MATPLOTLIB

https://www.w3schools.com/python/matplotlib_getting_started.asp (https://www.w3schools.com/python/matplotlib_getting_started.asp)

Plotting x and y points

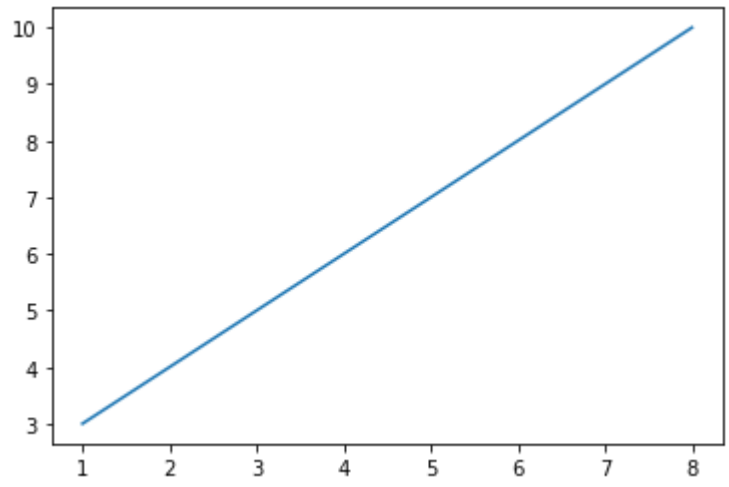
- *plot()* sirve para dibujar puntos en un diagrama.
- Por defecto, dibuja una línea recta de un punto a otro

Toma **dos parámetros**:

1. Un array que contiene los puntos en el eje de las x
2. Un array con los puntos en y

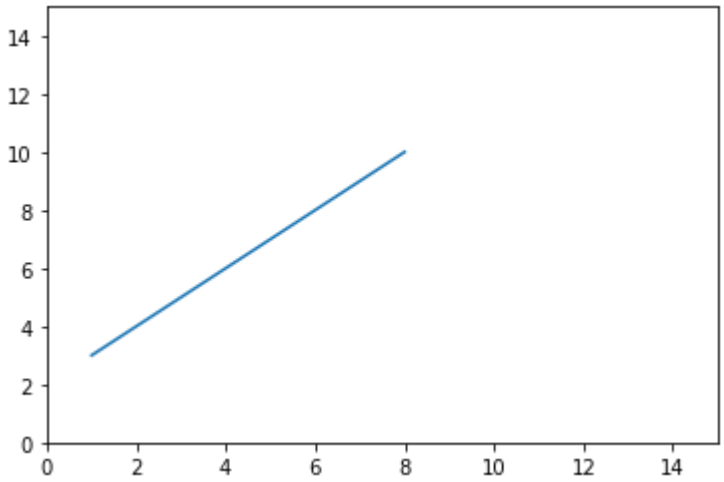
If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints) #dibuja una línea del punto (1, 3) hasta el (8
```

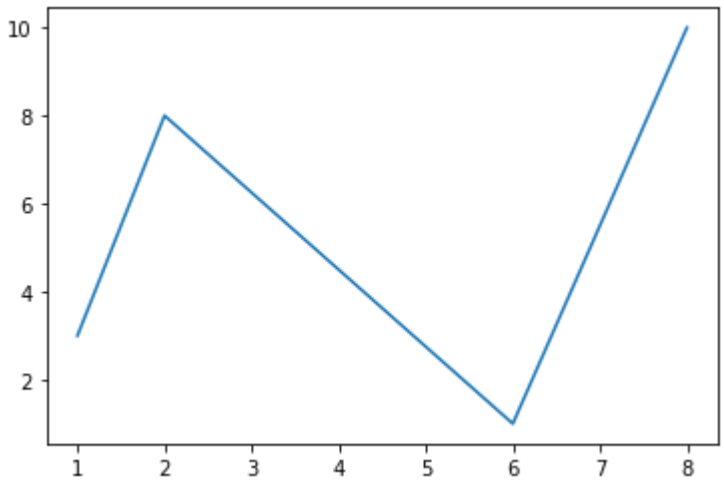


The axis function in the example below takes a list of [xmin, xmax, ymin, ymax] and specifies the viewport of the axes.

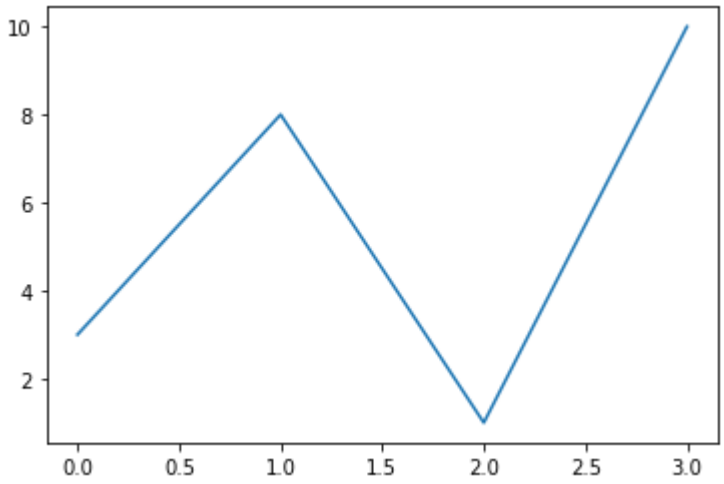
In [3]:
```
plt.plot(xpoints, ypoints)
plt.axis([0, 15, 0, 15])
```



In [4]:
```
#representar diversos puntos

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

plt.plot(xpoints, ypoints)
```



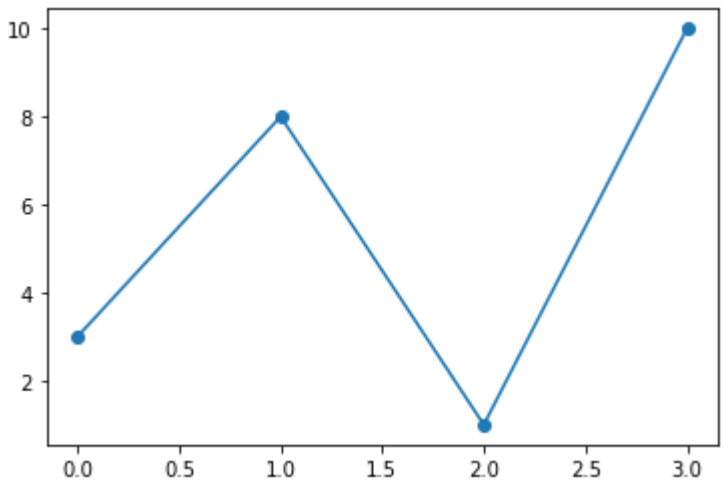
In [6]:
```
#si no le pasamos las x, por defecto ubicará los puntos de y en x = [0, 1

plt.plot(ypoints)
plt.show()
```



MARKERS

You can use the keyword **argument marker** to emphasize each point with a specified marker:

In [7]:
```
plt.plot(ypoints, marker = 'o')
```



Nota: se puede cambiar la forma del marcador. Más info en: https://www.w3schools.com/python/matplotlib_markers.asp (https://www.w3schools.com/python/matplotlib_markers.asp)

Format strings *fmt*

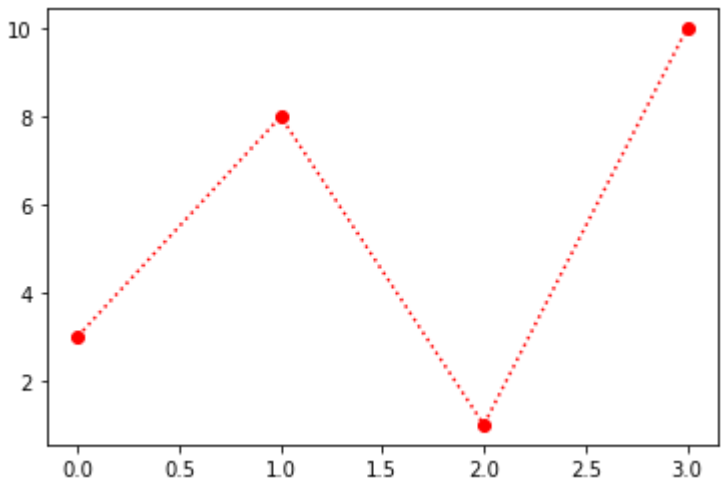
You can use also use the shortcut string notation parameter to specify the marker.

This parameter is also called fmt, and is written with this syntax:

marker|line|color

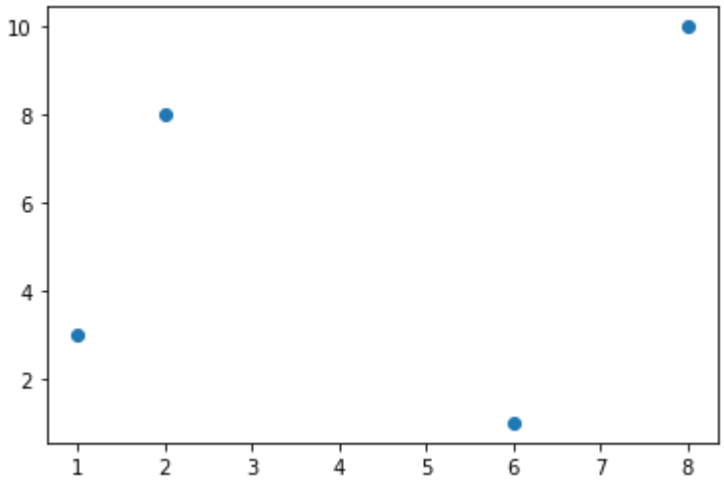
Más info en: https://www.w3schools.com/python/matplotlib_markers.asp

In [8]:
```
plt.plot(ypoints, 'o:r')
```



También se pueden dibujar solo los puntos mediante el **parámetro 'o'**, que significa "anillos" (*rings*):

In [5]:
```
plt.plot(xpoints, ypoints, 'o')
```



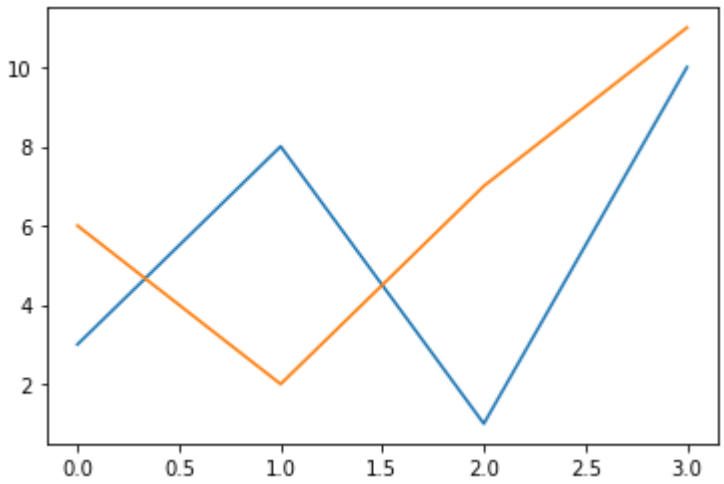
LÍNEA

Estilos:

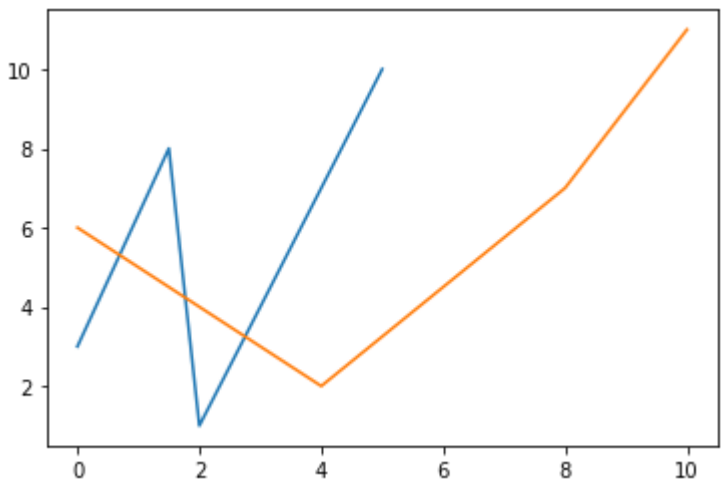
Para saber qué tipos de estilos de línea se pueden hacer y cómo: https://www.w3schools.com/python/matplotlib_line.asp (https://www.w3schools.com/python/matplotlib_line.asp)

Representar múltiples líneas:

In [9]:
```
import matplotlib.pyplot as plt
import numpy as np

y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])

plt.plot(y1)
plt.plot(y2)
```



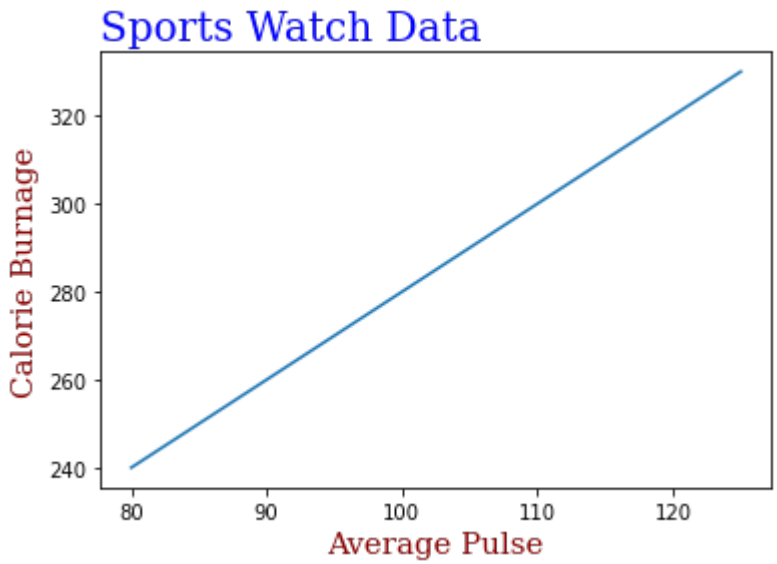
In [10]:
```
x1 = np.array([0, 1.5, 2, 5])
y1 = np.array([3, 8, 1, 10])
x2 = np.array([0, 4, 8, 10])
y2 = np.array([6, 2, 7, 11])

plt.plot(x1, y1, x2, y2)
```



ETIQUETAS

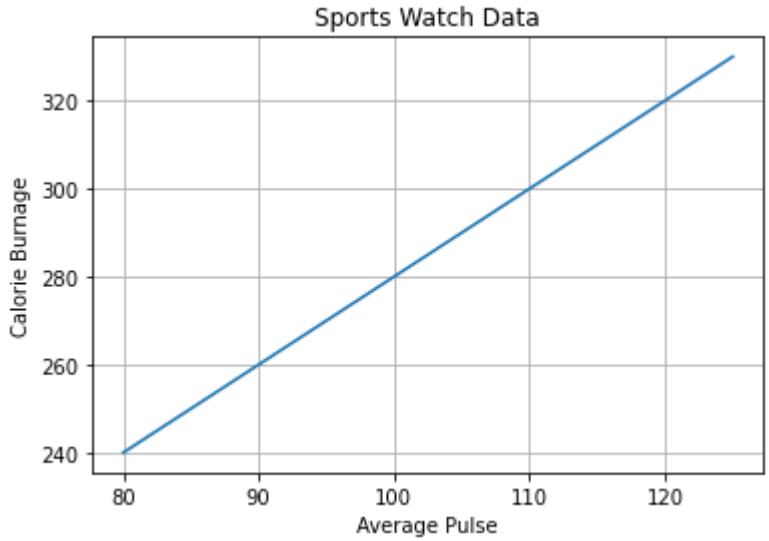
- *xlabel() & ylabel()*
- *title()*

In [6]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("Sports Watch Data", fontdict = font1, loc = 'left') #tb hemos
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)


plt.show()
```



GRID(): añadir rejas al plot

In [7]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid() #cuadricular
```



También hay:

- *plt.grid(axis = 'x')*
- *plt.grid(axis = 'y')*
- *plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)*

más info en https://www.w3schools.com/python/matplotlib_grid.asp (https://www.w3schools.com/python/matplotlib_grid.asp)

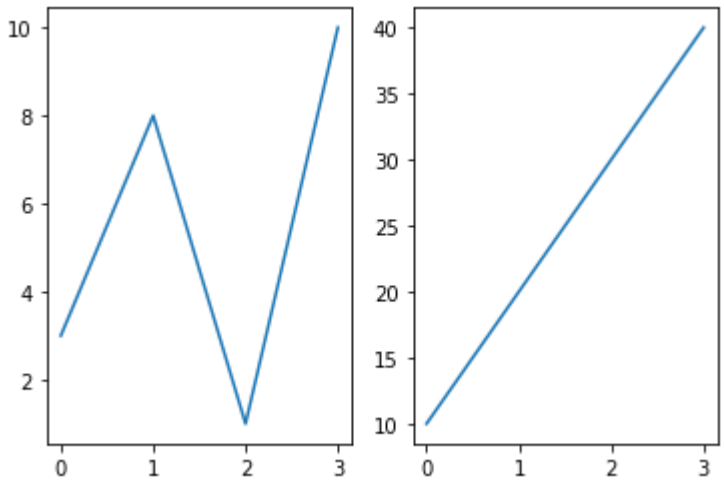
SUBPLOT

Dibujar múltiples representaciones en una figura.

```
plt.subplot(1, 2, 1)
#the figure has 1 row, 2 columns, and this plot is the first plo
t.

plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, and this plot is the second plo
t.
```

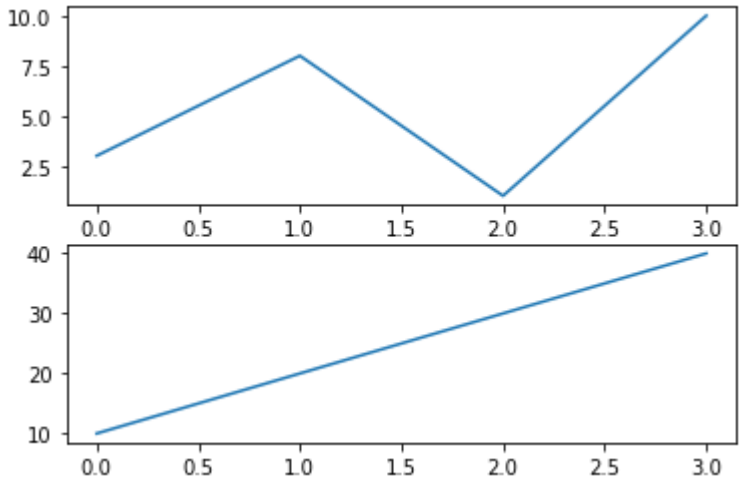
- The subplots() function takes three arguments that describes the layout of the figure.
- The layout is organized in rows and columns, which are represented by the first and second argument.
- The third argument represents the index of the current plot.

In [8]:
```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
```

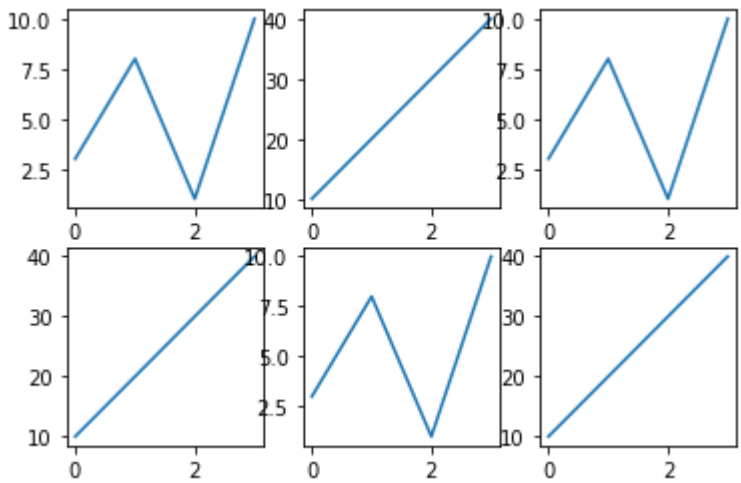


In [9]:
```
#So, if we want a figure with 2 rows an 1 column (meaning that the two pl
#will be displayed on top of each other instead of side-by-side), we can

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 1, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 1, 2)
```

```
plt.plot(x,y)

plt.show()
```



In [10]:
```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)
```

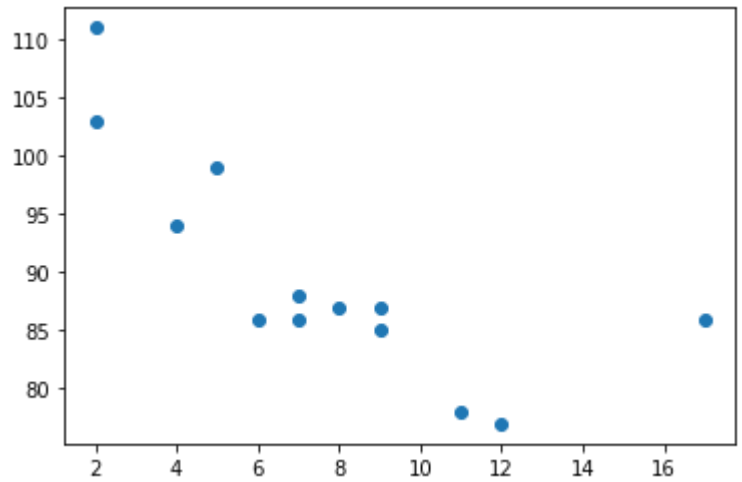


- *title()*: poner título a cada gráfica
- *suptitle()*: título general para todas las gráficas

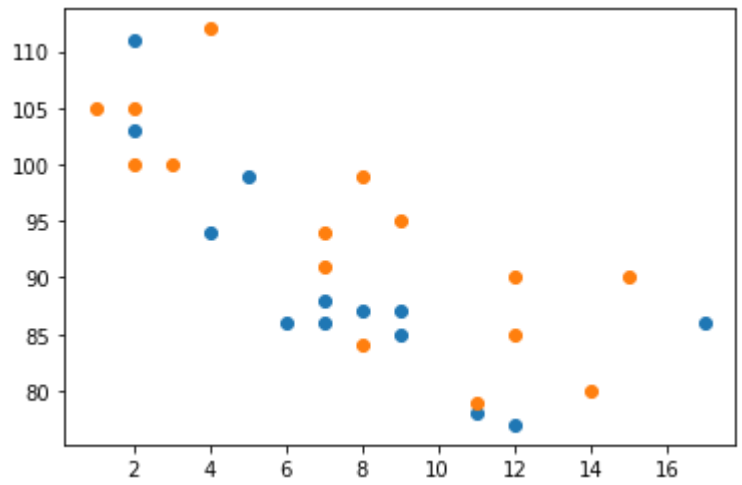
SCATTER()

- *scatter()* dibuja un punto por cada observación.
- Toma dos arrays de la misma longitud como parámetros: uno para los valores del eje X y otro para los del eje Y.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
```



In [2]:
```
#DIBUJAR DOS REPRESENTACIONES EN LA MISMA FIGURA CON SCATTER()

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)
```



Colors

You can set your own color for each scatter plot with the color or the c argument:

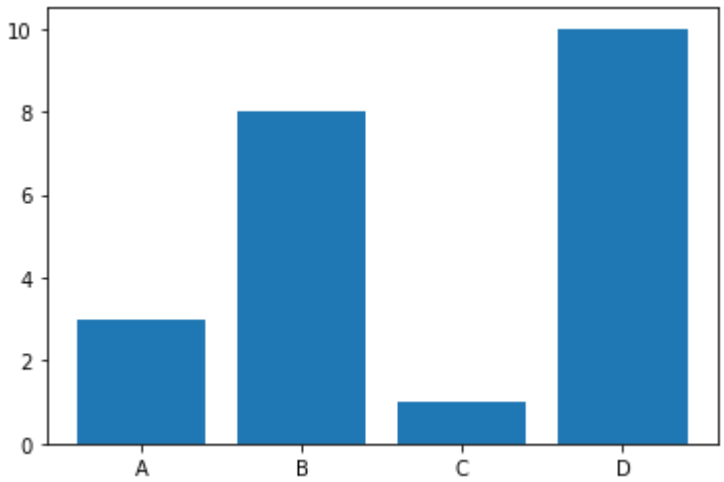
- plt.scatter(x, y, **color = 'hotpink'**)
- plt.scatter(x, y, **color = '#88c999'**)

Más sobre colorear puntos, determinar el **size** y el **alpha (transparencia)** en: https://www.w3schools.com/python/matplotlib_scatter.asp (https://www.w3schools.com/python/matplotlib_scatter.asp)

BAR(): diagramas de barras

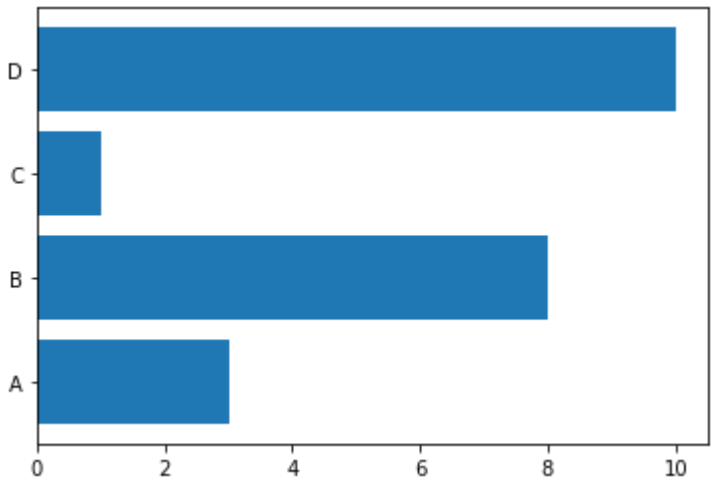
- *bar()* toma dos arrays como parámetros; uno de valores numéricos y otro para los nombres de las barras.

In [3]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
```



Utiliza ***barth()*** para hacer un diagrama de barras horizontal:

In [4]:
```
plt.barh(x, y)
```

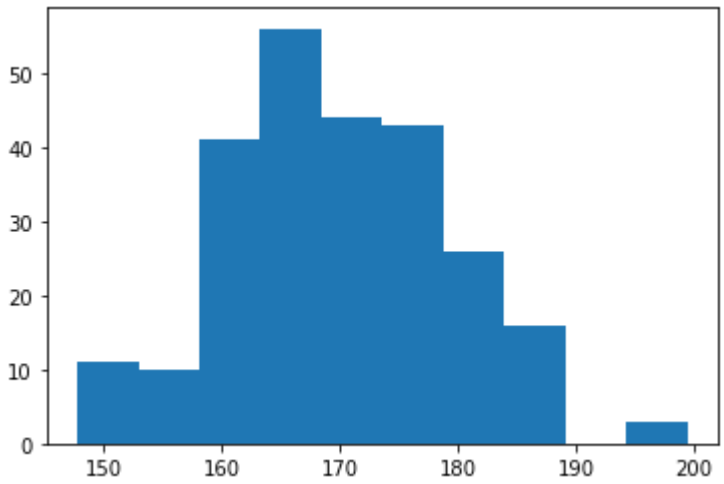


Para determinar los **colores** y la **anchura** de las barras: https://www.w3schools.com/python/matplotlib_bars.asp (https://www.w3schools.com/python/matplotlib_bars.asp)

HISTORIOGRAMAS: *hist()*

- A histogram is a graph showing frequency distributions. It is a graph showing the number of observations within each given interval.
- Toma como argumento un array de números (cada número será un caso) y representará la frecuencia de veces que ocurren casos entre cierto rango de valores.

In [5]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(170, 10, 250) #(media, desviación, nº de casos)

plt.hist(x)
```



OTRO TUTORIAL

https://www.youtube.com/watch?v=3Xc3CA655Y4 (https://www.youtube.com/watch?v=3Xc3CA655Y4)

Matplotlib básico

Métodos:

- plt.plot(x, y, label=, color=, linewitdth=, linestyle=, marker=, marksize=, markeredgecolor=,)
- plt.plot('[color][marker][line]')
- plt.title()
- plt.xlabel()
- plt.legend()
- plt.figure(figsize=(n,n), dpi=)
- plt.savefig('', dpi=)
- plt.bar(labels, values)
- .set_hatch('')

Parámetros:

- fontdict={'fontname': , 'fontsize': , 'fontweight': }

Matplotlib w Pandas (real examples)

- **df.plot()**: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html) (KIND)

Gas data

- plt.plot(gas.Year, gas.USA) #son ejemplos: gas es el dataframe y Year y Usa son las columnas
- plt.xticks(gas.Year[::3]) --> plt.xticks(gas.Year[::3].tolist()+[2011])

```
for country in gas:
    if country != 'Year':
        plt.plot(gas.Year, gas[country], marker='.')
```

Fifa data

- plt.hist(fifa.Overall, bins=, color=)
- plt.pie([x, y, z...], labels=, colors=, autopct='%.2f %%', pctdistance=, explode=)
- plt.style.use('')
- plt.boxplot([x, y])

FUENTES DE INFO

- Toda la documentación de pyplot: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html)
- Lista de todas las fuentes disponibles **(fontdict={'fontname': , 'fontsize': })**: http://jonathansoma.com/lede/data-studio/matplotlib/list-all-fonts-available-in-matplotlib-plus-samples/ (http://jonathansoma.com/lede/data-studio/matplotlib/list-all-fonts-available-in-matplotlib-plus-samples/)
- Buscar en google "color picker"

OTRO TUTORIAL

https://matplotlib.org/stable/tutorials/introductory/pyplot.html (https://matplotlib.org/stable/tutorials/introductory/pyplot.html)

- yscale('') : https://matplotlib.org/stable/tutorials/introductory/pyplot.html (https://matplotlib.org/stable/tutorials/introductory/pyplot.html)

Anotaciones: https://matplotlib.org/stable/tutorials/text/text_intro.html (https://matplotlib.org/stable/tutorials/text/text_intro.html)

Ejemplo:

In [3]:
```
import matplotlib.pyplot as plt
import numpy as np
ax = plt.subplot()

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black', shrink=0.05),
             )

plt.ylim(-2, 2)
```

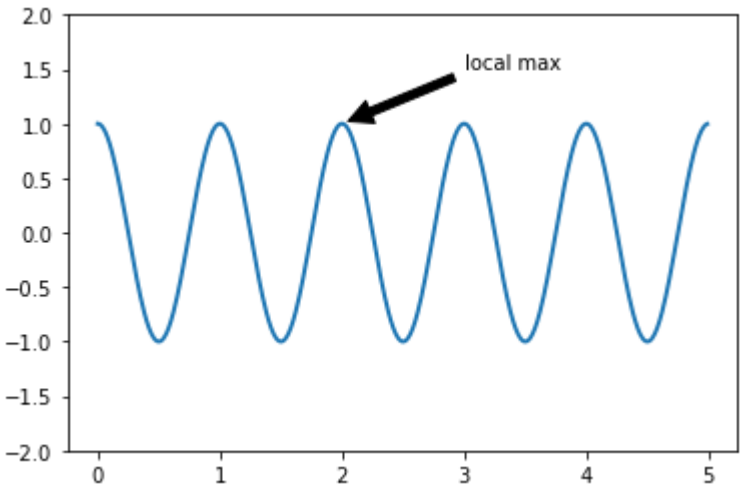


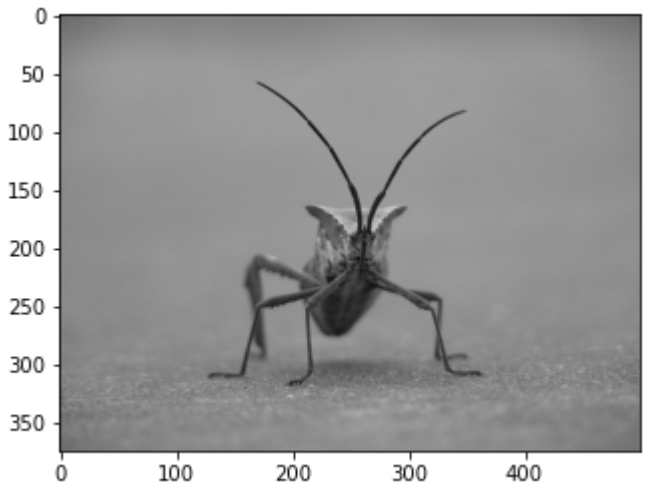
IMAGE TUTORIAL

In [1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
%matplotlib inline

img = mpimg.imread('stinkbug.png')
```

```
[[[0.40784314 0.40784314 0.40784314]
  [0.40784314 0.40784314 0.40784314]
  [0.40784314 0.40784314 0.40784314]
  ...
  [0.42745098 0.42745098 0.42745098]
  [0.42745098 0.42745098 0.42745098]
  [0.42745098 0.42745098 0.42745098]]

 [[0.4117647  0.4117647  0.4117647 ]
  [0.4117647  0.4117647  0.4117647 ]
  [0.4117647  0.4117647  0.4117647 ]
  ...
  [0.42745098 0.42745098 0.42745098]
  [0.42745098 0.42745098 0.42745098]
  [0.42745098 0.42745098 0.42745098]]

 [[0.41960785 0.41960785 0.41960785]
  [0.41568628 0.41568628 0.41568628]
  [0.41568628 0.41568628 0.41568628]
  ...
  [0.43137255 0.43137255 0.43137255]
  [0.43137255 0.43137255 0.43137255]
  [0.43137255 0.43137255 0.43137255]]

 ...

 [[0.4392157  0.4392157  0.4392157 ]
  [0.43529412 0.43529412 0.43529412]
  [0.43137255 0.43137255 0.43137255]
  ...
  [0.45490196 0.45490196 0.45490196]
  [0.4509804  0.4509804  0.4509804 ]
  [0.4509804  0.4509804  0.4509804 ]]
```

Plotting numpy arrays as images (imshow)

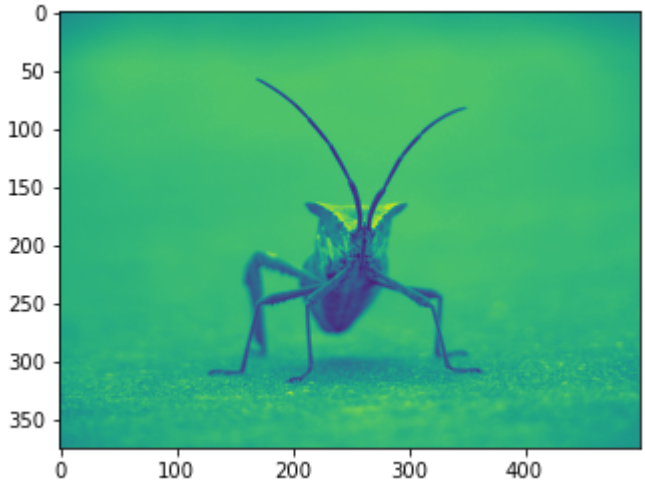
In [2]:



Applying pseudocolor schemes to image plots

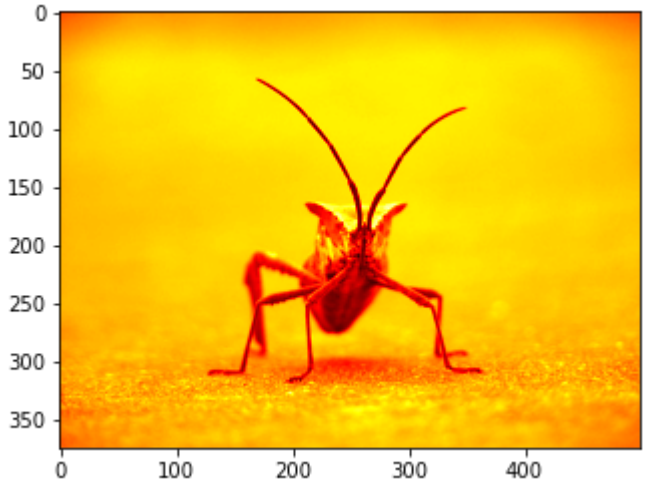
In [3]:
```
lum_img = img[:, :, 0]
```

Out[3]: <matplotlib.image.AxesImage at 0x20984ea3dc0>



In [4]:

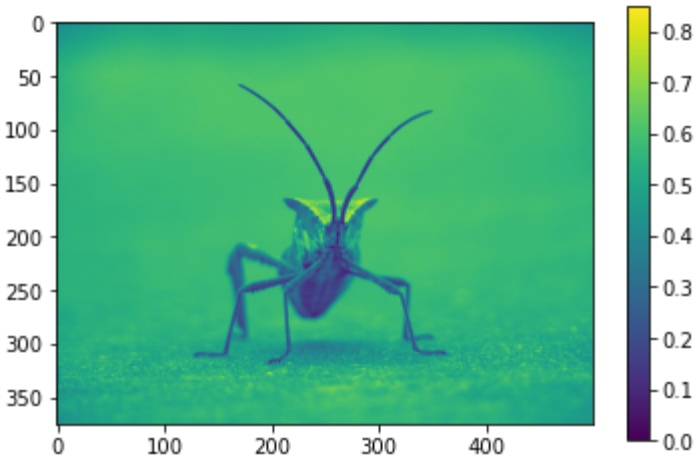
Out[4]: <matplotlib.image.AxesImage at 0x20984f14b80>



Color scale reference

In [5]:
```
#ADDING A COLORBAR
imgplot = plt.imshow(lum_img)
```

Out[5]:
```
<matplotlib.colorbar.Colorbar at 0x20984fad0d0>
```



Examining a specific data range

Sometimes you want to enhance the contrast in your image, or expand the contrast in a particular region while sacrificing the detail in colors that don't vary much, or don't matter. A good tool to find interesting regions is the histogram.

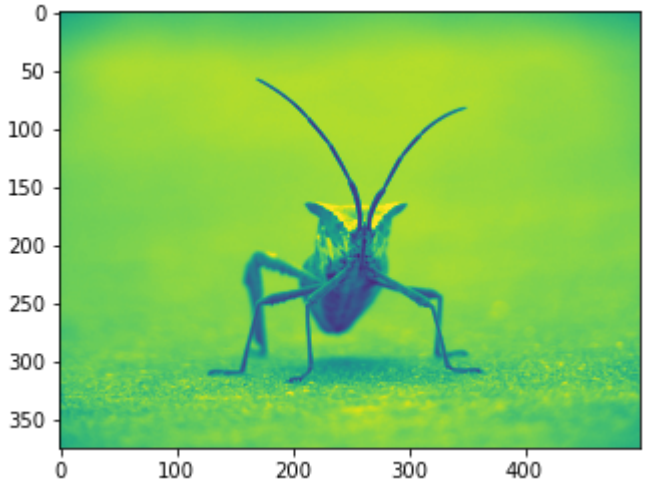
In [6]:

Out[6]:
```
(array([2.000e+00, 2.000e+00, 3.000e+00, 3.000e+00, 2.000e+00, 2.000e+
00,
        3.000e+00, 1.000e+00, 7.000e+00, 9.000e+00, 7.000e+00, 2.000e+
00,
        7.000e+00, 1.000e+01, 1.100e+01, 1.500e+01, 1.400e+01, 2.700e+
01,
        2.100e+01, 2.400e+01, 1.400e+01, 3.100e+01, 2.900e+01, 2.800e+
01,
        2.400e+01, 2.400e+01, 4.000e+01, 2.600e+01, 5.200e+01, 3.900e+
01,
        5.700e+01, 4.600e+01, 8.400e+01, 7.600e+01, 8.900e+01, 8.000e+
01,
        1.060e+02, 1.130e+02, 1.120e+02, 9.000e+01, 1.160e+02, 1.090e+
02,
        1.270e+02, 1.350e+02, 9.800e+01, 1.310e+02, 1.230e+02, 1.110e+
02,
        1.230e+02, 1.160e+02, 1.010e+02, 1.170e+02, 1.000e+02, 1.010e+
02,
        9.000e+01, 1.060e+02, 1.260e+02, 1.040e+02, 1.070e+02, 1.110e+
```

Most often, the "interesting" part of the image is around the peak, and you can get extra contrast by clipping the regions above and/or below the peak. In our histogram, it looks like there's not much useful information in the high end (not many white things in the image). Let's adjust the upper limit, so that we effectively "zoom in on" part of the histogram. We do this by passing the clim argument to imshow. You could also do this by calling the set_clim() method of the image plot

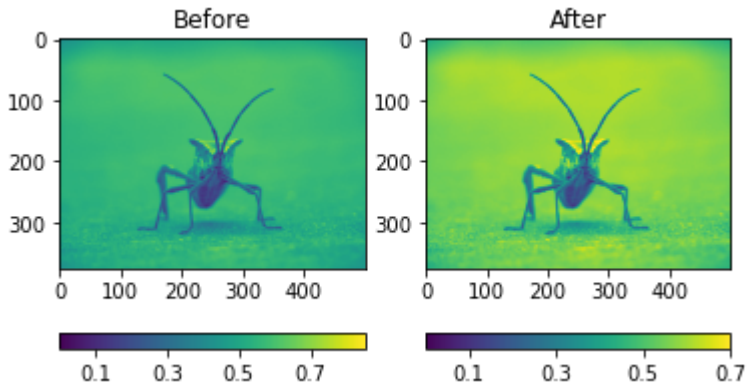
object

In [7]:



In [8]:
```
#You can also specify the clim using the returned object

#CREA DOS FIGURAS, DOS PLOTS
fig = plt.figure()
ax = fig.add_subplot(1, 2, 1)

#PRIMERA IMAGEN
imgplot = plt.imshow(lum_img)
ax.set_title('Before')
plt.colorbar(ticks=[0.1, 0.3, 0.5, 0.7], orientation='horizontal')#CREA L

#SEGUNDA IMAGEN
ax = fig.add_subplot(1, 2, 2)
imgplot = plt.imshow(lum_img)
imgplot.set_clim(0.0, 0.7) #HACE EL CLIP
ax.set_title('After')
plt.colorbar(ticks=[0.1, 0.3, 0.5, 0.7], orientation='horizontal')
```

Out[8]: <matplotlib.colorbar.Colorbar at 0x209864181c0>

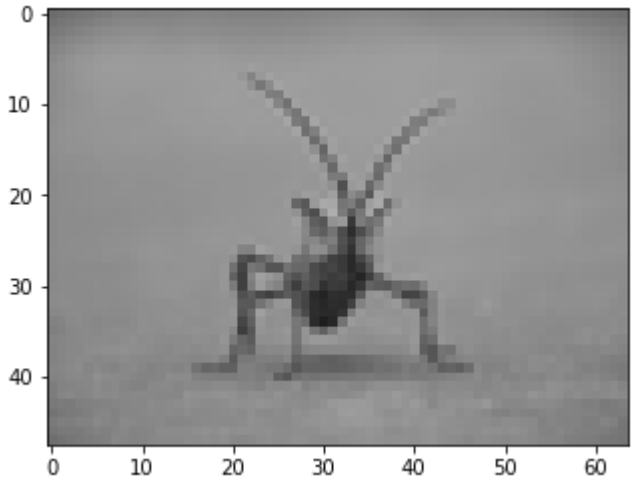


Array Interpolation schemes

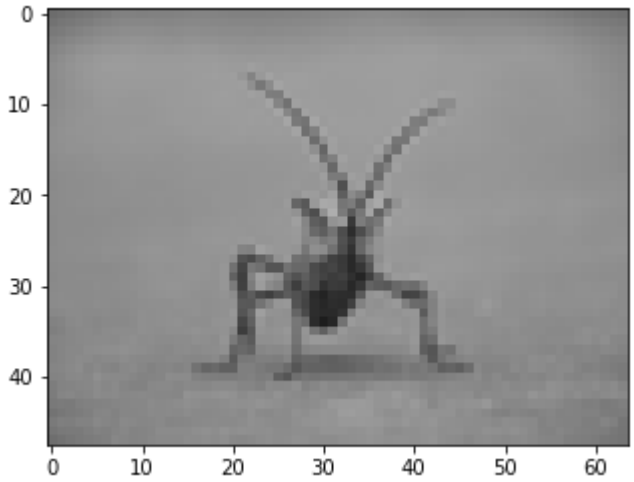
Interpolation calculates what the color or value of a pixel "should" be, according to different mathematical schemes. One common place that this happens is when you resize an image. The number of pixels change, but you want the same information. Since pixels are discrete, there's missing space. Interpolation is how you fill that space. This is why your images sometimes come out looking pixelated when you blow them up. The effect is more pronounced when the difference between the original image and the expanded image is greater. Let's take our image and shrink it. We're effectively discarding pixels, only keeping a select few. Now when we plot it, that data gets blown up to the size on your screen. The old pixels aren't there anymore, and the computer has to draw in pixels to fill that space.

We'll use the Pillow library that we used to load the image also to resize the image.

In [9]:
```
from PIL import Image

img = Image.open('stinkbug.png')
img.thumbnail((64, 64), Image.ANTIALIAS)  # resizes image in-place (HACE
                                          #PARA QUE FALTEN PIXELES POR REL
                                             #A HACERLO AUTOMÁTICAMENTE, P

imgplot = plt.imshow(img)
```



In [10]:
```
#NEAREST LO HACE SIN INTERPOLACIÓN
```



In [11]:
```
#BICUBIC INTERPOLATION
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

EXTRA

Cómo trazar ejes logarítmicos en Matplotlib: https://www.delftstack.com/es/howto/matplotlib/how-to-plot-logarithmic-axes-in-matplotlib/ (https://www.delftstack.com/es/howto/matplotlib/how-to-plot-logarithmic-axes-in-matplotlib/) (**¡EN GENERAL: PENSAR EN QUE LOS PLOTS SE PUEDEN HACER EN ESCALAS DIFERENTES!**)

Params en tiempo real: https://matplotlib-style-configurator.herokuapp.com/ (https://matplotlib-style-configurator.herokuapp.com/)

In [ ]: