Data Science amb Python

Cristiane de Souza da Silva

Abril 2021

Tasca 2-A : Estructura d'una Matriu**

Exercises 1

Creates an np.array of one dimension, including at least 8 integers, data type int64. Shows the size and shape of the array.

In [1]:
```
import numpy as np, pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# For creating an array of intengers, the function randit() will be used.
import random

new_array = np.array(random.sample(range(1, 100), 8))
new_array
```

Out[2]:
```
array([59, 84, 18, 24, 64, 68, 13, 93])
```

In [3]:
```
print('The data type of this array is', new_array.dtype)
print('The shape of this array is', new_array.shape)
print('The size of this array is', new_array.size)
print('The dimension of this array is', new_array.ndim)
```

```
The data type of this array is int64
The shape of this array is (8,)
The size of this array is 8
The dimension of this array is 1
```

In [ ]:
```

```

Exercises 2

From the matrix in Exercise 1, calculate the mean value of the values entered and subtract the average resulting from each of the values in the matrix.

In [4]:
```
#mean

print('The mean of this array is', np.mean(new_array))
```

```
The mean of this array is 52.875
```

In [5]:
```
# Substraction of the average from each value

np.mean(new_array) -  new_array
```

Out[5]:
```
array([ -6.125, -31.125,  34.875,  28.875, -11.125, -15.125,  39.875,
       -40.125])
```

Exercises 3

Create a two-dimensional array with a shape of 5 x 5. Extract the maximum value of the array, and the maximum values of each of its axes.

In [6]:
```
#Create a 2D array 5 x 5
array_2D = np.array(random.sample(range(1, 100), 25)).reshape(5,5)
array_2D
```

Out[6]:
```
array([[97, 28, 17, 14, 54],
       [52,  9,  5, 45, 65],
       [95, 62, 30, 70, 93],
       [55, 80, 48, 20, 40],
       [ 3,  2, 51, 42, 31]])
```

In [7]:
```
# Maximum value of array

print('The maximum value of this array is', np.amax(array_2D), '\n')

print('The maximum value along the  axe[0]  is', np.amax(array_2D, axis=0))
print('The maximum value along the axe[1]  is', np.amax(array_2D, axis=1))
```

```
The maximum value of this array is 97

The maximum value along the  axe[0]  is [97 80 51 70 93]

The maximum value along the axe[1]  is [97 65 95 80 51]
```

- Exercises 4

Show me with examples of different arrays, the fundamental rule of Broadcasting that says, "arrays can be transmitted / broadcast if their dimensions match or if one of the arrays has a size of 1."

Through broadcasting, the smaller matrix takes the form of the larger matrix so that the requested operations are carried out.

In [8]:
```
# 1-D array

first_array = np.array(random.sample(range(1, 50), 4))
print(first_array)
print(first_array.ndim)
```

```
[15 48 32 18]
1
```

In [9]:
```
# 2-D array
second_array = np.array(random.sample(range(1, 100), 12)).reshape(3,4)
print(second_array)
print(second_array.ndim)
```

```
[[26 88 54 85]
 [76 96 29  3]
 [28 57 64 27]]
2
```

In [10]:
```
print(first_array * second_array, '\n')
print(first_array + second_array)
```

```
[[ 390 4224 1728 1530]
 [1140 4608  928   54]
 [ 420 2736 2048  486]]

[[ 41 136  86 103]
 [ 91 144  61  21]
 [ 43 105  96  45]]
```

Above, it can be seen that arrays can be transmitted if their dimensions match or if one of the arrays is 1 "in size.

Exercises 5

Use Indexing to extract the values of a column and a row from the array. And add up their values.

In [11]:
```
array_index = np.array(random.sample(range(1, 100), 16)).reshape(4,4)
array_index
```

Out[11]:
```
array([[69, 82, 24, 29],
       [88, 65, 43, 87],
       [68,  9, 25, 58],
       [93, 99, 21,  6]])
```

In [12]:
```
#Extracting values

print(array_index[1,1])

print(array_index[2,3])
```

```
65
58
```

In [13]:
```
# Sum of values
array_index[1,1] + array_index[2,3]
```

Out[13]:
```
123
```

Exercises 6

Mask the above matrix, perform a vectorized Boolean calculation, taking each element and checking if it is evenly divided by four.

This returns a mask array in the same way as the elementary results of the calculation.

In [14]:
```
# Make a mask
# Divide each element by four

mask = (array_index % 4 ==0)
mask
```

Out[14]:
```
array([[False, False,  True, False],
       [ True, False, False, False],
       [ True, False, False, False],
       [False, False, False, False]])
```

Exercises 7

Then use this mask to index the original number array. This causes the array to lose its original shape, reducing it to one dimension, but you still get the data you are looking for.

In [15]:
```
array_index[mask]
```

Out[15]:
```
array([24, 88, 68])
```

Exercici 8

You will upload any image (jpg, png ..) with Matplotlib. note that RGB images (Red, Green, Blue) are really only widths × heights × 3 arrays (three channels Red, Green, and Blue), one for each color of int8 integers,

manipulate these bytes and use Matplotlib again to save the modified image once you're done.

Help: Import, import matplotlib.image as mpimg. study the mpimg.imread (()) method

Show me to see what happens when we remove the Green G or Blue B channel.

Show me what happens when we remove the Green G or Blue B channel. You should use indexing to select the channel you want to undo.

Use the method, mpimg.imsave () of the imported library, to save the modified images and you will need to upload them to your repository on github.

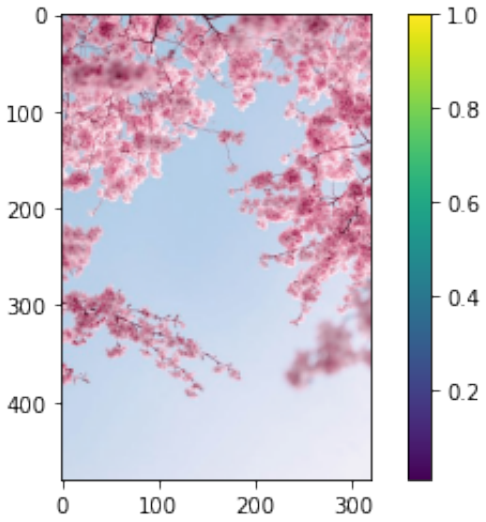
In [16]:
```
#Upload the image
import matplotlib.image as mpimg

%matplotlib inline

img = mpimg.imread('flores.png')
img
```

Out[16]:
```
array([[[0.70980394, 0.68235296, 0.7647059 , 1.        ],
        [0.6627451 , 0.64705884, 0.74509805, 1.        ],
        [0.6313726 , 0.7058824 , 0.79607844, 1.        ],
        ...,
        [0.8666667 , 0.6509804 , 0.72156864, 1.        ],
        [0.7058824 , 0.52156866, 0.6039216 , 1.        ],
        [0.5254902 , 0.34509805, 0.4509804 , 1.        ]],

       [[0.8627451 , 0.76862746, 0.827451  , 1.        ],
        [0.6627451 , 0.64705884, 0.74509805, 1.        ],
        [0.5921569 , 0.6392157 , 0.7372549 , 1.        ],
        ...,
        [0.7411765 , 0.49019608, 0.5764706 , 1.        ],
        [0.7058824 , 0.49019608, 0.5764706 , 1.        ],
        [0.6509804 , 0.4862745 , 0.5686275 , 1.        ]],

       [[1.        , 0.87058824, 0.94509804, 1.        ],
        [0.76862746, 0.6784314 , 0.7647059 , 1.        ],
        [0.5921569 , 0.6392157 , 0.7372549 , 1.        ],
        ...,
        [0.74509805, 0.45882353, 0.56078434, 1.        ],
        [0.7058824 , 0.49019608, 0.5764706 , 1.        ],
        [0.7058824 , 0.49019608, 0.5764706 , 1.        ]],

       ...,

       [[0.8117647 , 0.8627451 , 0.92941177, 1.        ],
        [0.8117647 , 0.8627451 , 0.92941177, 1.        ],
        [0.8117647 , 0.8627451 , 0.92941177, 1.        ],
        ...,
        [0.9372549 , 0.93333334, 0.9607843 , 1.        ],
        [0.9372549 , 0.93333334, 0.9607843 , 1.        ],
        [0.9372549 , 0.93333334, 0.9607843 , 1.        ]],

       [[0.8117647 , 0.8627451 , 0.92941177, 1.        ],
        [0.8117647 , 0.8627451 , 0.92941177, 1.        ],
        [0.8117647 , 0.8627451 , 0.92941177, 1.        ],
        ...,
        [0.9372549 , 0.93333334, 0.9607843 , 1.        ],
        [0.9372549 , 0.93333334, 0.9607843 , 1.        ],
        [0.9372549 , 0.93333334, 0.9607843 , 1.        ]],

       [[0.8117647 , 0.8627451 , 0.92941177, 1.        ],
        [0.8117647 , 0.8627451 , 0.92941177, 1.        ],
        [0.8117647 , 0.8627451 , 0.92941177, 1.        ],
        ...,
        [0.9372549 , 0.93333334, 0.9607843 , 1.        ],
        [0.9372549 , 0.93333334, 0.9607843 , 1.        ],
        [0.9372549 , 0.93333334, 0.9607843 , 1.        ]]], dtype=float32)
```

In [17]:
```
img.dtype
```

Out[17]:
```
dtype('float32')
```

In [18]:
```
#plot image with RGB values

#imgplot = plt.imshow(img)

imgplot = plt.imshow(img)
plt.colorbar()
plt.show()
```



According to the Decimal Code (R,G,B) float32, the color blue is (0,0,1) and green is (0,1,0) So, I create a mask with the collors different from blue.

In [19]:
```
img[:,:,2]==0
```

Out[19]:
```
array([[False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       ...,
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False]])
```

In [20]:
```
no_blue = img.copy() # Make a copy
no_blue[:, :, 2] = 0
```

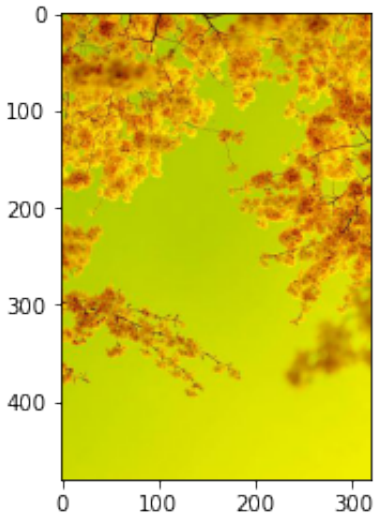
In [21]:
```
# Image without color blue
mpimg.imsave('no_blue.jpg', no_blue)
```

In [22]:
```
# Image without color green
no_green = img.copy() # Make a copy
no_green[:,:,1]= 0
mpimg.imsave('no_green.jpg', no_green)
```

In [23]:
```
img_no_blue = mpimg.imread('no_blue.jpg')
img_no_blue
```

Out[23]:
```
array([[[200, 156,   5],
        [183, 169,   8],
        [156, 184,  11],
        ...,
        [225, 167,  24],
        [178, 135,  23],
        [120,  83,  15]],

       [[231, 185,  27],
        [177, 159,   0],
        [141, 164,   0],
        ...,
        [187, 121,   0],
        [177, 128,   9],
        [167, 125,  43]],

       [[255, 217,  49],
        [195, 168,   0],
        [153, 165,   0],
        ...,
        [198, 122,   0],
        [183, 124,   0],
        [175, 127,  19]],

       ...,

       [[206, 220,   0],
        [206, 220,   0],
        [206, 220,   0],
        ...,
        [239, 238,   0],
        [239, 238,   0],
        [239, 238,   0]],

       [[206, 220,   0],
        [206, 220,   0],
        [206, 220,   0],
        ...,
        [239, 238,   0],
        [239, 238,   0],
        [239, 238,   0]],

       [[206, 220,   0],
        [206, 220,   0],
        [206, 220,   0],
        ...,
        [239, 238,   0],
        [239, 238,   0],
        [239, 238,   0]]], dtype=uint8)
```

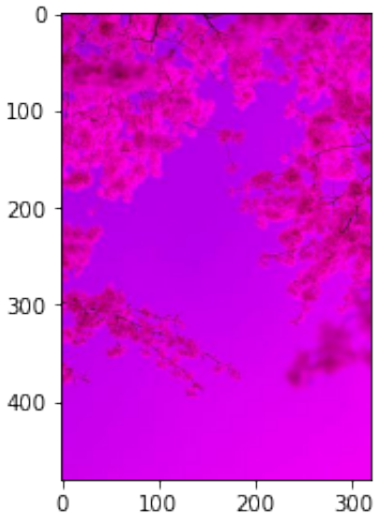
In [24]:
```
plt.imshow(img_no_blue)
plt.show()
```



In [25]:
```
img_no_green = mpimg.imread('no_green.jpg')
img_no_green
```

Out[25]:
```
array([[[194,   0, 210],
        [181,   0, 201],
        [168,   0, 196],
        ...,
        [196,  16, 175],
        [171,   5, 165],
        [142,   0, 145]],

       [[203,   6, 217],
        [185,   0, 204],
        [169,   0, 196],
        ...,
        [188,  10, 160],
        [174,   5, 158],
        [154,   0, 148]],

       [[216,  14, 224],
        [197,   0, 211],
        [175,   0, 197],
        ...,
        [185,   4, 143],
        [181,   8, 150],
        [176,   7, 150]],

       ...,

       [[205,   1, 236],
        [205,   1, 236],
        [205,   1, 236],
        ...,
        [239,   0, 244],
        [239,   0, 244],
        [239,   0, 244]],

       [[205,   1, 236],
        [205,   1, 236],
        [205,   1, 236],
        ...,
        [239,   0, 244],
        [239,   0, 244],
        [239,   0, 244]],

       [[205,   1, 236],
        [205,   1, 236],
        [205,   1, 236],
        ...,
        [239,   0, 244],
        [239,   0, 244],
        [239,   0, 244]]], dtype=uint8)
```

In [26]:
```
plt.imshow(img_no_green)
plt.show()
```



In [ ]: