FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES

CARRERA: COMPUTACIÓN **ASIGNATURA**: APRENDIZAJE MAQUINA

NRo. PRÁCTICA: 1 **TÍTULO PRÁCTICA**: Procesamiento de Imágenes con Numpy.

OBJETIVOS:

- Aprender a visualizar, generar, cargar y modificar imágenes en blanco y negro o color a través de las operaciones de matrices de la librería Numpy, y las facilidades de las librerías PyPlot y skimage.

INSTRUCCIONES (Detallar las instrucciones que se dará al estudiante):

1. Desarrollar cada sección propuesta en este documento, utilizando el formato para el estudiante.
2. Fecha de entrega en acuerdo con el docente. ***Es importante que su solución refleje todo el proceso que se debe seguir para obtener la respuesta (especifique adecuadamente la realización de su trabajo y bien argumentado).***
3. El puntaje final de la práctica guarda correspondencia con contenido en el sílabo de la asignatura (indicador de logro) Valor: 2 punto.
4. El indicador de logro a alcanzar es: *Documento con resultados de los comandos y procesos.*

ACTIVIDADES POR DESARROLLAR (Anotar las actividades que deberá seguir el estudiante para el cumplimiento de la práctica)

1. Instalación del Anaconda distribution.
2. Conectar con Jupiter Notebook.
3. Tratar imágenes con Numpy
4. Programar diferentes codigos de prueba
5. Presentar la solución sobre el mismo formato (subirlo al ambiente virtual) y bien expresado el proceso utilizado.
6. **Conocimientos previos:** Conceptos básicos de imágenes digitales color y escala de grises. Programación en Python con la librería Numpy.
7. **Importante**: Para que este cuaderno funcione correctamente, las imágenes 'lena.png' y 'lena_gris.png' deben encontrarse en el mismo directorio que el cuaderno. https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.misc.lena.html
8. **Links de Ayuda**:
   a. https://claudiovz.github.io/scipy-lecture-notes-ES/index.html
   b. https://www.unioviedo.es/galiano/master/lab01_introduccion_imagen_py.html

RESULTADO(S) OBTENIDO(S):

- Documento con resultados de los comandos y procesos.

CONCLUSIONES:

- Los practicantes tienen la habilidad de realizar una búsqueda de información utilizando métodos válidos, apropiados y efectivos.
- Los practicantes mejoran el conocimiento aprendido en base a la experimentación realizada identificando los aspectos claves para la adecuada resolución.

RECOMENDACIONES:

- Asistir a las jornadas de clases.
- Revisar detalladamente la información entregada por el docente
- Despejar la dudas que puedan surgir al momento de realizar la práctica.
- Extender tiempo para que los estudiantes puedan subir resultado al AVAC

***Docente / Técnico Docente*:** Ing. Miguel Quiroz., MSc.

***Firma*:**

Miguel Quiroz

ACTIVIDADES A DESARROLLAR POR SECCIONES

SECCIÓN 1

Creando matrices y visualizándolas como imágenes

Las imágenes se codifican como matrices. En particular, las imágenes de intensidad o escala de grises se codifican como una matriz de dos dimensiones, donde cada número representa la intensidad de un píxel.

Pero eso significa que cualquiera de estas matrices que generamos se puede visualizar como una matriz. Para visualizar imágenes, usamos el módulo `pyplot` de la librería `matplotlib`.

In [11]:
```
# importamos el modulo pyplot, y lo llamamos plt
import matplotlib.pyplot as plt

#configuracion necesaria de pyplot para ver las imagenes en escala de gri
ses
plt.rcParams['image.cmap'] = 'gray'


# comando de Jupyter para que las imagenes se muestren automaticamente
%matplotlib inline

#tambien importamos numpy ya que lo usamos para crear y manipular matrice
s
import numpy as np
```

La función que nos permite visualizar matrices es `imshow` del modulo `pyplot`, que invocaremos como `plt.imshow`, y recibe la imagen como parámetro.

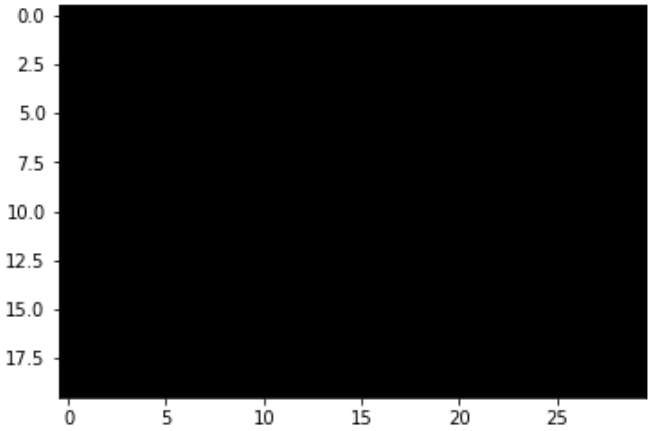
Debido a que pyplot intenta ajustar los colores automaticamente de una imagen, vamos a pasarle como parámetros también `vmin=0,vmax=1`. Si bien no los vimos antes, estos se llaman *keyword parameters* o *parámetros por palabra clave* de python.

In [12]:
```
#tamaño de las matrices a visualizar
size=(20,30)

# Una matriz de ceros.
imagen_negra = np.zeros(size)

#visualizamos la matriz
#Se ve como una imagen negra, ya que todos los elementos (pixeles) tienen
intensidad 0
plt.imshow(imagen_negra,vmin=0,vmax=1)
```

```
# (es necesario indicar vmin y vmax para que pyplot sepa que el minimo es
0 y el maximo 1)
# (solo imagenes escala de grises)
```

Out[12]:

```
<matplotlib.image.AxesImage at 0x7f246c670e10>
```



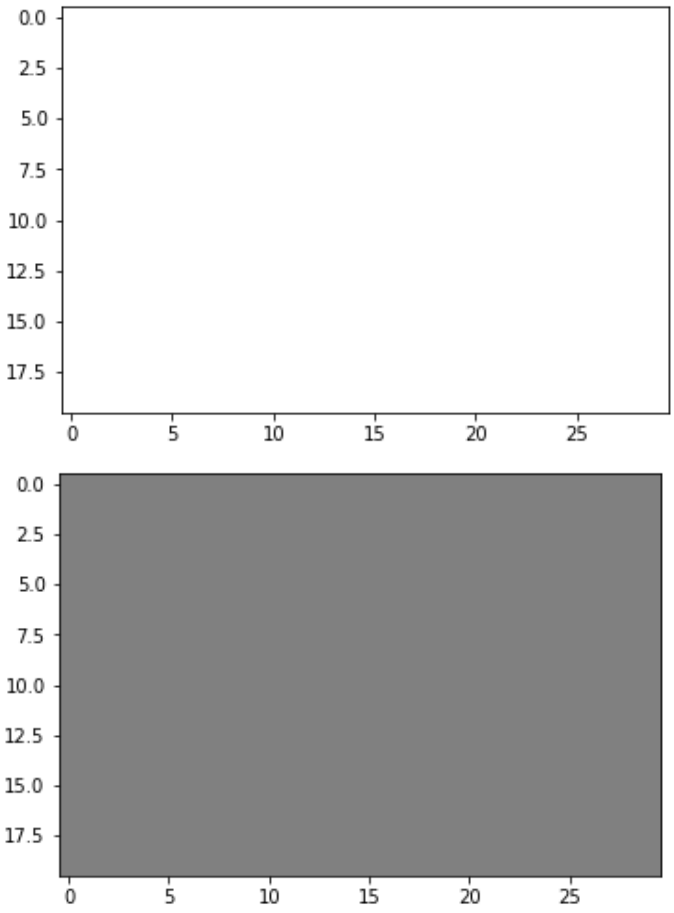
También podemos generar imágenes blancas, o grises. Si queremos mostrar más de una imagen en una celda, vamos a tener que ejecutar `plt.figure()` para crear la figura que contenga la imagen.

In [13]:

```
# IMAGEN BLANCA
# Una matriz de unos.
imagen_blanca = np.ones(size)

#visualizamos la matriz
#Se ve como una imagen blanca, ya que todos los elementos (pixeles) tiene
n intensidad 1
plt.imshow(imagen_blanca,vmin=0,vmax=1)

#creamos otra figura para mostrar la imagen (sino el proximo imshow sobre
escribe al anterior)
plt.figure()

# IMAGEN GRIS
# Una matriz con valor 0.5 en todos sus elementos
imagen_gris = np.ones(size)*0.5

#visualizamos la matriz
#Se ve como una imagen gris, ya que todos los elementos (pixeles) tienen
intensidad 0.5
plt.imshow(imagen_gris,vmin=0,vmax=1)
```

Out[13]:

```
<matplotlib.image.AxesImage at 0x7f24680ea9b0>
```



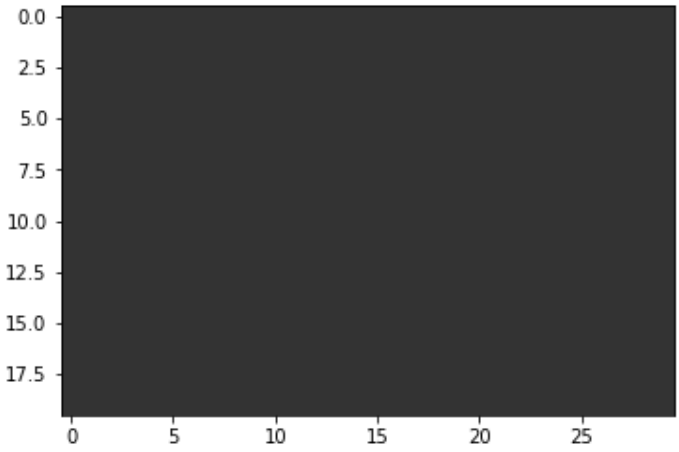
In [14]:

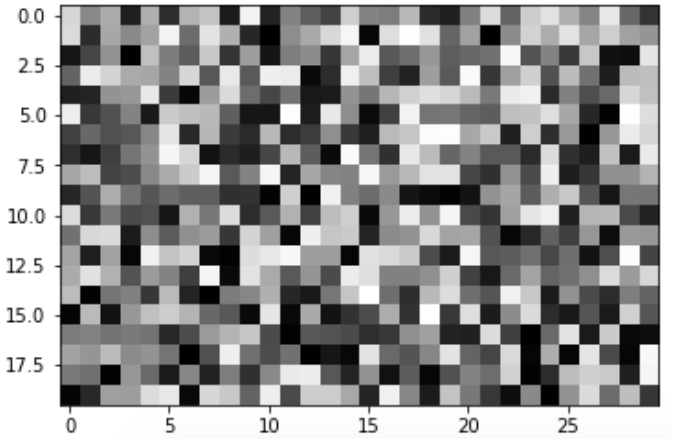
```
# IMAGEN GRIS OSCURO
# Una matriz con valor 0.2 en todos sus elementos
imagen_gris_oscuro = np.ones(size)*0.2

#visualizamos la matriz
#Se ve como una imagen gris, ya que todos los elementos (pixeles) tienen
intensidad 0.5
plt.imshow(imagen_gris_oscuro,vmin=0,vmax=1)

#creamos otra figura para mostrar la imagen (sino el proximo imshow sobre
escribe al anterior)
plt.figure()

# IMAGEN ALEATORIA
# Una matriz con valor aleatorio
```

```
imagen_aleatoria = np.random.rand(size[0],size[1])

#visualizamos la matriz
#Se ve como una imagen gris, ya que todos los elementos (pixeles) tienen
intensidad 0.5
plt.imshow(imagen_aleatoria,vmin=0,vmax=1)
```

Out[14]:

```
<matplotlib.image.AxesImage at 0x7f2468096828>
```





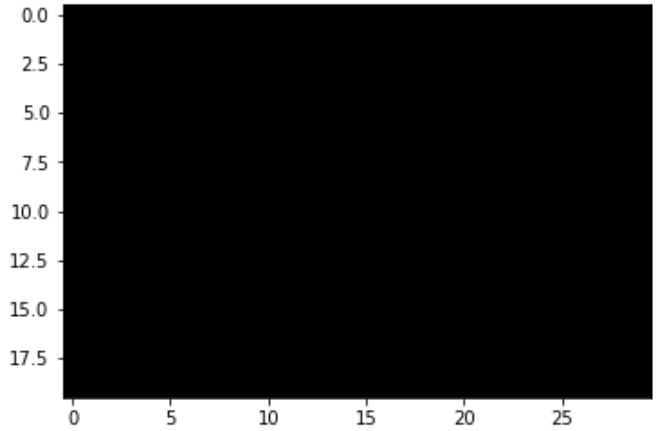
In [15]:

```
#Ejercicio
# Generar una imagen que sea toda blanca de la mitad para abajo
# Y que la mitad derecha de la parte de arriba sea gris
#       NG
#       BB

imagen_negra = np.zeros(size)

#IMPLEMENTAR
```

```
plt.imshow(imagen_negra)
```

Out[15]:

```
<matplotlib.image.AxesImage at 0x7f2463ff65f8>
```

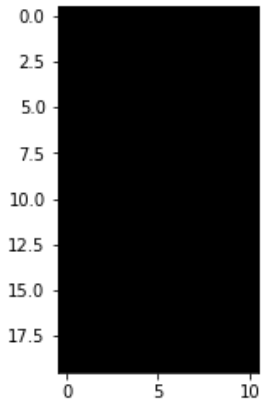


In [16]:

```
#Ejercicio para repasar (NO SE HARA EN EL TALLER)
# Modificar la variable imagen_gradiente para que cada fila tenga una int
ensidad creciente
#Resolver Problema 1
# La fila 0 debe tener intensidad 0
# La fila 1 debe tener intensidad 0.1
# La fila 1 debe tener intensidad 0.2
# ...
# La fila 9 debe tener intensidad 0.9
# La fila 10 debe tener intensidad 1

imagen_gradiente = np.zeros((20,11))

#IMPLEMENTAR

plt.imshow(imagen_gradiente)
```

Out[16]:

```
<matplotlib.image.AxesImage at 0x7f2463f5bdd8>
```



Lectura de imágenes

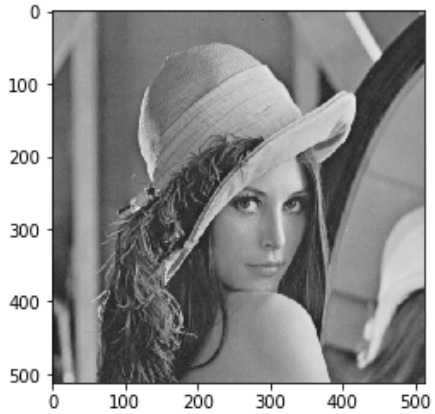
Para leer imágenes utilizaremos el módulo `io` de la librería `skimage`. En ese módulo tenemos la función `imread` para leer imágenes de un archivo(y también la función `imsave` para escribirlas a un archivo).

In [17]:

```
from skimage import io

image=io.imread("lena_gray.png")/255.0 # imread lee las imágenes con los
pixeles codificados como enteros
# en el rango 0-255. Por eso la convertimos a flotante y en el rango 0-1

print("- Dimensiones de la imagen:")
print(image.shape)
plt.imshow(image,vmin=0,vmax=1)
- Dimensiones de la imagen:
(512, 512)
```

Out[17]:

```
<matplotlib.image.AxesImage at 0x7f2463ec1208>
```



Esta es la imagen de Lena; es extremadamente conocida en el área de procesamiento de imágenes porque siempre se usa para evaluar nuevas técnicas y efectos.

Imágenes a color

También podemos cargar imágenes a color con `imread`.

In [18]:

```
lena_rgb=io.imread("lena.png")/255.0 # imread lee las imagenes con los pi
xeles codificados como enteros
# en el rango 0-255. Por eso la convertimos a flotante y en el rango 0-1
plt.imshow(lena_rgba) #no es necesario el vmin/vmax para imágenes a color

print("Dimensiones de la imagen:")
print(lena_rgba.shape)
```

```
Dimensiones de la imagen:
(512, 512, 3)
```

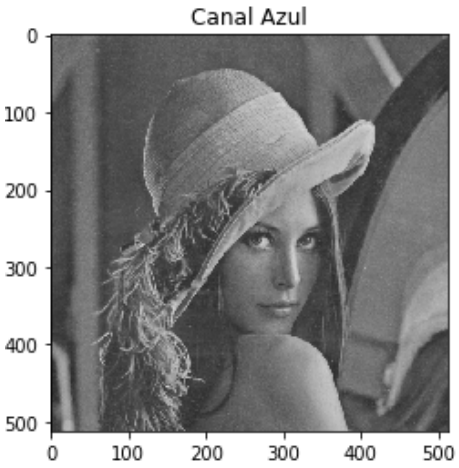
En este caso, tenemos tres dimensiones. Las primeras dos, de tamaño 512, corresponden a la cantidad de pixeles de la imagen. La última, de tamaño 3, contiene los tres canales R, G y B de la imagen.
Podemos ver cada canal individualmente, como si el canal fuera una imagen en escala de grises, para ver la intensidad de cada canal.

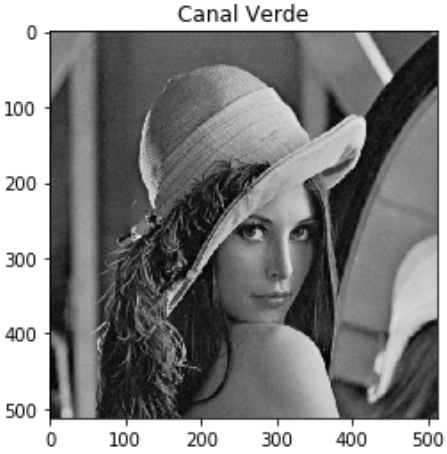
In [19]:

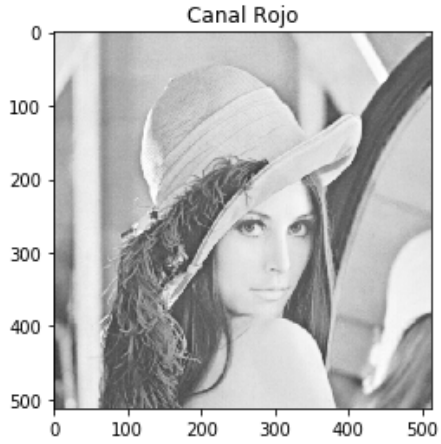
```
plt.imshow(lena_rgb[:,:,0],vmin=0,vmax=1)
plt.title("Canal Rojo")
plt.figure()
plt.imshow(lena_rgb[:,:,1],vmin=0,vmax=1)
plt.title("Canal Verde")
plt.figure()
plt.imshow(lena_rgb[:,:,2],vmin=0,vmax=1)
plt.title("Canal Azul")
```

Out[19]:

```
<matplotlib.text.Text at 0x7f2463efdba8>
```







Podemos ver que el canal rojo es el más activo, ya que la imagen contiene mucha piel y un fondo crema/anaranjado. Además, el canal verde es el que tiene más detalles de la imagen.

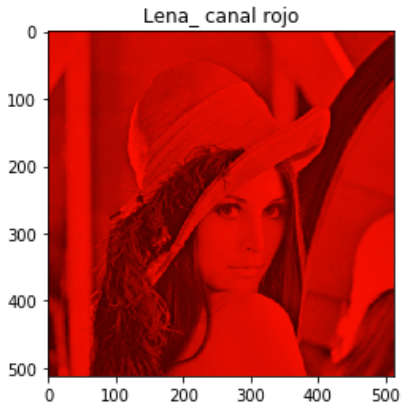
Otra manera de ver la información de cada canal es quitar la información de los otros canales. Por ejemplo, para ver el canal rojo, ponemos en 0 los canales azul y verde.

In [11]:

```
lena_red=np.copy(lena_rgb) # creo una copia de la imagen para preservar l
a original
lena_red[:,:,1]=0
lena_red[:,:,2]=0
plt.title("Lena_ canal rojo")
plt.imshow(lena_red)
```

Out[11]:

```
<matplotlib.image.AxesImage at 0x7fc53821c400>
```



Podemos observar que sin los canales azul y verde perdemos bastante información de color, aunque la silueta se mantiene.

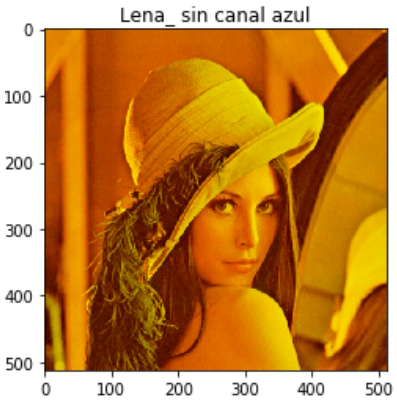
Ahora probemos quitando sólo un canal, por ejemplo, el azul:

In [12]:

```
lena_red_green=np.copy(lena_rgb) # creo una copia de la imagen para pres
rvar la original
lena_red_green[:,:,2]=0
plt.title("Lena_ sin canal azul")
plt.imshow(lena_red_green)
```

Out[12]:

```
<matplotlib.image.AxesImage at 0x7fc5381a3400>
```



De nuevo, se ha perdido información color, pero mucho menos, ya que justamente el canal azul era el menos importante.

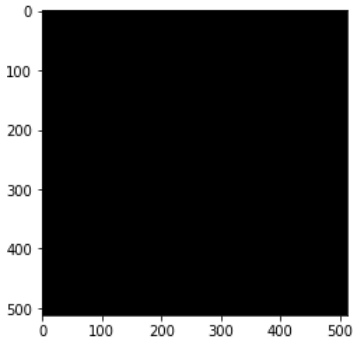
Conversión de color a escala de grises

La conversión de una imagen de color a escala de grises se puede hacer de varias formas. El pixel $(1,1,1)$(1,1,1) representa el blanco y el $(0,0,0)$(0,0,0) el negro en RGB. Entonces, cuanto más grandes son los valores de los canales, más "blanco" es el pixel y viceversa.

Por eso, una forma simple para hacer la conversión consiste en sacar el promedio de los canales y usar eso como la intensidad.
Entonces cada trio (r,g,b)(r,g,b) que representa un pixel se reemplaza por su promedio $(r+g+b)3$(r+g+b)3.

In [21]:
```
# Ejercicio: Convertir la imagen de lena color a escala de grises

h,w,c=lena_rgb.shape # obtenemos el tamaño de la imagen original

lena_gris=np.zeros((h,w)) # creamos una matriz donde generar la imagen

for i in range(h):
    for j in range(w):
        pass
        #IMPLEMENTAR
        # calcular el promedio de los canales r,g,b del pixel i,j con la
imagen original
        # guardar ese promedio en el pixel i,j de la imagen generada

plt.imshow(lena_gris)
```

Out[21]:
```
<matplotlib.image.AxesImage at 0x7f24680e1d30>
```



In [22]:

```
# Ejercicio: Convertir la imagen de lena color a escala de grises, pero s
in usar fors
# Sumar los canales individuales y dividirlos por 3.

lena_gris_simple=0 #IMPLEMENTAR

#plt.imshow(lena_gris_simple) #descomentar para probar
```

Guardar imágenes

Guardar imágenes es tan simple como cargarlas con la función `imsave` del módulo `skimage.io`.

In [23]:

```
#guardamos la imagen de lena gris generada

io.imsave("lena_generada.png",lena_gris)
```

```
#Problema 2
#Mensaje de error generado y corregirlo
/data/dev/dpt/.env/lib/python3.5/site-packages/skimage/io/_io.py:132: Use
rWarning: lena_generada.png is a low contrast image
  warn('%s is a low contrast image' % fname)
/data/dev/dpt/.env/lib/python3.5/site-packages/skimage/util/dtype.py:122:
UserWarning: Possible precision loss when converting from float64 to uint
16
  .format(dtypeobj_in, dtypeobj_out))
```

```
#Problema 3
#Entre la imagen azul y verde hacer una suma de matrices para generar una
imagen x
```

```
#Problema 4
#Convertir la imagen original en blanco y negro
```

```
#https://www.it-swarm.dev/es/python/usando-python-pil-para-convertir-una-imagen-rgb-en-una-imagen-en-blanco-y-negro-puro/942113941/


#Problema 5
# Segmentación basada en histograma (sin información espacial)
# https://claudiovz.github.io/scipy-lecture-notes-ES/advanced/image_processing/index.html
# https://claudiovz.github.io/scipy-lecture-notes-ES/packages/scikit-image/index.html
```