```
#########
# NUMPY #
#########
import numpy as np

data = np.array([1,0])
print(data[0])
print(type(data[0]))

data = np.array([[1,0],[2,0],[3,0]])
print(data[0:])
```

```
1
<class 'numpy.int64'>
[[1 0]
 [2 0]
 [3 0]]
```

```
print(data.shape)
print(data.size)
print(data.ndim)
print(data.dtype)
```

```
(3, 2)
6
2
int64
```

```
data = np.array([[1,0],[2,0]],dtype=np.int32)
data = np.array([[1,0],[2,0]],dtype=np.complex64)
data = np.array([[1,0],[2,0]],dtype=np.float16)
# Conversión
data = np.array(data,dtype=np.uint)
data = data.astype(np.bool_)
print(data)
```

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

```
## GENERACION

data = np.array(range(10))
print(data)

data = np.arange(10)
print(data)

data = np.zeros((10,5))
# https://numpy.org/doc/stable/reference/generated/numpy.zeros.html
print(data)

data = np.ones(10)
print(data)
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

```
shape = (10,80) # 10 rows x 80 cols
data = np.zeros(shape)
print(data)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
  0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
data = np.linspace(0,1,10)
print(data)
```

```
[0.         0.11111111 0.22222222 0.33333333 0.44444444 0.55555556
 0.66666667 0.77777778 0.88888889 1.        ]
```

```
data = np.logspace(0,2,10) # 10puntos entre 2**0 y 2**2
print(data)
```

```
[  1.           1.66810054   2.7825594    4.64158883   7.74263683
  12.91549665  21.5443469   35.93813664  59.94842503 100.        ]
```

```
data = np.identity(3)
print(data)
print("-"*10)

data = np.eye(3,k=1) # https://numpy.org/doc/stable/reference/generated/numpy.eye.html
print(data)
print("-"*10)

data = np.diag(range(1,4))
print(data)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
----------
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]]
----------
[[1 0 0]
 [0 2 0]
 [0 0 3]]
```

```
## GENERACION ALEATORIA

data = np.random.rand(3,3)
print(data)
print("-"*20)

data = np.random.randint(1,10,size=(2,2))
print(data)
print("-"*20)

data = np.random.randint(1,10,20).reshape(10,2)
print(data)
print("-"*20)
```

```
data = np.array(np.random.rand(20)*10,dtype=int).reshape(10,2)
print(data)
```

```
[[0.53847165 0.31087433 0.80688781]
 [0.12235827 0.29810494 0.65231138]
 [0.45809456 0.8971178  0.20520521]]
--------------------
[[6 6]
 [2 2]]
--------------------
[[2 6]
 [6 3]
 [9 1]
 [9 3]
 [1 7]
 [8 6]
 [6 5]
 [9 2]
 [8 3]
 [6 6]]
--------------------
[[4 2]
 [8 1]
 [9 0]
 [9 6]
 [6 0]
 [5 9]
 [0 2]
 [0 6]
 [2 8]
 [8 9]]
```

```
## ACTIVITAT 1 Imaginad la siguiente actividad donde tenemos que realizar la siguiente matriz:
data = np.array(range(10))
print(data)
np.repeat([data], 10,  axis = 0)
```

```
[0 1 2 3 4 5 6 7 8 9]
array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

```
## ACTIVITAT 2 Generar la siguiente estructura a partir del array([1, 2, 3])
A = ([1, 2, 3])
a = np.repeat(A, 3)
a
```

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

```
b = np.tile(A, 3)
b
```

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

```
np.append(a, b)
```

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

```
## Carga de datos con Numpy
data = np.loadtxt(
    "GALIBOS TUNELES MADRID.csv",
    delimiter=";",
    usecols=(1,),
    skiprows=1,
    encoding="latin1",  # prueba "utf-8-sig" si fuera necesario
    converters={1: lambda s: float(s.replace(",", "."))}  # aquí s ya es str
)
print(data)
print(data.shape)
```

```
[2.95 4.   4.   4.   4.   4.   3.9  3.5  3.5  3.5  4.3  4.5  3.   3.
 2.95 4.   4.5  3.   4.   3.   4.   3.   3.   4.   3.   2.95 4.   4.
 4.   3.   2.95 2.85 2.85 3.   3.   3.   2.85 3.   3.   3.   3.   3.
 3.   2.85 4.   4.   3.   2.4  4.   4.   4.   4.3  3.5  2.7  2.95 2.95
 2.95 2.95 1.95 2.4 ]
(60,)
```

```
## ACT 3
v1 = np.arange(1,4)
v2 = np.arange(4,7)
D = np.sqrt(np.sum((v1 - v2)**2))
D
```

```
np.float64(5.196152422706632)
```

```
## REESTRUCTURANDO DIMENSION DE UNA SERIE
a = np.arange(10)
print(a.shape)
print(a.reshape(2,5))
print(a)
```

```
(10,)
[[0 1 2 3 4]
 [5 6 7 8 9]]
[0 1 2 3 4 5 6 7 8 9]
```

```
a = a.reshape(2,5)
print(a.T)
print("-"*10)
print(np.hstack(a))  # https://numpy.org/doc/stable/reference/generated/numpy.hstack.html
```

```
[[0 5]
 [1 6]
 [2 7]
 [3 8]
 [4 9]]
----------
[0 1 2 3 4 5 6 7 8 9]
```

```
b = np.arange(10,20).reshape(2,5)
print(b)
print("-"*10)
print(np.hstack((a,b))) # axis-1
print(np.vstack((a,b))) # axis-0
```

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

```
c = np.dstack((a,b)) # axis-2  https://numpy.org/doc/stable/reference/generated/numpy.dstack.html
print(c)
print(c.shape)
```

```
[[[ 0 10]
  [ 1 11]
  [ 2 12]
  [ 3 13]
  [ 4 14]]

 [[ 5 15]
  [ 6 16]
  [ 7 17]
  [ 8 18]
  [ 9 19]]]
(2, 5, 2)
```

```
print(a)
print(np.ravel(a)) # https://numpy.org/doc/stable/reference/generated/numpy.ravel.html

print(np.ravel(a,order="F")) # ‘F’ means to index the elements in column-major,
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
[0 1 2 3 4 5 6 7 8 9]
[0 5 1 6 2 7 3 8 4 9]
```

```
print(a)
print(np.split(a,2))
c1,c2 = np.split(a,2)

print("-"*10)
```

```
print(c1)
print(c1.shape)
print(np.ravel(c1))
print(c2)
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
[array([[0, 1, 2, 3, 4]]), array([[5, 6, 7, 8, 9]])]
----------
[[0 1 2 3 4]]
(1, 5)
[0 1 2 3 4]
[[5 6 7 8 9]]
```

```
print(np.concatenate((a,b)))
print("-"*10)
print(np.concatenate((a,b),axis=1))
```

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

```
# https://pillow.readthedocs.io/en/stable/
```

```
!uv pip install pillow
```

```
Audited 1 package in 15ms
```

```
from PIL import Image

image = Image.open('OIP.jpg')
# summarize some details about the image
print(image.format)
print(image.size)
print(image.mode)

display(image)
```

```
JPEG
(195, 116)
RGB
```

```
data = np.asarray(image)
print(len(data[0]))
print(data.size)
print(data.shape)
w,h,_ = data.shape
```

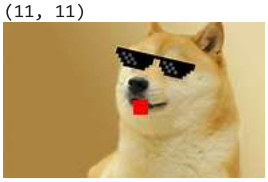
```
195
67860
(116, 195, 3)
```

```
print(data[w//2,h//2])
data2 = data.copy()
data2[w//2,h//2] = np.array([255,0,0]) # a red point

img = Image.fromarray(data2, 'RGB')

display(img)
```

```
[34 30 19]
```

```
center_mask = w//2,h//2
rectangle_size = int(w*0.1)
mask = np.ones(rectangle_size*rectangle_size).reshape(rectangle_size,rectangle_size)
print(mask.shape)

for x in range(w//2,w//2+rectangle_size):
    for y in range(h//2,h//2+rectangle_size):
        data2[x,y] =  np.array([255,0,0])


img = Image.fromarray(data2, 'RGB')
display(img)

# REALMENTE, el cuadrado está en el centro?
```

```
(11, 11)
```



```
# ACT Pasalo a blanco y negro
image_bw = image.convert("L")
image_bw
```

```
# Escala de grises (solo NumPy)
rgb = data[..., :3].astype(np.float32)
weights = np.array([0.299, 0.587, 0.114], dtype=np.float32)

gray = np.clip(np.round((rgb * weights).sum(axis=-1)), 0, 255).astype(np.uint8)  # (h, w)

# Si quieres mostrar en 'L' (1 canal):
img_gray_L = Image.fromarray(gray, mode='L')
display(img_gray_L)
```

```
# Sol profe
grayscale = np.mean(data, axis=2)
grayscale = grayscale.astype(np.uint8)
gray_img = Image.fromarray(grayscale)
gray_img
```

```
# OPERACIONES DE SLICING
a = np.arange(300).reshape(10,10,3)
print(a)
print("-"*10)
print(a[:1])
print("-"*10)

print(a[0][0])
print("-"*10)

print(a[:,0])
print("-"*10)
```

```
print(a[:,2:4])
print("-"*10)
```

```
[[[  0   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  31  32]
  [ 33  34  35]
  [ 36  37  38]
  [ 39  40  41]
  [ 42  43  44]
  [ 45  46  47]
  [ 48  49  50]
  [ 51  52  53]
  [ 54  55  56]
  [ 57  58  59]]

 [[ 60  61  62]
  [ 63  64  65]
  [ 66  67  68]
  [ 69  70  71]
  [ 72  73  74]
  [ 75  76  77]
  [ 78  79  80]
  [ 81  82  83]
  [ 84  85  86]
  [ 87  88  89]]

 [[ 90  91  92]
  [ 93  94  95]
  [ 96  97  98]
  [ 99 100 101]
  [102 103 104]
  [105 106 107]
  [108 109 110]
  [111 112 113]
  [114 115 116]
  [117 118 119]]

 [[120 121 122]
  [123 124 125]
  [126 127 128]
  [129 130 131]
  [132 133 134]
  [135 136 137]
  [138 139 140]
  [141 142 143]
  [144 145 146]
  [147 148 149]]

 [[150 151 152]
  [153 154 155]
  [156 157 158]
```

```
# ACT
# https://e2eml.school/convert_rgb_to_grayscale.html
data_gray = data[:,:,0] * 0.2126 + data[:,:,1] * 0.7152 + data[:,:,2] * 0.0722
print(data.shape)
img = Image.fromarray(np.uint8(data_gray))
display(img)
```

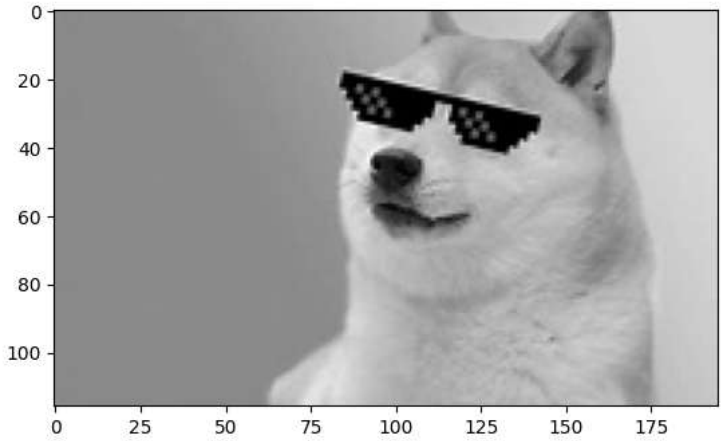
```
(116, 195, 3)
```

```
# Way 3

rgbcorrection = np.array([0.2989, 0.5870, 0.1140])

data = np.asarray(image)
print(data.shape)
data2 = np.dot(data,rgbcorrection)
```

```
print(data2.shape)
img = Image.fromarray(np.uint8(data2))
display(img)


# as a plot
import matplotlib.pyplot as plt
plt.imshow(data2, cmap = plt.get_cmap(name = 'gray'))
plt.show()
```

```
(116, 195, 3)
(116, 195)
```

```
Matplotlib is building the font cache; this may take a moment.
```




```
# FUNCIONES PROPIAS VECTORIZADAS
temperatura = np.random.randint(-10,43,1000)
```

```
temperatura[temperatura>33]
```

```
array([38, 38, 37, 35, 42, 42, 35, 39, 36, 38, 37, 37, 37, 40, 34, 41, 41,
       40, 42, 36, 41, 37, 34, 42, 35, 35, 36, 34, 39, 34, 37, 37, 34, 36,
       35, 39, 40, 42, 40, 37, 40, 39, 40, 42, 34, 38, 42, 36, 36, 34, 40,
       41, 38, 40, 37, 36, 42, 37, 35, 42, 37, 39, 41, 39, 38, 42, 34, 39,
       36, 40, 40, 37, 39, 39, 34, 42, 36, 42, 40, 35, 42, 36, 40, 36, 34,
       36, 35, 36, 39, 42, 35, 40, 38, 34, 39, 38, 36, 34, 37, 41, 35, 36,
       42, 37, 35, 41, 37, 36, 42, 34, 38, 36, 37, 37, 42, 36, 35, 39, 37,
       38, 34, 36, 34, 36, 38, 41, 38, 35, 35, 38, 42, 38, 35, 36, 38, 37,
       39, 37, 40, 38, 41, 40, 37, 34, 41, 35, 35, 34, 40, 41, 40, 38, 38,
       37, 42, 39, 42, 41, 42, 38, 34, 42, 35, 39, 40, 36, 36, 35, 36, 37,
       41, 37, 40], dtype=int32)
```

```
def isHot(grados):
    if grados>33 and grados<=40:
        return True

def isHot(grados):
    return grados>33 and grados<=40
# def equivalentes

fHot = np.vectorize(isHot)
print(fHot(temperatura))
```

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

```
 True False False  True False False False False False False False False
False  True False False False  True False False False False False False
False False False False False False False False False False False False
False False False  True 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
 True False  True False  True False False False False False False False
False False False False  True False False False False False False False
False False False False False False False  True False False False False
False False False False False False False False False False  True False
False False False False False False  True False False False False False
 True False False False False  True False False False False  True False
False  True False False False  True False False False False  True  True
 True False False False False False  True  True False False False False
False False False False  True False  True False False False False False
False False False False  True False False False False False  True False
False False False  True False False False False False False False False
False False False False False False  True False  True False False False
False False False False False False  True  True False False False False
False  True False False  True False False False False False False False
False False False False False  True False False False False  True False
False False False False  True False False False False False False False
False False False  True False False False False False False False False
False False False False False False False False False  True False  True
 True False False False False False False False False  True False False
 True False False False False False False False False  True False False
False  True False False False  True False False False False False False
False False False False  True False False False False False  True  True
False False  True False  True False False False False False False False
False False  True False False False False False False False False False
False False False False False False False False False  True  True False
False False False False False False False False False False False False
False False False False False  True False False False False False  True
False False False False False False False  True False False False False
False False False False False  True  True False False False False False
False  True  True  True False False  True False False  True False False
False False False False False False False False False  True  True False
 True False False False False False  True False False False False False
```

```
temperatura[fHot(temperatura)]
```

```
array([38, 40, 38, 35, 37, 37, 37, 40, 40, 34, 38, 36, 40, 38, 36, 40, 39,
       40, 38, 37, 34, 39, 34, 34, 38, 35, 36, 37, 38, 35, 36, 40, 40, 38,
       35, 38, 36, 38, 35, 38, 35, 39, 34, 36, 37, 37, 34, 36, 39, 39, 34,
       34, 35, 35, 39, 40, 40, 37, 35, 39, 40, 38, 39, 34, 35, 35, 39, 36,
       40, 40, 36, 39, 40, 37, 36, 39, 40, 40, 36, 36, 36, 39, 39, 35, 39,
       40, 35, 37, 35, 38, 39, 40, 36, 39, 36, 35, 37, 34, 36, 35, 36, 39,
       39, 35, 38, 35, 34, 34, 36, 34, 34, 36, 35, 38, 38, 40, 37, 35, 38,
       39, 35, 37, 35, 39, 36, 39, 37, 34, 39, 34, 37, 37], dtype=int32)
```

```
#Alternativa
# https://numpy.org/doc/stable/reference/routines.logic.html
np.logical_and(temperatura>30,temperatura<=40)
```

```
array([ True, False, False, False, False, False, False, False, False,
       False, False, False,  True, False, False,  True, False, False,
       False, False, False, False, False, False,  True, False,  True,
       False,  True, False,  True, False, False, False,  True,  True,
       False, False, False, False,  True, False, False, False, False,
        True, False,  True, False, False, False, False,  True, False,
       False, False,  True, False, False, False,  True, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False,  True, False, False, False,
       False, False, False, False, False, False, False,  True, False,
       False, False, False,  True, False, False, False, False, False,
       False, False, False,  True, False, False, False,  True,  True,
       False, False, False, False,  True, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False,  True, False, False, False,
        True,  True, False, False,  True, False, False, False,  True,
       False,  True,  True,  True, False, False, False, False, False,
       False, False, False, False, False, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
       False,  True, False,  True, False, False, False, False, False,
       False,  True, False, False,  True, False,  True,  True, False,
       False, False, False, False, False, False,  True, False, False,
        True, False, False,  True, False, False, False, False, False,
       False,  True, False, False, False,  True,  True, False, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False,  True,
        True, 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,  True, False,  True,
       False,  True, False, False, False, False, False,  True, False,
```

```
       False, False, False, False,  True, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False,  True, False,
       False, False, False, False, False, False,  True, False, False,
       False, False, False,  True, False, False, False, False, False,
       False,  True, False, False,  True, False, False,  True, False,
       False, False,  True, False, False, False, False,  True,  True,
        True, False,  True, False, False, False,  True,  True, False,
       False, False, False, False,  True, False, False,  True, False,
        True, False,  True, False, False, False, False, False, False,
       False,  True, False, False, False, False, False,  True, False,
       False, False, False,  True, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False,  True,  True, False,  True, False, False,
       False, False, False, False,  True, False,  True, False, False,
       False, False, False, False, False, False,  True,  True, False,
       False, False, False, False,  True, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
       False, False,  True, False, False,  True, False, False, False,
       False,  True, False, False, False, False,  True, False, False,
        True,  True, False, False, False, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
        True, False, False, False, False, False, False,  True, False,
       False, False, False, False,  True, False, False, False, False,
        True, False,  True,  True, False, False, False, False, False,
```

```
isHot = lambda x: (x>30 and x<=40)
index = list(map(isHot,temperatura))
temperatura[index]
# IGNORAR
```

```
array([38, 33, 31, 31, 40, 38, 35, 37, 32, 37, 37, 32, 31, 40, 40, 34, 38,
       31, 36, 40, 38, 36, 31, 40, 39, 32, 40, 38, 31, 32, 32, 37, 34, 31,
       39, 34, 34, 38, 35, 36, 31, 37, 38, 31, 33, 32, 35, 36, 40, 40, 31,
       38, 35, 38, 36, 38, 35, 38, 35, 39, 34, 36, 37, 33, 37, 34, 33, 36,
       39, 33, 39, 34, 34, 35, 31, 35, 39, 40, 40, 37, 35, 39, 40, 38, 31,
       39, 32, 34, 35, 35, 33, 31, 39, 36, 40, 40, 36, 39, 40, 37, 36, 39,
       40, 40, 36, 32, 36, 33, 31, 36, 39, 31, 39, 35, 39, 32, 33, 40, 35,
       37, 35, 38, 33, 39, 32, 40, 36, 39, 36, 31, 35, 37, 34, 36, 32, 33,
       35, 36, 33, 39, 39, 35, 32, 38, 35, 34, 34, 32, 36, 34, 34, 32, 32,
       36, 35, 38, 38, 40, 31, 37, 35, 38, 39, 35, 37, 35, 39, 36, 39, 37,
       31, 34, 39, 34, 37, 37], dtype=int32)
```

```
 # MAS FUNCIONES ALEATORIAS
np.random.seed(2022)
a = np.random.randint(-90,0,100)

index = np.where(a<-80) # Alerta: Son índices
print(index)
print(a[index]) # valores de las posiciones
```

```
(array([21, 50, 66, 71, 78, 91, 97]),)
[-88 -88 -81 -83 -86 -86 -88]
```

```
print(np.logical_or(a<-80,a<-90))
```

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

```
print(np.logical_not(np.logical_or(a<-40,a<-90)))
```

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

```
# ACT De array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) a array([ 0, -1,  2, -1,  4, -1,  6, -1,  8, -1])

act = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
modul = act%2
print(modul)
```

```
print("-"*10)
act[modul == 1] = -1
print(act)
```

```
[0 1 0 1 0 1 0 1 0 1]
----------
[ 0 -1  2 -1  4 -1  6 -1  8 -1]
```

```
# OPERACIONES CON GRUPOS
np.random.seed(2022)
a = np.random.randint(-30,45,100)
print(a)
```

```
[ 15  19  25 -12  -6 -14  23  11   3  -3 -19 -11  18 -11   8  42 -16 -14
 -19 -28 -15  -7   7  26 -18  15 -16 -17  22  12  23  34  15  33  -7   7
  -9  31  26  16 -28  34  16  11  31  31  42  34  21   3  -1  -7  17  19
  25  -2 -21  40  17   1  -4 -23  35  12  32   2 -26 -20  -5  23  32  -7
  -6  34  30   5 -26 -11  19  20  29 -28  -3  41  24 -27  -9  37 -19  15
  37  14  26   3   5  -5  25 -24 -29  -4]
```

```
9 in a
```

```
False
```

```
if -14 in a and not -6 in a:
    print("Something strange")
elif 45 in a:
    print("No 11")
else:
    print("Pues está el -14 y el -6, y no el 45")
```

```
Pues está el -14 y el -6, y no el 45
```

```
#https://numpy.org/doc/stable/reference/generated/numpy.unique.html?highlight=unique#numpy.unique

unique_a = np.unique(a)  # sort but
print(unique_a)
```

```
[-29 -28 -27 -26 -24 -23 -21 -20 -19 -18 -17 -16 -15 -14 -12 -11  -9  -7
  -6  -5  -4  -3  -2  -1   1   2   3   5   7   8  11  12  14  15  16  17
  18  19  20  21  22  23  24  25  26  29  30  31  32  33  34  35  37  40
  41  42]
```

```
unique_a, freq_a = np.unique(a,return_counts=True)
print(a)
print(len(a))
print("-"*10)
print(freq_a) # de menor a mayor
print(len(freq_a))
print("-"*10)
print(unique_a)
print(len(unique_a))

# ¿Cuántos elementos repetidos hay?
```

```
[ 15  19  25 -12  -6 -14  23  11   3  -3 -19 -11  18 -11   8  42 -16 -14
 -19 -28 -15  -7   7  26 -18  15 -16 -17  22  12  23  34  15  33  -7   7
  -9  31  26  16 -28  34  16  11  31  31  42  34  21   3  -1  -7  17  19
  25  -2 -21  40  17   1  -4 -23  35  12  32   2 -26 -20  -5  23  32  -7
  -6  34  30   5 -26 -11  19  20  29 -28  -3  41  24 -27  -9  37 -19  15
  37  14  26   3   5  -5  25 -24 -29  -4]
100
----------
[1 3 1 2 1 1 1 1 3 1 1 2 1 2 1 3 2 4 2 2 2 2 1 1 1 1 3 2 2 1 2 2 1 4 2 2 1
 3 1 1 1 3 1 3 3 1 1 3 2 1 4 1 2 1 1 2]
56
----------
[-29 -28 -27 -26 -24 -23 -21 -20 -19 -18 -17 -16 -15 -14 -12 -11  -9  -7
  -6  -5  -4  -3  -2  -1   1   2   3   5   7   8  11  12  14  15  16  17
  18  19  20  21  22  23  24  25  26  29  30  31  32  33  34  35  37  40
  41  42]
56
```

```
unique_a,index_a,freq_a = np.unique(a,return_counts=True,return_index=True)
print(freq_a)
print(index_a) # Donde esta colocado
print("-"*10)
print(np.where(freq_a==4))
```

```
print(unique_a[17]) # se repite cuatro veces
print(unique_a[33]) # se repite cuatro veces
print(unique_a[50]) # se repite cuatro veces

print(a[np.where(a==-7)])
```

```
[1 3 1 2 1 1 1 1 3 1 1 2 1 2 1 3 2 4 2 2 2 2 1 1 1 1 3 2 2 1 2 2 1 4 2 2 1
 3 1 1 1 3 1 3 3 1 1 3 2 1 4 1 2 1 1 2]
[98 19 85 66 97 61 56 67 10 24 27 16 20  5  3 11 36 21  4 68 60  9 55 50
 59 65  8 75 22 14  7 29 91  0 39 52 12  1 79 48 28  6 84  2 23 80 74 37
 64 33 31 62 87 57 83 15]
----------
(array([17, 33, 50]),)
-7
15
34
[-7 -7 -7 -7]
```

```
np.sort(freq_a) # Ordena
```

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

```
print(np.array([0,4,1,2,5,7,9])[::-1]) # ordena de mayora menor [::-1]
print(np.argsort(np.array([0,4,1,2,5,7,9]))) # ordena el array y muestra los indices asignados a cada posición del array
print(np.argsort(np.array([0,4,1,2,5,7,9]))[::-1])

print("-"*10)

index_sorted = np.argsort(freq_a)[::-1] #https://numpy.org/doc/stable/reference/generated/numpy.argsort.html
print(index_sorted)
```

```
[9 7 5 2 1 4 0]
[0 2 3 1 4 5 6]
[6 5 4 1 3 2 0]
----------
[50 33 17 43 44 37 47 41 15  8 26  1 52 48 34 18 31 28 27 35  3 21 16 55
 13 19 20 30 11 39 38 36 53 54 51 49 45 46 42 40 32 22 25 24 29 23 12 14
  9 10  7  6  4  5  2  0]
```

```
unique_a[index_sorted]
```

```
array([ 34,  15,  -7,  25,  26,  19,  31,  23, -11, -19,   3, -28,  37,
        32,  16,  -6,  12,   7,   5,  17, -26,  -3,  -9,  42, -14,  -5,
        -4,  11, -16,  21,  20,  18,  40,  41,  35,  33,  29,  30,  24,
        22,  14,  -2,   2,   1,   8,  -1, -15, -12, -18, -17, -20, -21,
       -24, -23, -27, -29], dtype=int32)
```

```
# ACT ¿Cuál es el color más frecuente en la imagen? Sustituye esos pixeles por un color azul: rgb=(0,0,255)
from PIL import Image

image = Image.open('OIP.jpg')
# summarize some details about the image
print(image.format)
print(image.size)
print(image.mode)

display(image)
```

```
JPEG
(195, 116)
RGB
```

```
a = np.asarray(image)
unique_a,index_a,freq_a = np.unique(a,return_counts=True,return_index=True)
print(freq_a)
print(len(freq_a))
print(np.max(freq_a))
print(np.where(freq_a==4771))
```

```
[ 572  166  140  100   84   62   58   45   37   27   26   22   20   25
   20   14   12   12   19   23   23   15   25   16   16   23   26   21
   20   26   24   29   30   23   22   25   34   37   26   39   32   33
   23   33   38   35   32   31   35   30   44   54   41   46   52   43
   48   50   53   60   60   57  229 4771 1809  301  336  203  204  320
```

```
  302  166  142  136  181  209  212  127  147  131  132  113  139  155
  149  151  130  112  130  131  102   79   81   94   83  103  104   91
  104   96  114  121  121  119  110  118  118  131  101  119  131  130
  117  148  159  145  155  149  137  156  159  156  146  144  160  160
  144  156  161  149  170  233  973 4606 1795  593  364  476  462  339
  315  336  312  281  304  290  277  312  257  286  282  224  210  237
  220  276  279  289  246  267  333  257  228  301  346  682  371  244
  246  371 4345 2494  829  525  731  553  422  415  331  355  337  364
  379  402  427  384  373  340  346  385  351  437  467  430  456  441
  471  426  454  425  402  418  437  427  423  354  339  381  401  370
  333  411  427  368  687  296  205  175  181  165  198  182  232  297
  294  232  197  245  242  225  539  179   59   51   46   41   25   27
   25   21   11   11    6    5    2    2    3    4    2    1    1    3
    4    2    8]
255
4771
(array([63]),)
```

```
# sol profe
image_np = np.asarray(image)
print(image_np.shape)
print(image_np.reshape(116*195,3))
print(image_np.reshape(116*195,3).shape)
print(image_np.reshape(image_np.shape[0]*image_np.shape[1],image_np.shape[2]).shape) # = linea anterior + profesional
print(image_np.reshape(-1,image_np.shape[2]).shape) # equivalente
print(np.unique(image_np.reshape(image_np.shape[0]*image_np.shape[1],image_np.shape[2]),axis=0).shape)
unique, freqs = np.unique(image_np.reshape(image_np.shape[0]*image_np.shape[1],image_np.shape[2]),axis=0, return_counts=Tru
max(freqs)
unique[np.argmax(freqs)]
# inacabado
```

UNION Y DIFERENCIA DE CONJUNTOS La intersección, unión y diferencia de conjuntos es una operación común en álgebra de conjuntos y análisis de datos. Numpy proporciona funciones eficientes para realizar estas operaciones en arrays:

np.isin: Comprueba si los elementos de un array están presentes en otro array.

np.intersect1d: Encuentra los elementos comunes entre dos arrays.

np.setdiff1d: Encuentra los elementos que están en un array pero no en otro.

np.union1d: Encuentra todos los elementos únicos presentes en ambos arrays.

```
a = np.arange(10)
b = np.arange(5,15)
print(a)
print(b)
print("-"*10)
print(np.isin(a,b))
print(np.intersect1d(a,b))
print("-"*10)
print(np.setdiff1d(a,b))
print(np.setdiff1d(b,a))
print("-"*10)
print(np.union1d(a,b))
```

```
[0 1 2 3 4 5 6 7 8 9]
[ 5  6  7  8  9 10 11 12 13 14]
----------
[False False False False False  True  True  True  True  True]
[5 6 7 8 9]
----------
[0 1 2 3 4]
[10 11 12 13 14]
----------
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

```
# ACT ¿Cómo podemos conseguir encontrar valores pico, valores mayores sobre sus vecinos? .SING .DIFF
a = np.array([0, 2, 2, 3, 4, 54, 6, 7, 80, 9])

print(np.diff(a))
print(np.sign(np.diff(a)))
print(np.diff(np.sign(np.diff(a))))
```

```
[  2   0   1   1  50 -48   1  73 -71]
[ 1  0  1  1  1 -1  1  1 -1]
[-1  1  0  0 -2  2  0 -2]
```

```
# ACT Existe alguna columna o fila que sólo tenga una única incógnita?

sudoku = np.array([[5,3,0,0,7,0,0,0,0],
                   [6,0,0,1,9,5,0,0,0],
```

```
                    [1,9,8,0,0,0,0,6,0],
                    [8,0,0,0,6,0,0,0,3],
                    [4,0,0,8,0,3,0,0,1],
                    [7,0,0,0,2,0,0,0,6],
                    [0,6,0,0,0,0,2,8,0],
                    [3,8,0,4,1,9,7,2,5],
                    [4,0,0,0,8,0,0,7,9]])

ceros_por_fila = np.count_nonzero(sudoku == 0, axis=1)
ceros_por_col = np.count_nonzero(sudoku == 0, axis=0)

print(ceros_por_fila, ceros_por_col)
print(np.where(ceros_por_fila==1))
print(np.where(ceros_por_col==1))
```

```
[6 5 5 6 5 6 6 1 5] [1 5 8 6 3 6 7 5 4]
(array([7]),)
(array([0]),)
```