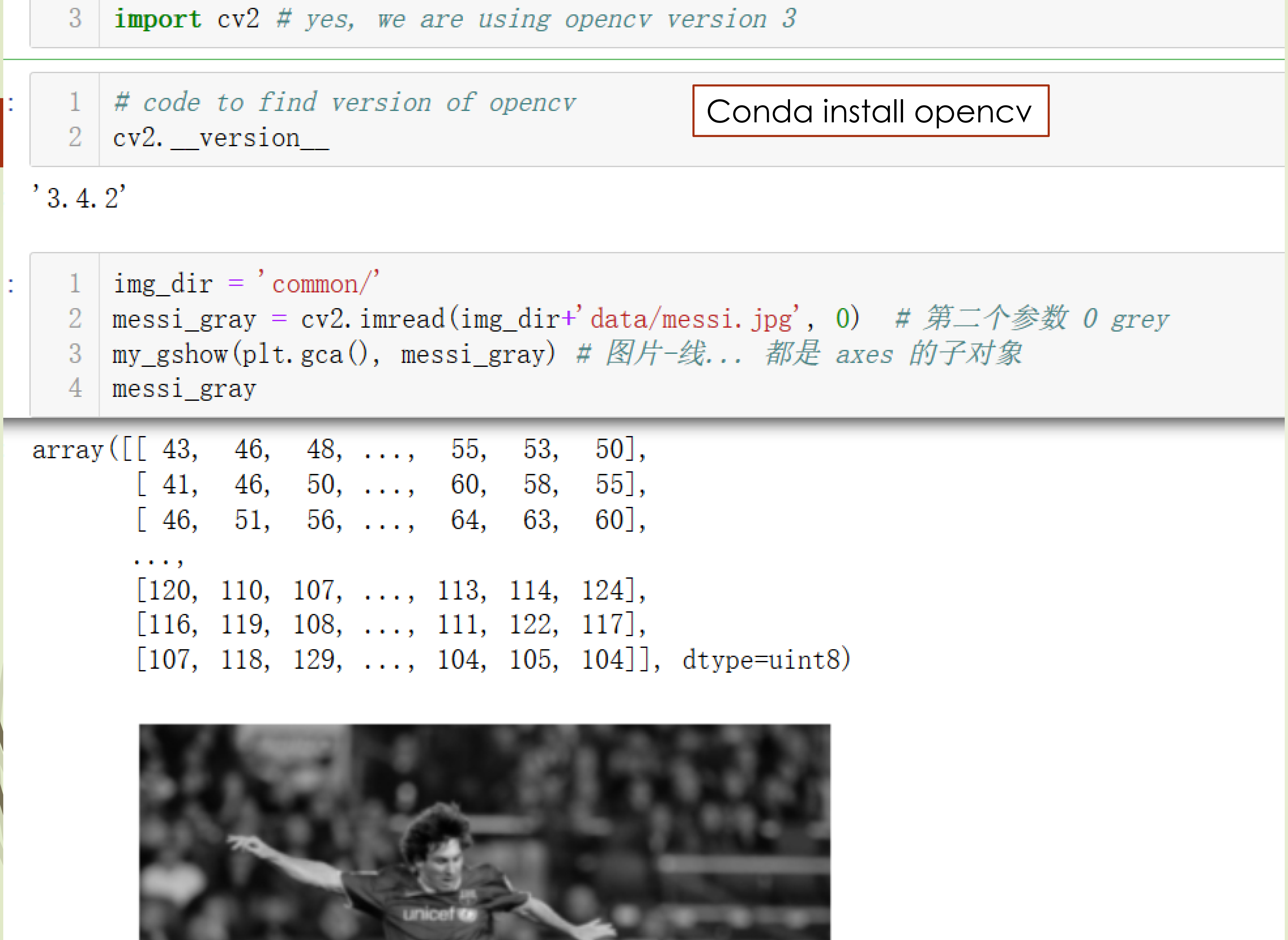
Python 图像处理基础

胡俊峰 2020/04/20

北京大学信息科学技术学院

内容

- 安装CV软件包
- Python的图像格式与存储方案
- 简单特征统计与直方图变换
- 卷积、角点
- 图像特征提取



```
import cv2 # yes, we are using opencv version 3
```

```
# code to find version of opencv
cv2.__version__
```

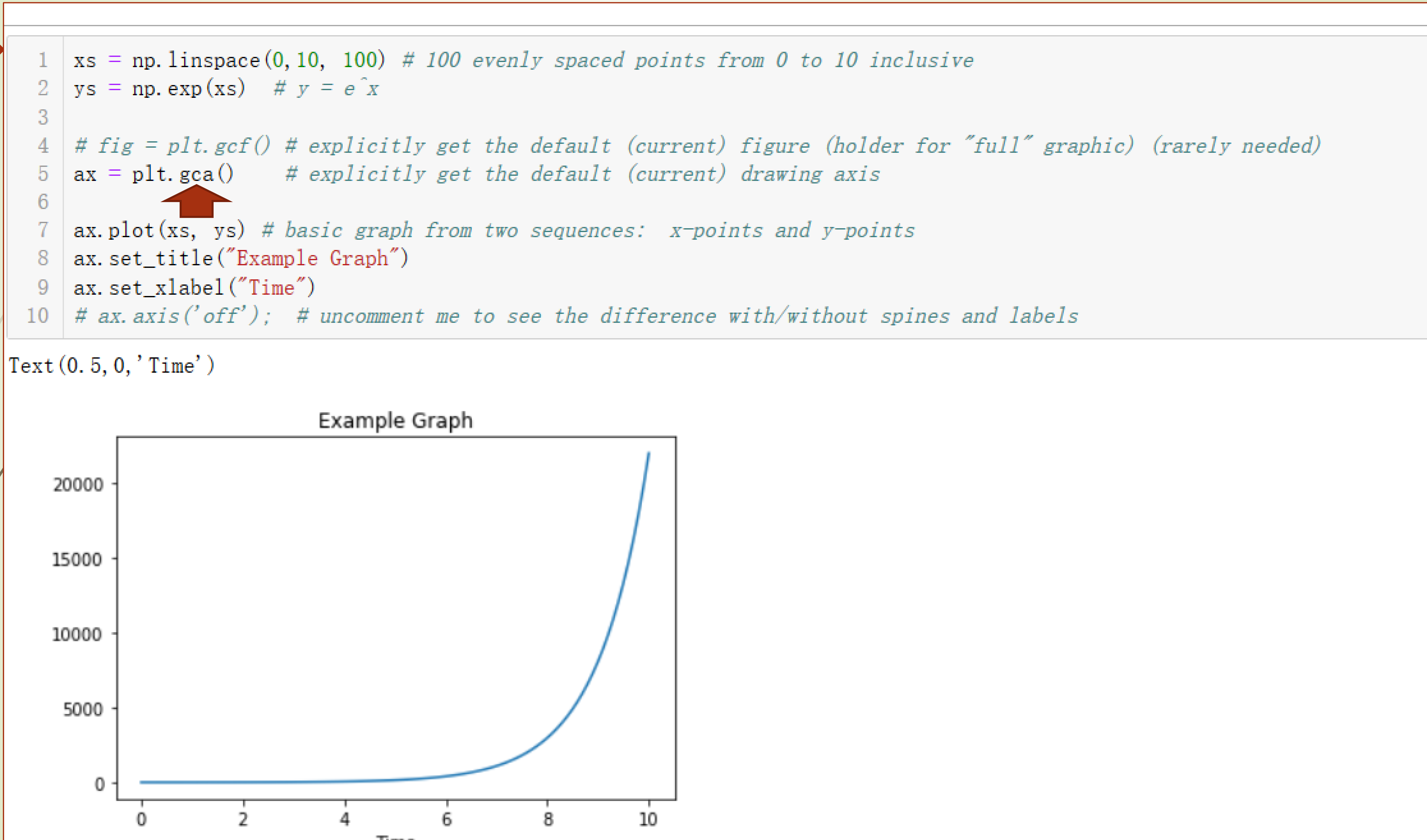
Conda install opencv

```
'3.4.2'
```

```
img_dir = 'common/'
messi_gray = cv2.imread(img_dir+'data/messi.jpg', 0)  # 第二个参数 0 grey
my_gshow(plt.gca(), messi_gray) # 图片-线... 都是 axes 的子对象
messi_gray
```

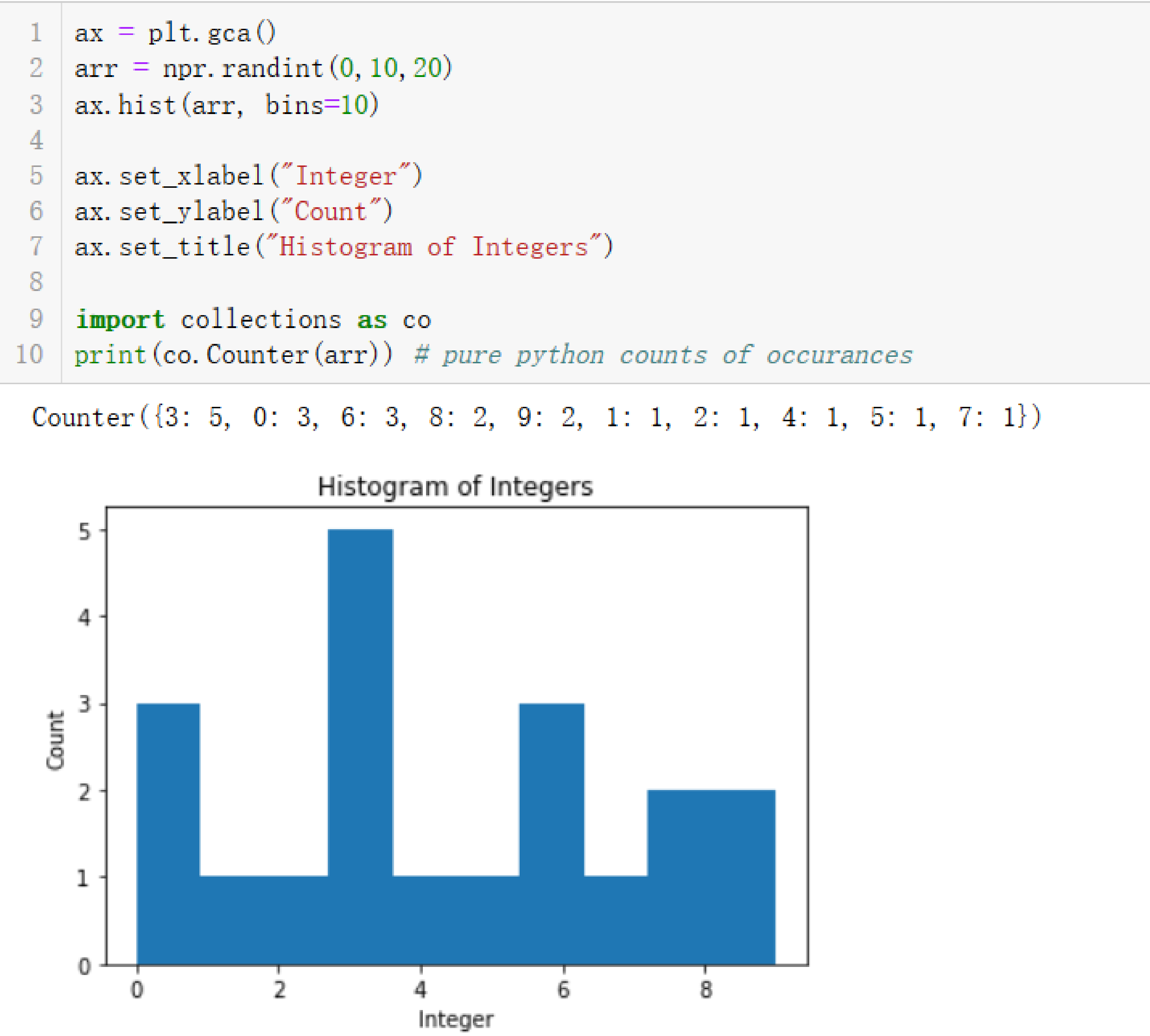
```
array([[ 43,  46,  48, ...,  55,  53,  50],
       [ 41,  46,  50, ...,  60,  58,  55],
       [ 46,  51,  56, ...,  64,  63,  60],
       ...,
       [120, 110, 107, ..., 113, 114, 124],
       [116, 119, 108, ..., 111, 122, 117],
       [107, 118, 129, ..., 104, 105, 104]], dtype=uint8)
```

Matplob get current axis



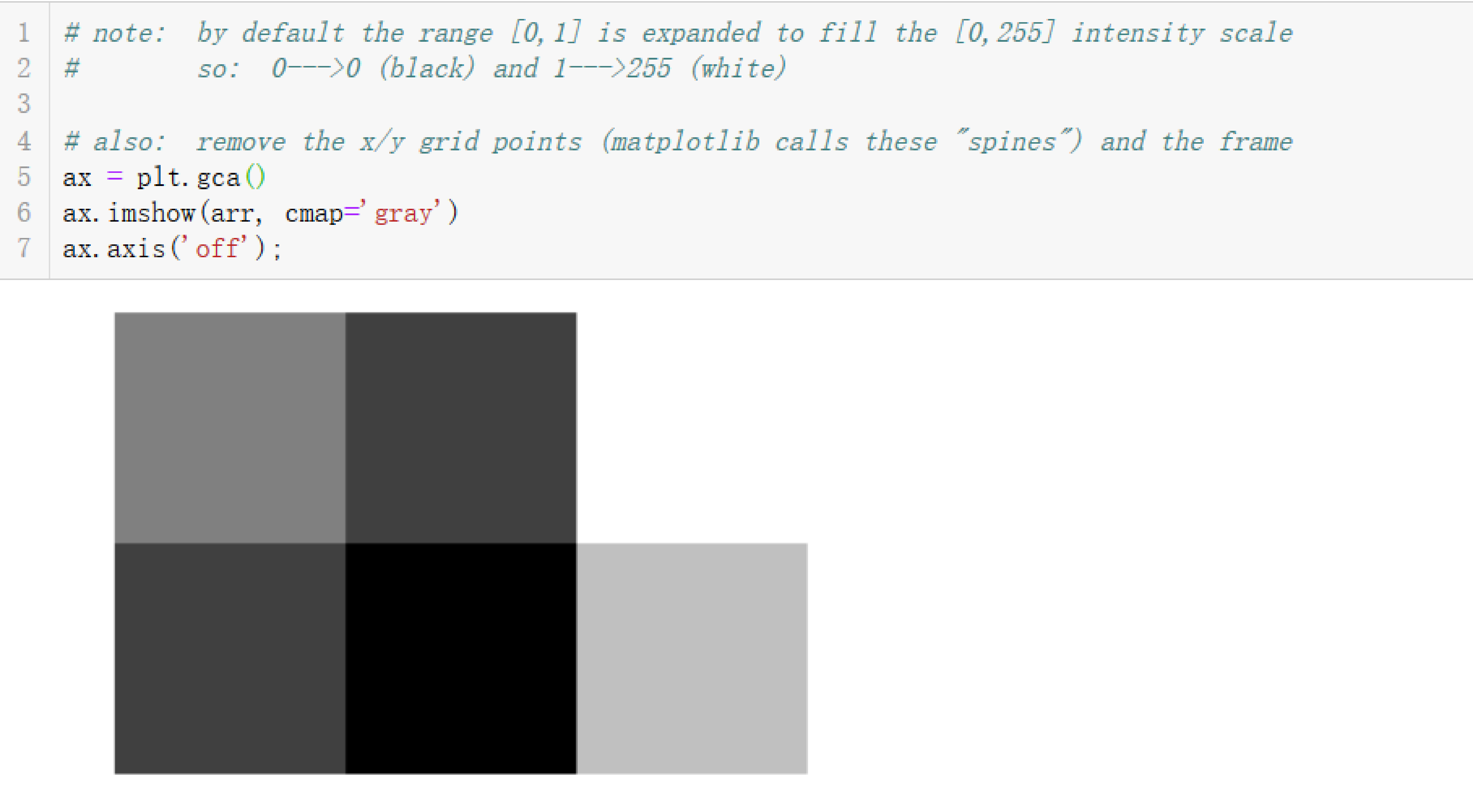
```
xs = np.linspace(0,10, 100) # 100 evenly spaced points from 0 to 10 inclusive
ys = np.exp(xs)  # y = e^x

# fig = plt.gcf() # explicitly get the default (current) figure (holder for "full" graphic) (rarely needed)
ax = plt.gca()    # explicitly get the default (current) drawing axis

ax.plot(xs, ys) # basic graph from two sequences:  x-points and y-points
ax.set_title("Example Graph")
ax.set_xlabel("Time")
# ax.axis('off');  # uncomment me to see the difference with/without spines and labels
```

```
Text(0.5,0,'Time')
```



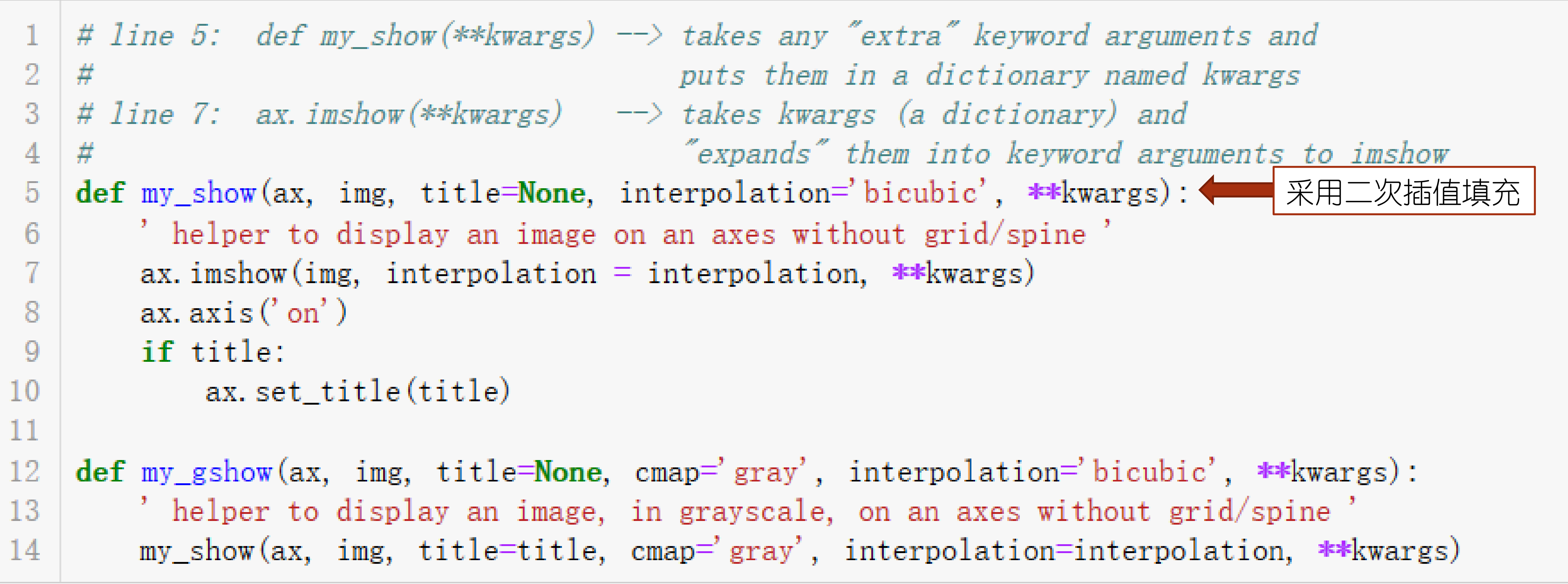
```python
ax = plt.gca()
arr = npr.randint(0,10,20)
ax.hist(arr, bins=10)

ax.set_xlabel("Integer")
ax.set_ylabel("Count")
ax.set_title("Histogram of Integers")

import collections as co
print(co.Counter(arr)) # pure python counts of occurances
```

```
Counter({3: 5, 0: 3, 6: 3, 8: 2, 9: 2, 1: 1, 2: 1, 4: 1, 5: 1, 7: 1})
```



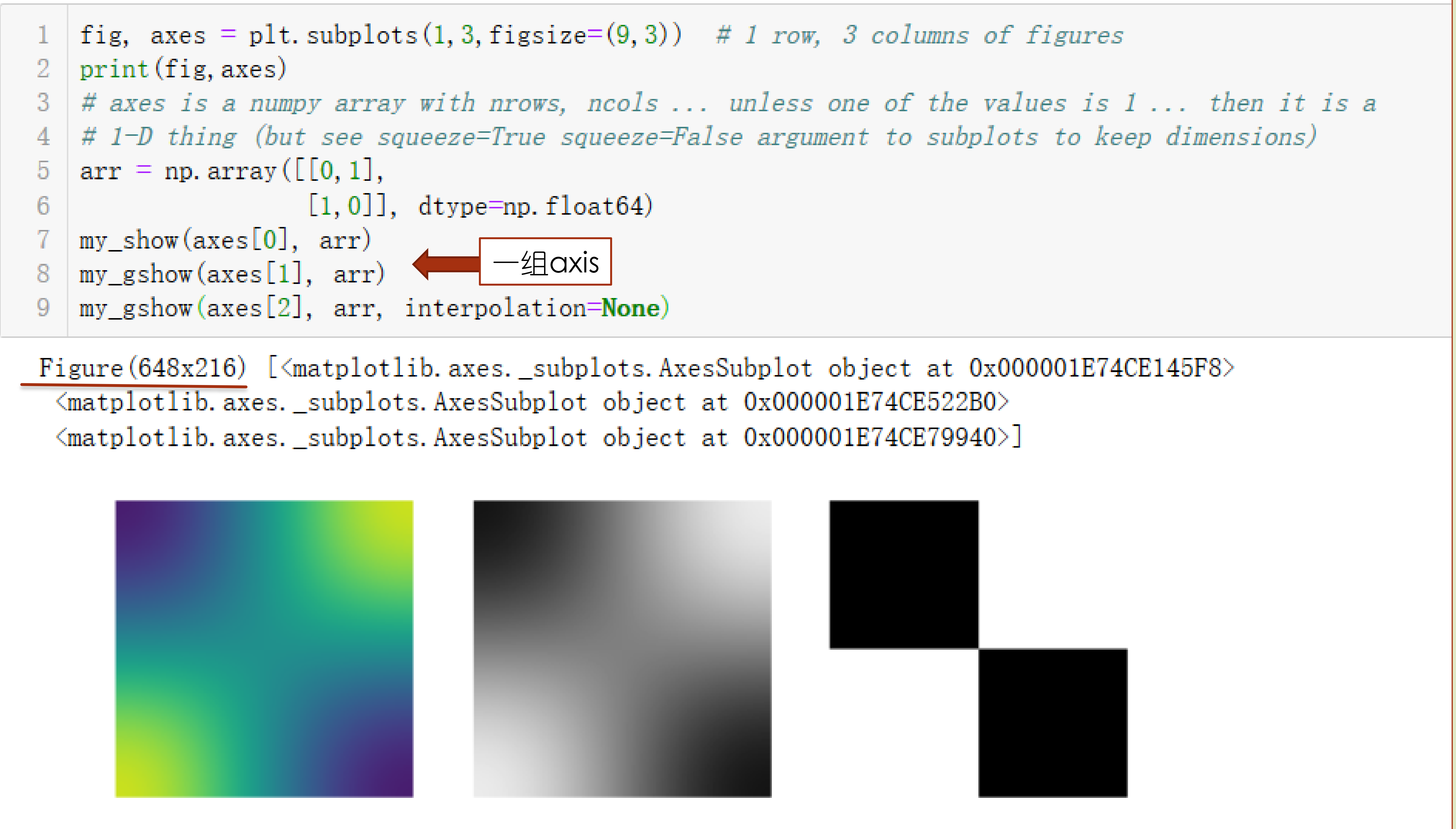
```
# note:  by default the range [0,1] is expanded to fill the [0,255] intensity scale
#        so:  0--->0 (black) and 1--->255 (white)

# also:  remove the x/y grid points (matplotlib calls these "spines") and the frame
ax = plt.gca()
ax.imshow(arr, cmap='gray')
ax.axis('off');
```

定义两个显示用函数



```
 1  # line 5:  def my_show(**kwargs) --> takes any ″extra″ keyword arguments and
 2  #                                    puts them in a dictionary named kwargs
 3  # line 7:  ax.imshow(**kwargs)    --> takes kwargs (a dictionary) and
 4  #                                    ″expands″ them into keyword arguments to imshow
 5  def my_show(ax, img, title=None, interpolation=’bicubic’, **kwargs):
 6      ’ helper to display an image on an axes without grid/spine ’
 7      ax.imshow(img, interpolation = interpolation, **kwargs)
 8      ax.axis(’on’)
 9      if title:
10          ax.set_title(title)
11
12  def my_gshow(ax, img, title=None, cmap=’gray’, interpolation=’bicubic’, **kwargs):
13      ’ helper to display an image, in grayscale, on an axes without grid/spine ’
14      my_show(ax, img, title=title, cmap=’gray’, interpolation=interpolation, **kwargs)
```

采用二次插值填充

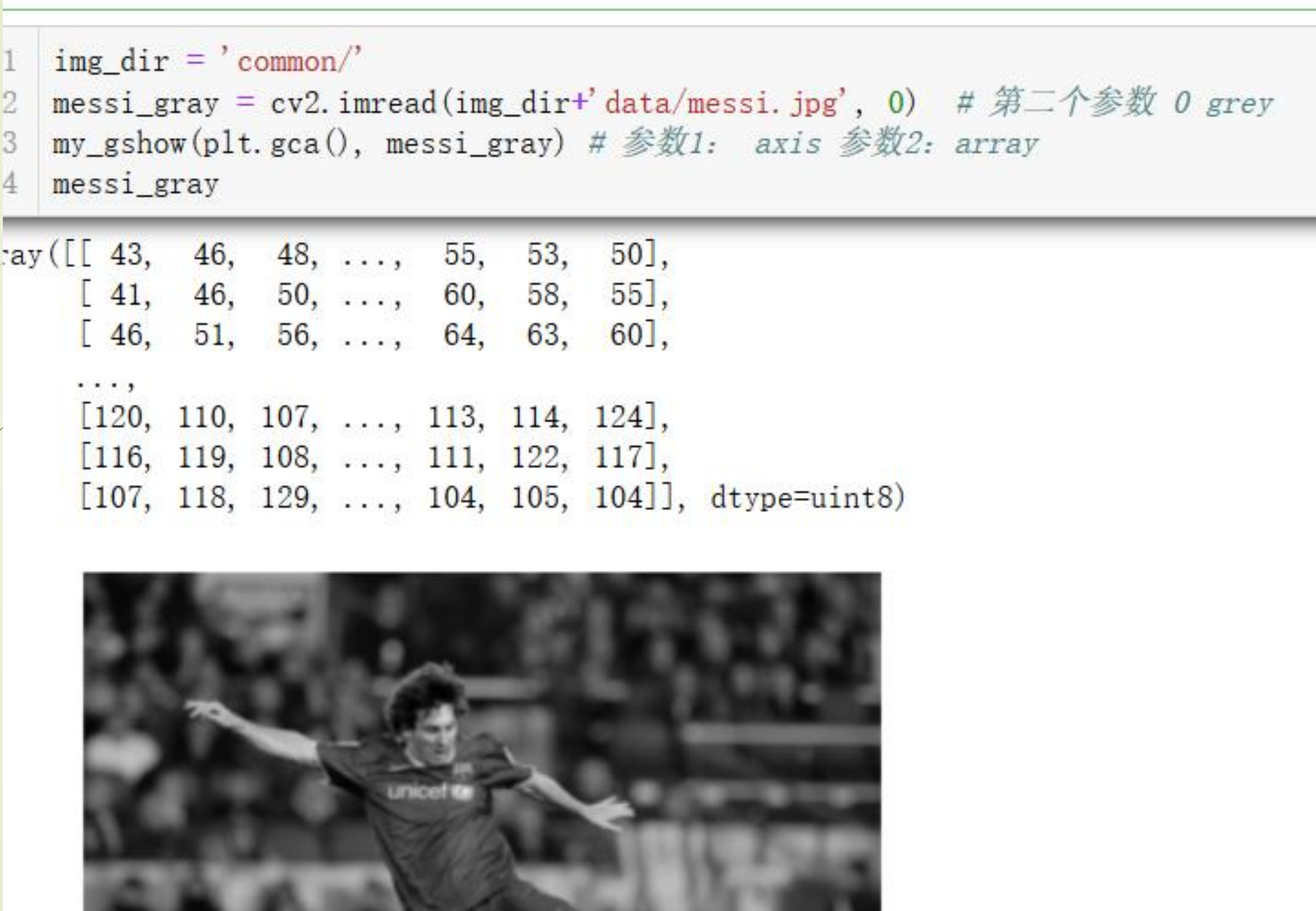


```
fig, axes = plt.subplots(1,3,figsize=(9,3))  # 1 row, 3 columns of figures
print(fig,axes)
# axes is a numpy array with nrows, ncols ... unless one of the values is 1 ... then it is a
# 1-D thing (but see squeeze=True squeeze=False argument to subplots to keep dimensions)
arr = np.array([[0,1],
                [1,0]], dtype=np.float64)
my_show(axes[0], arr)
my_gshow(axes[1], arr)
my_gshow(axes[2], arr, interpolation=None)
```

一组axis

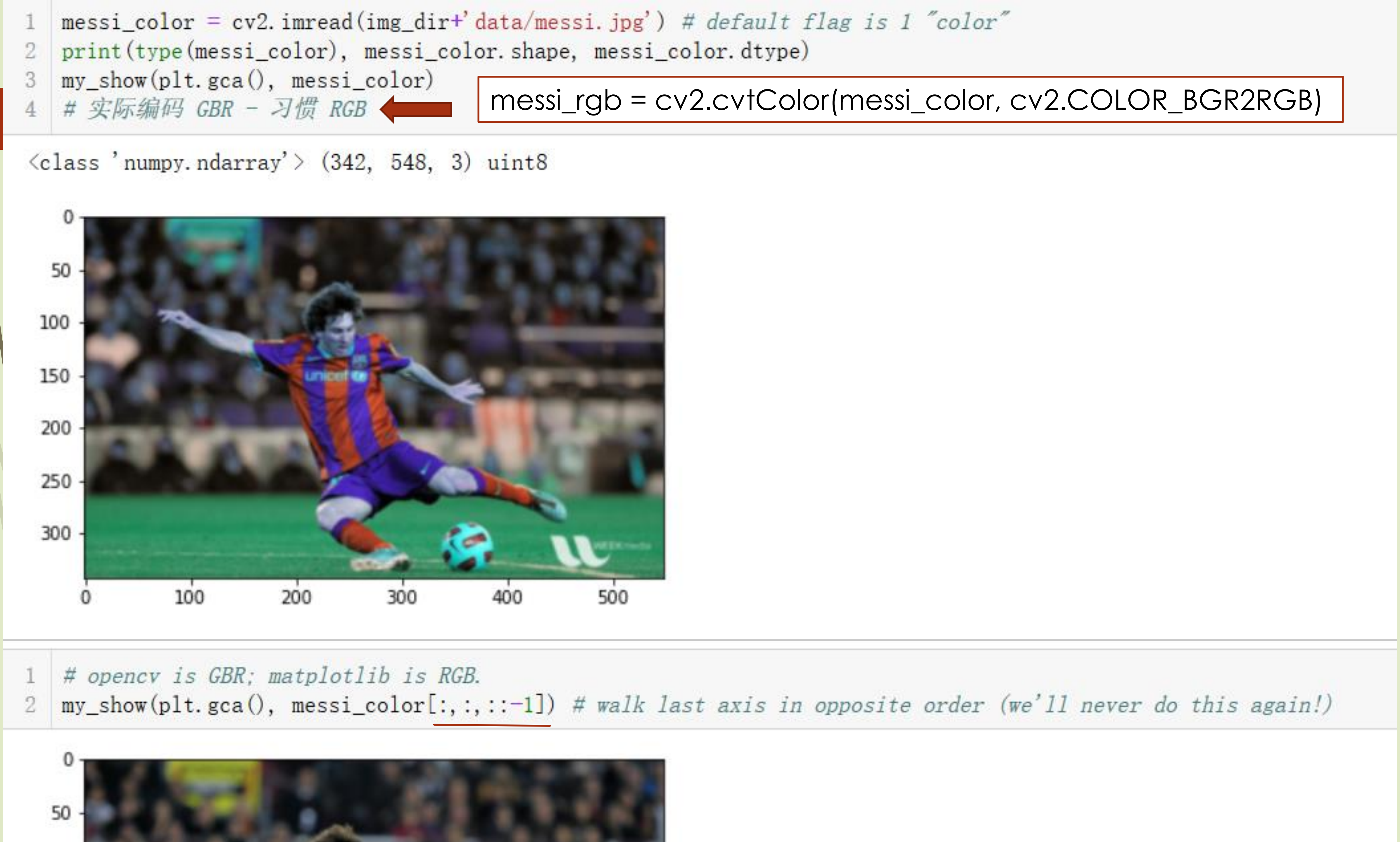
```
Figure(648x216) [<matplotlib.axes._subplots.AxesSubplot object at 0x000001E74CE145F8>
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E74CE522B0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x000001E74CE79940>]
```

- 加载显示图像



```
img_dir = 'common/'
messi_gray = cv2.imread(img_dir+'data/messi.jpg', 0)  # 第二个参数 0 grey
my_gshow(plt.gca(), messi_gray) # 参数1:  axis 参数2: array
messi_gray
```

```
ray([[ 43,  46,  48, ...,  55,  53,  50],
     [ 41,  46,  50, ...,  60,  58,  55],
     [ 46,  51,  56, ...,  64,  63,  60],
     ...,
     [120, 110, 107, ..., 113, 114, 124],
     [116, 119, 108, ..., 111, 122, 117],
     [107, 118, 129, ..., 104, 105, 104]], dtype=uint8)
```

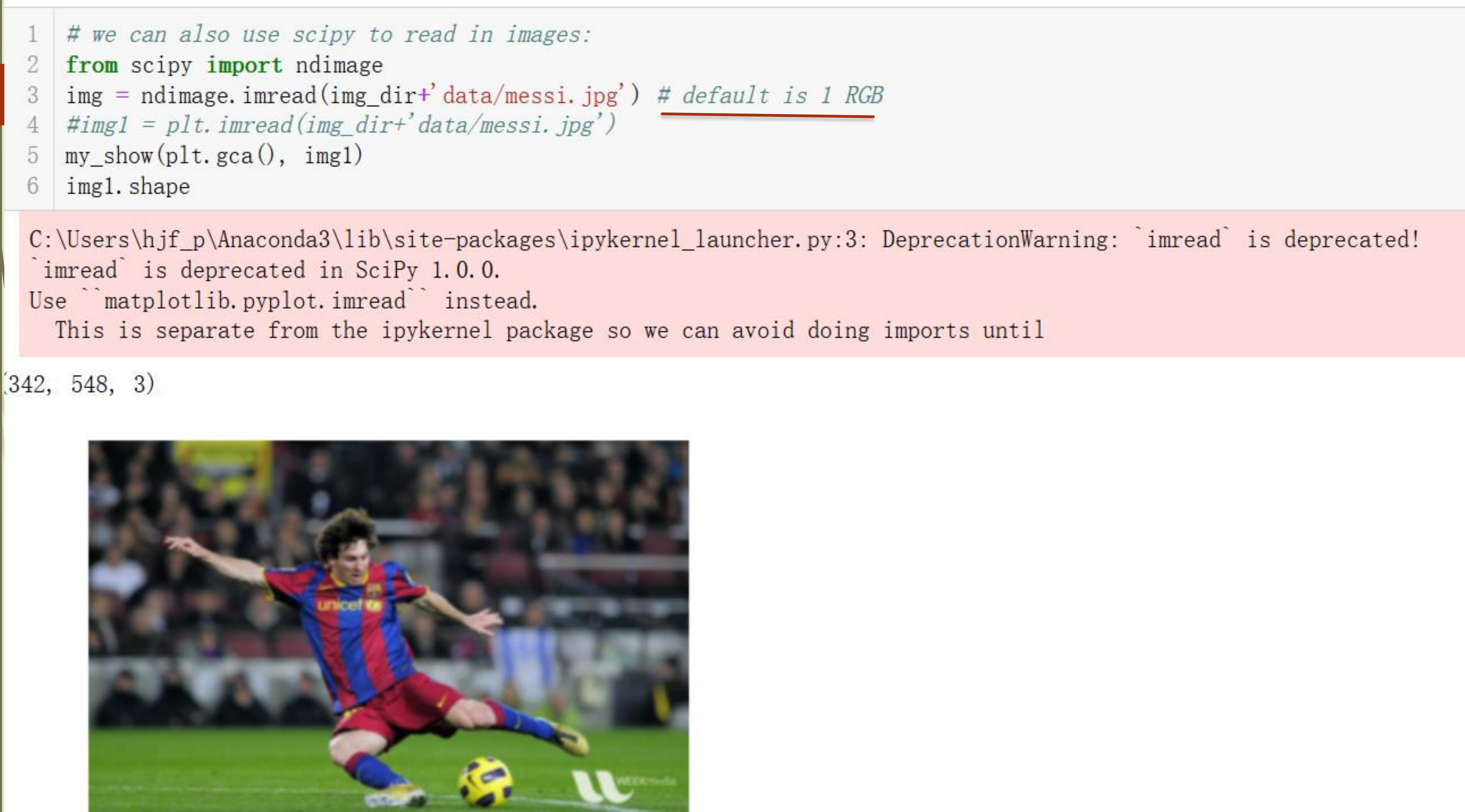


```
messi_color = cv2.imread(img_dir+'data/messi.jpg') # default flag is 1 "color"
print(type(messi_color), messi_color.shape, messi_color.dtype)
my_show(plt.gca(), messi_color)
# 实际编码 GBR - 习惯 RGB
```

```
messi_rgb = cv2.cvtColor(messi_color, cv2.COLOR_BGR2RGB)
```

```
<class 'numpy.ndarray'> (342, 548, 3) uint8
```

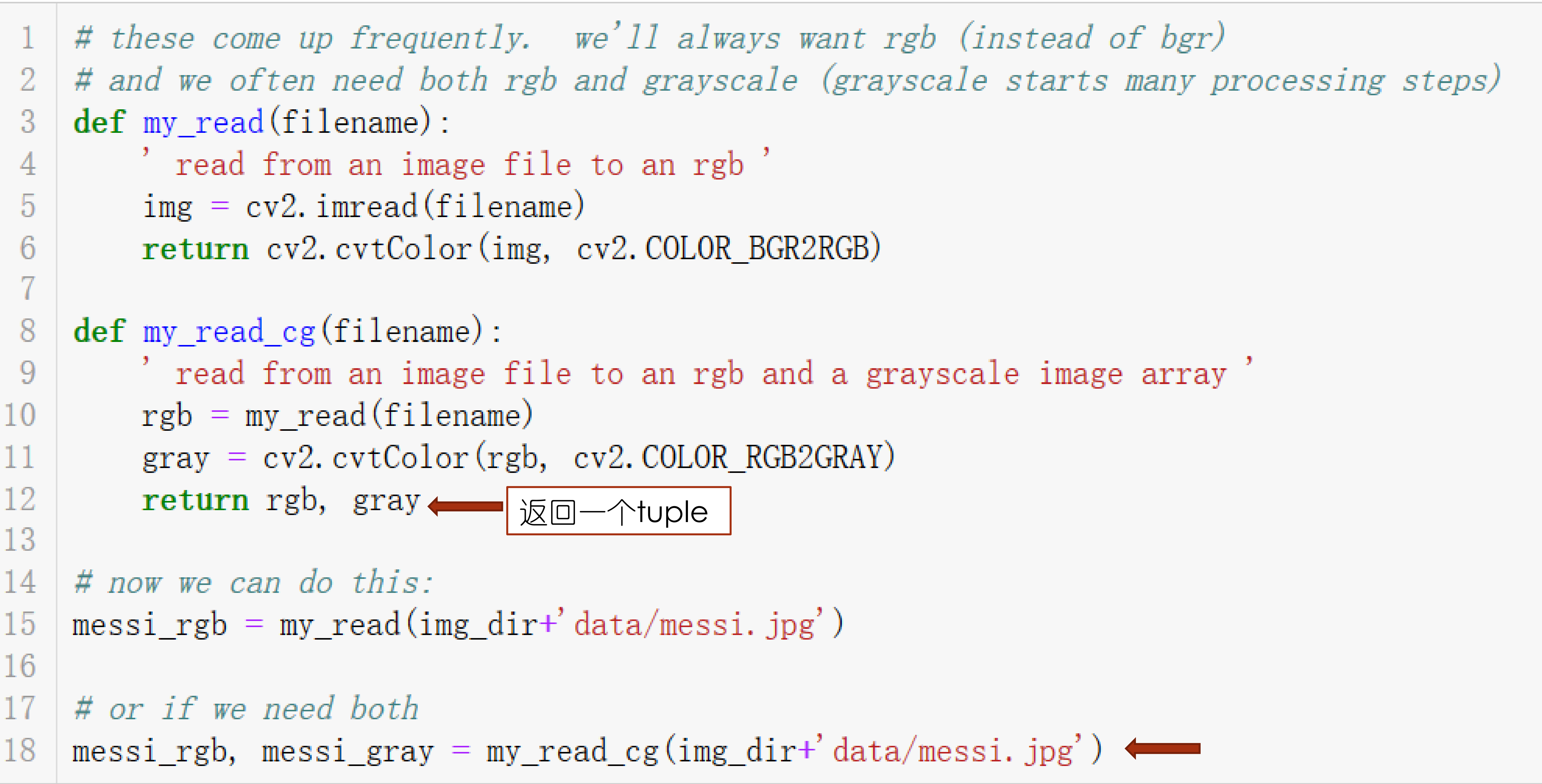
```
# opencv is GBR; matplotlib is RGB.
my_show(plt.gca(), messi_color[:,:,::-1]) # walk last axis in opposite order (we'll never do this again!)
```



```
# we can also use scipy to read in images:
from scipy import ndimage
img = ndimage.imread(img_dir+'data/messi.jpg') # default is 1 RGB
#img1 = plt.imread(img_dir+'data/messi.jpg')
my_show(plt.gca(), img1)
img1.shape
```

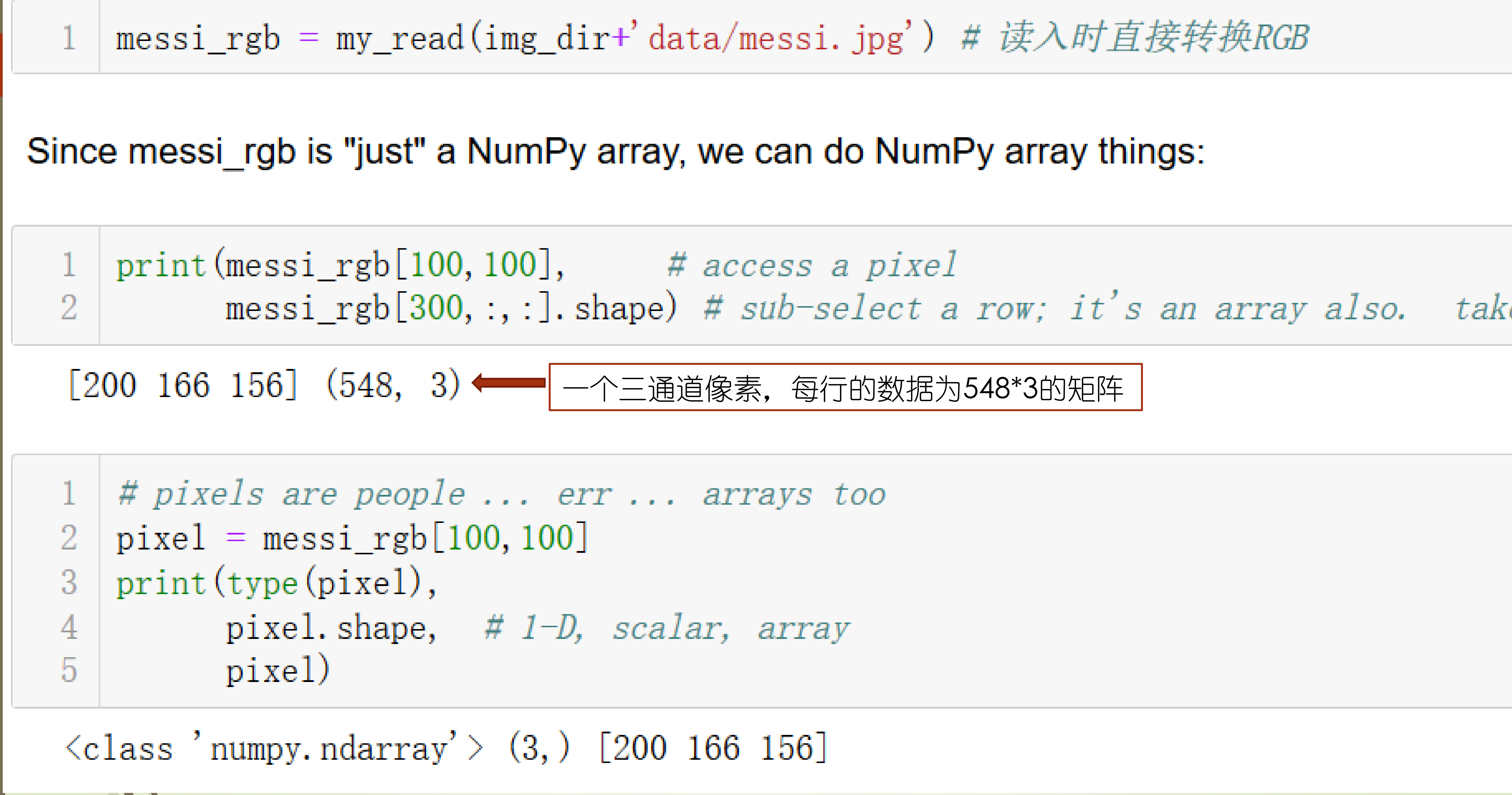
```
C:\Users\hjf_p\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0.
Use ``matplotlib.pyplot.imread`` instead.
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
(342, 548, 3)
```



```python
# these come up frequently.  we’ll always want rgb (instead of bgr)
# and we often need both rgb and grayscale (grayscale starts many processing steps)
def my_read(filename):
    ’ read from an image file to an rgb ’
    img = cv2.imread(filename)
    return cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

def my_read_cg(filename):
    ’ read from an image file to an rgb and a grayscale image array ’
    rgb = my_read(filename)
    gray = cv2.cvtColor(rgb, cv2.COLOR_RGB2GRAY)
    return rgb, gray

# now we can do this:
messi_rgb = my_read(img_dir+’data/messi.jpg’)

# or if we need both
messi_rgb, messi_gray = my_read_cg(img_dir+’data/messi.jpg’)
```

返回一个tuple



```
messi_rgb = my_read(img_dir+'data/messi.jpg') # 读入时直接转换RGB
```

Since messi_rgb is "just" a NumPy array, we can do NumPy array things:

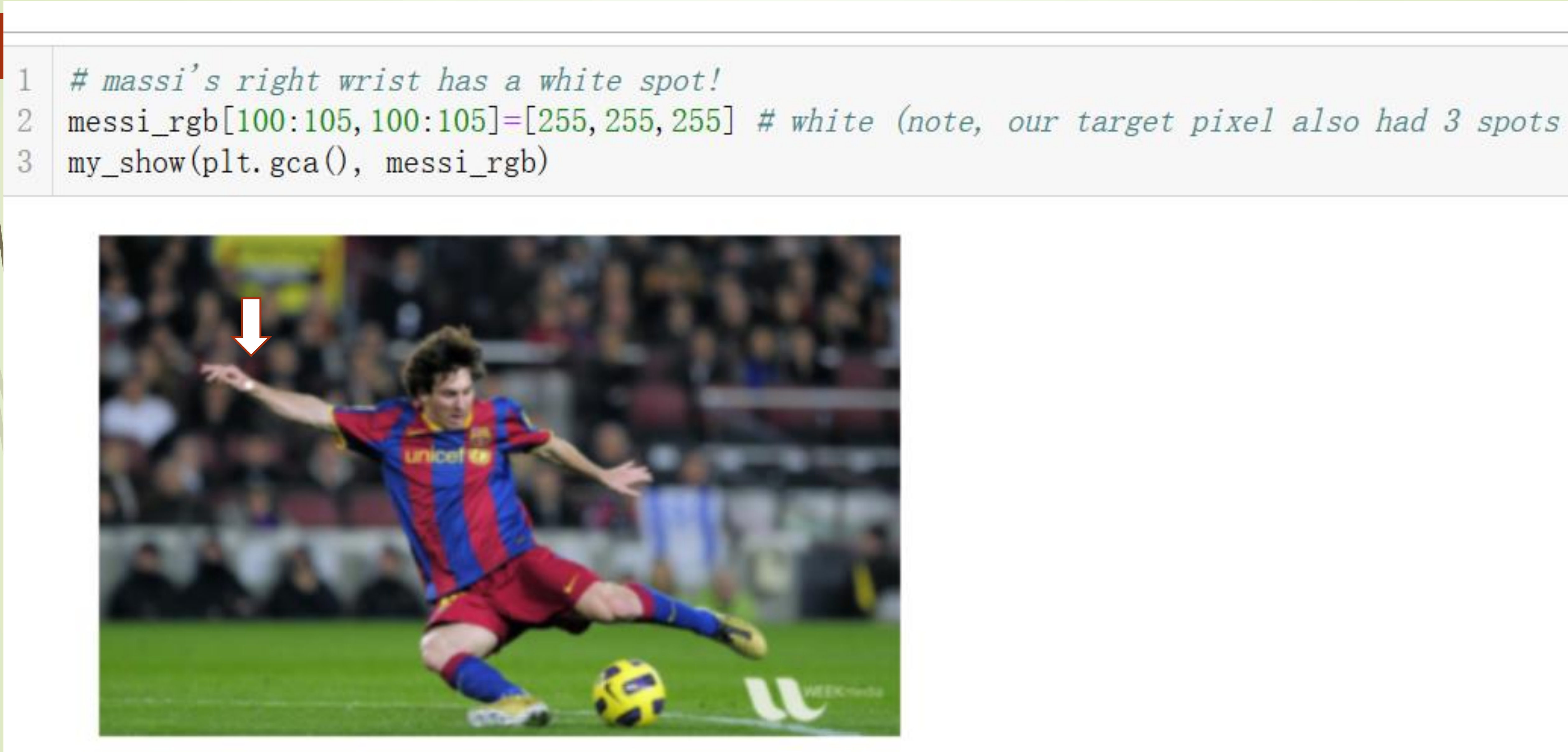
```
print(messi_rgb[100,100],       # access a pixel
      messi_rgb[300,:,:].shape) # sub-select a row; it's an array also.   tak
```

```
[200 166 156] (548, 3)
```

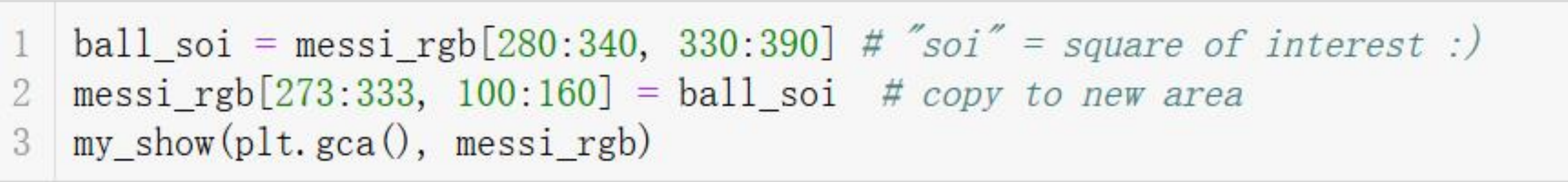
一个三通道像素，每行的数据为548*3的矩阵

```
# pixels are people ... err ... arrays too
pixel = messi_rgb[100,100]
print(type(pixel),
      pixel.shape,  # 1-D, scalar, array
      pixel)
```

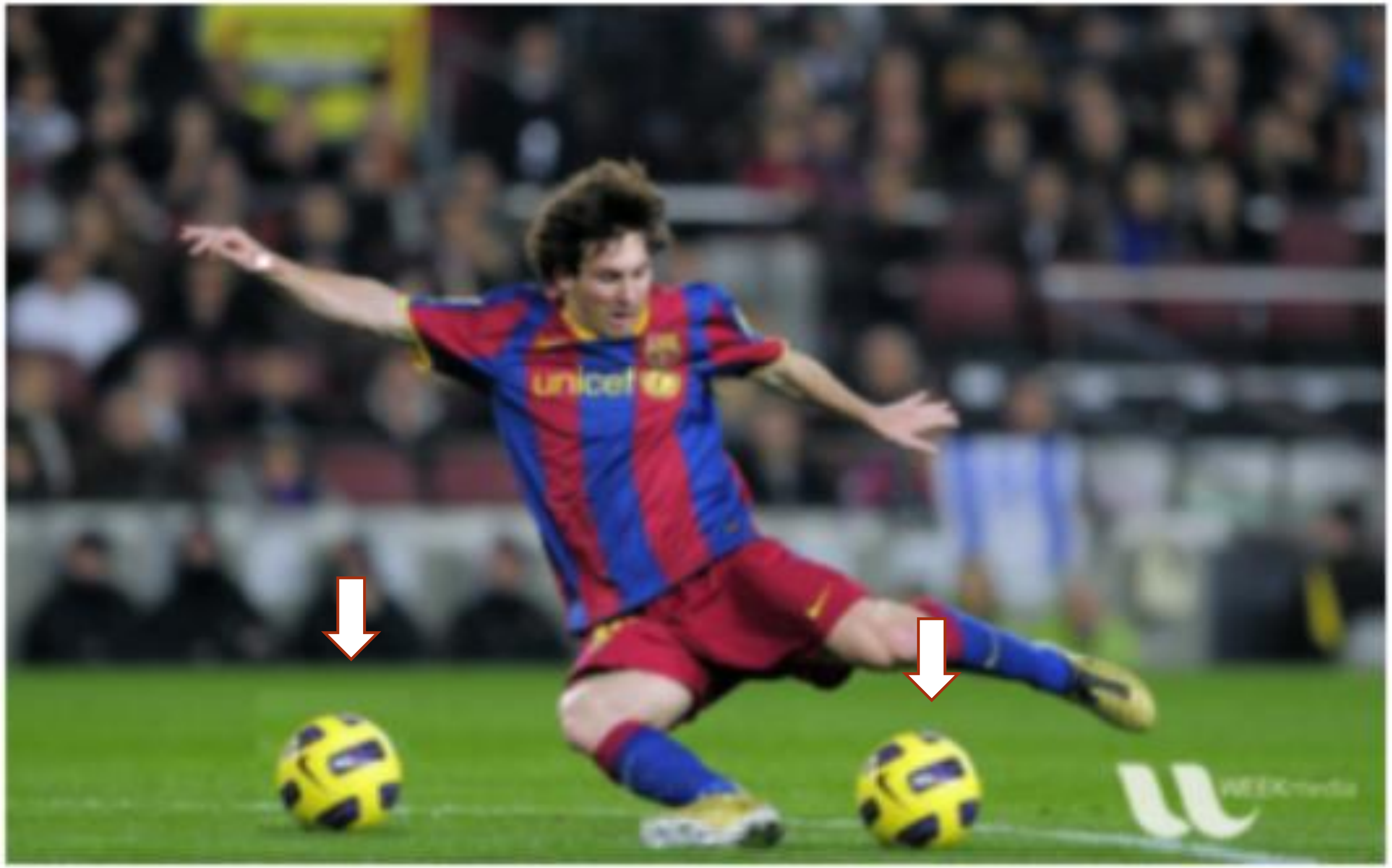
```
<class 'numpy.ndarray'> (3,) [200 166 156]
```

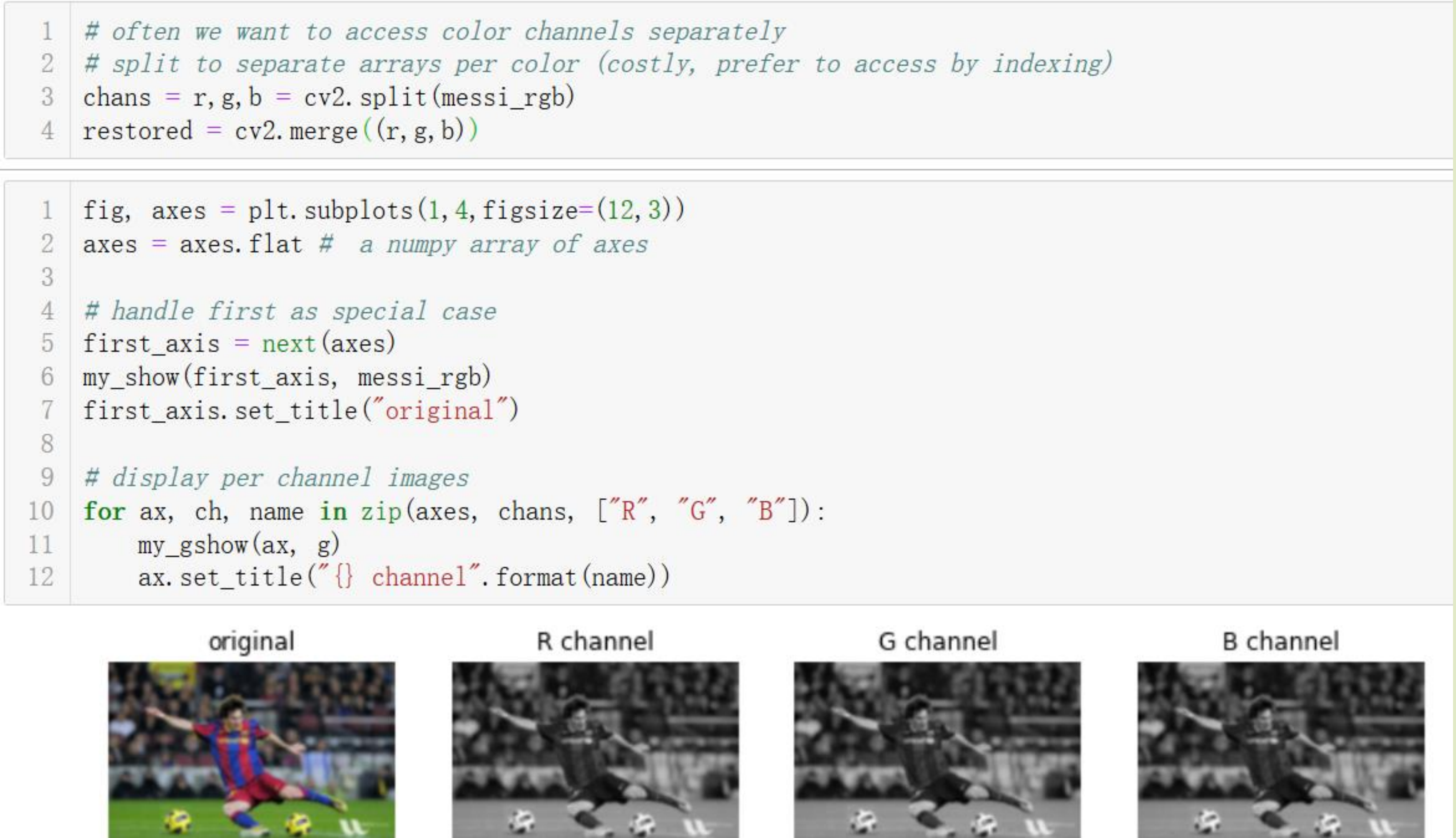


```
# massi's right wrist has a white spot!
messi_rgb[100:105,100:105]=[255,255,255] # white (note, our target pixel also had 3 spots
my_show(plt.gca(), messi_rgb)
```



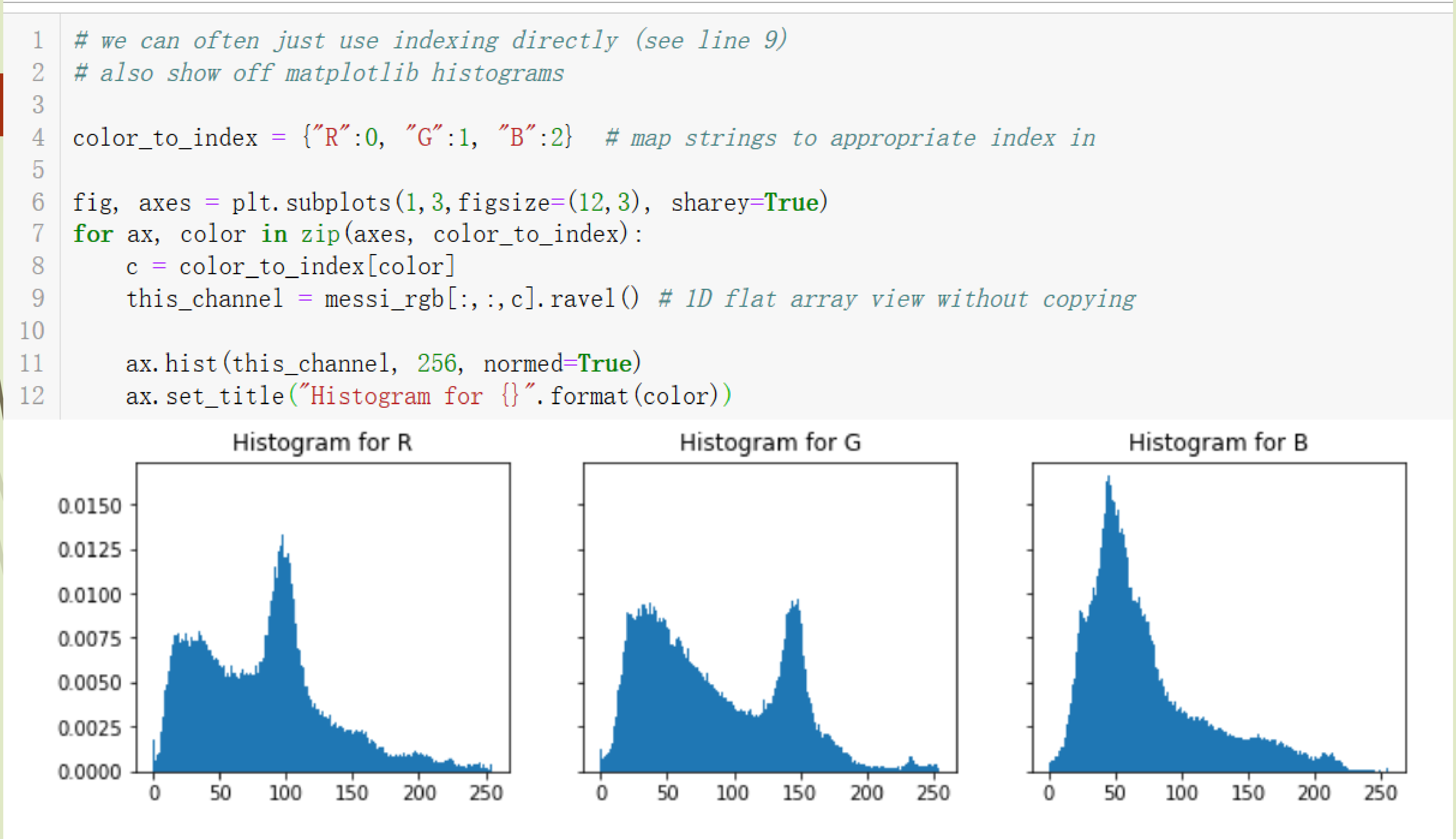
```python
ball_soi = messi_rgb[280:340, 330:390] # ″soi″ = square of interest :)
messi_rgb[273:333, 100:160] = ball_soi  # copy to new area
my_show(plt.gca(), messi_rgb)
```





```python
# often we want to access color channels separately
# split to separate arrays per color (costly, prefer to access by indexing)
chans = r,g,b = cv2.split(messi_rgb)
restored = cv2.merge((r,g,b))
```

```python
fig, axes = plt.subplots(1,4,figsize=(12,3))
axes = axes.flat #  a numpy array of axes

# handle first as special case
first_axis = next(axes)
my_show(first_axis, messi_rgb)
first_axis.set_title("original")

# display per channel images
for ax, ch, name in zip(axes, chans, ["R", "G", "B"]):
    my_gshow(ax, g)
    ax.set_title("{} channel".format(name))
```

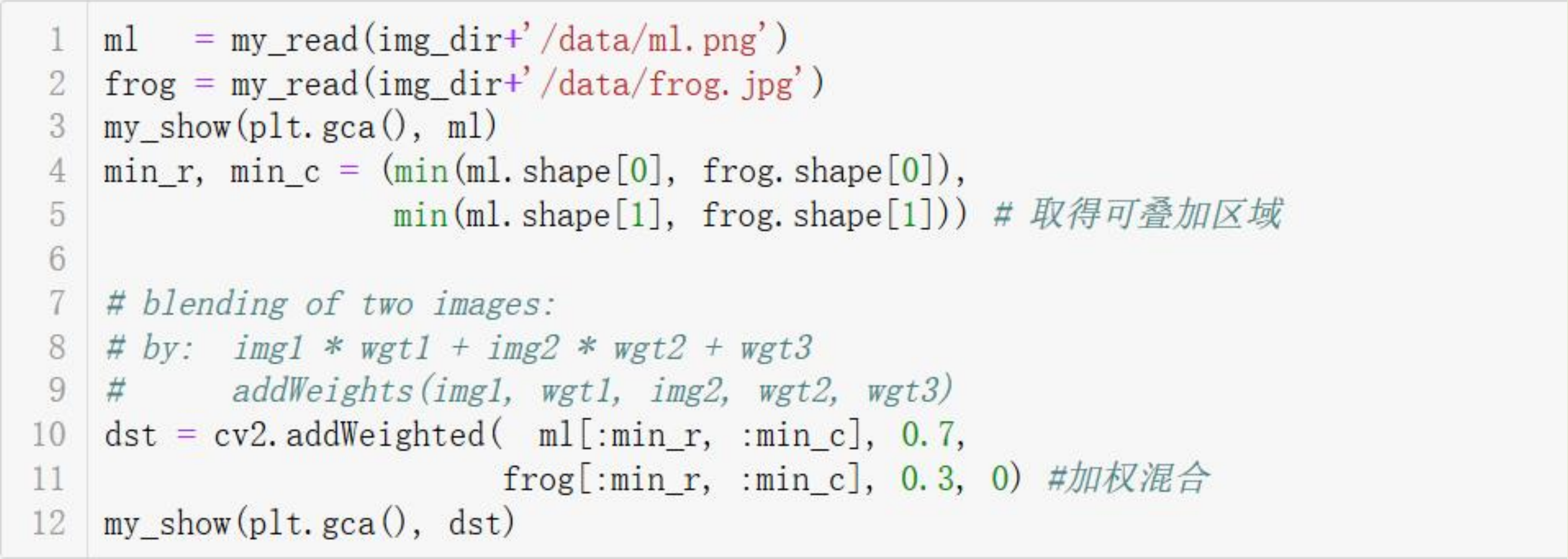


```
1  # we can often just use indexing directly (see line 9)
2  # also show off matplotlib histograms
3
4  color_to_index = {"R":0, "G":1, "B":2}  # map strings to appropriate index in
5
6  fig, axes = plt.subplots(1,3,figsize=(12,3), sharey=True)
7  for ax, color in zip(axes, color_to_index):
8      c = color_to_index[color]
9      this_channel = messi_rgb[:, :, c].ravel() # 1D flat array view without copying
10
11     ax.hist(this_channel, 256, normed=True)
12     ax.set_title("Histogram for {}".format(color))
```

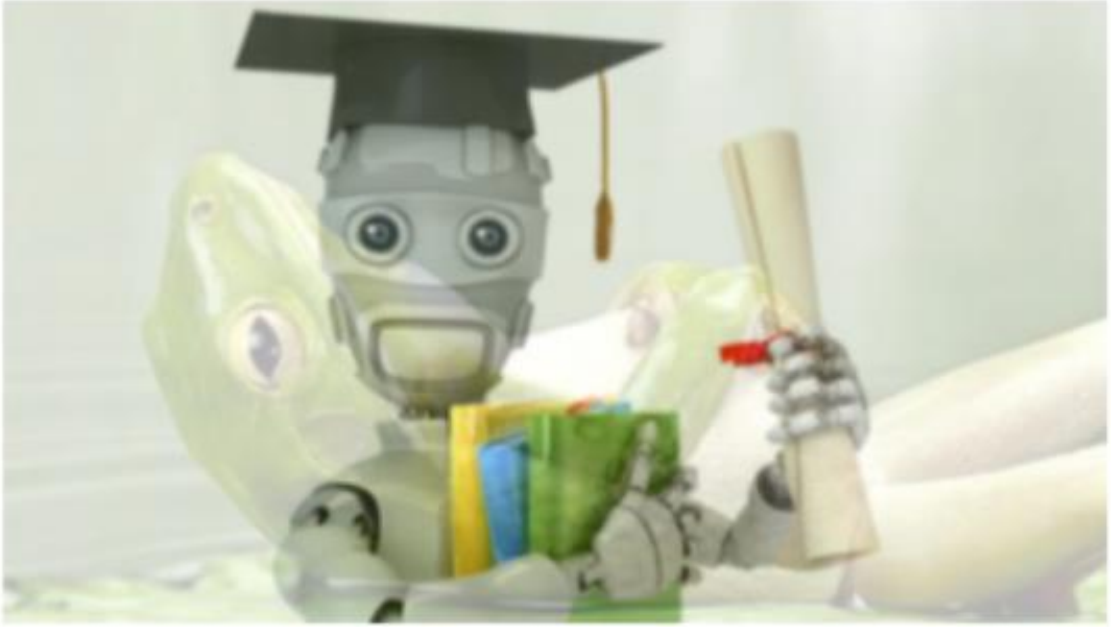
Histogram for R

Histogram for G

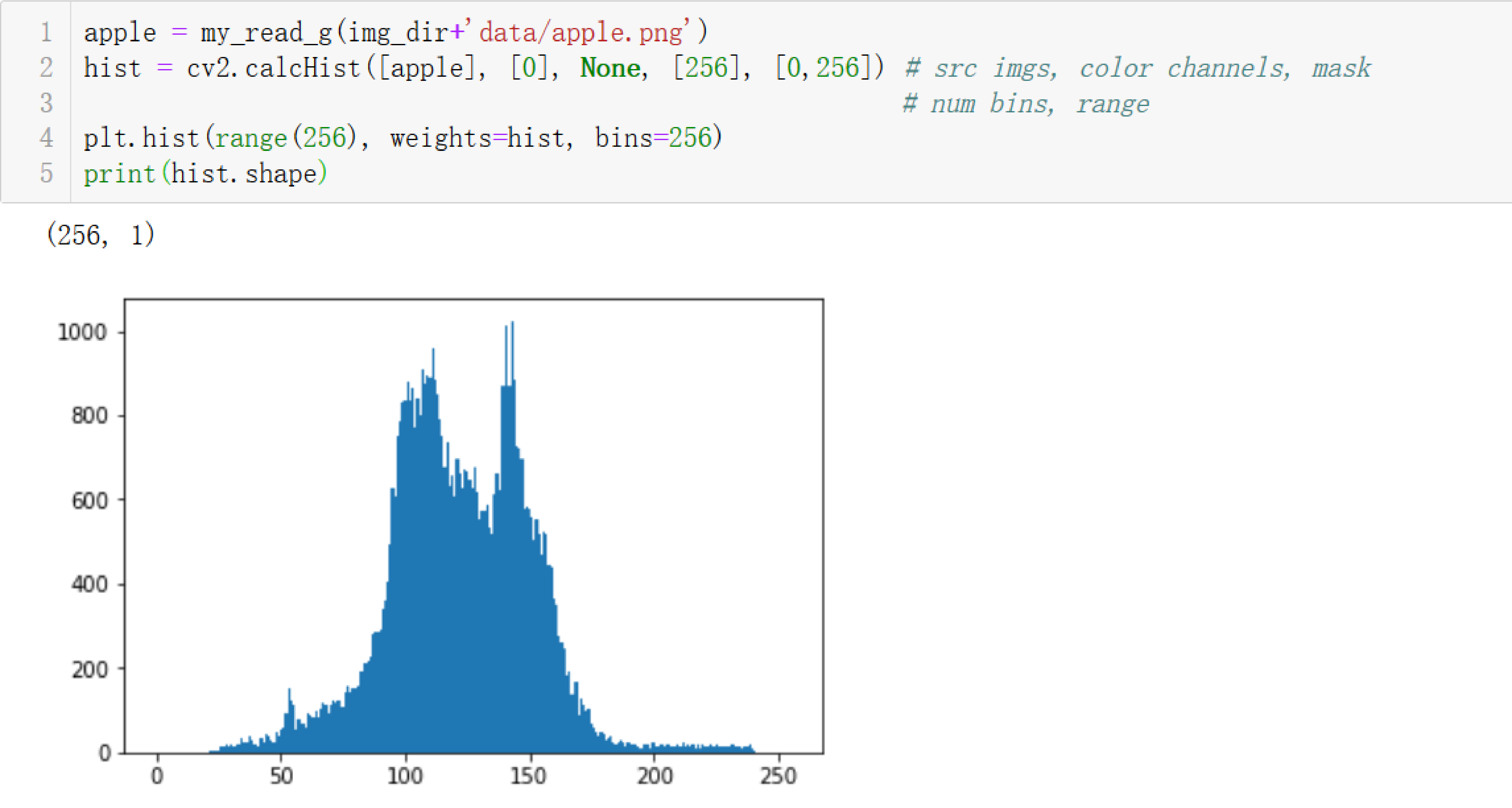
Histogram for B



```python
ml   = my_read(img_dir+'/data/ml.png')
frog = my_read(img_dir+'/data/frog.jpg')
my_show(plt.gca(), ml)
min_r, min_c = (min(ml.shape[0], frog.shape[0]),
                min(ml.shape[1], frog.shape[1])) # 取得可叠加区域

# blending of two images:
# by:  img1 * wgt1 + img2 * wgt2 + wgt3
#      addWeights(img1, wgt1, img2, wgt2, wgt3)
dst = cv2.addWeighted(  ml[:min_r, :min_c], 0.7,
                      frog[:min_r, :min_c], 0.3, 0) #加权混合
my_show(plt.gca(), dst)
```



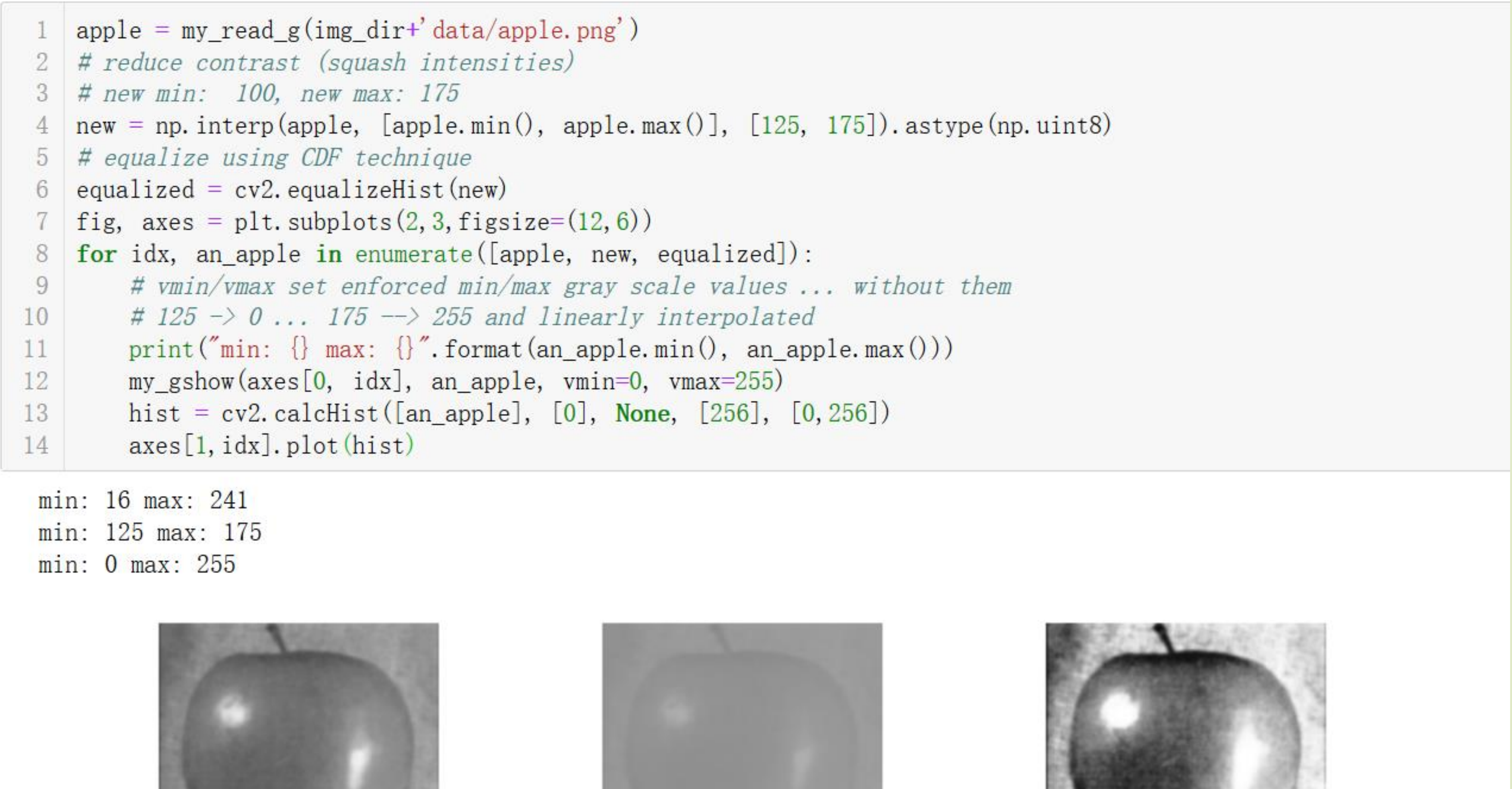
Histograms



```
apple = my_read_g(img_dir+’data/apple.png’)
hist = cv2.calcHist([apple], [0], None, [256], [0,256]) # src imgs, color channels, mask
                                                        # num bins, range
plt.hist(range(256), weights=hist, bins=256)
print(hist.shape)
```

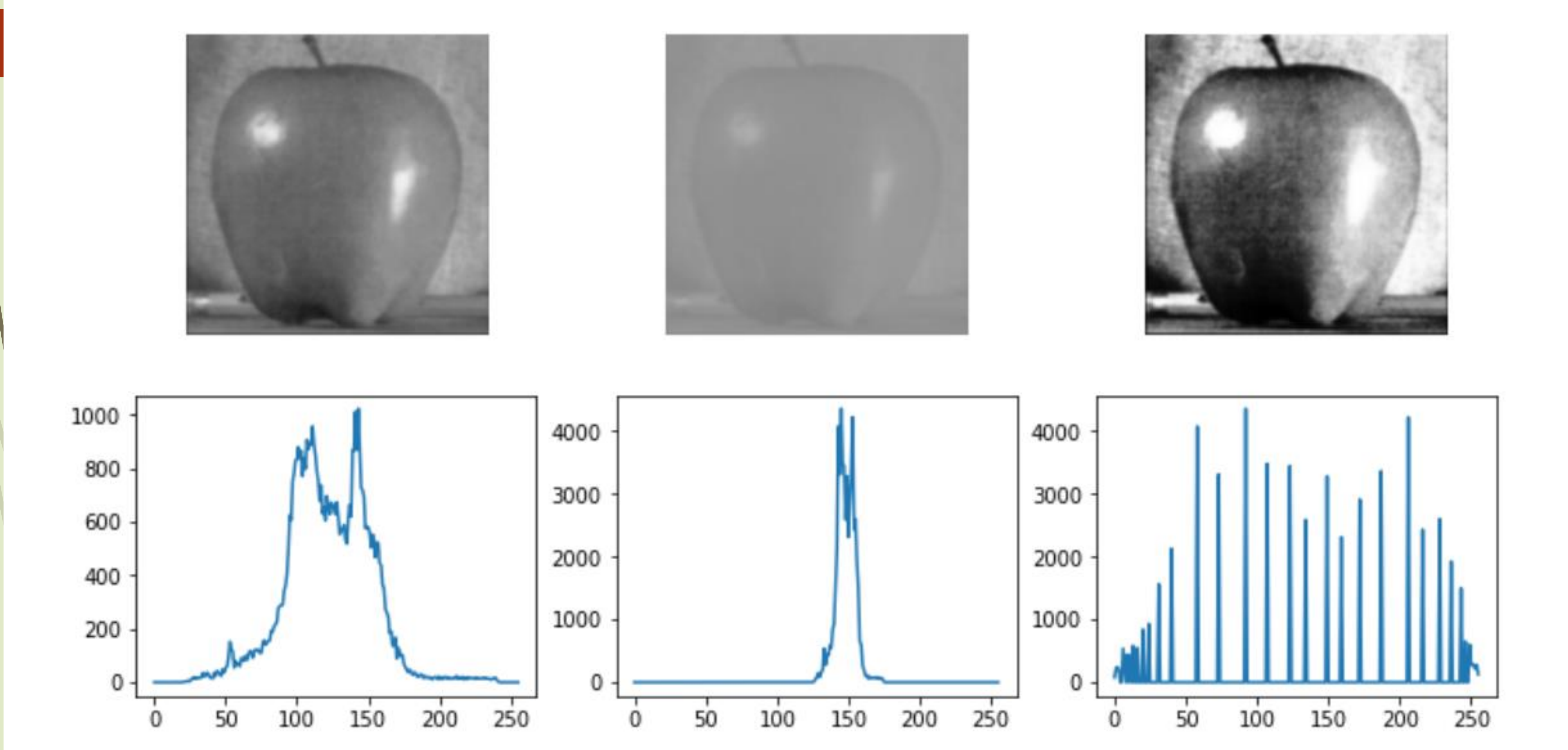
```
(256, 1)
```

Histogram Equalization

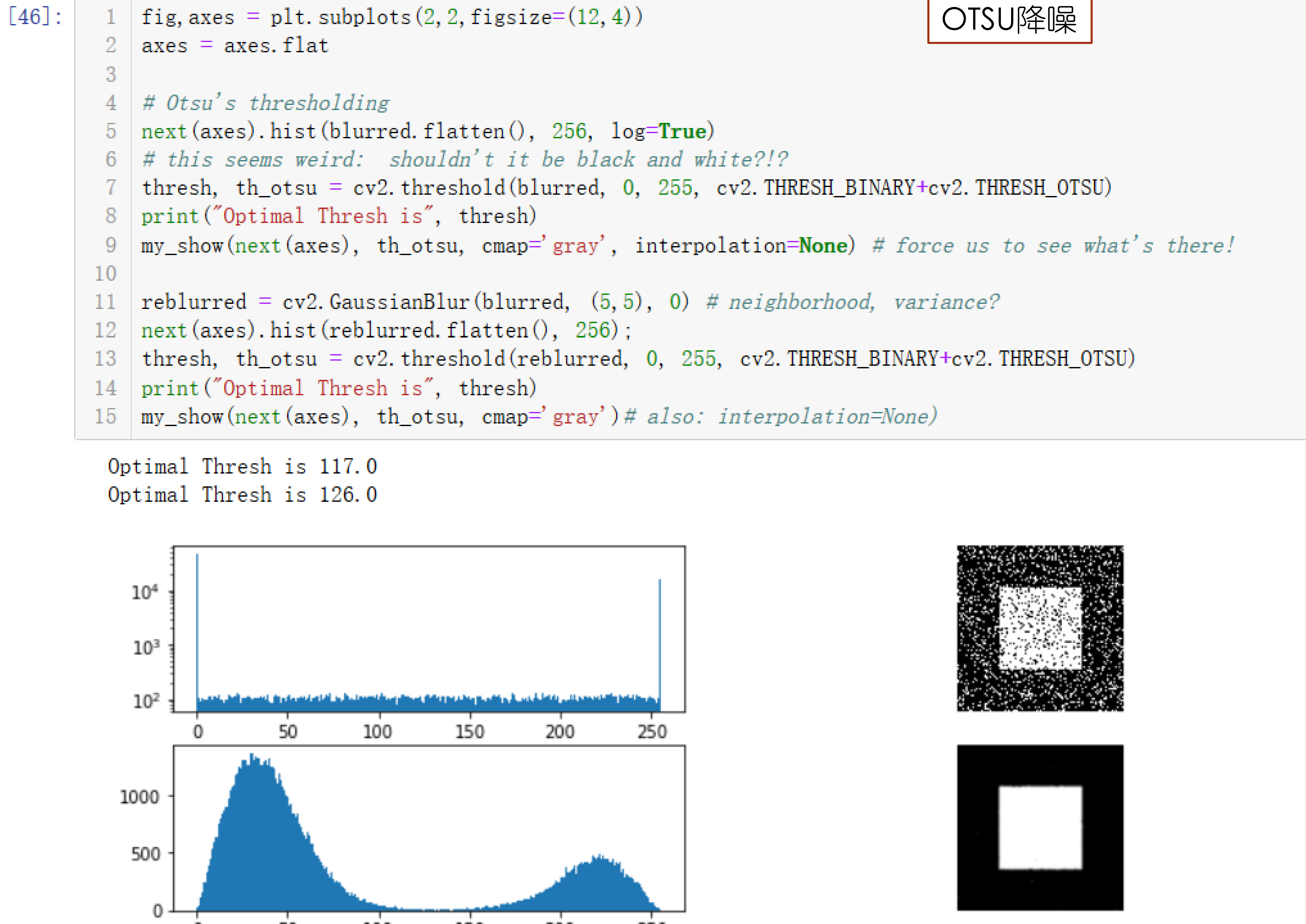


```python
apple = my_read_g(img_dir+'data/apple.png')
# reduce contrast (squash intensities)
# new min:  100, new max: 175
new = np.interp(apple, [apple.min(), apple.max()], [125, 175]).astype(np.uint8)
# equalize using CDF technique
equalized = cv2.equalizeHist(new)
fig, axes = plt.subplots(2,3,figsize=(12,6))
for idx, an_apple in enumerate([apple, new, equalized]):
    # vmin/vmax set enforced min/max gray scale values ... without them
    # 125 -> 0 ... 175 --> 255 and linearly interpolated
    print("min: {} max: {}".format(an_apple.min(), an_apple.max()))
    my_gshow(axes[0, idx], an_apple, vmin=0, vmax=255)
    hist = cv2.calcHist([an_apple], [0], None, [256], [0,256])
    axes[1,idx].plot(hist)
```

```
min: 16 max: 241
min: 125 max: 175
min: 0 max: 255
```


1000
800
600
400
200
0
0
50
100
150
200
250
4000
3000
2000
1000
0
0
50
100
150
200
250
4000
3000
2000
1000
0
0
50
100
150
200
250

OTSU降噪

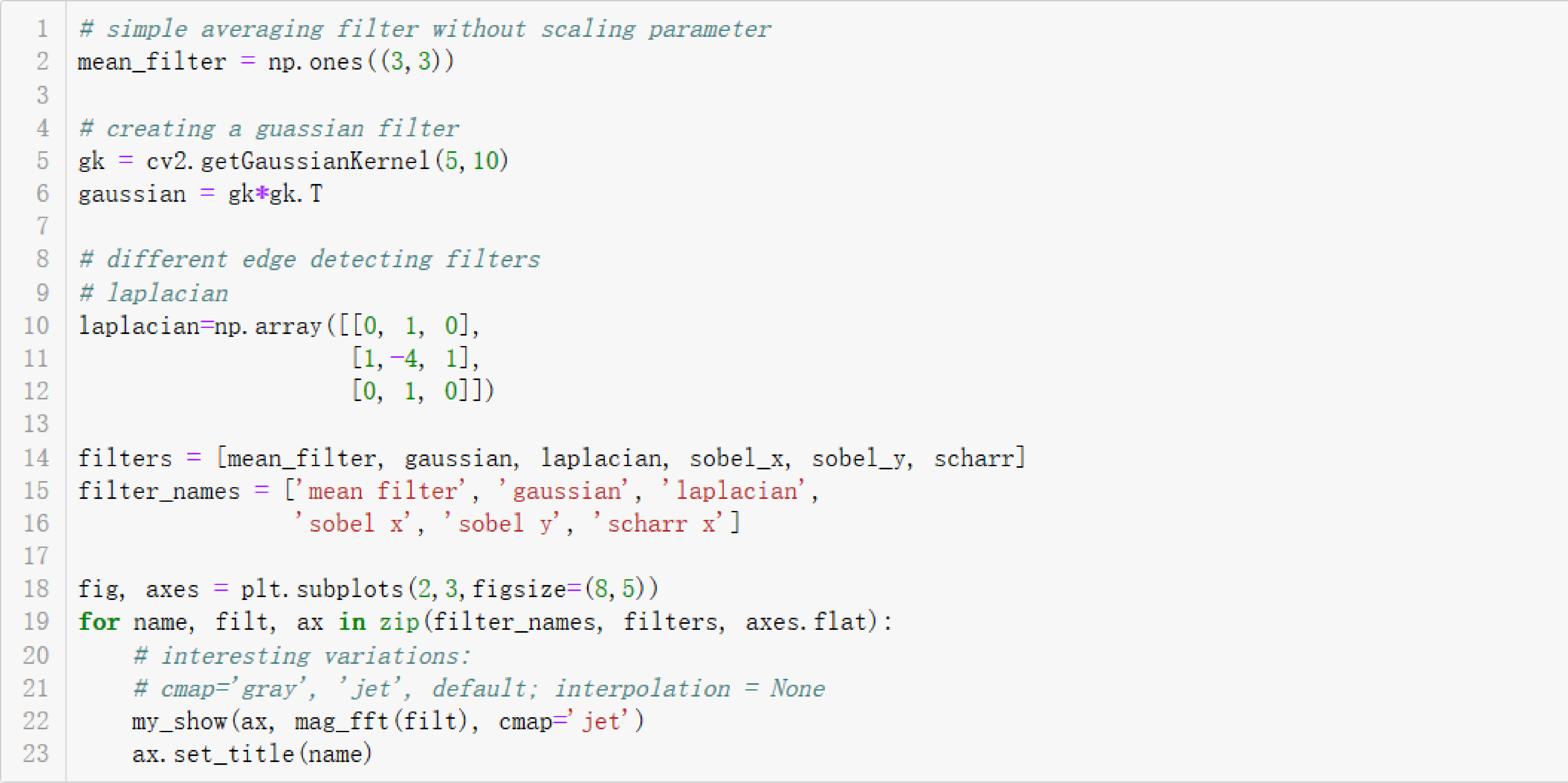


[46]:

```
fig, axes = plt.subplots(2,2,figsize=(12,4))
axes = axes.flat

# Otsu's thresholding
next(axes).hist(blurred.flatten(), 256, log=True)
# this seems weird:  shouldn't it be black and white?!?
thresh, th_otsu = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
print("Optimal Thresh is", thresh)
my_show(next(axes), th_otsu, cmap='gray', interpolation=None) # force us to see what's there!

reblurred = cv2.GaussianBlur(blurred, (5,5), 0) # neighborhood, variance?
next(axes).hist(reblurred.flatten(), 256);
thresh, th_otsu = cv2.threshold(reblurred, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
print("Optimal Thresh is", thresh)
my_show(next(axes), th_otsu, cmap='gray')# also: interpolation=None)
```

```
Optimal Thresh is 117.0
Optimal Thresh is 126.0
```

Filters and Convolutions



```python
# simple averaging filter without scaling parameter
mean_filter = np.ones((3,3))

# creating a guassian filter
gk = cv2.getGaussianKernel(5,10)
gaussian = gk*gk.T

# different edge detecting filters
# laplacian
laplacian=np.array([[0,  1,  0],
                    [1, -4,  1],
                    [0,  1,  0]])

filters = [mean_filter, gaussian, laplacian, sobel_x, sobel_y, scharr]
filter_names = ['mean filter', 'gaussian', 'laplacian',
                'sobel x', 'sobel y', 'scharr x']

fig, axes = plt.subplots(2,3,figsize=(8,5))
for name, filt, ax in zip(filter_names, filters, axes.flat):
    # interesting variations:
    # cmap='gray', 'jet', default; interpolation = None
    my_show(ax, mag_fft(filt), cmap='jet')
    ax.set_title(name)
```

作业课后布置