Ejercicio 1

Busca en internet la imagen de una bandera en la que aparezcan, al menos, tres colores distintos sobre un cielo mayormente azul y descárgala.

Antes de programar nada, describe el contenido que esperarías encontrar en cada uno de los canales de color R, G, B de dicha imagen si se visualizase cada canal por separado.

Escribe un programa que visualice la imagen en color y cada canal por separado. Comprueba si el contenido de cada canal coincide o no con el que habías previsto inicialmente.

```python
import skimage as ski
import matplotlib.pyplot as plt

imagen_en_color = ski.io.imread("images/banderaGuinea.jpg")

plano_rojo = imagen_en_color[:, :, 0]
plano_verde = imagen_en_color[:, :, 1]
plano_azul = imagen_en_color[:, :, 2]

fig, axs = plt.subplots( nrows: 2,  ncols: 3, layout="constrained")
axs[0, 1].imshow(imagen_en_color)
axs[1, 0].imshow(plano_rojo, cmap=plt.cm.gray)
axs[1, 1].imshow(plano_verde, cmap=plt.cm.gray)
axs[1, 2].imshow(plano_azul, cmap=plt.cm.gray)

for ax in axs.ravel():
    ax.set_axis_off()
plt.show()
```

En cuanto a los resultados esperados puedo decir lo siguiente:

- Canal Rojo (R): La franja de la bandera que es roja será más notoria. Es probable que el resto de la bandera aparezca más oscura, en el caso del cielo azul, o incluso negra -como creo que será el caso de la franja verde y la amarilla-, ya que no contiene mucha información en el canal rojo.
- Canal Verde (G): Los elementos que contienen verde serán más evidentes, como la franja verde. Los detalles en rojo, amarillo y azul tenderán a aparecer más oscuros, ya que estos tonos no están representados en el canal.

- Canal Azul (B): El fondo azul será la característica más destacada, mientras que el resto de elementos tenderán a aparecer más oscuros o casi negros.

La comparación entre resultados esperados y obtenidos es la siguiente:

- Canal Rojo (R): Esperaba que el elemento rojo de la bandera fuese el más visible, y aunque es muy visible, y al contrario de mis expectativas, el color amarillo destaca más.
- Canal Verde (G): En este caso, pasa igual que en el rojo, el amarillo destaca aun más que el verde. Sin embargo la imagen, en general, no es muy oscura.
- Canal Azul (B): Tal y como habíamos predicho el azul destaca más. Siendo el cielo muy claro.

Ejercicio 2

Considera las imágenes *banderaIrlanda.jpg* y *banderaItalia.jpg*. Escribe un programa que a partir de ambas imágenes visualice una nueva imagen como la que se muestra a continuación:

El programa debe centrar la imagen más pequeña dentro de la mayor, con independencia de las imágenes usadas y de su tamaño. Es decir, para un par de imágenes cualesquiera, el programa debe mostrar una nueva imagen con la imagen pequeña en vertical y centrada dentro de la mayor.

NOTA AÑADIDA DESPUÉS DE LA CLASE. Puedes asumir que la imagen pequeña en vertical siempre cabrá dentro de la imagen grande. No es necesario que lo verifiques. Si ya lo has hecho, déjalo. Mejor así.

```python
import skimage as ski
import matplotlib.pyplot as plt
import numpy as np

grande = ski.io.imread("images/banderaItalia.jpg")
pequenya = ski.io.imread("images/banderaIrlanda.jpg")

if pequenya.size > grande.size:
    pequenya, grande = grande, pequenya

pequenya = np.rot90(pequenya)

inicio_x = (grande.shape[0] - pequenya.shape[0]) // 2
inicio_y = (grande.shape[1] - pequenya.shape[1]) // 2

nueva = grande.copy()
nueva[inicio_x:inicio_x + pequenya.shape[0], inicio_y:inicio_y + pequenya.shape[1], :] = pequenya

plt.imshow(nueva)
plt.axis('off')
plt.show()
```

En este caso, el código difiere más del original que en el anterior ejercicio.

```
if pequenya.size > grande.size:

    pequenya, grande = grande, pequenya
```

Con esto nos aseguramos de que la imagen de fondo que no se altera, sea siempre la más grande, que en este será la bandera de Italia. Para esto usamos *size*.

```
pequenya = np.rot90(pequenya)
```

Esta línea sirve para rotar la imagen 90º en sentido contrario a las agujas del reloj, que es lo que hace de forma predeterminada.

```
inicio_x = (grande.shape[0] - pequenya.shape[0]) // 2

inicio_y = (grande.shape[1] - pequenya.shape[1]) // 2
```

Para poder centrar la imagen pequeña dentro de la grande, hemos calculado las coordenadas de inicio en la imagen grande para superponer la pequeña. Pero esto lo hemos hecho después de rotar la imagen, porque en caso contrario no se centraría bien.

Esto lo hace de modo que calcula la coordenada de inicio en el eje x/y de la grande y le resta la altura/anchura de la pequeña de la grande y divide entre 2 eso para centrarlo vertical o horizontalmente.

nueva[inicio_x:inicio_x + pequenya.shape[0], inicio_y:inicio_y + pequenya.shape[1], :] = pequenya

Por último, con esto ubicamos la pequeña en el centro de la grande.

En general, la solución demuestra un proceso eficiente y claro para superponer imágenes de manera centrada y mostrar el resultado. Algo a mencionar de los resultados, es que mientras la imagen de Irlanda ocupa un espacio de 229,61 kB y la de Italia 37,28 kB, la nueva imagen generada ocupa 146 kB.

Ejercicio 3

Considera la imagen *mapas.png*. A partir de ella, genera imágenes *jpg* con calidades del 100%, 75% y 15%. Calcula los errores cometidos para cada banda de cada imagen al codificar las imágenes en *jpg*.

NOTA. En los programas de ejemplo tienes funciones para calcular el máximo error absoluto, el error medio y el error cuadrático medio.

```python
import skimage as ski
import matplotlib.pyplot as plt
import numpy as np

imagenOriginal = ski.io.imread("images/mapas.png")
ski.io.imsave( fname: "mapas15.jpg", imagenOriginal, quality=15)
imag15 = ski.io.imread("mapas15.jpg")
ski.io.imsave( fname: "mapas75.jpg", imagenOriginal, quality=75)
imag75 = ski.io.imread("mapas75.jpg")
ski.io.imsave( fname: "mapas100.jpg", imagenOriginal, quality=100)
imag100 = ski.io.imread("mapas100.jpg")

imagenOriginal = ski.util.img_as_float(imagenOriginal)  # Valores flotantes en el rango [0,1]
imagen15 = ski.util.img_as_float(imag15)
imagen75 = ski.util.img_as_float(imag75)
imagen100 = ski.util.img_as_float(imag100)


1 usage
def errorCuadráticoMedioBanda(m1, m2):  # Las bandas dadas deben ser flotantes
    mo1 = m1 * 255  # Queremos medir el error en el rango [0,255]
    mo2 = m2 * 255
    return np.sum(np.power(mo2 - mo1, 2),  axis: None) / m1.size


1 usage
def errorMedioBanda(m1, m2):  # Las bandas dadas deben ser flotantes
    mo1 = m1 * 255  # Queremos medir el error en el rango [0,255]
    mo2 = m2 * 255
    return np.sum(abs(mo2 - mo1),  axis: None) / m1.size
```

```python
def maximoErrorAbsolutoBanda(m1, m2):  # Las bandas dadas deben ser flotantes
    mo1 = m1 * 255  # Queremos medir el error en el rango [0,255]
    mo2 = m2 * 255
    return np.max(np.abs(mo2 - mo1))


bandas = ["Roja", "Verde", "Azul"]
imagenes = [imagen15, imagen75, imagen100]
nombres = ['de 15', 'de 75', 'de 100']

fig, axs = plt.subplots( nrows: 4,  ncols: 3, layout="constrained")
axs[0][0].imshow(imagenOriginal)

for i in range(len(imagenes)):
    print(f"---Imagen {nombres[i]} ---")
    for nBanda, nombre in enumerate(bandas):
        print(f"Banda {nombre}")
        print(f"   Máximo error: {maximoErrorAbsolutoBanda(imagenOriginal[:, :, nBanda], imagenes[i][:, :, nBanda])}")
        print(f"   Error medio: {errorMedioBanda(imagenOriginal[:, :, nBanda], imagenes[i][:, :, nBanda])}")
        print(
            f"   Error cuadrático medio: {errorCuadráticoMedioBanda(imagenOriginal[:, :, nBanda], imagenes[i][:, :, nBanda])}")

    errores = []
    for nBanda in range(3):
        errorBanda = abs(imagenes[i][:, :, nBanda] - imagenOriginal[:, :, nBanda])
        errores.append(errorBanda)
    errorGlobal = np.stack(errores, axis=-1)

    axs[i+1][0].imshow(imagenes[i])
    axs[i+1][1].imshow(errorGlobal/errorGlobal.max())  # Piensa cómo mejorar la visualización del error

for ax in axs.ravel():
    ax.set_axis_off()
plt.show()
```

Primero vamos a comentar el código. Estas sentencias se repiten 3 veces, y lo que hacen es guardar una imagen con el nombre que le demos, a partir de la imagen original que ya tenemos, y como estamos convirtiendo de png a jpg, también podemos definir la calidad de la imagen que en este caso será 15, 75 o 100.

```
ski.io.imsave("mapas15.jpg", imagenOriginal, quality=15)

imag15 = ski.io.imread("mapas15.jpg")
```

La primera sentencia convierte la imagen a tipo float64, donde los valores de los píxeles están en el rango [0, 1]. Esta misma conversión se realiza en el resto de imágenes de igual forma.

```
imagenOriginal = ski.util.img_as_float(imagenOriginal)
```

Con este bloque, se crea una ‘fig’ de matplotlib con una matriz de dimensiones 4x3. Y luego muestra la imagen original en la posición [0, 0].

```
fig, axs = plt.subplots(4, 3, layout="constrained")

axs[0][0].imshow(imagenOriginal)
```

El bucle principalmente lo que pretende hacer es lo mismo que en el código original pero con todas las imágenes. Para mejorar el error, hemos hecho que el error global de la imagen se divida entre el error máximo posible entre todas las imágenes. De modo, que si del 1 al 200, el máximo error se encuentre entre 1 y 20, no se vea representado de forma ínfima, y por tanto, se diferencian muy poco, por esto, hemos hecho que si el máximo es 20, ese sea el 100, y así se aprecie mejor.

```
axs[i+1][1].imshow(errorGlobal/errorGlobal.max())
```

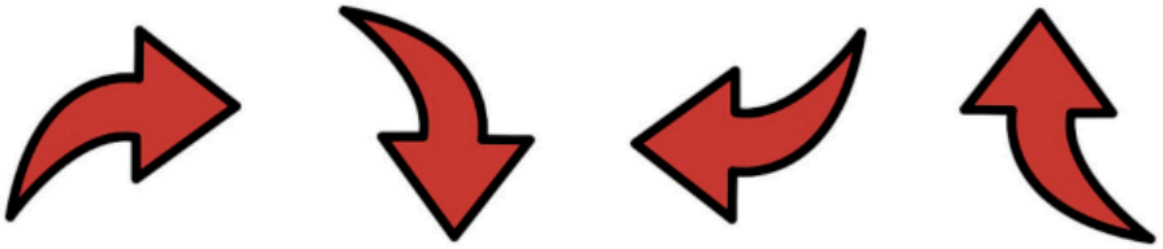
Lo que podemos decir de las soluciones es que a simple vista el mayor error que hay se da en la imagen de menor calidad. Así mismo, vamos a comentar las soluciones usando la siguiente tabla:

		Máximo error	Error medio	Error cuadrático medio
Imagen de calidad 15	Banda Roja	58.0	5.632	51.608
	Banda Verde	42.0	3.439	19.309
	Banda Azul	60.0	5.383	48.664
Imagen de calidad 75	Banda Roja	46.0	3.673	22.709
	Banda Verde	22.0	2.219	8.177
	Banda Azul	50.0	4.232	29.817
Imagen de calidad 100	Banda Roja	28.0	1.784	5.832
	Banda Verde	14.0	0.931	1.663
	Banda Azul	39.0	2.263	9.184

Viendo esta tabla podemos mencionar el hecho de que el mayor error se encuentra en la imagen de calidad 15 y dentro de esta en la banda azul, seguido de la roja y por último la verde. A su vez, el menor error se encuentra en la imagen de calidad 100, que aunque tendamos a pensar que el error sería nulo, no es así, pero es muy pequeño.

Ejercicio 4

A partir de la imagen *flecha_transparente.png* genera cuatro imágenes como las que se muestran a continuación. Puedes usar el método para transponer una matriz así como operaciones para invertir sus filas o columnas:



A partir de estas cuatro imágenes, genera un *gif* animado y visualízalo con un navegador.

El hecho de que la imagen inicial sea transparente hace que el *gif* animado no resulte adecuado. Por lo tanto, añade a tu programa una línea de código para eliminar el canal de transparencia de la imagen original. Ajusta los parámetros del *gif* para que parezca que la flecha gira sobre su centro de manera indefinida.

```python
import skimage as ski
import numpy as np
import matplotlib.pyplot as plt

flecha = ski.io.imread("images/flecha_transparente.png")[:, :, 0:3]
# pille le 3


1 usage
def rotar(flecha, nveces=3):
    rotadas = [flecha]
    imagen = flecha  # Esto se hace así para que sobre la imagen ori
    # que el anterior

    for i in range(nveces):
        imagen = np.rot90(imagen, k=-1)
        rotadas.append(imagen)

    return rotadas


secuencia = np.stack(rotar(flecha), axis=0)

ski.io.imsave( fname: "images/flecha.gif", secuencia, loop=0, fps=4)
```

Al comentar el código podemos decir que esta línea además de leer la imagen, se encarga de que no pille el fondo transparente de la imagen para así poder evitar posibles errores. Si con *[:, :, 3]* pilla solo el transparente y es el último valor, hacemos qué pille todos menos ese.

```
flecha = ski.io.imread("images/flecha_transparente.png")[:, :, 0:3]
```

Luego se crea una función, que rotará la imagen 3 veces más esas imágenes las añadirá a una lista y devolverá al llamar a la función. Y con esta línea creará una secuencia con estas imágenes rotadas.

```
secuencia = np.stack(rotar(flecha), axis=0)
```

Y con esta última línea lo que se hará es guardar la secuencia de imágenes como un archivo gif, con *loop=0* de modo que el gif no parará de reproducirse y con *fps=4* le decimos que mostrará las 4 imágenes en un mismo segundo.

```
ski.io.imsave("images/flecha.gif", secuencia, loop=0, fps=4)
```