In [4]:

```
# Image and Audio Visualization
# You can read digital images with Matplotlib, which supports many image formats,
# although you do have to install a library called pillow. Install pillow as shown here:

!pip3 install Pillow
```

```
Requirement already satisfied: Pillow in c:\python\lib\site-packages (7.2.0)
```

In [5]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
# You can read digital images with the function imread() on Windows as follows:
img = plt.imread("nature.jpg")
# Let’s see the contents of the variable now, as shown here:
print(img)
```

```
[[[111 140  76]
  [ 89 114  56]
  [ 36  53   9]
  ...
  [ 10  21   0]
  [ 35  46  14]
  [ 39  50  18]]

 [[ 58  87  21]
  [ 58  84  23]
  [ 31  50   4]
  ...
  [ 38  49  19]
  [ 47  58  26]
  [ 36  47  15]]

 [[125 157  84]
  [ 94 123  57]
  [ 35  58   6]
  ...
  [ 31  42  12]
  [ 32  43  13]
  [ 25  36   6]]

 ...

 [[ 14  21   5]
  [ 24  31  15]
  [ 38  45  29]
  ...
  [  1   1   1]
  [  1   1   1]
  [  1   1   1]]

 [[ 51  59  36]
  [ 55  63  40]
  [ 35  43  22]
  ...
  [  1   1   1]
  [  1   1   1]
  [  1   1   1]]

 [[ 17  25   1]
  [ 50  58  34]
  [ 23  31   8]
  ...
  [  2   2   2]
  [  2   2   2]
  [  2   2   2]]]
```

In [6]:

```
# The output is an Ndarray after all. We can confirm this with the following code:
type(img)
```

Out[6]:

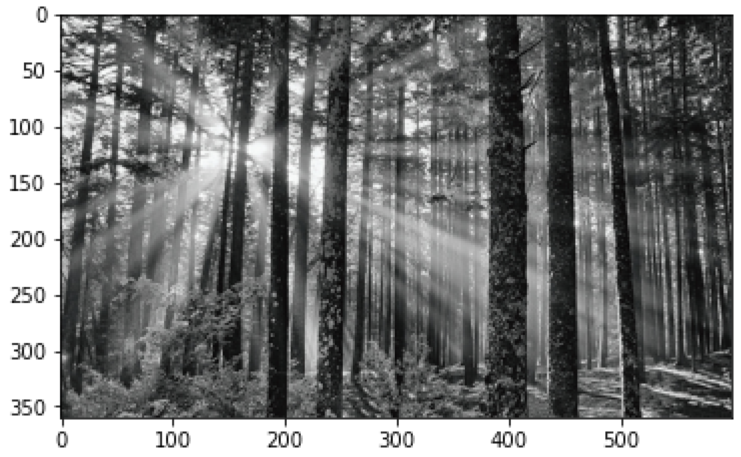
```
numpy.ndarray
```

In [7]:

```
# To learn more about the image, you can check the properties of the Ndarray that
# is storing the image data. A color image is stored as a 3D matrix, and each individual
# dimension of that matrix is used to visualize the intensity of the color channel. Color
# images are read and stored in red, green, blue (RGB) format. Since there are no colors in
# grayscale images, there is only a single plane (a 2D matrix) that stores the intensities of
# the grayscale values.
# You can use the routine imshow() to show any Ndarray as an image as follows:

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.imshow(img)
plt.show()
```

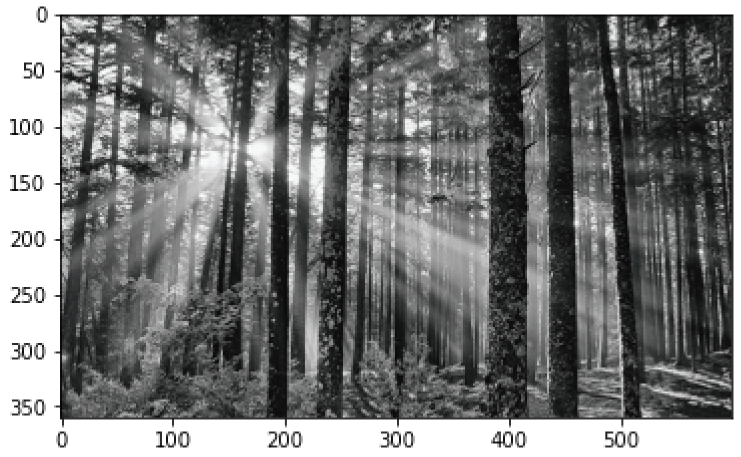
In [8]:

```
# This is a color image. The Matplotlib library automatically detects that the image has
# multiple channels and shows it as a color image. However, it goofs up a little bit when
# show grayscale images.

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
img1 = plt.imread("nature1.jpg")
plt.imshow(img1)
plt.show()
```



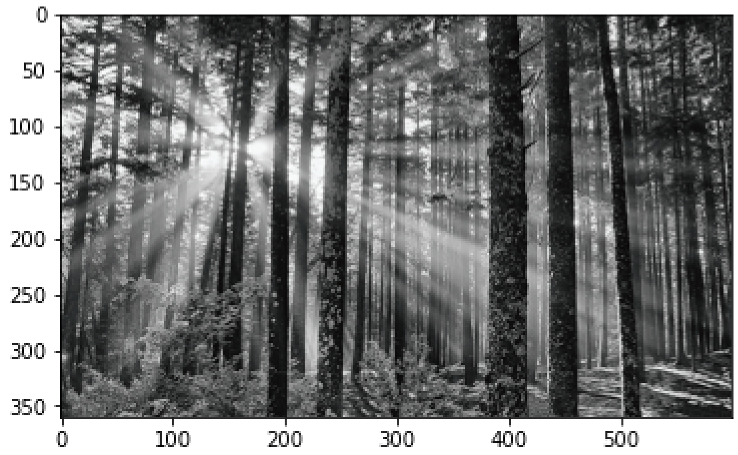
In [9]:

```
# The image data is interpreted correctly, but there seems to be some problem with
# the color. For grayscale images, Matplotlib uses the default color map, so you have to
# manually specify the color map as follows:

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
img1 = plt.imread("nature1.jpg")
plt.imshow(img1, cmap = 'gray')
plt.show()
```



In [10]:

```
# A color map is a matrix of values defining the colors for visualizations. Let’s try
# another color map for the image, as shown here:

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
img1 = plt.imread("nature1.jpg")
plt.imshow(img1, cmap = 'cool')
plt.show()
```



In [11]:

```
# You can display a list of color maps in the current version of Matplotlib by using the
# following statement:

plt.colormaps()
```

Out[11]:

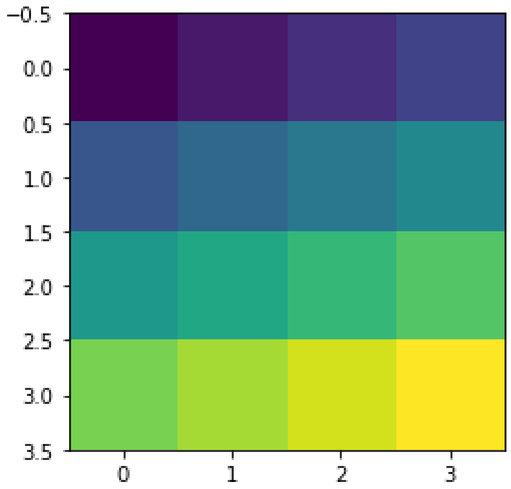
```
['Accent',
 'Accent_r',
 'Blues',
 'Blues_r',
 'BrBG',
 'BrBG_r',
 'BuGn',
 'BuGn_r',
 'BuPu',
 'BuPu_r',
 'CMRmap',
 'CMRmap_r',
 'Dark2',
 'Dark2_r',
 'GnBu',
 'GnBu_r',
 'Greens',
 'Greens_r',
 'Greys',
```

In [12]:

```
# Image Masking
# You can mask the areas of an image with a circle as follows:

import matplotlib.patches as patches
fig, ax = plt.subplots()
im = ax.imshow(img)
patch = patches.Circle((245, 200),radius=200,transform=ax.transData)
im.set_clip_path(patch)
ax.axis('off')
plt.show()

# In this code example, we are creating a circle with the routine Circle() at the XY
# co-ordinates 245, 200. The radius is 200 pixels. Also, we are clipping the image with the
# circle using the routine set_clip_path() and showing it.
```

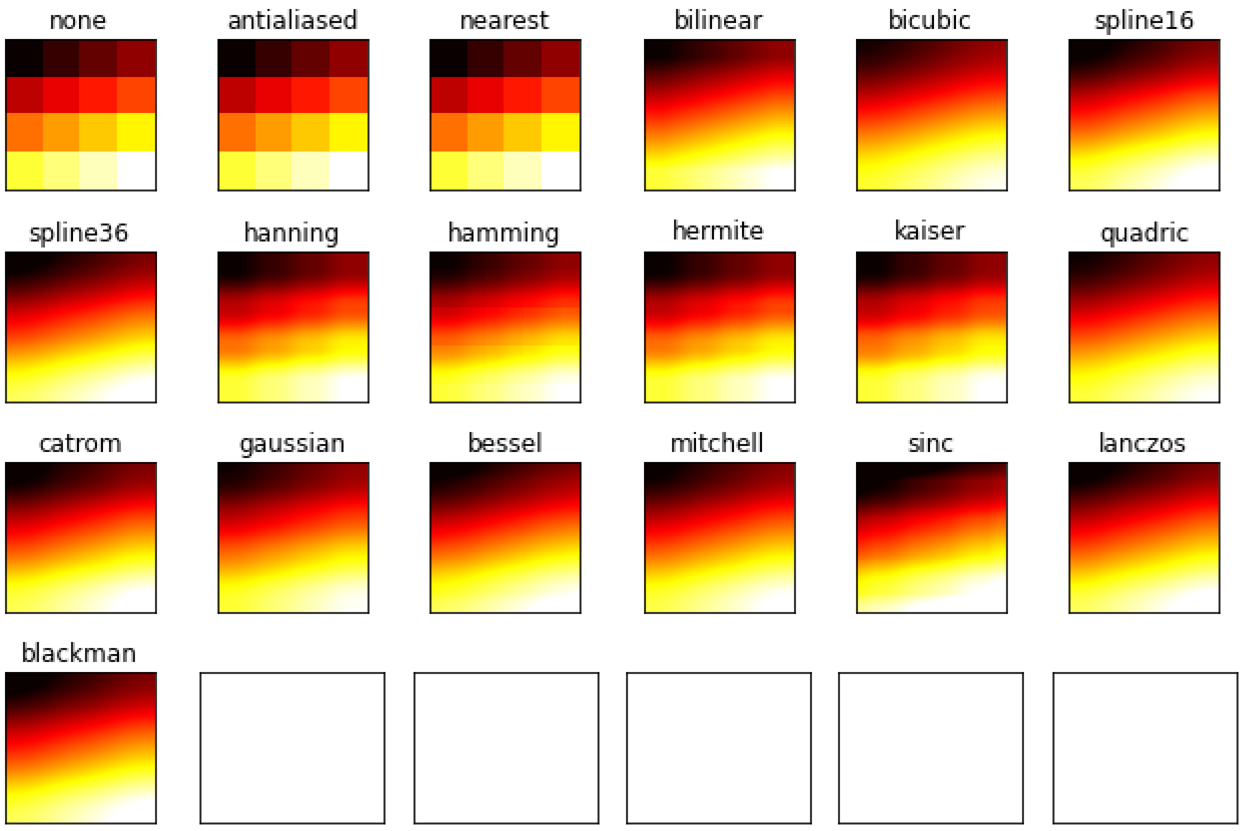
In [13]:

```
# Interpolation Methods
# You can show a simple NumPy Ndarray as an image as follows:

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
img2 = [[1, 2, 3, 4],
        [5, 6, 7, 8],
        [9, 10, 11, 12],
        [13, 14, 15, 16]]
plt.imshow(img2)
plt.show()
```



In [14]:

```
# The image is using no interpolation method for visualization. We can demo
# interpolation methods as follows:

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
methods = ['none', 'antialiased', 'nearest', 'bilinear',
           'bicubic', 'spline16', 'spline36', 'hanning',
           'hamming', 'hermite', 'kaiser', 'quadric',
           'catrom', 'gaussian', 'bessel', 'mitchell',
           'sinc', 'lanczos', 'blackman']
fig, axs = plt.subplots(nrows=4, ncols=6, figsize=(9, 6),
                        subplot_kw={'xticks': [], 'yticks': []})
for ax, interp_method in zip(axs.flat, methods):
    ax.imshow(img2, interpolation=interp_method, cmap='hot')
    ax.set_title(str(interp_method))
plt.tight_layout()
plt.show()
```



In [15]:

```
# Audio Visualization
# You can use Matplotlib to visualize audio. You just need the SciPy library to read an
# audio file and store that data to an Ndarray. Let’s install it, as shown here:

!pip3 install scipy
```

```
Requirement already satisfied: scipy in c:\python\lib\site-packages (1.4.1)
Requirement already satisfied: numpy>=1.13.3 in c:\python\lib\site-packages (from scipy) (1.19.4)
```

In [16]:

```
# Let’s import all the required libraries, as shown here:

%matplotlib inline
import matplotlib.pyplot as plt
from scipy.io import wavfile
# Let’s read an audio file now. I am reading a WAV file as follows:
samplerate, data = wavfile.read('samplesound.wav')
# Let’s see the sampling rate of the music, as shown here:
print(samplerate)
```

```
44100
```

In [17]:

```
# This (44.1 kHz) is a common sampling rate.
# You can also display the data as follows:

%matplotlib inline
import matplotlib.pyplot as plt
from scipy.io import wavfile
samplerate, data = wavfile.read('samplesound.wav')
print(samplerate)
print(data)
```

```
44100
[[0 0]
 [0 0]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

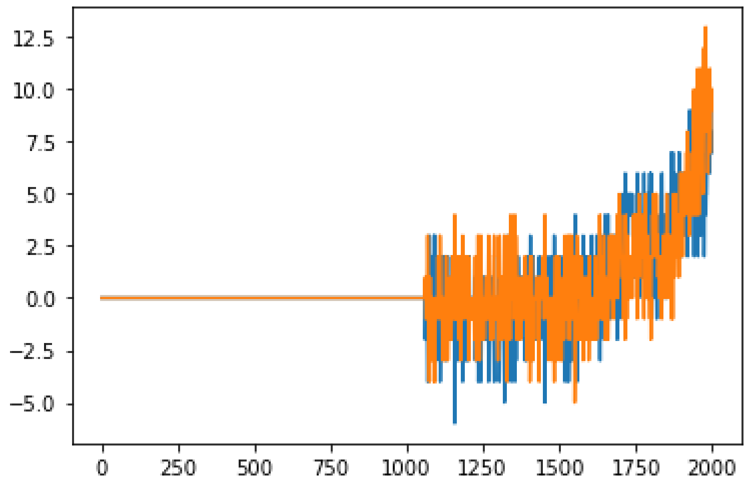
In [18]:

```
# You can check the properties of the audio as follows:

print(type(data))
print(data.shape)
print(data.ndim)
print(data.dtype)
print(data.size)
print(data.nbytes)
```

```
<class 'numpy.ndarray'>
(35854848, 2)
2
int16
71709696
143419392
```

In [19]:

```
# The audio data is retrieved and stored in the NumPy, as you have seen. It is stored
# in a 2D matrix. Suppose that there are N data points (also known as sample points) for
# the audio data; then the size of the NumPy array is N×2. As you can see, the audio has
# two channels, left and right. So, each channel in stored in a separate array of size N, and
# thus we have N×2. This is known as stereo audio. In this example, we have 2,601,617
# points (samples). Each point or sample is represented using a pair of integers of 16 bits
# (2 bytes). Thus, each sample needs four bytes. So, we can compute the total raw memory
# required for storing the audio data by multiplying the sample size by 4. When we
# visualize audio, we show the value of both channels of the sample. Let’s visualize the first
# 2,000 data points as follows:

%matplotlib inline
import matplotlib.pyplot as plt
from scipy.io import wavfile
samplerate, data = wavfile.read('samplesound.wav')
plt.plot(data[:2000])
plt.show()
```



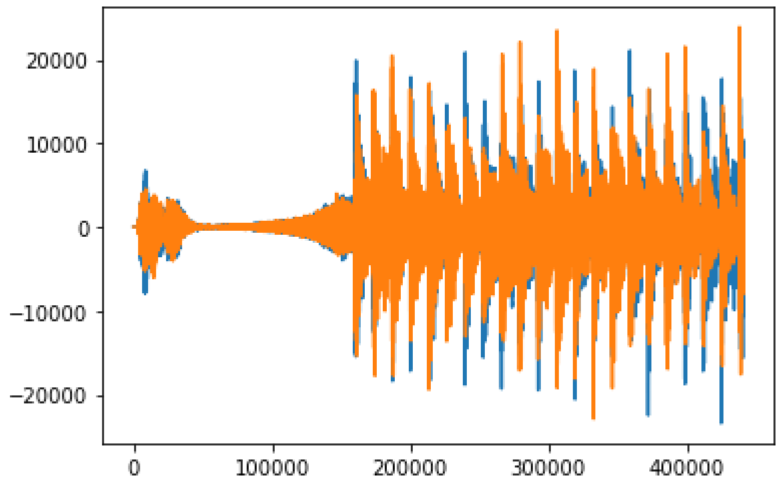
In [20]:

```
# You can check the number of audio samples as follows:

samples = data.shape[0]
print(samples)
```

```
35854848
```

In [21]:

```
# You can create a different visualization of the data as follows:

plt.plot(data[:10*samplerate])
plt.show()
```



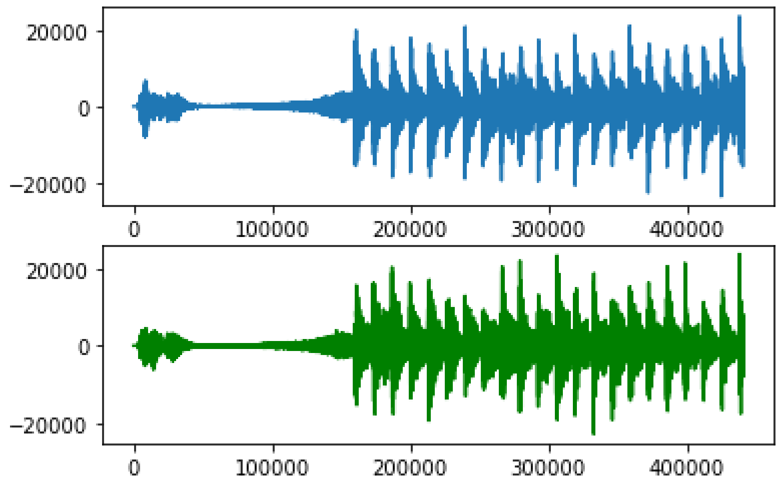
In [22]:

```
# Let’s separate the data for both channels as follows:

channel1 = data[:, 0]
channel2 = data[:, 1]
print(channel1, channel2)
```

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

In [23]:

```
# Let’s visualize the data as follows:

plt.subplot(2, 1, 1)
plt.plot(channel1[:10*samplerate])
plt.subplot(2, 1, 2)
plt.plot(channel2[:10*samplerate], c='g')
plt.show()
```


20000
0
−20000
0
100000
200000
300000
400000
20000
0
−20000
0
100000
200000
300000
400000