```python
from skimage.io import imread
from skimage.exposure import adjust_gamma
from skimage.morphology import dilation, square, remove_small_objects, label
from skimage.segmentation import flood_fill
from utilities import roberts, transform, unsharp_mask
import matplotlib.pyplot as plt
import numpy as np
from random import randint, choice

"""# 1. Read “balloons.jpg” image. Find the outer edges (not the patterns inside) of the air balloons."""
# load the image and use only the green channel
orig_img = imread("balloons.jpg")
img = orig_img[:, :, 2] / 255.0

# Preprocess the image such that balloons are darker and background is uniform color.
img = adjust_gamma(img, gamma=2)
img = unsharp_mask(img)
img = img < 0.12
img = dilation(img, square(3))

# Perform edges detection with Roberts Cross
img = roberts(img)
img = img > 0
bin_img = remove_small_objects(img, 75)


"""2. Count the total number of the balloons"""
labels_img = label(bin_img)
num_bln = np.max(labels_img)


"""# 3. Plot the resulting image from step 1, and as a title of your image, write the total number of the balloons you found in step 2. (No hard coding please) and then save the resulting image as a png."""
fig, ax = plt.subplots(1, 2)
fig.suptitle(f"There are {num_bln:d} balloons in the image.", fontsize=16)
ax[0].set_title("Original image")
ax[0].imshow(orig_img)
ax[1].set_title("Edges image")
ax[1].imshow(bin_img, cmap="gray")
fig.savefig("3. detected_edges.png", dpi=1000)
plt.close()


"""# 4. Choose a random air balloon in your binary image, change the pixels inside to white. Explain how you did that."""
# Pick a balloon based on label
bln = randint(1, num_bln)

# Find all pixels belong to that balloon
rows, cols = np.where(labels_img == bln)

# Find center the ballon
cen_row, cen_col = round(np.average(rows)), round(np.average(cols))

# Use flood_fill algo to fill the ballon
```

```python
fill_img = np.copy(bin_img)
fill_img = flood_fill(fill_img, (cen_row, cen_col), 1)

# Save filled image
fig = plt.figure()
plt.imshow(fill_img, cmap="gray")
fig.suptitle(f"Balloons {bln:d} is randomly picked to be fill.")
fig.savefig("4. fill_a_balloons.png", dpi=1000)
plt.close()


"""# 6. Move the balloon 20 pixels in any direction of 45-degree angle. Explain how you did that."""
# Find all pixels belong to a ballon
# Fill the ballon
fill_img = flood_fill(labels_img, (cen_row, cen_col), bln)
rows, cols = np.where(fill_img == bln)

# Create a new transform image based on the original image
tf_img = np.copy(orig_img)

# Set transform parameters
tf_row = choice([-20, 20])
tf_col = choice([-20, 0, 20])
angle = 45  # in degree

# Set pixels of the balloon's orignal location to white
for row, col in zip(rows, cols):
    tf_img[row, col, :] = 255

# Copy pixels from the balloon's orignal location to the new location
for row, col in zip(rows, cols):
    new_row, new_col = transform(
        (row, col), (cen_row, cen_col), (tf_row, tf_col), angle
    )
    if 0 <= new_row < tf_img.shape[0] and 0 <= new_col < tf_img.shape[1]:
        tf_img[new_row, new_col, :] = orig_img[row, col, :]

# Save transform image
fig = plt.figure()
plt.imshow(tf_img)
fig.suptitle(
    f"Balloons {bln:d} shifts vertically by {tf_row:d} pixels, horizontally by {tf_col:d} pixels, and rotate by {angle:d}\N{DEGREE SIGN}",
    fontsize=10,
)
fig.savefig("6. transform_balloons.png", dpi=1000)
plt.close()


"""# 7. Rotate the air balloon 60 degrees clockwise after step 6."""
# Find all pixels belong to a ballon
# Fill the ballon
fill_img = flood_fill(labels_img, (cen_row, cen_col), bln)
rows, cols = np.where(fill_img == bln)

# Create a new transform image based on the original image
tf_img = np.copy(orig_img)
```

```python

# Set transform parameters
angle = 45 + 60  # in degree

# Set pixels of the balloon's orignal location to white
for row, col in zip(rows, cols):
    tf_img[row, col, :] = 255

# Copy pixels from the balloon's orignal location to the new location
for row, col in zip(rows, cols):
    new_row, new_col = transform(
        (row, col), (cen_row, cen_col), (tf_row, tf_col), angle
    )
    if 0 <= new_row < tf_img.shape[0] and 0 <= new_col < tf_img.shape[1]:
        tf_img[new_row, new_col, :] = orig_img[row, col, :]

# Save transform image
fig = plt.figure()
plt.imshow(tf_img)
fig.suptitle(
    f"Balloons {bln:d} shifts vertically by {tf_row:d} pixels, horizontally by {tf_col:d} pixels, and rotate by {angle:d}\N{DEGREE SIGN}",
    fontsize=10,
)
fig.savefig("7. rotate_balloons_again.png", dpi=1000)
plt.close()
```

```python
import numpy as np
import math


def convolve(in_arr, kernel, stride=1, padding=0, padding_mode="edge"):
    """
    Convolve a 2d array with a giving kernel.
    Args:
        in_arr (array): input array.
        kernel (array): kernel.
        stride (int, optional): the stride of moving windows. Tuple of (int, int) can be given to control the vertical stride and the horizontal stride independently. Defaults to 1.
        padding (int, optional): size of padding for the input array. Tuple of ((int,int), (int,int)) can be given to control top, bottom, left, and right padding size independently. Defaults to 0.
        padding_mode (str, optional): padding type. More info: https://numpy.org/doc/stable/reference/generated/numpy.pad.html. Defaults to "edge".

    Returns:
        [array]: convolved array.
    """
    # Expand scaler values to tuple
    if np.isscalar(padding):
        padding = (
            (padding, padding),  # (top, bottom)
            (padding, padding),  # (left, right)
        )
    if np.isscalar(stride):
        stride = (stride, stride)  # (verticle stride, horizontal stride)

    # Calculate output array size
    out_arr_height = (
        math.floor((in_arr.shape[0] + np.sum(padding[0]) - kernel.shape[0]) / stride[0])
        + 1
    )
    out_arr_width = (
        math.floor((in_arr.shape[1] + np.sum(padding[1]) - kernel.shape[1]) / stride[1])
        + 1
    )

    # Add padding to input array
    in_arr = np.pad(in_arr, pad_width=padding, mode=padding_mode)

    # Create output array
    out_arr = np.zeros((out_arr_height, out_arr_width))

    # Perform convolution between input array and kernel
    for h in range(0, out_arr_height):
        for w in range(0, out_arr_width):
            out_arr[h, w] = np.sum(
                in_arr[
                    h * stride[0] : h * stride[0] + kernel.shape[0],
                    w * stride[1] : w * stride[1] + kernel.shape[1],
                ]
                * kernel
            )

    return out_arr


def roberts(img):
    """Perform Roberts Cross filter on a given grayscale image.

    Args:
        img (array): a given grayscale image.

    Returns:
        [array]: convolved image.
    """
    Gx = np.array([[1, 0], [0, -1]])
    Gy = np.array([[0, 1], [-1, 0]])
    roberts_x = convolve(img, Gx, padding=((0, 1), (0, 1)))
    roberts_y = convolve(img, Gy, padding=((0, 1), (0, 1)))
    img = np.sqrt(roberts_x ** 2 + roberts_y ** 2)
    return img


def gaussian_kernel(n=3, sigma=1.0):
    """Generate a 2d gaussian kernel. Credit: https://stackoverflow.com/questions/29731726/how-to-calculate-a-gaussian-kernel-matrix-efficiently-in-numpy

    Args:
        n (int, optional): size of the kernel. Defaults to 3.
        sigma (float, optional): the standard deviation used to generate the kernel. Defaults to 1.0.

    Returns:
        [array]: gaussian kernel.
    """
    ax = np.linspace(-(n - 1) / 2.0, (n - 1) / 2.0, n)
    gauss = np.exp(-0.5 * np.square(ax) / np.square(sigma))
    kernel = np.outer(gauss, gauss)
    return kernel / np.sum(kernel)


def gaussian(img, sigma=1.0):
    """Perform Gaussian filter on a given grayscale image.

    Args:
        img (array): a grayscale image.
        sigma (float, optional): the standard deviation. Defaults to 1.0.

    Returns:
        [array]: convolved image.
    """
    kernel = gaussian_kernel(sigma=sigma)
    out_img = convolve(img, kernel, padding=1)
    return out_img


def unsharp_mask(img, scaling=0.45):
    """Perform Unsharp Masking filter on a given grayscale image.

    Args:
        img (array): a grayscale image.
        scaling (float, optional): the scaling factor of unsharp masking. Defaults to 0.45.

    Returns:
        [array]: convolved image.
    """
    return img + scaling * (img - gaussian(img))


def transform(point, origin, translate=(0, 0), angle=0):
    """Transform a point to a new coordinate.

    Args:
        point (tuple): the original point. Ex: (4, 5).
        origin (tuple): the origin. Ex: (0,0).
        translate (tuple, optional): the vertical and the horizontal translation. Ex: (-10, 5). Defaults to (0, 0).
        angle (int, optional): the angle of rotation in degree. Defaults to 0.

    Returns:
        [tuple]: the new point.
    """
    # Convert angle from degree to radians
    angle = math.radians(angle)

    # Translate pixels based by translate
    point = tuple(np.add(point, translate))
    origin = tuple(np.add(origin, translate))

    # Rotate pixels based on the given angle
    row0, col0 = origin
    row1, col1 = point
    col2 = math.cos(angle) * (col1 - col0) - math.sin(angle) * (row1 - row0) + col0
    row2 = math.sin(angle) * (col1 - col0) + math.cos(angle) * (row1 - row0) + row0

    return (round(row2), round(col2))
```

4. Choose a random air balloon in your binary image, change the pixels inside to white. Explain how you did that.

Ans:

    1. Start with binary image contains edges.
    2. Using 'label' function to label all connected components.
    3. Find the number of connected components and randomly pick one.
    4. Find all pixels that have that the component's label.
    5. Find the center by averaging the x and y independently.
    6. Start from the center and use 'flood_fill' function to fill every pixel with the component's label until you reach the edge.

6. Move the balloon 20 pixels in any direction of a 45-degree angle. Explain how you did that

Ans:

    1. Start with binary image contains edges.
    2. Using 'label' function to label all connected components.
    3. Find the number of connected components and randomly pick one.
    4. Find all pixels that have that the component's label.
    5. Find the center by averaging the x and y independently.
    6. Start from the center and use 'flood_fill' function to fill every pixel with the component's label until you reach the edge.
    7. Find all pixels that have the component's label.
    8. Translate all pixels in a random direction by 20 pixels.
    9. Rotate all pixels by 45-degree in the clockwise direction with respect to the center of the balloon using Pythagorean Identities.
        Formula:
            newX = cos(θ)*(x-cx) - sin(θ)*(y-cy) + cx
            newY = sin(θ)*(x-cx) + cos(θ)*(y-cy) + cy