

▼ Procesamiento de imágenes #02

Lectura, visualización y escritura de imágenes en Python

Dra. C. Miriela Escobedo Nicot

▼ Preparando el entorno de trabajo

Software requerido

- Python 3

Con las siguientes bibliotecas:

- pylab
- numpy
- scipy
- matplotlib
- scikit-image

Para facilitar el proceso de instalación se recomienda instalar [Anaconda](#) para **Python 3**, que es una distribución de Python que incluye todas las bibliotecas necesarias para trabajar en este curso. Se recomienda utilizar **Jupyter Notebook** para la codificación para mayor comodidad.

En todo código del curso se utilizará inicialmente la sentencia:

```
from pylab import *
```

Para importar las funciones iniciales necesarias para trabajar con imágenes.

▼ Lectura de imágenes

Las imágenes son leídas utilizando la función `imread` a la cuál se le pasa como parámetro la ruta donde se encuentra la imagen y nos devuelve la matriz correspondiente, la cuál podemos almacenar en una variable para posteriormente realizar el procesamiento y/o visualización.

```
A = imread('dataset/flores.tif')
```

En la línea anterior hemos cargado la imagen que se encuentra dentro de la carpeta `dataset` con el nombre `flores.tif` hemos almacenado su matriz correspondiente en la variable `A`.

Podemos imprimir el contenido de `A` y notar que la variable efectivamente contiene la matriz correspondiente a la imagen que hemos cargado.

A



```

array([[ 57,  63,  27],
       [ 64,  65,  31],
       [ 66,  68,  31],
       ...,
       [104, 123,  67],
       [103, 123,  64],
       [101, 119,  67]],

       [[ 63,  65,  41],
        [ 68,  64,  35],
        [ 76,  70,  38],
        ...,
        [103, 120,  65],
        [106, 121,  64],
        [107, 121,  68]],

       [[ 60,  65,  33],
        [ 66,  72,  38],
        [ 58,  63,  31],
        ...,
        [103, 121,  71],
        [108, 122,  71],
        [111, 125,  72]],

       ...,

       [[149, 121, 231],
        [148, 120, 231],
        [152, 126, 236],
        ...,
        [ 57,  68,  38],
        [ 60,  68,  47],
        [ 61,  63,  50]],


       [[145, 115, 227],
        [149, 121, 232],
        [148, 120, 231],
        ...,
        [ 53,  67,  41],
        [ 57,  69,  49],
        [ 62,  69,  53]],

       [[150, 121, 231],
        [149, 121, 231],
        [147, 121, 231],
        ...,
        [ 58,  69,  53],
        [ 61,  71,  60],
        [ 59,  70,  56]]], dtype=uint8)

```

Para conocer con que tipo de imagen estamos tratando contamos con las propiedades `shape` y `dtype`, para conocer las dimensiones de la imagen y el tipo de valores numéricos que almacena:

A.shape

 (600, 600, 3)

En este caso estamos tratando con una imagen a color (porque la última dimensión es 3 correspondiente a los valores de RGB), y que tiene una resolución de 600 x 600 píxeles.

A.dtype

```
dtype('uint8')
```

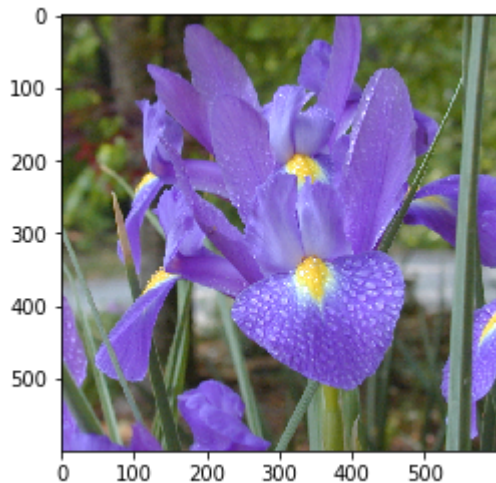
Y podemos ver que el tipo de dato utilizado son enteros sin signo de 8 bits

▼ Visualización de imágenes

Las imágenes se muestran utilizando la función `imshow`, a la cual pasamos como parámetro la variable que contiene la matriz correspondiente a la imagen que queremos visualizar.

```
imshow(A)
```

```
<matplotlib.image.AxesImage at 0xb335f120f0>
```



▼ Extracción de componentes de una imagen en color

Recordemos que la variable `A` almacena la matriz de la imagen en color, podemos extraer cada componente RGB como una matriz independiente utilizando los índices de la matriz. Debemos tener en cuenta que en Python los índices de los arreglos y matrices van de 0 a n , tomando como n el tamaño.

```
R = A[:, :, 0] # Extraer la primera componente Red
G = A[:, :, 1] # Extraer la segunda componente Green
B = A[:, :, 2] # Extraer la tercera componente Blue
```

Y podemos visualizar cada componente de manera independiente utilizando `imshow`, en este caso cada matriz extraída por cada componente tiene una dimensión de (600, 600, 1) por lo que para visualizarla correctamente es preferible utilizar un mapa de color en escala de grises, esto se hace configurando el parámetro `cmap` de la función `imshow` y otorgándole el valor `"gray"`

```
R.shape # Mostrar dimensión de de las componentes extraídas (600, 600)
```



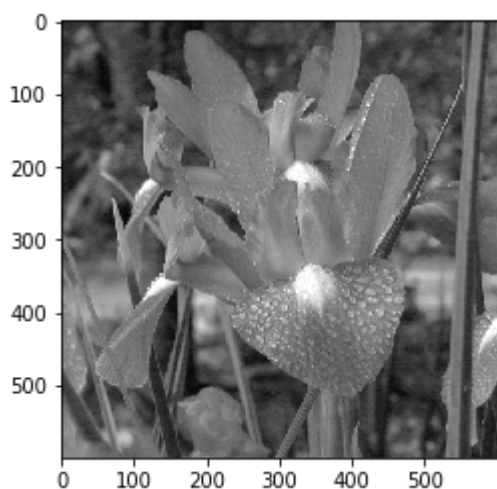
```
(600, 600)\nR # Mostrar que la componente también es una matriz
```

```
array([[ 57,  64,  66, ..., 104, 103, 101],\n       [ 63,  68,  76, ..., 103, 106, 107],\n       [ 60,  66,  58, ..., 103, 108, 111],\n       ...,\n       [149, 148, 152, ...,  57,  60,  61],\n       [145, 149, 148, ...,  53,  57,  62],\n       [150, 149, 147, ...,  58,  61,  59]], dtype=uint8)
```

```
## Visualizar componentes en escala de grises
```

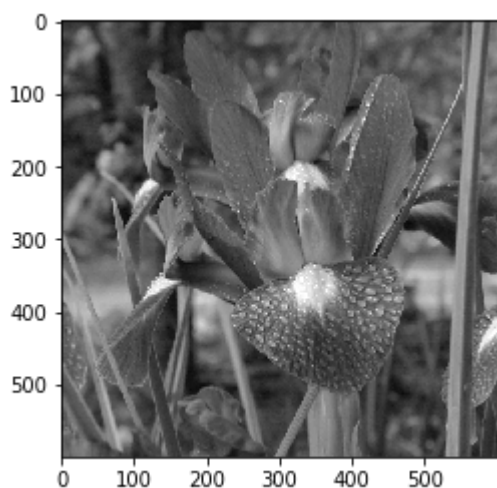
```
imshow(R, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0xb335f4c320>
```



```
imshow(G, cmap="gray")
```

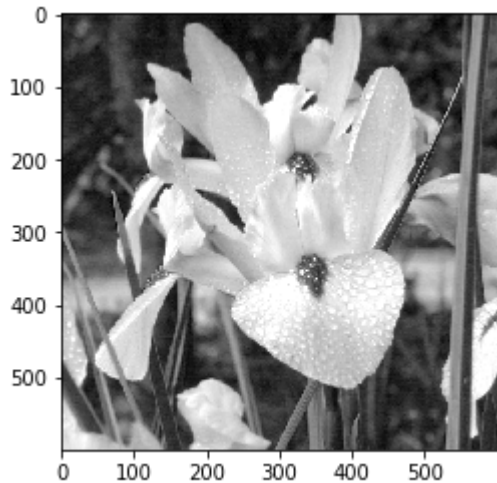
```
<matplotlib.image.AxesImage at 0xb335ffb588>
```



```
imshow(B, cmap="gray")
```



```
<matplotlib.image.AxesImage at 0xb338871cc0>
```

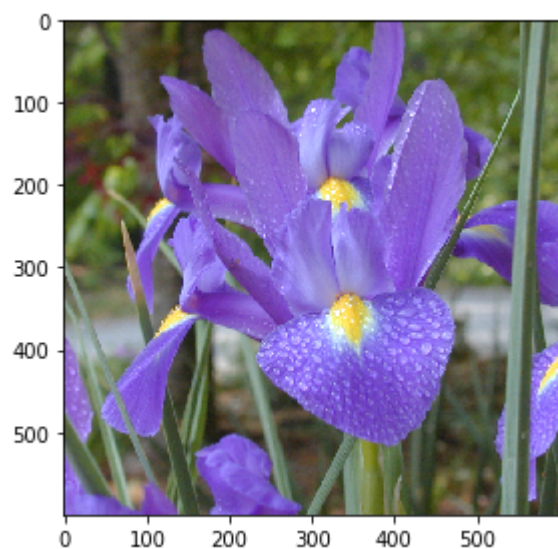
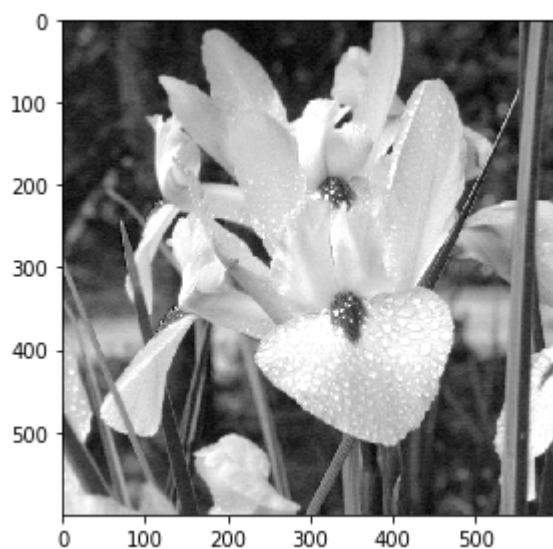
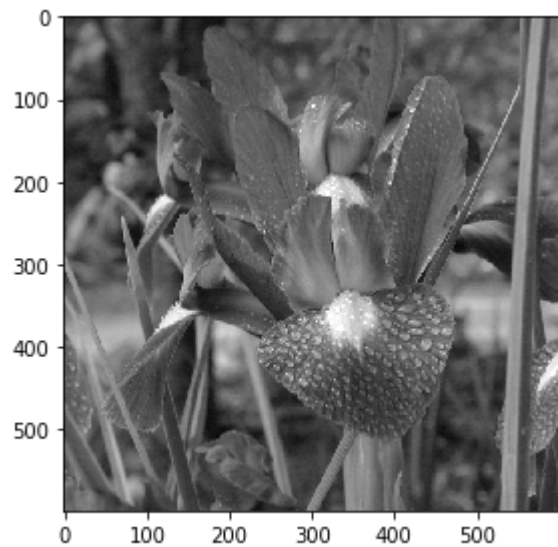
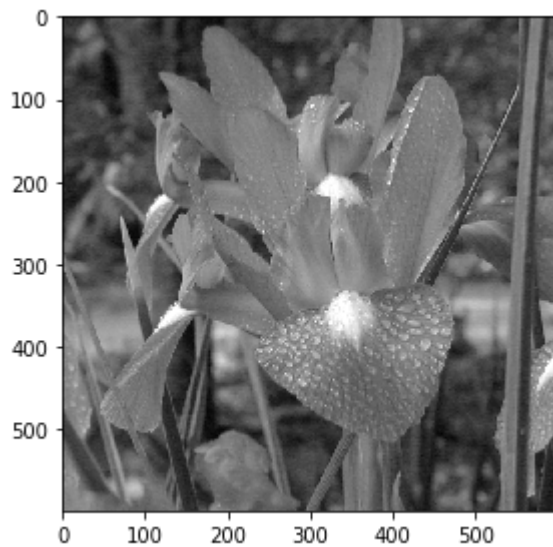


En ocasiones resulta más cómodo poder visualizar varias imágenes a la vez. Para esto podemos utilizar la función `subplots` a la cual pasamos como parámetros la cantidad de filas y columnas que utilizaremos para organizar nuestras imágenes, y las dimensiones de cada una de ellas. Esta función nos devuelve dos parámetros, el primero de ellos lo descartaremos utilizando `_` y el segundo es una matriz donde cada elemento le corresponde a un lienzo que utilizaremos para llamar a la función `imshow` y mostrar la imagen que deseemos.

```
_, ((ax0, ax1), (ax2, ax3)) = subplots(2, 2, figsize=(10, 10))
ax0.imshow(R, cmap="gray")
ax1.imshow(G, cmap="gray")
ax2.imshow(B, cmap="gray")
ax3.imshow(A)
```



<matplotlib.image.AxesImage at 0xb3389a9dd8>



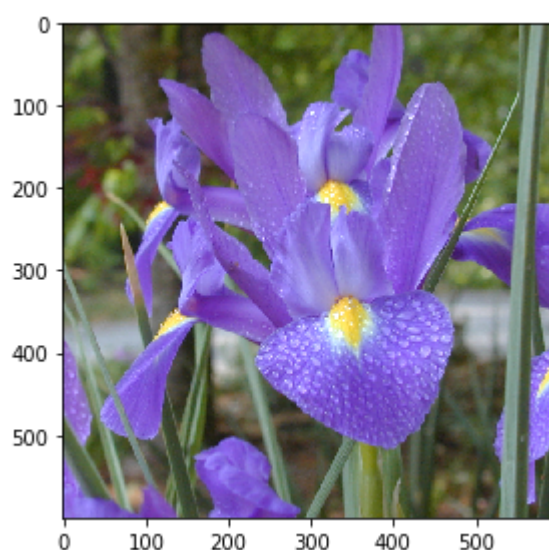
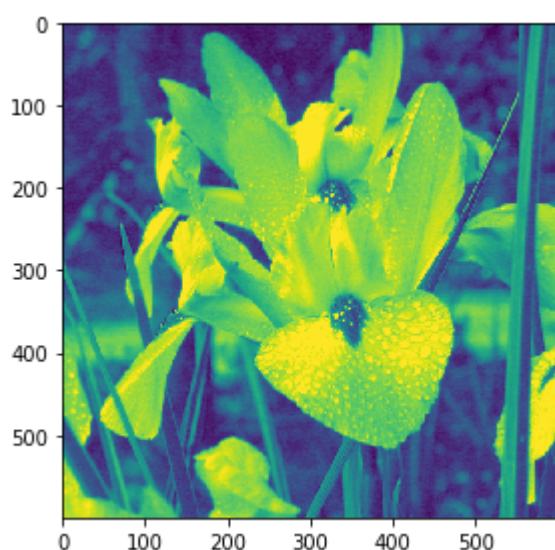
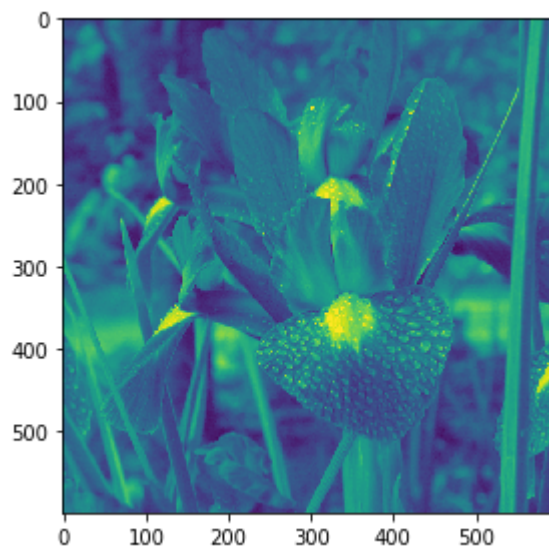
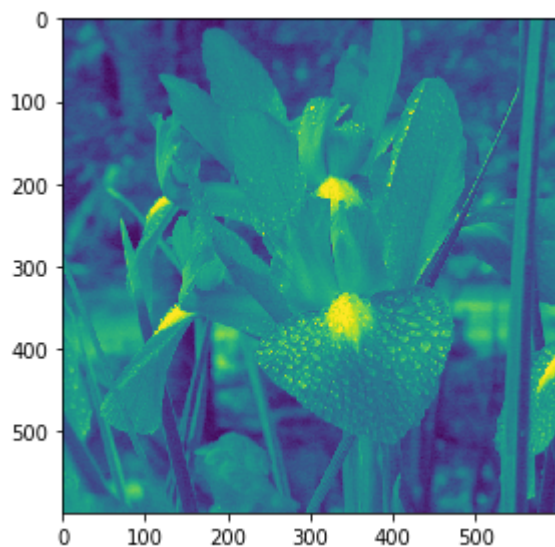
En el siguiente ejemplo vemos como se muestran las componentes RGB sin utilizar el mapa de color a escala de grises.

```
_, ((ax0, ax1), (ax2, ax3)) = subplots(2, 2, figsize=(10, 10))
```

```
ax0.imshow(R)
ax1.imshow(G)
ax2.imshow(B)
ax3.imshow(A)
```




```
<matplotlib.image.AxesImage at 0xb338b74ba8>
```



Ahora supongamos que queremos observar cada imagen de cada componente RGB pero con su color correspondiente.

Intuitivamente lo que se nos ocurre es conservar los valores de la imagen en la componente que deseamos visualizar y el resto ponerlos a cero.

Comencemos por crear una matriz con las mismas dimensiones de A , que es donde almacenamos nuestra imagen original pero con todos sus valores a 0 esto se puede lograr facilmente utilizando la función `zeros_like` de la siguiente manera.

```
T = zeros_like(A)
```

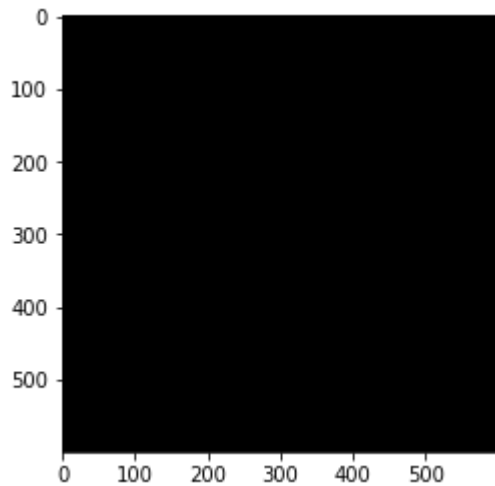
Ahora tenemos en T una matriz con las mismas dimensiones de A pero con todos sus valores a 0

```
T.shape
```

```
(600, 600, 3)
```

```
imshow(T) # Todos los valores están a cero
```

 <matplotlib.image.AxesImage at 0xb3391a5e48>



Solo nos resta copiar cada componente en su lugar correspondiente e ir almacenando el valor resultante para mostrarlo luego.

```
T[:, :, 0] = R # Asignamos el valor de la componente R en su lugar correspondiente
RR = T # Guardamos el resultado
```

Volvemos a crear otra matriz para el resto de las componentes y repetimos el procedimiento

```
T = zeros_like(A)
T[:, :, 1] = G
GG = T
```

```
T = zeros_like(A)
T[:, :, 2] = B
BB = T
```

Ahora tenemos tres matrices RR , GG , BB con los valores de cada componente de la imagen original A en su lugar correspondiente y podemos proceder a visualizar el resultado.

```
# Obtener imagen original a partir de las componentes por separado
```

```
original = zeros_like(A)
```

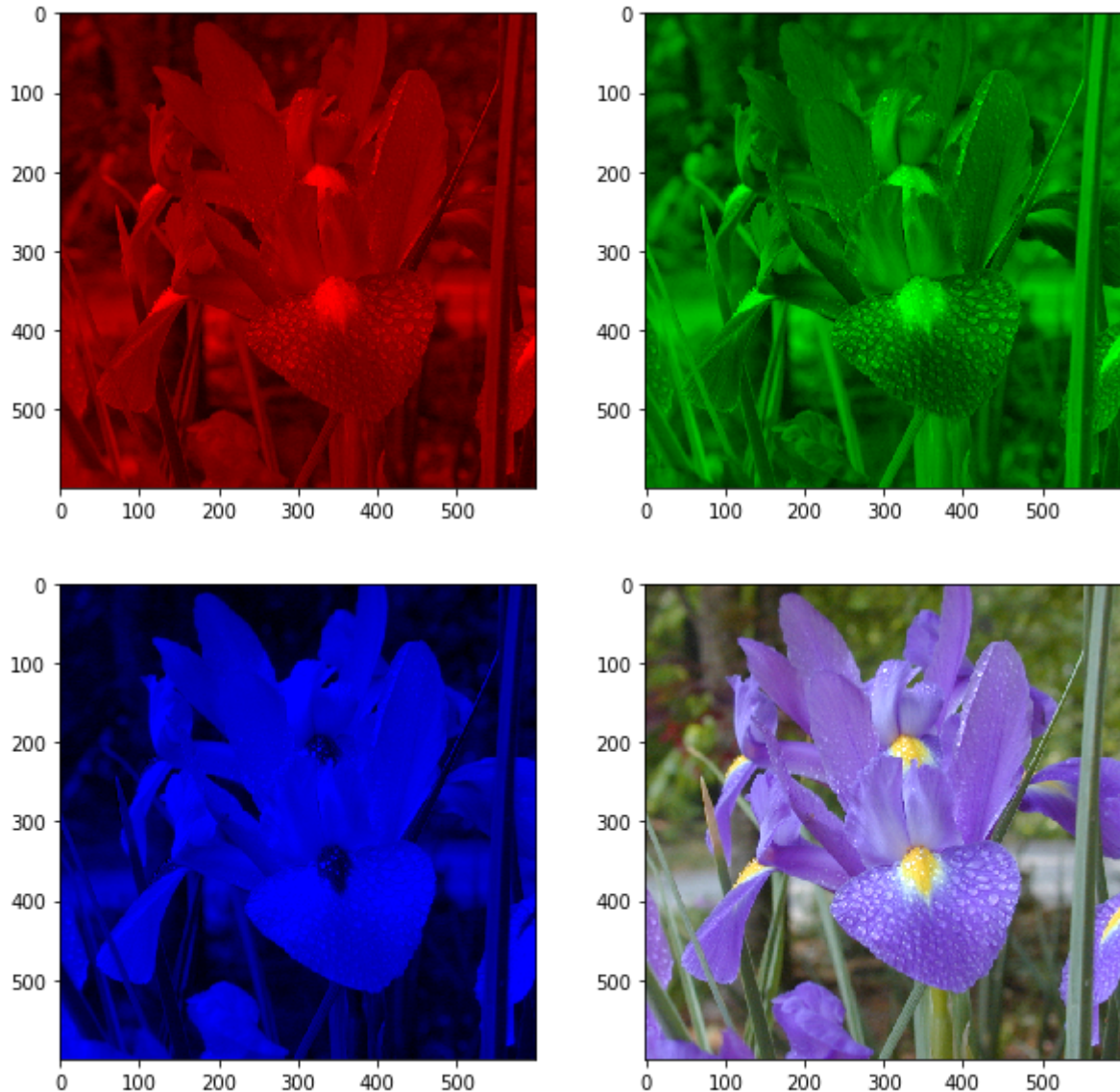
```
original[:, :, 0] = R
original[:, :, 1] = G
original[:, :, 2] = B
```

```
_, ((ax0, ax1), (ax2, ax3)) = subplots(2, 2, figsize=(10, 10))
```

```
ax0.imshow(RR)
ax1.imshow(GG)
ax2.imshow(BB)
ax3.imshow(original)
```




```
<matplotlib.image.AxesImage at 0xb33910d9b0>
```



Y así obtenemos los tres canales de la imagen por separado y visualizados a color. Como podemos ver la imagen original es el resultado de la combinación de todas las componentes.

▼ Convertir una imagen a color en escala de grises (rgb2gray)

En este caso tendremos que importar la función `rgb2gray` que tenemos disponible en la biblioteca `skimage`

```
from skimage.color import rgb2gray
```


Y llamar a la función pasándole como parámetro la matriz de nuestra imagen y almacenar el valor en una nueva variable.

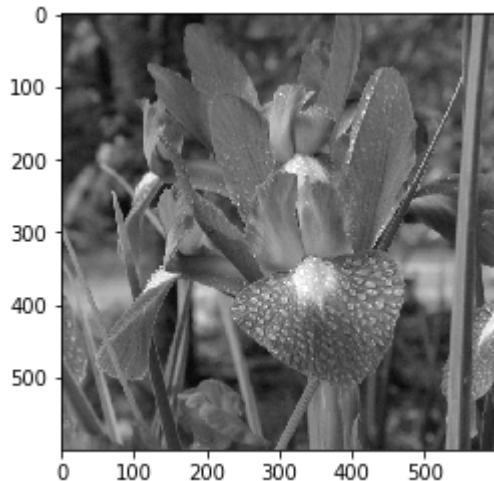
```
AG = rgb2gray(A)
```

```
AG.shape # Se observa que ahora tiene un solo canal
```

```
(600, 600)
```

```
imshow(AG, cmap="gray") # Visualización de la imagen
```

 <matplotlib.image.AxesImage at 0xb33e5d27b8>



▼ Escritura de imágenes

Una vez tenemos nuestra imagen como resultado de alguna operación que hemos realizado sobre la imagen original, nos puede interesar guardarla en disco. Para ello utilizamos la función `imwrite`, a la cuál pasamos como parámetros la matriz de la imagen que deseamos guardar y la ruta para el fichero, es importante colocar en el nombre del fichero la extensión que representa el formato en el que queremos guardar nuestra imagen.

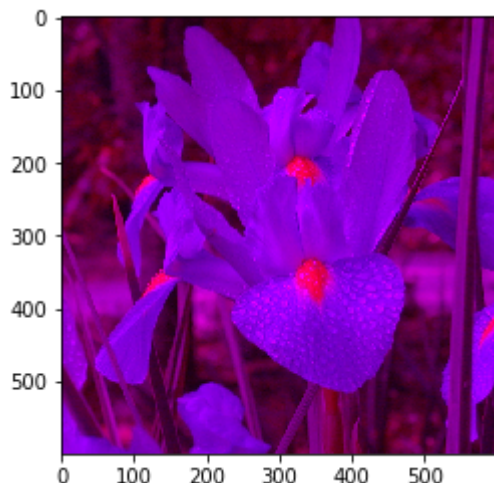
Supongamos que hemos realizado la siguiente operación de eliminar el canal G de nuestra imagen original y queremos guardar el resultado en una imagen con formato JPG.

```
T = zeros_like(A)
```

```
T[:, :, 0] = R
T[:, :, 2] = B
```

```
imshow(T)
```

 <matplotlib.image.AxesImage at 0xb3404bb208>



```
### Guardar el resultado
```

```
imsave("dataset/outputs/flores_sin_g.jpg", T)
```

Para guardar imágenes en escala de grises solo hay que especificar al igual que con imshow el mapa de color a utilizar.

```
### Guardar imagen en escala de grises  
imsave("dataset/outputs/flores_gray.png", AG, cmap="gray")
```

En este caso hemos utilizado el formato PNG