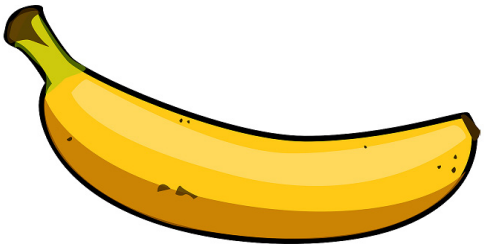
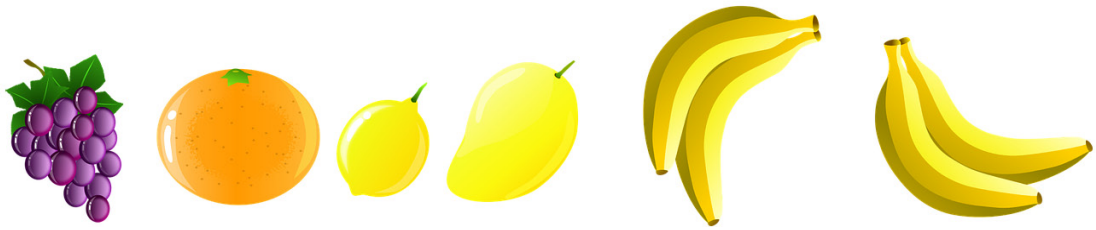
实验：轮廓匹配应用

实验目标

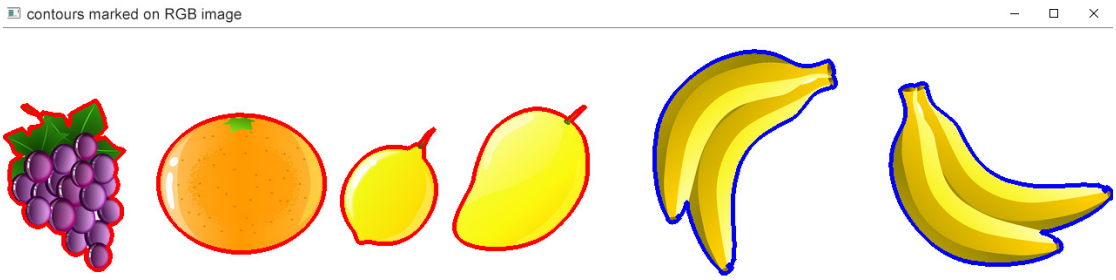
本实验为前面实验的延伸，您将获得以下包含香蕉的参考图像，并以此为模板，把原图像中的香蕉检测出来。



应用轮廓匹配技术，在原图像中把香蕉检测出来 ——

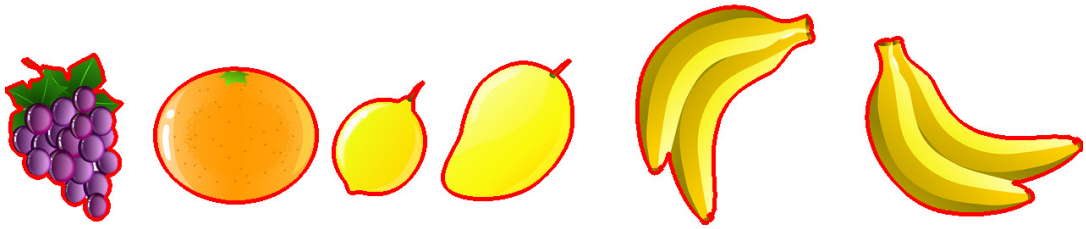


您的任务是识别上面图像中存在的所有香蕉，并将它们标记为蓝色，用不同颜色的轮廓线与其他水果区分开来，如下所示：



1. 恢复实验进度

在前面的实验中，我们已经完成了对水果图像全部轮廓的识别，并且在图像上使用红色轮廓线对全部水果进行了轮廓绘制。如下图 ——

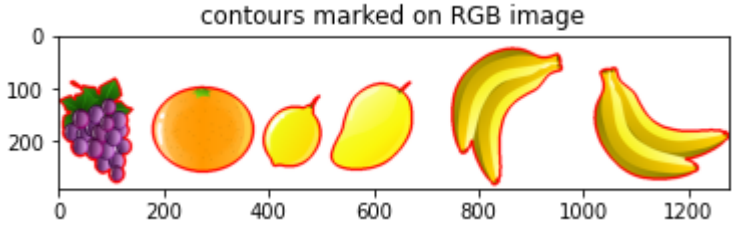


本实验我们将从之前的实验结果开始，使用不同颜色的轮廓线，将香蕉与其他水果区分开来。

首先，创建一个原图像的副本 `imagecopy` ，并使用以下代码恢复实验进度：

In [1]:

```
import cv2                          # 导入OpenCV
import matplotlib.pyplot as plt   # 导入matplotlib

# 魔法指令，使图像直接在Notebook中显示
%matplotlib inline

# 设置输入输出路径
import os
base_path = os.environ.get("BASE_PATH",'../data/')
data_path = os.path.join(base_path + "lab4/")
result_path = "result/"
os.makedirs(result_path, exist_ok=True)

# 读取本地图像
image = cv2.imread('./data/many fruits.png')
# 拷贝原图像副本
imagecopy= image.copy()

# cv2.imshow( 'Original image' , image )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 将图像转换为灰度图像
gray_image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

# cv2.imshow( 'gray' , gray_image )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 直接输入阈值与最大值，执行图像二值化
ret,binary_im = cv2.threshold(gray_image,245,255,cv2.THRESH_BINARY)

# cv2.imshow( 'binary' , binary_im )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

binary_im= ~binary_im # 反转图像（黑白像素互换）

# cv2.imshow( 'inverted binary' , binary_im )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 从二值图像中找到外部轮廓
contours,hierarchy = cv2.findContours(binary_im,
                        cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

# 绘制轮廓
with_contours = cv2.drawContours(image,contours,-1,(0,0,255),3)

# cv2.imshow( 'contours marked on RGB image' , with_contours )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 将图像从 BGR 转换为 RGB
plt.imshow(with_contours[:,:,::-1])
# 指定输出图像的标题
plt.title('contours marked on RGB image')
# 显示图像
plt.show()
```



注意： 这里的 `contours` 是我们需要进行轮廓匹配的第一个轮廓列表。

而第二个轮廓： 香蕉 ，我们在后面的实验中，将用 `reference_contour` 标识。

统计当前轮廓数量为 `6` ，

In [2]:

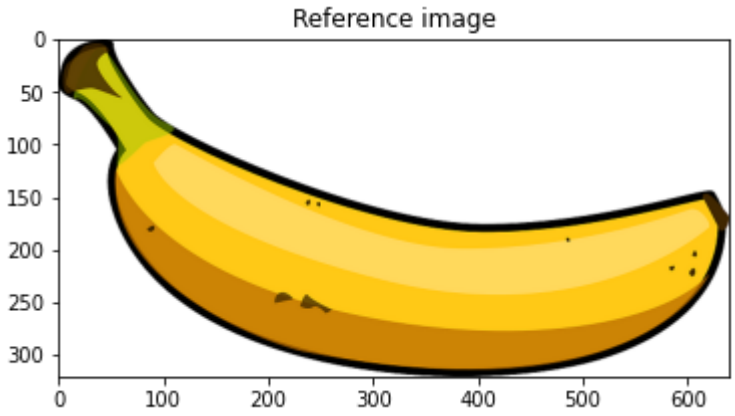
```
# 设置输出提示
print('Total number of contours:')
# 设置输出提示
print(len(contours))
```

```
Total number of contours:
6
```

2. 加载参考图像

In [3]:

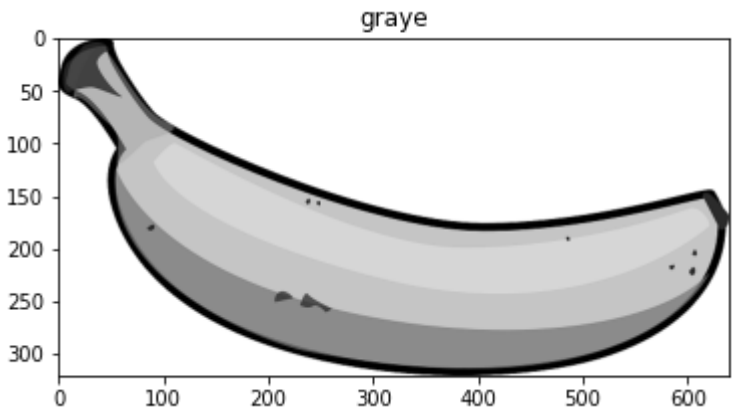
```
# 读取本地参考图像
ref_image = cv2.imread('./data/bananaref.png')

# 在云实验环境下忽略以下代码，避免程序尝试打开系统窗口显示图片；
# 使用matplotlib替换，使图像直接在 Jupyter Notebook 中输出。

# cv2.imshow( 'Reference image' , ref_image )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

plt.imshow(ref_image[:,:,::-1])          # 将图像从 BGR 转换为 RGB
plt.title('Reference image')             # 指定输出图像的标题
plt.show()                               # 显示图像
```



3. 转换灰度图像

In [4]:

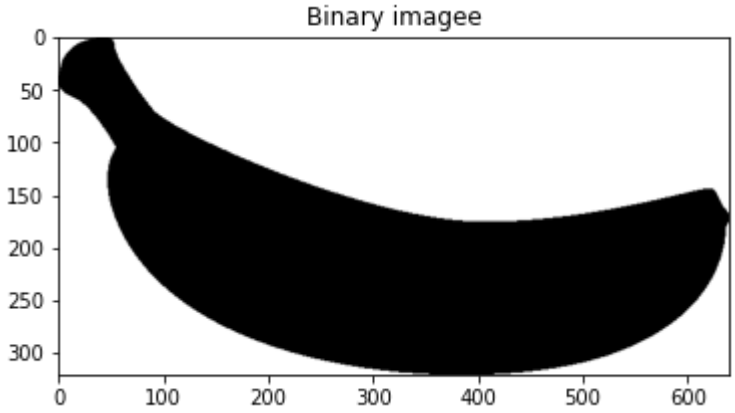
```
# 将图像转换为灰度图像
gray_image = cv2.cvtColor(ref_image,cv2.COLOR_BGR2GRAY)

# 在云实验环境下忽略以下代码，避免程序尝试打开系统窗口显示图片；
# 使用matplotlib替换，使图像直接在 Jupyter Notebook 中输出。

# cv2.imshow( 'Grayscale image' , gray_image )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 使用灰色“喷涂”图像输出显示
plt.imshow(gray_image, cmap='gray')
# 指定输出图像的标题
plt.title('graye')
# 显示图像
plt.show()
```



4. 图像二值化

In [5]:

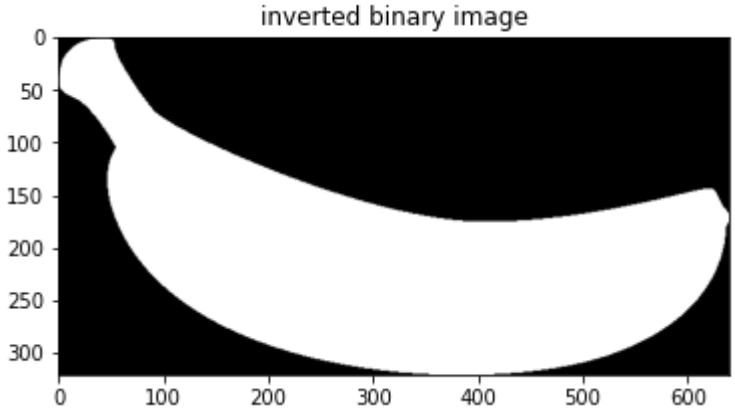
```
ret,binary_im = cv2.threshold(gray_image,245,255,cv2.THRESH_BINARY)

# 在云实验环境下忽略以下代码，避免程序尝试打开系统窗口显示图片；
# 使用matplotlib替换，使图像直接在 Jupyter Notebook 中输出。

# cv2.imshow( 'Binary image' , binary_im )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 使用灰色“喷涂”图像输出显示
plt.imshow(binary_im, cmap='gray')
# 指定输出图像的标题
plt.title('Binary imagee')
# 显示图像
plt.show()
```



5. 反转图像

In [6]:

```
binary_im= ~binary_im  # 反转图像（黑白像素互换）

# 在云实验环境下忽略以下代码，避免程序尝试打开系统窗口显示图片；
# 使用matplotlib替换，使图像直接在 Jupyter Notebook 中输出。

# cv2.imshow( 'inverted binary image' , binary_im )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 使用灰色“喷涂”图像输出显示
plt.imshow(binary_im, cmap='gray')
# 指定输出图像的标题
plt.title('inverted binary image')
# 显示图像
plt.show()
```

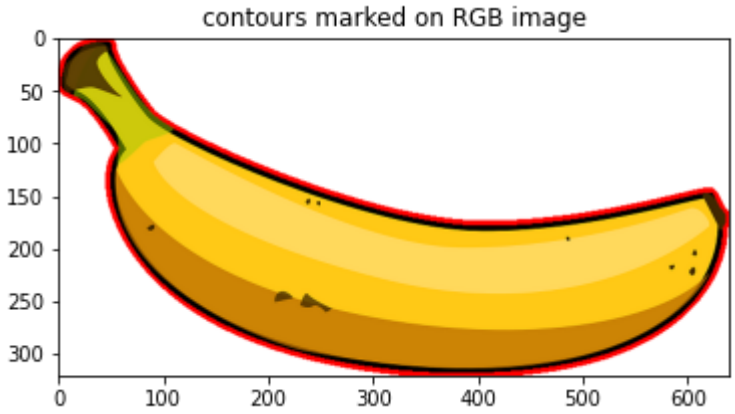


6. 轮廓检测

找到此形状的外部边界，并在其上绘制红色轮廓：

In [7]:

```
# 从二值图像中找到外部轮廓
ref_contour_list,hierarchy = cv2.findContours(binary_im,
                                cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

# 绘制轮廓
with_contours = cv2.drawContours(ref_image,ref_contour_list,-1,(0,0,255),3)

# 在云实验环境下忽略以下代码，避免程序尝试打开系统窗口显示图片；
# 使用matplotlib替换，使图像直接在 Jupyter Notebook 中输出。

# cv2.imshow( 'contours marked on RGB image' , with_contours )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

plt.imshow(with_contours[:,:,::-1])          # 将图像从 BGR 转换为 RGB
plt.title('contours marked on RGB image')    # 指定输出图像的标题
plt.show()                                   # 显示图像
```



此图像中应该只有一个轮廓：香蕉的轮廓。

要进行确认，可以使用 `print(len(ref_contour_list))` 命令，检查 Python 列表 `ref_contour_list` 中的轮廓数量。您会发现仅存在一个轮廓：

In [8]:

```
print('Total number of contours:')          # 设置输出提示
print(len(ref_contour_list))                 # 设置输出提示
```

```
Total number of contours:
1
```

该轮廓就是我们需要检测的 `香蕉` ，将该轮廓放在列表的索引 `0` 处，实际上，它也是此列表中唯一的轮廓。

注意： 这里的 `reference_contour` 是我们需要进行轮廓匹配的第二个轮廓。

In [9]:

```
# 将ref_contour_list轮廓索引中的第一个轮廓（也是唯一的轮廓）
# 命名为reference_contour
# 在后面的轮廓匹配中，
# 用reference_contour与contours轮廓列表索引中的每一个轮廓进行匹配
reference_contour = ref_contour_list[0]
```

7. 轮廓匹配

现在，我们必须将在水果的原始图像中检测到的每个水果轮廓与香蕉的参考轮廓进行比较。所以，我们执行的是一个单一轮廓 香蕉 （ reference_contour ）与一个轮廓列表 contours 内多个轮廓的一一匹配。因此 —— for 循环是最好的方法 ——

我们将创建一个遍历 contours 轮廓列表索引号码的 for 循环，按照 contours 轮廓列表索引号码提取轮廓，与参考轮廓 reference_contour 进行匹配。

另外，在开始 for 循环之前，我们将使用 dist_list 的名称初始化一个空列表。

在 for 循环的每次迭代中，我们将在列表中附加上参考轮廓 reference_contour 与轮廓 contours 之间的数值差：

In [10]:

```
# 创建空列表，用于保存每次轮廓匹配后的数值差
dist_list= []
# 创建contours轮廓列表索引号码的for循环
for cnt in contours:
    # 按照contours轮廓列表索引号码提取轮廓，与reference_contour参考轮廓进行匹配
    # 其中，比较方法我们使用cv.CONTOURS_MATCH_I1
    # 而没用的parameter参数我们用0进行传递
    retval= cv2.matchShapes(cnt, reference_contour,1,0)


    # 将本次轮廓匹配数值差填入dist_list列表，之后执行下一次循环
    dist_list.append(retval)
```

完成 for 循环后，输出 dist_list ，显示 reference_contour 与 contours 中每一个轮廓匹配后，产生的轮廓之间的数值差 ——

可以看到我们的参考图像中的 香蕉 轮廓，与水果图片中最后两个轮廓（同样是香蕉）的数值差最少。

In [11]:

```
print(dist_list) # 输出轮廓匹配数值差列表
```

```
[1.9329380069362876, 1.6618014776874734, 1.771151523726344, 1.8150005423795166, 1.26
92130505335848, 1.269213050534173]
```

8. 查找匹配轮廓

下一个任务，是找到在 contours 轮廓列表中，与参考轮廓 reference_contour 最小距离的两个轮廓。

使用 sort() 命令，对轮廓匹配数值差从小到大排列。

完成排序后， sorted_list 的第一个元素 （ sorted_list [0] ）是最小的距离，而它的第二个元素 （ sorted_list [1]） 是第二小的距离。那么，这两个距离对应于图像中存在的两束香蕉。根据这个值，在原始 dist_list 列表中，找到存在最小距离和第二最小距离的索引编号。

初始化一个新的空列表 banana_cnts ，并将轮廓列表当中。

In [12]:

```
# 复制dist_list并将其存储在单独的变量中
sorted_list= dist_list.copy()
# 将列表从最小到最大排序
sorted_list.sort()

# 在原始 `dist_list` 列表中，找到存在最小距离和第二最小距离的索引编号
# 距离最小的索引号
ind1_dist= dist_list.index(sorted_list[0])
# 距离第二小的索引号
ind2_dist= dist_list.index(sorted_list[1])

# 初始化一个新的空列表，并将轮廓添加到以下两个索引处：
# 创建空列表
banana_cnts= []
# 将距离最小的索引号的轮廓附加到新建的空列表中
banana_cnts.append(contours[ind1_dist])
# 将距离第二小的索引号的轮廓附加到新建的空列表中
banana_cnts.append(contours[ind2_dist])
```

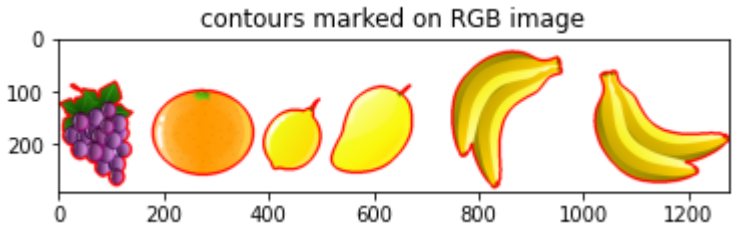
9. 绘制轮廓

在之前已经绘制过并显示的 image 图像上再叠加一次 香蕉 轮廓的绘制。

因此，输出图像将同时保持原有几个非香蕉水果的轮廓（红色），以及根据 banana_cnts= [] 列表中的轮廓重新绘制的香蕉的轮廓（蓝色）。

In [13]:

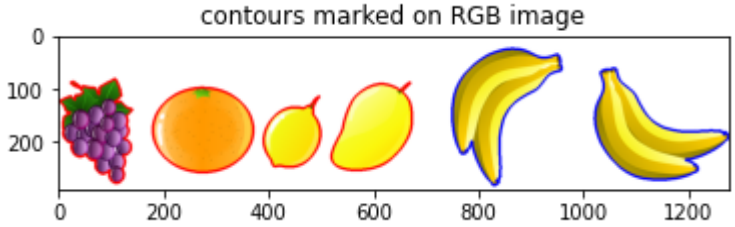
```
# 为了对比方便，先重新显示当前之前已经绘制过的 `image` 图像

# 将图像从 BGR 转换为 RGB
plt.imshow(image[:,:,::-1])
# 指定输出图像的标题
plt.title('contours marked on RGB image')
# 显示图像
plt.show()
```



之后，执行叠加绘制：

In [14]:

```
# 在已绘制过一次红色轮廓的图像image上，
# 叠加绘制banana_cnts列表内的轮廓（只有香蕉），
# 使用蓝色轮廓线BGR(255,0,0)绘制：
with_contours = cv2.drawContours(image,banana_cnts,-1,(255,0,0),3)

# 在云实验环境下忽略以下代码，避免程序尝试打开系统窗口显示图片；
# 使用matplotlib替换，使图像直接在 Jupyter Notebook 中输出。

# cv2.imshow( 'contours marked on RGB image' , with_contours )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

plt.imshow(with_contours[:,:,::-1])          # 将图像从 BGR 转换为 RGB
plt.title('contours marked on RGB image')  # 指定输出图像的标题
plt.show()                                 # 显示图像
```



实验小结

在本实验中，我们实现了轮廓匹配技术的场景，其中，我们必须检测图像中存在的两束香蕉。

首先，提供了香蕉的参考图像，以使我们能够进行匹配。参考图像中仅存在一个香蕉，而在所有水果的图像中，存在两束香蕉。每束都有两个香蕉。一束从左到右面对，而另一束处于直立位置，但轮廓匹配检测到它们都成功了。这是因为，一束两个香蕉的形状与单个香蕉的形状非常相似。

通过本实验的演示，您掌握了轮廓匹配技术从图像中检索外观相似的对象的能力。