🤖PERCEPCIÓN COMPUTACIONAL👁

Inteligencia Artificial

Jesús Andrés Infante Páez

1. Percepción de Audio 🎤

Para percepción de audio se utilizarán las librerias SpeechRecognition y pyaudio que permiten la conversión de audio a texto, ejemplo presente a continuación.

👉 Para usar las librerias se deben en primer instancia instalar (pip install ------) para posteriormente hacer la importación en el código.

In [1]:
```
pip install SpeechRecognition
```

```
Requirement already satisfied: SpeechRecognition in c:\programdata\anaconda3\lib\site-packages (3.9.0)
Requirement already satisfied: requests>=2.26.0 in c:\programdata\anaconda3\lib\site-packages (from SpeechRecognition)
(2.27.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anaconda3\lib\site-packages (from requests>=2.26.0-
>SpeechRecognition) (2021.10.8)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\programdata\anaconda3\lib\site-packages (from requests>=
2.26.0->SpeechRecognition) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\programdata\anaconda3\lib\site-packages (from requests>=2.26.0->Speec
hRecognition) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\programdata\anaconda3\lib\site-packages (from requests>=2.2
6.0->SpeechRecognition) (1.26.9)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
pip install pyaudio
```

```
Requirement already satisfied: pyaudio in c:\programdata\anaconda3\lib\site-packages (0.2.13)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
#Se importa y asigna un nombre a la libreria SpeechRecognition
import speech_recognition as sr
```

In [4]:
```
#Se importa y asigna un nombre a la libreria pyaudio
import pyaudio as pa
```

👉 El siguiente código graba el audio desde el micrófono, utiliza el servicio web de reconocimiento de voz de Google para transcribir el audio en texto y finalmente imprime el texto transcritó en la consola.

In [6]:
```
#Se crea una instancia del objeto Recognizer de speech_recognition
r = sr.Recognizer()
# Se utiliza el micrófono del sistema como fuente de entrada de audio se usa Microphone()
with sr.Microphone() as source:
    print('Esperando comando: ')
    audio = r.listen(source,2,6)  #r.listen(FuenteEntradaAudio, Timeout, TiempoMaximoDeEspera)
    try:
        text = r.recognize_google(audio, language='es-ES') #Transcripcion de audio en texto
        print('Tu dijiste: {}'.format(text))
    except:
        print('Perdon, no pude oirte')
```

```
Esperando comando:
result2:
{   'alternative': [   {   'confidence': 0.94797289,
                           'transcript': 'grabando audio Jesús Andrés Infante'},
                       {'transcript': 'grabando audio Jesús Andrés infa'}],
    'final': True}
Tu dijiste: grabando audio Jesús Andrés Infante
```

2. Procesamiento de imagen 🖼🎨

Se instala la biblioteca scikit-image esta es una biblioteca de procesamiento de imágenes para Python que proporciona herramientas para la manipulación, análisis y procesamiento de imágenes.

👉 Posteriormente se importan de scikit-image los submodulos io y color. El submódulo io proporciona funciones para leer y escribir imágenes en diferentes formatos, como JPEG, PNG y TIFF. También proporciona herramientas para trabajar con imágenes almacenadas en memoria como matrices NumPy. El submódulo color proporciona funciones para convertir imágenes entre diferentes espacios de

color, como RGB, escala de grises, YUV y HSV. También proporciona herramientas para ajustar el balance de blancos y la corrección de color de las imágenes.

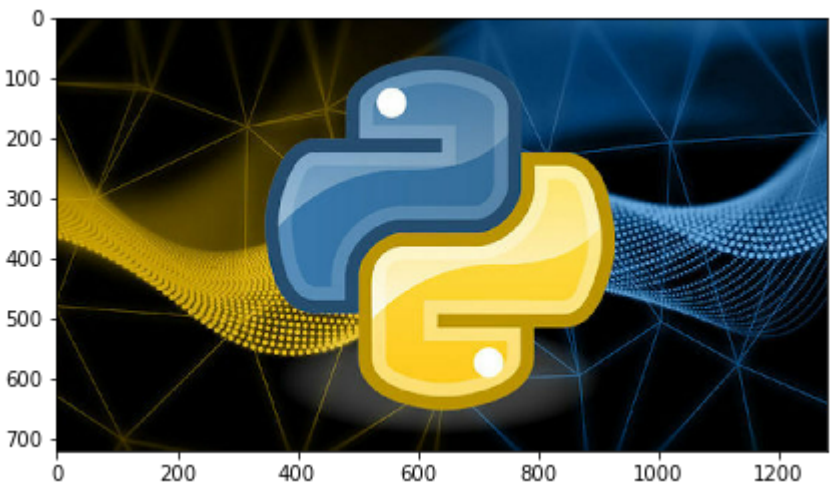
In [7]:
```
pip install scikit-image
```

```
Requirement already satisfied: scikit-image in c:\programdata\anaconda3\lib\site-packages (0.19.2)Note: you may need to
restart the kernel to use updated packages.

Requirement already satisfied: scipy>=1.4.1 in c:\programdata\anaconda3\lib\site-packages (from scikit-image) (1.7.3)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-image) (21.3)
Requirement already satisfied: PyWavelets>=1.1.1 in c:\programdata\anaconda3\lib\site-packages (from scikit-image) (1.
3.0)
Requirement already satisfied: tifffile>=2019.7.26 in c:\programdata\anaconda3\lib\site-packages (from scikit-image) (2
021.7.2)
Requirement already satisfied: networkx>=2.2 in c:\programdata\anaconda3\lib\site-packages (from scikit-image) (2.7.1)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in c:\programdata\anaconda3\lib\site-packages (fro
m scikit-image) (9.0.1)
Requirement already satisfied: numpy>=1.17.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-image) (1.21.5)
Requirement already satisfied: imageio>=2.4.1 in c:\programdata\anaconda3\lib\site-packages (from scikit-image) (2.9.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\programdata\anaconda3\lib\site-packages (from packaging>=
20.0->scikit-image) (3.0.4)
```

In [8]:
```
#importacion de los submódulos io y color
from skimage import io, color
```

In [9]:
```
img = io.imread('imagen.jpg') #Se lee una imagen desde un archivo
io.imshow(img) #Para mostrar la imagen
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x1e1da2c3610>
```



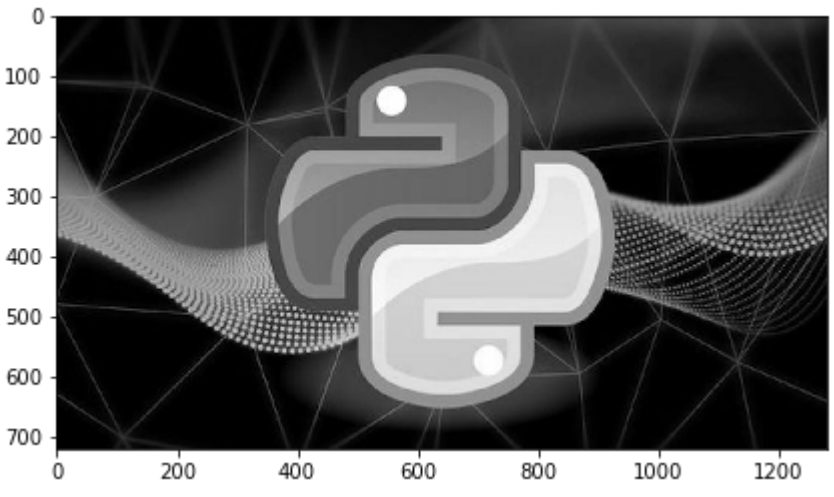
In [10]:
```
img.shape  #Al imprimir (y,x,tercer digito cantidad de canales)
```

Out[10]:
```
(720, 1280, 3)
```

2.1 Escala de grises 🔲

👉 Se utiliza la función rgb2gray del submódulo color de la biblioteca scikit-image para convertir una imagen en color en una imagen en escala de grises

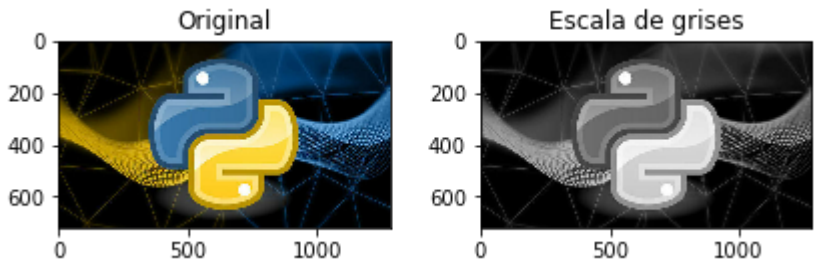
In [11]:
```
# El argumento img[...,0:3] se utiliza para extraer los canales de color rojo, verde y azul de la imagen.
img_gris=color.rgb2gray(img[...,0:3])
io.imshow(img_gris) #Se puede posteriormente usar io.show() para esconder el AxesImage
io.show()
```



👉 Se importa el módulo pyplot de la biblioteca matplotlib y lo asigna al alias plt, esto nos serve para ajustar las visualizaciones de las imagenes, posteriormente en una sola impresión se muestra la imagen original vs la de escala de grises

In [12]:
```
import matplotlib.pyplot as plt
```

In [13]:
```
# Se crea una figura con dos subgráficos para cada imagen
plt.subplot(221) #subplot(matriz 2x2 en el indice 1)
plt.title('Original')
io.imshow(img)
plt.subplot(222) #subplot(matriz 2x2 en el indice 2)
plt.title('Escala de grises')
io.imshow(img_gris)
io.show()
```



👉 A continuación se muestra en la terminal algunos de los métodos que permiten reconocer propiedades de la imagen.

In [14]:
```
print(type(img)) #tipo
print(img.shape) #forma
print(img.shape[0]) #ancho
print(img.shape[1]) #Altura
print(img.shape[2]) #canal de imagen
print(img.size) #total px
print(img.max()) #max de px
print(img.min()) #min de px
print(img.mean()) #promedio px
```

```
<class 'numpy.ndarray'>
(720, 1280, 3)
720
1280
3
2764800
255
0
61.25415617766204
```

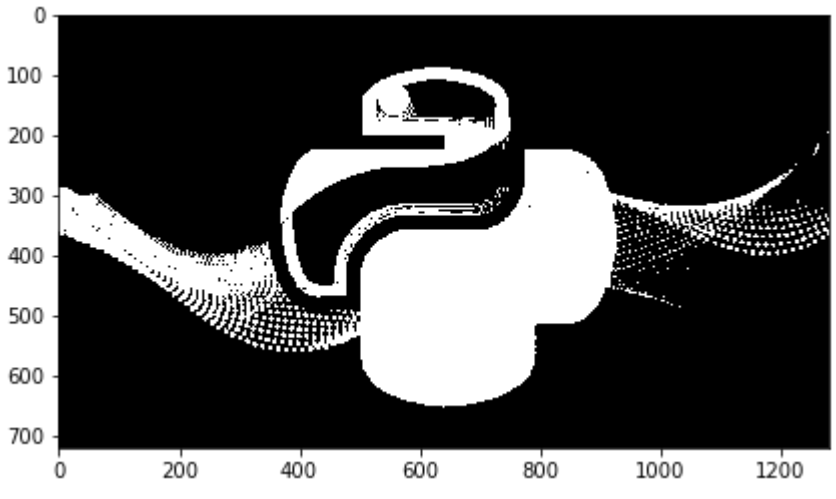
In [15]:
```
# Se accede a cualquier pixel y se modifica, (con una iteracion se puede recorrer todo)
pixel=img[400,749,2]
print(pixel)
```

```
169
```

2.2 Binarización de imagen ◻◼

👉 Se aplica nuevamente una escala de grises a la imagen para posterirmente acceder a cada pixel desde el recorrido de filas y columnas, dependiendo si es mayor o menor a 0.5 se asignan valores (0 o 1) para cambiar el pixel y binarizar la imagen.

In [16]:
```
img_gray = color.rgb2gray(img) #Conversión de imagen a grises
rows,cols = img_gray.shape #dimensiones de la imagen

#recorrido y binarización
for i in range(rows):
    for j in range(cols):
        if (img_gray[i,j] <= 0.5):
            img_gray[i,j]= 0.0
        else:
            img_gray[i,j]= 1.0

io.imshow(img_gray)
io.show()
```



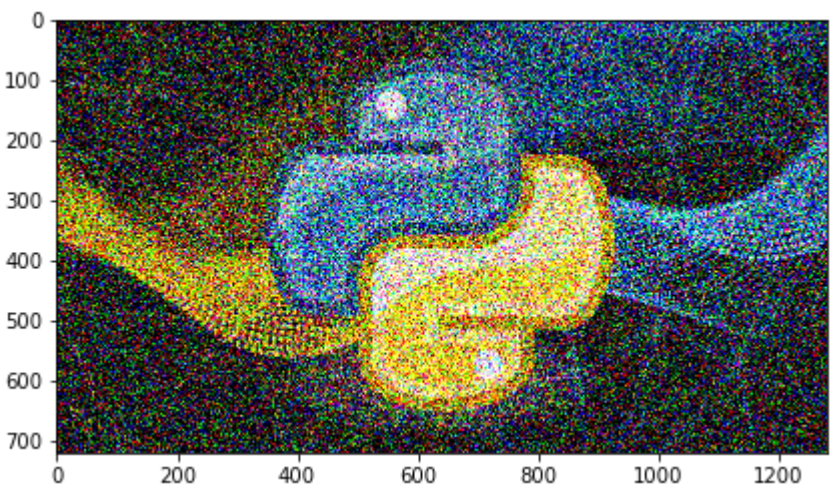
2.3 Ruido Gaussiano 🔊

👉 Se importa la libreria skimage para posteriormente hacer uso del submodulo util y random_noise o ruido gaussiano que es un tipo de ruido que se distribuye normalmente y puede ser generado mediante una distribución de probabilidad gaussiana.

In [17]:
```
import skimage
```

In [18]:
```
#Ruido Gaussiano
#... .random_noise(imgen original, tipo de ruido a aplicar, varianza del ruido gaussiano)()
noise=skimage.util.random_noise(img, mode='gaussian', var=0.15)
io.imshow(noise)
```

Out[18]:
```
<matplotlib.image.AxesImage at 0x1e1dcb67310>
```

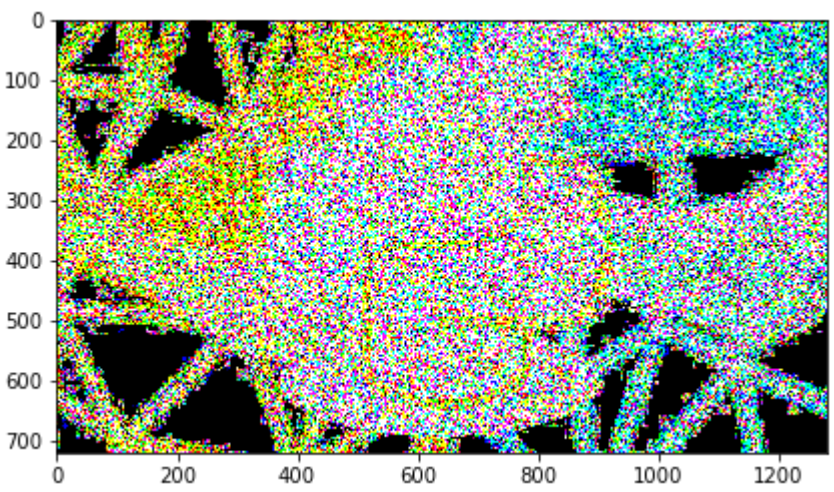


👉Se importa la librería numpy y ahora se agrega ruido gaussiano a la imagen utilizando la función np.random.normal de NumPy.

In [19]:
```
import numpy as np
```

In [20]:
```
mean = 0.5  #Media
var = 0.1 #Varianza

noise = np.random.normal (mean, var**0.05, img.shape)
#Se multiplica la imagen original por la matriz de ruido noise para agregar el ruido a la imagen
noisy = img * noise
#se limitan los valores de la matriz resultante noisy entre 0 y 1
noisy = np.clip(noisy,0.0,1.0)
io.imshow(noisy)
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x1e1dcbd1c10>
```



3. Scipy 👀

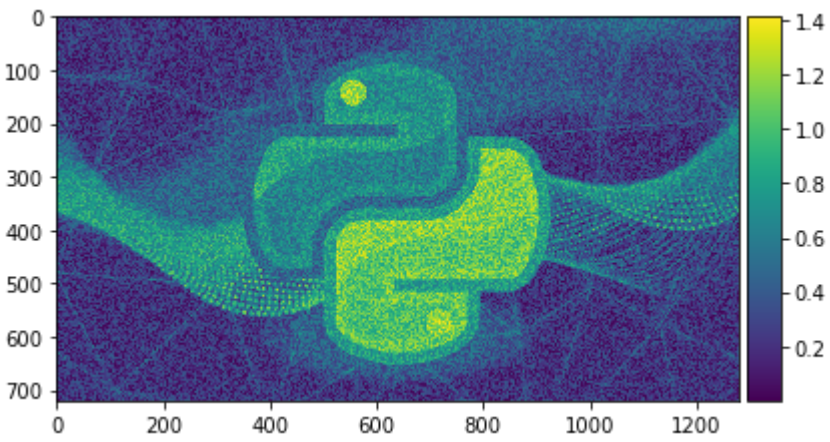
Procesesamiento de imagenes a partir de pixeles

👉 misc es un submódulo de scipy que contiene diversas funciones para trabajar con imágenes, como por ejemplo para leer y escribir imágenes, operaciones de manipulación de imágenes y transformaciones geométricas. ndimage es otro submódulo de scipy que proporciona una gran cantidad de funciones para operaciones de procesamiento de imágenes, como filtrado, segmentación, morfología matemática, entre otras.

In [21]:
```
#Importando submodulos de la librería scipy
from scipy import misc
from scipy import ndimage
```

In [24]:
```
#Procesamiento de imagen, genera ruido en la imagen a partir de una escala de grises
#1.5*img_gris.std() controla la amplitud del ruido
noisy = img_gris + 1.5*img_gris.std()*np.random.random(img_gris.shape)
io.imshow(noisy)
```

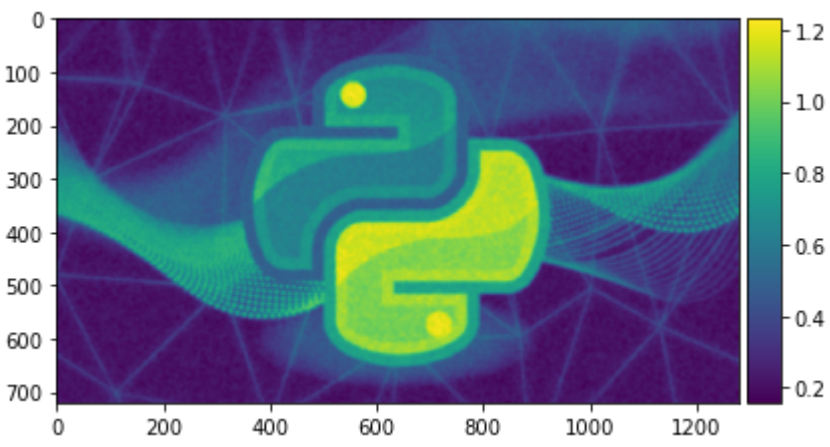
Out[24]:
```
<matplotlib.image.AxesImage at 0x1e1da69f2b0>
```



👉 En la línea gauss_denoised = ndimage.gaussian_filter(noisy, 2.8), se está aplicando un filtro gaussiano a la imagen noisy con un valor de desviación estándar de 2.8. Esto suaviza la imagen y reduce el ruido generado anteriormente.

In [25]:
```
#Reduciendo Ruido
gauss_denoised = ndimage.gaussian_filter(noisy, 2.8)
io.imshow(gauss_denoised)
```

Out[25]:
```
<matplotlib.image.AxesImage at 0x1e1da7011f0>
```



👉 Filtros a la imagen en escala de grises

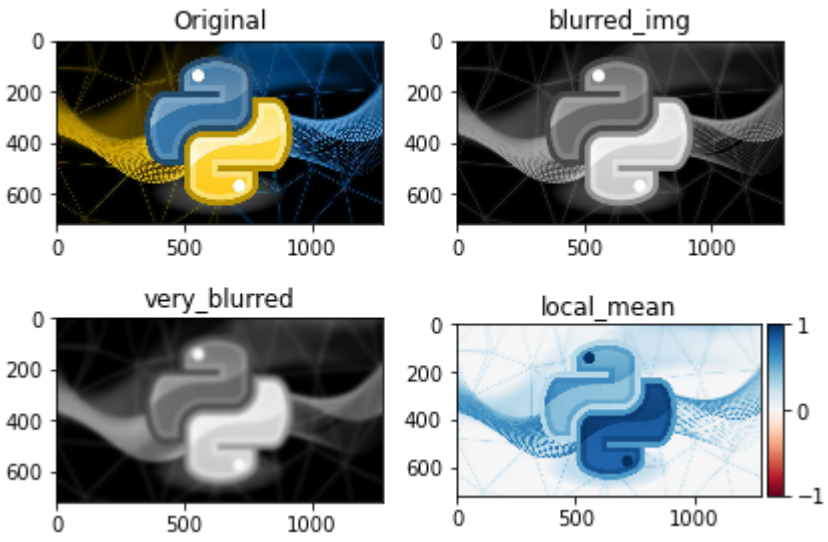
In [26]:
```
#aplicando un filtro gaussiano con un valor de desviación estándar de 2. Esto suaviza la imagen y reduce el ruido.
blurred_img = ndimage.gaussian_filter(img_gris, sigma=2)
#filtro gaussiano con un valor de desviación estándar de 8. Esto suaviza aún más la imagen y reduce más el ruido.
very_blurred = ndimage.gaussian_filter(img_gris, sigma=8)
#se está aplicando un filtro uniforme con un tamaño de ventana de 2x2 píxeles.
#Cada píxel se reemplaza por el promedio de los píxeles de su vecindario de 2x2
local_mean = ndimage.uniform_filter(img_gris, size=2)
```

👉 Importando la libreria matplotlib y el submodulo pyplot cada imagen se muestra en una subfigura separada en el mismo espacio de visualización.

In [27]:
```
import matplotlib.pyplot as plt
```

In [28]:
```
# 2x2, posicion 1
plt.subplot(221)
plt.title('Original')
io.imshow(img)

# 2x2, posicion 2
plt.subplot(222)
plt.title('blurred_img')
io.imshow(blurred_img)

# 2x2, posicion 2
plt.subplot(223)
plt.title('very_blurred')
io.imshow(very_blurred)

# 2x2, posicion 4
plt.subplot(224)
plt.title('local_mean')
```

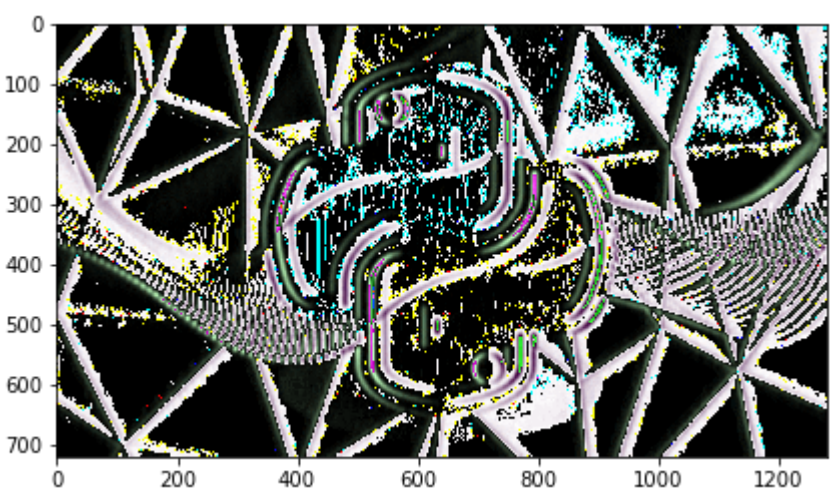
```
io.imshow(local_mean)
io.show()
```



👉 El siguiente código utiliza una técnica de procesamiento de imágenes para detectar bordes en una imagen mediante la diferenciación de la intensidad de los píxeles.

In [29]:
```
#Aplica un filtro gaussiano a la imagen para suavizarla.
im = ndimage.gaussian_filter(img, 3.5)
#Sobel es un operador de gradiente utilizado para detectar bordes en imágenes.
#Se aplica sobel en direcciones horizontal (axis=0) y vertical (axis=1) a la imagen suavizada
sx = ndimage.sobel(im, axis=0, mode='constant')
sy = ndimage.sobel(im, axis=1, mode='constant')
#se calcula la magnitud de los gradientes obtenidos en cada dirección
sob = np.hypot(sx, sy)

io.imshow(sy)
```

Out[29]:
```
<matplotlib.image.AxesImage at 0x1e1da95cd90>
```

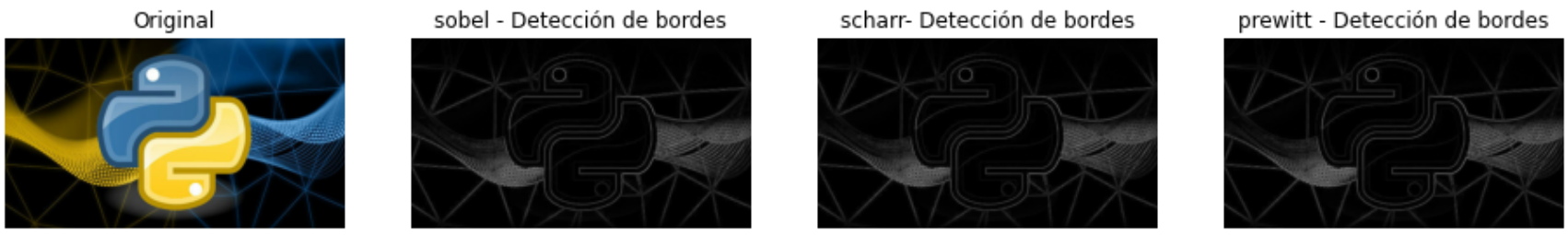


4. Detección de bordes 🎨

👉 El siguiente código utiliza el submodulo filters de la librería Scikit-image para realizar la detección de bordes en una imagen en escala de grises. Se aplican diferentes filtros de detección de bordes (Sobel, Scharr y Prewitt)

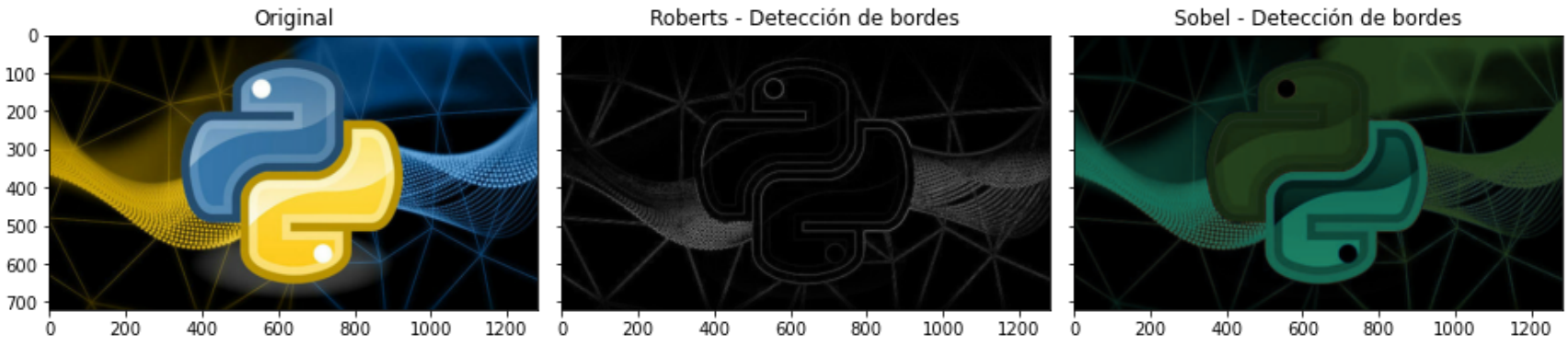
In [30]:
```
from skimage import filters

#creación de tres variables donde se llaman cada uno de los filtros a usar
filtros = [filters.sobel, filters.roberts, filters.prewitt]

#Procesamiento para la deteccion de bordes
edge_sobel = filters.sobel(img_gris)
edge_scharr = filters.scharr(img_gris)
edge_prewitt = filters.prewitt(img_gris)

#Se genera una figura con 3 subfiguras en una sola fila (ncols=3)
fig, axes = plt.subplots(ncols=4, sharex=True, sharey=True,
                         figsize=(16, 16))


#comparacion con escala de grises
axes[0].imshow(img)
axes[0].set_title('Original')

axes[1].imshow(edge_sobel, cmap=plt.cm.gray)
axes[1].set_title('sobel - Detección de bordes')

axes[2].imshow(edge_scharr, cmap=plt.cm.gray)
axes[2].set_title('scharr- Detección de bordes')
```

```
axes[3].imshow(edge_prewitt, cmap=plt.cm.gray)
axes[3].set_title('prewitt - Detección de bordes')

#no se mostrarán las etiquetas y los valores correspondientes de los ejes.
#Para activar dejar predeterminado o cambiar a on.
for ax in axes:
    ax.axis('off')
```



👉 El siguiente código muestra la detección de bordes utilizando los filtros de Roberts y Sobel en una imagen en escala de grises y la original. El resultado se muestra en una figura con dos subfiguras, una para cada filtro.

In [31]:
```
#calculo de los bordes de la imagen en escala de grises utilizando el filtro de detección de bordes Roberts.
edge_roberts = filters.roberts(img_gris)
#calculo de los bordes de la imagen original utilizando el filtro de detección de bordes Sobel.
edge_sobel = filters.sobel(img)

#Tres columnas en una sola visualización
fig, axes = plt.subplots(ncols=3, sharex=True, sharey=True,
                         figsize=(13,13))

axes[0].imshow(img)
axes[0].set_title('Original')

axes[1].imshow(edge_roberts, cmap=plt.cm.gray)
axes[1].set_title('Roberts - Detección de bordes')

axes[2].imshow(edge_sobel, cmap=plt.cm.gray)
axes[2].set_title('Sobel - Detección de bordes')

#no se mostrarán las etiquetas y los valores correspondientes de los ejes.
#Para activar dejar predeterminado o cambiar a on.
for ax in axes:
    ax.axis('on')

#Se ajusta automáticamente los espacios entre las subfiguras
plt.tight_layout()
plt.show()
```



Nota

👉 El archivo .ipynb con el código se encuentra en el siguiente enlace:
https://drive.google.com/file/d/1ZnGZPZkmyyEAyCCjEzhtUGjs2a1O260z/view?usp=sharing

---

¡FIN!

In [ ]: