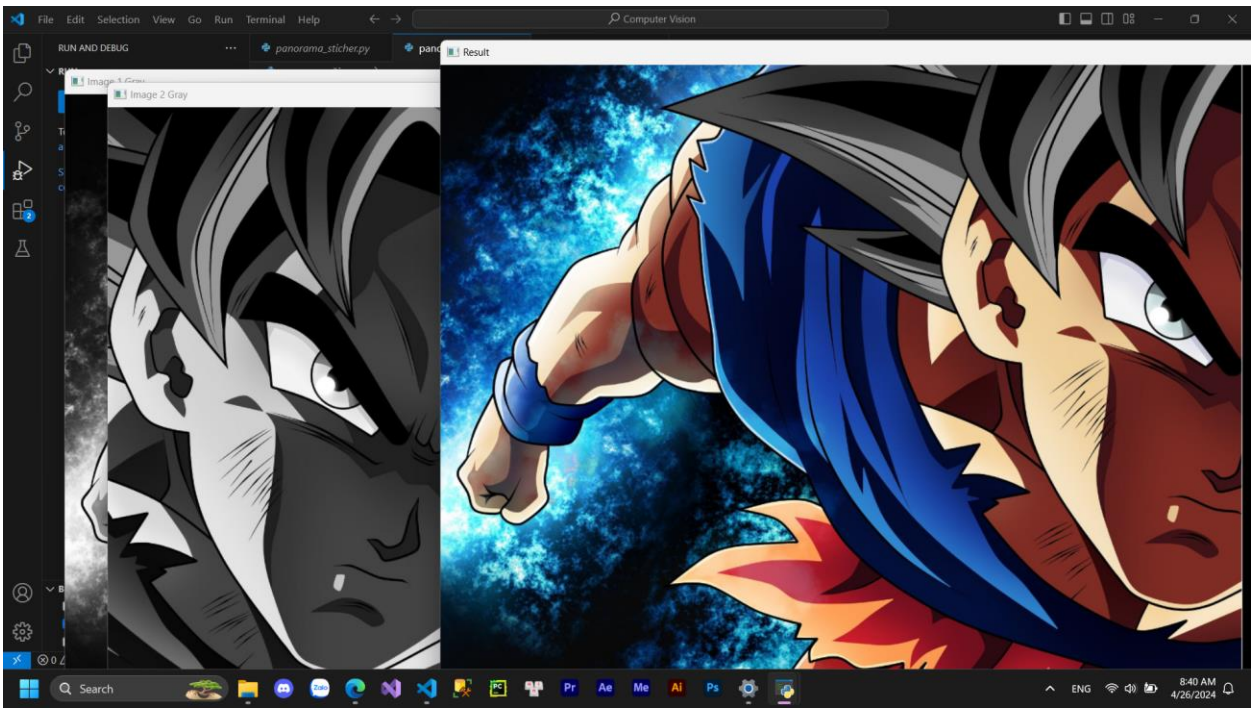
Assignment 1

```python
import sys
import cv2
import numpy as np

# Load the images
img1 = cv2.imread("panorama_1.png")
img2 = cv2.imread("panorama_2.png")
img3 = cv2.imread("panorama_3.png")

# Convert images to grayscale
img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
img3_gray = cv2.cvtColor(img3, cv2.COLOR_BGR2GRAY)


# Display grayscale images
cv2.imshow("Image 1 Gray", img1_gray)
cv2.imshow("Image 2 Gray", img3_gray)
cv2.imshow("Image 3 Gray", img3_gray)
# Create the ORB detector
orb = cv2.ORB_create(nfeatures=2000)

# Find keypoints and descriptors for all three images
keypoints1, descriptors1 = orb.detectAndCompute(img1_gray, None)
keypoints2, descriptors2 = orb.detectAndCompute(img2_gray, None)
keypoints3, descriptors3 = orb.detectAndCompute(img3_gray, None)

# Create a BFMatcher object
bf = cv2.BFMatcher_create(cv2.NORM_HAMMING)

# Find matches between image 1 and image 2
matches12 = bf.knnMatch(descriptors1, descriptors2, k=2)

# Find matches between image 2 and image 3
matches23 = bf.knnMatch(descriptors2, descriptors3, k=2)

# Filter good matches for image 1 and image 2
good_matches12 = []
for m, n in matches12:
    if m.distance < 0.6 * n.distance:
        good_matches12.append(m)

# Filter good matches for image 2 and image 3
good_matches23 = []
```

```
for m, n in matches23:
    if m.distance < 0.6 * n.distance:
        good_matches23.append(m)

# Set minimum match count
MIN_MATCH_COUNT = 10

if len(good_matches12) > MIN_MATCH_COUNT and len(good_matches23) >
MIN_MATCH_COUNT:
    # Convert keypoints to an argument for findHomography for image 1 and image 2
    src_pts12 = np.float32([keypoints1[m.queryIdx].pt for m in
good_matches12]).reshape(-1, 1, 2)
    dst_pts12 = np.float32([keypoints2[m.trainIdx].pt for m in
good_matches12]).reshape(-1, 1, 2)

    # Convert keypoints to an argument for findHomography for image 2 and image 3
    src_pts23 = np.float32([keypoints2[m.queryIdx].pt for m in
good_matches23]).reshape(-1, 1, 2)
    dst_pts23 = np.float32([keypoints3[m.trainIdx].pt for m in
good_matches23]).reshape(-1, 1, 2)

    # Find homography matrices
    M12, _ = cv2.findHomography(src_pts12, dst_pts12, cv2.RANSAC, 5.0)
    M23, _ = cv2.findHomography(src_pts23, dst_pts23, cv2.RANSAC, 5.0)

    # Warp images
    result12 = cv2.warpPerspective(img2, M12, (img1.shape[1] + img2.shape[1],
img1.shape[0]))
    result12[0:img1.shape[0], 0:img1.shape[1]] = img1

    result123 = cv2.warpPerspective(result12, M23, (result12.shape[1] +
img3.shape[1], result12.shape[0]))

   # Calculate the width and height of the stitched images
result_width = result12.shape[1] + img3.shape[1]
result_height = max(result12.shape[0], img3.shape[0])

# Warp the third image
result123 = cv2.warpPerspective(img3, M23, (result_width, result_height))

# Blend the first two images with the warped third image
result123[0:result12.shape[0], 0:result12.shape[1]] = result12

# Calculate the overlapping region
overlap_width = result_width - result12.shape[1]
```

```
# Place img3 in the correct position
result123[0:img3.shape[0], result12.shape[1]:result12.shape[1] + img3.shape[1]] =
img3[:, 0:overlap_width]

# Show the result
cv2.imshow("Result", result123)


cv2.waitKey(0)
cv2.destroyAllWindows()
```



Assignment 2:

```
import cv2
image_paths=['1.jpg','2.jpg','3.jpg']
# initialized a list of images
imgs = []

for i in range(len(image_paths)):
    imgs.append(cv2.imread(image_paths[i]))
    imgs[i]=cv2.resize(imgs[i],(0,0),fx=0.4,fy=0.4)
```

```
    # this is optional if your input images isn't too large
    # you don't need to scale down the image
    # in my case the input images are of dimensions 3000x1200
    # and due to this the resultant image won't fit the screen
    # scaling down the images
# showing the original pictures
cv2.imshow('1',imgs[0])
cv2.imshow('2',imgs[1])
cv2.imshow('3',imgs[2])

stitchy=cv2.Stitcher.create()
(dummy,output)=stitchy.stitch(imgs)

if dummy != cv2.STITCHER_OK:
# checking if the stitching procedure is successful
# .stitch() function returns a true value if stitching is
# done successfully
    print("stitching ain't successful")
else:
    print('Your Panorama is ready!!!')

# final output
cv2.imshow('final result',output)

cv2.waitKey(0)
```