Welcome to Lab 1 – Registration and Introduction to the Lab

Student IDs: 301613501, 208560086

Mount to Google Drive

In [1]:

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

```
Mounted at /content/drive/
```

Change working directory to lab folder

- **Tip: use the *%ls* command to view files within the working directory.**

In [2]:

```
%cd '/content/drive/My Drive/IP Labs/1/'
import os
path = os.getcwd()
print('path: ' + path)
```

```
/content/drive/My Drive/IP Labs/1
path: /content/drive/My Drive/IP Labs/1
```

Import the necessary libraries:

In [3]:

```
%matplotlib inline
import numpy as np
import matplotlib.pylab as plt
from skimage import color
from skimage import img_as_ubyte
```

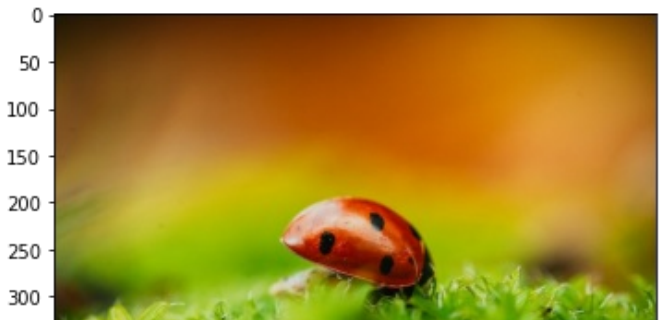
Now you need to write your code:

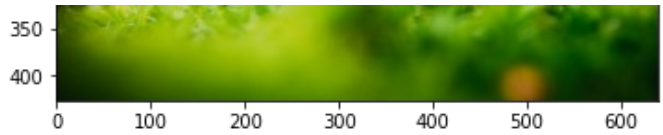
1. **Load your image in Python using imread function, use:**

```
my_img = plt.imread('my_image.bmp')
plt.imshow(my_img)
plt.show()
```

In [4]:

```
my_img = plt.imread('Lab1_Image.bmp')
plt.imshow(my_img)
plt.show()
```





1. **Note the data type and values of your image, use:**

```
my_img.dtype
my_img.shape
```

and print the results

In [5]:

```
print('Image Type:', my_img.dtype)
print('Image Shape:', my_img.shape)
```

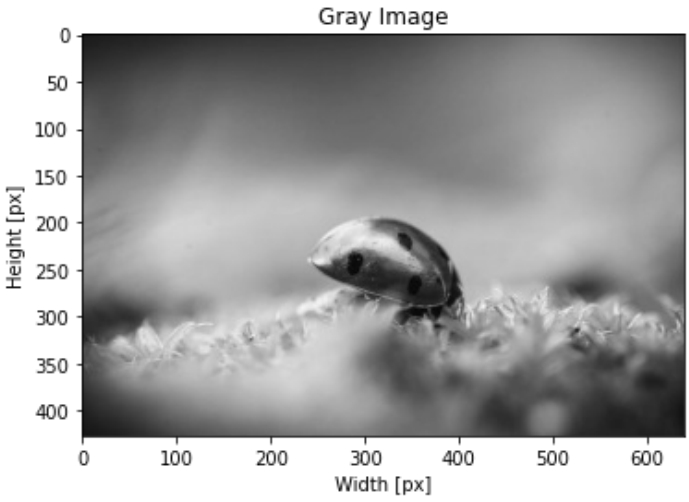
```
Image Type: uint8
Image Shape: (427, 640, 3)
```

1. **Study the color.rgb2gray(image) and plt.imshow(image) functions. notice the cmap argument and it's default value.**
   - **Use *color.rgb2gray(image)* on your image and show the grayscale image.**
   - **Use *image.shape* to determine the image dimensions. print the result**
   - **Has the type changed? print the image type.**

In [6]:

```
# convert and plot the original image as a grayscale (one dimension) image
im_gray = color.rgb2gray(my_img)
plt.imshow(im_gray,cmap = 'gray')
plt.title('Gray Image')
plt.xlabel('Width [px]')
plt.ylabel('Height [px]')

# print the image type and shape
print('Gray Image Type:', im_gray.dtype)
print('Gray Image Shape:', im_gray.shape)
```

```
Gray Image Type: float64
Gray Image Shape: (427, 640)
```



- **How can we use the matrix dimensions to see if our image is in grayscale?**

Answer

Gray scale means that any pixel is represented by a number between 0-255 (1 Byte). Since the matrix dimension have only two dimensions, we can say that the picture is in a grayscale.

To clarify, in case of RGB matrix, we will expect to see 3D matrix that the 3rd dimanssion will be represent the 3 channels for each pixel.

1. **Notice the minimum and maximum values of the matrix (the gray image), use:**

```
np.max
np.min
```

In [7]:

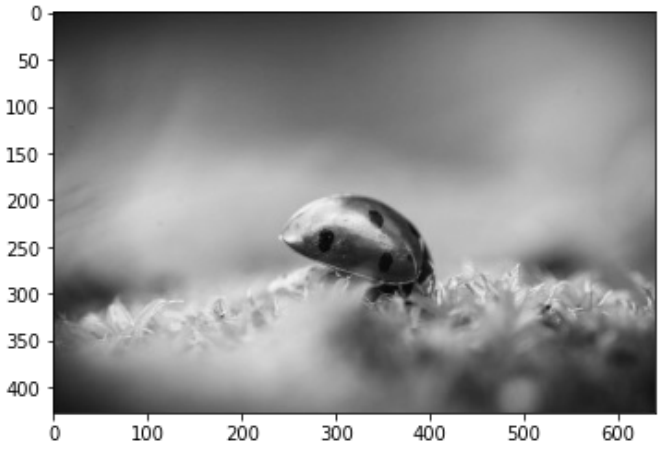
```
print('Gray Image Matrix max:', np.max(im_gray))
print('Gray Image Matrix min:', np.min(im_gray))
```

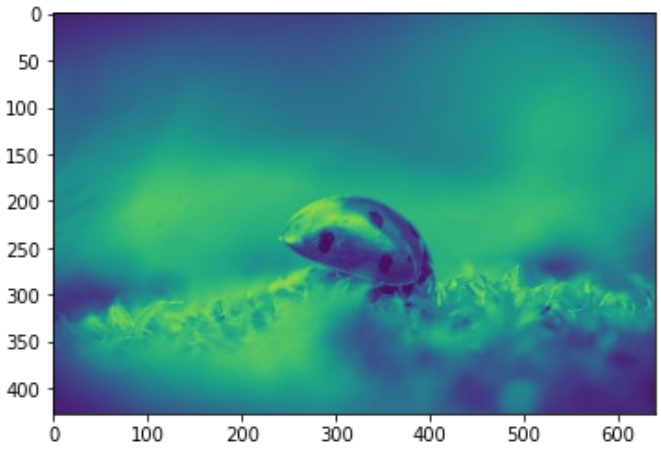
```
Gray Image Matrix max: 0.8845882352941177
Gray Image Matrix min: 0.0008333333333333333
```

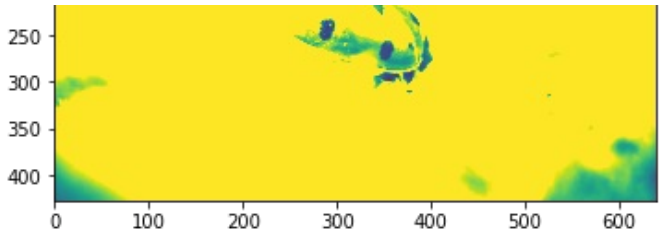
1. **Display the image using imshow. Read about imshow from the following link or Help section, and observe the differences between the following:**

In [8]:

```
plt.imshow(im_gray, cmap = 'gray')
plt.show()
plt.imshow(im_gray,vmin=0, vmax=1) #
plt.show()
plt.imshow(im_gray,vmin=0, vmax=0.5) # data range will be between 0 to 0.5
plt.imshow(im_gray,vmin=0, vmax=0.2) #  data range will be between 0 to 0.2
plt.show()
```







we can see that as much as we decrease the data range the image will be come more and more white (or yellow depends on the color map) (In General in gray images 1 represnt white and 0 represnt black). In the imshow function Vmax will represnt white & Vmin will represnt black (By default, the colormap covers the complete value range of the supplied data) so by decreasing Vmax all the pixels that higher the Vmax will count as white! . We will cover all the data range if Vmax >MAX.VALUE AND Vmin < MIN.VALUE

1. **Save the gray image to a png file in your folder, use:**

```
plt.imsave()
```

In [9]:

```
plt.imsave(path + "/im_gray.png", im_gray, cmap = 'gray')
```

1. **Save the image to a different file and format (jpg).**

In [10]:

```
plt.imsave(path + "/im_gray.jpg", im_gray, cmap = 'gray')
```

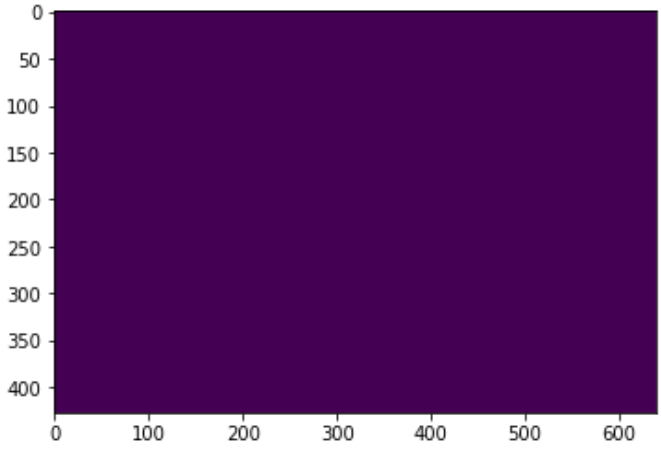
1. **Now we will change the image type from float to uint8 and repeat section 5. Run the following cell and:**
   - **Compare using np.array(image, dtype=np.uint8) and img_as_ubyte(image). State both the differences and similarities. What issue arrises in this example?**
   - **Suggest an alternative solution for the above issue. Add your result and code.**

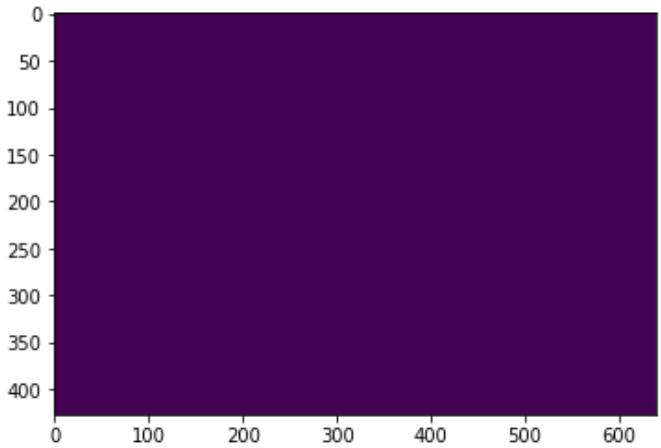
In [11]:

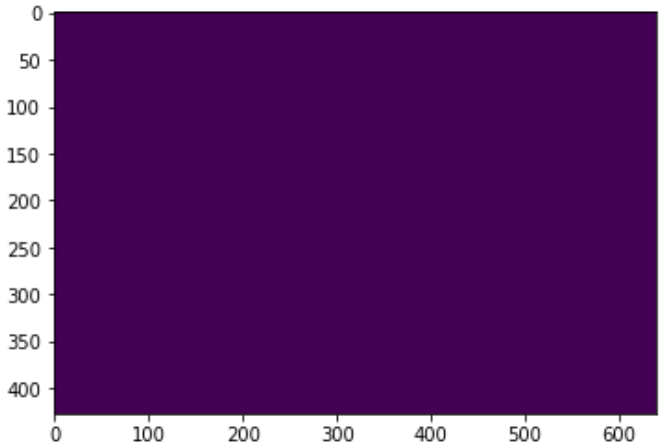
```
image = np.array(im_gray, dtype=np.uint8)
print("image = np.array(im_gray, dtype=np.uint8):")
print(image)
print(image.dtype)

plt.imshow(image)
plt.show()
plt.imshow(image,vmin=0, vmax=1*255)
plt.show()
plt.imshow(image,vmin=0, vmax=0.5*255)
plt.show()
plt.imshow(image,vmin=0, vmax=0.2*255)
plt.show()

im_uint8 = img_as_ubyte(im_gray)
print("\nim_uint8 =img_as_ubyte(im_gray):")
print(im_uint8)
print(im_uint8.dtype)

plt.imshow(im_uint8)
plt.show()
plt.imshow(im_uint8,vmin=0, vmax=1)
plt.show()
plt.imshow(im_uint8,vmin=0, vmax=125)
plt.show()
plt.imshow(im_uint8,vmin=0, vmax=255)
plt.show()
```

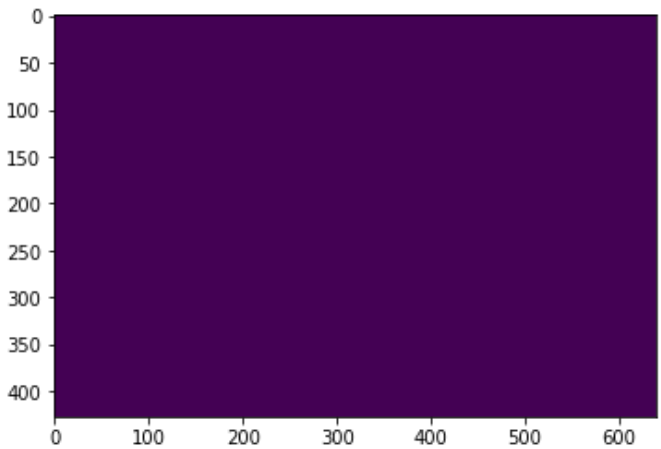
```
image = np.array(im_gray, dtype=np.uint8):
[[0 0 0 ... 0 0 0]
```

```
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
uint8
```



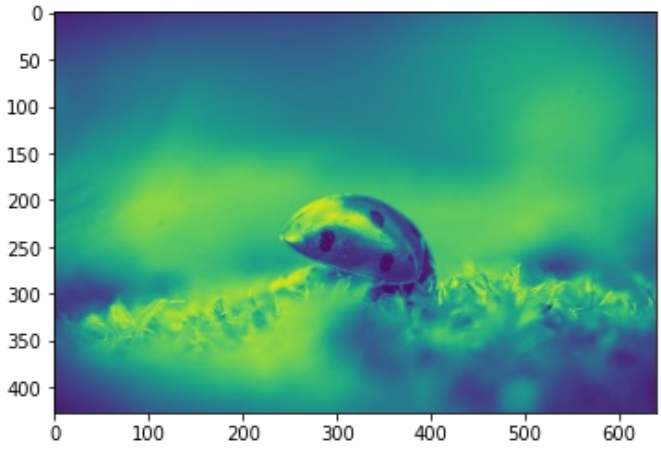


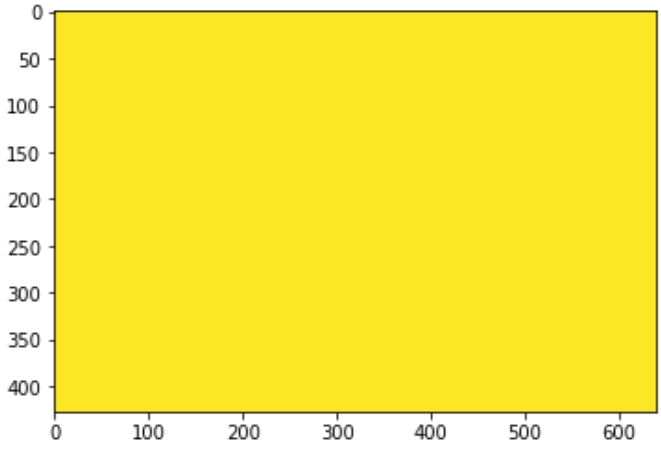


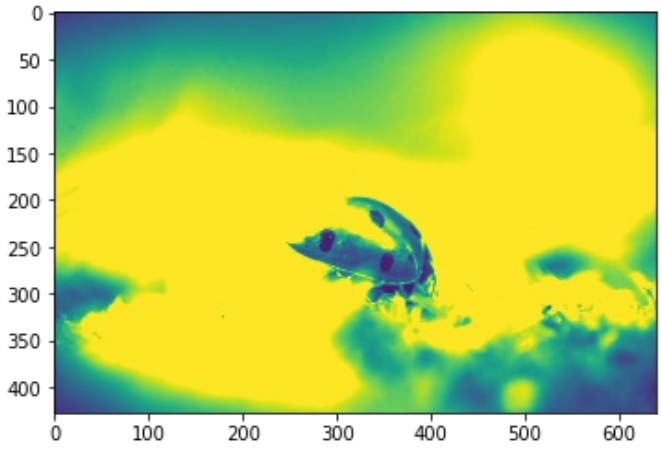


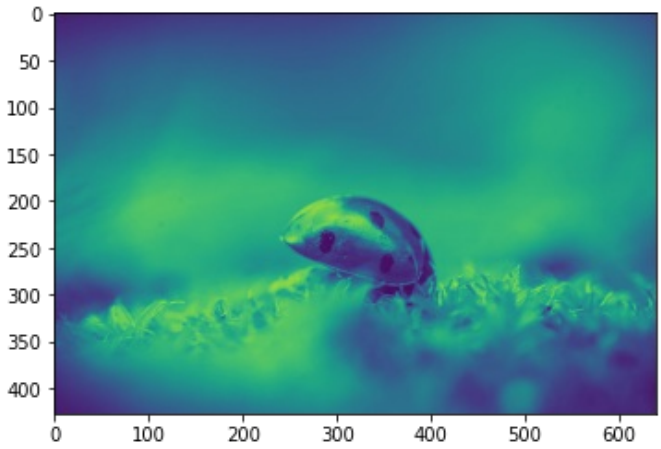

```
im_uint8 =img_as_ubyte(im_gray):
```

```
[[23 23 23 ... 59 58 58]
 [23 23 23 ... 60 59 59]
 [24 24 24 ... 61 60 60]
 ...
 [19 19 19 ... 23 21 21]
 [19 19 19 ... 23 21 21]
 [19 19 19 ... 22 21 21]]
uint8
```









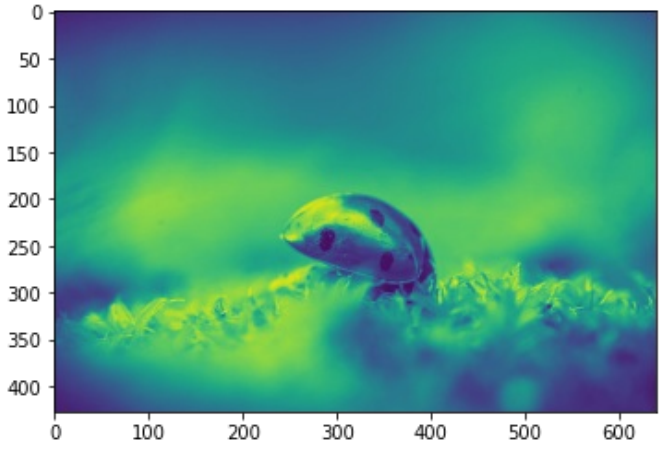
Answer

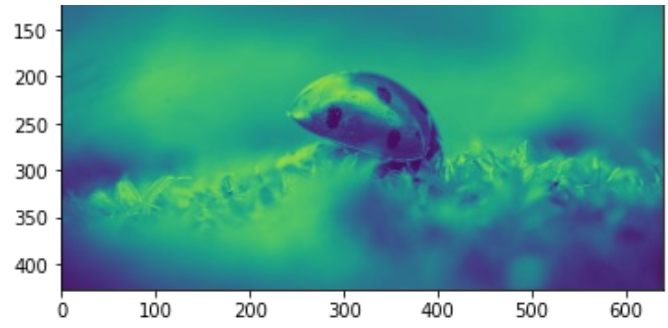
1. **While the image (matrix) is represented by float numbers, each pixel gets its number related to the 256 grayscale level, so the numbers will be between 0 and 1. Therefore, once we converted the matrix to uint8, all the pixels became 0, so the image will be black no matter what.**
2. **When we used the dedicated function "img_as_ubyte()", we can see that the conversion doesn't affect the image.**
3. **Solution: we can use the first method by multiple the matrix values by 255 and update the imshow()'s min\max values accordingly.**

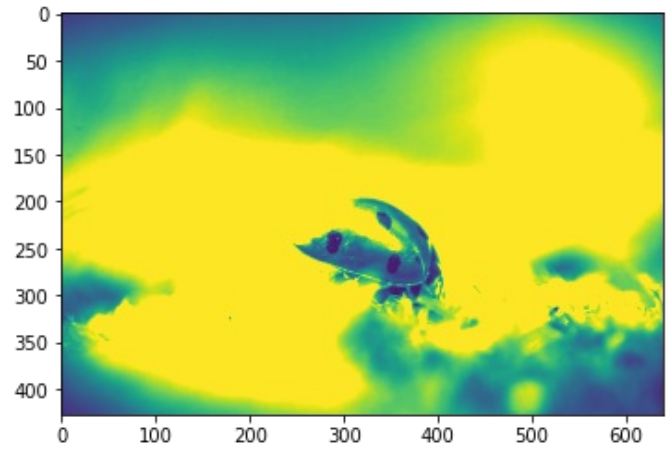
In [12]:

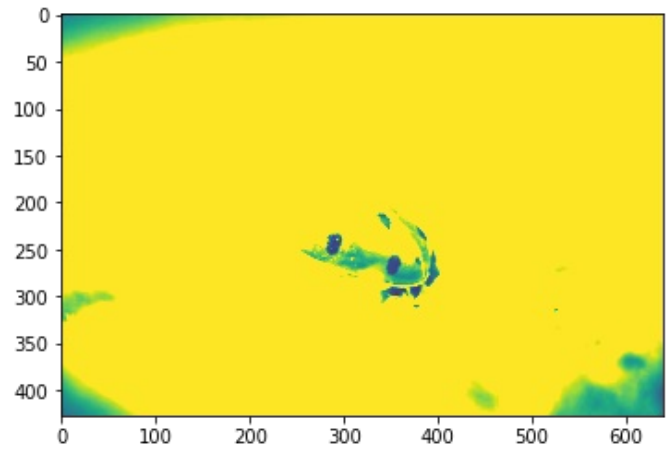
```
image = np.array(im_gray*255, dtype=np.uint8)
print("image = np.array(im_gray, dtype=np.uint8):")
print(image)
print(image.dtype)

plt.imshow(image)
plt.show()
plt.imshow(image,vmin=0, vmax=1*255)
plt.show()
plt.imshow(image,vmin=0, vmax=0.5*255)
plt.show()
plt.imshow(image,vmin=0, vmax=0.2*255)
plt.show()

im_uint8 = img_as_ubyte(im_gray)
print("\nim_uint8 =img_as_ubyte(im_gray):")
print(im_uint8)
print(im_uint8.dtype)

plt.imshow(im_uint8)
plt.show()
plt.imshow(im_uint8,vmin=0, vmax=1)
plt.show()
plt.imshow(im_uint8,vmin=0, vmax=125)
plt.show()
plt.imshow(im_uint8,vmin=0, vmax=255)
plt.show()
```

```
image = np.array(im_gray, dtype=np.uint8):
[[23 23 23 ... 58 57 57]
 [23 23 23 ... 59 58 58]
 [24 24 24 ... 60 59 59]
 ...
 [18 18 19 ... 22 21 21]
 [18 18 19 ... 22 21 21]
 [18 18 18 ... 22 21 21]]
uint8
```

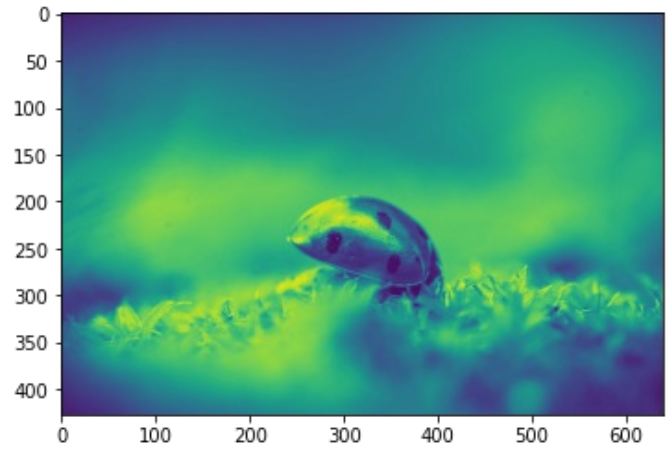


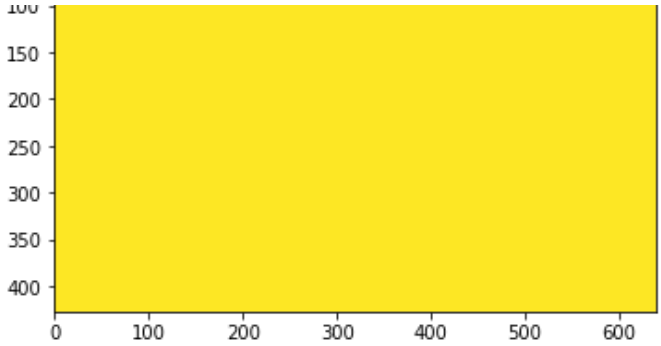


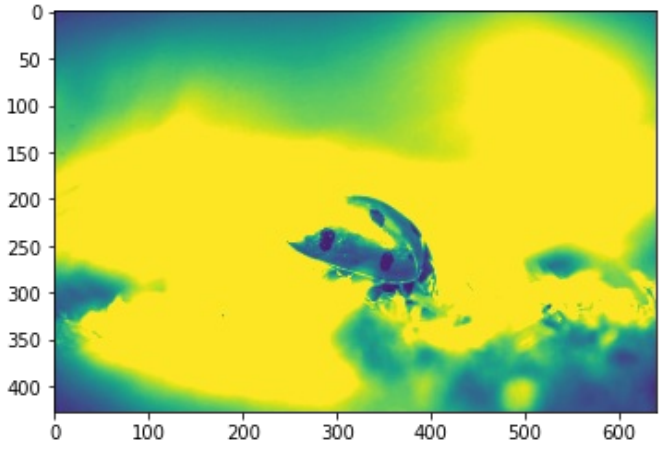


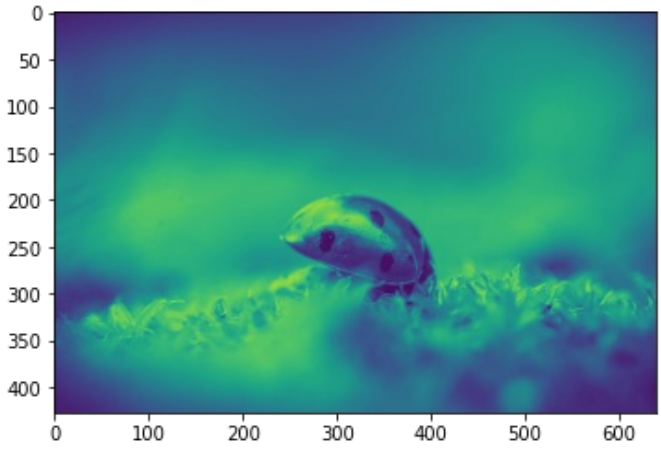


```
im_uint8 =img_as_ubyte(im_gray):
[[23 23 23 ... 59 58 58]
 [23 23 23 ... 60 59 59]
 [24 24 24 ... 61 60 60]
 ...
 [19 19 19 ... 23 21 21]
 [19 19 19 ... 23 21 21]
 [19 19 19 ... 22 21 21]]
uint8
```









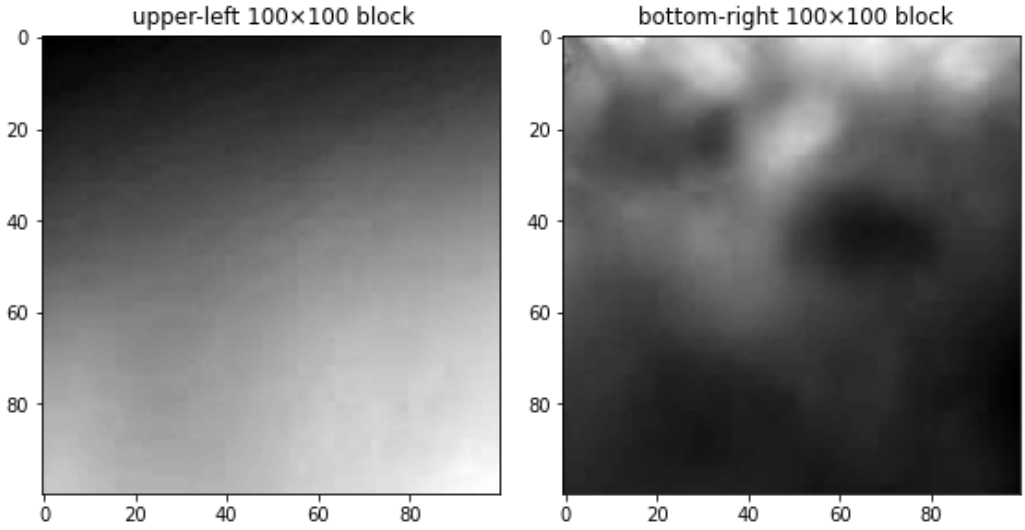
1. **Crop the upper-left 100×100 block of your image. Crop another 100×100 block from the bottom-right part of the image.**

   Hold the 2 blocks in new uint8 variables: B1 and B2. Display the two sections using imshow.

In [13]:

```
B1 = img_as_ubyte(im_gray[0:100,0:100])
B2 = img_as_ubyte(im_gray[-101:-1,-101:-1])
print("B1 Shape is ",B1.shape)
print("B2 Shape is ",B2.shape)


fig, axes = plt.subplots(1, 2, figsize=(8, 4))
ax = axes.ravel()

ax[0].imshow(B1, cmap='gray')
ax[0].set_title(" upper-left 100×100 block")
ax[1].imshow(B2, cmap='gray')
ax[1].set_title(" bottom-right 100×100 block")

fig.tight_layout()
plt.show()
```

```
B1 Shape is  (100, 100)
B2 Shape is  (100, 100)
```



Now save your results and your notebook and go back to lab 1 Manual

ImgThresholding

November 10, 2022

Student IDs: 301613501, 208560086

Mount to Google Drive

[6]:
```
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).
```

Change working directory to lab folder

[7]:
```
%cd '/content/drive/My Drive/IP Labs/1/'
import os
path = os.getcwd()
print('path: ' + path)
```

```
/content/drive/My Drive/IP Labs/1
path: /content/drive/My Drive/IP Labs/1
```

Imports

[8]:
```
%matplotlib inline
import numpy as np
import matplotlib.pylab as plt
from skimage import color, img_as_ubyte
```

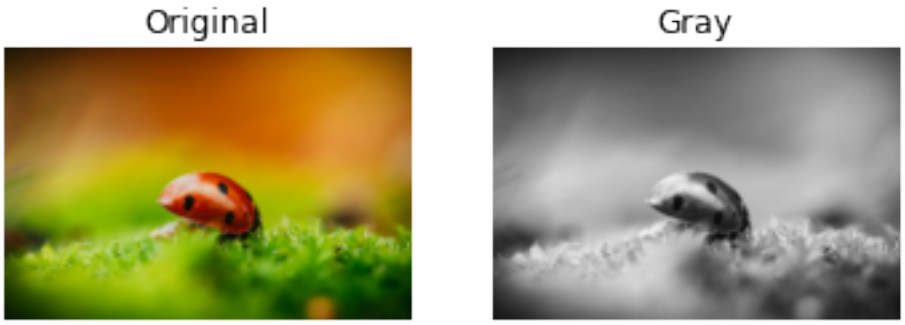
Load image and convert to grayscale * Load your image and convert it to grayscale.

[9]:
```
# load original
my_img = plt.imread('Lab1_Image.bmp')

# conversion to uint8 grayscale
im_gray = color.rgb2gray(my_img)
im_gray = img_as_ubyte(im_gray)
```

- Display your original and grayscale images. Add titles to your images.

[10]:
```
# create figure
fig = plt.figure()
```

```
# setting values to rows and column variables
rows = 1
columns = 2

# Adds a subplot at the 1st position
fig.add_subplot(rows, columns, 1)

# showing image
plt.imshow(my_img)
plt.axis('off')
plt.title("Original")

# Adds a subplot at the 2nd position
fig.add_subplot(rows, columns, 2)

# showing image
plt.imshow(im_gray,cmap = 'gray')
plt.axis('off')
plt.title("Gray")

plt.show()
```



Apply thresholding on your grayscale image

- Write a new function named *img_thresholding(img, T)*.
- Your function should generate a binary image *bin_img* through thresholding the grayscale image *img*: >> *bin_img* is white where *img* > *T*, *bin_img* is black elsewhere.
- The function returns *bin_img*.
- **Use no loops!**

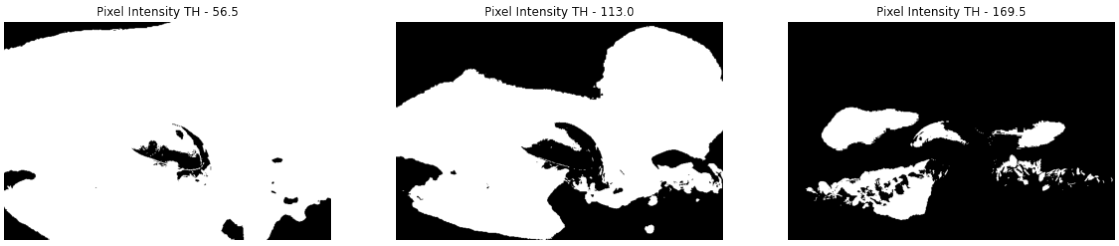
[11]:
```
### This function gets image and 256Bits tresholds ###
### and returns black and white image               ###

def img_thresholding(img, T):

```

```
  bin_img = (img > T)
  return bin_img
```

- Call your function setting $T = 0.25\ x\ M,\ 0.5\ x\ M,\ 0.75\ x\ M$ where M is the maximum pixel intensity of your grayscale image.
- Display your generated images. Don't forget to add titles!

[12]:
```
# Find maximum pixel intensity
M = np.max(im_gray)
print('The pixle intensity threshold is: ' ,M , '\n')

# define a T vector
T = ( 0.25 * M, 0.5 * M, 0.75 * M )

# create figure
fig_new = plt.figure(figsize=(20,7))

# setting values to rows and column variables
rows = 1
columns = 3

for i,t in enumerate (T):
  fig_new.add_subplot(rows, columns, i+1)
  plt.imshow(img_thresholding(im_gray, t),cmap = 'gray')
  plt.axis('off')
  plt.title('Pixel Intensity TH - ' + str(t))
```

```
The pixle intensity threshold is:  226
```



ImgMasking

November 10, 2022

Student IDs: 301613501, 208560086

Mount to Google Drive

[1]:
```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

Change working directory to lab folder

[2]:
```
%cd '/content/drive/My Drive/IP Labs/1/'
import os
path = os.getcwd()
print('path: ' + path)
```

```
/content/drive/My Drive/IP Labs/1
path: /content/drive/My Drive/IP Labs/1
```

Imports

[3]:
```
%matplotlib inline
import numpy as np
import matplotlib.pylab as plt
from skimage import color, img_as_ubyte
```

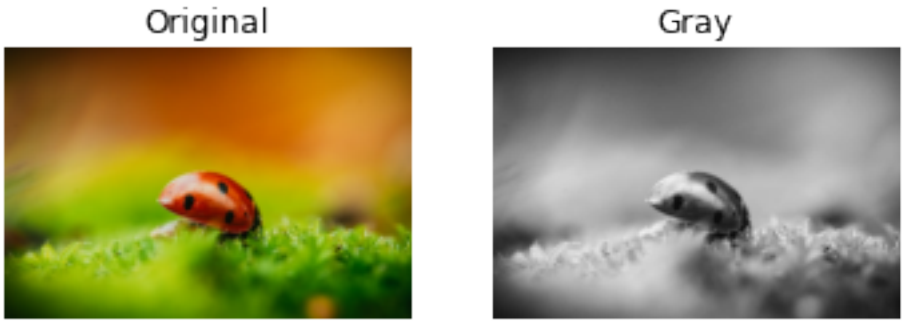
Load image and convert to grayscale * Load your image and convert it to grayscale.

[4]:
```
# load original
my_img = plt.imread('Lab1_Image.bmp')

# conversion to uint8 grayscale
im_gray = color.rgb2gray(my_img)
im_gray = img_as_ubyte(im_gray)
```

- Display your original and grayscale images. Add titles to your images.

[5]:
```
# create figure
fig = plt.figure()

# setting values to rows and column variables
```

```
rows = 1
columns = 2

# Adds a subplot at the 1st position
fig.add_subplot(rows, columns, 1)

# showing image
plt.imshow(img_as_ubyte(my_img))
plt.axis('off')
plt.title("Original")

# Adds a subplot at the 2nd position
fig.add_subplot(rows, columns, 2)

# showing image
plt.imshow(img_as_ubyte(im_gray), cmap='gray')
plt.axis('off')
plt.title("Gray")

plt.show()
```



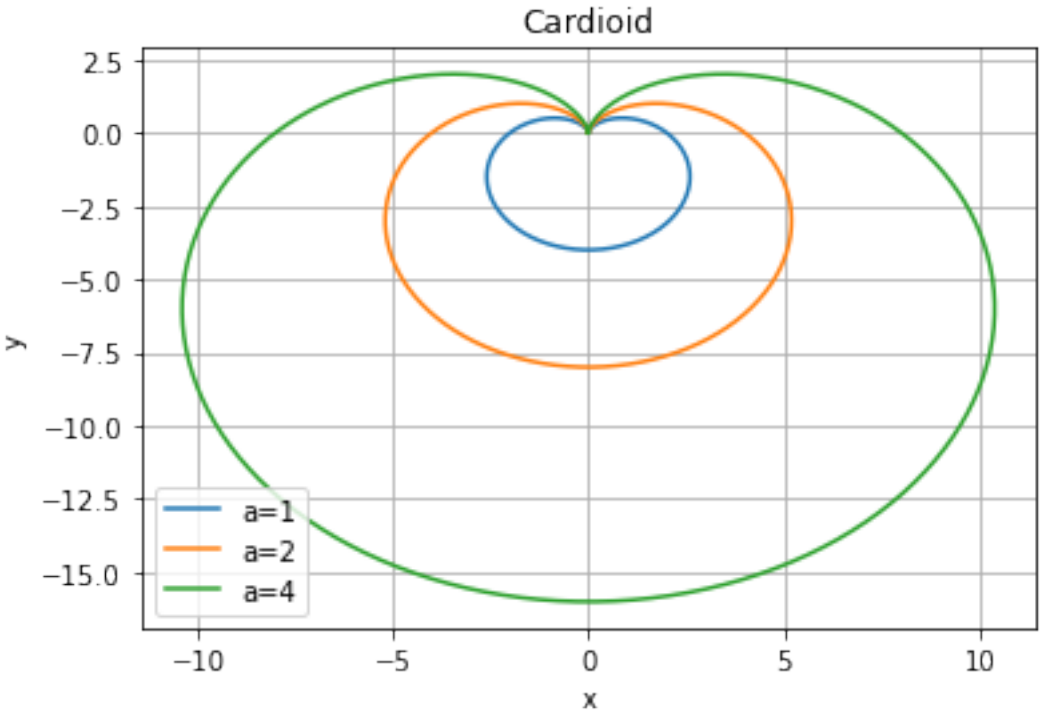
Plot 2D Cardioids

- Having relatively simple geometric interpretation, the Cardioid is well known for its resemblance to a heart.
- All the Cardioid equations needed for this notebook can be found here.
- Recreate figure 1 (left) from the lab manual. Generate a 2D plot containing generated rotated Cardioids setting *a=1,2,4*.
- Note that here (x,y) may be swapped for obtaining the desired result (rotated).
- **Hint:** Use matplotlib.pyplot.plot

[6]:
```
import matplotlib
from matplotlib.pyplot import plot

# set parameters
```

```
dphi = 0.01

pi = np.pi
phi = np.arange(0, 2*pi, dphi)

A = [1,2,4]

for a in A:
  x = 2*a*(1-np.cos(phi))*np.sin(phi)
  y = 2*a*(1-np.cos(phi))*np.cos(phi)
  plot(x,y, label = 'a=' + str(a))

plt.legend()
plt.grid()
plt.title('Cardioid')
plt.xlabel('x')
plt.ylabel('y')
```

[6]: `Text(0, 0.5, 'y')`



Generate a heart shaped binary mask using Cardioid * Write a new function named *generate_heart_mask(dim,a,c)*. * Your function should generate a Cardioid shaped binary mask of size *dim=[h,w]*, where all pixels in its interior are assigned *1*, and *0* elsewhere. See figure 1 (middle). * The argument *a* defines the Cardiod radius (in pixels). * The argument *c=[cx,cy]* defines the

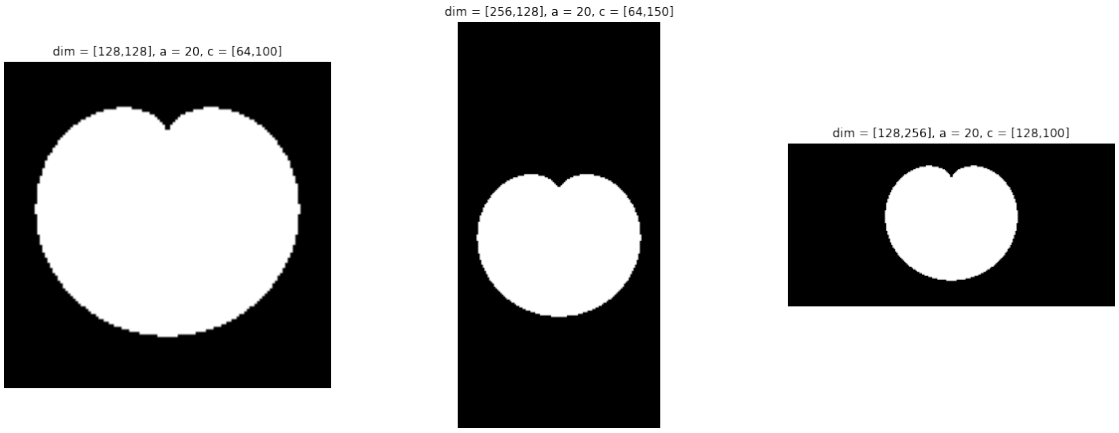
position of its cetner. We consider the point (0,0) in figure 1 (left) to be the Cardioid center. * **Use no loops!** * **Hints:** 1. You may want to use *numpy.meshgrid* (read online). 2. Recall that the origin of images is commonly located at the upper left corner with the y-axis facing down.

[7]:
```
def generate_heart_mask(dim,a,c):
  # parse dimensions
  h, w = dim
  cx, cy = c

  # since the origin of image is located at the upper left corner, we will move
  # it to the bottom left corner by inverse the y vector order
  x = np.linspace(0, w, num=w)
  y = np.linspace(h, 0, num=h)

  xv,yv = np.meshgrid(x,y)

  # in order to rotate the shape in 90 degrees, we will replace between x and y
  x = yv-cy
  y = xv-cx

  mask = ((x**2+y**2)**2+4*a*x*(x**2+y**2)-(2*a*y)**2 < 0)

  return mask
```

Plot your generated mask setting: * *dim = [128,128], a = 20, c = [64,100]* * *dim = [256,128], a = 20, c = [64,150]* * *dim = [128,256], a = 20, c = [128,100]*

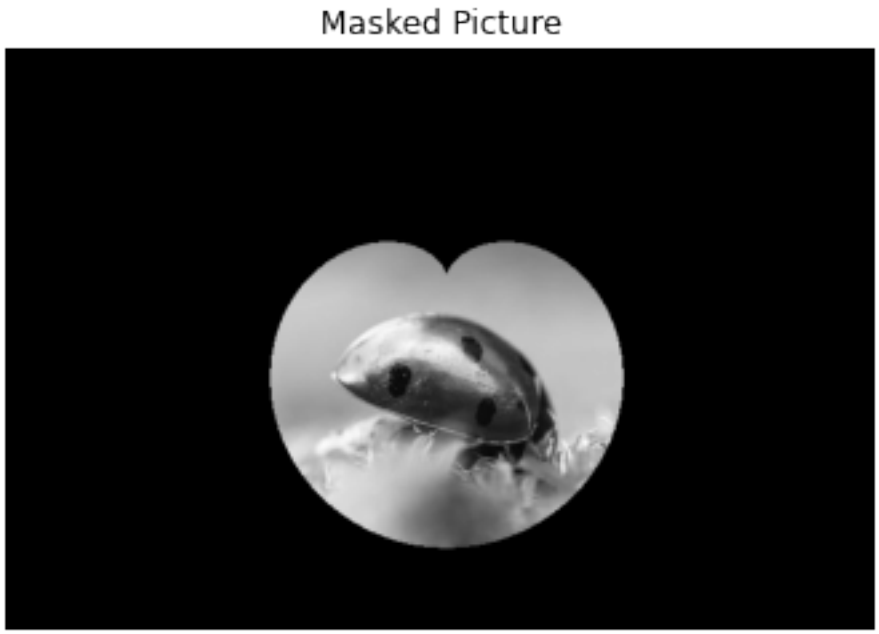
[8]:
```
# create figure
fig = plt.figure(figsize=(20,7.5))

# setting values to rows and column variables
rows = 1
columns = 3

# Adds a subplot at the 1st position
fig.add_subplot(rows, columns, 1)

# showing image
plt.imshow(img_as_ubyte(generate_heart_mask([128,128],20,[64,100])),␣
 ↪cmap='gray')
plt.axis('off')
plt.title("dim = [128,128], a = 20, c = [64,100]")

# Adds a subplot at the 2nd position
fig.add_subplot(rows, columns, 2)

# showing image
plt.imshow(generate_heart_mask([256,128],20,[64,150]), cmap='gray')
```

```
plt.imshow(img_as_ubyte(generate_heart_mask([256,128],20,[64,150])),␣
 ↪cmap='gray')
plt.axis('off')
plt.title("dim = [256,128], a = 20, c = [64,150]")


# Adds a subplot at the 2nd position
fig.add_subplot(rows, columns, 3)

# showing image
plt.imshow(img_as_ubyte(generate_heart_mask([128,256],20,[128,100])),␣
 ↪cmap='gray')
plt.axis('off')
plt.title("dim = [128,256], a = 20, c = [128,100]")

plt.show()
```



Generate a masked image * Generate a new Cardioid mask and use it to obtain a masked image. See figure 1 (right). * Find the combinarion of arguments to your function which yields the best results.

[9]:
```
dim = im_gray.shape
a = 50
location = [325,260]

mask = (generate_heart_mask(dim,a,location))
plt.imshow(img_as_ubyte(mask * im_gray), cmap='gray')
plt.axis('off')
plt.title("Masked Picture")

plt.show()
```



[9]: