# Data, Metadata and APIs

## Part 4: Exploring JPEG Metadata with the Google Maps API

What kinds of metadata can be stored in the average *.jpg* file? It turns out there is quite a bit. As is the case with many technologies, metadata can provide a great deal of convenience for users but can also pose a serious privacy risk. You should think about this balance between convenience and privacy as you work through this notebook.

### How is Metadata stored in JPEG files?

Many media files, including JPEG images, use the Exif standard for storing metadata (https://en.wikipedia.org/wiki/Exif (https://en.wikipedia.org/wiki/Exif)). In this notebook, we will use the *exifread* Python module to see what metadata we can find in a few preselected *.jpg* files (and if you choose, a few of your own *.jpg* images from your phone or iPad).

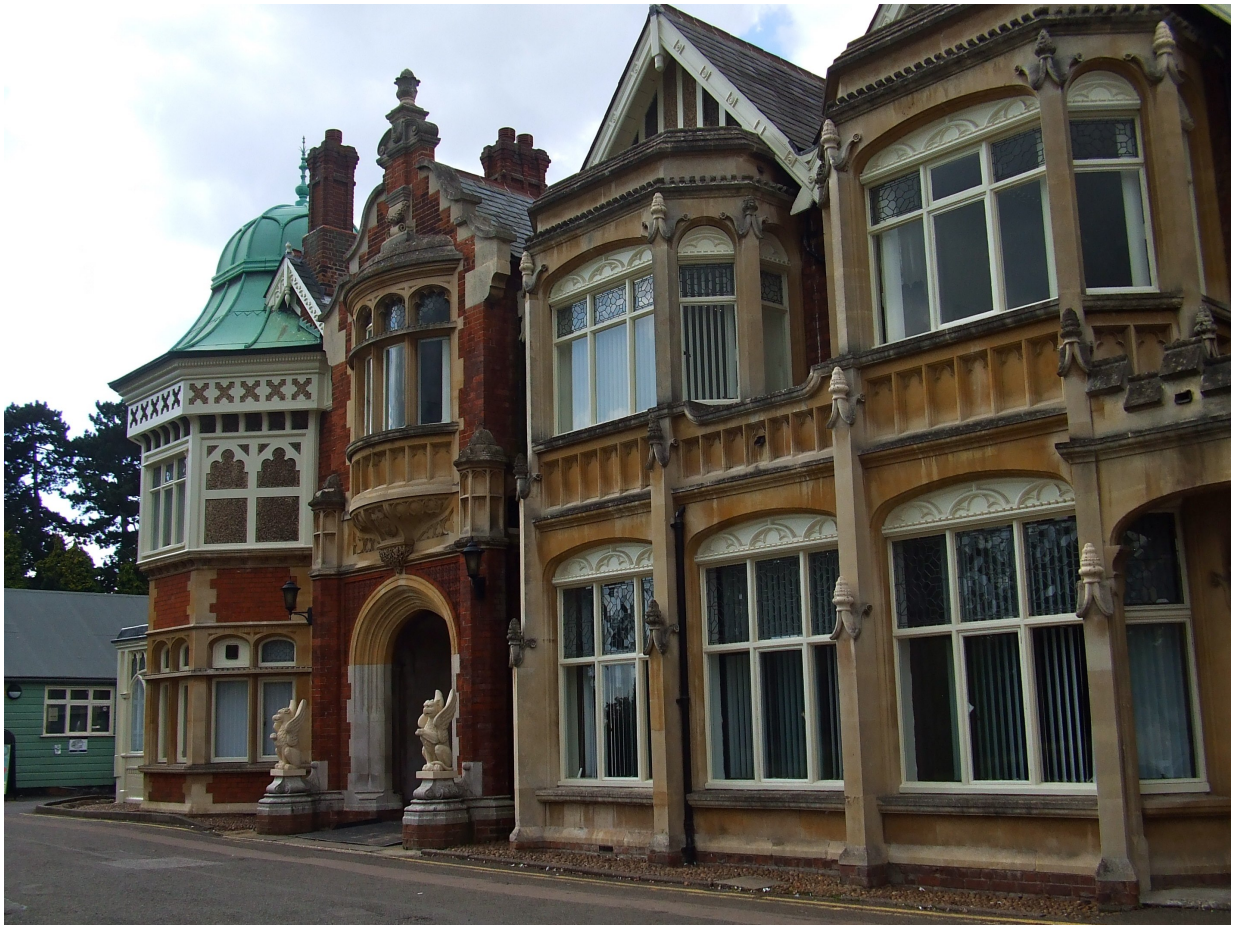
### Extract Metadata from a Photograph of "Mystery Location #1"

Let's start with a file named *mystery1.jpg* and see what data we can extract from it using the *exifread* module.

Here's the photo (from a mystery location):

In [1]:
```
from IPython.display import Image
Image(filename="mystery1.jpg")
```

Out[1]:



Yesterday finding metadata was a labor-intensive search. The *exifread* module makes this metadata extraction much more palatable.

First, read the metadata and store it as the variable *tags1* :

In [2]:
```
# https://pypi.python.org/pypi/ExifRead
import exifread

# Open image file for reading (binary mode)
mystery1 = open('mystery1.jpg', 'rb')

# Return Exif tags
tags1 = exifread.process_file(mystery1)
```

The metadata for the image is now stored in *tags1*.

Let's see what data structure is stored in this variable:

In [3]:
```
type(tags1)
```

Out[3]:
```
dict
```

It's a dictionary (type dict), but it's difficult to read in this form:

In [4]:
```
# Print out the metadata
print(tags1)
```

```
\x00\x1e\xaf-gr\xecF#\x9eP\xc4)\x8a1\xce:\x12=\x0f\xaf\xe3P-\xbcew\xbc\xe5\x0
7?+\x0ekNKgPM\xcd\xca(\xeb\xb4/=;T+\xf6H\x1c\xf9\xa1\xcfp\x06\x0e\x07\xa0\xc9
\xf6\xa1\x08\xad\x0c\x10\x16M\xb2\x97\x19\x19@z\x8a\xce\xb9\xcd\xb5\xe34,P\xa
b\x1d\xac\x0fj\xd3\x90A,\xaa\xd0\xa3(\xcf\xcd\x9e3\xf9~5CQ\x8bd\xad\x8eFs\xf4
\xa1=G\xb9\xed\xb4QEY"0\xc8\xaa\xec\xa4v\xab\x07\xa7\x15R\xf6\xe5-`2\xc8x\x1c
\x01\xdd\x8f\xa0\xf7\xaa\x8b&D\x17\xd2\x18\xec\xa7ppDlG8\xed\xeb\\V\x85\x7fys
\xe2t\x91\xa5S$\xc0\xac\x84\xa7U\x0b\x9cq\xd3\xee\x8a\xd2\xba\xd6\xeen\x16Uo

(D\xc3\x05T\x13\xc6q\xd7\xa9\xce@\xed\xf4\xae{K\xb86z\x92\xdc.w\xaa\xb8\\c\xe
f\x95!z\xfb\x91M\xb1A\xde\xe5\x8dw\x9dj\xe84\x99\xcc\x87q\x0f\xbb\xb7\x03\xdb
\x1c\x0cv\xe9\xcdn\xdbk\xf0E\xa4\x1bP$\x8a\xea\x0bm\xa8J\x86V`0\xb8\xc7\xaf\x
1dF+\x92s\xba\x1e6\xed$\xed\xc9\xc1\xc6G\xff\x00^\xa6\x845\xc6\xd1!\xc3\xa7\x
dd\xc7\x19\xe3\xa7\x15\x9f6\xa6\x8dh;O\xd4o\xa1\xd4\xd2E\x9d\x12Y%%\xfc\xd1\x
84\xcb\x91\xbb#\xb08\x19\xc7\xa5t>#\xf1\x14\r\xe5\xdb\xd9K\xbc\x02\xc5\xca\xe
7\xa8\xe0`\xf7\xc7\'\xa63\x8ek\x9a\xb9u\x91\nJdV\x0cK\x1cr@\xe3\x1d\xbd\x7f\x
9f\xb51!eB\xf8X\x14q\x92y\xe9\xfeq\x8a\x1b\xb0Z\xe3\x19\x18\xe0\xc8\xc2%=7u?
\x87\xe1G\x97\x1b\xb1\xdfw\x929\xc9\x19\xfdOZ\xb2\x91\xc6\x96\xf22\xee\x92G\x
e5\xc9\x8c\x9c\x8e\xbd\xfd\xc0\xa7\x8bK\x93\x07\x98\xb0G\xb7\xae\xdd\xa78\xff
\x00\x1a\x9b\x8c\xa6\x97\x8dj\xcf\x1b?\x99m \xc3\x01\xce\x0f\xae=j\xed\xcc"\x
15u\x87q\\\x0c\xb3\x13\xf3\x12\x18g\xd3\xa0\xed\x8cVm\xca\xecua\x10\xec\xd8n
```

Yikes. That's a lot of metadata. If you want to sift through it, you might want to convert to a Pandas dataframe:

In [5]:
```
import pandas as pd

tags1_DF = pd.DataFrame(tags1, index=[0])

tags1_transpose = pd.DataFrame.transpose(tags1_DF)

tags1_transpose
```

Out[5]:

| | 0 |
|---|---|
| Image ImageDescription | https://www.flickr.com/photos/chatiryworld/ |
| Image Make | FUJIFILM |
| Image Model | FinePix A610 |
| Image Orientation | Horizontal (normal) |
| Image XResolution | 72 |
| Image YResolution | 72 |
| Image ResolutionUnit | Pixels/Inch |
| Image Software | QuickTime 7.6.6 |
| Image DateTime | 2010:09:21 18:09:26 |
| Image Artist | Katherine (chatirygirl on Flickr) |
| Image HostComputer | Mac OS X 10.6.4 |
| Image Copyright | Attribution-NoDerivs 2.0 Generic (CC BY-ND 2.0) |
| Image ExifOffset | 402 |
| GPS GPSVersionID | [2, 2, 0, 0] |
| GPS GPSLatitudeRef | N |
| GPS GPSLatitude | [51, 59, 5157/100] |
| GPS GPSLongitudeRef | W |
| GPS GPSLongitude | [0, 44, 2647/100] |
| GPS GPSAltitudeRef | 0 |
| GPS GPSAltitude | 0 |
| GPS GPSMapDatum | WGS-84 |
| Image GPSInfo | 908 |
| Thumbnail Compression | JPEG (old-style) |
| Thumbnail JPEGInterchangeFormat | 1114 |
| Thumbnail JPEGInterchangeFormatLength | 4948 |
| EXIF ExposureTime | 1/170 |
| EXIF FNumber | 8 |
| EXIF ExposureProgram | Landscape Mode |
| EXIF ISOSpeedRatings | 100 |
| EXIF ExifVersion | 0220 |

| | 0 |
|---|---|
| EXIF DateTimeOriginal | 2010:09:16 23:03:36 |
| EXIF DateTimeDigitized | 2010:09:16 23:03:36 |
| EXIF CompressedBitsPerPixel | 2 |
| EXIF ShutterSpeedValue | 743/100 |
| EXIF ApertureValue | 6 |
| EXIF BrightnessValue | 837/100 |
| EXIF ExposureBiasValue | 0 |
| EXIF MaxApertureValue | 16/5 |
| EXIF MeteringMode | Pattern |
| EXIF LightSource | Unknown |
| EXIF Flash | Flash did not fire, compulsory flash mode |
| EXIF FocalLength | 33/5 |
| EXIF FlashPixVersion | 0100 |
| EXIF ColorSpace | sRGB |
| EXIF ExifImageWidth | 2848 |
| EXIF ExifImageLength | 2136 |
| EXIF FocalPlaneXResolution | 4853 |
| EXIF FocalPlaneYResolution | 4853 |
| EXIF FocalPlaneResolutionUnit | 3 |
| EXIF SensingMethod | One-chip color area |
| EXIF CustomRendered | Normal |
| EXIF ExposureMode | Auto Exposure |
| EXIF WhiteBalance | Auto |
| EXIF SceneCaptureType | Landscape |
| EXIF Sharpness | Normal |
| EXIF SubjectDistanceRange | 0 |
| JPEGThumbnail | b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01... |

**Question 1:** Comment on what you see in the metadata. Who took the picture? When? With what kind of camera? Anything else interesting?

Your Answer:

```
Who: Katherine

When: 11:03PM of September 16th of 2010

Camera type:FUJIFILM Fine Pixe A610

Intresting fact:She had a brightness value over the limit designed in th
e camera
```

## GPS Metadata and the Google Maps API

Many cameras and smartphones will include GPS metadata in your photos. Storing this kind of metadata is called "geotagging," and it can be a great way to organize your vacation photos. For example, the photosharing website Flickr has a world map of 1.5 million geotagged photos: https://www.flickr.com/map (https://www.flickr.com/map)

As you can imagine, the unintentional sharing of GPS metadata can also introduce privacy concerns: http://www.atlasobscura.com/articles/this-map-plots-the-geocoordinates-you-may-have-accidentally-left-behind-in-digital-photos (http://www.atlasobscura.com/articles/this-map-plots-the-geocoordinates-you-may-have-accidentally-left-behind-in-digital-photos). The pictures described in this article can be found here: http://www.psych.mcgill.ca/labs/ottolab/accidental_geography/# (http://www.psych.mcgill.ca/labs/ottolab/accidental_geography/)

Let's see if we can figure out where *mystery1.jpg* was taken by extracting GPS metadata:

In [6]:
```
# https://pypi.python.org/pypi/ExifRead
import exifread

# Open image file for reading (binary mode)
mystery1 = open('mystery1.jpg', 'rb')

# Return Exif tags
tags1 = exifread.process_file(mystery1)

print(tags1['GPS GPSLongitudeRef'].values)
print(tags1['GPS GPSLongitude'].values)
print(tags1['GPS GPSLatitudeRef'].values)
print(tags1['GPS GPSLatitude'].values)
```

```
W
[0, 44, 2647/100]
N
[51, 59, 5157/100]
```

Bingo! We just extracted GPS coordinates in DMS (degrees-minutes-seconds). In order to plot these coordinates, we need to convert them to DD (decimal degree) format.

It's abstraction to the rescue again. We can use **abstraction to reduce the complexity of our program** by encapsulating some tedious steps in two functions, *list_to_DMS(gps_list)* and *tags_to_DMS(tags)*:

In [7]:
```
# This function converts degrees, minutes, and seconds into decimal coordinates
# Parameters: Single GPS coordinate in the form [degrees, minutes, seconds]
# Return: Decimal format of the GPS coordinate
def list_to_DD(gps_list):
    d = float(gps_list[0].num)/float(gps_list[0].den)
    m = float(gps_list[1].num)/float(gps_list[1].den)
    s = float(gps_list[2].num)/float(gps_list[2].den)
    return d + m/60.0 + s/3600.0

# This function takes exifreader data, extracts the latitude and longitude data,
# Parameter: Exif tags
# Return: Tuple conatining two decimals in the form (latitude decimal, longitude
def tags_to_DD(tags):
    longDirection = tags['GPS GPSLongitudeRef'].values
    longList = tags['GPS GPSLongitude'].values
    latDirection = tags['GPS GPSLatitudeRef'].values
    latList = tags['GPS GPSLatitude'].values

    latFloat = list_to_DD(latList)
    longFloat = list_to_DD(longList)

    if latDirection == "S":
        latFloat = -1*latFloat

    if longDirection == "W":
        longFloat = -1*longFloat

    return (latFloat,longFloat)
```

Let's use these functions to convert our mystery location to traditional GPS coordinates:

In [8]:
```
mystery_location1 = tags_to_DD(tags1)

print(mystery_location1)
```

```
(51.997658333333334, -0.740686111111111)
```

## Import the Google Maps API

Our next goal is to put these coordinates onto an interactive Google map. In order to do this, we will need to use the Google Maps API. An API is an "Application Programming Interface," which is a tool that can be used to communicate between two different pieces of software. Often times, large companies (Google, Amazon, Yelp, etc) will provide an API for their products to programmers. The goal is to encourage programmers to innovate using their platform.

In our case, we will use the Google Maps API to get Google Maps data into our project: https://en.wikipedia.org/wiki/Application_programming_interface (https://en.wikipedia.org/wiki/Application_programming_interface)

In [9]:
```
# Import the gmaps python module and load in your API Key:

import gmaps

gmaps.configure(api_key="AIzaSyCLla6Q7krE9xNg6SnNMoGNIzjCLddE9EU") # you need to
```

Now get a map of Mystery Location 1:

In [10]:
```
# This is a list that will hold all of your GPS coordinates
# For now, locations will only have one set of coordinates, but you'll be adding

locations = []
```

In [11]:
```
from ipywidgets.embed import embed_minimal_html # Allows us to create a separte j

locations.append(mystery_location1)

markers = gmaps.marker_layer(locations)

markermap = gmaps.Map()
markermap.add_layer(markers)

embed_minimal_html('MarkerMap1.html', views=[markermap])
print("*** Check your 'Metadata Part 4' folder to find the new HTML file named \
```

```
*** Check your 'Metadata Part 4' folder to find the new HTML file named "Marker
Map1". ***
```

**Question 2:** Where was this photograph of "Mystery Location #1" taken? Zoom the map in/out to get a better idea of the location.

*Note: This location has important significance in the history of computer science. You may want to look it up if you are curious.*

Your Answer: Bletchley Park

## Task 1: Practicing with Abstraction

Write a function, *gps_from_image(file_name)*, that takes a filename as input and returns decimal degree (DD) coordinates.

- Open the file
- Use exifread to process the file
- Return coordinates using tags_to_DD(), feeding it your processed tags

*Note: To test your function, you must run *gps_from_image("mystery1.jpg")* to show that it returns (51.997658333333334, -0.740686111111111).**

In [12]:
```
# Your code here
def gps_from_image(file_name):
    mystery = open(file_name, 'rb')
    tags = exifread.process_file(mystery)
    mystery_location = tags_to_DD(tags)
    return(mystery_location)

gps_from_image("mystery1.jpg")
```

Out[12]:
```
(51.997658333333334, -0.740686111111111)
```

In [13]:
```
mystery = open("mystery2.jpg", 'rb')
tags = exifread.process_file(mystery)
mystery_location = tags_to_DD(tags)
print(mystery_location)
```

```
(37.086241666666666, -76.38088055555555)
```

**Question 3:** Explain how *gps_from_image(file_name)* is an **abstraction that reduces the complexity of your program**.

Your Answer: By putting all the the code required for the program into it, all the user has to do is run the function again and again, and then you can run the code multiple times without rewriting it

**Question 4:** Explain how *gps_from_image(file_name)* is an **algorithm that uses two other algorithms**. Be specific about which other algorithms are used in *gps_from_image(file_name)*.

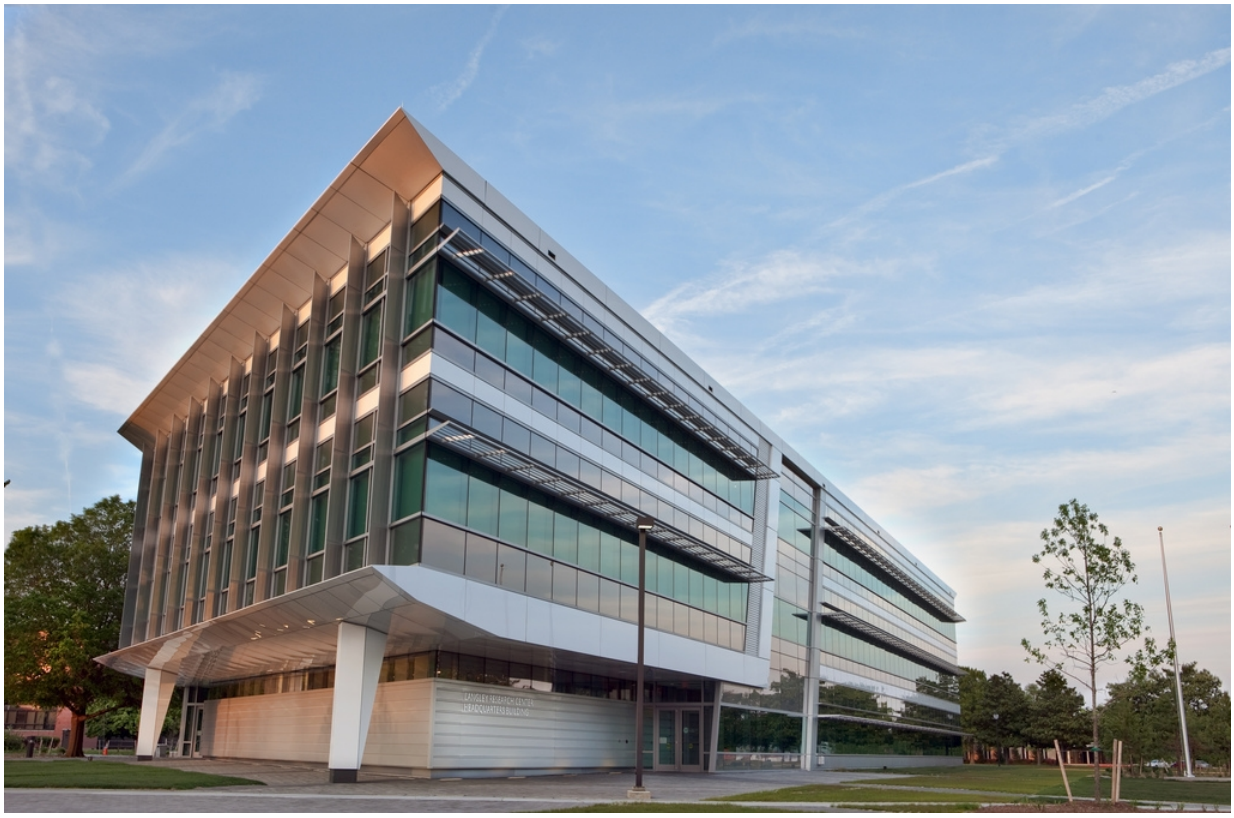
Your Answer: This function is an algorithm, a step by step proccess of how to do something. It does this by first accessing exifread, and then opening the file needed. It then processes the file useing exifread, and gets the metadata. The program then uses an older program to find the specific information needed, and then prints it.n The old program itself is an algorithm itself, as it uses step by step instructions, and open() is a algorith that first finds a file, and then opens it.

## Task 2: Extract GPS Metadata from a Photograph of "Mystery Location #2"

Here's a photo from a second mystery location:

In [14]:
```
from IPython.display import Image
Image(filename="mystery2.jpg")
```

Out[14]:



**Question 5:** What are the decimal degree (DD) coordinates of the location where this photograph was taken?

*Hint: Use the function you defined in Task 1.*

Your Answer:

In [15]:
```
# Your code here
gps_from_image("mystery2.jpg")
```

Out[15]:
```
(37.086241666666666, -76.38088055555555)
```

**Question 6:** Where in the world was this photograph taken?

Note:

- You must add this location to your Google Map. You should write your code so that Mystery Location #1 and Mystery Location #2 are together on the same map.
- Don't forget to add in the necessary code to produce your Google Maps as an HTML file
  - **Make sure to change the output file name to "MarkerMap2.html"!**
- If you do some research, you will find that this location also has important significance in the history of computer science.

Your Answer:

NASA Langley Reasearch Center

In [16]:
```
# Your code here
import gmaps

gmaps.configure(api_key="AIzaSyCLla6Q7krE9xNg6SnNMoGNIzjCLddE9EU") # you need to

locations = []

from ipywidgets.embed import embed_minimal_html # Allows us to create a separte j

locations.append(mystery_location)

markers = gmaps.marker_layer(locations)

markermap = gmaps.Map()
markermap.add_layer(markers)

embed_minimal_html('MarkerMap2.html', views=[markermap])
print("*** Check your 'Metadata Part 4' folder to find the new HTML file named \
```

```
*** Check your 'Metadata Part 4' folder to find the new HTML file named "Marker
Map2". ***
```

## Task #3: Add More Locations to your Map

Add more locations to your map, either with geotagged photos from your phone/iPad or with geotagged photos you found online. Remember that many geotagged photos can be found here: http://www.psych.mcgill.ca/labs/ottolab/accidental_geography/# (http://www.psych.mcgill.ca/labs/ottolab/accidental_geography/)

*Note: Many modern smart phones delete GPS metadata from images that are exported from the phone. This is a security feature meant to protect your privacy. If you want to view photos from your own phone/iPad on the map, you will need an app that does not delete GPS metadata. In 2017, the Flickr app worked for this. You can also retain GPS metadata if you download from iCloud and select "Export Unmodified Original." If you find another method/app that works, please let your teacher know so they can share this information with the rest of the class.*

- If from your phone, make sure it is a .png or .jpg!
- Don't forget to add in the necessary code to produce your Google Maps as an HTML file
  - **Make sure to change the output file name to "MarkerMap3.html"!**

In [17]:
```
mystery_location_2 = gps_from_image("photo10.jpg")
```

In [18]:
```
# Your code here
import gmaps

gmaps.configure(api_key="AIzaSyCLla6Q7krE9xNg6SnNMoGNIzjCLddE9EU") # you need to

location = []

from ipywidgets.embed import embed_minimal_html # Allows us to create a separte j

print(mystery_location_2)

location.append(mystery_location_2)

print(location)

marker = gmaps.marker_layer(location)

markermaps = gmaps.Map()
markermaps.add_layer(marker)

embed_minimal_html('MarkerMap3.html', views=[markermaps])
print("*** Check your 'Metadata Part 4' folder to find the new HTML file named \
```

```
(37.42026361111111, -122.08380416666667)
[(37.42026361111111, -122.08380416666667)]
*** Check your 'Metadata Part 4' folder to find the new HTML file named "Marker
Map3". ***
```

## Task #4: Creating a Heat Map

While taking off for a flight from O'Hare airport, you find a digital camera left behind from a previous passenger in the seat pocket. The camera has ten photos: photo1.jpg, photo2.jpg, photo3.jpg, photo4.jpg, photo5.jpg, photo6.jpg, photo7.jpg, photo8.jpg, photo9.jpg, and photo10.jpg.

Where does the rightful owner of the camera live?

To find out, get a list of coordinates from the 10 photographs.

*Note: This code uses your *gps_from_image(file_name)* function from above. If you get any errors, remember that this function will only work on a photograph that has actually been geotagged:*

In [19]:
```
location_list = []

for i in range(10):
    index = str(i + 1)
    name = 'photo' + index + '.jpg' # this lets us loop over photo1.jpg through p
    coordinates = gps_from_image(name)
    location_list.append(coordinates)

print(location_list)
```

```
[(37.7645, -122.4035), (37.776833333333336, -122.40833333333333), (37.789183333
333334, -122.39533333333334), (37.768, -122.40333333333334), (37.753525, -122.4
3464722222222), (37.74899444444444, -122.42501111111112), (37.42616666666667, -
122.1715), (37.42666666666667, -122.17183333333334), (37.413875, -122.138825000
00001), (37.42026361111111, -122.08380416666667)]
```

Get a heatmap from the photographs:

In [20]:
```
heatm = gmaps.Map()
heatm.add_layer(gmaps.heatmap_layer(location_list))
embed_minimal_html("MarkerMap4.html", views=[heatm])
print("*** Check your 'Metadata Part 4' folder to find the new HTML file named \
```

```
*** Check your 'Metadata Part 4' folder to find the new HTML file named "Marker
Map4". ***
```

**Question 7:** Where does the owner of this camera probably live? Where does the owner probably work? Explain.

*Note: Be a detective here. These two questions should have different answers.*

Your Answer: Tis person lives near san fransico, as there is a cluster of heat signature near the area. There also were heat signature miles away from the persons living area, and there were more heat signatures near the standford university school.

**Question 8:** What are the privacy and data security implications of this photo metadata? What about its benefits?

Your Answer:

Privacy: People can see you you are, your intrests, where you live, your habits, your friends and family, your contact info, your school/workplace, and camera settings

Security: Finding missing people, examining crime photos for information, discovering something new, research a crime suspect.

**Question 9:** Can you create your own heatmap? Use geotagged photos from your phone/iPad or from a collection of photos you found online.

- Don't forget to add in the necessary code to produce your Google Maps as an HTML file
  - **Make sure to change the output file name to "MarkerMap5.html"!**

In [32]:
```
# Your code here:
location_lists = []

for i in range(1):
    names = '20200307_100257.jpg'
    coordinate = gps_from_image(names)
    location_lists.append(coordinate)

print(location_lists)
```

```
[(42.112500000000004, -88.04333333333334)]
```

In [33]:
```
heatma = gmaps.Map()
heatma.add_layer(gmaps.heatmap_layer(location_lists))
embed_minimal_html("MarkerMap5.html", views=[heatma])
print("*** Check your 'Metadata Part 4' folder to find the new HTML file named \
```

```
*** Check your 'Metadata Part 4' folder to find the new HTML file named "Marker
Map5". ***
```