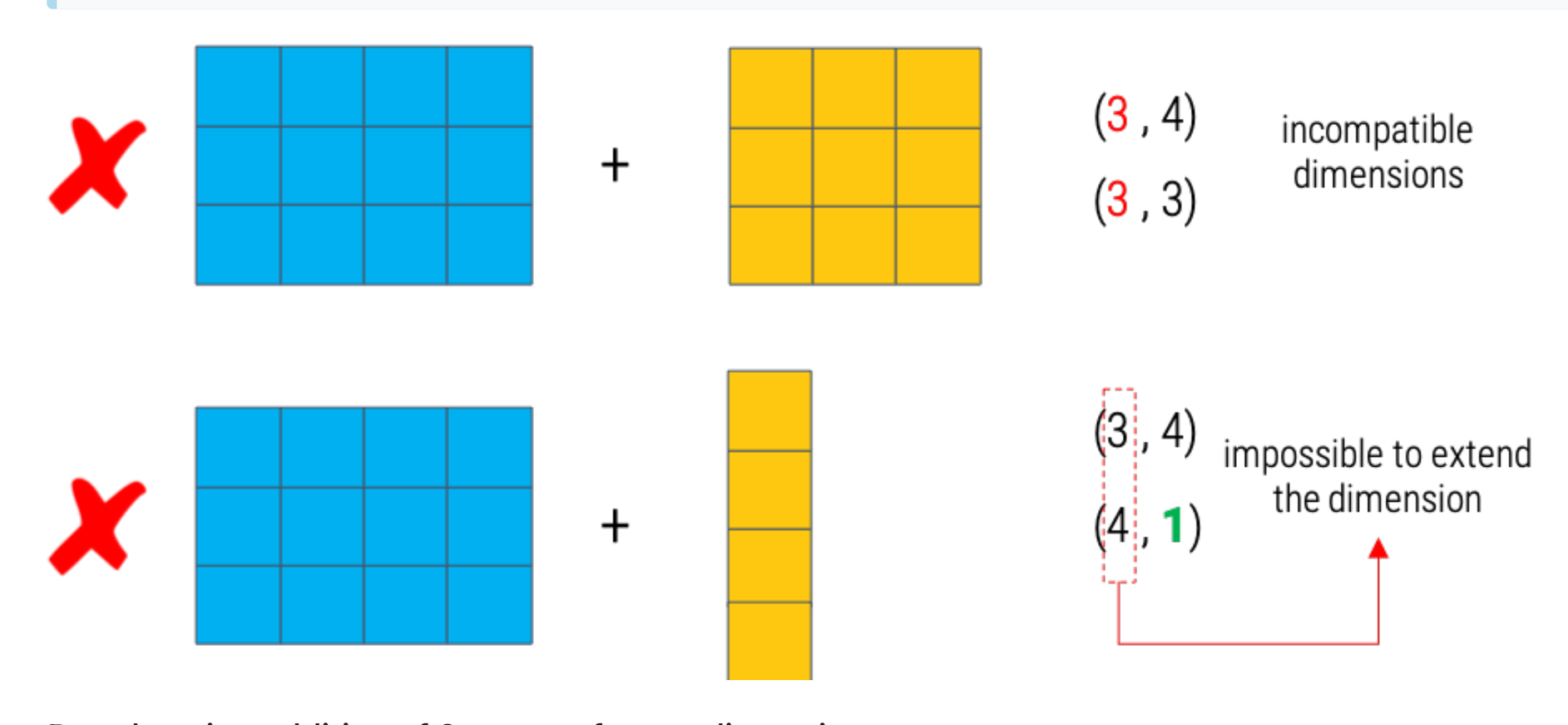
```python
# modification on the view has changed the original array
print(my_arr)
```

Broadcasting operations between 2 arrays

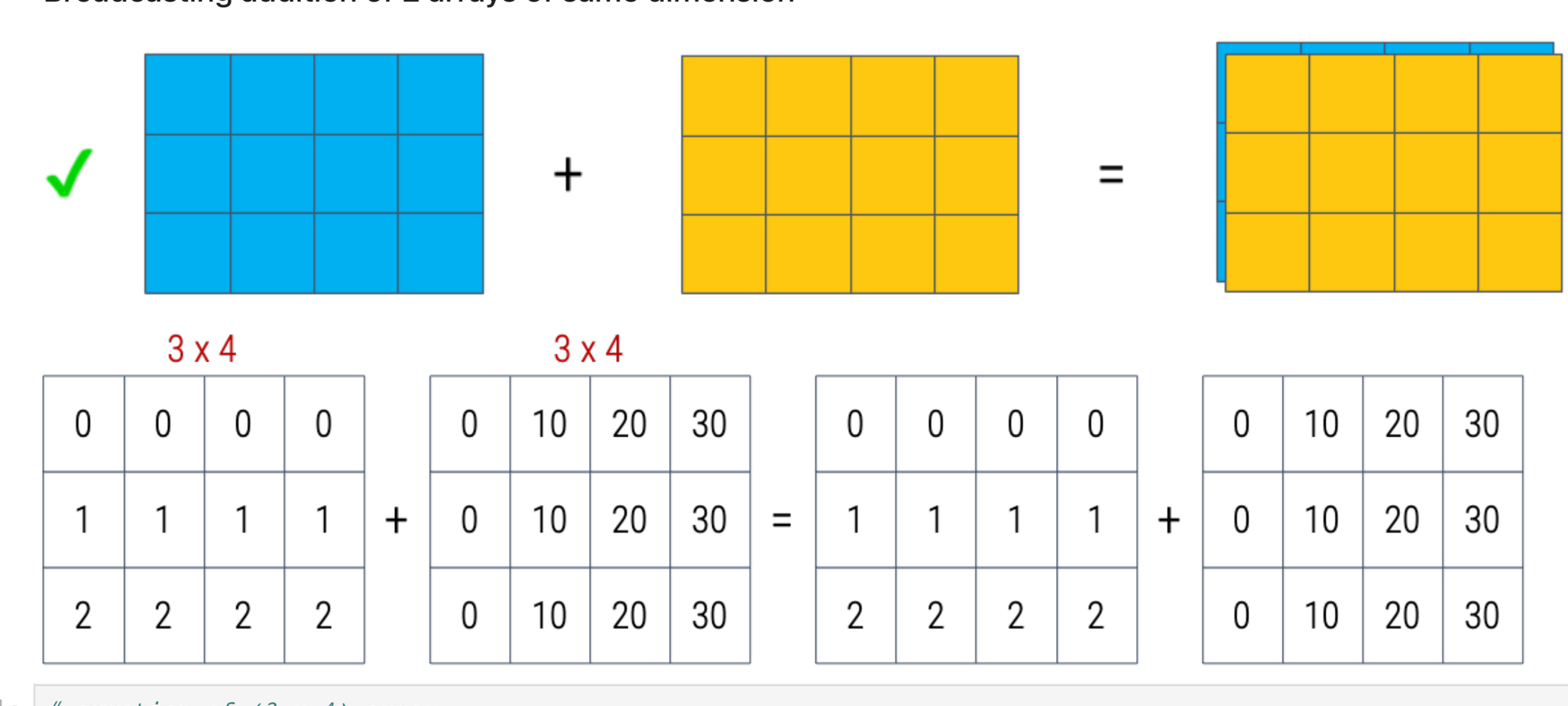
An operation, between arrays, e.g. addition, can be made by broadcasting without the need for loops.

However, one fundamental rule must be respected:

- either the dimensions (axes) of the two arrays are of same size,
- either one of the dimensions is of size `1`, and the size of the other dimension can be extended to the other array.



Broadcasting addition of 2 arrays of same dimension




```python
# creation of (3 x 4) array
# 1st row composed of 0, 2nd with 1 and 3rd with 2
arr_1 = np.array([0,0,0,0,1,1,1,1,2,2,2,2]).reshape(3,4)
print(arr_1)
```

```python
# creation of another (3 x 4) array
# 1st column composed of 0, 2nd with 10, ..., 4-th with 30
arr_2 = np.array([0,10,20,30,0,10,20,30,0,10,20,30]).reshape(3,4)
print(arr_2)
```

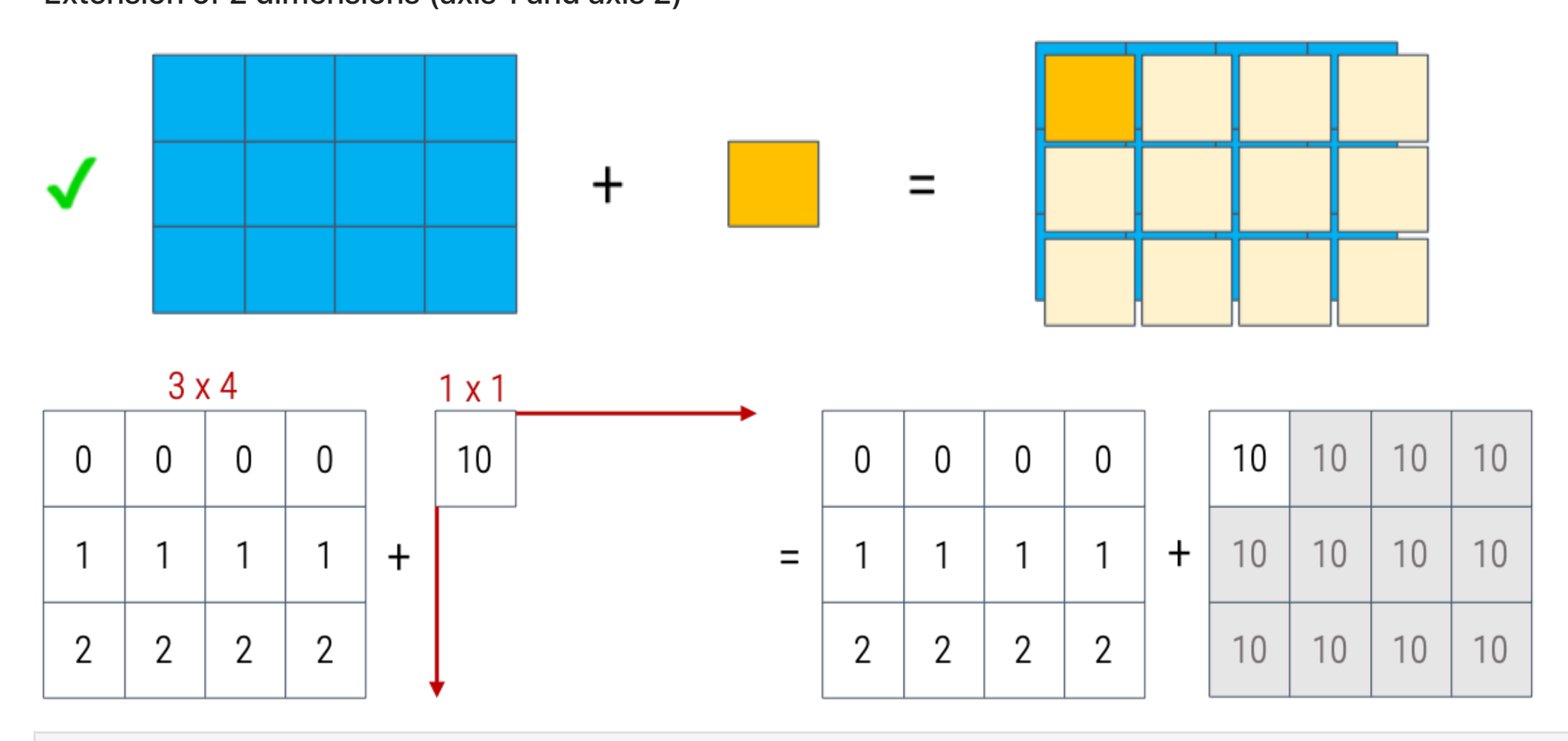
```python
# check the shape (dimension) of the 2 array
print(f'Shape of array 1: {arr_1.shape}')
print(f'Shape of array 2: {arr_2.shape}')
```

```python
# broadcasting addition on 2 arrays of same dimension (3 , 4)
print(arr_1 + arr_2)
```

Broadcasting on 2 arrays of different dimensions

One of the arrays has one dimension of size 1 and the other one can be extended on the second array.

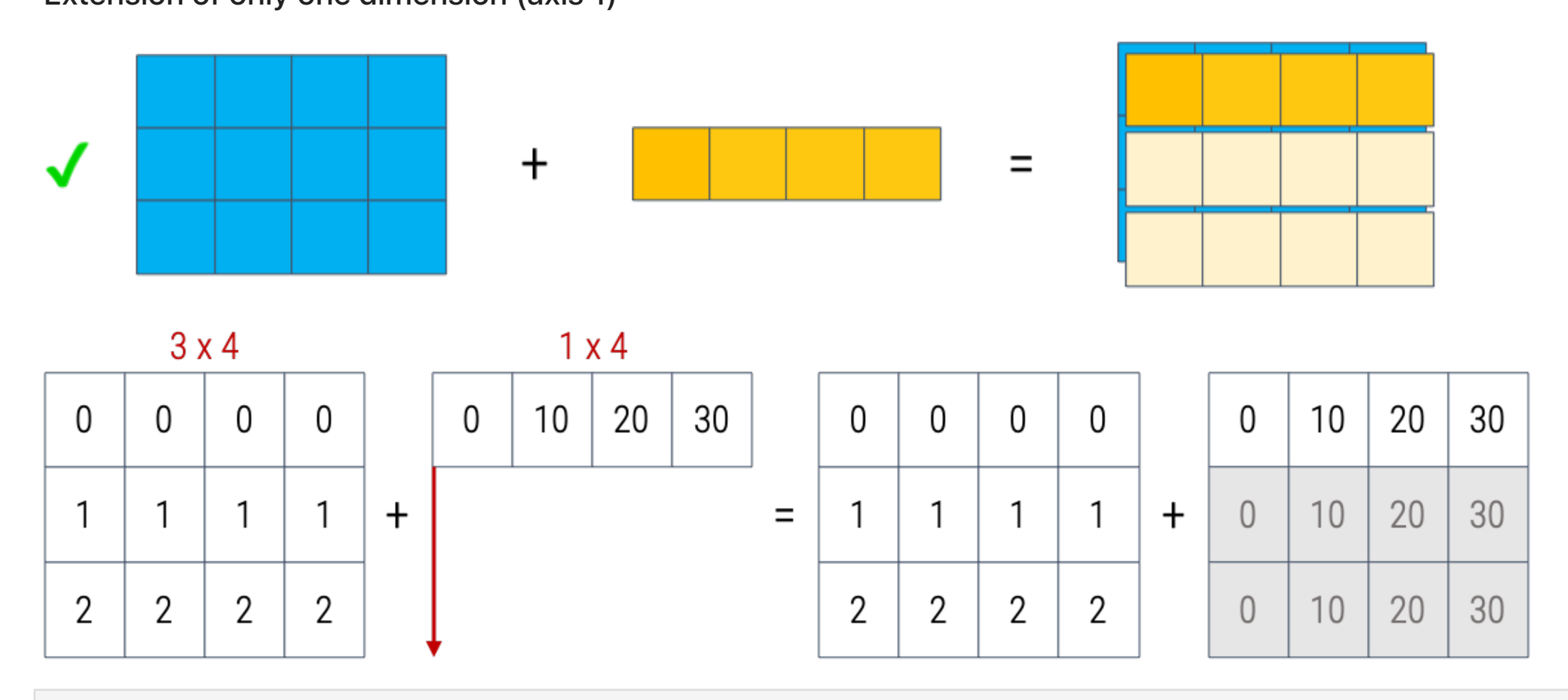
Extension of 2 dimensions (axis 1 and axis 2)




```python
# Dimensions of the 2 arrays
print(f'Shape of array 1: {arr_1.shape}')
print(f'Shape of the scalar (array 2): {np.shape(10)}')
```

```python
# broadcasting on arrays of different dimensions : (3 , 4) et (1 , 1)
print(arr_1 + 10)
```

Extension of only one dimension (axis 1)

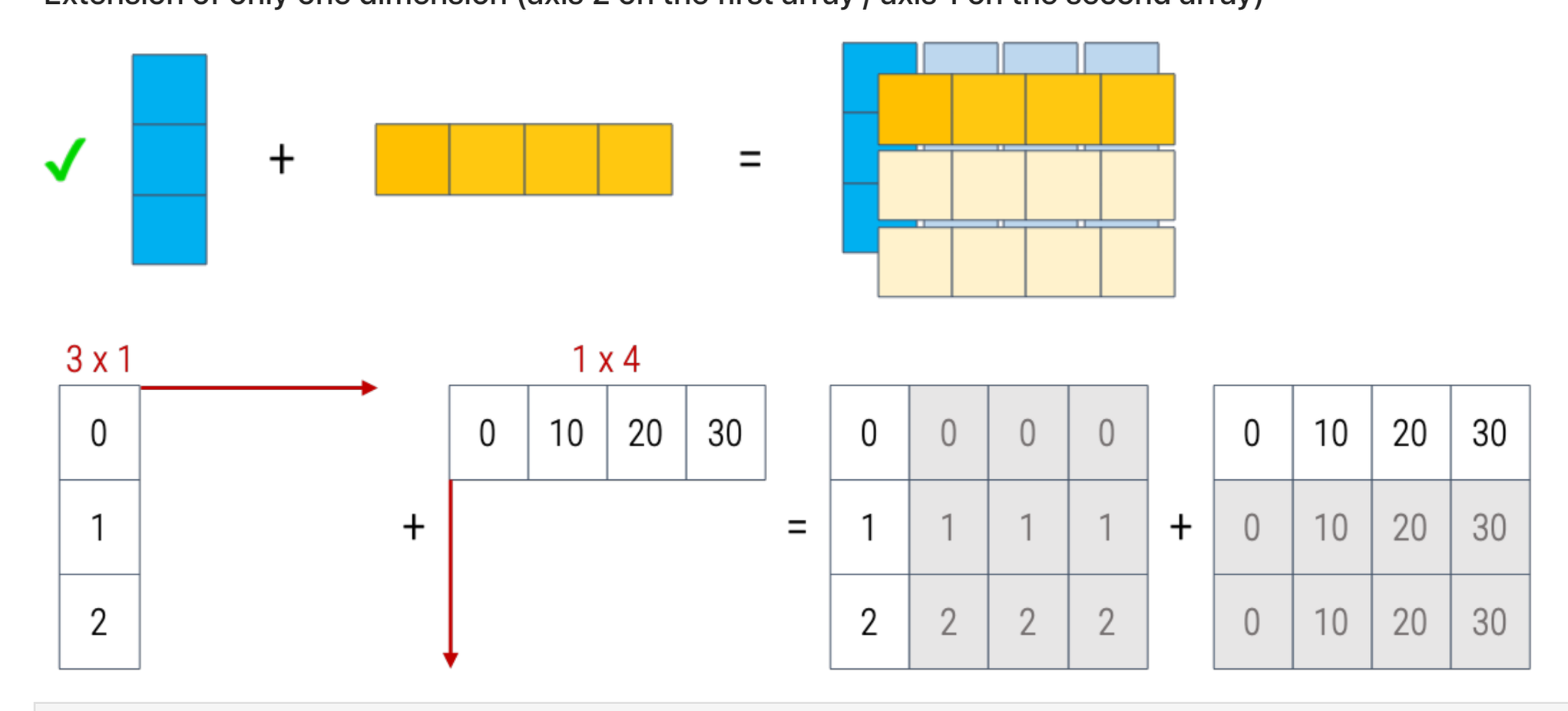



```python
# creation of an array of dimension (3 , 4)
arr_3 = np.array([0,10,20,30]).reshape(1,4)
print(arr_3)
```

```python
# Check the imensions of the 2 arrays
print(f'Shape of array 1: {arr_1.shape}')
print(f'Shape of array 3: {arr_3.shape}')
```

```python
# broadcasting on arrays of different dimensions: (3 , 4) et (1 , 4)
print(arr_1 + arr_3)
```

Extension of only one dimension (axis 2 on the first array / axis 1 on the second array)



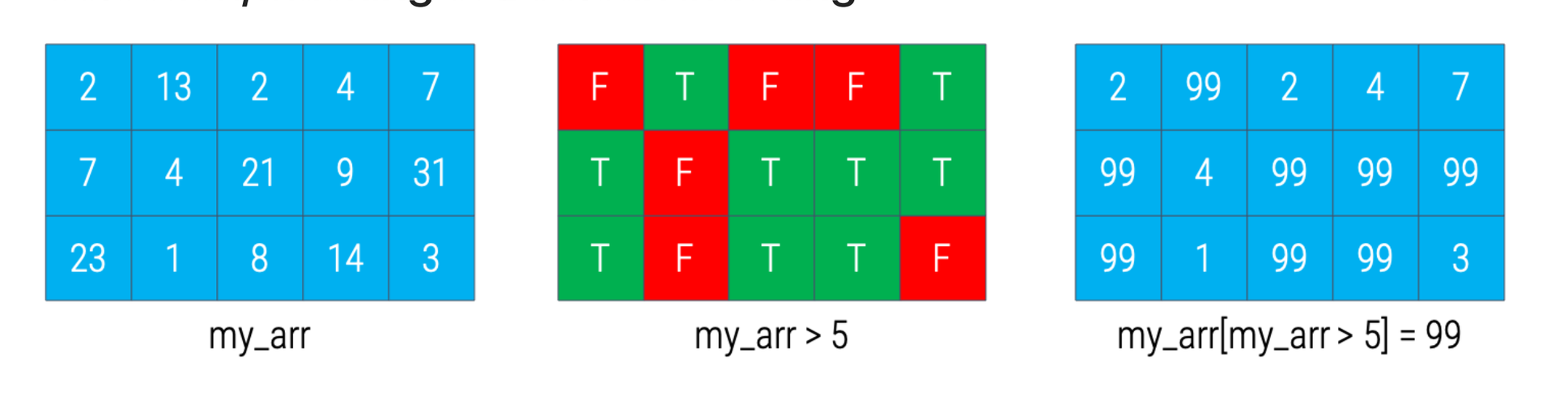

```python
# creation of an array of dimension (3 , 1)
arr_4 = np.array([0,1,2]).reshape(3,1)
print(arr_4)
```

```python
# Check the shapes of the 2 arrays
print(f'Shape of array 4: {arr_4.shape}')
print(f'Shape of array 3: {arr_3.shape}')
```

```python
# broadcasting on arrays of different dimensions
# (3 , 1) et (1 , 4)
print(arr_4 + arr_3)
```

[Outline]

Selection/Masking et Boolean indexing



Array for the illustration

```python
# creation of an array from a nested list
my_arr = np.array([[2,13,2,4,7], [7,4,21,9,31], [23,1,8,14,3]])
print(my_arr)
```

Examples of indexing and boolean indexing

```python
# indexing
print(my_arr > 5)
```

```python
# boolean indexing
print(my_arr[my_arr > 5])
```

```python
# practical example:
# using boolean indexing to modify the values of an array
# -> broadcasting
my_arr[my_arr > 5] = 99
print(my_arr)
```

Slicing on unstructured data (e.g. images)

```python
# let's import useful librairies for images processing
from scipy import misc
import matplotlib.pyplot as plt
%matplotlib inline
```

```python
# retrieve the image in the form of a NumPy array
# visualisation of its shape (3-dimensional arrray) :
# dim 1 = gray scale on axis 1, dim 2 = gray scale on axis 2
# dim 3 = colour on axis 3 (Red, Green, or Blue)
face = misc.face()
print(face.shape)
```

```python
# display the image -> resolution of 1024 x 768
plt.imshow(face);
```

```python
# image compression, skipping 2 pixels in width and height
# resolution divided by 2: 512 x 384 => memory save
# Note: by increasing the skip step -> memory save but loss of quality
plt.imshow(face[::2,::2]);
```

```python
# Deletion of the 3rd dimension = colour
face_gray = misc.face(gray=True)
print(face_gray.shape)
```

```python
# image en noir et blanc
plt.imshow(face_gray, cmap=plt.cm.gray);
```

```python
# compression of the black and white image
plt.imshow(face_gray[::2,::2], cmap=plt.cm.gray);
```

```python
# zoom on the animal
plt.imshow(face_gray[50:490,370:945], cmap=plt.cm.gray);
```

[Outline]

Congratulations !!!

```python
from IPython.core.display import HTML
def css_styling():
    styles = open("../styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[1]: