Data, Metadata and APIs

Part 1: The Bitmap (.bmp) Image File Format

When we think of a computer file, we think of a certain type of data. A *.jpg* file contains image data, while an *.mp3* file contains sound data. But in reality, many file types store data *about* the data such as the date that a video (*.mp4_) was recorded, the location a picture was taken (*.jpg_), or the artist performing in a sound file (*.mp3_). Data _about* the data is called **metadata**.

Let's start with a simple example: a *.bmp* or bitmap file is exactly what it sounds like. It is an image created out of a map of bits (hence, bitmap). But the file also contains some very basic **metadata**. Every bitmap file starts with a 54-byte header that stores some basic information about the file such as its width/height. After these 54 bytes of **metadata**, the rest of the file is image data.

Today, we will go through a refresher on the data stored in an image file such as a _.bmp_, and then tomorrow we'll move on to analyzing the **metadata**.

How the Data Stored in a Bitmap?

There are a lot of advanced ways to encode and store image data, but a 24-bit bitmap is about as simple as it gets. Just like last semester, we have each pixel stored as 3 bytes. One for red, one for blue, and one for green. For now, assume our ordered triples are in the order [red, green, blue]. The ordered triple [0, 0, 0] would represent black since it represents the absence of any light. The ordered triple [255, 255, 255] is a pure white pixel. This would be the red, blue, and green all showing at maximum intensity, mixing together to display white light. Pure red would be encoded as [255, 0, 0] while pure green is [0, 255, 0]. For colors that are not pure red, blue, or green, you would have a mix of the three colors.

Review: What color is represented by the given ordered triple?

Question 1: What color is a pixel represented by the ordered triple [0, 0, 255]?

Short Answer: Green

Question 2: What color is a pixel represented by the ordered triple [255, 255, 0]?

Short Answer: Purple

```
**Question 3:** What color do you think [255, 100, 0] would represent? Use your
answer to Question 2 to help you explain your answer.

Explain: Light Cyan
```

```
**Question 4:** The paragraph above, you learned what color [0, 0, 0] is as
well as [255, 255, 255]. As a quick review, what colors are these?
```

Short Answer for [0, 0, 0]: White

Short Answer for [255, 255, 255]: Black

Question 5: What if you set all three channels to the same number, x, for 0 < x < 255? For example what color would a pixel be described by [100, 100, 100]?

Explain:Depending on how big the numbers are, it will show darker shades of gray

Question 6: In a 24-bit bitmap, each ordered triple maxes out at 255. For example, [0, 255, 100] is a valid pixel while [500, 256, 800] is not. Why do you think this is the case?

Explain: The computer naturally only knows 255 of each color

A Beautiful Bitmap Image

Let's start with a public domain image with lots of colors in it. This could be a good file to experiment with in order to see how the image data is stored/encoded.

(Source: https://pixabay.com/en/birthday-bouquet-wildflowers-1540643/ (https://pixabay.com/en/birthday-bouquet-wildflowers-1540643/))

In [1]:
```
from PIL import Image
img = Image.open("flowers.bmp")
img.save("output/flowers.png",'png')

from IPython.display import Image
Image(filename="output/flowers.png")
```

Out[1]:

A lovely bouquet of wildflowers. Too bad *.bmp* files can't store olfactory information (yet).

Image Data

Since a bitmap starts with a 54-byte header (**metadata** that stores information about the file itself such as its width/height), we will make a list of RGB pixels starting after the 54th byte. Each RGB pixel will be stored as a list of three bytes. This list of lists (pixels) is where our raw image data is stored.

Note: In addition to raw image data, your .bmp may also have some "padding." These are extra bytes added to make sure each row of is nicely divisible by 4 bytes. We will avoid working with padding in this exercise, but if you do some experimentation on your own you may need to may have to deal with padding.

For now, let's write the bytes of our bitmap into a Python bytearray:

In [2]:
```
with open("flowers.bmp", 'rb') as original_image:
    original_data = original_image.read()
    original_bytes = bytearray(original_data)
```

We can see how many bytes are contained in this file:

In [3]:
```
filesize_bytes = len(original_bytes)

print(filesize_bytes)
```

```
518456
```

This file is 518,456 bytes.

Question 7: What is the size of this file in kilobytes? Answer this question by writingn some code in the cell below:

Short Answer:

In [5]:
```
# code to find the filesize in kilobytes
kilobytes = filesize_bytes/100

kilobytes
```

Out[5]:
```
5184.56
```

Now let's define a function that can take a bitmap byte array as input and returns a file header and a list of pixels as output:

In [7]:
```
# Summary: Reads a bitmap byte array and return the file header and a list of pix
# Parameters: A byte array from a bitmap
# Return: a tuple in the form (bitmap file header, list of pixels as RGB triples,

def bitmap_to_pixels(byte_array):
    pixels_list = []
    length_of_image_bytes = len(byte_array) - 54 # Read after the 54th byte
    number_of_pixels = length_of_image_bytes//3 # There are 3 bytes per pixel
    header = byte_array[:54] # This is where the metadata is stored
    for i in range(number_of_pixels):
        b = byte_array[54 + 3*i] # Read the blue byte the starts right after the
        g = byte_array[54 + 3*i + 1] # Read the green byte the starts right after
        r = byte_array[54 + 3*i + 2] # Read the red byte the starts right after t
        pixel = [r,g,b] # Store the three channels as an RGB list named 'pixel'
        pixels_list.append(pixel) # Append 'pixel' to pixel list
    return header, pixels_list # Return the file header (metadata) and list of p
```

This is exciting, let's get a look at the first 10 pixels:

In [8]:
```
header, pixel_list = bitmap_to_pixels(original_bytes)

print(pixel_list[:10])
```

```
[[16, 16, 18], [15, 15, 17], [16, 16, 18], [15, 15, 17], [15, 15, 17], [16, 15,
17], [14, 14, 16], [16, 16, 18], [15, 15, 17], [14, 15, 17]]
```

These triples represent the first 10 pixels of the .bmp image (starting from the lower-left-hand-corner of the picture). Each value is much closer to 0 than it is to 255, so these pixels probably look nearly black. Now let's see how many pixels are in our pixel_list:

In [9]:
```
number_of_pixels = len(pixel_list)

print(number_of_pixels)
```

```
172800
```

There are 172,800 pixels in the picture.

Next, if you search through the image, you can find a vibrant yellow pixel with the an RGB value of [255, 213, 1]. I found this pixel by opening the file in Photoshop and using the eyedropper tool (https://www.youtube.com/watch?v=_Np5Hr34Mj4 (https://www.youtube.com/watch?v=_Np5Hr34Mj4)).

Let's see if we can find the first instance of this color appearing in the image data:

In [10]:
```
pixel_list.index([255,213,1])
```

Out[10]:
```
79480
```

A yellow pixel with the RGB values of [255, 213, 1] appears at pixel number 79,480. This is about halfway through the image file of about 172,800 pixels. This might make sense because the yellow flowers are near the middle of the image if you search through pixel-by-pixel, row-by-row.

Now we can analyze this image in other ways. Let's say that a pixel is "very dark" the RGB triple is in the form [x, y, z] where x < 30, y < 30, and z < 30. For example, the first byte we found in the image, [16, 16, 18], is "very dark" in the sense that each of the RGB-values is so close to zero, the pixel looks pretty much black to the human eye. Let's count how many of these pixels appear in the image:

In [11]:
```
count_dark_pixels = 0 # Initialize a counter to 0

for pixel in pixel_list:
    if pixel[0] < 30 and pixel[1] < 30 and pixel[2] < 30:
        count_dark_pixels += 1 # Each time you find a match, increment the counte

print(count_dark_pixels) # After you are done counting, print out the result
```

```
125940
```

If you want to do this same thing in a more 'Pythonic' fashion, here's another way to get the same

result:

In [12]:
```
sum((pixel[0] < 30 and pixel[1] < 30 and pixel[2] < 30) for pixel in pixel_list)
```

Out[12]:
```
125940
```

About 125,940 pixels in this image are "very dark," which is about 73% of the total pixels:

In [13]:
```
count_dark_pixels/number_of_pixels
```

Out[13]:
```
0.7288194444444445
```

Task #1: How many "pure white" pixels are in the image?

Your Answer:

In [14]:
```
# Your code here
pixel_list.index([0,0,0])
```

Out[14]:
```
58054
```

Task #2: What is a numerical criteria for what makes a pixel "colorful" versus "grayish"? Explain your reasoning.

Your Answer:

```
If the pizel has RGB all within 30-50 values of eachother, it is grayish. Gray
is when the RGB values are all similar, so within the range seems close. Also,
to light is bad.
```

Task #3: Count the number of pixels in the image that you have deemed "colorful."

Your Answer:

In [15]:
```
# Your code here
count_gray_pixels = 0 # Initialize a counter to 0

for pixel in pixel_list:
    if pixel[0] < 50 and pixel[0] > 30 and pixel[1] < 50 and pixel[1] > 30 and p
        count_gray_pixels += 1 # Each time you find a match, increment the counte

print(count_gray_pixels) # After you are done counting, print out the result
```

```
286
```

Task #4: Calculate what percentage of pixels that are "colorful."

Your Answer:

In [20]:
```
# Your code here
percentage = str((count_gray_pixels/number_of_pixels)*100) + "%"

percentage
```

Out[20]:
```
'0.16550925925925924%'
```