HW 3 (programming part)
I'll post the written questions separately.

Write a program that parses the EXIF tags from a JPEG file stated on the command line.

I've broken up assignment explanation below for ease of grading, but it's also the best way to approach the assignment. Your program should:

1. [5pts] Open a file and verify that it is a JPEG by confirming that the first 2 bytes are 0xFFD8. If it is not, print a message saying so and exit.
2. [10pts] Find each marker, print its location from the start of the file, the marker number, and its length. The first marker is 2 bytes into the file (i.e., it's right after the 0xFFD8) and is very likely 0xFFE0. Each marker begins like so: 0xFFmmdddd, where mm is the marker number (e.g., E0), and dddd is the length of the marker *including the 2-bytes of 0xFFmm.* The 0xFFDA marker is special: it precedes the actual image data. Once you find it, you can stop looking for other markers and exit (it should be the very last marker). If you only got this far in the assignment, your program would output the following:

```
[0x0002] Marker 0xFFE0 size=0x0010
[0x0014] Marker 0xFFE1 size=0x04DC
[0x04F2] Marker 0xFFE1 size=0x0A0D
[0x0F01] Marker 0xFFED size=0x0038
[0x0F3B] Marker 0xFFC0 size=0x0011
[0x0F4E] Marker 0xFFC4 size=0x001F
[0x0F6F] Marker 0xFFC4 size=0x00B5
[0x1026] Marker 0xFFC4 size=0x001F
[0x1047] Marker 0xFFC4 size=0x00B5
[0x10FE] Marker 0xFFDB size=0x0043
[0x1143] Marker 0xFFDB size=0x0043
[0x1188] Marker 0xFFDD size=0x0004
[0x118E] Marker 0xFFDA size=0x000C
```

3. [10pts] Now add to your program by locating the marker that contains the Exif header as described on the next page, and confirm that it's a big endian Exif block. If it's little endian, please have your program exit(). If it's big endian, we'll do some parsing in parts below. Note that in my example of the full program, I find the other markers and their lengths etc as described in part 2 above (part 3 is not a replacement of part 2!).
4. [10pts] With in the marker that you've confirmed has the Exif data, find the start of the IFD and then print the number of entries.
5. [20pts] For each entry, printing their names in hex and the string equivalent. And then print the value iff the format is one of 1,2,3,4,5, or 7. You can simply ignore the other types. In the table that follows, I state whether you need to print only the first component or all components for a given format (in all cases, the value can be printed in 1 or 2 lines of code, which I provide the basics of). You should be careful to understand how to compute the length of an entry's value, and know that when the length is less than or equal to 4 bytes, the data field contains the value, and it is an offset to the actual value otherwise.

The program should accept one argument on the command line: the jpeg file to be examined. An example would be:

```
% python hw3.py photo.jpg
```

Note that you do not need to (and cannot, in fact) parse the "Maker Note" tag (0x927c)! It will appear as a string of characters, mostly blank/unprintable.

This assignment is a lot tougher than the previous two, please get started right now so that you have time to contact us when you get stuck.

As always, you should follow Python style guidelines, use well reasoned programming logic and structure, etc., as we've discussed in class.

Value	Format	Bytes per component	`Python` `(assuming data[0:] holds the value we'll acquire)`	`First or all components?`
1	Unsigned byte	1	`unpack(">B",data[0:1])` *(one component)*	`First`
2	ASCII string	1	`bytes.decode(data[0:length])` *(one component)*	`First`
3	Unsigned short	2	`unpack(">%dh" % components, data[0:length])` *(all component)*	`All`
4	Unsigned long	4	`unpack(">L",data[0:4])` *(one component)*	`One`
5	Unsigned rational	8	`(numerator, denominator)=` `unpack(">LL",data[0:8])` `"%s/%s" % (numerator,denominator)` *(one component)*	`One`
6	Signed byte	1	`ignore`	-
7	Undefined (raw)	1	`unpack(">%dB" % length, data[0:length])` `"".join("%c" % x for x in value)` *(all component)*	`All`
8	Signed short	2	`ignore`	-
9	Signed long	4	`ignore`	-
10	Signed rational	8	`ignore`	-
11	Single float	4	`ignore`	-
12	Double float	8	`ignore`	-

```
00000000  ff d8 ff e0 00 10 4a 46  49 46 00 01 01 00 00 48  |......JFIF.....H|
[2] ffd8 => jpeg header
[2] ffe0 => marker number (length of this field is 2)
[2] 0010 => Size, including these two bytes.
          Since 0x0010=16, read in 16-2=14 more bytes to grab the entire app
          Location of next marker is:
           = location_of_marker + length_of_marker_number + size
           = 0x2 + 0x2 + 0x10 = 0x14
00000010  00 48 00 00 ff e1 04 dc  45 78 69 66 00 00 4d 4d  |.H......Exif..MM|
00000020  00 2a 00 00 00 08 00 09  01 0f 00 02 00 00 00 06  |.*..............|
[4] 00 48 00 00 => is part of the previous 0xffe0 app.
[2] ffe1=> marker number
[2] 04dc=> size, including these two bytes.
          0x04dc=1244, and so next 1242 bytes must be read in
          Location of next marker is 0x14+0x02+0x04dc=0x04f2
[10] 4578696600 4d4d002a== b'Exif\x00\x00MM\x00\x2a'
     The first 6 bytes state that this is in fact the Exif we are looking for.
     If you don't see these bytes, then this marker app is not an Exif, and so move on.
     0x4d4d= tells you that the exif entries are big endian ("M" stands for "Motorola")
     0x002a== constant of 42, assuming big endian.
     (It would have been 0xII2a00 if little endian)
        your program can quit if it finds little endian exif entries.
[4] 00000008 => ifd_offset, the offset in bytes to Image File Directory (IFD) from 0x4d (the first one
in 0x4d4d). How many bytes do we need to skip ahead in order to find the start of the IFD? Well, first
let's ask, How many bytes since and include 0x4d have we read in to parse the offset? 2 bytes for
0x4d4d, 2 more for 0x002a, 4 more for the offset value. That's 2+2+4 =8. So we don't have to skip ahead
at all: The IFD starts next.
[2] 0009 => entries, there are 0x9 entries in this IFD. The first starts immediately.
All entries are each 12 bytes long, starting after this last byte.
Therefore, the entry i is at offset+i*12+2
Each entry is tag,format,number_of_components,data like so: 0xttttffffnnnnnnnndddddddd
[2] 010f=> tag, in this case "Make"
[2] 0002=> format, in this case 2
[4] 0000006=> components, in this case 6
00000030  00 00 00 7a 01 10 00 02  00 00 00 09 00 00 00 80  |...z............|
00000040  01 1a 00 05 00 00 00 01  00 00 00 8a 01 1b 00 05  |................|
00000050  00 00 00 01 00 00 00 92  01 28 00 03 00 00 00 01  |.........(......|
00000060  00 02 00 00 01 31 00 02  00 00 00 06 00 00 00 9a  |.....1..........|
00000070  01 32 00 02 00 00 00 14  00 00 00 a0 87 69 00 04  |.2...........i..|
00000080  00 00 00 01 00 00 00 b4  88 25 00 04 00 00 00 01  |.........%......|
00000090  00 00 03 d2 00 00 00 00  41 70 70 6c 65 00 69 50  |........Apple.iP|
000000a0  68 6f 6e 65 20 35 00 00  00 00 00 48 00 00 00 01  |hone 5.....H....|
000000b0  00 00 00 48 00 00 00 01  38 2e 31 2e 32 00 32 30  |...H....8.1.2.20|
000000c0  31 35 3a 30 31 3a 31 30  20 31 36 3a 31 38 3a 34  |15:01:10 16:18:4|
I've shown a lot of output suddenly cause we need it. Continuing with the first tag:
[4] 0000007a=> data, which is 0x7a or 122. Let's come back to that in a second.
That was 12 bytes. And so now tag 2 starts:
[2] 0110=> tag, in this case "Model"
[2] 0002=> format, in this case 2
[4] 00000009=> components, in this case 9
[4] 00000080=> data, in this case 80.
```

Here's how you parse each tag:

- The **tag** converts to a string using a dictionary, which I'll provide for you.
- The **format** tells you how many bytes_per_component for this tag. This is defined as bytes_per_component = (0,1,1,2,4,8,1,1,2,4,8,4,8)
- The length (in bytes) of this tag's data is equal to length = bytes_per_component[format]*components
- If the length <=4, then the **data** field is the value. Otherwise, it's the **offset** to the data. And here, length = 1*6 = 6, which is > 4.
- Offset from what you ask? From the 0x4d. So to make programming easier, it's best to send the IFD to a function as an array starting from the 0x4d.
- Let's check that: location of 0x4d: 0x1E
- Add offset: 0x1E+0x7A = 0x98
- We find length of 6 bytes: : **41 70 70 6c 65 00** or "**Apple**"; all strings end with 0x0 (the null bit)

Let's do the second tag:

- length is bytes_per_component[format]*components = 1*9 = 9
- 9 is greater than 4, and so the data field is an offset and not a value itself.
- Add offset to location of 0x4d: 0x1E + 0x80 = 0x9E
- Above we see that's 9 bytes: **69 50 68 6f 6e 65 20 35 00** or "**iPhone 5**".

```
$ ./hw3.py FullSizeRender.jpg
[0x0002] Marker 0xFFE0 size=0x0010
[0x0014] Marker 0xFFE1 size=0x04DC
Number of IFD Entries: 9
 8825 GPSInfoIFDPointer [978]
  128 ResolutionUnit   2
 8769 ExifIFDPointer   [180]
  10f Make             Apple
  110 Model            iPhone 5
  131 Software         8.1.2
  132 DateTime         2015:01:10 16:18:44
  11a XResolution      ['72/1']
  11b YResolution      ['72/1']

[0x04F2] Marker 0xFFE1 size=0x0A0D
[0x0F01] Marker 0xFFED size=0x0038
[0x0F3B] Marker 0xFFC0 size=0x0011
[0x0F4E] Marker 0xFFC4 size=0x001F
[0x0F6F] Marker 0xFFC4 size=0x00B5
[0x1026] Marker 0xFFC4 size=0x001F
[0x1047] Marker 0xFFC4 size=0x00B5
[0x10FE] Marker 0xFFDB size=0x0043
[0x1143] Marker 0xFFDB size=0x0043
[0x1188] Marker 0xFFDD size=0x0004
[0x118E] Marker 0xFFDA size=0x000C
```

Here's a verbose version of my program for the same file that will help you write yours.

```
[0x0002] Marker 0xFFE0 size=0x0010 (next marker at 0x0002+0x0010-0x2=0x0014)
[0x0014] Marker 0xFFE1 size=0x04DC (next marker at 0x0014+0x04dc-0x2=0x04f2)
Offset to IFD: 8
Byte offset to first entry from 0x42: 8
Number of IFD Entries: 9
> 01 0F 00 02 00 00 00 06 00 00 00 7A |...........z|
entry 0: tag(0x010F):Make             [format( 2) component( 6) length( 6) offset(0x007a)]
> 01 10 00 02 00 00 00 09 00 00 00 80 |.......       ....|
entry 1: tag(0x0110):Model            [format( 2) component( 9) length( 9) offset(0x0080)]
> 01 1A 00 05 00 00 00 01 00 00 00 8A |............|
entry 2: tag(0x011A):XResolution      [format( 5) component( 1) length( 8) offset(0x008a)]
> 01 1B 00 05 00 00 00 01 00 00 00 92 |............|
entry 3: tag(0x011B):YResolution      [format( 5) component( 1) length( 8) offset(0x0092)]
> 01 28 00 03 00 00 00 01 00 02       |.(........|
entry 4: tag(0x0128):ResolutionUnit   [format( 3) component( 1) length( 2) value(0x0042)]
> 01 31 00 02 00 00 00 06 00 00 00 9A |.1..........|
entry 5: tag(0x0131):Software         [format( 2) component( 6) length( 6) offset(0x009a)]
> 01 32 00 02 00 00 00 14 00 00 00 A0 |.2..........|
entry 6: tag(0x0132):DateTime         [format( 2) component(20) length(20) offset(0x00a0)]
> 87 69 00 04 00 00 00 01 00 00 00 B4 |.i..........|
entry 7: tag(0x8769):ExifIFDPointer   [format( 4) component( 1) length( 4) value(0x0066)]
> 88 25 00 04 00 00 00 01 00 00 03 D2 |.%..........|
entry 8: tag(0x8825):GPSInfoIFDPointer [format( 4) component( 1) length( 4) value(0x0072)]
 8825 GPSInfoIFDPointer [978]
  128 ResolutionUnit   2
 8769 ExifIFDPointer   [180]
  10f Make             Apple
  110 Model            iPhone 5
  131 Software         8.1.2
  132 DateTime         2015:01:10 16:18:44
  11a XResolution      ['72/1']
  11b YResolution      ['72/1']

[0x04F2] Marker 0xFFE1 size=0x0A0D (next marker at 0x04F2+0x0a0d-0x2=0x0f01)
[0x0F01] Marker 0xFFED size=0x0038 (next marker at 0x0F01+0x0038-0x2=0x0f3b)
[0x0F3B] Marker 0xFFC0 size=0x0011 (next marker at 0x0F3B+0x0011-0x2=0x0f4e)
[0x0F4E] Marker 0xFFC4 size=0x001F (next marker at 0x0F4E+0x001f-0x2=0x0f6f)
[0x0F6F] Marker 0xFFC4 size=0x00B5 (next marker at 0x0F6F+0x00b5-0x2=0x1026)
[0x1026] Marker 0xFFC4 size=0x001F (next marker at 0x1026+0x001f-0x2=0x1047)
[0x1047] Marker 0xFFC4 size=0x00B5 (next marker at 0x1047+0x00b5-0x2=0x10fe)
[0x10FE] Marker 0xFFDB size=0x0043 (next marker at 0x10FE+0x0043-0x2=0x1143)
[0x1143] Marker 0xFFDB size=0x0043 (next marker at 0x1143+0x0043-0x2=0x1188)
[0x1188] Marker 0xFFDD size=0x0004 (next marker at 0x1188+0x0004-0x2=0x118e)
[0x118E] Marker 0xFFDA size=0x000C (next marker at 0x118E+0x000c-0x2=0x119c)
```

```
$ ./hw3.py gore-superman.jpg
[0x0002] Marker 0xFFE0 size=0x0010
[0x0014] Marker 0xFFE1 size=0x1404
Number of IFD Entries: 7
  128 ResolutionUnit   2
 8769 ExifIFDPointer   [164]
  132 DateTime         2006:06:06 21:02:57
  131 Software         Adobe Photoshop Elements 2.0
  112 Orientation      1
  11a XResolution      ['72/1']
  11b YResolution      ['72/1']

[0x141A] Marker 0xFFED size=0x18EA
[0x2D06] Marker 0xFFE1 size=0x13CB
[0x40D3] Marker 0xFFEE size=0x000E
[0x40E3] Marker 0xFFDB size=0x0084
[0x4169] Marker 0xFFC0 size=0x0011
[0x417C] Marker 0xFFDD size=0x0004
[0x4182] Marker 0xFFC4 size=0x013F
[0x42C3] Marker 0xFFDA size=0x000C

Verbose output:

[0x0002] Marker 0xFFE0 size=0x0010 (next marker at 0x0002+0x0010-0x2=0x0014)
[0x0014] Marker 0xFFE1 size=0x1404 (next marker at 0x0014+0x1404-0x2=0x141a)
Offset to IFD: 8
Byte offset to first entry from 0x42: 8
Number of IFD Entries: 7
> 01 12 00 03 00 00 00 01 00 01       |..........|
entry 0: tag(0x0112):Orientation      [format( 3) component( 1) length( 2) value(0x0012)]
> 01 1A 00 05 00 00 00 01 00 00 00 62 |...........b|
entry 1: tag(0x011A):XResolution      [format( 5) component( 1) length( 8) offset(0x0062)]
> 01 1B 00 05 00 00 00 01 00 00 00 6A |...........j|
entry 2: tag(0x011B):YResolution      [format( 5) component( 1) length( 8) offset(0x006a)]
> 01 28 00 03 00 00 00 01 00 02       |.(........|
entry 3: tag(0x0128):ResolutionUnit   [format( 3) component( 1) length( 2) value(0x0036)]
> 01 31 00 02 00 00 00 1D 00 00 00 72 |.1.........r|
entry 4: tag(0x0131):Software         [format( 2) component(29) length(29) offset(0x0072)]
> 01 32 00 02 00 00 00 14 00 00 00 8F |.2..........|
entry 5: tag(0x0132):DateTime         [format( 2) component(20) length(20) offset(0x008f)]
> 87 69 00 04 00 00 00 01 00 00 00 A4 |.i..........|
entry 6: tag(0x8769):ExifIFDPointer   [format( 4) component( 1) length( 4) value(0x005a)]
  128 ResolutionUnit   2
 8769 ExifIFDPointer   [164]
  132 DateTime         2006:06:06 21:02:57
  131 Software         Adobe Photoshop Elements 2.0
  112 Orientation      1
  11a XResolution      ['72/1']
  11b YResolution      ['72/1']

[0x141A] Marker 0xFFED size=0x18EA (next marker at 0x141A+0x18ea-0x2=0x2d06)
[0x2D06] Marker 0xFFE1 size=0x13CB (next marker at 0x2D06+0x13cb-0x2=0x40d3)
[0x40D3] Marker 0xFFEE size=0x000E (next marker at 0x40D3+0x000e-0x2=0x40e3)
[0x40E3] Marker 0xFFDB size=0x0084 (next marker at 0x40E3+0x0084-0x2=0x4169)
[0x4169] Marker 0xFFC0 size=0x0011 (next marker at 0x4169+0x0011-0x2=0x417c)
[0x417C] Marker 0xFFDD size=0x0004 (next marker at 0x417C+0x0004-0x2=0x4182)
[0x4182] Marker 0xFFC4 size=0x013F (next marker at 0x4182+0x013f-0x2=0x42c3)
[0x42C3] Marker 0xFFDA size=0x000C (next marker at 0x42C3+0x000c-0x2=0x42d1)
```