Ejercicio 1

Considera una imagen en niveles de gris, por ejemplo, *lena256.pgm*. Utiliza la función *rotate* para rotar la imagen *N* grados: *rotate(img, N)*. Si lees detenidamente la documentación de dicha función, podrás comprobar que el ángulo se mide en grados y el giro se realiza en sentido antihorario desde el centro de la imagen.

Queremos obtener una transformación euclídea que realice exactamente la misma transformación. Para ello, deberás generar una matriz de transformación que combine los siguientes pasos:

1. Trasladar el centro de la imagen hasta la posición (0, 0).
2. Rotar la imagen el ángulo deseado.
3. Trasladar el centro de la imagen hasta su posición original.

Una vez que hayas obtenido la transformación deseada, aplícala a la imagen utilizando la función *warp*. Si todo ha ido bien, deberías obtener exactamente el mismo resultado que en el caso anterior

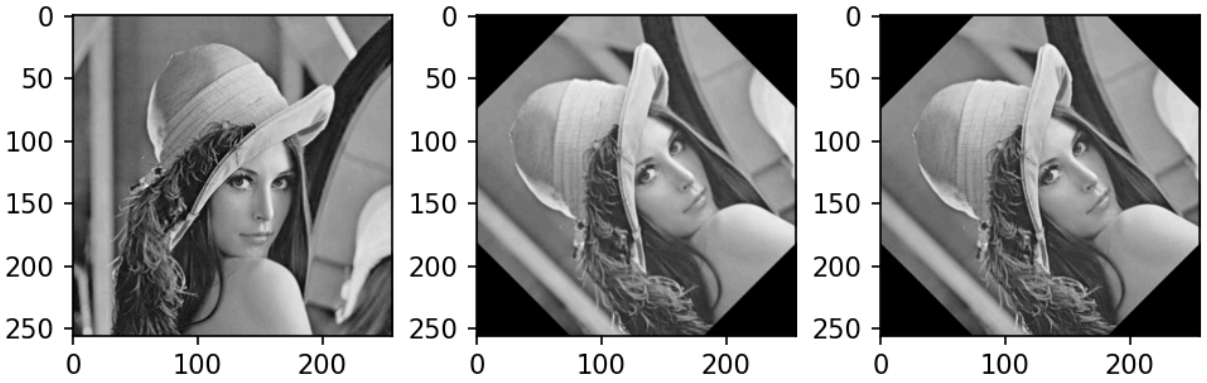
Al crear la matriz de transformación, deberás tener en cuenta que, aunque la función *rotate* considera el giro en sentido antihorario, las matrices de rotación lo consideran en sentido horario. Además, la función *rotate* mide el ángulo en grados y para las matrices tendrás que expresarlo en radianes.

```
# Definir el ángulo de rotación en grados
angle = 45

# Convertir el ángulo de grados a radianes
angle_rad = np.radians(360 - angle)

translation = np.array([img_original.shape[0] / 2, img_original.shape[1] / 2])
```

Lo más destacable de este código es la conversión del ángulo de grados a radianes, pues al tener en cuenta que con la función rotate el giro se realiza en sentido antihorario, tenemos que calcularlo como 360 - angle.

También podemos mencionar que hemos usado translation para encontrar las coordenadas de origen tanto en el eje x como en el y.

Como podemos apreciar en los resultados que se encuentran abajo, obtenemos la misma imagen de cualquiera de las dos formas que la hemos calculado.



Ejercicio 2

Considera la misma imagen del ejercicio anterior. A partir de ella, genera y visualiza las siguientes transformaciones afines (debes mostrar, en cada caso, la matriz de transformación utilizada y el resultado de aplicarla sobre la imagen que se indica):

1. Sobre la imagen original, un escalado de 0.5 en X y 0.75 en Y.
2. Sobre el resultado anterior, un desplazamiento de 64 píxeles en X y 0 en Y.
3. Sobre el resultado anterior, una rotación de 15 grados.
4. Sobre el resultado anterior, una inclinación de -10 grados tanto en X como en Y.
5. A partir de la clase AffineTransform, crea una transformación que realice simultáneamente las cuatro transformaciones anteriores. Aplícala sobre la imagen original. ¿Obtienes el mismo resultado que en el punto 4? Justifica tu respuesta.
6. A partir de las matrices de transformación que has empleado en los cuatro primeros puntos, crea tu propia matriz de transformación multiplicando las anteriores en el orden adecuado. Aplica la transformación que define dicha matriz a la imagen original. ¿Obtienes el mismo resultado del punto 4, del punto 5 o ninguno de ellos? Razona tu respuesta.

La forma en la que he calculado las primeras 4 imágenes ha sido a partir de la que he encontrado en los ejemplos del tema. Sin embargo, a partir de la 5, el código se vuelve más interesante.

```
ta_simul = ski.transform.AffineTransform(scale=(0.5, 0.75), translation=DISTANCIA_TRASLACION,
                                         rotation=np.radians(ANGULO_ROTACION_EN_GRADOS),
                                         shear=ANGULO_INCLINACION_EN_GRADOS)
img_simultanea = ski.transform.warp(img_original.copy(), ta_simul.inverse)
```

En este tramado de código, podemos observar que hemos llevado a cabo las 4 primeras transformaciones de manera simultánea, mediante la función AffineTransform. Pero tal y como podemos apreciar en los resultados, la imagen que se ha obtenido no es exactamente igual a la anterior. La razón es que aunque los parámetros sean los mismos en ambas transformaciones, se generan imágenes diferentes debido a la secuencia distinta en la aplicación de las transformaciones. El orden que sabemos que se sigue es el siguiente: escalar, inclinación, rotación y traslación.

```
matriz_total = mshear @ mrotacion @ mtranslacion @ mscalado
ta_propia = ski.transform.AffineTransform(matriz_total)
img_matriz = ski.transform.warp(img_original.copy(), ta_propia.inverse)
```

Para calcular la 6º imagen, se ha calculado una matriz de transformación total combinando las matrices individuales de inclinación, rotación, traslación y escala. Luego, se usa esta matriz total para crear una transformación afín personalizada. Finalmente, se aplica esta transformación a una copia de la imagen original.

La imagen en este caso es bastante similar a la 4, pero tampoco es la misma, como ya hemos explicado previamente. Inicialmente, el resultado de la 5º y la 6º son prácticamente iguales debido a que se realiza lo mismo pero de diferente forma, sin que esto afecte al resultado final.

Ejercicio 3

Considera las imágenes de Einstein, de Marilyn o alguna otra imagen tuya. Define sobre una de ellas una malla inicial y una modificación de la misma para deformar alguna característica de la imagen. Usa tu imaginación para crear una deformación divertida: ojos con rasgos orientales, agrandar la nariz, alargar el mentón o ¡todo a la vez!

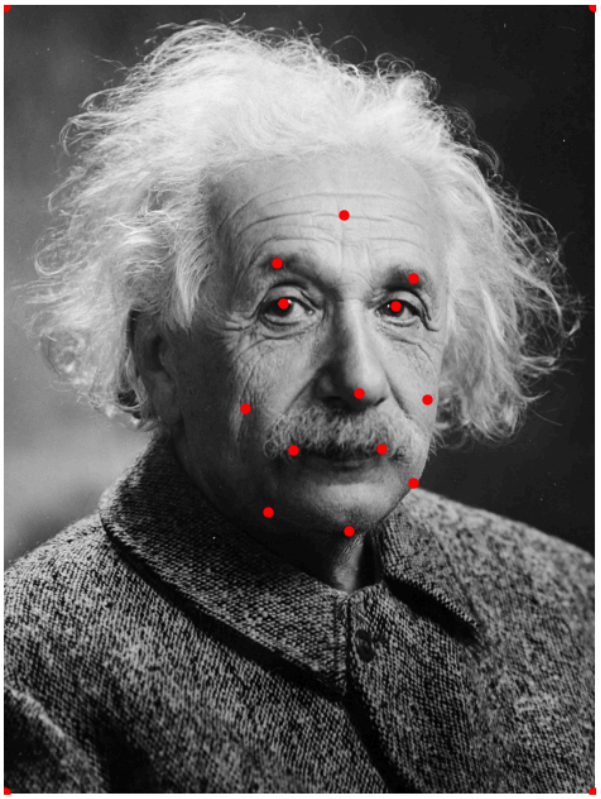
Cuando tengas definidas las dos mallas, crea un objeto de la clase *PiecewiseAffineTransform*, estima los parámetros para realizar la transformación a trozos desde la malla original hasta la malla transformada y aplica dicha transformación a la imagen mediante la función *warp*.

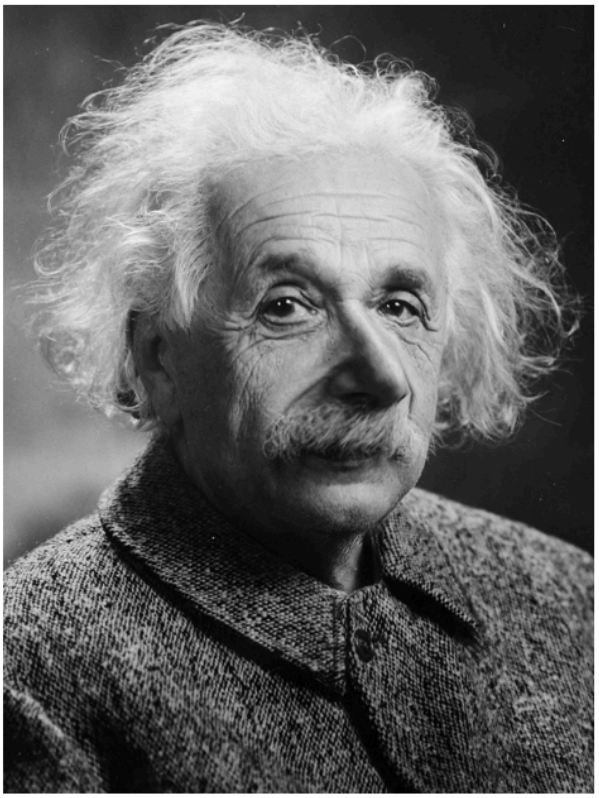
```
src = [[0, 0],
       [0, 1066],
       [377, 403],  # Ojo izquierdo
       [530, 406],  # Ojo derecho
       [480, 524],  # nariz
       [572, 533],  # moflete derecho
       [326, 545],  # moflete izquierdo
       [369, 348],  # ceja izqda
       [554, 369],  # ceja dcha
       [391, 601],  # labios izqda
       [511, 600],  # labios dcha
       [466, 711],  # barbilla
       [459, 283],  # frente mitad
       [357, 685],  # barbilla izqda
       [554, 646],  # barbilla dcha
       [799, 0],
       [799, 1066]]
```

Con esto hemos calculado la posición de las diferentes partes de la cara.

```
dst[4] = [531, 516]  # Nariz alargada
```

Aquí lo que hemos hecho ha sido calcular las nuevas coordenadas de la nariz de modo que se vea alargada.





Estudiante: Irene Mejía Hurtado
Profesor: Pedro García Sevilla