# Práctica 1: Manejo Básico de Imágenes

Reconocimiento de Patrones - 0757

*Facultad de Ingeniería*
*Universidad Nacional Autónoma de México*

Ceballos Equihua C. N.
*Ingeniería en Computación*
*Facultad de Ingeniería, UNAM*
Ciudad de México, México
ceballos.equihua@gmail.com

Murrieta Villegas A.
*Ingeniería en Computación*
*Facultad de Ingeniería, UNAM*
Ciudad de México, México
alfonsomvmx@gmail.com

**Profesores:**
Dr. Boris Escalante Ramírez
Dra. Olveres Montiel Jimena
I.I.M.A.S. - UNAM

*Resumen*— **Esta práctica consiste en realizar manipulaciones básicas a imágenes utilizando bibliotecas de Python, como son Matplotlib, OpenCV, Scikit-Image, PIL y SciPy. Entre estas manipulaciones, se encuentran desplegar, cambiar el espacio de color, redimensionar, rotar y recortar imágenes, además de imprimir su información.**

## I. Introducción

A lo largo de esta práctica, se leerán, desplegarán y manipularán imágenes utilizando scripts codificados en Python. Una imagen es una matriz, o arreglo, de píxeles cuadrados (elementos de imagen) dispuestos en columnas y filas. En una imagen en escala de grises de 8 bits, cada elemento de la imagen tiene una intensidad asignada que va de 0 a 255 . Las imágenes en escala de grises contienen sólo un canal, aunque otro tipo de imágenes, como las imágenes a color RGB, contienen más.

Las imágenes que se utilizarán son de diferentes resoluciones, formatos y tipos; entre estas imágenes, hay imágenes de 512*512, 800*600 y 300*209 píxeles, imágenes con formato TIFF, RAW y JPG e imágenes en escala de grises y RGB.

Las bibliotecas de Python que se utilizarán para manipular las imágenes en la presente práctica son:

**Matplotlib:** Biblioteca para crear imágenes y gráficas para visualizar información.

**OpenCV:** Biblioteca orientada a herramientas de visión computacional, además de *machine learning*.

**Scikit-Image:** Biblioteca de procesamiento de imágenes que nasa su entrada en la biblioteca de matrices y matemática Numpy.

**PIL o en inglés "*Python Imaging Library*":** Biblioteca para abrir, manipular y guardar diferentes formatos de imágenes.

**SciPy:** Biblioteca orientada a usos científicos, ingenieriles, matemáticos y de cómputo técnico.

## II. Desarrollo

La práctica consiste en 5 ejercicios, algunos de los cuales consisten en varias actividades. En todos los casos, se manipulan distintas imágenes con diferentes formatos y resoluciones.

Algunos sitios consultados como apoyo en el desarrollo de la práctica fueron , , y .

### *A. 4.1 Carpeta de imágenes*

Para la primera actividad se desarrolló un script de Python para leer y mostrar las diferentes imágenes a través de diferentes bibliotecas (Matplotlib, OpenCV, Sikit-image, PIL y Scipy), a continuación, se muestran los resultados obtenidos.

Imágenes que originalmente estaban en escala de grises:

Fig. 1 *cameraman.tif*.

Fig. 2 *lake.tif*.

Fig. 3 *house.tif*.

Imágenes a color:

Fig. 4 *lena_color_512.tif*.

Fig. 5 *peppers_color.tif*.

Fig. 6 *rosa800x600.raw*.

Fig. 7 *Anonymized20200210.dcm*.

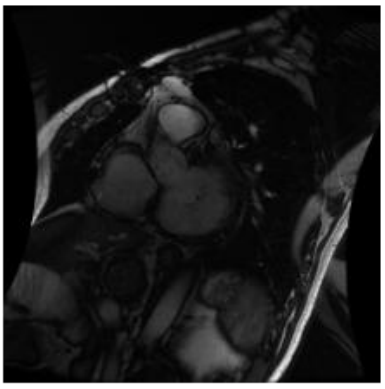


Fig. 8 *IM-0001-0007.dcm*.

Para las imágenes que tenían formatos RAW y DCM se empleó **Pydicom** y **Numpy** como recurso de lectura y muestreo.

Las imágenes TIF se mostraron con las bibliotecas Matplotlib, OpenCV, Scikit-Image y PIL; en el caso de SciPy, la función que permitía desplegar la imagen está obsoleta , en su lugar, se recomienda utilizar Matplotlib. En el caso de las imágenes DCM, como se mencionó anteriormente, se utilizó la biblioteca Pydicom. En el caso de la imagen RAW, se utilizó Numpy. En ambos casos, para desplegar como tal las imágenes, se utilizó Matplotlib.

Otro ejercicio de esta actividad consistió en imprimir la información de las imágenes. Entre estos datos, están el formato de la imagen, el tamaño y las dimensiones de la imagen. Por ejemplo, de la imagen *cameraman* (Fig. 1), se obtuvo la siguiente salida:

```
[METADATA]
File type = tiff
Size = 262750 b
ImageWidth = (512,)
ImageLength = (512,)
BitsPerSample = (8,)
Compression = (1,)
PhotometricInterpretation = (1,)
ImageDescription = ('',)
StripOffsets = (24, 9752, 19480, 29208,
38936, 48664, 58392, 68120, 77848, 87576,
97304, 107032, 116760, 126488, 136216,
145944, 155672, 165400, 175128, 184856,
194584, 204312, 214040, 223768, 233496,
243224, 252952)
Orientation = (1,)
SamplesPerPixel = (1,)
RowsPerStrip = (19,)
StripByteCounts = (9728, 9728, 9728, 9728,
9728, 9728, 9728, 9728, 9728, 9728, 9728,
9728, 9728, 9728, 9728, 9728, 9728, 9728,
9728, 9728, 9728, 9728, 9728, 9728, 9728,
9728, 9728, 9728, 9728, 9728, 9728, 9728,
9728, 9216)
XResolution = ((2834646, 100000),)
YResolution = ((2834646, 100000),)
ResolutionUnit = (3,)
NewSubfileType = (0,)
```

En el caso de la imagen de Lena (Fig. 4), se obtuvo lo siguiente:

```
[METADATA]
File type = tiff
Size = 787460 b
ImageWidth = (512,)
ImageLength = (512,)
BitsPerSample = (8, 8, 8)
Compression = (1,)
PhotometricInterpretation = (2,)
ResolutionUnit = (2,)
StripOffsets = (8, 7688, 15368, 23048,
30728, 38408, 46088, 53768, 61448, 69128,
76808, 84488, 92168, 99848, 107528, 115208,
122888, 130568, 138248, 145928, 153608,
161288, 168968, 176648, 184328, 192008,
199688, 207368, 215048, 222728, 230408,
238088, 245768, 253448, 261128, 268808,
276488, 284168, 291848, 299528, 307208,
314888, 322568, 330248, 337928, 345608,
353288, 360968, 368648, 376328, 384008,
391688, 399368, 407048, 414728, 422408,
430088, 437768, 445448, 453128, 460808,
468488, 476168, 483848, 491528, 499208,
506888, 514568, 522248, 529928, 537608,
545288, 552968, 560648, 568328, 576008,
583688, 591368, 599048, 606728, 614408,
622088, 629768, 637448, 645128, 652808,
660488, 668168, 675848, 683528, 691208,
698888, 706568, 714248, 721928, 729608,
737288, 744968, 752648, 760328, 768008,
775688, 783368)
Orientation = (1,)
SamplesPerPixel = (3,)
RowsPerStrip = (5,)
StripByteCounts = (7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
7680, 7680, 7680, 7680, 7680, 7680, 7680,
3072)
XResolution = ((1207959552, 16777216),)
YResolution = ((1207959552, 16777216),)
PlanarConfiguration = (1,)
```

Finalmente, se obtuvo también la información de la imagen de los pimientos (Fig. 5), sin embargo, no se incluye aquí por la extensión.

La última parte de esta actividad consistió en desarrollar un script para cambiar el espacio de color dos imágenes (Fig. 4 y Fig. 5), además de mostrar los canales de dichas imágenes en RGB y HSV.

Primero, se cambió de RGB a escala de grises.

Fig. 9 Resultado de cambio de RGB a escala de grises en la imagen de Lena con OpenCV.

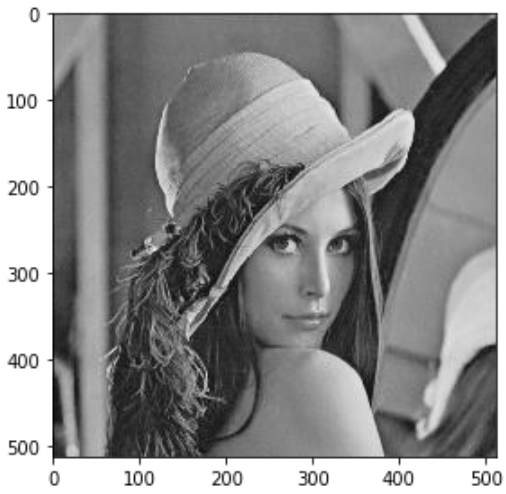



Fig. 10 Resultado de cambio de RGB a escala de grises en la imagen de Lena con Scikit-Image.

Fig. 11 Resultado de cambio de RGB a escala de grises en la imagen de los pimientos con OpenCV.

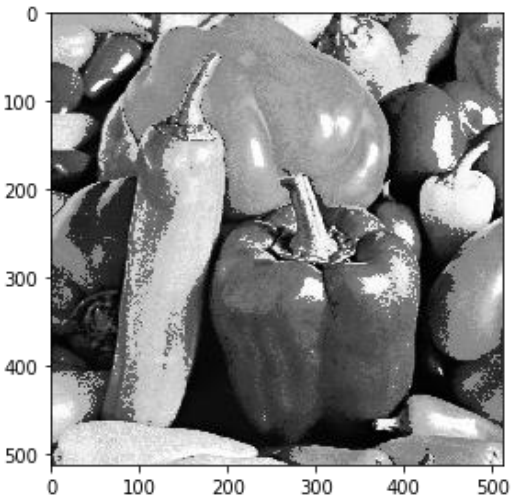



Fig. 12 Resultado de cambio de RGB a escala de grises en la imagen de los pimientos con Scikit-Image.

En segundo lugar, se realizó un cambio de RGB a YUV.

Fig. 13 Resultado de cambio de RGB a YUV en la imagen de Lena con OpenCV.

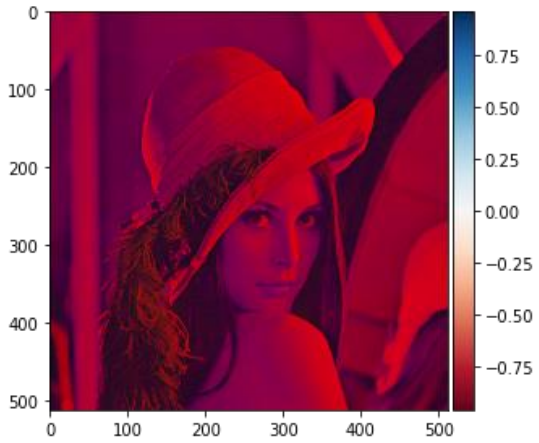



Fig. 14 Resultado de cambio de RGB a YUV en la imagen de Lena con Scikit-Image.

Fig. 15 Resultado de cambio de RGB a YUV en la imagen de los pimientos con OpenCV.

A continuación, se realizó un cambio de RGB a HSV.

Fig. 16 Resultado de cambio de RGB a HSV en la imagen de Lena con OpenCV.

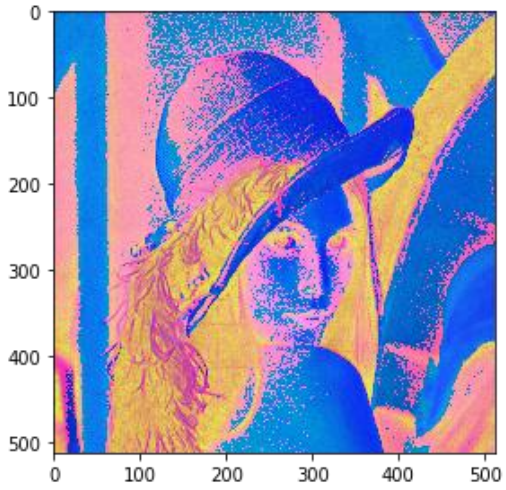



Fig. 17 Resultado de cambio de RGB a HSV en la imagen de Lena con Scikit-Image.

Fig. 18 Resultado RGB a HSV en la imagen de los pimientos con OpenCV.

Finalmente, se mostraron los canales tanto para RGB como para HSV. A continuación, sólo se muestran los resultados para la imagen de Lena, los resultados de la imagen del pimiento se muestran en el anexo.

Fig. 19 Canal B (blue) de la imagen de Lena.

Fig. 20 Canal G (green) de la imagen de Lena.

Fig. 21 Canal R (red) de la imagen de Lena.

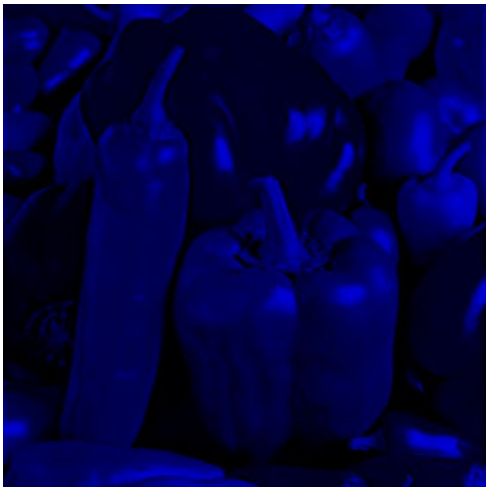


Fig. 22  Canal B (blue) de la imagen de los pimientos.

Fig. 23  Canal G (green) de la imagen de los pimientos.

Fig. 24  Canal R (red) de la imagen de los pimientos.

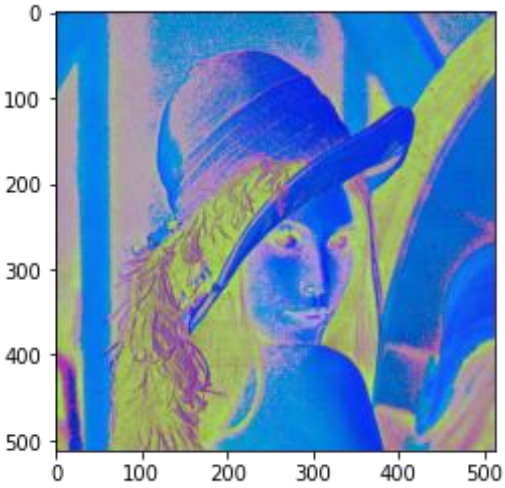



Fig. 25  Imagen HSV de la imagen de Lena obtenida con OpenCV y desplegada con Matplotlib.

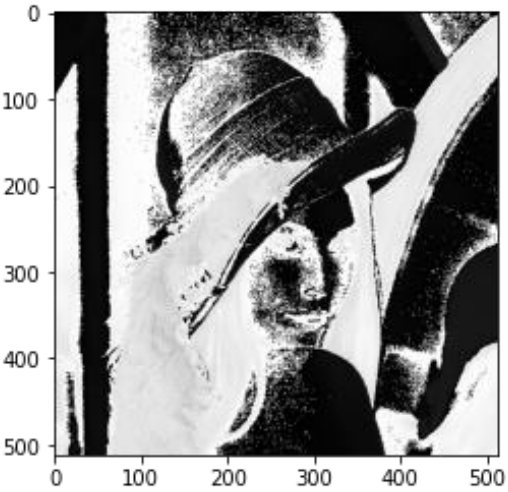



Fig. 26  Canal H (hue) de la imagen de Lena.

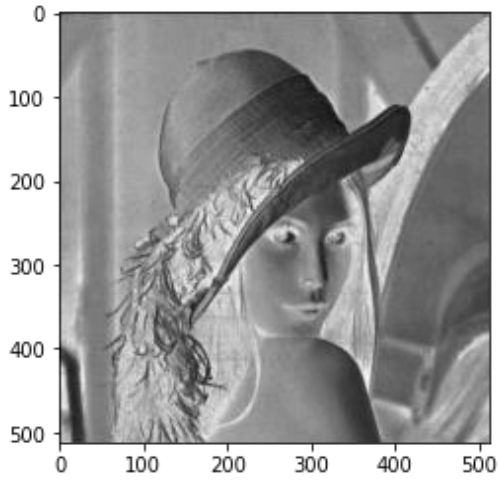



Fig. 27  Canal S (saturation) de la imagen de Lena.

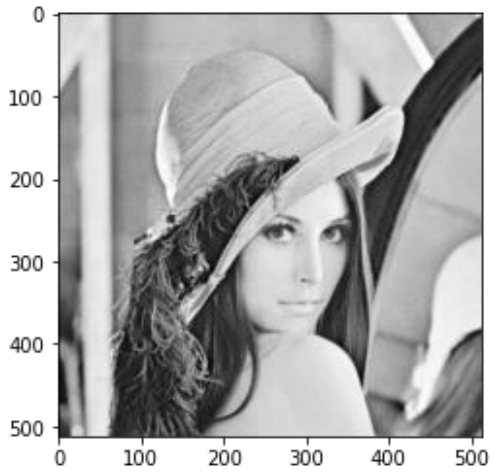



Fig. 28 Canal V (value) de la imagen de Lena.

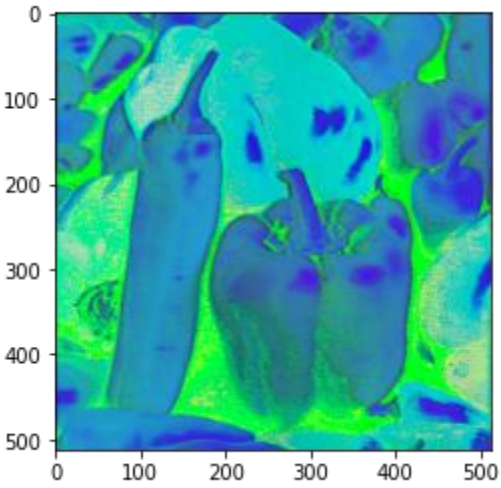



Fig. 29 Imagen HSV de la imagen de los pimientos obtenida con OpenCV y desplegada con Matplotlib.

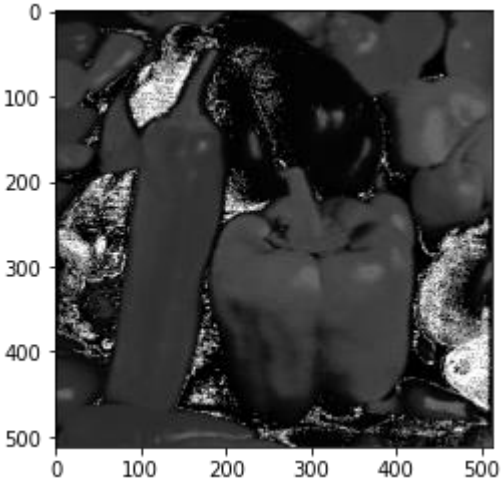



Fig. 30 Canal H (hue) de la imagen de los pimientos.

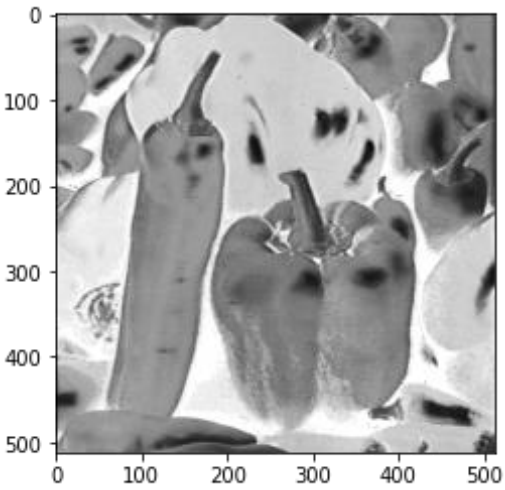



Fig. 31 Canal S (saturation) de la imagen de los pimientos.

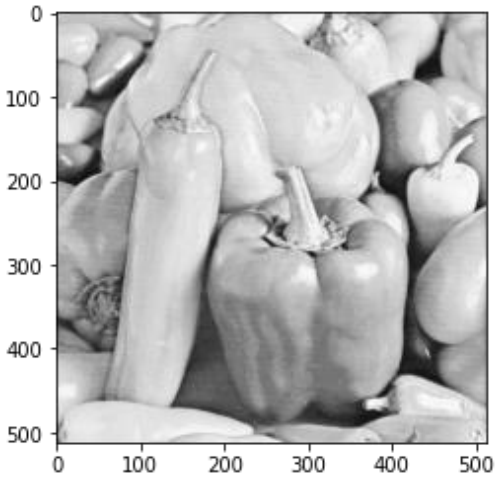



Fig. 32 Canal V (value) de la imagen de los pimientos.

### *B. 4.2. "Decimation" en imágenes*

En esta actividad, se desarrolló un script para realizar *decimation* en una imagen. Este proceso involucra aplicar un filtro antes de realizar el submuestreo; según , se puede aplicar un filtro gaussiano, por lo que se aplicó dicho filtro. Sin embargo, también se realizó el proceso sin aplicar filtro, lo que arrojó un resultado satisfactorio, pues no se produjo *anti-aliasing* evidente. Cabe destacar que las imágenes se convirtieron a escala de grises antes de ser procesadas.

La imagen procesada en esta actividad fue la imagen de Lena, mostrada en la Fig. 4. Como se puede observar en las imágenes mostradas a continuación, cuando se aplicó el filtro gaussiano, el resultado es ligeramente más "suave", mientras que, en el otro caso, el resultado se nota más nítido.

Fig. 33 Resultado con filtro gaussiano.

Fig. 34 Resultado sin filtro gaussiano.

### *C. 4.3. Redimensión y rotación de imágenes*

Para esta actividad, se desarrollaron scripts para redimensionar y rotar imágenes utilizando funciones de Python ya existentes.

En primer lugar, se realizaron redimensionamientos a 10 y 3 veces el tamaño original de las imágenes escogidas; estas imágenes son la de Lena (Fig. 4) y la mostrada a continuación.

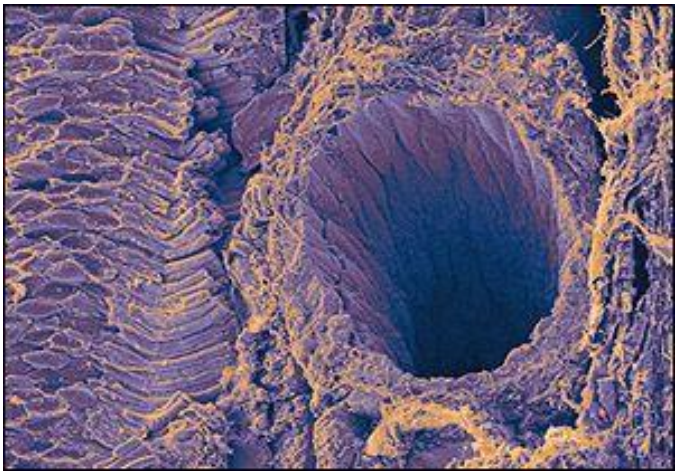


Fig. 35 *retinaRGB.jpg*.

Evidentemente, por el tamaño de los resultados, no es posible mostrarlos directamente; sin embargo, el cambio en el tamaño se puede notar fácilmente en las dimensiones de las imágenes. En el caso de la Fig. 35, la resolución original era de 300*209 píxeles, y el resultado fue de 3000*2090 píxeles en el caso de la redimensión a 10 veces el tamaño original, y de 900*627 píxeles en el caso de la redimensión a 3 veces el tamaño original. Para la Fig. 4, la resolución original era de 512*512 píxeles, y el resultado fue de 5120*5120 píxeles en el caso de la redimensión a 10 veces el tamaño original, y de 1536*1536 píxeles en el caso de la redimensión a 3 veces el tamaño original.

Finalmente, fue necesario rotar una imagen varias veces y guardarla en formato PNG.

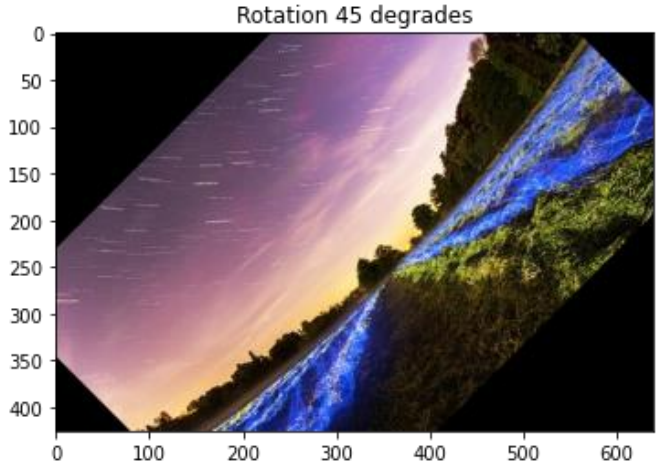



Fig. 36 Rotación de la imagen a 45 grados.

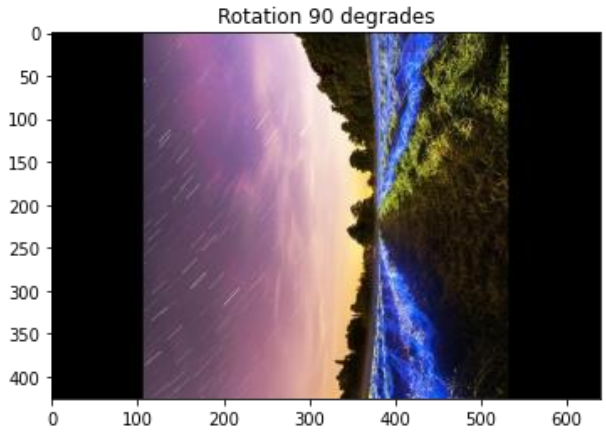



Fig. 37 Rotación de la imagen a 90 grados.

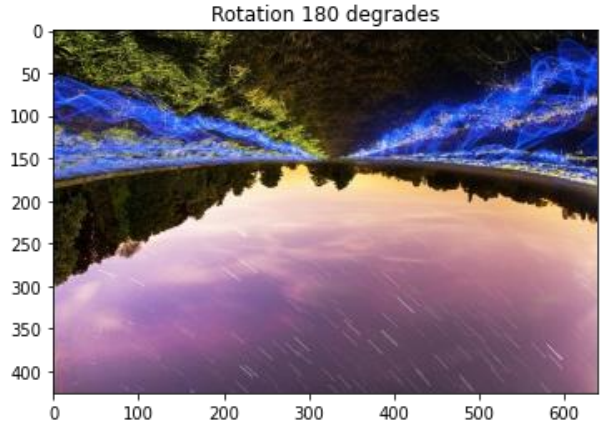



Fig. 38 Rotación de la imagen a 180 grados.

### *D. 4.4. Recorte y guardado de imágenes*

En esta actividad, sólo se requirió convertir la imagen de los pimientos (Fig. 5) a escala de grises, recortarla y guardarla en formato JPG. Para esto, se utilizó la función *crop* de PIL.

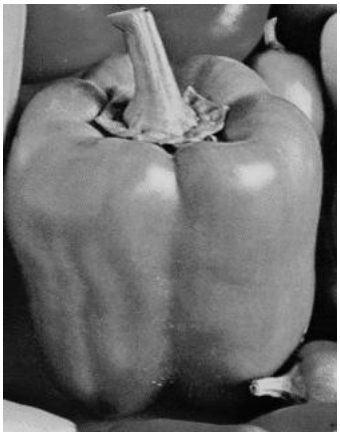


Fig. 39 Recorte de la imagen de los pimientos en escala de grises.

### *E. 4.5. Formato RAW*

Como se mencionó anteriormente, para desplegar la imagen RAW, se utilizó Numpy y Matplotlib; más concretamente, la instrucción *np.fromfile(dirRaw, np.uint8).reshape(800, 600)* junto con *imshow()*.

Fig. 40 Resultado del despliegue de la imagen RAW.

## III. CONCLUSIONES

En la presente práctica aprendimos a utilizar clases y métodos de distintas bibliotecas de Python que tienen como propósito principal el procesamiento de imágenes. A su vez aprendimos a trabajar con distintos tipos de imágenes desde formatos más tradicionales como ***.tif*** hasta imágenes de formatos crudos como "***.raw*****"** o de ambientes médicos "***.dcm*****"**, también aprendimos a manejar estas mismas imágenes en distintos espacios de color que están destinados a entornos concretos es el caso del ***HSV*** o ***YUV***.

Por último, aprendimos a realizar distintas transformaciones con las imágenes, desde aspectos básicos como la rotación hasta aspectos más complejos como el escalamiento donde debía contemplarse el muestreo y otros aspectos relacionados con la interpolación o con la "*decimation*".

## REFERENCIAS

[1] Hubble Space Telescope, "Introduction to image processing", esahubble.org, s. f. [En línea]. Disponible en: https://esahubble.org/static/projects/fits_liberator/image_processing.pdf. [Consultado el 6 de Septiembre del 2021].

[2] A. Ybodon, "[Python In-depth] Image handling in Python with OpenCV (2)", Medium, 2019. [En línea]. Disponible en: https://financial-engineering.medium.com/python-in-depth-image-handling-in-python-with-opencv-2-5260e56c186f. [Consultado el 4 de Septiembre del 2021].

[3] A. P. Chazhoor, "Image processing using scikit image", Towards Data Science, 2019. [En línea]. Disponible en: https://towardsdatascience.com/image-processing-using-scikit-image-cb57ce4321ed. [Consultado el 4 de Septiembre del 2021].

[4] R. Khandelwal, "Loading and Saving Images in Python", Towards Data Science, 2020. [En línea]. Disponible en: https://towardsdatascience.com/loading-and-saving-images-in-python-ba5a1f5058fb. [Consultado el 4 de Septiembre del 2021].

[5] Scikit-Image, “A crash course on NumPy for images”, s. f. [En línea]. Disponible en: https://scikit-image.org/docs/stable/user_guide/numpy_images.html. [Consultado el 4 de Septiembre del 2021].

[6] SciPy, “scipy.misc.imshow”, 2019. [En línea]. Disponible en: https://docs.scipy.org/doc/scipy-1.2.1/reference/generated/scipy.misc.imshow.html. [Consultado el 6 de Septiembre del 2021].

[7] M. Fabien, “Image subsampling and downsampling”, 2019. [En línea]. Disponible en: https://maelfabien.github.io/computervision/cv_3/. [Consultado el 6 de Septiembre del 2021].

## IV. ANEXO

Para el apartado del anexo adjuntamos en este mismo documento el “ejercicio básico” para manipulación de imágenes, además de que compartimos los enlaces de nuestros scripts desarrollados en Google Colaboratory.

Script del ejercicio:
https://drive.google.com/file/d/1K74g_FhaFyyH9SHUCAqLg_myrFutcuLV/view?usp=sharing

Script de la práctica:
https://colab.research.google.com/drive/1tNMeXyoag5V7WIGAfk_oNlcjeZtDiZ_7?usp=sharing

## Ejercicios Básicos

**Integrantes**:

- Murrieta Villegas Alfonso
- C. Nathaniel Ceballos Equihua

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

## 1. Abrir y escribir una imagen en un archivo

```
from scipy import misc
from scipy import ndimage
import imageio as io
import matplotlib.pyplot as plt
import numpy as np
```

```
f = misc.face()
io.imwrite('face.png', f) # Para guardar
plt.imshow(f)
plt.show()
```

## 2. Creación de un arreglo numpy de un archivo de imagen

```
from scipy import misc
import imageio as io

face = misc.face()
io.imwrite('face.png', face)
face = io.imread('face.png')
print(type(face))
face.shape
#face.dtype
```

```
<class 'imageio.core.util.Array'>

(768, 1024, 3)
```

```
from skimage.color import  rgb2gray
face = rgb2gray(face)
print(type(face))
print(face.shape)
plt.imshow(face, cmap = 'gray')
plt.show()
```

```
<class 'numpy.ndarray'>
(768, 1024)
```


**¿De qué tipo es la variable face?**

Es una clase u objeto de tipo "imageio", que podría interpretarse como un arreglo o matriz con datos de tipo "int de 8 bits"

**¿Qué resultado arroja face.shape?**

Nos muestra en los dos primeros datos el tamaño en pixeles ancho y largo, además de la "profundidad", es decir, en este caso el RGB por eso 3

**Si fuera una imagen en tonos de gris, ¿cuál sería el resultado esperado de face.shape?**

Una matriz de datos, específicamente un arreglo bidimensional donde al igual que en en la versión de color se tiene el largo y el ancho solo que no se tiene las componentes del esquema de color

## 3. Abrir archivos raw

```
face = misc.face()
io.imsave('face.png', face)
face = io.imread('face.png')

face.tofile('face.raw')
face_from_raw = np.fromfile('face.raw', dtype=np.uint8)
face_from_raw.shape
```

```
(2359296,)
```

**¿Qué resultado arroja la primera instrucción face_from_raw.shape?**

(2359296,)

## 4. Despliegue de imágenes

```
f = misc.face(gray=True)
plt.imshow(f, cmap=plt.cm.gray)
```

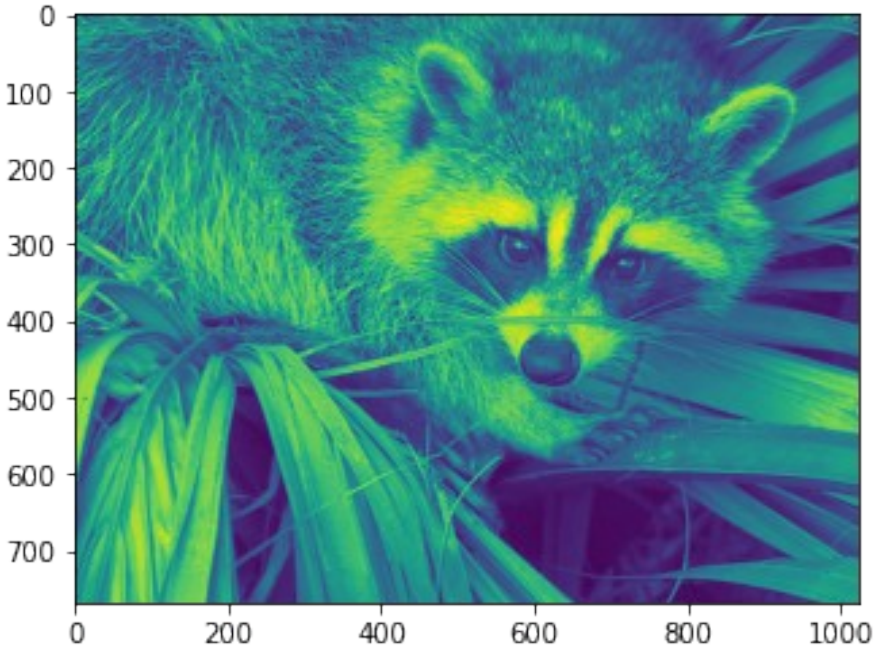
```
<matplotlib.image.AxesImage at 0x7f13d85590d0>
```


**¿Qué pasa si al desplegar la imagen con plt.imshow no se especifica el mapa de color plt.cm.gray?**

Tomaria el valor de la variable cmap

```
face = misc.face(gray=True)
plt.imshow(face)
```

```
<matplotlib.image.AxesImage at 0x7f13d84ca4d0>
```



**¿Cuál es el mapa de color default de imshow?**

Por defecto el color default de imshow es 'viridis'

**Imprima la forma de f (propiedad shape de f) y compare contra la forma de la misma imagen a color**

```
f = misc.face(gray=True) #imagen en escala de grises
print("Gray image:",f.shape)
```

```
Gray image: (768, 1024)
```

```
f = misc.face() # imagen en colores
print("Color image:", f.shape)
```

```
Color image: (768, 1024, 3)
```

Observamos como la componente de los colores no se contempla en la versión blanco/negro (gray)

**Se puede incrementar el contraste especificando los valores mínimo y máximo en el despliegue**

```
f = misc.face(gray=True)
plt.imshow(f, cmap=plt.cm.gray, vmin=50, vmax=200)
plt.axis('off')
```

```
(-0.5, 1023.5, 767.5, -0.5)
```

**Se pueden dibujar líneas de contorno con la instrucción contour.**

```
plt.imshow(f, cmap=plt.cm.gray, vmin=30, vmax=200)
```

```
plt.axis('off')
plt.contour(f, [50, 200])
```

```
<matplotlib.contour.QuadContourSet at 0x7f13d84287d0>
```

**Investigue y ponga una breve descripción sobre la instrucción contour.**

Contour sirve para dibujar las líneas de contorno. Recibe como parámetros a variables predictoras X,Y, que pueden ser como una matriz y una variable de respuesta que sería Z como contornos.

contour([X,Y,] Z, [levels], *kwargs)

- X, Y similar a una matriz, opcional Las coordenadas de los valores en Z .
- X y Y debe ser tanto 2-D con la misma forma como Z (por ejemplo, creada a través de numpy.meshgrid), o ambos deben ser 1-D de tal manera que es el número de columnas en Z y es el número de filas en Z .len(X) == Mlen(Y) == N

Si no se da, que se supone que son número entero índices, es decir , .X = range(M)Y = range(N)

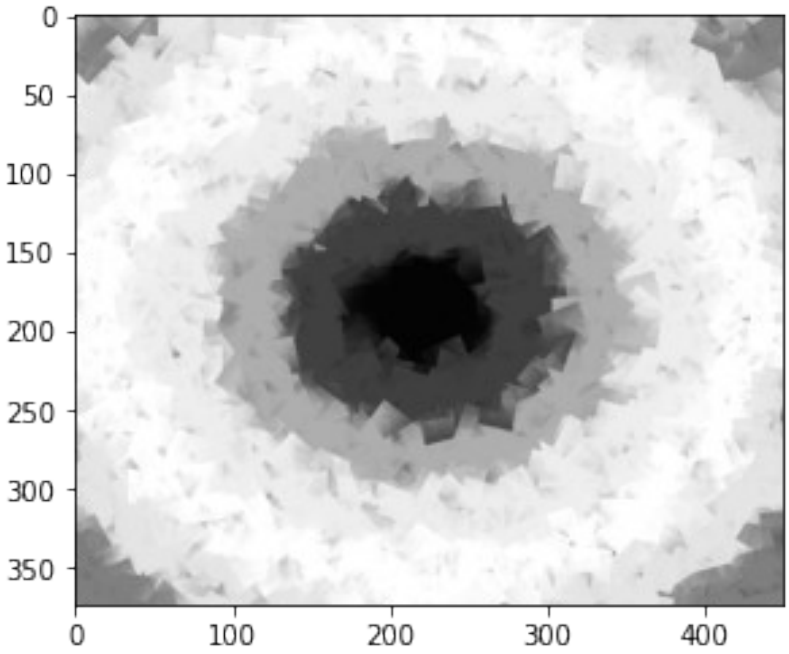
- Z como una matriz (N, M) Los valores de altura sobre los que se dibuja el contorno.
- levels: int o similar a una matriz, opcional Determina el número y las posiciones de las curvas de nivel / regiones.

**Obtenga y despliegue los contornos de la imagen contour_gray.png con las siguientes instrucciones (antes debe cargar la imagen contour_gray.png en la variable f):**

```
f= misc.face(gray=True)
f=
```

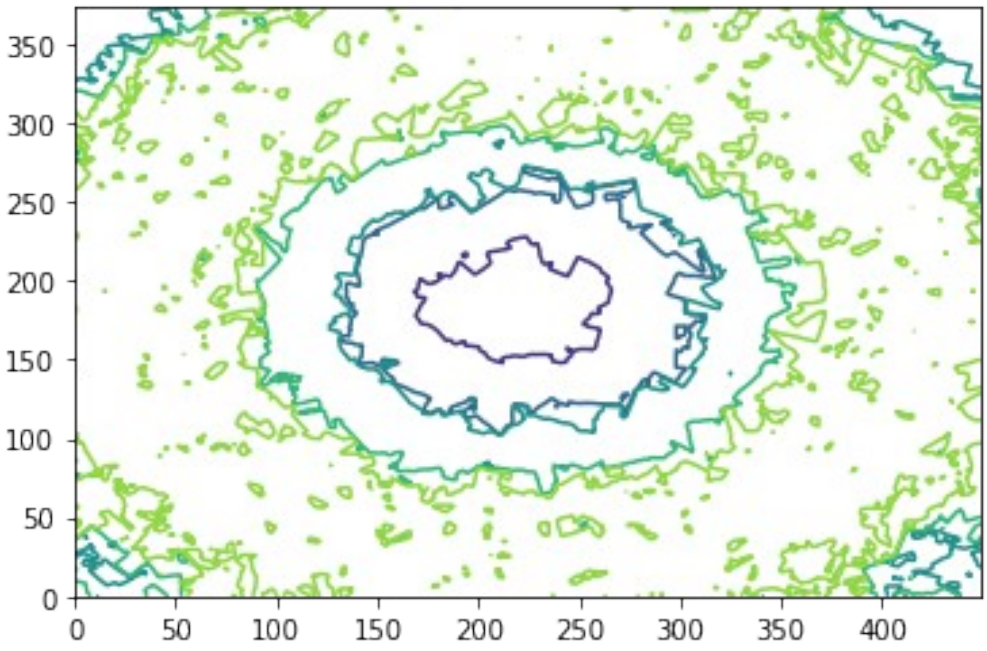
```
io.imread('/content/drive/MyDrive/Reconocimiento/homework/contour_gray
.png')
plt.imshow(f, cmap=plt.cm.gray)
```

```
<matplotlib.image.AxesImage at 0x7f13d7fbf390>
```



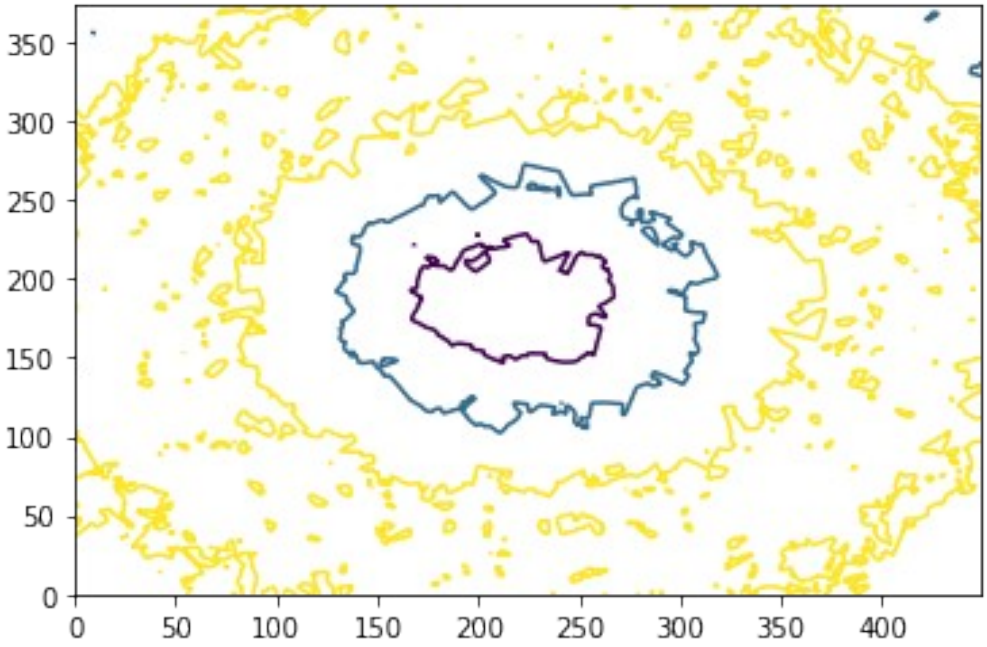

```
plt.contour(f, 5)
```

```
<matplotlib.contour.QuadContourSet at 0x7f13d7f25dd0>
```



```
plt.contour(f, [50, 100, 200])
```

```
<matplotlib.contour.QuadContourSet at 0x7f13d7e85350>
```




## 5. Manipulaciones Básicas

**¿Cuánto vale el pixel face[0, 40]?**

```
face = misc.face(gray=True)
face[0, 40]
```

```
127
```

Vale 127

**¿Qué efecto tiene la instrucción face 100:120 = 255 en la imagen de abajo?**

```
face = misc.face(gray=True)
face[100:120] = 255
plt.imshow(face, cmap=plt.cm.gray)
plt.show()
```


Asigna el valor de 255 a los pixeles verticales del 100 al 120 (Es decir toda esa sección es una línea de clor blanco)

**Pinte una franja vertical gris en la imagen que vaya de la columna 200 a la columna 220. Tip: en los índices tiene que elegir todas las filas**

```
face = misc.face(gray=True)
face[:, 200:220] = 125
plt.imshow(face, cmap=plt.cm.gray)
plt.show()
```


**¿Cuánto vale lx, ly?**

```
lx, ly = face.shape
print(lx, ly)
```

```
768 1024
```

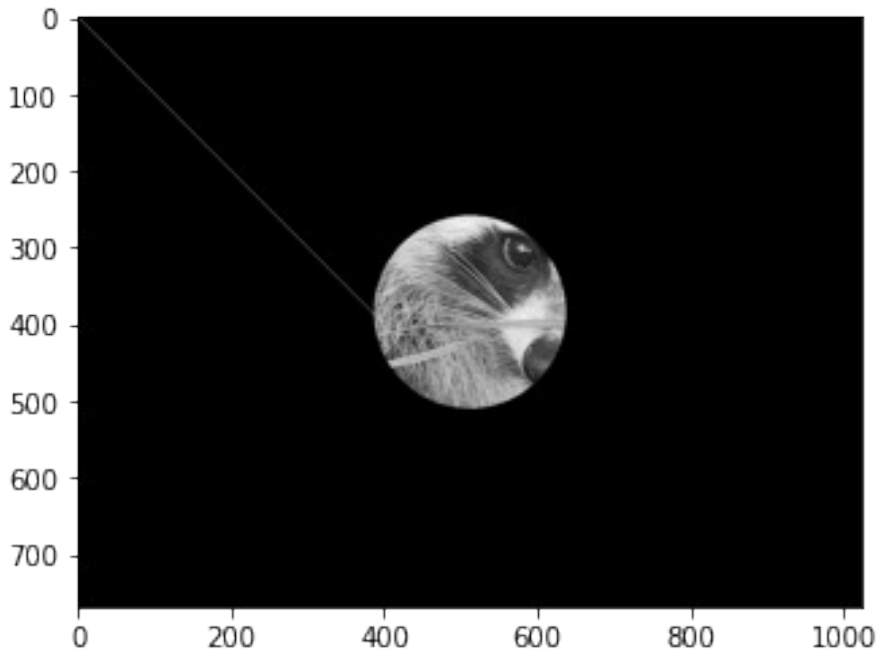
**¿Qué efecto tiene en la imagen la instrucción face[range(400), range(400)] = 255?**

```
X, Y = np.ogrid[0:lx, 0:ly]
mask = (X - lx / 2) ** 2 + (Y - ly / 2) ** 2 > lx * ly / 4
# Masks
face[mask] = 0

face[range(400), range(400)] = 255
plt.imshow(face, cmap=plt.cm.gray)
plt.show()
```


Crea una linea del punto (0,0) al (400,400)

**Modifique el código para que la máscara sea un círculo más pequeño y despliegue el resultado.**

```
face = misc.face(gray=True)
X, Y = np.ogrid[0:lx, 0:ly]
mask = (X - lx / 2) ** 2 + (Y - ly / 2) ** 2 > lx * ly / 50
# Masks
face[mask] = 0

face[range(400), range(400)] = 255
plt.imshow(face, cmap=plt.cm.gray)
plt.show()
```




#6 Información Estadística

**Se puede obtener información estadística de la imagen usando el módulo misc.**

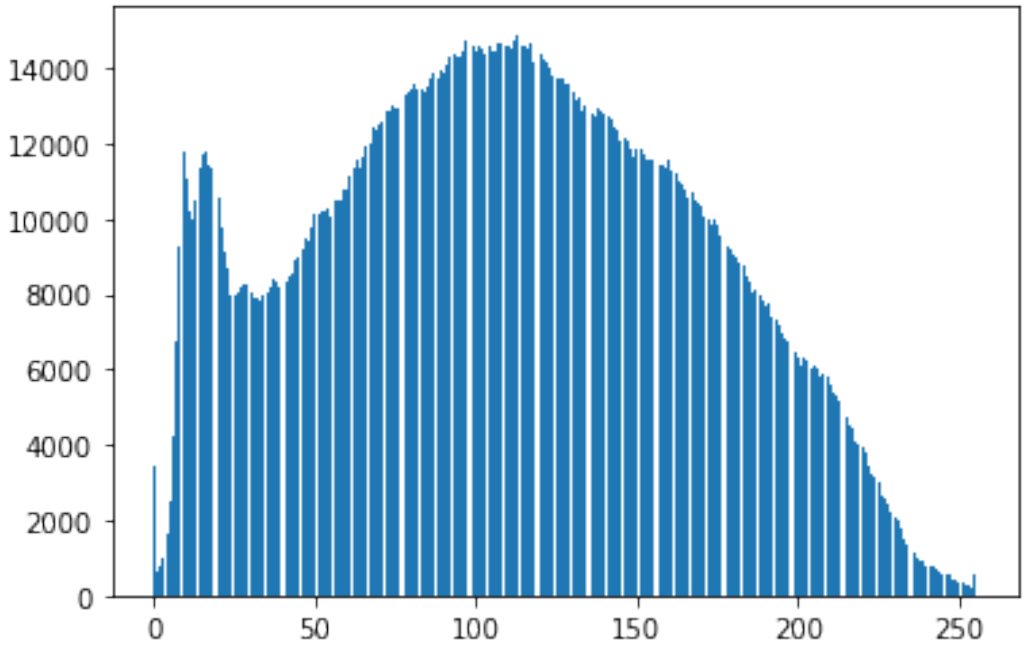
```
face = misc.face(gray=True)
face.mean()
face.max(), face.min()
```

```
(250, 0)
```

**Una manera de obtener el histograma es con la instrucción histogram de numpy.**

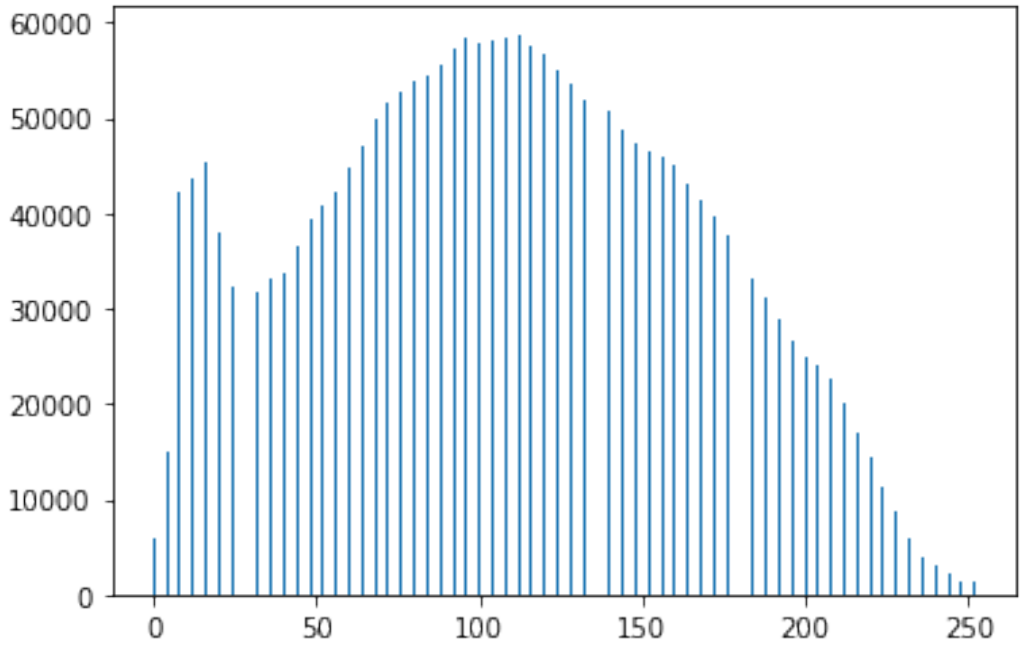
**Despliegue el histograma de la imagen en grises del mapache original (es decir, antes de poner las franjas y de aplicar la máscara)**

```
face = misc.face(gray=True)
face = io.imread('face.png')

hist, bins = np.histogram(face, bins=256, range=(0,256))
plt.bar(bins[0:-1], hist)
plt.show()
```



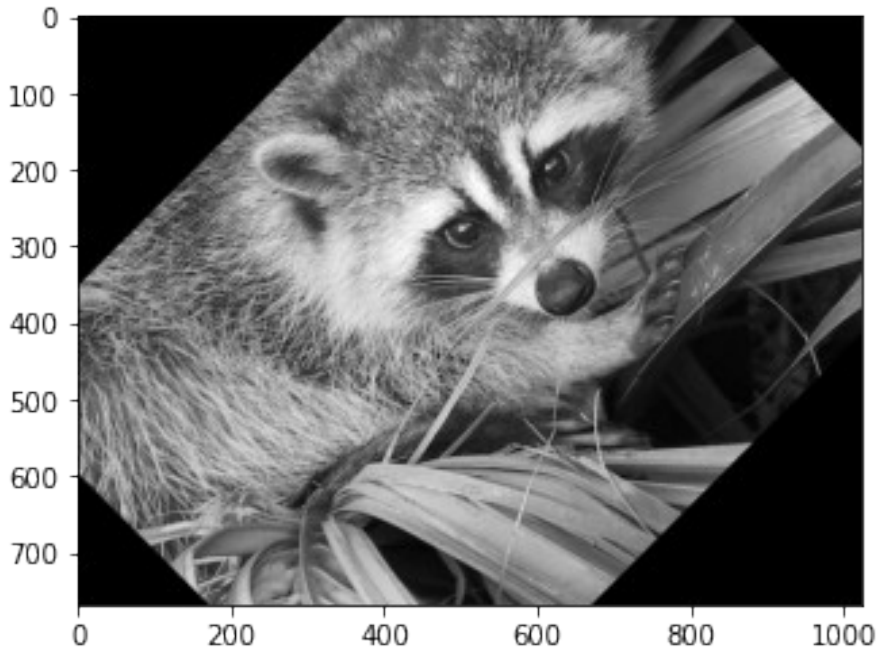

**Modifique el código para que sólo se tengan 64 bins en el histograma y muestre el histograma resultante.**

```
face = misc.face(gray=True)
face = io.imread('face.png')

hist, bins = np.histogram(face, bins=64, range=(0,256))
plt.bar(bins[0:-1], hist)
plt.show()
```



**Transformaciones geométricas**

```
face = misc.face(gray=True)
lx, ly = face.shape
# Recorte
crop_face = face[lx // 4: - lx // 4, ly // 4: - ly // 4]
# up <-> down Voltear verticalmente
flip_ud_face = np.flipud(face)
# Rotación
rotate_face = ndimage.rotate(face, 45)
rotate_face_noreshape = ndimage.rotate(face, 45, reshape=False)
plt.imshow(rotate_face_noreshape, cmap=plt.cm.gray)
plt.show()
```




**Investigue el operador // y ponga una breve descripción.**

El operador "//" sirve para calcular el cociente de una división.

1. Su resultado siempre será un número entero
2. Este operador tiene la misma prioridad que la división

**¿Qué efecto tiene el signo negativo en crop_face = face[lx // 4: - lx // 4, ly // 4: - ly // 4]?**

El sentido de recorte

**Voltee la imagen original del mapache, pero esta vez horizontalmente, de izquierda a derecha.**

```
f = misc.face(gray=True)
io.imwrite('face.png', f)

flip_ud_face = np.flipud(face)
rotate_face = ndimage.rotate(flip_ud_face, 180)
plt.imshow(rotate_face, cmap=plt.cm.gray)
plt.show()
plt.show()
```