S2 T2: Estructura de una Matriu

- Ejercicio 1

Crea un np.array de una dimensión, que incluya al menos 8 números enteros,

fecha type int64. Muestra la dimensión y la forma de la matriz.

In [7]:

```python
import numpy as np
# sea v1 la matriz de dimensión uno
v1 = np.array ( [ 11,3, -7, -1, 2, 1, -5, 19, 23, 17 ], dtype = "int64")

print (" la dimensión del vector es ", v1. ndim)
print (v1)
```

```
 la dimensió del vector es  1
[11  3 -7 -1  2  1 -5 19 23 17]
```

-Ejercicio 2

De la matriz del ejercicio 1, calcula el valor medio de los valores introducidos

y resta la media resultante de cada uno de los valores de la matriz.

In [8]:

```python
med= v1. mean()
med = int ( med)

v2 = v1 - med

print (v2)
```

```
[  5  -3 -13  -7  -4  -5 -11  13  17  11]
```

- Ejercicio 3

Crea una matriz bidimensional con una forma de 5 x 5. Extrae el valor máximo

de la matriz, y los valores máximos de cada uno de sus ejes.

In [9]:

```python
# sea la matriz Mij
Mij = np.zeros((5,5))

import random

for x in range(0,5):
    for y in range (0,5) :
        Mij [x][y] = random.randint (0,30) * ( -1)** ( random.randint (0,10))
print (Mij)
#vamos a calcular el valor máximo
max = np.max( Mij )
print ( " \n el valor máximo de la matriz es : ", max)

# ahora los valores máximos a través de sus ejes
max0 = np.max( Mij, axis = 0 )
print ( " \n el valor máximo  a través del eje 0 : ", max0)

max1 = np.max( Mij, axis = 1 )
print ( " \n el valor máximo  a través del eje 1 : ", max1)
```

```
[[ 27. -28.  15.   8.   7.]
 [ 17.   5.  26. -24.  -6.]
 [ 19. -16.  15.  20.   0.]
 [  5.  26. -29.   1.   0.]
 [ -8.  25.   2.  20.  -8.]]

 el valor máximo de la matriz es :  27.0

 el valor máximo  a través del eje 0 :  [27. 26. 26. 20.  7.]

 el valor máximo  a través del eje 1 :  [27. 26. 20. 26. 25.]
```

- Ejercicio 4

Muéstrame con ejemplos de diferentes matrices, la regla fundamental de Broadcasting que dice: "las matrices se pueden transmitir

/ broadcast si sus dimensiones coinciden o si una de las matrices tiene un tamaño de 1".

In [14]:

```python
# pongamos un ejemplo de transmisibilidad de matrices que no funciona, al no ser estas de dimensiones compatibl

a1 = np.array ( [[0,1,1], [1,0,1]])
a2= np.array ( [1,1])
a1+a2

# en este caso las dimesniones no coinciden y la dimension de una de sus direeciones no es uno
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_22616/3272317467.py in <module>
      3 a1 = np.array ( [[0,1,1], [1,0,1]])
      4 a2= np.array ( [1,1])
----> 5 a1+a2
      6
      7 # en este caso las dimesniones no coinciden y la dimension de una de sus direeciones no es uno

ValueError: operands could not be broadcast together with shapes (2,3) (2,)
```

In [97]:

```python
# vamos a expandir la matriz a2, añadiendo una dimension más con un solo elelmento
# al no coincidir la cantidad de elementos de la matriz, ni que uno de ellos sea uno
a3 = a2 .reshape(2,1)
print (a3)
print ("\n")

print ( a3 +a1)
```

```
[[1]
 [1]]


[[1 2 2]
 [2 1 2]]
```

La matriz **a1** es de 3 columnas por dos filas, mientras que **a3** es de 1 columna por dos filas. Lo que hace la transmisibilidad es trasponer a **a3**, y lo expande dos veces, tal que una columna de [[1][1]] se convierte en una matriz [[1,1,1,][1,1,1]], y al tener el mismo número de elementos puede luego sumarlas

In [12]:

```python
# ejemplo 2, sea una matriz de todo unos
unos = np. ones ( (3,5,1), dtype= "int")

# sea otra matriz de todo 1
unos2 = np. ones ( (3,4,1), dtype= "int")

print ( unos+ unos2)
# nos mandará un mensaje de error puesto que nos son coincidentes en elementos de matriz
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_22616/3829991584.py in <module>
      5 unos2 = np. ones ( (3,4,1), dtype= "int")
      6
----> 7 print ( unos+ unos2)
      8 # nos mandará un mensaje de error puesto que nos on coincidentes en elementos de matriz

ValueError: operands could not be broadcast together with shapes (3,5,1) (3,4,1)
```

In [15]:

```python
# Supongamos que canviamos la matriz unos2, canviandola dimensionalidad de las filas a uno
unos2 = np. ones ( (3,1,1), dtype= "int")
print ( unos+ unos2)
```

```
[[[2]
  [2]
  [2]
  [2]
  [2]]

 [[2]
  [2]
  [2]
  [2]
  [2]]

 [[2]
  [2]
  [2]
  [2]
  [2]]]
```

Podemos observas que al reducir la dimensión discrepante(las filas) a uno, esto permite que la matriz **unos2**

pueda expandirse 5 veces, hasta igualar el volumen de la primera.

Podemos también expandir una matriz añadiendo dimensiones;

sea por ejemplo **unos3** una matriz de dimension dos en vez de tres.

In [16]:

```python
unos3 = np.ones(( 1,1), dtype= "int")
print ( unos + unos3)
```

```
[[[2]
  [2]
  [2]
  [2]
  [2]]

 [[2]
  [2]
  [2]
  [2]
  [2]]

 [[2]
  [2]
  [2]
  [2]
  [2]]]
```

En este caso la matriz unos 3 aumenta una dimension más y se expande hasta ser una matriz de (5,3,1)

- Ejercicio 5 Utilice la Indexación para extraer los valores de una columna y una fila de la matriz. Y suma sus valores.

In [17]:

```python
# reutilizamos la matriz Mij del ejercicio 2
import random
print (Mij)
index= Mij. shape
```

```
[[ 27. -28.  15.   8.   7.]
 [ 17.   5.  26. -24.  -6.]
 [ 19. -16.  15.  20.   0.]
 [  5.  26. -29.   1.   0.]
 [ -8.  25.   2.  20.  -8.]]
```

In [18]:

```python
# para sacar los valores de una matriz vamos a usar a función a la que llamaremos suma
def suma (x, index):
    sumi= 0
    sumj= 0 # creamos dos variables auxiliares para hacer la suma
    u  =   random. randint( 0, index [0]-1)# servirá cómo indice de fila
    v   =  random. randint( 0, index [1]-1)#servirá cómo indice de columna
    for s in range ( 0, index [1]):
        sumj = sumj + Mij  [u][s]
    for t in range ( 0, index [0]):
        sumi = sumi + Mij  [t][v]
    print ( " la suma de valores de la fila ", u, " es igual a " , sumj )
    print ( " la suma de valores de la columna ", v, " es igual a " , sumi )
    print ( " y la suma de todos los valores es ", sumj+sumi)

suma ( Mij, index)
```

```
 la suma de valores de la fila  2  es igual a  38.0
 la suma de valores de la columna  2  es igual a  29.0
 y la suma de todos los valores es  67.0
```

- Ejercicio 6

Mask la matriz anterior, realice un cálculo booleano vectorizado, cogiendo cada elemento y comprobando si se divide

uniformemente por cuatro.Esto devuelve una matriz de mask de la misma forma con los resultados elementales del cálculo.

In [21]:

```python
# el objetivo es averiguar que elementos de la matriz son divisibles por 4,
# tal que M[i][j]% =0, mediante una operacon que no implique ningún tipo de
# iteración

# primero, para no alterar la matriz anterior, vamos hacer una copia  ésta
Mij2 = Mij.copy()
def fun ( mat):# creamos una función que calcula que valores son divisibles uniformemte
    mask = ((mat%4 == 0)&(mat!= 0))
    return mask
vect =np. vectorize ( fun)# vectoriza el cálculo en una sola operación
mascara = np.array (vect (Mij2))
print (mascara)
```

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

- Ejercicio 7

A continuación, utilice esta máscara para indexar en la matriz de números original.

Esto hace que la matriz pierda su forma original,

reduciéndola a una dimensión, pero todavía obtenga los datos que está buscando.

In [24]:

```python
# para ello indexando a la matriz la mascara
Mij3 = Mij2.copy()
Mij4 = Mij2.copy()
Mij3 [mascara]
```

Out[24]:

```
array([-28.,   8., -24., -16.,  20.,  -8.,  20.,  -8.])
```

In [ ]:

- Ejercicio 8

Cargad cualquier imagen (jpg, png..) con Matplotlib. Date cuenta de que las imágenes RGB (Red, Green, Blue) son realmente sólo

anchuras × alturas × 3 matrices (tres canales Rojo, Verde y Azul), una por cada color de números enteros int8,

manipule estos bytes y vuelva a utilizar Matplotlib para guardar la imagen modificada una vez haya terminado.

Ayuda:Importe, import matplotlib.image as mpimg. estudie el método de mpimg.imread(()

Muéstrame a ver qué ocurre cuando eliminamos el canal G Verde o B Azul.

Muéstrame a ver qué ocurre cuando eliminamos el canal G Verde o B Azul.

Deberías utilizar la indexación para seleccionar el canal que desea anular.

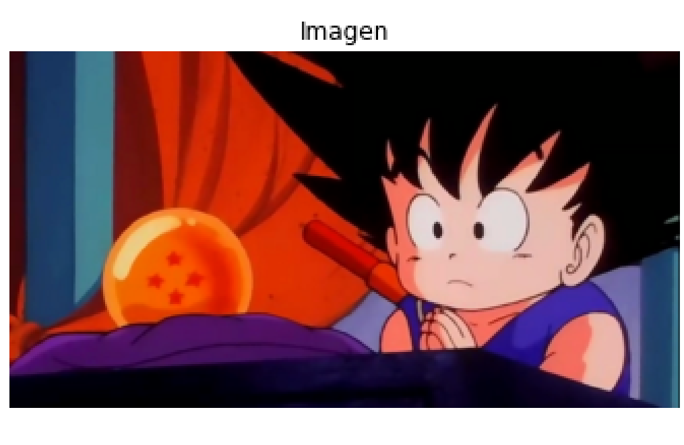
Utilizar el método, mpimg.imsave () de la librería importada, para guardar

las imágenes modificadas y que deberá subir a su repositorio en github.

In [92]:

```python
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
img = mpimg.imread("goku.png")
plt.imshow(img)
plt.title('Imagen')
plt.axis('off')
print ( img.shape)# para ver la forma de la array


plt.show()



mpimg.imsave("img.png", img)
```

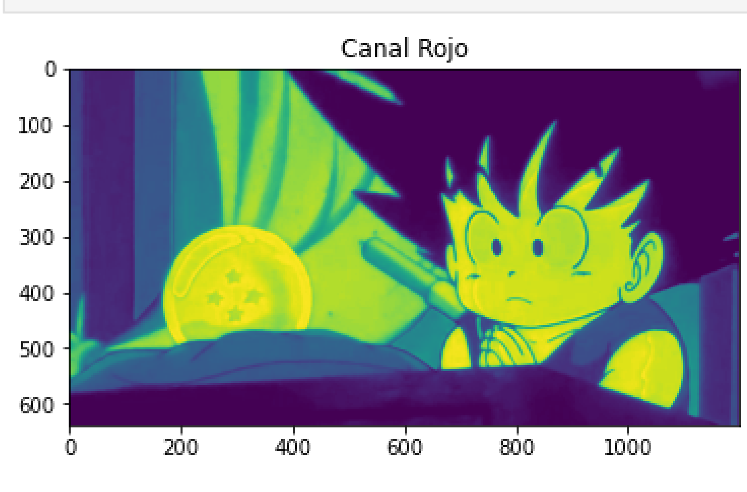
```
(638, 1200, 4)
```

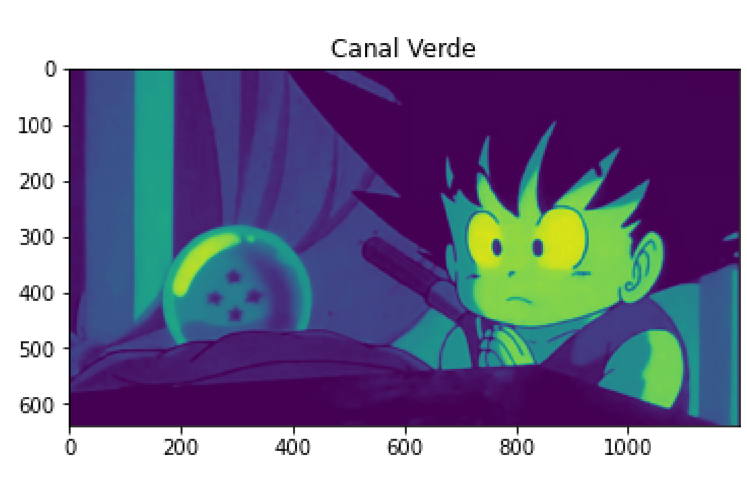


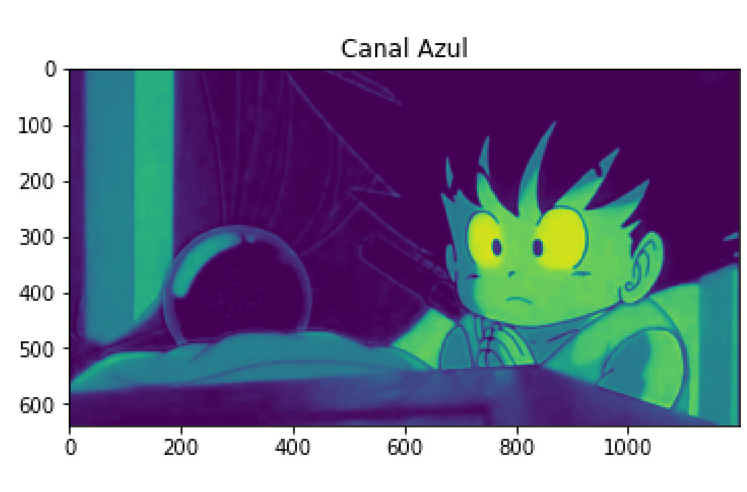
Imagen

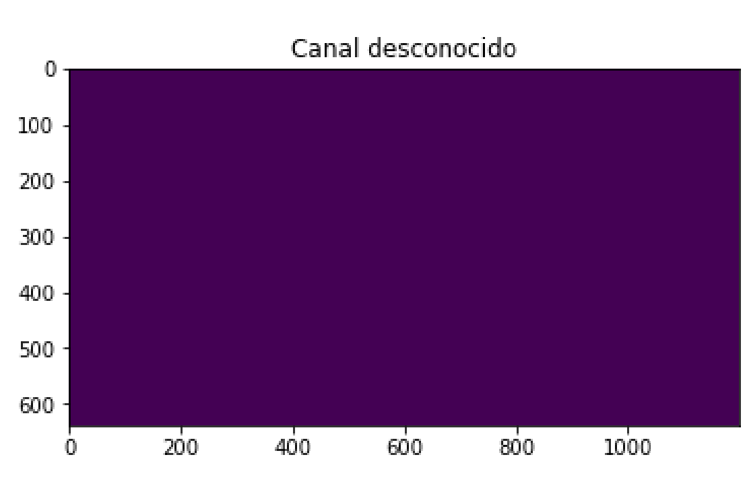
In [59]:

```python
# vemos que tiene cuatro canales de color en vez de 3
#vamos a manipularlos.

plt.imshow(img[:,:,0])
plt.title("Canal Rojo")
plt.show()
plt.imshow(img[:,:,1])
plt.title("Canal Verde")
plt.show()
plt.imshow(img[:,:,2])
plt.title("Canal Azul")
plt.show()
plt.imshow(img[:,:,3])
plt.title("Canal desconocido")
plt.show()
```


Canal Rojo


Canal Verde


Canal Azul


Canal desconocido

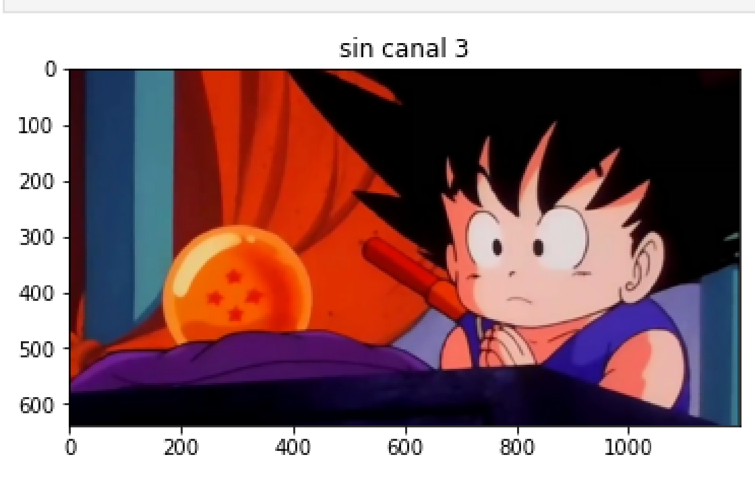
In [70]:

```python
# vamos a desmontar la array en los cuatro canales.
col0 = img[:,:,0].copy()
col1 = img[:,:,1].copy()
col2 = img[:,:,2].copy()
col3 = img[:,:,3].copy()

# miremos como es este último canal, el desconocido
print (col3)
```

```
[[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. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]]
```

In [76]:

```python
# vemos que es uniforme, probemos que pasa si quitamos esta capa
colprueba= img[:,:,0:3].copy()
plt.imshow(colprueba)
plt.title("sin canal 3")
plt.show()

# y comparamos con la imagen normal
plt.imshow(img)
plt.title("imagen sin filtro")
plt.show()
```


sin canal 3

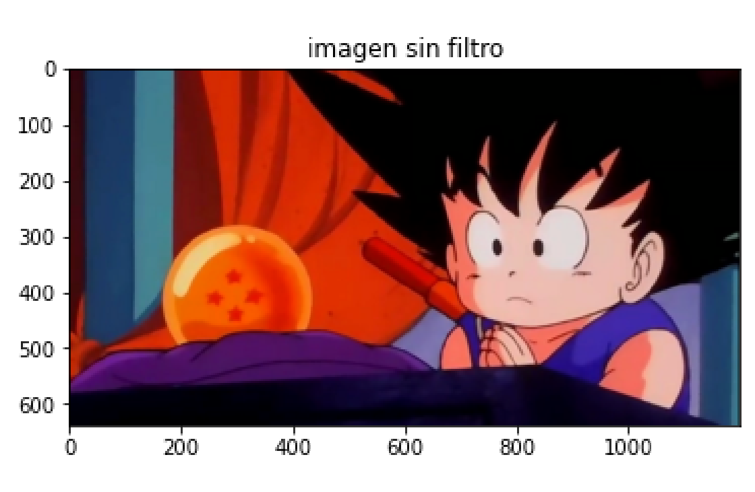


imagen sin filtro

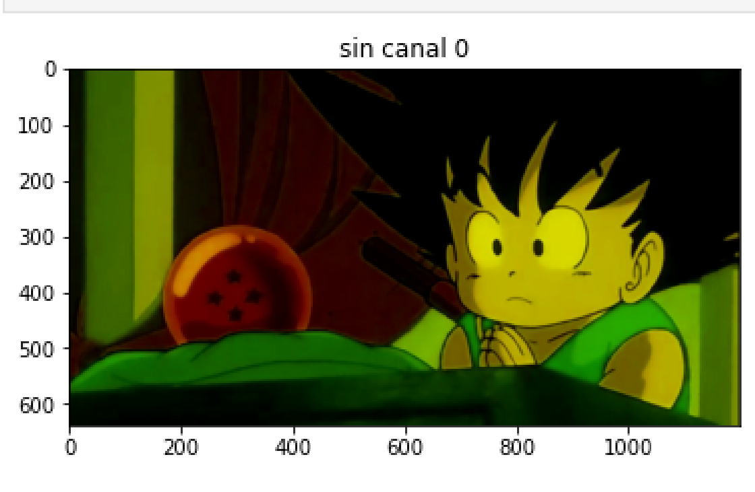
In [ ]:

```python
# observamos que son idénticas
```

In [ ]:

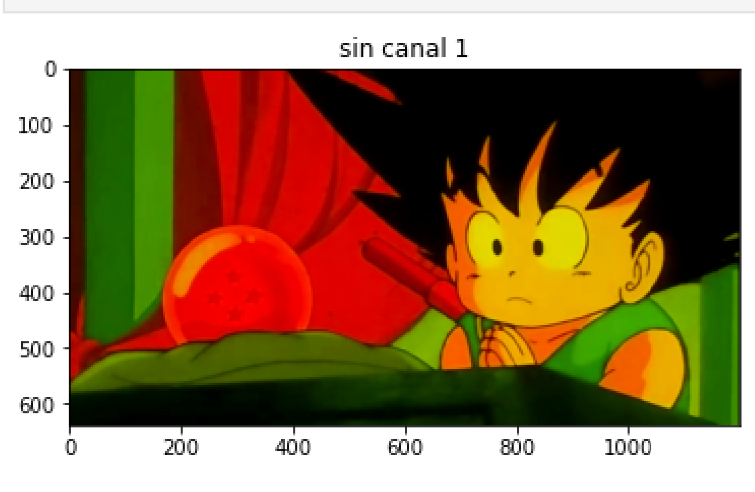
In [93]:

```python
# probemos de filtrar el rojo
zeros = np.zeros ((638,1200 ))
imgnotR= np.dstack ((col1, col2, zeros))
plt.imshow(imgnotR)
plt.title("sin canal 0")
plt.show()

mpimg.imsave("imgnotR.png", imgnotR)
```


sin canal 0

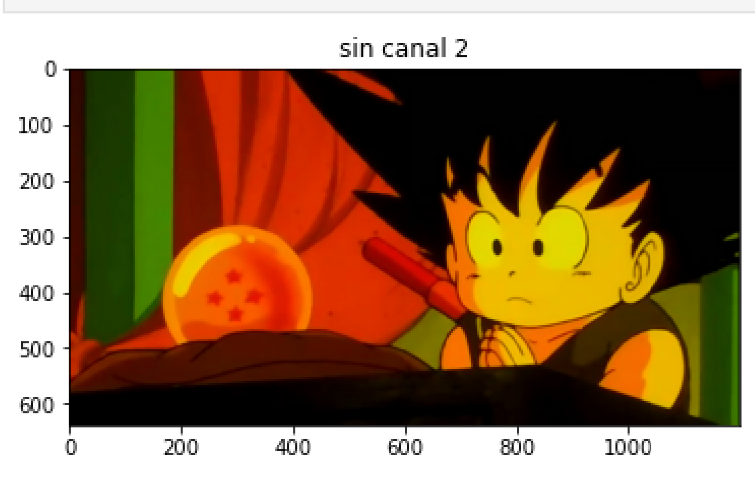
In [94]:

```python
# probemos de filtrar el verde
zeros = np.zeros ((638,1200 ))
imgnotG= np.dstack ((col0, col2, zeros))
plt.imshow(imgnotG)
plt.title("sin canal 1")
plt.show()
mpimg.imsave("imgnotG.png", imgnotG)
```


sin canal 1

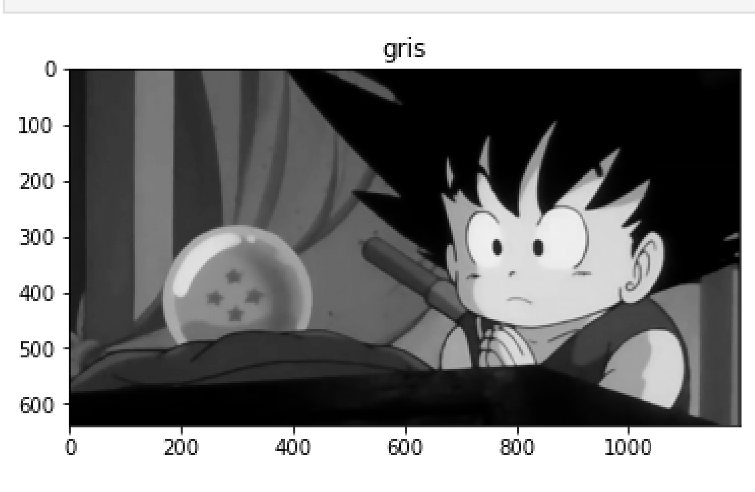
In [95]:

```python
# probemos de filtrar el azul
zeros = np.zeros ((638,1200 ))
imgnotB= np.dstack ((col0, col1, zeros))
plt.imshow(imgnotB)
plt.title("sin canal 2")
plt.show()
mpimg.imsave("imgnotB.png", imgnotB)
```


sin canal 2

In [91]:

```python
# vamos a pasarlo a grises.
R,G,B = col0,col1,col2
imag_grey = 0.2989 * R + 0.5870 * G + 0.1140 * B # donde el vector (0.2989, 0.5870, 0.114) es el color gris.
plt.imshow(imag_grey, cmap = "gray") # ponemos un mapa de colores de grises
plt.title("gris")
plt.show()
```


gris