```python
# Title: Programming Assignment 2
# Due date: Wednesday, September 9, 2021 at 11:59pm
# Author: Sotheanith Sok
# Description:
#  1. Segment the given rocks in “colorful rocks 2.jpg” image
#  2. Plot the result and then save the resulting image as png.
#  3. Count the total number of the gray rocks in the image and print the result.
#  4. Calculate the area of each gray rock and save the result in a file. Explain how you did that.
#  5. Estimate the center of each gray rock and plot the image with red stars on the calculated centers. Explain how you found the centers.
#  6. Upload a pdf file of your code, your answers to question 4 and 5, and the resulting images.

# ------------------------------------------------------------------------------
# Imports
from skimage import io
import matplotlib.pyplot as plt
import numpy as np
from im2bw import im2bw
from bwareaopen import bwareaopen
from bwlabeln import bwlabeln


# 1. Segment the given rocks in “colorful rocks 2.jpg” image
# Load the image and normalize it between 0 and 1
image = io.imread("./colorful rocks 2.jpg")
image = image / 255.0

# Convert image to binary image
image = im2bw(image, 0.72)

# Inverse 0 and 1 with each other for bwareopen and bwlabeln functions
image = np.subtract(1, image)

# Remove all connected components that has less than 800 pixels
image = bwareaopen(image, 800)

# Label connect components
image = bwlabeln(image)


# 2. Plot the result and then save the resulting image as png.
# Setting pyplot settings
fig = plt.figure()
fig.suptitle("Segmented Color Rock 2")
plt.xlabel("Columns")
plt.ylabel("Rows")

# Plot image
plt.imshow(image, cmap="gray")

# Save image to file
print('2. Segmented image has been saved to "segmented_colorful_rock_2.png"')
plt.savefig("segmented_colorful_rock_2.png")

# Show figure
```

```
# plt.show()


# 3. Count the total number of the gray rocks in the image and print the result.
nums_gray_rock = image.max()
print("3. Number of gray rocks is %d" % nums_gray_rock)


# 4. Calculate the area of each gray rock and save the result in a file. Explain how you did that
# We can calculate the area of each gray rock by couting the number of pixels belong to each gray rocks based on the label that we generate with bwlabeln function
labels, areas = np.unique(image, return_counts=True)
print("4. Calculate the area of each gray rock.")
for i in range(1, len(labels)):
    print("Gray rock %d has area %d pixels" % (labels[i], areas[i]))


# 5. Estimate the center of each gray rock and plot the image with red stars on the calculated centers. Explain how you found the centers
# We can caluclate the center of each gray rock by suming each pixels coordinate seperately (sum of row indexs and sum of coloumn indexs) and divide the result of the number of point.

# Calculate the center for each connected components
labels = np.unique(image)[1:]
centers = []
for label in labels:
    rows, columns = np.where(image == label)
    center_row_index = np.mean(rows)
    center_col_index = np.mean(columns)
    centers.append((center_row_index, center_col_index))

# Plot the center
for center in centers:
    plt.plot(center[1], center[0], "r*")

# Save figure to files
print('5. Result has been saved to "center_plotted_segmented_colorful_rock_2.png"')
plt.savefig("center_plotted_segmented_colorful_rock_2.png")
```

```python
# Title: Programming Assignment 2
# Due date: Wednesday, September 9, 2021 at 11:59pm
# Author: Sotheanith Sok
# Description: Converts the grayscale image I to binary image BW, by replacing all pixels in the input image with luminance greater than level with the value 1 (white) and replacing all other pixels with the value 0 (black).
# ------------------------------------------------------------------------------
# imports
import numpy as np
from skimage import color

def im2bw(I, level):
    """Converts image I to binary image BW, by replacing all pixels in the input image with luminance greater than level with the value 1 (white) and replacing all other pixels with the value 0 (black). If I isn't grayscale image, I will be converted to one.

    Args:
        I (array): image.
        level (double): luminance threshold, specified as a number in the range [0, 1].

    Returns:
        [array]: binary image.
    """
    # Check if image should be converted to grayscale
    if len(np.shape(I)) == 3:
        I = color.rgb2gray(I)

    # Convert a grayscale image to a binary image
    if len(np.shape(I)) < 3:
        I[I > level] = 1
        I[I <= level] = 0
    else:
        print("Error: Image isn't grayscale")
    return I
```

```python
# Title: Programming Assignment 2
# Due date: Wednesday, September 9, 2021 at 11:59pm
# Author: Sotheanith Sok
# Description: Remove connected components whos area is less than P pixels. Mathlab version of bwareaopen assumes that 1 is connected components and 0 is background.

# ------------------------------------------------------------------------------
# imports
import numpy as np

def bwareaopen(BW, P):
    """Removes all connected components (objects) that have fewer than P pixels from the binary image BW.

    Args:
        BW (array): binary image.
        P (int): maximum number of pixels in objects, specified as a nonnegative integer.

    Returns:
        [array]: binary image
    """
    # Find connected components by looping through all pixels that hasn't been visited.
    rows = np.shape(BW)[0]
    cols = np.shape(BW)[1]
    tag = 2
    for row in range(rows):
        for col in range(cols):
            if BW[row, col] == 1:
                BW = _find_connected_components(BW, row, col, tag)
                tag = tag + 1

    # Remove connected componets that contains less than P pixels.
    for component in range(2, tag):
        pixels = np.count_nonzero(BW == component)
        if pixels < P:
            BW[BW == component] = 0
        else:
            BW[BW == component] = 1

    return BW


def _find_connected_components(BW, initial_row, initial_col, tag):
    """Perform non-recursive flooding algorithm to find all pixels connected to a component.

    Args:
        BW (array): binary image.
        initial_row (int): starting row index.
        initial_col (int): starting column index.
        tag (int): tag used to identify this connected component.

    Returns:
        [array]: binary image with tagged area of this connected component
    """
    #Add initial row and col to a set of unvisted pixels (set is desired since we don't want duplicated unvisted pixels).
    unvisted_pixels = set()
```

```
    unvisted_pixels.add((initial_row, initial_col))

    #Loop through all unvisted pixels
    while len(unvisted_pixels) > 0:

        #Remvove the first unvisited pixel from the set
        row, col = unvisted_pixels.pop()

        #Tag the pixel
        BW[row, col] = tag

        # Add unvisted neighboring pixels to the set
        # # Top left
        if row > 0 and col > 0 and BW[row - 1, col - 1] == 1:
            unvisted_pixels.add((row - 1, col - 1))
        # Top
        if row > 0 and BW[row - 1, col] == 1:
            unvisted_pixels.add((row - 1, col))
        # Top right
        if row > 0 and col < np.shape(BW)[1] - 1 and BW[row - 1, col + 1] == 1:
            unvisted_pixels.add((row - 1, col + 1))
        # Left
        if col > 0 and BW[row, col - 1] == 1:
            unvisted_pixels.add((row, col - 1))
        # Right
        if col < np.shape(BW)[1] - 1 and BW[row, col + 1] == 1:
            unvisted_pixels.add((row, col + 1))
        # Bottom left
        if row < np.shape(BW)[0] - 1 and col > 0 and BW[row + 1, col - 1] == 1:
            unvisted_pixels.add((row + 1, col - 1))
        # Bottom
        if row < np.shape(BW)[0] - 1 and BW[row + 1, col] == 1:
            unvisted_pixels.add((row + 1, col))
        # Bottom right
        if (
            row < np.shape(BW)[0] - 1
            and col < np.shape(BW)[1] - 1
            and BW[row + 1, col + 1] == 1
        ):
            unvisted_pixels.add((row + 1, col + 1))

    return BW
```

```python
# Title: Programming Assignment 2
# Due date: Wednesday, September 9, 2021 at 11:59pm
# Author: Sotheanith Sok
# Description: Label connected components starting from 1.
# ------------------------------------------------------------------------------
# imports
import numpy as np

def bwlabeln(BW):
    """Returns a label matrix, L, containing labels for the connected components in BW.

    Args:
        BW (array): binary image.

    Returns:
        [array]: binary image contains labels unique for each connected components. Starting from 1.
    """
    # Find connected components
    rows = np.shape(BW)[0]
    cols = np.shape(BW)[1]
    tag = 2
    for row in range(rows):
        for col in range(cols):
            if BW[row, col] == 1:
                BW = _find_connected_components(BW, row, col, tag)
                tag = tag + 1

    # Adjust labeling so that it starts with 1
    BW = np.subtract(BW, 1)
    BW[BW == -1] = 0

    return BW


def _find_connected_components(BW, initial_row, initial_col, tag):
    """Perform non-recursive flooding algorithm to find all pixels connected to a component.

    Args:
        BW (array): binary image.
        initial_row (int): starting row index.
        initial_col (int): starting column index.
        tag (int): tag used to identify this connected component.

    Returns:
        [array]: binary image with tagged area of this connected component
    """
    # Add initial row and col to a set of unvisted pixels (set is desired since we don't want duplicated unvisted pixels).
    unvisted_pixels = set()
    unvisted_pixels.add((initial_row, initial_col))

    # Loop through all unvisted pixels
    while len(unvisted_pixels) > 0:

        # Remvove the first unvisited pixel from the set
        row, col = unvisted_pixels.pop()
```

```
56
57          # Tag the pixel
58          BW[row, col] = tag
59
60          # Add unvisted neighboring pixels to the set
61          # # Top left
62          if row > 0 and col > 0 and BW[row - 1, col - 1] == 1:
63              unvisted_pixels.add((row - 1, col - 1))
64          # Top
65          if row > 0 and BW[row - 1, col] == 1:
66              unvisted_pixels.add((row - 1, col))
67          # Top right
68          if row > 0 and col < np.shape(BW)[1] - 1 and BW[row - 1, col + 1] == 1:
69              unvisted_pixels.add((row - 1, col + 1))
70          # Left
71          if col > 0 and BW[row, col - 1] == 1:
72              unvisted_pixels.add((row, col - 1))
73          # Right
74          if col < np.shape(BW)[1] - 1 and BW[row, col + 1] == 1:
75              unvisted_pixels.add((row, col + 1))
76          # Bottom left
77          if row < np.shape(BW)[0] - 1 and col > 0 and BW[row + 1, col - 1] == 1:
78              unvisted_pixels.add((row + 1, col - 1))
79          # Bottom
80          if row < np.shape(BW)[0] - 1 and BW[row + 1, col] == 1:
81              unvisted_pixels.add((row + 1, col))
82          # Bottom right
83          if (
84              row < np.shape(BW)[0] - 1
85              and col < np.shape(BW)[1] - 1
86              and BW[row + 1, col + 1] == 1
87          ):
88              unvisted_pixels.add((row + 1, col + 1))
89
90      return BW
```

4. Calculate the area of each gray rock and save the result in a file. Explain how you did that.
Ans: We can calculate the area of each gray rock by counting the number of pixels belong to each gray rocks based on the label that we generate with bwlabeln function.

5. Estimate the center of each gray rock and plot the image with red stars on the calculated centers. Explain how you found the centers.
Ans: We can caluclate the center of each gray rock by suming each pixels coordinate seperately (sum of row indexs and sum of coloumn indexs) and divide the result of the number of point.