Contents

1 Basic Test Results

```
Mon 05 Dec 2022 08:58:58 IST
Mon 05 Dec 2022 08:58:58 IST
Archive:  /tmp/bodek.3hr76crr/intro2cs1/ex5/mallis/final/submission
  inflating: src/image_editor.py
11 passed tests out of 11 in test set named 'presubmit'.
result_code    presubmit    11    1
21 passed tests out of 21 in test set named 'separate'.
result_code    separate    21    1
21 passed tests out of 21 in test set named 'combine'.
result_code    combine    21    1
512 passed tests out of 512 in test set named 'rgb2gray'.
result_code    rgb2gray    512    1
--> BEGIN TEST INFORMATION
Test name: blur_3b
Module tested: image_editor
Function call: blur_kernel(3)
Expected return value: [[0.1111111111111111, 0.1111111111111111, 0.1111111111111111], [0.1111111111111111, 0.111111111111111
More test options: {'comment': 'Verifies rows are separate lists'}
--> END TEST INFORMATION
The test named 'blur_3b' failed.
Wrong result, input: [3]:
expected: [[0.1111111111111111, 0.1111111111111111, 0.1111111111111111], [0.1111111111111111, 0.1111111111111111, 0.11111111
actual:   [[0.1111111111111111, 0.1111111111111111, 0.1111111111111111], [0.1111111111111111, 0.1111111111111111, 0.11111111
result_code    blur_3b    wrong    1
5 passed tests out of 6 in test set named 'blur'.
result_code    blur    5    1
--> BEGIN TEST INFORMATION
Test name: applyker_one3
Module tested: image_editor
Function call: apply_kernel([[0, 128, 255], [20, 150, 200]],[[-0.5]])
Expected return value: [[0, 0, 0], [0, 0, 0]]
More test options: {}
--> END TEST INFORMATION
The test named 'applyker_one3' failed.
Wrong result, input: [[[0, 128, 255], [20, 150, 200]], [[-0.5]]]:
expected: [[0, 0, 0], [0, 0, 0]]
actual:   [[-0.0, -64.0, -127.5], [-10.0, -75.0, -100.0]]
result_code    applyker_one3    wrong    1
--> BEGIN TEST INFORMATION
Test name: applyker_one4
Module tested: image_editor
Function call: apply_kernel([[0, 128, 255], [20, 150, 200]],[[1.5]])
Expected return value: [[0, 192, 255], [30, 225, 255]]
More test options: {}
--> END TEST INFORMATION
The test named 'applyker_one4' failed.
Wrong result, input: [[[0, 128, 255], [20, 150, 200]], [[1.5]]]:
expected: [[0, 192, 255], [30, 225, 255]]
actual:   [[0.0, 192.0, 382.5], [30.0, 225.0, 300.0]]
result_code    applyker_one4    wrong    1
--> BEGIN TEST INFORMATION
Test name: applyker_three01
Module tested: image_editor
Function call: apply_kernel([[20, 70, 60, 20, 70, 60, 20, 70, 60], [90, 50, 10, 90, 50, 10, 90, 50, 10], [40, 30, 80, 40, 30
Expected return value: [[20, 70, 60, 20, 70, 60, 20, 70, 60], [90, 50, 10, 90, 50, 10, 90, 50, 10], [40, 30, 80, 40, 30, 80,
More test options: {}
--> END TEST INFORMATION
The test named 'applyker_three01' failed.
Test did not complete, exited with exitcode -15.
```

```
This probably means your code caused an exception to be raised.
result_code    applyker_three01    exception    1
--> BEGIN TEST INFORMATION
Test name: applyker_three03
Module tested: image_editor
Function call: apply_kernel([[20, 70, 60, 20, 70, 60, 20, 70, 60], [90, 50, 10, 90, 50, 10, 90, 50, 10], [40, 30, 80, 40, 30
Expected return value: [[37, 57, 53, 40, 57, 53, 40, 57, 54], [63, 50, 50, 50, 50, 50, 50, 50, 37], [47, 50, 50, 50, 50, 50,
More test options: {}
--> END TEST INFORMATION
The test named 'applyker_three03' failed.
Test did not complete, exited with exitcode -15.
This probably means your code caused an exception to be raised.
result_code    applyker_three03    exception    1
--> BEGIN TEST INFORMATION
Test name: applyker_three05
Module tested: image_editor
Function call: apply_kernel([[20, 70, 60, 20, 70, 60, 20, 70, 60], [90, 50, 10, 90, 50, 10, 90, 50, 10], [40, 30, 80, 40, 30
Expected return value: [[110, 170, 160, 120, 170, 160, 120, 170, 163], [190, 150, 150, 150, 150, 150, 150, 150, 110], [140,
More test options: {}
--> END TEST INFORMATION
The test named 'applyker_three05' failed.
Test did not complete, exited with exitcode -15.
This probably means your code caused an exception to be raised.
result_code    applyker_three05    exception    1
--> BEGIN TEST INFORMATION
Test name: applyker_three07
Module tested: image_editor
Function call: apply_kernel([[20, 70, 60, 20, 70, 60, 20, 70, 60], [90, 50, 10, 90, 50, 10, 90, 50, 10], [40, 30, 80, 40, 30
Expected return value: [[220, 255, 255, 240, 255, 255, 240, 255, 255], [255, 255, 255, 255, 255, 255, 255, 255, 220], [255,
More test options: {}
--> END TEST INFORMATION
The test named 'applyker_three07' failed.
Test did not complete, exited with exitcode -15.
This probably means your code caused an exception to be raised.
result_code    applyker_three07    exception    1
--> BEGIN TEST INFORMATION
Test name: applyker_three09
Module tested: image_editor
Function call: apply_kernel([[20, 70, 60, 20, 70, 60, 20, 70, 60], [90, 50, 10, 90, 50, 10, 90, 50, 10], [40, 30, 80, 40, 30
Expected return value: [[37, 58, 51, 41, 58, 51, 41, 58, 54], [65, 50, 48, 52, 50, 48, 52, 50, 35], [45, 49, 52, 49, 49, 52,
More test options: {}
--> END TEST INFORMATION
The test named 'applyker_three09' failed.
Test did not complete, exited with exitcode -15.
This probably means your code caused an exception to be raised.
result_code    applyker_three09    exception    1
--> BEGIN TEST INFORMATION
Test name: applyker_three11
Module tested: image_editor
Function call: apply_kernel([[20, 70, 60, 20, 70, 60, 20, 70, 60], [90, 50, 10, 90, 50, 10, 90, 50, 10], [40, 30, 80, 40, 30
Expected return value: [[30, 0, 40, 40, 0, 40, 40, 0, 10], [0, 0, 60, 0, 0, 60, 0, 0, 40], [20, 120, 0, 0, 120, 0, 0, 120, 0
More test options: {}
--> END TEST INFORMATION
The test named 'applyker_three11' failed.
Test did not complete, exited with exitcode -15.
This probably means your code caused an exception to be raised.
result_code    applyker_three11    exception    1
--> BEGIN TEST INFORMATION
Test name: applyker_three13
Module tested: image_editor
Function call: apply_kernel([[20, 70, 60, 20, 70, 60, 20, 70, 60], [90, 50, 10, 90, 50, 10, 90, 50, 10], [40, 30, 80, 40, 30
Expected return value: [[0, 80, 80, 0, 80, 80, 0, 80, 40], [160, 0, 0, 240, 0, 0, 240, 0, 0], [0, 0, 180, 0, 0, 180, 0, 0, 1
More test options: {}
--> END TEST INFORMATION
The test named 'applyker_three13' failed.
Test did not complete, exited with exitcode -15.
This probably means your code caused an exception to be raised.
result_code    applyker_three13    exception    1
```

```
--> BEGIN TEST INFORMATION
Test name: applyker_three15
Module tested: image_editor
Function call: apply_kernel([[20, 70, 60, 20, 70, 60, 20, 70, 60], [90, 50, 10, 90, 50, 10, 90, 50, 10], [40, 30, 80, 40, 30
Expected return value: [[0, 120, 60, 0, 120, 60, 0, 120, 50], [240, 0, 0, 255, 0, 0, 255, 0, 0], [0, 0, 255, 0, 0, 255, 0, 0
More test options: {}
--> END TEST INFORMATION
The test named 'applyker_three15' failed.
Test did not complete, exited with exitcode -15.
This probably means your code caused an exception to be raised.
result_code    applyker_three15    exception    1
--> BEGIN TEST INFORMATION
Test name: applyker_three17
Module tested: image_editor
Function call: apply_kernel([[20, 70, 60, 20, 70, 60, 20, 70, 60], [90, 50, 10, 90, 50, 10, 90, 50, 10], [40, 30, 80, 40, 30
Expected return value: [[0, 150, 140, 0, 150, 140, 0, 150, 100], [250, 50, 0, 255, 50, 0, 255, 50, 0], [20, 0, 255, 0, 0, 25
More test options: {}
--> END TEST INFORMATION
The test named 'applyker_three17' failed.
Test did not complete, exited with exitcode -15.
This probably means your code caused an exception to be raised.
result_code    applyker_three17    exception    1
--> BEGIN TEST INFORMATION
Test name: applyker_five2
Module tested: image_editor
Function call: apply_kernel([[20, 70, 60, 20, 70, 60, 20, 70, 60], [90, 50, 10, 90, 50, 10, 90, 50, 10], [40, 30, 80, 40, 30
Expected return value: [[31, 60, 54, 38, 58, 54, 38, 60, 56], [71, 49, 42, 59, 49, 42, 59, 50, 29], [46, 46, 51, 50, 49, 51,
More test options: {}
--> END TEST INFORMATION
The test named 'applyker_five2' failed.
Test did not complete, exited with exitcode -15.
This probably means your code caused an exception to be raised.
result_code    applyker_five2    exception    1
12 passed tests out of 24 in test set named 'applyker'.
result_code    applyker    12    1
199 passed tests out of 199 in test set named 'bilinear'.
result_code    bilinear    199    1
28 passed tests out of 28 in test set named 'resize'.
result_code    resize    28    1
136 passed tests out of 136 in test set named 'rotate'.
result_code    rotate    136    1
6 passed tests out of 6 in test set named 'edges'.
result_code    edges    6    1
465 passed tests out of 465 in test set named 'quantize'.
result_code    quantize    465    1
96 passed tests out of 96 in test set named 'quantcolor'.
result_code    quantcolor    96    1
1 passed tests out of 1 in test set named 'commandline'.
result_code    commandline    1    1
TESTING COMPLETED
```

2 image editor.py

```python
#################################################################
# FILE : image_editor.py
# WRITER : Nimrod M.
# EXERCISE : intro2cs ex5 2022-2023
# DESCRIPTION: TBA
# STUDENTS I DISCUSSED THE EXERCISE WITH: N/A
# WEB PAGES I USED: N/A
# NOTES: N/A
#################################################################

##############################################################################
#                                  Imports                                   #
##############################################################################
from ex5_helper import *
from typing import Optional
import copy
import math
import sys

##############################################################################
#                                 Constants                                  #
##############################################################################

# Indices of each channel in an RGB Pixel
RED_CHANNEL_INDEX = 0
GREEN_CHANNEL_INDEX = 1
BLUE_CHANNEL_INDEX = 2

# Values for the grayscale summation of RGB Pixel
RED_GRAYSCALE_VALUE = 0.299
GREEN_GRAYSCALE_VALUE = 0.587
BLUE_GRAYSCALE_VALUE = 0.114

# Commands
QUIT_COMMAND_VALUE = 8

##############################################################################
#                                 Functions                                  #
##############################################################################

def separate_channels(image: ColoredImage) -> List[SingleChannelImage]:
    """
    Separating a colored image to multiple separate channels.
    Can probably handle as many channels as possible (tested on single, dual and triple channels)
    :param image: The colored image.
    """
    channels = [[] for channel in range(len(image[0][0]))]

    for row in image:
        channel_row = list(zip(*row))
        for channel in range(len(channel_row)):
            channels[channel].append(list(channel_row[channel]))

    return channels

def combine_channels(channels: List[SingleChannelImage]) -> ColoredImage:
    """
    Combining a colored image separated to different channels.
    :parm channels: A list of 2D lists, each one represents the channel image.
    :return: The colored image.
    """
    image = []
    for row in zip(*channels):
        current_row = []
        for pixel in zip(*row):
            current_row.append(list(pixel))
        image.append(current_row)

    return image

def _calc_grayscale_sum(rgb_pixel):
    """
    Calculating the Grayscale Sum for each colored RGB Pixel.
    The summation is modified by constant factors.
    :param rgb_pixel: The colored pixel. Expects 3 channels.
    """
    sum_value = (rgb_pixel[RED_CHANNEL_INDEX] * RED_GRAYSCALE_VALUE) + \
                (rgb_pixel[GREEN_CHANNEL_INDEX] * GREEN_GRAYSCALE_VALUE) + \
                (rgb_pixel[BLUE_CHANNEL_INDEX] * BLUE_GRAYSCALE_VALUE)
    if 255 < sum_value:
        sum_value = 255
    elif 0 > sum_value:
        sum_value = 0
    return sum_value

def RGB2grayscale(colored_image: ColoredImage) -> SingleChannelImage:
    """
    Converts a RGB (3-channel) image to single-channel grayscale image.
    """
    return [[round(_calc_grayscale_sum(pixel)) for pixel in row] for row in colored_image]

def blur_kernel(size: int) -> Kernel:
    """
    Creates a blurring kernel, with each cell being the inverse of the size squared.
    :return: size x size blurring kernel.
    """
    return [[1/(size**2)]*size]*size

def _get_matrix_center(matrix):
    """
    Getting the center of the matrix.
    If the matrix is of a single cell, then the center is obviously 1.
    """
    # This function is not one of my proudest hacks
    kernel_center = int((len(matrix)-1)/2)
    # Getting the center of the kernel. If the kernel size is 1 then the center is 1 (the calculation above yields 0)
    return kernel_center if 0 != kernel_center else 1

def _apply_kernel_to_matrix(matrix, kernel):
    """
    Applies a kernel to matrix of the SAME size.
    Invalid matrices and kernels will most likely cause an exception.
    :param matrix: 2D List of the same size as kernel. The matrix to calculate the kernel on.
    :param kernel: 2D List of the same size as the matrix.
    """
    matrix_sum = 0
    kernel_center = _get_matrix_center(kernel)

    for row in zip(matrix, kernel):
        for pixel, kernel_cell in zip(*row):
            matrix_sum += (matrix[kernel_center][kernel_center] if pixel is None else pixel) * kernel_cell

    matrix_sum = round(matrix_sum)
    # Checkng if the sum is going out of bounds
    if 0 > matrix_sum:
        matrix_sum = 0
    elif 255 < matrix_sum:
```

```python
        matrix_sum = 255

    return matrix_sum

def _get_padded_image(image, size):
    """
    Padding an image with the given size (in pixels).
    The function does not modify the original image.
    The value of all the padded pixels is None.
    """
    padded_image = copy.deepcopy(image)

    for row_pads in range(size):
        padded_image.insert(0, [None for row_len in range(len(image[0]))]) # Insert "above"
        padded_image.append([None for row_len in range(len(image[0]))]) # Insert "below"

    for row_index in range(len(padded_image)):
        for column_pads in range(size):
            padded_image[row_index].insert(0, None) # Insert "left"
            padded_image[row_index].append(None) # Insert "right"

    return padded_image

def apply_kernel(image: SingleChannelImage, kernel: Kernel) -> SingleChannelImage:
    """
    Applying a kernel to the given image.
    The original image is not modified.
    """
    padded_image = _get_padded_image(image, _get_matrix_center(kernel))

    manipulated_image = []
    for row_index in range(len(image)):
        image_row = []

        for column_index in range(len(image[row_index])):
            current_matrix = []

            # Also not my proudest hacks
            # We give special treatment for single-cell kernels.
            if 1 == len(kernel):
                image_row.append(image[row_index][column_index] * kernel[0][0])
            else:
                for current_row in padded_image[row_index : row_index + len(kernel)]:
                    current_matrix.append(current_row[column_index : column_index + len(kernel)])

                image_row.append(_apply_kernel_to_matrix(current_matrix, kernel))

        manipulated_image.append(image_row)

    return manipulated_image

def bilinear_interpolation(image: SingleChannelImage, y: float, x: float) -> int:
    """
    Calculating the bilinear interpolation on the given image with the given coordinates.
    The given image is single-channel.
    """
    delta_x = x%1 if x != 1 else 1
    delta_y = y%1 if y != 1 else 1

    # Rounding to the ceiling or floor, according to each location requirements.
    a = image[math.floor(y)][math.floor(x)]
    b = image[math.ceil(y)][math.floor(x)]
    c = image[math.floor(y)][math.ceil(x)]
    d = image[math.ceil(y)][math.ceil(x)]

    return round((a*(1-delta_x)*(1-delta_y)) + \
                 (b*delta_y*(1-delta_x)) + \
                 (c*delta_x*(1-delta_y)) + \
                 (d*delta_x*delta_y))

def resize(image: SingleChannelImage, new_height: int, new_width: int) -> SingleChannelImage:
    """
    Resizing an image to the given height and width properties.
    The given image is single-channel.
    """
    new_image = [[0 for columns in range(new_width)] for rows in range(new_height)]

    # Taking care of all pixels
    for row_index in range(len(new_image)):
        for pixel_index in range(len(new_image[row_index])):
            new_image[row_index][pixel_index] = \
                bilinear_interpolation(image,
                                       (row_index/(len(new_image)-1))*(len(image)-1),
                                       (pixel_index/(len(new_image[row_index])-1)*(len(image[0])-1)))

    # Giving the corners a special treatment
    new_image[0][0] = image[0][0]
    new_image[0][len(new_image[0])-1] = image[0][len(image[0])-1]
    new_image[len(new_image)-1][0] = image[len(image)-1][0]
    new_image[len(new_image)-1][len(new_image[0])-1] = image[len(image)-1][len(image[0])-1]

    return new_image

def rotate_90(image: Image, direction: str) -> Image:
    """
    Rotates by 90-degress the given image.
    The image can be of multiple or single channel.
    :param direction: Either 'L' for Left, or 'R' for Right.
    """
    new_image = []
    for combination in zip(*image):
        current = list(combination)
        if 'R' == direction:
            current.reverse()
            new_image.append(current)
        if 'L' == direction:
            new_image.insert(0, current)

    return new_image

def get_edges(image: SingleChannelImage, blur_size: int, block_size: int, c: float) -> SingleChannelImage:
    """
    Creating a edge-highlighted image for the single channel image.
    """
    edges_image = []
    blurred_image = apply_kernel(image, blur_kernel(blur_size))
    thresholds_image = apply_kernel(blurred_image, blur_kernel(block_size))

    for row in zip(thresholds_image, blurred_image):
        current_row = []

        for threshold_pixel, blurred_pixel in zip(*row):
            if threshold_pixel - c > blurred_pixel:
                current_row.append(0)
            else:
                current_row.append(255)

        edges_image.append(current_row)

    return edges_image

def quantize(image: SingleChannelImage, N: int) -> SingleChannelImage:
    """
    Quantizing (hue control) the given single-channel image,
    according to the given hue constant.
    For multi-channel image quantization see 'quantize_colored_image' func.
    """
    return [[round(math.floor(pixel*(N/256))*(255/(N-1))) for pixel in row] for row in image]


def quantize_colored_image(image: ColoredImage, N: int) -> ColoredImage:
    """
    Quantizing (hue control) the given colored image,
    according to the given hue constant.
    For single-channel image quantization see 'quantize' func.
    """
    quantized_channels = [quantize(channel, N) for channel in separate_channels(image)]
    return combine_channels(quantized_channels)

def _is_single_channel(image):
    """
    Checks if an image is single channels.
    Expects an at-least 2D list.
    """
    return list != type(image[0][0])

def _handle_command_line():
    """
    Getting the image path from the command line.
    """
    if 2 != len(sys.argv):
        print("[!] Invalid parameters amount received. Usage: image_editor.py {image_path}")
        return None

    return sys.argv[1]

def _get_number_input(user_input, is_integer=True, bigger_than_one=False, is_odd=False):
    """
    Checking and converting the numerical user input.
    Use the boolean flags according to what you with to check.
    """
    if (not user_input.isdecimal()) and is_integer:
        print("[!] Received a non-integer")
        return None
    elif is_integer:
        user_input = int(user_input)
        if 0 == user_input%2 and is_odd:
            print("[!] Received an even integer, it should be odd")
            return None
        elif 1 >= user_input and bigger_than_one:
            print("[!] Number should be bigger than 1")
            return None

    if not is_integer:
        try:
            user_input = float(user_input)
        except ValueError:
            print("[!] Received invalid floating-point number")
            return None

    return user_input

def _do_action_on_image(image, action):
    """
    Automatically separates the channels from a colored image,
    and calls the action for each channel.
    If you wish to pass extra parameters to action, do it in a lambda.
    """
    new_image = None
    # Image is RGB
    if not _is_single_channel(image):
        new_image = combine_channels([action(channel) for channel in separate_channels(image)])
    else: # Image is single-channel
        new_image = action(image)
    return new_image

def _grayscale_command(image):
    """
    Wrapper for the grayscale command.
    """
    # Checking if there is only a single channel, if so, it's a grayscale
    if _is_single_channel(image):
        print("[!] Image is already grayscaled. Returning to Menu.")
        return image

    return RGB2grayscale(image)

def _blur_command(image):
    """
    Wrapper for the blur command.
    Receives a single input from the user.
    """
    kernel_size = _get_number_input(input("Enter an odd & positive kernel size: "), is_odd=True)
    if kernel_size is None:
        return image
    return _do_action_on_image(image, lambda img: apply_kernel(img, blur_kernel(kernel_size)))

def _resize_command(image):
    """
    Wrapper for the resize command.
    Receives a single input from the user.
    """
    user_input = input("Enter height & width (separated by comma): ").split(',')
    if 2 != len(user_input):
        print("[!] Incorrect amount of parameters")
        return image

    height = _get_number_input(user_input[0], bigger_than_one=True)
    if height is None:
        return image

    width = _get_number_input(user_input[1], bigger_than_one=True)
    if width is None:
        return image

    return _do_action_on_image(image, lambda img: resize(img, height, width))

def _rotate_command(image):
    """
    Wrapper for the rotate 90 degree command.
    Receives a single input from the user.
    """
    direction_input = input("Enter L(eft) or R(ight) for 90 degree rotation: ")
    if direction_input not in ['L', 'R']:
        print("[!] Incorrect parameter - Insert L or R")
        return image

    return rotate_90(image, direction_input)

def _edges_command(image):
    """
    Wrapper for the edge highlighting command.
    Receives a single input from the user.
    """
    user_input = input("Enter blur & block kernel sizes, and a constant: ").split(',')
    if 3 != len(user_input):
        print("[!] Incorrect amount of parameters")
        return image

    blur_kernel_size = _get_number_input(user_input[0], is_odd=True)
    if blur_kernel_size is None:
        return image
```

```python

    block_kernel_size = _get_number_input(user_input[1], is_odd=True)
    if block_kernel_size is None:
        return image

    constant_value = _get_number_input(user_input[2], is_integer=False)
    if constant_value is None:
        return image

    if not _is_single_channel(image):
        image = RGB2grayscale(image)

    return get_edges(image, blur_kernel_size, block_kernel_size, constant_value)

def _quantize_command(image):
    """
    Wrapper for the quantization command.
    Receives a single input from the user.
    """
    hue_input = input("Insert hue value for quantization: ")
    hue_value = _get_number_input(hue_input, bigger_than_one=True)
    if hue_value is None:
        return image

    return _do_action_on_image(image, lambda img: quantize(img, hue_value))

def _show_image_command(image):
    """
    Wrapper for the image showing command
    """
    show_image(image)
    return image

def _execute_command(image, filename):
    """
    Executing a single command from the user.
    :return: The most up-to-date image.
    """
    commands = {
        1: _grayscale_command,
        2: _blur_command,
        3: _resize_command,
        4: _rotate_command,
        5: _edges_command,
        6: _quantize_command,
        7: _show_image_command,
        8: None
    }

    user_command = None
    while not (user_command in commands.keys()):
        print("Available commands:\n \
              1: Grayscaling\n \
              2: Blurring\n \
              3: Resizing\n \
              4: Rotating\n \
              5: Edged Image\n \
              6: Quantizing\n \
              7: Show Image\n \
              8: Quit Program")
        user_input = input("Choose a command (1-8): ")
        if user_input.isdecimal():
            user_command = int(user_input)
            if not (user_command in commands.keys()):
                print("[!] Invalid command number - Only 1-8 available")
        else:
            print("[!] Invalid command - Only numbers 1-8 are available")

    if QUIT_COMMAND_VALUE == user_command:
        save_image(image, input("Insert path for the image to be saved: "))
        return None

    return commands[user_command](image)

def main():
    """
    The main program.
    Executes commands from the user until he/she ceases it.
    """
    image_path = _handle_command_line()
    if image_path is None:
        return

    current_image = load_image(image_path)
    while current_image is not None:
        current_image = _execute_command(current_image, image_path)

if __name__ == '__main__':
    main()
```