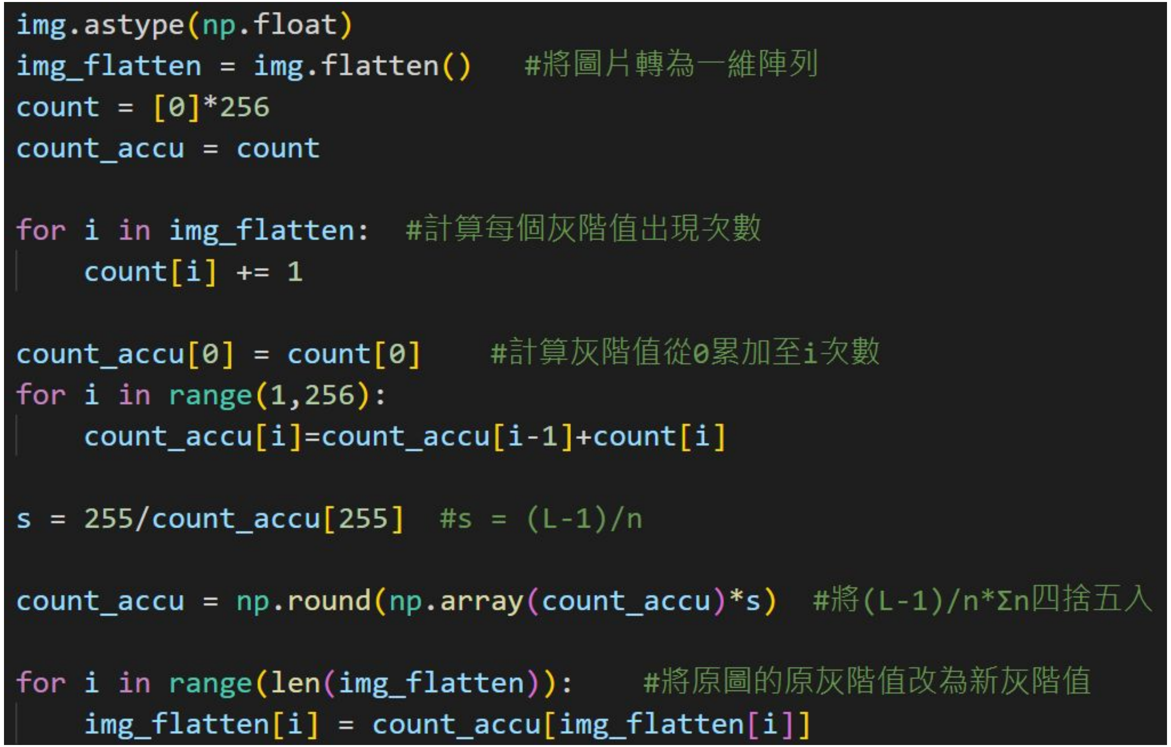
HW1-檢討

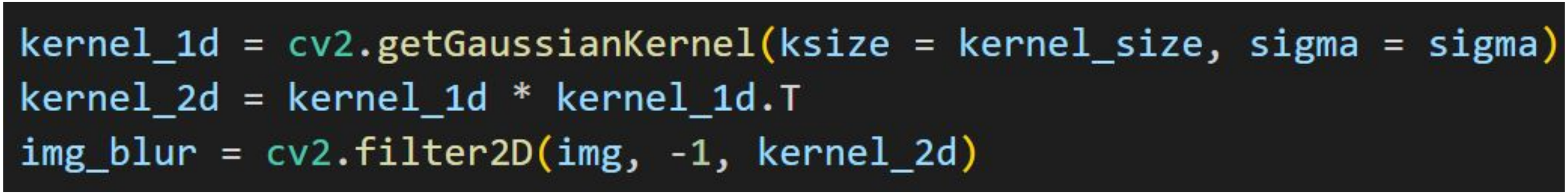
1.Histogram equalization



```
img.astype(np.float)
img_flatten = img.flatten()   #將圖片轉為一維陣列
count = [0]*256
count_accu = count

for i in img_flatten:  #計算每個灰階值出現次數
    count[i] += 1

count_accu[0] = count[0]    #計算灰階值從0累加至i次數
for i in range(1,256):
    count_accu[i]=count_accu[i-1]+count[i]

s = 255/count_accu[255]  #s = (L-1)/n

count_accu = np.round(np.array(count_accu)*s)  #將(L-1)/n*Σn四捨五入

for i in range(len(img_flatten)):    #將原圖的原灰階值改為新灰階值
    img_flatten[i] = count_accu[img_flatten[i]]
```

2.Harris corner detector

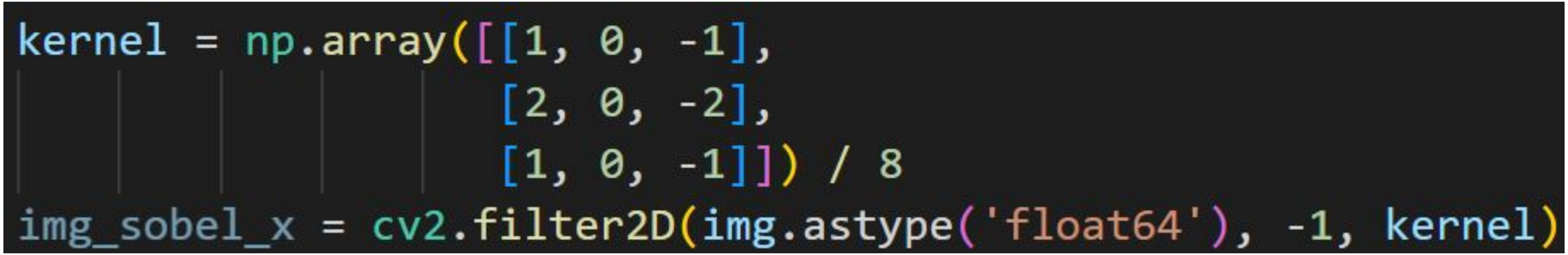
Gaussian blur



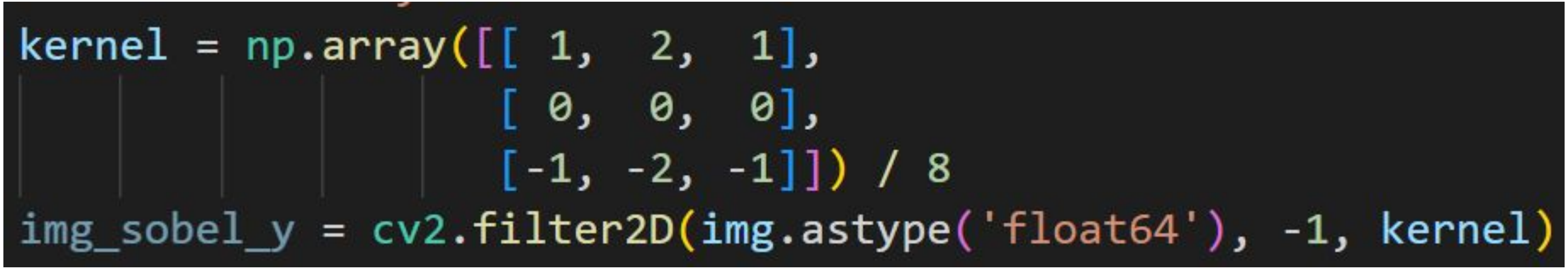
```
kernel_1d = cv2.getGaussianKernel(ksize = kernel_size, sigma = sigma)
kernel_2d = kernel_1d * kernel_1d.T
img_blur = cv2.filter2D(img, -1, kernel_2d)
```

2.Harris corner detector

Sobel operator



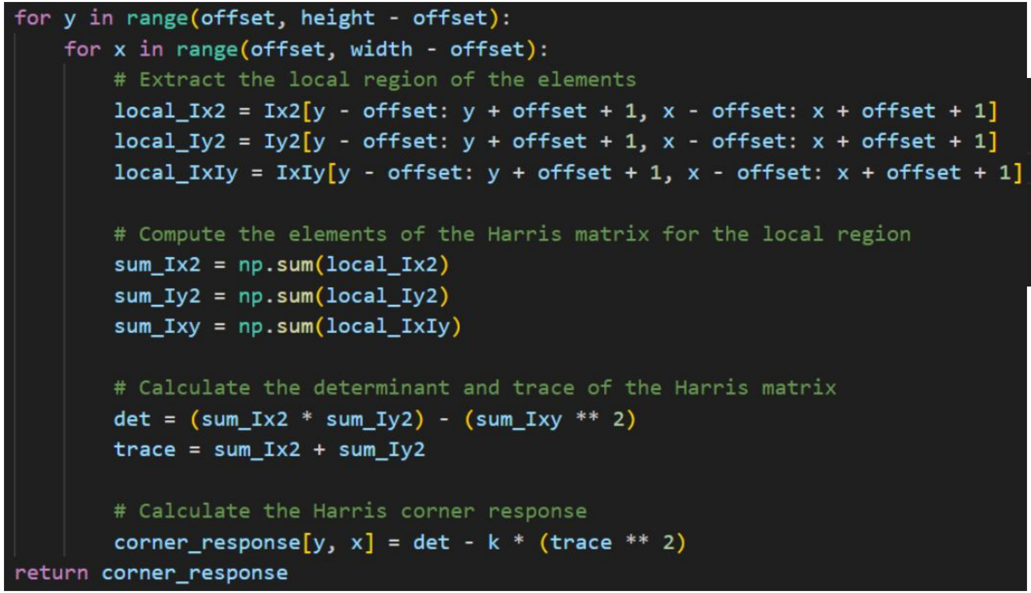
```python
kernel = np.array([[1, 0, -1],
                   [2, 0, -2],
                   [1, 0, -1]]) / 8
img_sobel_x = cv2.filter2D(img.astype('float64'), -1, kernel)
```



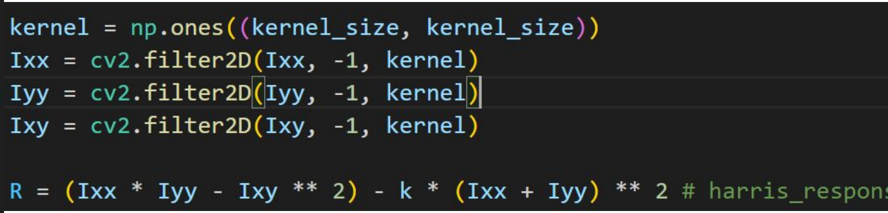
```python
kernel = np.array([[ 1,  2,  1],
                   [ 0,  0,  0],
                   [-1, -2, -1]]) / 8
img_sobel_y = cv2.filter2D(img.astype('float64'), -1, kernel)
```

2.Harris corner detector

Structure tensor & Harris response



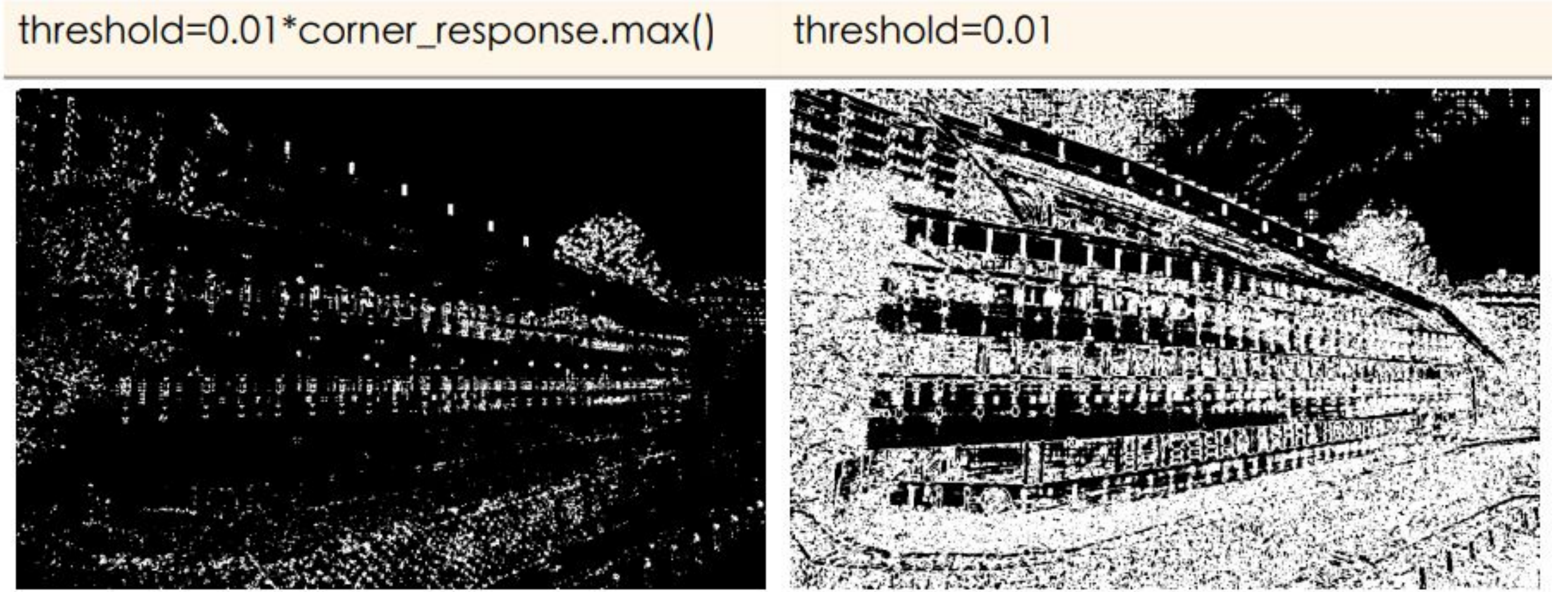
```python
for y in range(offset, height - offset):
    for x in range(offset, width - offset):
        # Extract the local region of the elements
        local_Ix2 = Ix2[y - offset: y + offset + 1, x - offset: x + offset + 1]
        local_Iy2 = Iy2[y - offset: y + offset + 1, x - offset: x + offset + 1]
        local_IxIy = IxIy[y - offset: y + offset + 1, x - offset: x + offset + 1]

        # Compute the elements of the Harris matrix for the local region
        sum_Ix2 = np.sum(local_Ix2)
        sum_Iy2 = np.sum(local_Iy2)
        sum_Ixy = np.sum(local_IxIy)

        # Calculate the determinant and trace of the Harris matrix
        det = (sum_Ix2 * sum_Iy2) - (sum_Ixy ** 2)
        trace = sum_Ix2 + sum_Iy2

        # Calculate the Harris corner response
        corner_response[y, x] = det - k * (trace ** 2)
return corner_response
```



```python
kernel = np.ones((kernel_size, kernel_size))
Ixx = cv2.filter2D(Ixx, -1, kernel)
Iyy = cv2.filter2D(Iyy, -1, kernel)
Ixy = cv2.filter2D(Ixy, -1, kernel)

R = (Ixx * Iyy - Ixy ** 2) - k * (Ixx + Iyy) ** 2 # harris_respons
```

2.Harris corner detector

Threshold

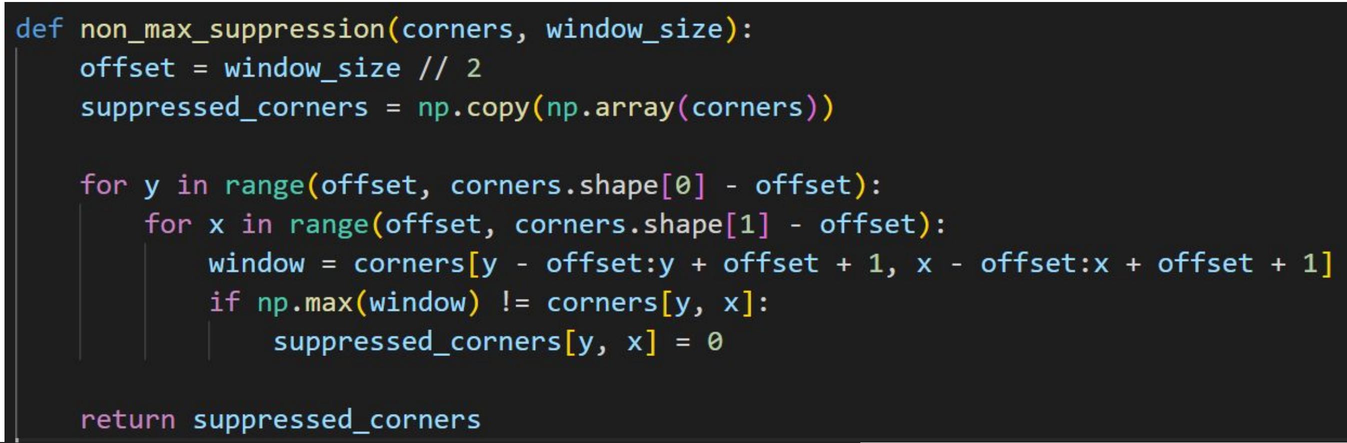


threshold=0.01*corner_response.max()

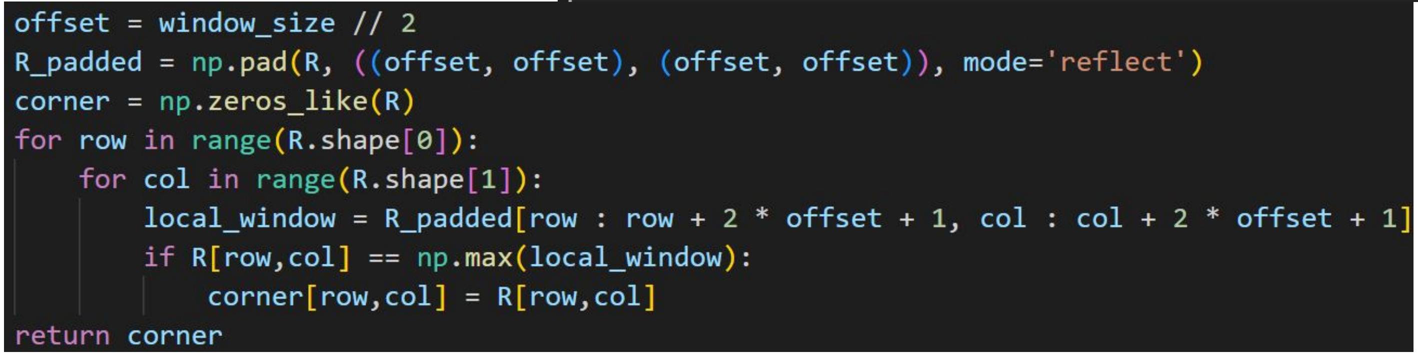
threshold=0.01

2.Harris corner detector

Non-maximum suppression

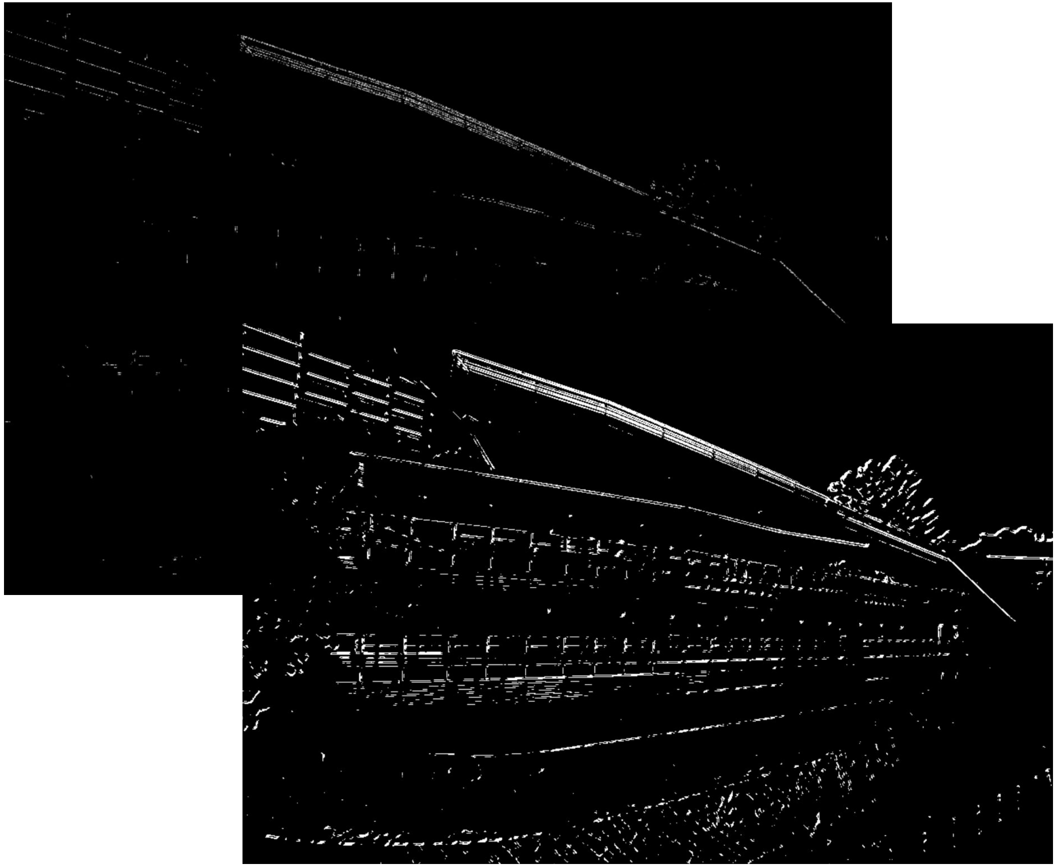


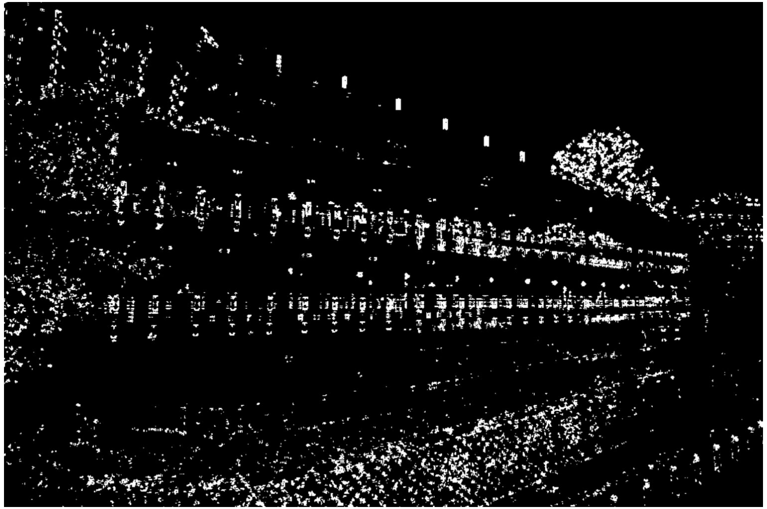
```
def non_max_suppression(corners, window_size):
    offset = window_size // 2
    suppressed_corners = np.copy(np.array(corners))

    for y in range(offset, corners.shape[0] - offset):
        for x in range(offset, corners.shape[1] - offset):
            window = corners[y - offset:y + offset + 1, x - offset:x + offset + 1]
            if np.max(window) != corners[y, x]:
                suppressed_corners[y, x] = 0

    return suppressed_corners
```



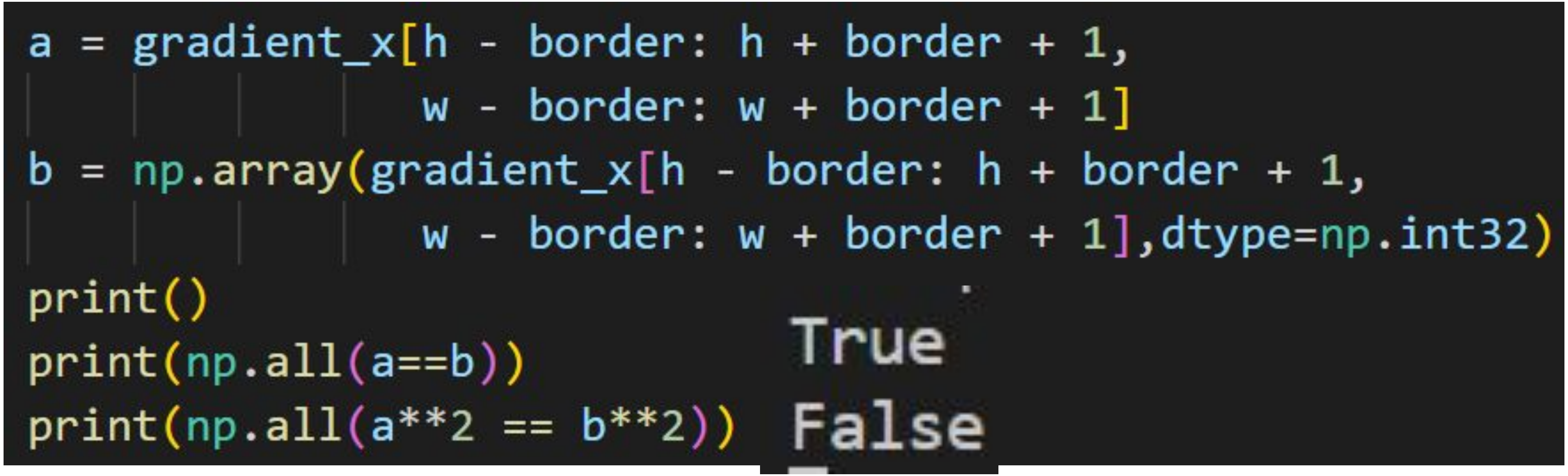
```
offset = window_size // 2
R_padded = np.pad(R, ((offset, offset), (offset, offset)), mode='reflect')
corner = np.zeros_like(R)
for row in range(R.shape[0]):
    for col in range(R.shape[1]):
        local_window = R_padded[row : row + 2 * offset + 1, col : col + 2 * offset + 1]
        if R[row,col] == np.max(local_window):
            corner[row,col] = R[row,col]
return corner
```

2.Harris corner detector





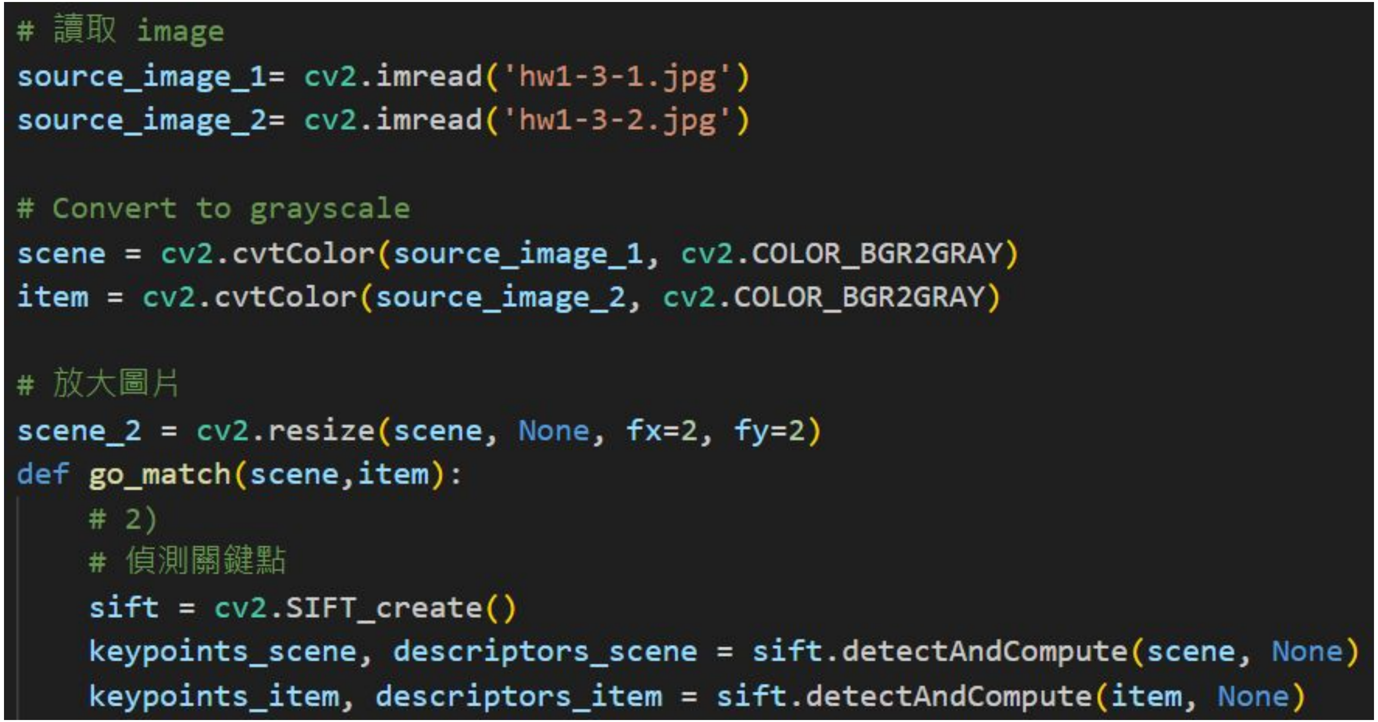
2.Harris corner detector



```
a = gradient_x[h - border: h + border + 1,
               w - border: w + border + 1]
b = np.array(gradient_x[h - border: h + border + 1,
               w - border: w + border + 1],dtype=np.int32)
print()
print(np.all(a==b))
print(np.all(a**2 == b**2))
```

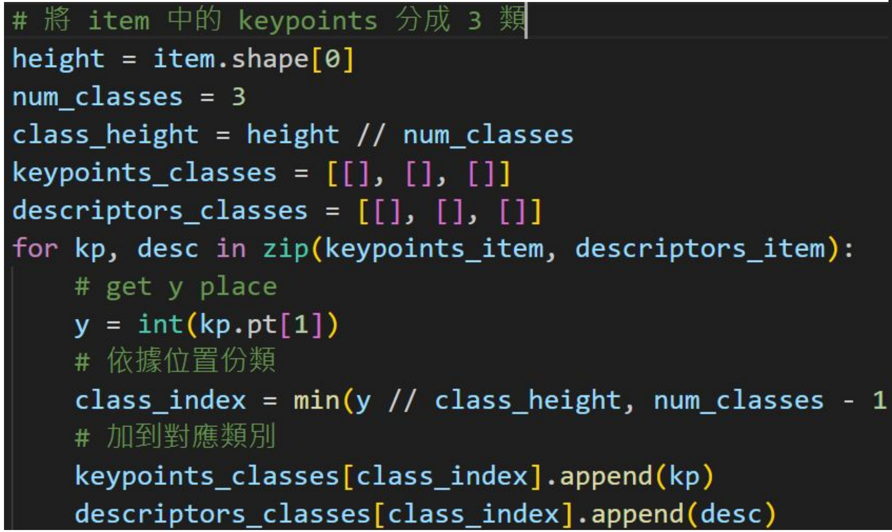
```
True
False
```

3.SIFT object recognition

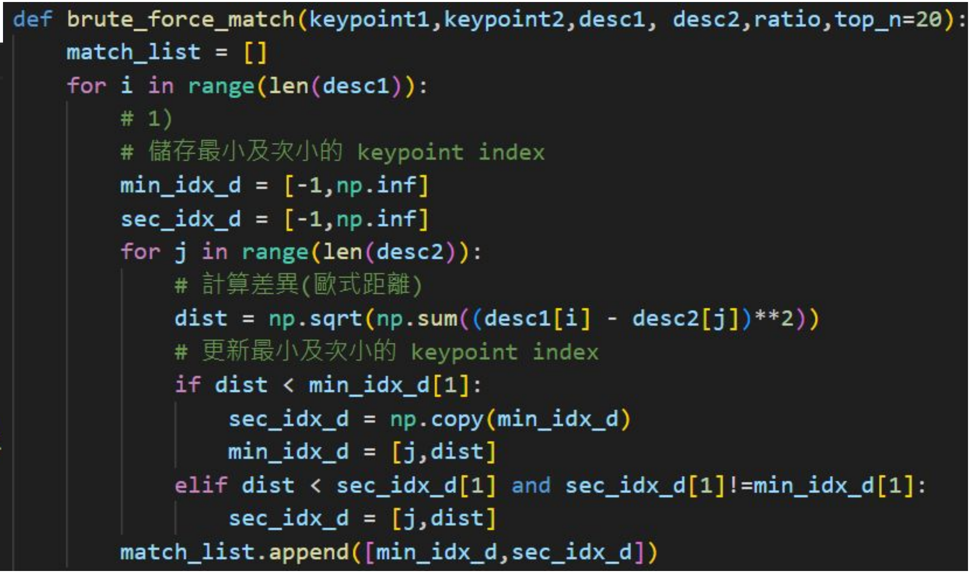


```
# 讀取 image
source_image_1= cv2.imread('hw1-3-1.jpg')
source_image_2= cv2.imread('hw1-3-2.jpg')

# Convert to grayscale
scene = cv2.cvtColor(source_image_1, cv2.COLOR_BGR2GRAY)
item = cv2.cvtColor(source_image_2, cv2.COLOR_BGR2GRAY)

# 放大圖片
scene_2 = cv2.resize(scene, None, fx=2, fy=2)
def go_match(scene,item):
    # 2)
    # 偵測關鍵點
    sift = cv2.SIFT_create()
    keypoints_scene, descriptors_scene = sift.detectAndCompute(scene, None)
    keypoints_item, descriptors_item = sift.detectAndCompute(item, None)
```

3.SIFT object recognition

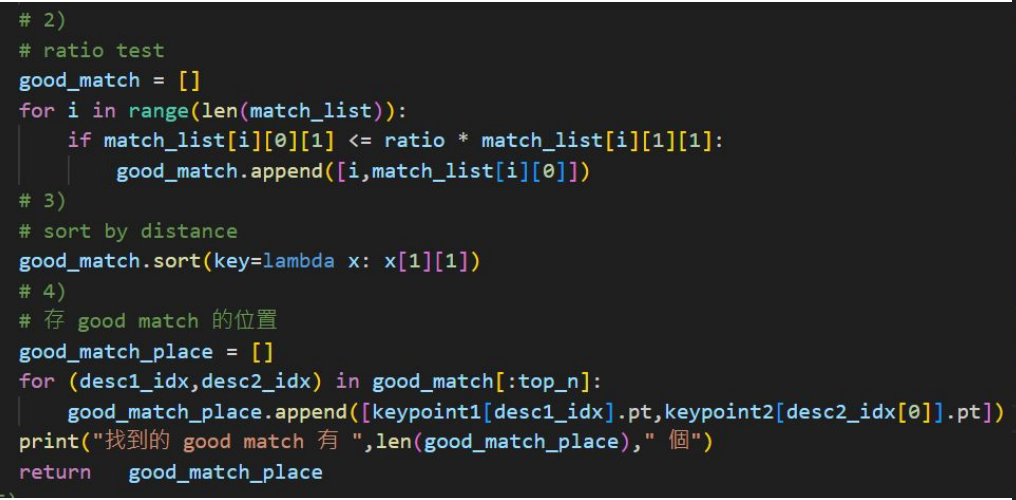


```
# 將 item 中的 keypoints 分成 3 類
height = item.shape[0]
num_classes = 3
class_height = height // num_classes
keypoints_classes = [[], [], []]
descriptors_classes = [[], [], []]
for kp, desc in zip(keypoints_item, descriptors_item):
    # get y place
    y = int(kp.pt[1])
    # 依據位置份類
    class_index = min(y // class_height, num_classes - 1
    # 加到對應類別
    keypoints_classes[class_index].append(kp)
    descriptors_classes[class_index].append(desc)
```

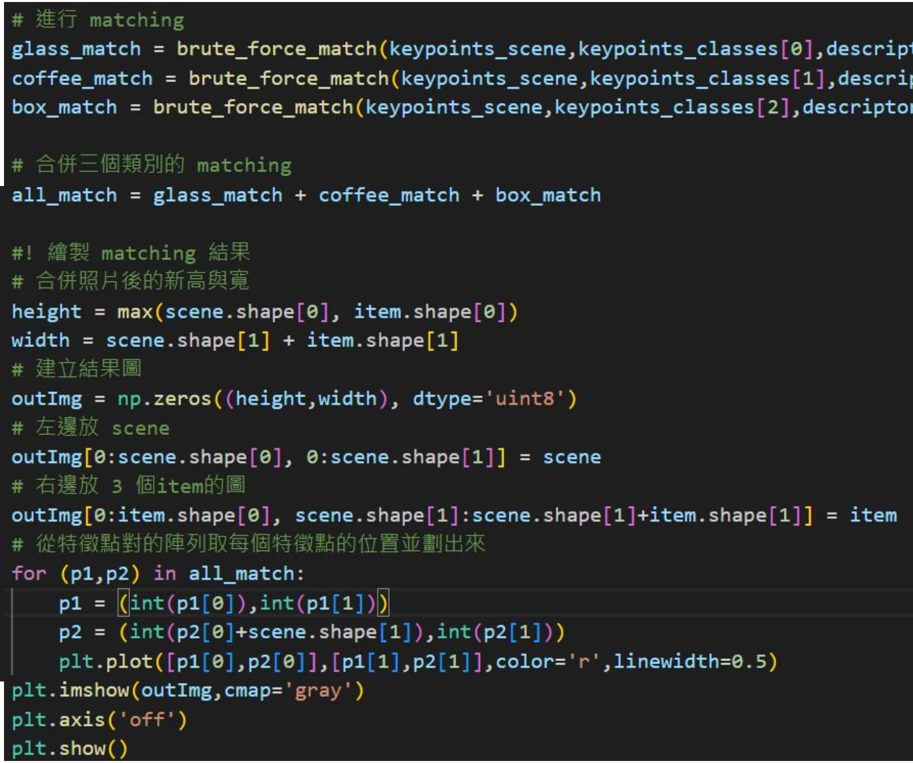


```
def brute_force_match(keypoint1,keypoint2,desc1, desc2,ratio,top_n=20):
    match_list = []
    for i in range(len(desc1)):
        # 1)
        # 儲存最小及次小的 keypoint index
        min_idx_d = [-1,np.inf]
        sec_idx_d = [-1,np.inf]
        for j in range(len(desc2)):
            # 計算差異(歐式距離)
            dist = np.sqrt(np.sum((desc1[i] - desc2[j])**2))
            # 更新最小及次小的 keypoint index
            if dist < min_idx_d[1]:
                sec_idx_d = np.copy(min_idx_d)
                min_idx_d = [j,dist]
            elif dist < sec_idx_d[1] and sec_idx_d[1]!=min_idx_d[1]:
                sec_idx_d = [j,dist]
        match_list.append([min_idx_d,sec_idx_d])
```

3.SIFT object recognition



```
# 2)
# ratio test
good_match = []
for i in range(len(match_list)):
    if match_list[i][0][1] <= ratio * match_list[i][1][1]:
        good_match.append([i,match_list[i][0]])
# 3)
# sort by distance
good_match.sort(key=lambda x: x[1][1])
# 4)
# 存 good match 的位置
good_match_place = []
for (desc1_idx,desc2_idx) in good_match[:top_n]:
    good_match_place.append([keypoint1[desc1_idx].pt,keypoint2[desc2_idx[0]].pt])
print("找到的 good match 有 ",len(good_match_place)," 個")
return   good_match_place
```



```
# 進行 matching
glass_match = brute_force_match(keypoints_scene,keypoints_classes[0],descript
coffee_match = brute_force_match(keypoints_scene,keypoints_classes[1],descri
box_match = brute_force_match(keypoints_scene,keypoints_classes[2],descriptor

# 合併三個類別的 matching
all_match = glass_match + coffee_match + box_match

#! 繪製 matching 結果
# 合併照片後的新高與寬
height = max(scene.shape[0], item.shape[0])
width = scene.shape[1] + item.shape[1]
# 建立結果圖
outImg = np.zeros((height,width), dtype='uint8')
# 左邊放 scene
outImg[0:scene.shape[0], 0:scene.shape[1]] = scene
# 右邊放 3 個item的圖
outImg[0:item.shape[0], scene.shape[1]:scene.shape[1]+item.shape[1]] = item
# 從特徵點對的陣列取每個特徵點的位置並劃出來
for (p1,p2) in all_match:
    p1 = (int(p1[0]),int(p1[1]))
    p2 = (int(p2[0]+scene.shape[1]),int(p2[1]))
    plt.plot([p1[0],p2[0]],[p1[1],p2[1]],color='r',linewidth=0.5)
plt.imshow(outImg,cmap='gray')
plt.axis('off')
plt.show()
```

3.SIFT object recognition

(b)you need to find as more as possible but not excess 20 matches for each object