CSE 107: Lab 01: Simple Image Manipulations in Python.

Vedant Sinha

LAB: Thursday 10:30am-1:20pm

September 27, 2025

Task 1: Computing the maximum value of an image. Rotating an image.

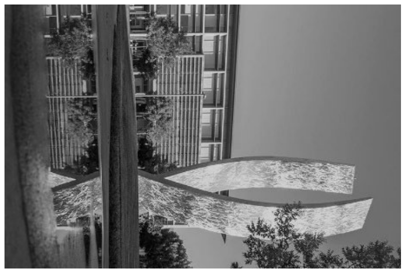


Figure 1: The grayscale Beginnings image rotated 90 degrees clockwise.

Answers for Task 1:
1) Max grayscale Beginnings value: 211
2) Max clockwise-rotated grayscale value: 211
3) Yes. Rotation reorders pixels but doesn't change pixel intensities, so extrema are preserved.
4) Most difficult: implementing correct index mappings for 90° rotations and ensuring shapes swap without hardcoding; also avoiding built-ins while keeping loops correct and efficient.

Task 2: Writing a function that computes the inverse of a grayscale image.

Figure 2: The inverse of the Tree image.

Answers for Task 2:
1) Maximum pixel value of inverse image: 236
2) It equals 255 − min(original), inversion maps low originals to high inverses.
3) Most difficult: ensuring the inverse math and loop indices are correct while avoiding built ins.

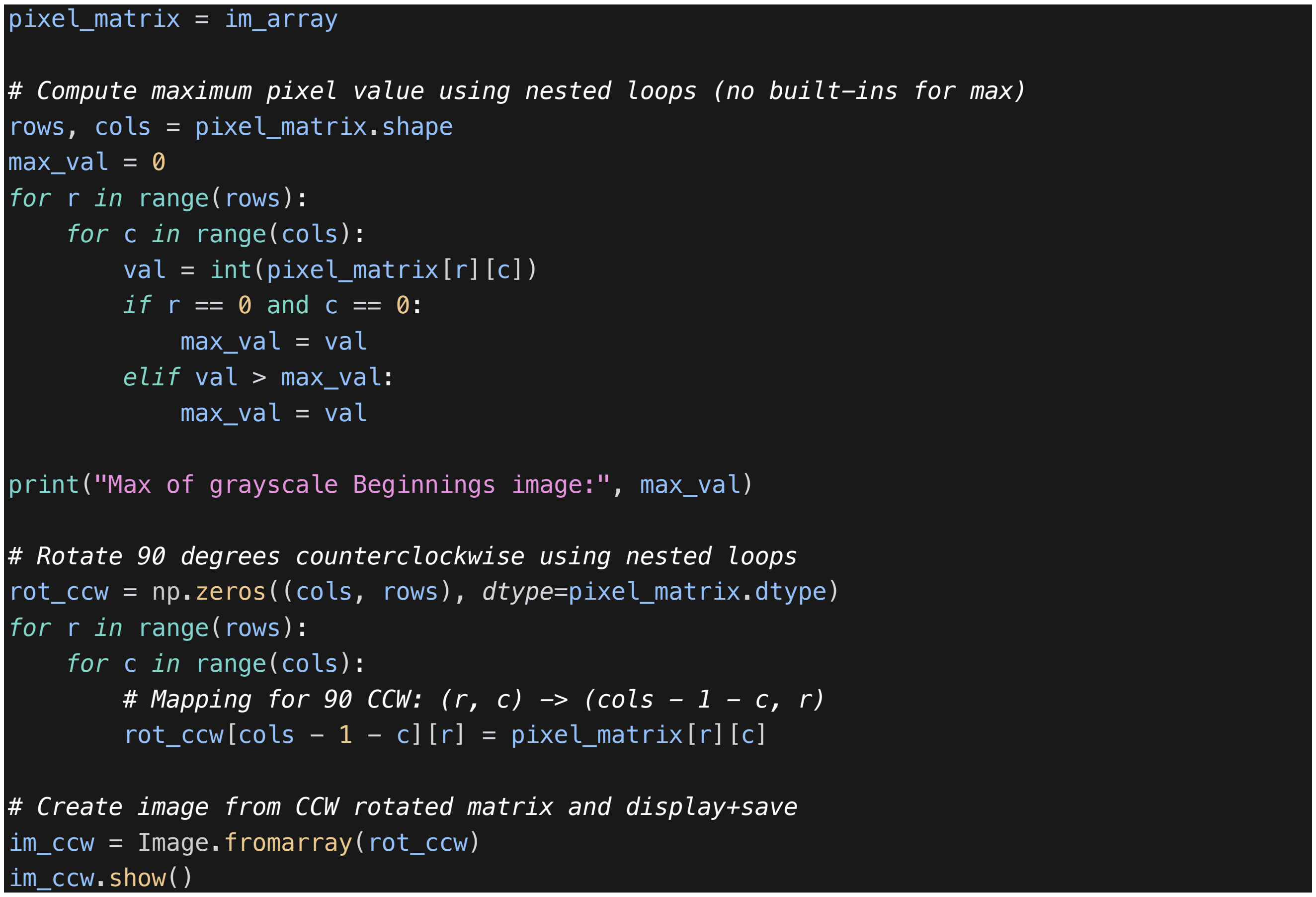
Task 3: Creating a gradient grayscale image. Computing the image average.

Figure 3: The gradient image.

<Your answers to the task questions>
1) Average pixel value: 127.5
2) Expected because values are uniformly 0..255 in each row, so the mean is (0+255)/2.
3) Most difficult: ensuring the gradient fills correctly using loops and maintaining dtype as uint8 when saving.

Task1.py

```
# Import pillow from PIL import
Image, ImageOps

# Import numpy import
numpy as np from numpy
import asarray

# Read the image from file. im =
Image.open('Beginnings.jpg')

# Show the image.
im.show()

# Convert image to gray scale.
im_gray = ImageOps.grayscale(im)

# Show the grayscale image.
im_gray.show()

<the rest of your code>
```



```
pixel_matrix = im_array

# Compute maximum pixel value using nested loops (no built-ins for max)
rows, cols = pixel_matrix.shape
max_val = 0
for r in range(rows):
    for c in range(cols):
        val = int(pixel_matrix[r][c])
        if r == 0 and c == 0:
            max_val = val
        elif val > max_val:
            max_val = val

print("Max of grayscale Beginnings image:", max_val)

# Rotate 90 degrees counterclockwise using nested loops
rot_ccw = np.zeros((cols, rows), dtype=pixel_matrix.dtype)
for r in range(rows):
    for c in range(cols):
        # Mapping for 90 CCW: (r, c) -> (cols - 1 - c, r)
        rot_ccw[cols - 1 - c][r] = pixel_matrix[r][c]

# Create image from CCW rotated matrix and display+save
im_ccw = Image.fromarray(rot_ccw)
im_ccw.show()
```

```python
im_ccw.save('Beginnings_gray_rotated_ccw.jpg')

# Rotate 90 degrees clockwise using nested loops
rot_cw = np.zeros((cols, rows), dtype=pixel_matrix.dtype)
for r in range(rows):
    for c in range(cols):
        # Mapping for 90 CW: (r, c) -> (c, rows - 1 - r)
        rot_cw[c][rows - 1 - r] = pixel_matrix[r][c]

# Create image from CW rotated matrix and display+save
im_cw = Image.fromarray(rot_cw)
im_cw.show()
im_cw.save('Beginnings_gray_rotated_cw.jpg')

# Compute maximum pixel value of the clockwise rotated image using nested loops
rows_cw, cols_cw = rot_cw.shape
max_val_cw = 0
for r in range(rows_cw):
    for c in range(cols_cw):
        val = int(rot_cw[r][c])
        if r == 0 and c == 0:
            max_val_cw = val
        elif val > max_val_cw:
            max_val_cw = val

print("Max of clockwise rotated grayscale image:", max_val_cw)
```

Task2.py

```
 # Import pillow from PIL
import Image, ImageOps

# Import numpy import
numpy as np from numpy
import asarray

# Read the image from file.
im = Image.open('Tree.tif')

# Show the image.
im.show()

# Print the image mode.
print("image mode is:", im.mode)

# Create numpy matrix to access the pixel values.
im_pixels = asarray(im)

# Import myImageInverse from myImageInverse

from MyImageFunctions import myImageInverse

im_inv_pixels = myImageInverse(im_pixels)

# Create an image from im_inv_pixels. im_inv =
Image.fromarray(np.uint8(im_inv_pixels))

# Show the inverse image.
im_inv.show()
```

```
# Save the inverse image to a file.
im_inv.save("Tree_inv.tif")
```

```
<the rest of your code>
```

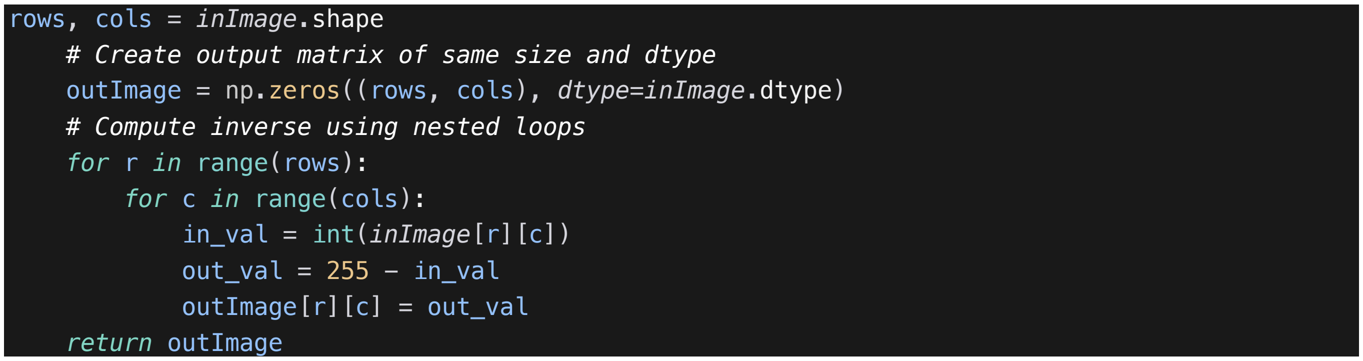
```python
# Compute maximum pixel value of the inverse image using nested loops (no built-ins)
rows_i, cols_i = im_inv_pixels.shape
max_inv = 0
for r in range(rows_i):
    for c in range(cols_i):
        val = int(im_inv_pixels[r][c])
        if r == 0 and c == 0:
            max_inv = val
        elif val > max_inv:
            max_inv = val
print("Max of inverse Tree image:", max_inv)

# Also compute the minimum of the original image using nested loops (for relationship)
rows_o, cols_o = im_pixels.shape
min_orig = 0
for r in range(rows_o):
    for c in range(cols_o):
        val = int(im_pixels[r][c])
        if r == 0 and c == 0:
            min_orig = val
        elif val < min_orig:
            min_orig = val
```

```
print("Min of original Tree image:", min_orig)
print("255 - min(original) =", 255 - min_orig)
```

MyImageFunctions.py

```
# MyImageFunctions.py

# Import pillow from PIL import
Image, ImageOps

# Import numpy import
numpy as np from numpy
import asarray

def myImageInverse( inImage_pixels ):
    # This function takes as input a numpy matrix representing a grayscale image and
    # outputs another numpy matrix which is the image inverse of the input.
    # That is, for each pixel, output_value = 255 - input_value
    # #
    Syntax:
    #   out_numpy_matrix = myImageInverse( in_numpy_matrix )
    # #
    Input:
    #   in_numpy_matrix = the grayscale values of the input image
    # #
    Output:
    #   out_numpy_matrix = the grayscale of the inverse image
    # #
    History:
    #   S. Newsam   9/12/25     Created
    #   S. Newsam   9/13/25     Modified to use numpy instead of PIL image
```

<the rest of your code>



```
rows, cols = inImage.shape
    # Create output matrix of same size and dtype
    outImage = np.zeros((rows, cols), dtype=inImage.dtype)
    # Compute inverse using nested loops
    for r in range(rows):
        for c in range(cols):
            in_val = int(inImage[r][c])
            out_val = 255 - in_val
            outImage[r][c] = out_val
    return outImage
```

Task3.py

```
# Import pillow from PIL import
Image, ImageOps

# Import numpy import
numpy as np from numpy
import asarray

# The size of the gradient image.
rows = 100
cols = 256

# Create a numpy matrix of this size.
im_pixels = np.zeros(shape=(rows, cols))

<the rest of your code>
```

```
for r in range(rows):
    for c in range(cols):
        im_pixels[r][c] = c

# Create image from numpy matrix (ensure uint8 type)
im_grad = Image.fromarray(np.uint8(im_pixels))

# Display the image
im_grad.show()

# Save the image as .tif
```

```python
im_grad.save('Gradient.tif')

# Compute the average pixel value using nested loops (no built-ins)
total = 0
count = 0
for r in range(rows):
    for c in range(cols):
        total += int(im_pixels[r][c])
        count += 1

avg = total / count if count > 0 else 0
print("Average pixel value of gradient image:", avg)
```