This assignment is not required. You may choose to complete it for extra credit in your assignment grade. It is worth up to 75% the number of points of a regular assignment.

PIC 16, Winter 2018 – (Optional) Assignment 4M

Assigned 1/29/2018. Code (a single `.py` file) due by the end of class 2/2/2018 on CCLE. Hand in a printout of this document with the self-assessment portion filled out by the end of class on 2/2/2018.

In this homework, you will learn to read from a Windows bitmap (`.bmp`) file, interpret the data as an image, and write a plain text file that can be viewed as a corresponding stereogram ("Magic Eye").

Example files include `PIC.bmp`, the original bitmap file; `PIC_img.txt`, the bitmap interpreted a string of "1"s and "0"s; and `PIC_stereogram.txt`, an example ASCII stereogram generated from the bitmap. However, you can also generate stereograms from your own monochrome bitmaps created in MS Paint.

Tasks

- Write a function `load_img` that accepts as an argument the filename of a Windows bitmap (`.bmp`) and reads in the file. Note that the data does not represent text; keep this in mind when considering which *mode* should be used when opening the file. Nonetheless, the data type Python uses to store the file information is a `str`.
  Note that if you try to `print` this string to the console, you will get a lot of unusual characters, such as . On the other hand, if you `print` the `repr` of the string to the console (or if you just enter the name of the string variable into the console and press enter), you will see an alternative representation consisting almost entirely of sequences like `\x00`. The `\x` prefix means that the next two characters are to be interpreted as the hexadecimal code (e.g. `\x00` represent 0, and `\xff` represent 255) of a single byte of data. In the first string representation, the funny characters are one way of representing bytes outside the range of the printable ASCII characters. The latter representation allows us to more easily interpret these values, so this is the one we'll want to work with. Save this representation of the file to a new text file `PIC_hex.txt`, which should look like this. Don't worry about there being a few normal symbols like `B`, `M`, and `?`. These simply represent bytes that correspond with a printable ASCII character.
- The file is divided into several portions, but we'll refer to the first few portions of the file collectively as the "header" and the remainder as the "data". The header specifies how the data is to be interpreted, such as the dimensions (in pixels) of the image and the colors to be used. In order to extract the information you'll need from the header, write a function `read_hex` that accepts three arguments:
  - `bmp`, the entire string of bytes read from the bitmap file,
  - `offset`, the index of the first byte you need to extract, and
  - `size`, the number of bytes to extract starting in that location,

  and returns the *decimal* integer representation of the extracted information.
  For example, according to the bmp file format information on Wikipedia, the header contains the total size of the file in bytes. It is located 2 bytes into the file and is 4 bytes long. The first 10 bytes in the string representation of the file are: `B M \x12 \x02 \x00 \x00 \x00 \x00 \x00 \x00`. Beginning at an offset of two and with size 4, the data we want is: `\x12 \x02 \x00 \x00`. Using the `ord` function, we can get the values of each of these hexadecimal keys in decimal: 18 2 0 0.
  It gets a little weird. By convention, the bytes are ordered from least significant to most significant. Let's consider how this would look like in decimal first: it's as if we wrote the decimal number 12,345,678 as 78 56 34 12. To reconstruct the number, we'd have to perform the following: $78 \times 10^0 + 56 \times 10^2 + 34 \times 10^4 + 12 \times 10^6$. For the hexadecimal information, $18 \times 16^0 + 2 \times 16^2 + 0 \times 16^4 + 0 \times 16^6 = 530$. You can confirm in Windows (by right clicking the file and selecting "Properties") that `PIC.bmp` is 530 bytes in size.

This assignment is not required. You may choose to complete it for extra credit in your assignment grade. It is worth up to 75% the number of points of a regular assignment.

Generalize from this example to write `read_hex`. I suggest that you begin by writing it however you want – but since this is an extra credit assignment, for full credit you should write it in a single line using a generator expression and `xrange` (to practice these features of the Python language).

- Let's turn our attention back to `load_img`; we won't be done until it returns something like `PIC img.txt`. The next thing it will need to do, after loading the string of bytes, is to extract four pieces of information from the header:
    - the offset of the byte at which the data portion of the file begins, which is specified in the header at offset (index) 10 and is four bytes long;
    - the width of the image, which begins at offset 18 and is four bytes long;
    - the height of the image, which begins at offset 22 and is four bytes long; and
    - the number of bits used to describe the color of each pixel, which begins at offset 28 and is only two bytes long.

  Within the `load_img` function, use `read_hex` to get that information. So that you can check your work, `PIC.bmp` is 69 px wide by 39 px high, the bitmap data begins at index 62, and it's purely black and white (not grayscale), so only a single bit (not byte) is needed to represent each pixel.
- Our program is only going to be able to create stereograms from black and white images, so check that the number of bits used to describe each pixel is 1. If not, raise a `ValueError` reporting that the file being loaded doesn't contain a monochrome image. (You can test this with `PIC2.bmp`, which uses 24 bits per pixel – that's one byte for the intensity of red, another for the intensity of green, and one for blue. We'll see 24-bit images like this later in Track A.)
- Extract the data portion of the file, which begins at the offset specified in the header and continues until the end of the file. This is just a one-dimensional stream of bytes, but our image is 2D, so we need to break this up into rows.
    - Determine the number of bytes used to represent each row by dividing the length of the data by the height of the image. Of course, this isn't the same as the width of the image in pixels, nor is it even the width of the image in pixels divided by 8 and rounded up. It turns out that according to the bmp specification, it's the width of the image in pixels divided by 8 and rounded up to the nearest multiple of 4! This probably stems from the fact that the `bmp` format can support 32-bit images, in which 4 bytes are needed to describe the color of each pixel.
    - Break the string of bytes into a list of strings of this width. (This could probably be a generator rather than a list, and you're welcome to try this, but not required.)
- Since the image is monochrome, when the bytes in the data portion are expressed in binary, the 0s and 1s directly correspond with pixels being black and white. For each row:
    - Convert each byte of the string of bytes into binary using the `format` function.
    - Join these strings of "0"s and "1"s together to form a single string for the whole row.
    - You'll find that each row may have a lot of 0s on the end. Those 0s are just the padding required to make each row width a multiple of four bytes. Eliminate them by extracting the first *n* pixels of each row, where *n* is the width of the original image in pixels.

  When you're done, you should have a list of strings containing 0s and 1s (whereas you started with a list of strings containing hexadecimal codes.) If you print each string to the console, you should be able to make out your original image with 0s representing black and 1s representing white. There's just one problem – it's upside down. Reverse the order of the rows in the list, and finish `load_img` by `return`ing the list. Finally we can work on turning this image into a stereogram!
- You may have noticed that the example stereogram is composed of random ASCII symbols. Write a function `randstr(p)` that returns a string of length `p` of *unique*, random ASCII printable characters.
- We're almost there. To convert each row of the bitmap image into a row of the stereogram:
    - Generate a string of `p` random characters.
    - Append *to this random string* the following (based on pixels in the row of the bitmap):
        - whatever character appeared `p` characters previously, if the pixel is white;

This assignment is not required. You may choose to complete it for extra credit in your assignment grade. It is worth up to 75% the number of points of a regular assignment.

- whatever character appeared `p - 1` characters before, if the pixel is black; or
- a new, random character, if the previous pixel was black but the present pixel is white (i.e. at transitions from black to white).

"Form" the stereogram in a function `img_to_stereogram(img, p)` where img is the output of `load_img` and `p` we'll call the "period" of the histogram. I put "form" in quotation marks because you don't need to generate a new of strings; you only need to process one line at a time. So do this in a generator function, and create an instance of a generator object. Use the string `join` method to create a single string with rows separated by line breaks, and `return` it.

- Use your functions to create a stereogram with period 10 for `PIC.bmp`, and (programmatically) save the stereogram as text to a file.

Self-Assessment

For credit, each criterion below must be satisfied perfectly following the corresponding instructions in the assignment. As usual; do not assign your own partial credit for a given criterion.

Criterion	**Points**	**Your Score**
`load_img` reads a string of hexadecimal values:	10	_____
`read_hex` correctly reads image width, height, etc…:	10	_____
`load_img` raises `ValueError` as required:	5	_____
`load_img` separates data evenly into correct # of rows:	5	_____
`load_img` returns a list of *binary* strings:	20	_____
`rand_str` returns a string of `p` random characters:	5	_____
`img_to_stereogram` generates a valid stereogram:	20	_____
Total:	75	_____