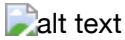
Images and Their Pixels Lab

In [1]:
```
# run this to import numpy and matplotlib
import numpy as np
import matplotlib.pyplot as plt
```

Create a numpy array representing an RGB image shown below. The dimensions are 100 pixels by 100 pixels. Hint: Use np.zeros(shape_tuple, dtype="uint8").

You'll need to use slicing to initialize your array. For example: array[: ,: ,0] = 255 would set the Red component of all pixels to its maximum value of 255.


alt text

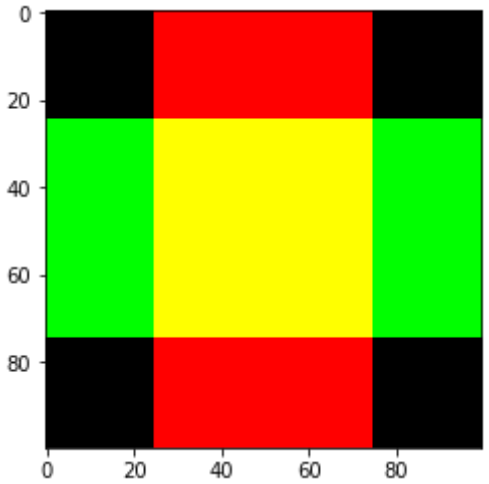
In [2]:
```
a = np.zeros((100,100,3),dtype="uint8")
print(a.shape)
```

```
(100, 100, 3)
```

In [3]:
```
a[25:75,:,1] = 255
a[:,25:75,0] = 255
```

In [4]:
```
fig, ax = plt.subplots()
ax.imshow(a)
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x117a92da0>
```



Use the code below to read in an image. The filename is "sunflower.png".

```
img = plt.imread(filename_str)
```

In [5]:
```
img = plt.imread("sunflower.png")
```

What is the width and height of the image? The type of data? (img.dtype).

In [6]:
```
img.shape
```

Out[6]:
```
(113, 150, 3)
```

In [7]:
```
img.shape[0]
```

Out[7]:
```
113
```

In [8]:
```
img.dtype
```

Out[8]:
```
dtype('float32')
```

In [9]:
```
img[100, 110, :]
```

Out[9]:
```
array([0.3647059 , 0.36078432, 0.09411765], dtype=float32)
```

Print out any pixel. Note that the pixel values are 32-bit floats in the range [0.0, 1.0].

In [10]:
```
img[100, 110, :]
```

Out[10]:
```
array([0.3647059 , 0.36078432, 0.09411765], dtype=float32)
```

Use the following code to display the image of the flower.

```
fig, ax = plt.subplots()
ax.imshow(img)
```

In [11]:
```
fig, ax = plt.subplots()
ax.imshow(img)
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x119f15f98>
```

Estimate the location of a yellow petal pixel and print out its RGB components. You should see high red(close to 1.0) and green and low blue(close to 0.0).

In [12]:
```
print(img[20,80,:])
```

```
[0.91764706 0.7254902  0.16078432]
```

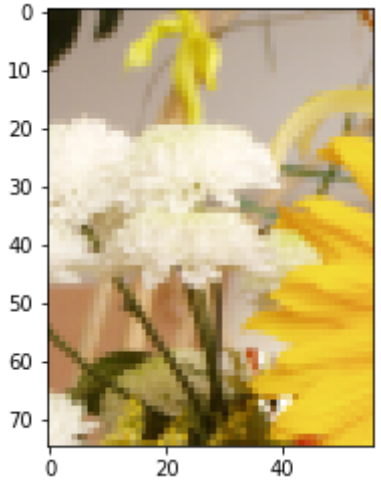
Use slicing to create a new img(numpy array) consists of the top left quarter of the image. Display this new image.

In [13]:
```
width = img.shape[1]//2
height = img.shape[0]//2

top_left = img[:width,:height,:]


fig, ax = plt.subplots()
ax.imshow(top_left)
```

Out[13]: <matplotlib.figure.AxesImage at 0x11a3e2128>

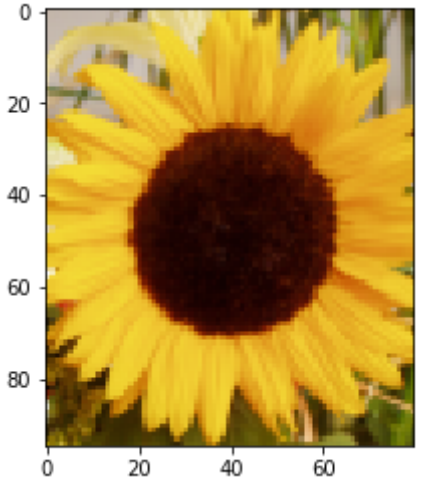


In [14]:
```
# USE THIS OUTPUT CELL AS A REFERENCE
# DO NOT WRITE CODE IN OR RUN THIS CELL
```

Use slicing to crop out only the sunflower, removing all of the background. Try to get as close as possible. Display the new image.

In [15]:
```
flower = img[10:105,40:120,:]
fig, ax = plt.subplots()
ax.imshow(flower)
```

Out[15]: <matplotlib.image.AxesImage at 0x11a3d6cc0>



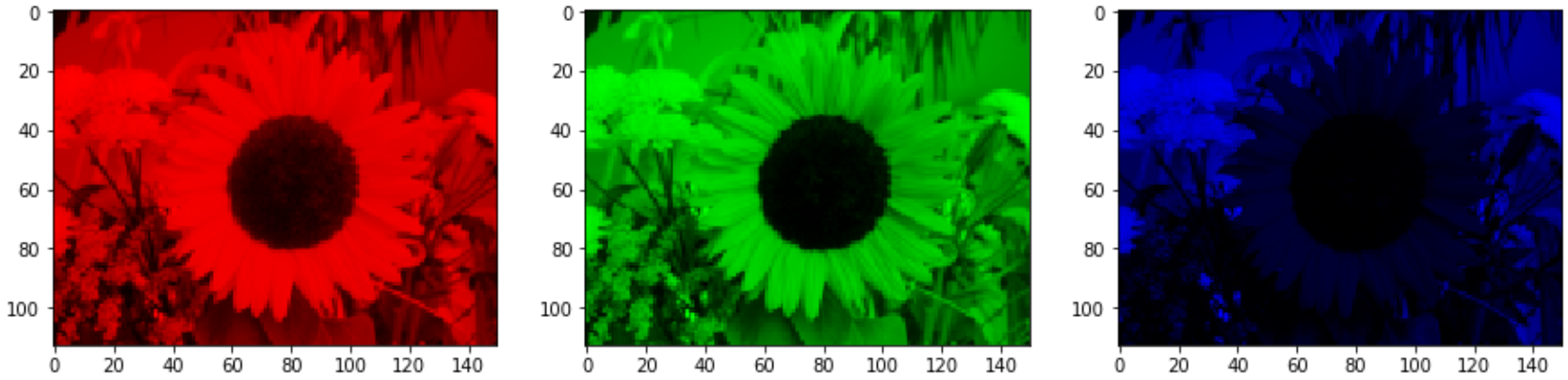
In [16]:
```
# USE THIS OUTPUT CELL AS A REFERENCE
# DO NOT WRITE CODE IN OR RUN THIS CELL
```

Use the code from the Lecture to extract and display each Red, Green and Blue components on separate Axes objects for the sunflower.

In [17]:
```
# use np.zeros(shape, dtype)
red = np.zeros(img.shape, dtype=img.dtype)
red[:,:,0] = img[:,:,0]
green = np.zeros(img.shape, dtype=img.dtype)
green[:,:,1] = img[:,:,1]
blue = np.zeros(img.shape, dtype=img.dtype)
blue[:,:,2] = img[:,:,2]
```

In [18]:
```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))
axes[0].imshow(red)
axes[1].imshow(green)
axes[2].imshow(blue)
```

Out[18]: <matplotlib.image.AxesImage at 0x11a75ee48>



In [19]:
```
# USE THIS OUTPUT CELL AS A REFERENCE
# DO NOT WRITE CODE IN OR RUN THIS CELL
```

Use the average method to convert the sunflower to grayscale. When display the image, use the keyword argument cmap="gray" in imshow() to specify the grayscale colormap.

In [20]:
```
ave = (img[:,:,0] + img[:,:,1] + img[:,:,2])/3
```

In [21]:
```
fig, ax = plt.subplots()
ax.imshow(ave, cmap="gray")
```

Out[21]: <matplotlib.image.AxesImage at 0x11a9bbba8>

Use the luminosity method to create the grayscale image.

In [22]:
```
lum = img[:,:,0]*0.21 + img[:,:,1]*0.72 + img[:,:,2]+0.07
fig, ax = plt.subplots()
ax.imshow(lum, cmap="gray")
```

Out[22]: <matplotlib.image.AxesImage at 0x11aa3d278>

To tint an image is to mix its colors with white. This will increase the lightness of the image. Write a Python function, which takes an image and a percentage value as a parameter. Setting 'percentage' to 0 will not change the image, setting it to one means that the image will be completely whitened.

For example, suppose a pixel with RGB components of [0.80, 0.60, 0.40]. Tinting it by 25% means that the pixel is now [0.85, 0.70, 0.55].

Hint: Use np.ones(shape). The formula for one pixel in the above example is:

tinted_pixel = [0.80, 0.60, 0.40] + ([1.0, 1.0, 1.0] - [0.80, 0.60, 0.40]) * 0.25

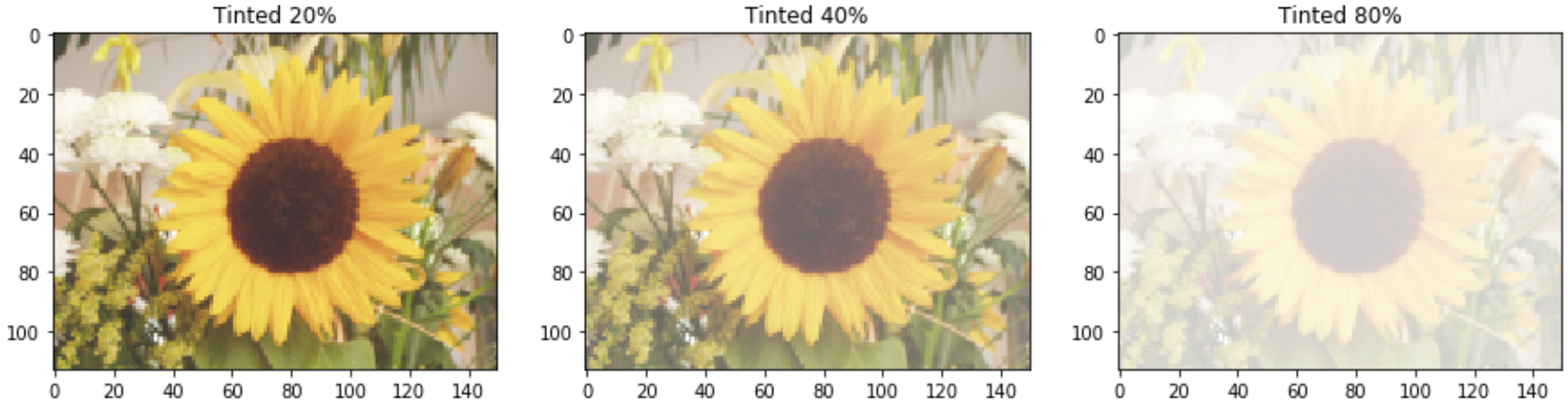
In [23]:
```
def tint(imag, percent):
    """
    imag: the image which will be tinted
    percent: a value between 0 (image will remain unchanged
             and 1 (image will completely white)
    """
    # just add one line of code.
    tinted_imag = img + (np.ones(img.shape) - img)*percent
    return tinted_imag
```

In [ ]:
```
```

If your tint() function is written correctly above, run the code below to see the tinting at three different levels: 20%, 40%, 80%. Play around with different tint levels.

In [24]:
```
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15,5))
tinted_10 = tint(img, 0.20)
tinted_40 = tint(img, 0.40)
tinted_80 = tint(img, 0.80)
axs[0].imshow(tinted_10)
axs[0].set_title("Tinted 20%")
axs[1].imshow(tinted_40)
axs[1].set_title("Tinted 40%")
axs[2].imshow(tinted_80)
axs[2].set_title("Tinted 80%")
```

Out[24]:
```
Text(0.5, 1.0, 'Tinted 80%')
```



In [ ]: