```
% a. SCILAB Basic commands
disp('Hello, I am a SCILAB program!')

% b. Read and display image
img = imread('your_image.jpg');  % Replace 'your_image.jpg' with the actual file name
imshow(img);

% c. Resize image
resized_img = imresize(img, [new_height, new_width]);  % Replace new_height and
new_width with desired values
imshow(resized_img);

% d. Convert color image to gray-scale
gray_img = rgb2gray(img);
imshow(gray_img);

% e. Convert image to black & white
bw_img = im2bw(img, threshold);  % Replace threshold with a suitable value
imshow(bw_img);

% f. Draw image profile
profile = mean(img, 2);  % Profile along vertical axis, adjust axis as needed
plot(profile);

% g. Separate color image into R, G, and B planes
R_plane = img(:,:,1);
G_plane = img(:,:,2);
B_plane = img(:,:,3);

% h. Create color image using R, G, and B planes
color_img = cat(3, R_plane, G_plane, B_plane);
imshow(color_img);

% i. Flow control and LOOP
for i = 1:5
   disp('This is iteration number ' + string(i));
end

% j. Write 2-D data to image file
your_2D_data = rand(100, 100);  % Replace with your actual 2-D data
imwrite(your_2D_data, 'output_image.png');  % Change 'output_image.png' to desired output
file name



% 2. Point Processing Methods
```

```
% a. Obtain Negative Image
negative_img = 255 - original_img; % Assuming original_img is your input image
imshow(negative_img);

% b. Obtain Flip Image
flipped_img = flipud(original_img); % Vertical flip
imshow(flipped_img);

% c. Thresholding
threshold_value = 128; % Adjust threshold as needed
binary_img = original_img > threshold_value;
imshow(binary_img);

% d. Contrast Stretching
min_intensity = min(original_img(:));
max_intensity = max(original_img(:));
contrast_stretched_img = (original_img - min_intensity) / (max_intensity - min_intensity) *
255;
imshow(uint8(contrast_stretched_img));
```

```
% 3. Image Arithmetic Operations

% a. Addition of Two Images
image_sum = image1 + image2; % Assuming image1 and image2 are two input images

% b. Subtract One Image from Another
image_difference = abs(image1 - image2);

% c. Calculate Mean Value of Image
mean_value = mean(original_img(:));

% d. Adjust Brightness by Changing Mean Value
brightness_adjusted_img = original_img + (desired_mean - mean_value);
```

```
% 4. Image Logical Operations

% a. AND Operation Between Two Images
and_result = image1 & image2;

% b. OR Operation Between Two Images
or_result = image1 | image2;
```

```
% c. Calculate Intersection of Two Images
intersection_result = image1 .* image2;

% d. Water Marking Using EX-OR Operation
watermarked_img = xor(original_img, watermark);

% e. NOT Operation (Negative Image)
not_img = ~original_img;



% 5. Histogram Calculation and Equalization

% a. Standard MATLAB Function
histeq_img = histeq(original_img);

% b. Program Without Using Standard MATLAB Functions
[hist, bins] = imhist(original_img);
cdf = cumsum(hist) / sum(hist);
equalized_img = interp1(cdf, bins, original_img, 'nearest');

% c. C Program (Not provided here due to text limitations)



% 6. Geometric Transformation

% a. Translation
translated_img = imtranslate(original_img, [tx, ty]);

% b. Scaling
scaled_img = imresize(original_img, scale_factor);

% c. Rotation
rotated_img = imrotate(original_img, angle);

% d. Shrinking
shrink_factor = 0.5; % Adjust as needed
shrunken_img = imresize(original_img, shrink_factor);

% e. Zooming
zoom_factor = 2; % Adjust as needed
zoomed_img = imresize(original_img, zoom_factor);
```

```
% 7. Image Noise Models and Restoration

% a. Image Restoration (Not provided here due to text limitations)

% b. Remove Salt and Pepper Noise
noise_density = 0.02; % Adjust as needed
noisy_img = imnoise(original_img, 'salt & pepper', noise_density);
denoised_img = medfilt2(noisy_img);

% c. Minimize Gaussian Noise
gaussian_noisy_img = imnoise(original_img, 'gaussian', 0, 0.01); % Adjust as needed
denoised_img = imgaussfilt(gaussian_noisy_img, 1); % Adjust standard deviation as needed

% d. Median Filter and Weiner Filter (Not provided here due to text limitations)


% 8. Noise Removal Using Spatial Filters

% a. 1-D Convolution
input_signal = randn(1, 100); % Replace with your actual 1-D signal
kernel = [0.25, 0.5, 0.25]; % Replace with your desired kernel
smoothed_signal = conv(input_signal, kernel, 'same');

% b. 2-D Convolution with 3x3 Masks
original_img = imread('your_image.jpg'); % Replace 'your_image.jpg' with the actual file
name
low_pass_filter = ones(3) / 9;
low_pass_result = conv2(original_img, low_pass_filter, 'same');

high_pass_filter = [-1, -1, -1; -1, 8, -1; -1, -1, -1];
high_pass_result = conv2(original_img, high_pass_filter, 'same');

imshowpair(low_pass_result, high_pass_result, 'montage');

% 9. Image Frequency Domain Filtering

% a. Apply FFT on Given Image
fft_img = fft2(original_img);

% b. Low Pass and High Pass Filtering in Frequency Domain
cutoff_frequency = 30; % Adjust as needed
low_pass_filter = fspecial('gaussian', [size(original_img, 1), size(original_img, 2)],
cutoff_frequency);
high_pass_filter = 1 - low_pass_filter;

low_pass_result = fft_img .* low_pass_filter;
```

```
high_pass_result = fft_img .* high_pass_filter;

% c. Apply IFFT to Reconstruct Image
reconstructed_low_pass = ifft2(low_pass_result);
reconstructed_high_pass = ifft2(high_pass_result);

% 10. Edge Detection

% MATLAB/SCILAB: Use built-in functions such as edge()
% C: Implement edge detection using convolution with appropriate masks

% 11. Image Morphological Operations

% Erosion
se = strel('square', 3); % Replace 'square' with desired shape
eroded_img = imerode(original_img, se);

% Dilation
dilated_img = imdilate(original_img, se);

% 12. Wavelet Transform

% MATLAB/SCILAB: Use built-in functions like wavedec2() and waverec()
% C: Implement wavelet transform and inverse transform using appropriate libraries
```