Procesamiento de imagenes

Importar imagenes

In [ ]:
```
from PIL import Image
import matplotlib.pyplot as plt

suricatas = Image.open(r"images\suricatas.jpeg")

print(isinstance(suricatas, Image.Image))
print(type(suricatas))
print(id(suricatas))
```

```
True
<class 'PIL.JpegImagePlugin.JpegImageFile'>
2478439134112
```

Imprimiendo imagen

In [ ]:
```
suricatas
```

Out[ ]:

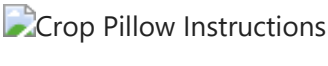
In [ ]:
```
print("Image format:",suricatas.format)
print("Image size:",suricatas.size)
print("Image mode:",suricatas.mode)
```

```
Image format: JPEG
Image size: (948, 465)
Image mode: RGB
```

Operaciones con imagen

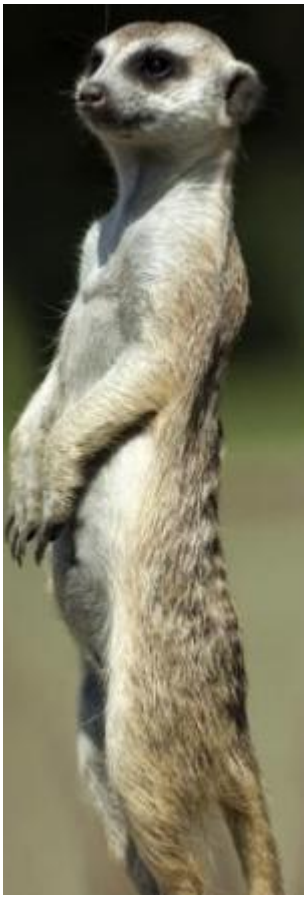
Crop imagen

Ejemplo de como funciona la sintaxis del metodo crop.



In [ ]:
```
crop_suricatas = suricatas.crop(box=(300,20,450,465))
crop_suricatas
```

Out[ ]:



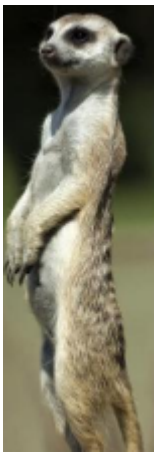
Redimensionar imagen

In [ ]:
```
# Redimensiona la imagen en la proporcion deseada
suricatas_lowres = crop_suricatas.resize((crop_suricatas.width//2,crop_suricatas.heigh
suricatas_lowres
```

Out[ ]:

In [ ]:
```
# Redimensiona la imagen en la misma proporcion
crop_suricatas.reduce(2)
```

Out[ ]:



In [ ]:
```
# Exporta las imagenes a la carpeta export
crop_suricatas.save(r"export\crop_suricatas.jpg")
suricatas_lowres.save(r"export\suricatas_lowres.jpg")
```

Manejo de imagen

Flip

Flip es un filtro que te permite realizar diferentes transformaciones, segun el argumento dado:

- FLIP_LEFT_RIGHT: Rota de manera horizontal.
- FLIP_TOP_BOTTOM: Rota de manera vertical.
- ROTATE_90: Rota 90 grados.
- ROTATE_180: Rota 180 grados.
- ROTATE_270: Rota 270 grados.

In [ ]:
```
flip_suricatas = suricatas.transpose(Image.FLIP_TOP_BOTTOM)
flip_suricatas
```

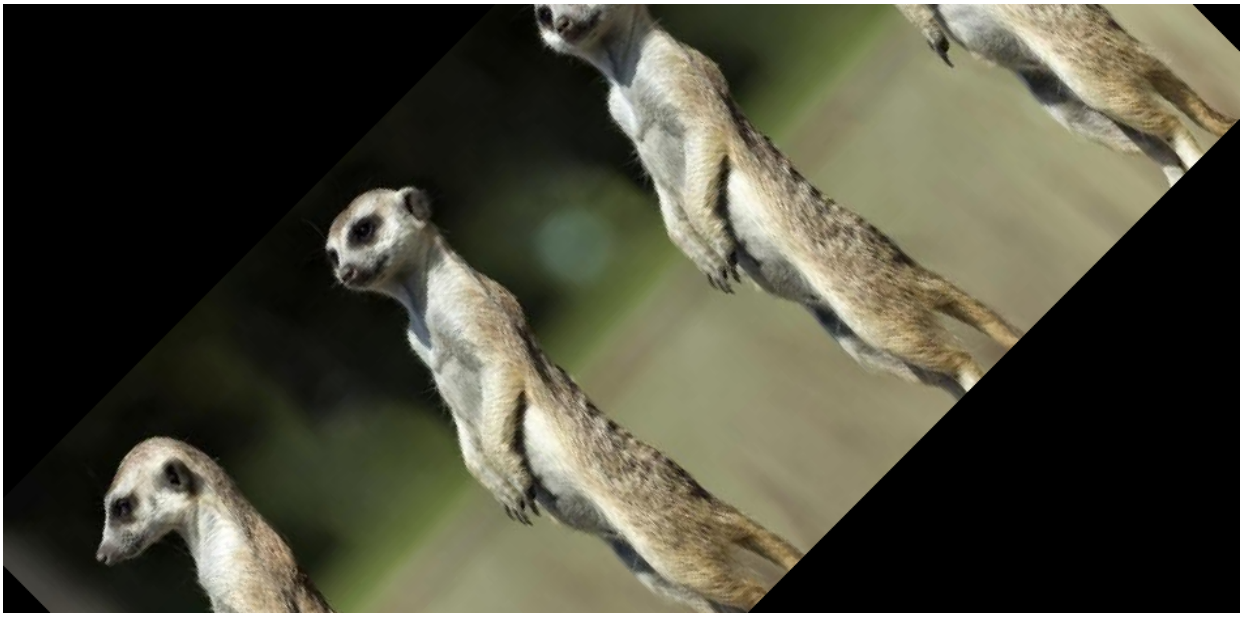
```
C:\Users\chccr\AppData\Local\Temp\ipykernel_2568\3161080539.py:1: DeprecationWarning:
FLIP_TOP_BOTTOM is deprecated and will be removed in Pillow 10 (2023-07-01). Use Tran
spose.FLIP_TOP_BOTTOM instead.
  flip_suricatas = suricatas.transpose(Image.FLIP_TOP_BOTTOM)
```

Out[ ]:

Rotacion

In [ ]:
```
rotated_suricatas = suricatas.rotate(45)
rotated_suricatas
```

Out[ ]:



In [ ]:
```
rotated_suricatas = suricatas.rotate(45, expand=True)
rotated_suricatas
```

Out[ ]:

Modo de imagen

Modos de color en imagenes "RGB", "CMYK" y "Gray".

In [ ]:

```
def get_concat_h(im1, im2):
    """Concatena dos imágenes del mismo tamaño y altura.

    Args:
        im1 (PIL.Image): Primera imagen.
        im2 (PIL.Image): Segunda imagen.

    Returns:
        PIL.Image: Imagen concatenada
    """
    image_merge = Image.new('RGB', (im1.width + im2.width, im1.height))
    image_merge.paste(im1, (0, 0))
    image_merge.paste(im2, (im1.width, 0))
    return image_merge.reduce(2)
```

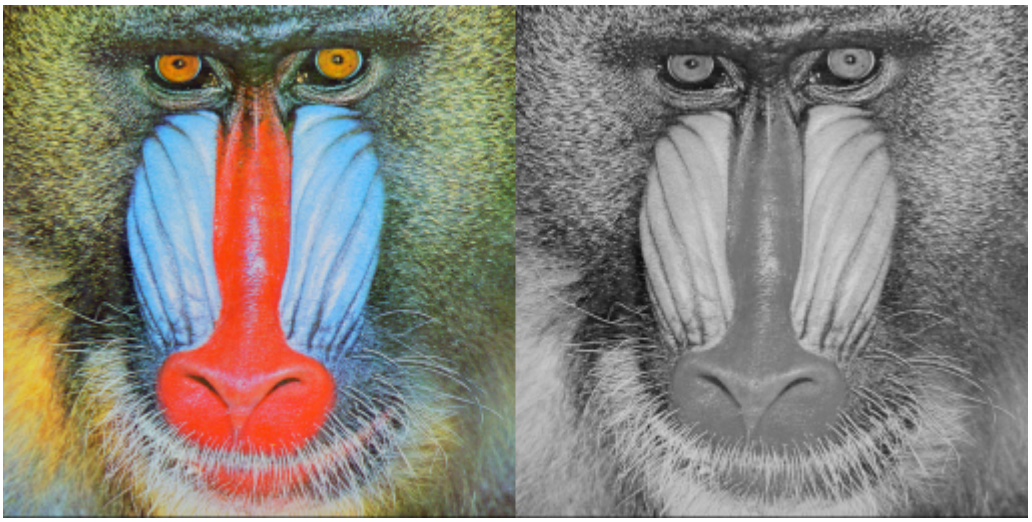
In [ ]:
```
baboon = Image.open(r"images\baboon.png")
baboon.load()
baboon.reduce(2)
```

Out[ ]:

Grayscale image

In [ ]:
```
# Convierte la imagen a escala de grises.
get_concat_h(baboon, baboon.convert("L"))
```

Out[ ]:



Capas RBG

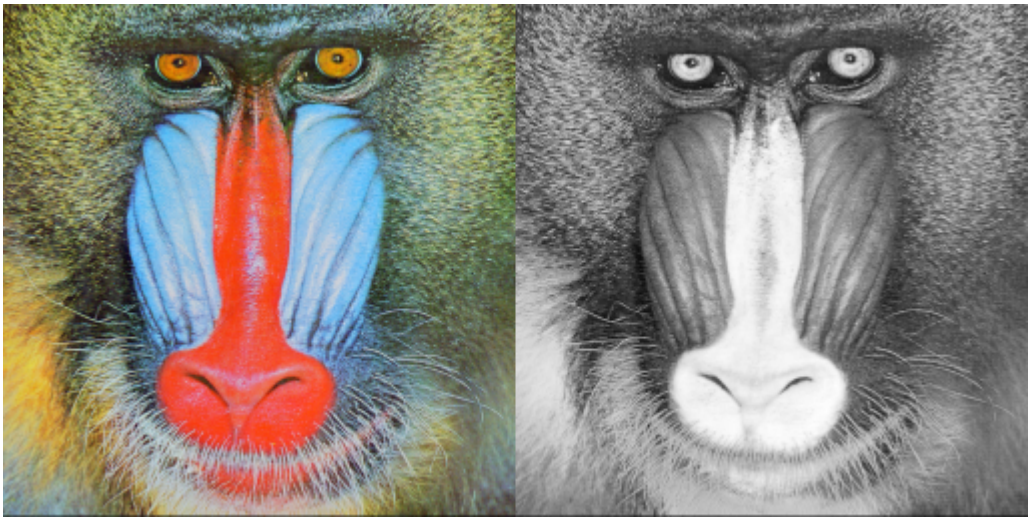
In [ ]:
```
# Muestra las capas segun el Modo de colores
print(baboon.getbands())

# Separa y asigna las diferentes capas de colores.
red, green, blue = baboon.split()
```

```
('R', 'G', 'B')
```

Las capas de colores se encuentran en modo escala de grises.

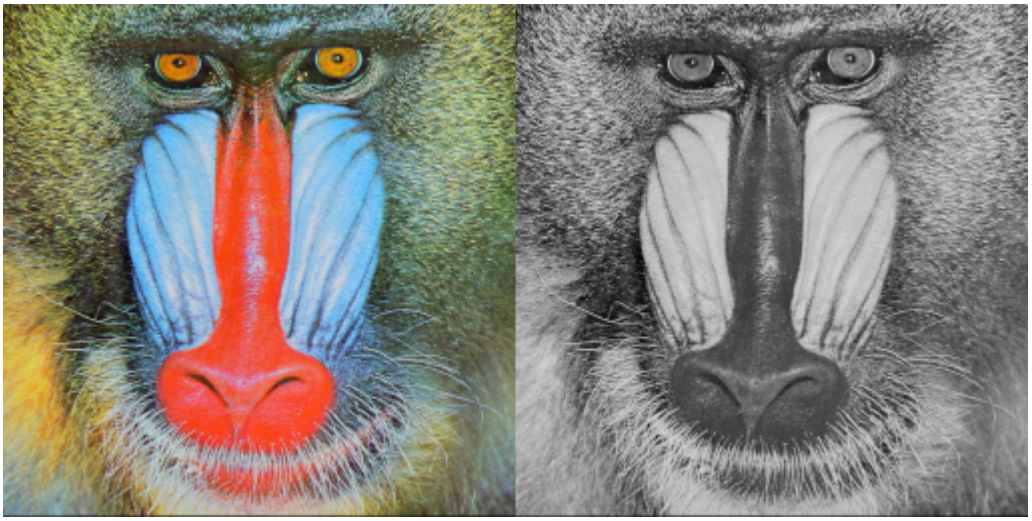
In [ ]:
```
# Capa roja
get_concat_h(baboon,red)
```

Out[ ]:



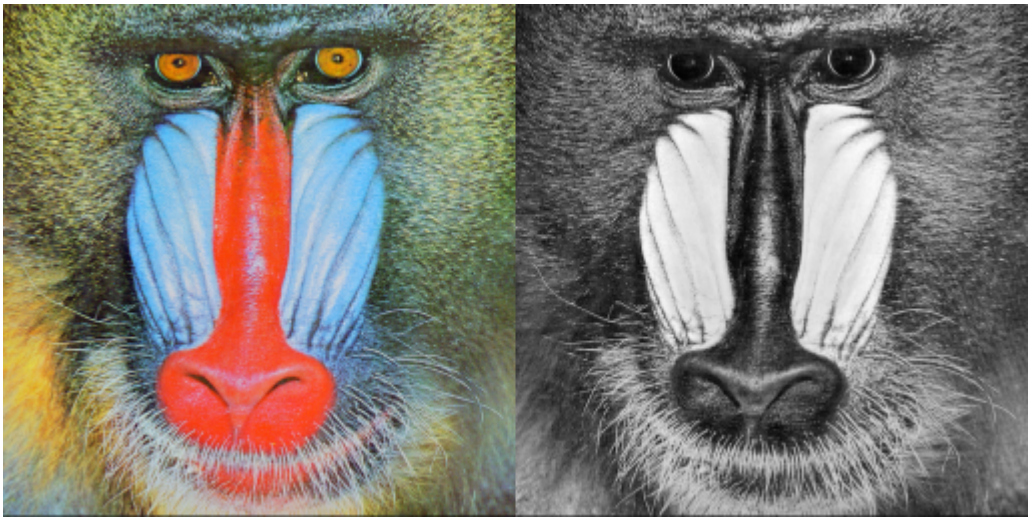
In [ ]:
```
# Capa verde
get_concat_h(baboon, green)
```

Out[ ]:



In [ ]:
```
# Capa azul
get_concat_h(baboon, blue)
```

Out[ ]:



Kernels (Filtros)

Los kernels realizan transformaciones para diferentes propósitos.

In [ ]:
```
from PIL import ImageFilter
```

Blur

In [ ]:
```
baboon.filter(ImageFilter.BLUR)
```

Out[ ]:

Boxblur

In [ ]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
for i in range(6):
    plt.subplot(2,3,i+1)
    plt.imshow(baboon.filter(ImageFilter.BoxBlur(i)))
    plt.axis("off")
plt.show()
```

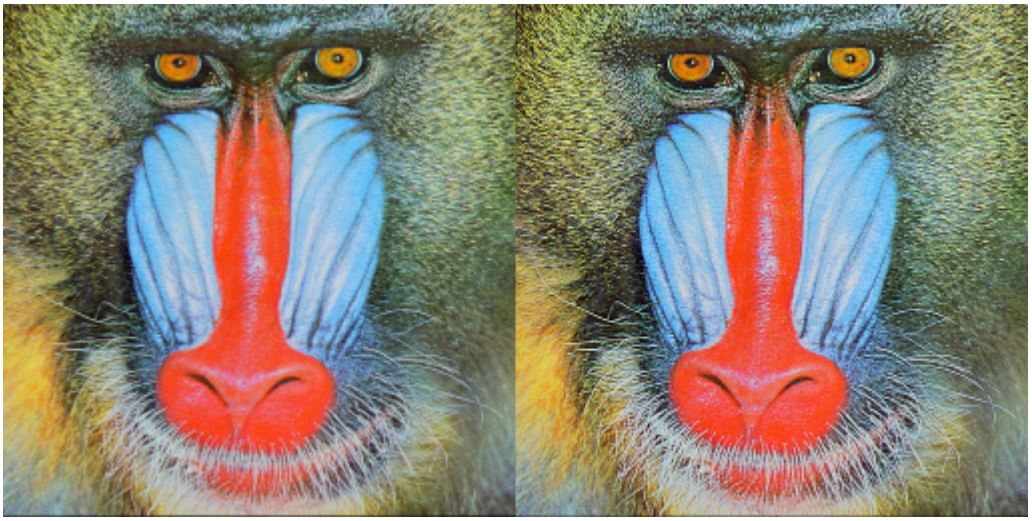
Gaussian Blur

In [ ]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(15, 10))
for i in range(6):
    plt.subplot(2, 3, i+1)
    plt.imshow(baboon.filter(ImageFilter.GaussianBlur(i)))
    plt.axis("off")
plt.show()
```

Sharpen

In [ ]:
```
get_concat_h(baboon,baboon.filter(ImageFilter.SHARPEN))
```

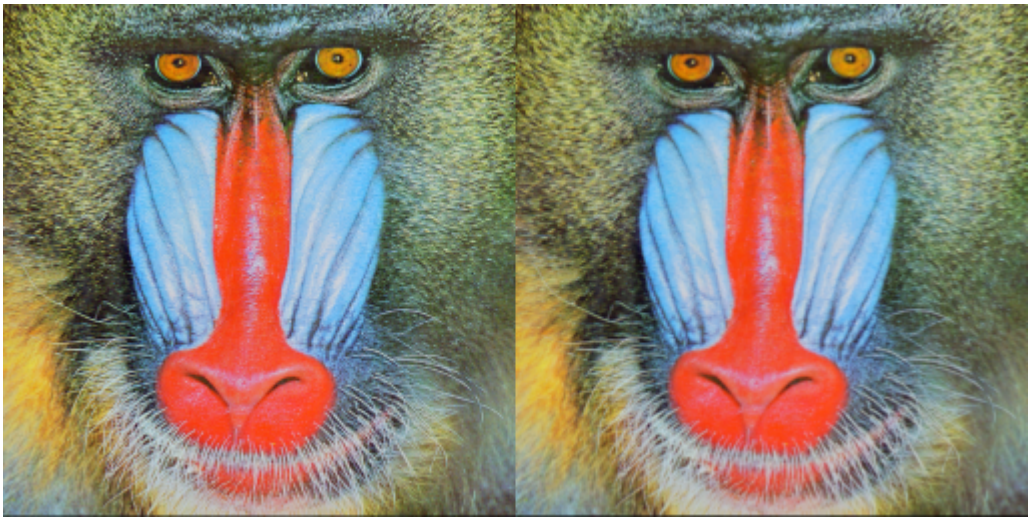
Out[ ]:



Smooth

In [ ]:
```
get_concat_h(baboon, baboon.filter(ImageFilter.SMOOTH))
```

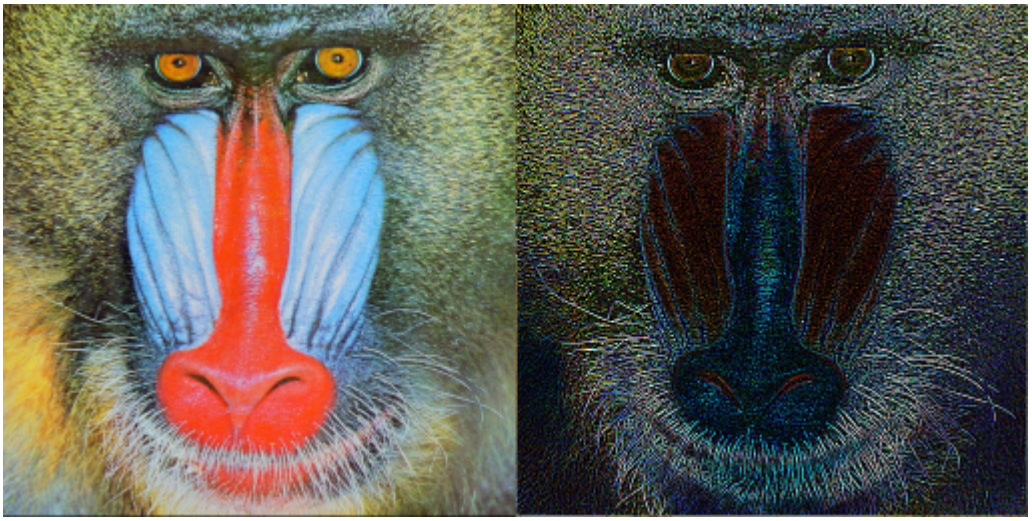
Out[ ]:



Find edges

In [ ]:
```
get_concat_h(baboon, baboon.filter(ImageFilter.FIND_EDGES))
```

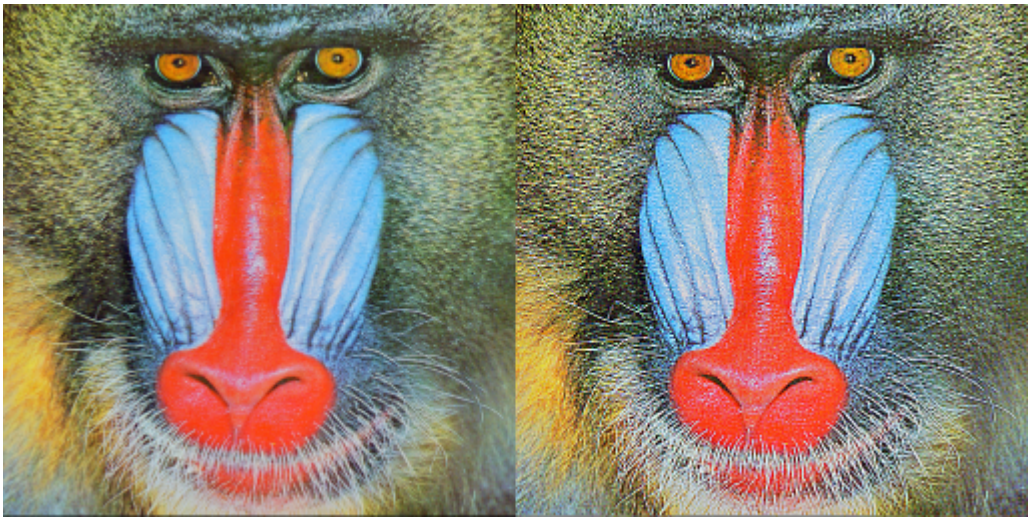
Out[ ]:



Edge Enhance

In [ ]:
```
get_concat_h(baboon, baboon.filter(ImageFilter.EDGE_ENHANCE))
```

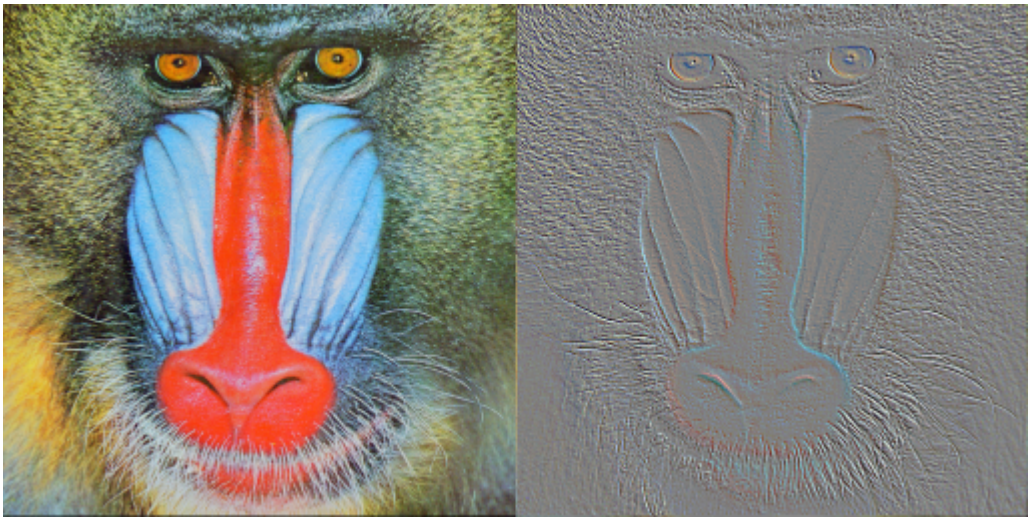
Out[ ]:



Edge Emboss

In [ ]:
```
get_concat_h(baboon, baboon.filter(ImageFilter.EMBOSS))
```

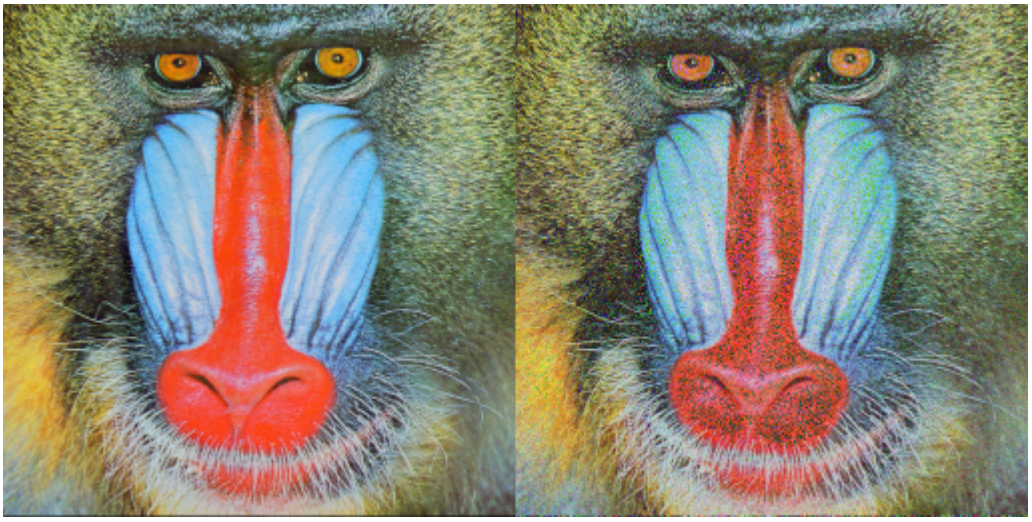
Out[ ]:



Añadiendo ruido a imagen

In [ ]:
```
# Obtenemos las filas y columnas
rows, cols = baboon.size

# Crea valores usando la distribucion normal con la media 0 y desviacion estandar de 2
# los valores son convertidos a unit8 lo que significa que se encuenran entre 0 y 255
noise = np.random.normal(0, 20, (rows, cols, 3)).astype(np.uint8)

# Añade ruido a la imagen
noisy_image = baboon + noise

# Crea una imagen PIL de un arreglo
noisy_image = Image.fromarray(noisy_image)

# Imprime la imagen original y la imagen con ruido
get_concat_h(baboon,noisy_image)
```

Out[ ]:



Kernel personalizado

In [ ]:
```
# Kernel comun para sharpening
kernel = np.array([[-1, -1, -1],
                   [-1, 9, -1],
                   [-1, -1, -1]])
kernel = ImageFilter.Kernel((3, 3), kernel.flatten())

# Aplica el filtro sharpening usando el kernelen la imagen original
sharpened = noisy_image.filter(kernel)

# Plots the sharpened image and the original image without noise
get_concat_h(baboon,sharpened)
```

Out[ ]:

Image segmentation

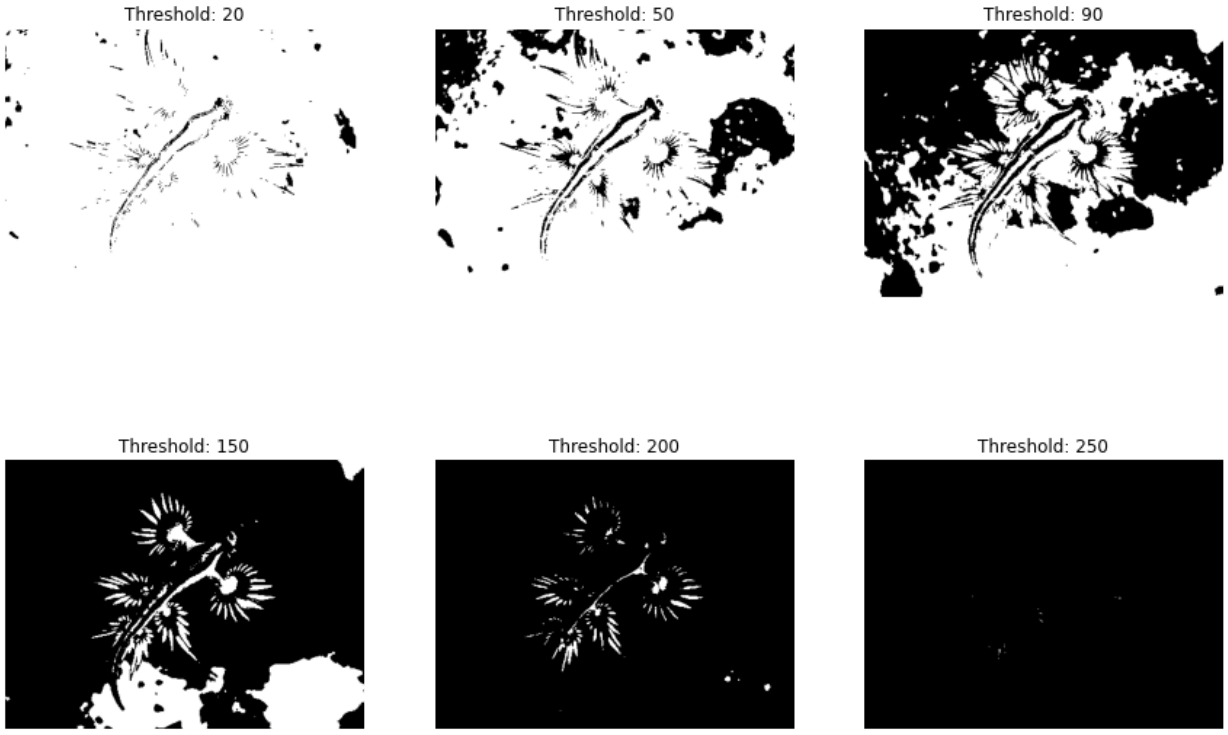
In [ ]:
```
animal = Image.open(r"images\animals.jpg")
animal.load()
animal
```

Out[ ]:

Filtro Thresold

Iterador donde varia el umbral para obtener el umbral optimo.

In [ ]:
```
threshold_list = [20,50,90,150,200,250]
counter = 1
plt.figure(figsize=(15, 10))
for threshold in threshold_list:
    plt.subplot(2, 3, counter)
    plt.title("Threshold: {}".format(threshold))
    plt.imshow(animal.convert("L").point(
        lambda x: 255 if x > threshold else 0),
               cmap="gray")
    plt.axis("off")
    counter += 1
plt.show()
```

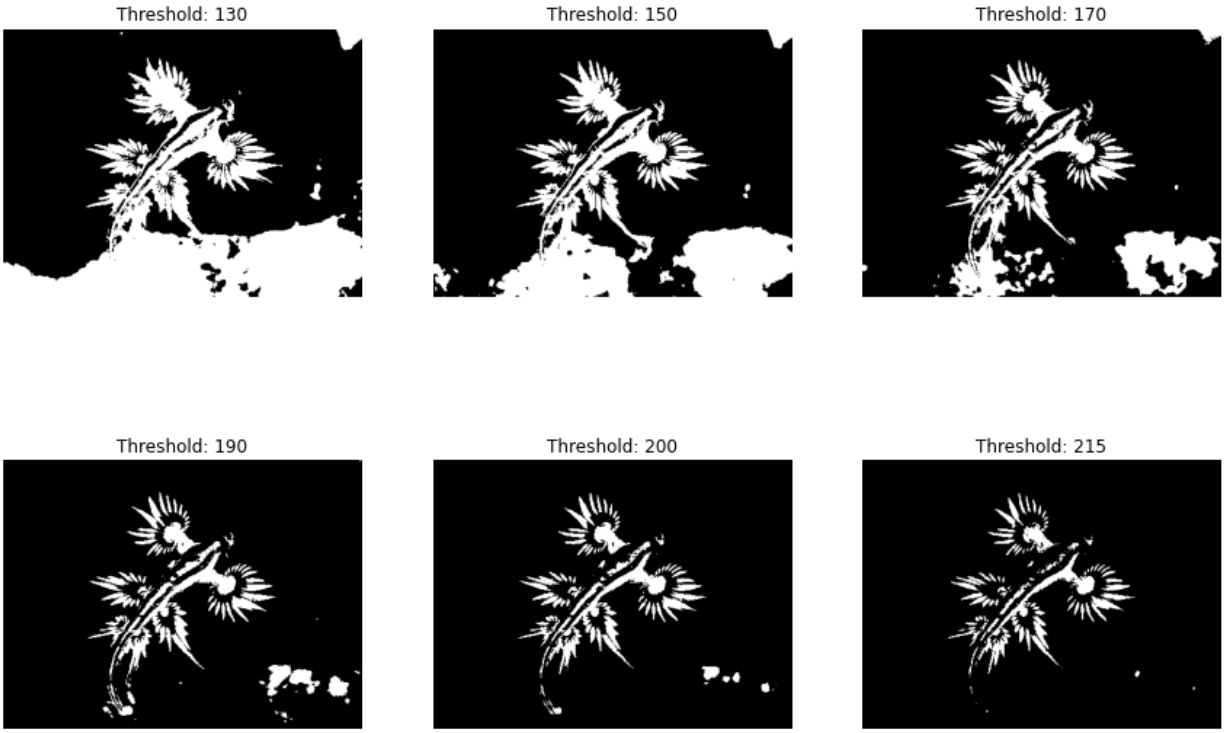


Capas RGB

Capas de la imagen, la capa azul muestra una mayor intensidad sobre el animal.

In [ ]:
```
red, green, blue = animal.split()
counter = 1

plt.figure(figsize=(15, 10))
for layer in [red, green, blue]:
    plt.subplot(1, 3, counter)
    plt.imshow(layer, cmap="gray")
    plt.axis("off")
    counter += 1
plt.show()
```

In [ ]:
```
# Se itera el umbral sobre la capa azul de la imagen importada
threshold_list = [130, 150, 170, 190, 200, 215]
counter = 1

plt.figure(figsize=(15, 10))
for threshold in threshold_list:
    plt.subplot(2, 3, counter)
    plt.title("Threshold: {}".format(threshold))
    plt.imshow(blue.point(
        lambda x: 255 if x > threshold else 0), cmap="gray")
    plt.axis("off")
    counter += 1
plt.show()
```



Erosion and expansion

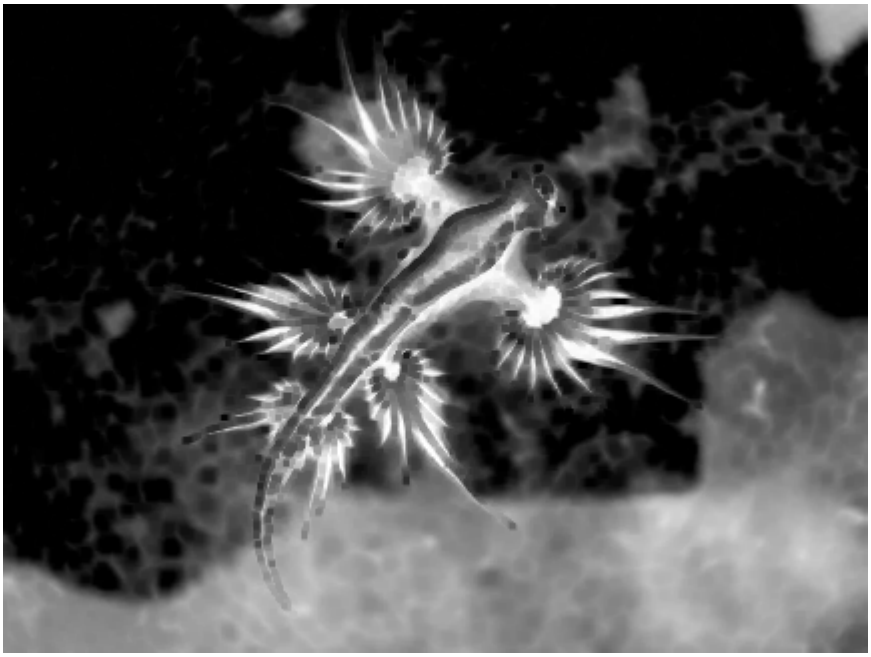
Estos filtros pueden utilizarse durante la segmentación de imagen, pues permite quitar porciones de imagen que no necesitemos.

MinFilter

Sustituye el valor minimo de dentro de la matriz de la dimension deseada.

In [ ]:
```
blue.filter(ImageFilter.MinFilter(3))
```

Out[ ]:



MaxFilter

Sustituye el valor maximo de dentro de la matriz de la dimension deseada.

In [ ]:
```
blue.filter(ImageFilter.MaxFilter(5))
```

Out[ ]:

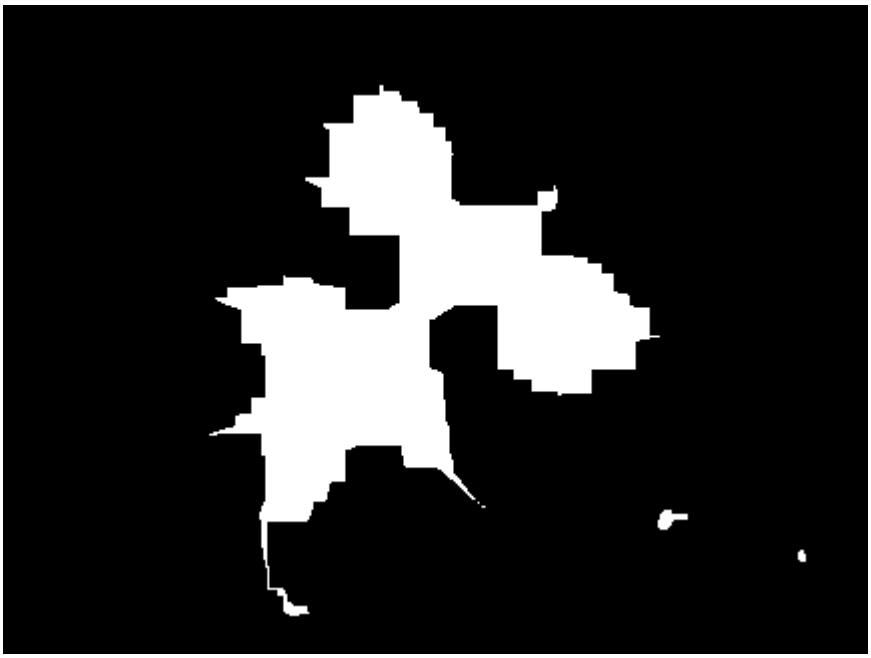
In [ ]:
```
# Funciones para aplicar de manera consecuente los filtros mencionados

def erode(cycles:int, image):
    """La funcion erosion disminuye la intensidad de manchas aisladas.

    Args:
        cycles (int): Ciclos de erosion que se aplicaran.
        image (PIL.Image): Imagen a la que se le aplicara la erosion.
```

```
    Returns:
        PIL.Image: Retorna una imagen con los ciclos aplicados.
    """
    for _ in range(cycles):
        image = image.filter(ImageFilter.MinFilter(3))
    return image

def dilate(cycles:int, image):
    """La funcion dilatacion expande la intensidad de manchas.

    Args:
        cycles (int): Ciclos de erosion que se aplicaran.
        image (PIL.Image): Imagen a la que se le aplicara la dilatacion.

    Returns:
        PIL.Image: Retorna una imagen con los ciclos aplicados.
    """
    for _ in range(cycles):
        image = image.filter(ImageFilter.MaxFilter(3))
    return image
```

Segmentacion con Thresold, Erosion y Expansion

In [ ]:
```
animal_steps = blue.point(lambda x: 255 if x > 210 else 0)
animal_steps = dilate(10,animal_steps)
animal_steps = erode(10,animal_steps)
animal_mask = animal_steps
animal_mask
```

Out[ ]:



In [ ]:
```
animal_mask = animal_mask.filter(ImageFilter.BoxBlur(5))
animal_mask
```

Out[ ]:

Imagen segmentada

In [ ]:
```
blank = animal.point(lambda _: 0)
animal_segment = Image.composite(animal, blank, animal_mask)
animal_segment
```

Out[ ]:

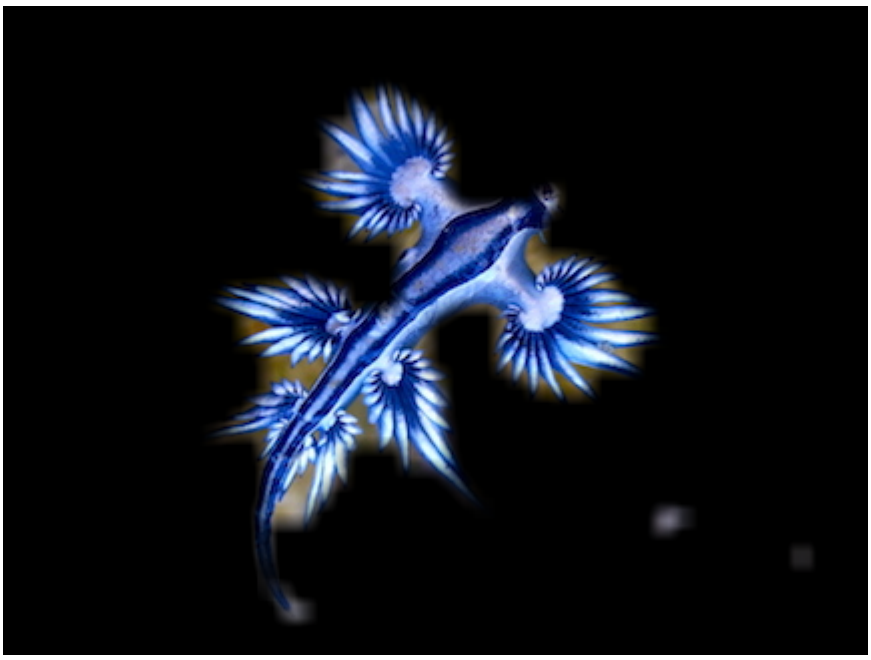


Imagen de fondo

In [ ]:
```
background = Image.open(r"images\background.jpg")
to_paste = animal_segment.reduce(4)
to_mask = animal_mask.reduce(4)

print(animal.size)
print(to_paste.size)
print(to_mask.size)
```

```
(432, 324)
(108, 81)
(108, 81)
```

Fondo e imagen segmentada

In [ ]:
```
import numpy as np
from PIL import ImageDraw

n = 50
x_list = np.random.randint(background.width,size=n)
y_list = np.random.randint(background.height,size=n)

draw = ImageDraw.Draw(background)
draw.text((10,10),"Juan de Jesús Aguilar Solano")

for x,y in zip(x_list,y_list):
    background.paste(to_paste,(x,y),to_mask)
background
```

Out[ ]:

Marca de agua

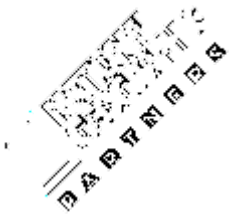
In [ ]:
```
logo = Image.open(r"images\nat_logo.png")
logo
```

Out[ ]:

Filtro Contour

In [ ]:
```
logo = logo.reduce(10)

logo = logo.filter(ImageFilter.CONTOUR)
logo = logo.point(lambda x: 0 if x == 255 else 255)
logo = logo.rotate(45, expand=True)
logo
```

Out[ ]:



In [ ]:
```
for x in range(0,background.width,logo.width):
    for y in range(0,background.height,logo.height):
        background.paste(logo,(x,y),logo)
background
```

Out[ ]: