1 Introduction

The goal of this project is to write a program that allows one to embed (secret) messages as invisible data into (PNG) images. This is done by slightly changing the brightness values in the picture but without it having any visible effect. In a second step we can try to embed another image by a similar approach.

2 General Information about Images

The images used for the following project should be of PNG type, which is a lossless format. This ensures no data is corrupted/lost when the file is stored to disk. Furthermore they should be 8 bit images. This means each of the three color components (red, green and blue) can take an integer value in the range 0 to $2^8 - 1 = 255$.

An 8-bit integer in binary form looks like 0b10100101. Note that one can use binary numbers directly in python if entered in the above form. To get the decimal representation you can use this formula: Let $b_7 b_6 \dots b_0$ be a binary number then:

$$\sum_{i=0}^{7} b_i \, 2^i$$

is the same number in decimal form. Note that in Python bits and bytes are both treated as integers. Here we just mean a bit is one of the two integers 0 and 1. And a byte is an integer number between 0 and 255.

For the project we want to store some hidden message into an image and be able to later extract it again. It should not be visible by eye that the image has been altered. To accomplish this we modify the Least Significant Bit (LSB) of each color value in each pixel. This means, we make each color a very little more bright or more dark, but with the smallest possible value so the change is not visible to the human eye.

3 Some Information about Binary Operators

The following operations might come in handy while working with binary numbers.

```
# Bitwise AND
print 0b11110101 & 0b11000001
# This will output: 193 (0b11000001 in binary form)

# Left shift and right shift
0b10110000 >> 1 # = 0b01011000
0b10110000 << 1 # = 0b101100000

# Modulo
```

```
12 % 2 # = 0
13 % 2 # = 1
```

4 Working with Image Files

To read and modify an image you can follow the following example.

```
import Image # Import the necessary module for image handling

image  = Image.open('filename.png')
pixels = image.load() # returns pixels in a dictionary
width  = image.size[0]
height = image.size[1]
# Store the color components for red, green, blue
r,g,b  = pixels[293,133]

# modify some pixel (delete the blue component)
pixels[1,2] = r,g,0

# greyscale conversion (of pixel at pos. 23,12)
r,g,b  = pixels[23,12]
a      = (r + b + g) / 3
pixels[23,12] = a,a,a

# Apply to each pixel value
output = image.point(lambda c : 0b10000000 & c )

# Save the modifications to a file
output.save('result.png')

img = Image.open('face.png')
ri,gi,bi = img.split()
ri.show() # show red component
ri = ri.point(lambda i : 0) # kill red component
img = Image.merge("RGB", (ri,gi,bi))
img.show()

img = Image.open('face.png')
# set too bright components all to zero. leave others as is.
out = img.point(lambda l: 0 if l > 100 else l)
out.show()
```

See `http://effbot.org/imagingbook/image.htm` for a complete description of the image module.

5 Tasks

5.1 Understand the Problem

Have a look at the provided template file. Make sure you understand what all parts do and what the missing parts are supposed to be doing.

5.2 Implement the Empty Functions

Implement the missing parts of the functions in the template file.

1. Easy:
   - `openimage` takes a string as argument and returns an Image object. See the above example on how to do this. Furthermore it should check if the image format is "PNG" and result in an error if it is not of this type.
   - `saveimage` takes an Image object and a file name as arguments. It saves the image given by the Image object in the file specified by filename.
   - `showimage` takes an image object and displays it on the screen.
   - `getlsb` Takes a byte as argument and returns its least significant bit.
   - `setlsb` Takes a byte and a bit as argument and changes the LSB of the byte to match bit.
   - `bitlisttobyte` Take a list containing 8 bits and convert it into a byte (integer).
   - `addmagicstring` We need a function that adds some magic marker to the messages we embed in the images. This helps us to later separated images with messages from normal images. This just wraps any message into some string.
   - `checkmagic` This checks if an extracted message from an image contains a magic string.
2. Medium:
   - `messagetobitstream` Takes a string as argument and converts it into a stream of bits. This can be solved by using a for loop over all characters in the string and another for loop over all (8) bits in each character. Use the `ord` function to convert a character into its byte representation.
   - `getlsbfromimage` This accesses each pixel and each color component in sequential order and returns the lsb of each value.

   Test that everything is working and you can successfully embed text messages into image files and extract them again.

5.3 Find Hidden Image

In the file "hiddenimage.png" there is an image hidden inside the picture. Try if you can extract it. Note: the image has the same dimensions as the original image, but only uses 1 bit per color. Write the Python code necessary to extract and display the hidden image. You can experiment in the Python shell first and then write the code you found into the template as a function `findhiddenimage`.

5.4 Embed Image into Image

Write a function which embeds a secret image into another image. In the end result it should not be visible that an image has been embedded. Write a function to extract the image again. Maybe you can use the code from the previous task.

5.5 Improvements

There are a few ideas for improvements that could be done after the project is running.

- The magic string is not escaped when added to the message. So this causes a problem if someone wants to embed the magic string itself in the image. Find a way to avoid this problem. (medium)
- Implement some simple encryption. (for example using PyCrypt, easy)
- Write some test cases to see that all functions perform as expected. (easy)
- As written the program does not support non-ascii characters. In particular letters like ä, ö, ü, are not supported. Find a way to fix this. (May be difficult)