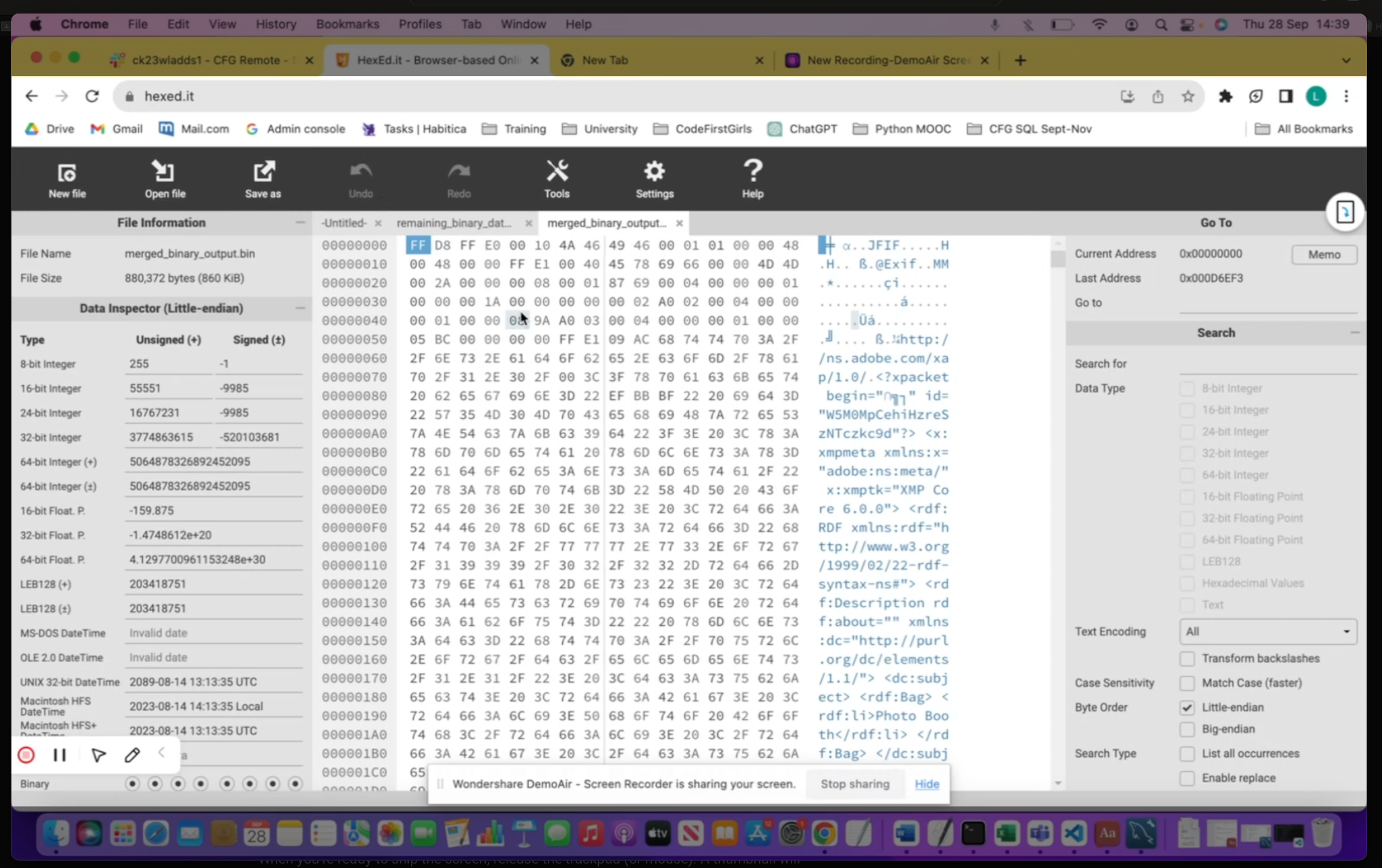
The other thing to consider is how hexed.it can help us - it’s clearly given to us for a reason - Laura had it open in the intro video



To convert the remaining data to JSON I had thought to mimic the binary to text code we have been given, just be able to use a different encoding than encoding='utf-8'

However, JSON files are just text… so maybe when we have extracted the ‘protocols’ the binary to text code will work. So sorry, no real progress on that.

The Scenario said:” Radio and electronic engineers have stated that protocols often include

synchronisation words or preamble, to signal to the receiver when specific segments

of data are about to be received. If identified, these will repeat in the message. The

purpose of these words is to synchronise the receiver to the message ahead of

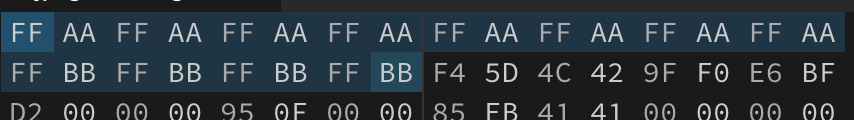
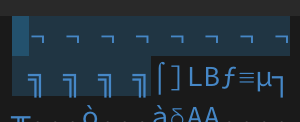
processing. In summary, we believe there is data, video frames and synchronisation

messages within the binary we have supplied you for analysis.

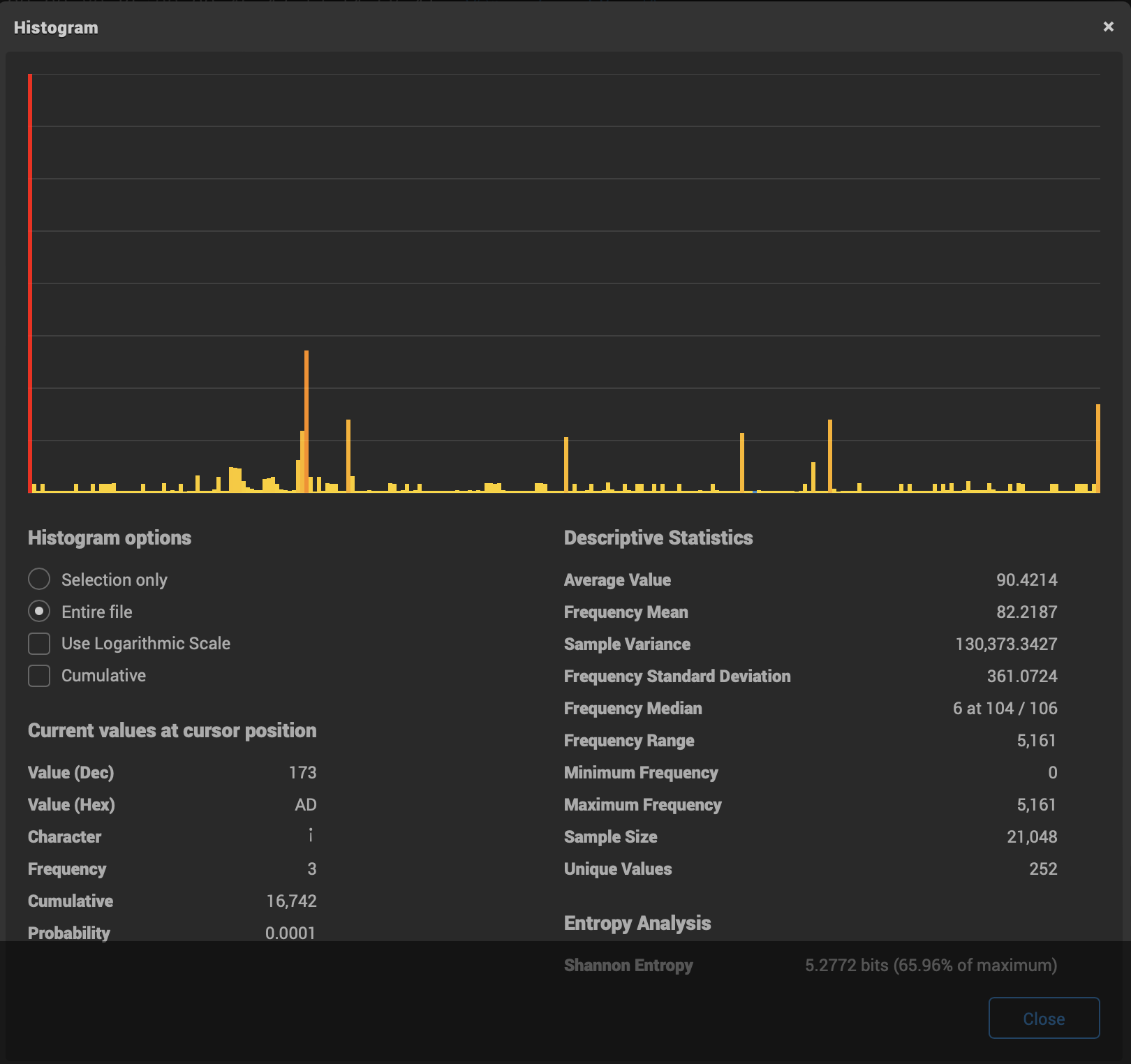
By eye, using Hexed.it, I see a repeating string “47 57 42 6F”, GWBo, we could try extracting that from the code then trying to turn it to text again.

The error message when trying to decode to text is Error decoding binary data: 'utf-8' codec can't decode byte 0xff in position 0: invalid start byte

Just found a rather more obvious repeating pattern in the Hexed.it…it was because it said ‘in position 0’ I thought that might mean at the start of the binary…



Looking at the Histogram in Hexed.it: ( tools > histogram > entire file )



Might help identify repeating strings. If you hover on the spikes it says which character they represent.

Thinking ahead - although there’s online converters, JSON to GPX, we should probably try to do it in Python…

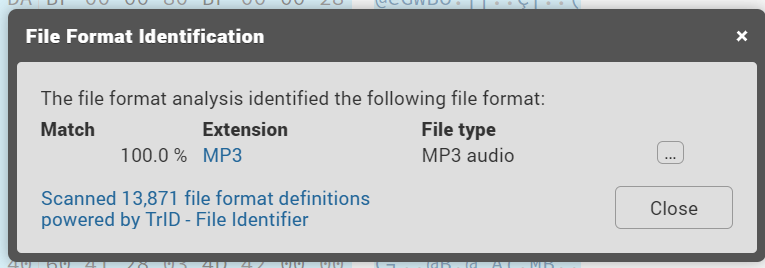
<https://pypi.org/project/gpx-converter/>

**Step 2 - Synchronization words**

*In computer networks, a syncword, sync character, sync sequence or preamble is used to synchronize a data transmission by indicating the end of header information and the start of data.The syncword is a known sequence of data used to identify the start of a frame, and is also called reference signal or midamble in wireless communications.*

<https://en.wikipedia.org/wiki/Syncword>

File Signature

* A file signature corresponds to the first bytes seen in the hex dump. The hex dump is the entire “dstl\_MOOC\_Challenge\_v1.bin” file.
* File signatures are set and every file has a specific code relating to its file type. These are not randomly generated. For example if a hex dump had “4E 45 53” at the very beginning then anyone analyzing it would automatically know this was a Nintendo Entertainment System image file and so on.
* In our case FF AA FF AA… is a repeating code. With FF at the very start being the F.S
  + *Therefore FF AA FF AA FF AA FF AA FF AA FF AA FF AA FF AA FF BB FF BB FF BB FF BB is a synchronization word*
* So when I selected the entire hex dump i was able to use hexit to identify file format as well.
* 
* **The first byte at offset 0 is FF = F.S**
* **https://file-extension.net/seeker/file\_extension\_mp3**
* Thus, we are working with a MP3 and not text data.

Further proof

* If you select any ASCII coded letter in Hexit on the right hand column e.g. “L”
  + 
* If you select any ANSI coded letter, which is not ASCII(plain text) then the result is this
  + 

This is just further proof that the hexdump is not a plain text file

Synchronization word - FF AA FF AA FF AA FF AA FF AA FF AA FF AA FF AA FF BB FF BB FF BB FF BB

* In the hexit search tool, when we enter the full sync word FF AA… and click search we can see there are
* Therefore, this sync word is marking the start of each section. Additionally every separate section has the same File Signature. Proving that these are separate files in one hexdump.

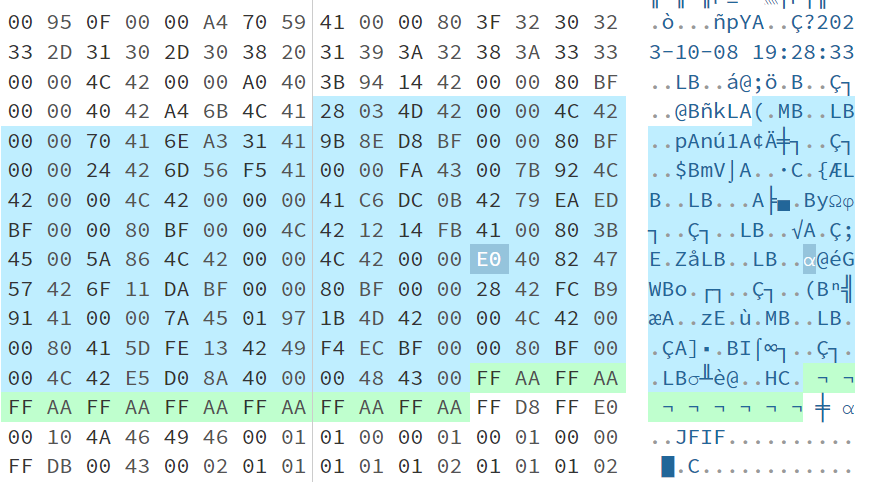
File Structure

* Each section begins with the same sync word and F.S. (Header?)
* Each section ends with the same code (footer?)
  + (MB LB pAn£1A›ŽØ¿ €¿ $BmVõA úC {’LB LB AÆÜ

Byêí¿ €¿ LBûA €;E Z†LB LB à@‚GWBoÚ¿ €¿ (Bü¹‘A zE—MB LB €A]þBIôì¿ €¿ LBåÐŠ@ HC (MB LB pAn£1A›ŽØ¿ €¿ $BmVõA úC {’LB LB AÆÜ

Byêí¿ €¿ LBûA €;E Z†LB LB à@‚GWBoÚ¿ €¿ (Bü¹‘A zE—MB LB €A]þBIôì¿ €¿ LBåÐŠ@ HC (MB LB pAn£1A›ŽØ¿ €¿ $BmVõA úC {’LB LB AÆÜ

Byêí¿ €¿ LBûA €;E Z†LB LB à@‚GWBoÚ¿ €¿ (Bü¹‘A zE—MB LB €A]þBIôì¿ €¿ LBåÐŠ@ HC

* + In HEX: 28 03 4D 42 00 00 4C 42 00 00 70 41 6E A3 31 41 9B 8E D8 BF 00 00 80 BF 00 00 24 42 6D 56 F5 41 00 00 FA 43 00 7B 92 4C 42 00 00 4C 42 00 00 00 41 C6 DC 0B 42 79 EA ED BF 00 00 80 BF 00 00 4C 42 12 14 FB 41 00 80 3B 45 00 5A 86 4C 42 00 00 4C 42 00 00 E0 40 82 47 57 42 6F 11 DA BF 00 00 80 BF 00 00 28 42 FC B9 91 41 00 00 7A 45 01 97 1B 4D 42 00 00 4C 42 00 00 80 41 5D FE 13 42 49 F4 EC BF 00 00 80 BF 00 00 4C 42 E5 D0 8A 40 00 00 48 43 00
  + 
  + (blue is the sync word for the end of the previous section and green is the sync word for the beginning of a new section)
  + We care more about the hex data(left column) as it is more easily translated to decimal/binary etc.
* As this is another repeating section of code we can classify it as another sync word
* The body of each section follows the same structure, there is a time argument in the same area etc.
* Right before the footer element (see attached screenshot above), you can see a date/time. Now going through each section it appears they are in order
* E.g.
  + 2023-10-08 19:28:31
  + 2023-10-08 19:28:32
  + 2023-10-08 19:28:33…
* From the sequence of time stamps, it could be assumed it is video frames as the difference is 1 second between each section and it aligns with the type of file signature we found.
* 89 instances / separate frames
* **Note: I did not check if each section was the exact length. I just found the start/end so now we can define the protocol and go ahead with step 3.**

**Step 3 - Defining the protocol**

**Next steps/how?**

1. Examine content of each ‘frame’ between sync words
   1. Sub sections?
   2. Checksums?

I’m thinking I should run the same Python code I used to separate the image frames, but with the new start and trailer codes, on the binary data left from extracting the image files, to see if it will extract the MP3 files, and collect any remaining binary data (the JSON file presumably).

I used b’\xFF\xAA\xFF\xAA\xFF\xAA\xFF\xAA\xFF\xAA\xFF\xAA\xFF\xAA\xFF\xAA\xFF\xBB\xFF\xBB\xFF\xBB\xFF\xBB’

for start code as I think that means read this hex as binary.

And similarly

\x28\x03\x4D\x42\x00\x00\x4C\x42\x00\x00\x70\x41\x6E\xA3\x31\x41\x9B\x8E\xD8\xBF\x00\x00\x80\xBF\x00\x00\x24\x42\x6D\x56\xF5\x41\x00\x00\xFA\x43\x00\x7B\x92\x4C\x42\x00\x00\x4C\x42\x00\x00\x00\x41\xC6\xDC\x0B\x42\x79\xEA\xED\xBF\x00\x00\x80\xBF\x00\x00\x4C\x42\x12\x14\xFB\x41\x00\x80\x3B\x45\x00\x5A\x86\x4C\x42\x00\x00\x4C\x42\x00\x00\xE0\x40\x82\x47\x57\x42\x6F\x11\xDA\xBF\x00\x00\x80\xBF\x00\x00\x28\x42\xFC\xB9\x91\x41\x00\x00\x7A\x45\x01\x97\x1B\x4D\x42\x00\x00\x4C\x42\x00\x00\x80\x41\x5D\xFE\x13\x42\x49\xF4\xEC\xBF\x00\x00\x80\xBF\x00\x00\x4C\x42\xE5\xD0\x8A\x40\x00\x00\x48\x43\x00

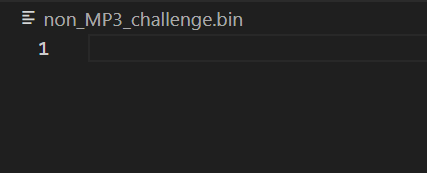
It could be the data was a sequence of jpg image, timestamp, audio and GPS location for every second as the drone flew?

I tried it - I’ll email the python workbook to you.

It extracted files but I couldn’t play them.

We now have the binary left over after the JPG and MP3s have been extracted and so need to extract it as a JSON….

When I ran your code it said there was no mp3 data. Could you send what was extracted?



I am just playing around seeing if I can extract as mp4 etc and still nothing.

Reread the brief and it says to just extract the data to a JSON directly after defining the protocol

So sync words = protocol

Made a rule/protocol for extracting to JSON

I have to convert to binary in the process overwise i geterrors

I told VSCODE to extract data for any 2 components it found in the file. I wasn’t expecting it to work, but the code ran and i was able to get:

{ "ExtractedData1": 4290510779, "ExtractedData2": 3850849346 },

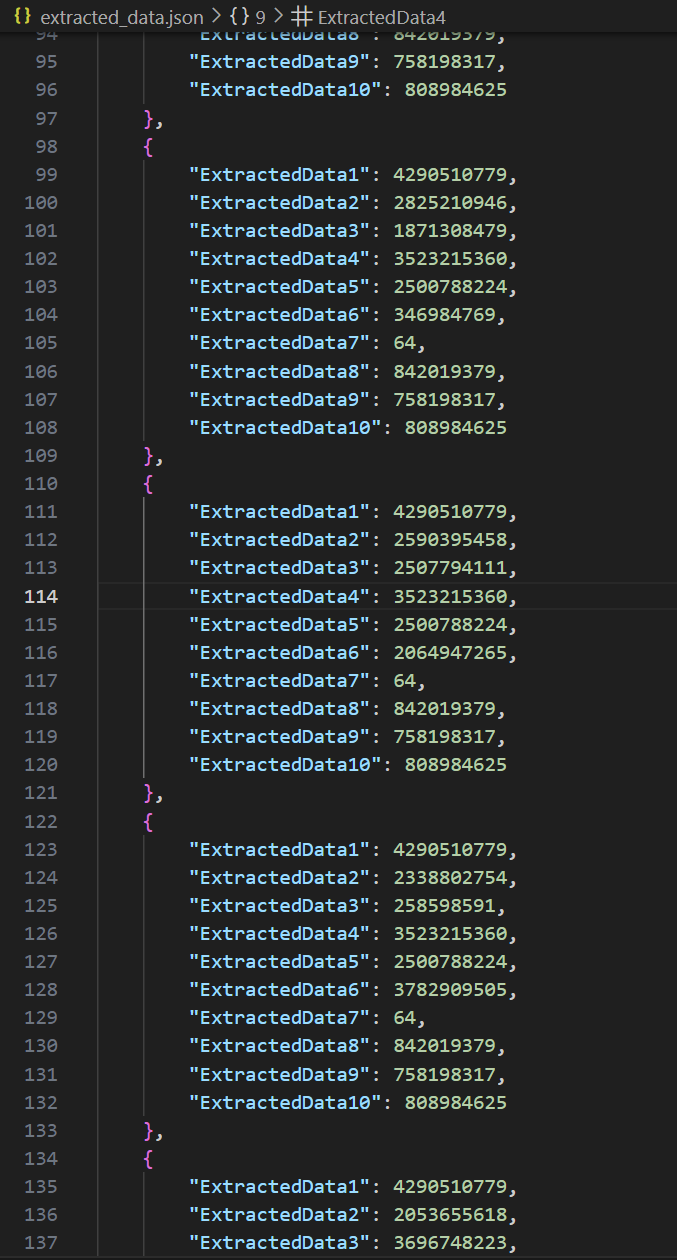
I wasn’t sure what the numbers meant until I saw the next step about GPX.

So the 10 digit numbers are actually coordinates i think.

3850849346 becomes 38.508, -49.346 etc

So in my jSON file, 88 data blocks were extracted, I decided to go and and say to find 10 pieces of data instead of 2. Again, the code ran and even more numbers were found.

I pasted a small screenshot just so you can see the majority of the data is the same for each block with small changes.

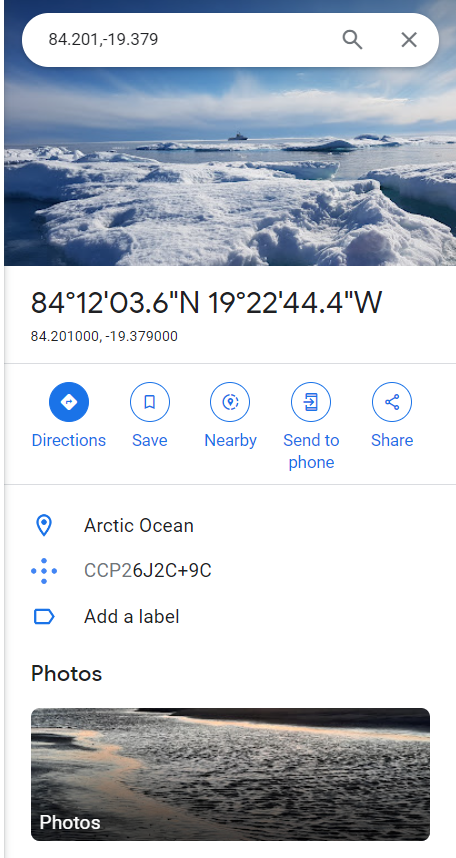
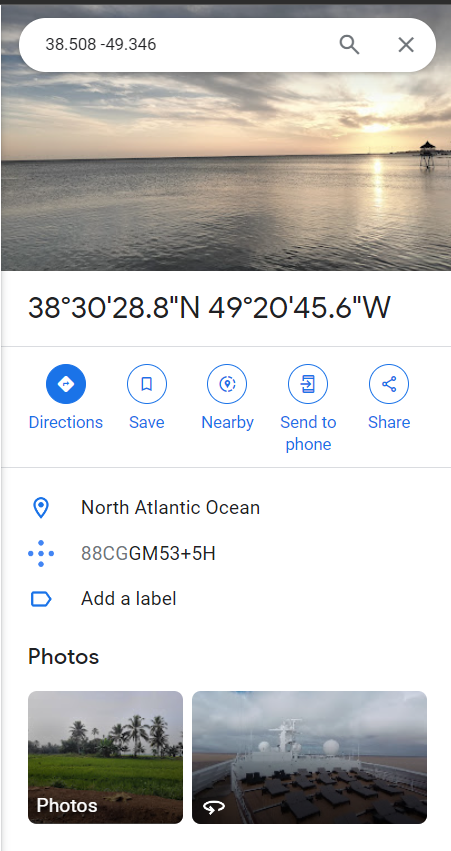


So i put in the 10 digit numbers into google maps.

All of the co-ordinates pointed to one of these 2 locations.

We might find more locations with more extracted data?

But what do the other numbers (not 10 digits) mean?



You can see a military ship in screenshot 1. It wont let me view photos at all or click on anything. Suspicious?

Also, these locations don’t look like the land images extracted at jpg in step 1.

Hidden data?

Also the data protocol goes up in multiples of 4 as “because typical data extraction from binary files often involves working with 4-byte (32-bit) integers. This is a common practice when dealing with binary data because many data structures and file formats use 32-bit integers for various types of information.” - google.

So i decided to go up to 30 just to see if there was any more new numbers/data… and yep there is.

I added the end\_sync\_word to hopefully just extract the data between the start/end S.Ws and cut down on the amount of data…

Nope it didn’t cut it down at all

I tried to get it to loop around for 30 times, but the code didn’t work

18/10/23

I opened your code in a clean environment, and I was able to extract and see what you were seeing so it makes much more sense to me now.

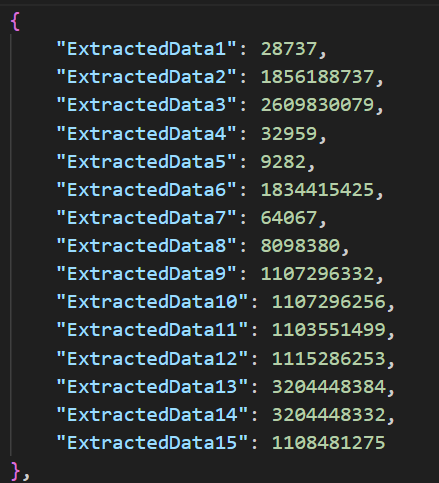
**Opened “non\_MP3\_challenge.bin”**

* Start sync word = ‘\x4D\x42\x00\x00\x4C\x42'
* End sync word 1 = '\x00\x00\x7A\x45\x01\x97\x1B'
* End sync word 2 = '\x00\x00\x48\x43\x00'

Total number of data blocks extracted = **348**

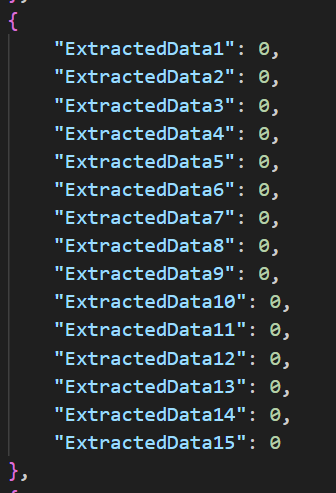
When i run the code to extract as a JSON

* Data blocks ending in '\x00\x00\x7A\x45\x01\x97\x1B' have data extracted
* Data blocks ending in '\x00\x00\x48\x43\x00' have no data extracted



A pattern of

* Starts with ‘\x4D\x42\x00\x00\x4C\x42' && end with '\x00\x00\x7A\x45\x01\x97\x1B'



* Starts with ‘\x4D\x42\x00\x00\x4C\x42' && ends with '\x00\x00\x48\x43\x00'
* Starts with ‘\x4D\x42\x00\x00\x4C\x42' && ends with '\x00\x00\x7A\x45\x01\x97\x1B' BUT AS SECOND INSTANCE IN THIS PATTERN 

**So every 3 blocks of data this same pattern repeats with the same values seen**

**EVERY MP3 file also follows this pattern EXCEPT for the header**

**87 instances of the filled data block (**MORE **THAN 40 BYTES LENGTH)**

**261 instances of non-filled data block(**LESS **THAN 40 BYTES LENGTH)**

MP3 file header that contains date/time and does not follow this rule

The data in between is different as it is date/time

* Starts with ‘0xFF,0xAA,0xFF,0xAA,0xFF,0xAA,0xFF,0xAA,0xFF,0xAA,0xFF,0xAA,0xFF,0xAA,0xFF,0xAA,0xFF,0xBB,0xFF,0xBB,0xFF,0xBB,0xFF,0xBB,0xFF,0xAA,0xFF,0xAA,0xFF,0xAA,0xFF,0xAA,0xFF,0xAA,0xFF,0xAA,0xFF,0xAA,0xFF,0xBB,0xFF,0xBB,0xFF,0xBB,0xFF,0xBB’
* Ends with ‘0x41,0x28,0x03’

Entire transmission lasted 90 seconds

New information just in:

**Aside from the images...**

(Going full method actress here and putting on my best M from James Bond voice)

Another team has identified long and short sync words that hint there is a data packet composed of 32 bit floats, integers, booleans, and strings attached to each image file.

:bangbang: Hint: I suggest you view the pictures and see if there is anything can be gleaned from the each video frame... perhaps a data packet's contents and structure? :upside_down_face:

It seems that the structure of the file is a repeating pattern of:

* Long sync word
* Image
* Short sync word
* Data packet - analysts suspect the data packet is 215 bytes long.

:bangbang: Hint: extract the images, then subtract the binary that makes them up. Open the remaining binary data in [hexed.it](http://hexed.it/) (online hexadecimal viewer) to see if you can identify the sync words!

That's all the extra intelligence I can give you guys for now!

:heart: Laura

And looking back at the challenge document it says:

1. Extract as many video frames as possible

2. Extract protocol elements such as synchronisation words

3. Define the protocol

4. Extract the data as a JSON.

I think that means extract the synchronisation words, leaving us with a 215 bytes package after each ‘short sync word’ - which we thought was the start sync word, but it looks like it’s the end one. So I think we need to study those 215 bytes and work out the structure in them, the protocol in them, Laura says “there is a data packet composed of 32 bit floats, integers, booleans, and strings attached to each image file.”

So we need to identify what mix of those make up each packet, then use ‘struct pack’ like Laura did in session 3 and 4, the protocol refers to the type and order of the data, so e.g.

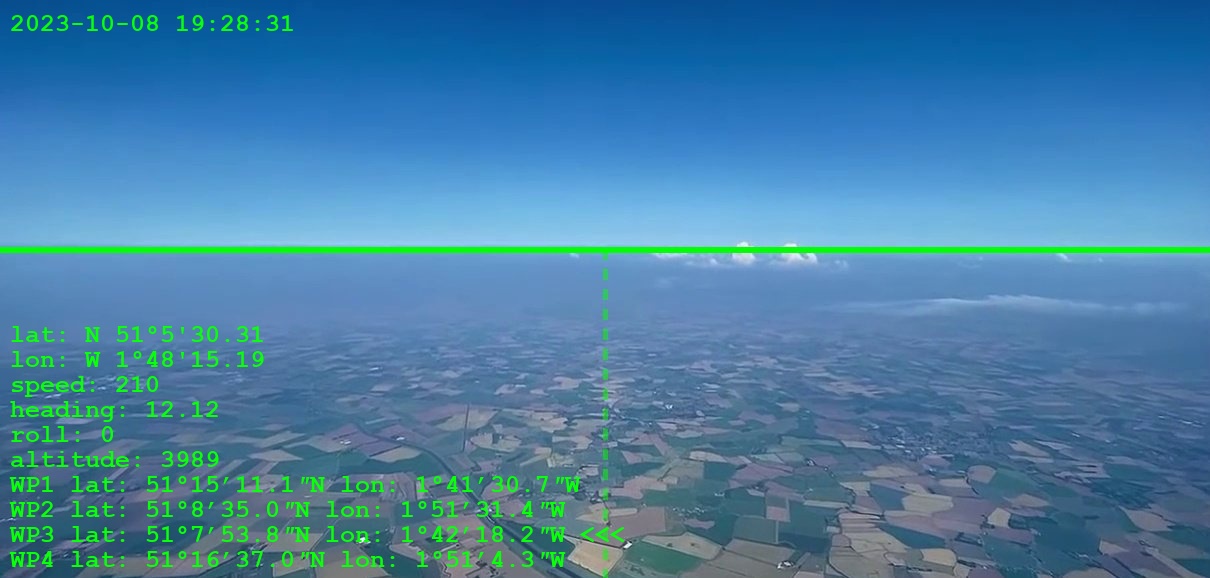
my\_packed\_data = struct.pack('if??', int\_value, float\_value, boolean\_true, boolean\_false)

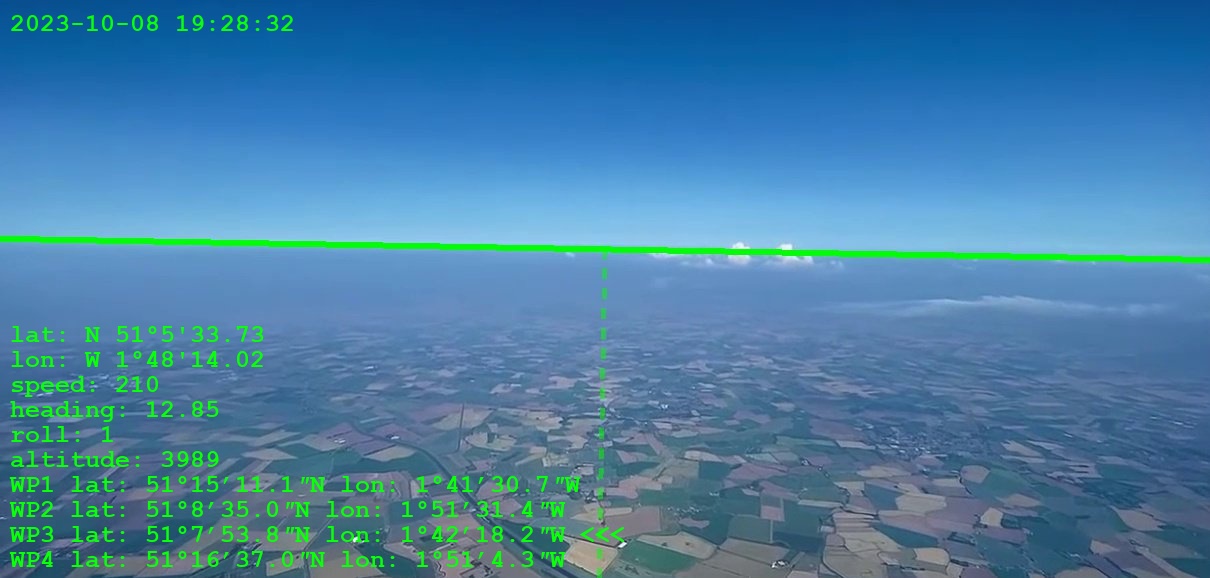
For this the protocol is “if??”

I’ve been trying to work with Hexed.it but am struggling - I can’t seem to copy selected code in order to search for it.

I thought to extract MP3 files by extracting the data between the synchronisation words, but I think that was wrong - partly because it removes the data stamps when I did that.

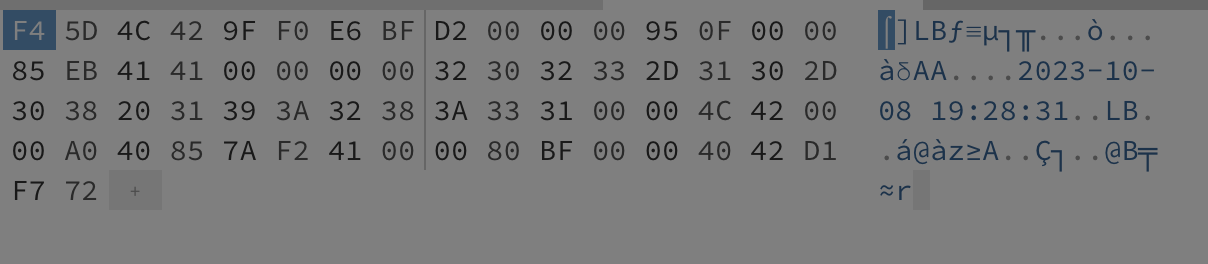
I think we just needed to extract the synchronisation words and then look at what we have between the long sync and the short sync.





The writing on the photos should tell us the structure of the data: time stamp, lat, lon, speed, heading, roll, altitude

This is the first block between the two synchronisation words (a last 41 got cut off) - that’s not 215?



I don’t feel like I’ve got us very far!... In the first session they said to learn more about data types <https://realpython.com/python-variables/> but it doesn’t tell me much. I more want to know how, for example, a 32 bit float or integer is stored in binary…

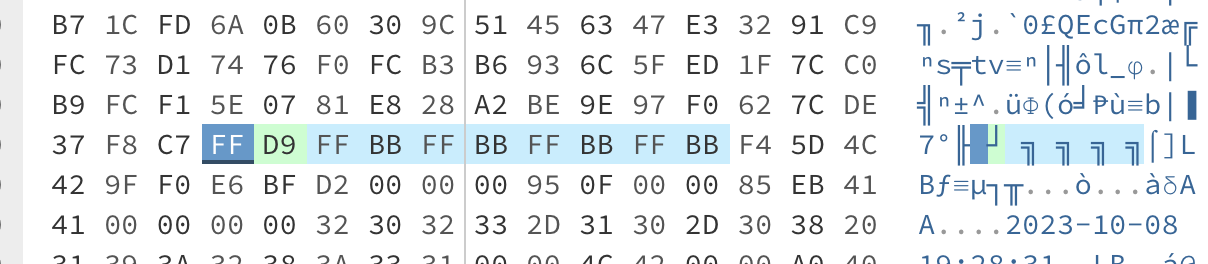
I don’t think we have the correct synchronisation codes. Laura said the order is

Long sync word

* Image
* Short sync word
* Data packet - analysts suspect the data packet is 215 bytes long.

So

FF AA FF AA FF AA FF AA FF AA FF AA FF AA FF AA FF BB FF BB FF BB FF BB

Is the long sync word not the short, we should find the short sync word after the jpeg image trailer marker: FF D9 trailer , start of JPEG is FF D8

Looking at it I think the short sync word is FF BB FF BB FF BB and the long FF AA FF AA FF AA FF AA FF AA FF AA FF AA

8 bits in one byte

<https://www.rapidtables.com/convert/number/hex-to-binary.html>

First 215 data block

F45D4C429FF0E6BFD2000000950F000085EB414100000000323032332D31302D30382031393A32383A333100004C420000A040857AF241000080BF00004042D1F7724128034D4200004C42000070416EA331419B8ED8BF000080BF000024426D56F5410000FA43007B924C4200004C4200000041C6DC0B4279EAEDBF000080BF00004C421214FB4100803B45005A864C4200004C420000E040824757426F11DABF000080BF00002842FCB9914100007A4501971B4D4200004C42000080415DFE134249F4ECBF000080BF00004C42E5D08A400000484300

Photo 1 (Differences to photo 2 highlighted)

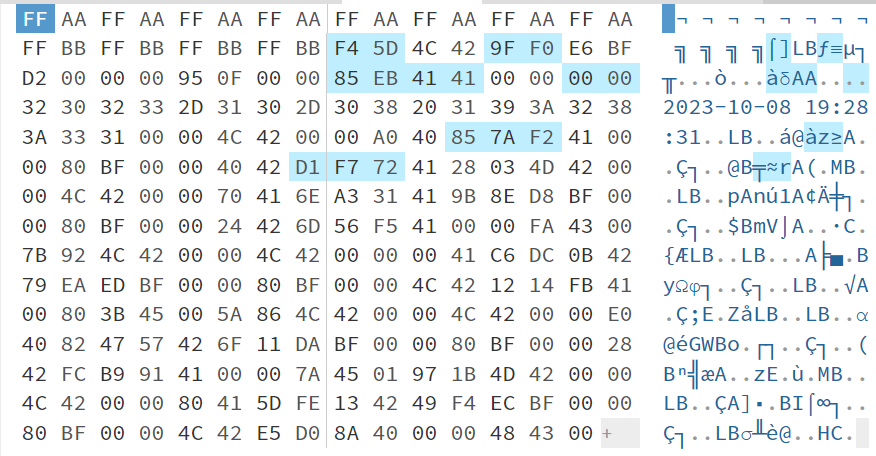


Photo 2 (Differences to photo 1 & 3 highlighted)

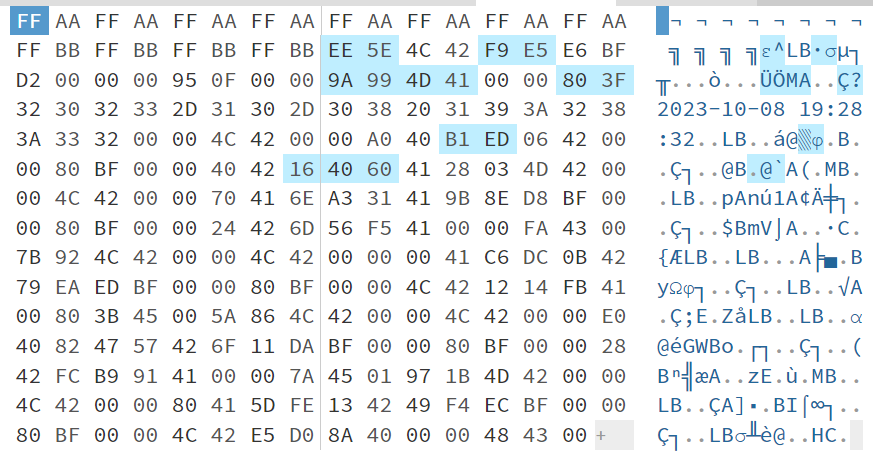
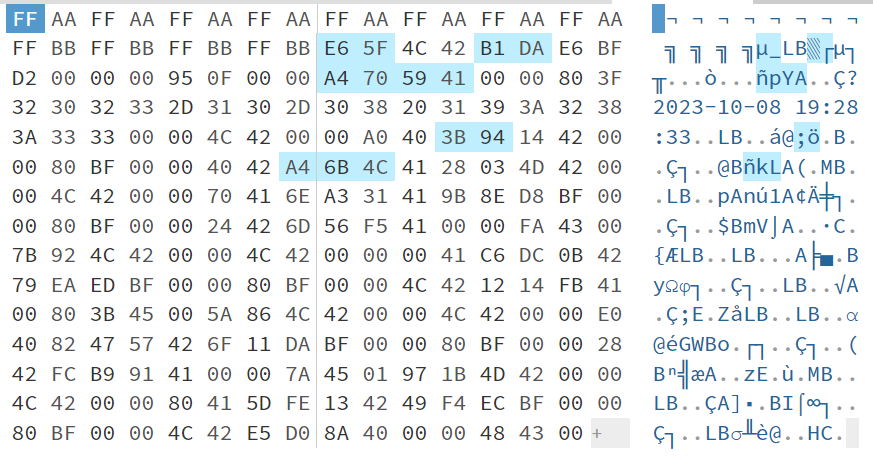
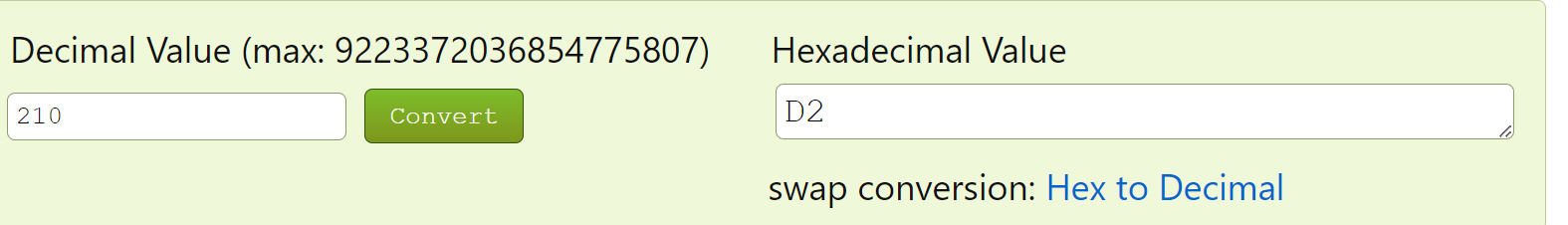


Photo 3 (Differences to photo 2 highlighted)



**What we know from the jpegs**

* Date/time located at offset 30
* Header is always the same
* Between photos 1 & 2 there is a difference
  + “\x00\x00” in photo 1 ( seen as blank space)
  + “\x80\x3F” in photos 2 (& 3) (seen as C?)
  + Could this be the “roll” attribute?
* The last portion is always entirely the same → Altitude values ++ WP1,WP2,WP3,WP4
* In each jpeg, we can see that 3 figures always change (lat,lan,heading)
  + Likewise in the hex data,
* A number that appears to not change is “speed:210”
  + 
  + Seen as the π

I am unsure if this is the right path as some similarities are not adding up.

For example, in photos 1 & 2 the first value for “heading” is both 12, but that does not correspond to anywhere in the hex data.

Another thing, is that the order of the different data in hexit does not make sense when looking at the photos.

I think you are on the right path - because of what Laura told us.

I’m going to take a look back at the session powerpoints / videos to see if I can see more about how different data types are stored.

Floating point: 4 bytes, consisting of a sign bit, an 8-bit excess-127 binary exponent, and a 23-bit mantissa.

Whole numbers (integers) are usually represented with 4 bytes, or 32 bits.

Boolean seems to be 2 bytes

ASCII

There are three sets of 8 bytes before the date stamp. I don’t think it’s the same each time but what follows seems to be the same…

The string after the date stamp ( 00 00 4C 42 00 00 A0 40 ) repeats for a few iterations then changes.

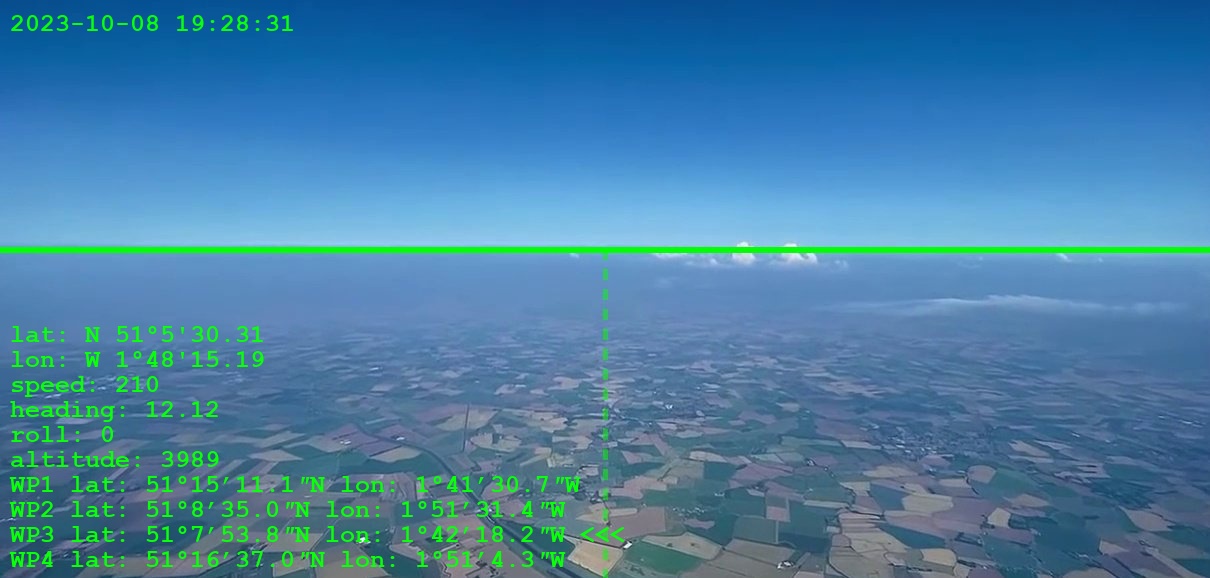
I felt at a bit of a dead end so I messaged Laura outlining where I thought we were now and asking for a hint.

Hi Laura, my working partner and I have managed to extract the images, and extract the sync codes, and found the data packets (215 bytes) between them. We are now struggling to define the protocol. I assume this is the protocol to describe the content of the 215 bytes, e.g., your example using my\_packed\_data = struct.pack('if??', int\_value, float\_value, boolean\_true, boolean\_false) has protocol "if??" . Looking at the images we extracted we see time and date stamp, lat, Lon, speed, heading, roll and altitude. This is followed by the 4 waypoints the craft is following. However, looking at the Hex, the time stamp doesn't come first, there's three sets of 8 bytes before that. The time stamp takes 19 bytes. The last section of each packet doesn't seem to change so I'm guessing it is the waypoints. I did some research and found floating points and integers take 4 bytes and booleans 2. But how to know which bytes are what form of data? Should we just guess and see if what we guessed makes sense when unpacked? We need a little help please! Thanks, Sara

And got the following response:

Hey Sara, you're very much on the right track! My hint is - ignore most of the text (e.g. the lat and lon text) and focus on the numbers and booleans. You're right that the date would be a 19 character long string, so 19s in a struct.unpack() parameter.

I’m thinking to play with python, packing some of the data we’ve got, like the numbers in the jpegs



And seeing what it looks like in binary, comparing that to the binary packet we have. I don’t see any booleans though, as the roll goes up to 4 later so isn’t a boolean.

How would we pack a lat or lon string? N 51º5’30.31

A string 1 length, int, int, float? So

my\_packed\_data = struct.pack('1siff', hemisphere\_char, degrees\_int, mins\_int, secs\_float)

What do you think? I’ll give it a try when I get some time, see what binary it produces and look for it in the data packet - but this all feels very much like guess work!

<https://docs.python.org/3/library/struct.html>

| **Format** | **C Type** | **Python type** | **Standard size** |  |
| --- | --- | --- | --- | --- |
| x | pad byte | no value |  |  |
| c | char | bytes of length 1 | 1 |  |
| b | signed char | integer | 1 |  |
| B | unsigned char | integer | 1 |  |
| ? | \_Bool | bool | 1 |  |
| h | short | integer | 2 |  |
| H | unsigned short | integer | 2 |  |
| i | int | integer | 4 |  |
| I | unsigned int | integer | 4 |  |
| l | long | integer | 4 |  |
| L | unsigned long | integer | 4 |  |
| q | long long | integer | 8 |  |
| Q | unsigned long long | integer | 8 |  |
| n | ssize\_t | integer |  |  |
| N | size\_t | integer |  |  |
| e | (6) | float | 2 |  |
| f | float | float | 4 |  |
| d | double | float | 8 |  |
| s | char[] | bytes |  |  |
| p | char[] | bytes |  |  |
| P | void\* | integer |  |  |

When I searched for the lan/lon values in the hex translation. For e.g. i would translate what the number 55 was in hex, then search for it and it wouldn’t appear

Is the data mismatched? broken ? encrypted?

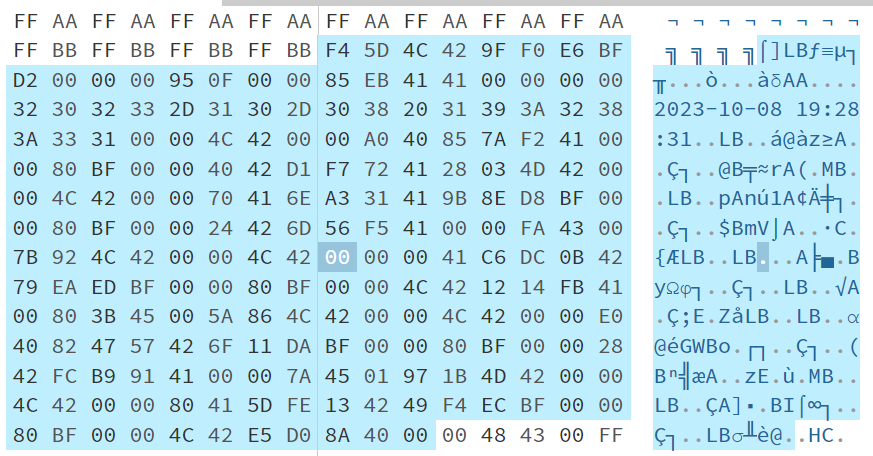
This is why ^ i’m so confused as the data seen in the images is not exactly seen in the data.

Just an update from me: I was finishing my web dev project. So i just need to record a live demo of that and i’ll take a look at this tomorrow - t

**In non\_jpeg\_challenge.bin**

-Start long sync word at start \xFF\xAA…

-215 bytes in between

End short sync word \x00\x48\x43

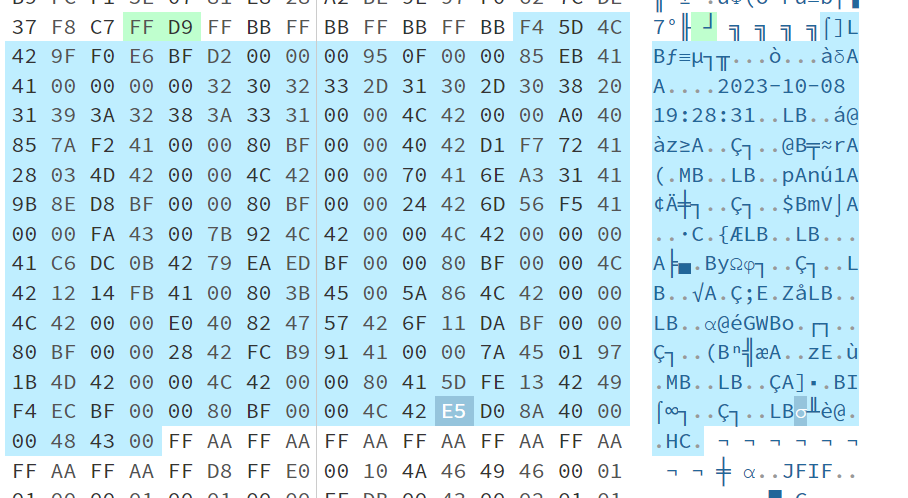
We have extracted the images already in part 1?

**In dstl\_mooc\_challenge\_v1.bin**

Between the image trailers FFD8 and FFD9, whats extracted makes up non\_jpeg\_challenge.bin

I scrolled up and looked at what you thought the sync words here were, and i agree.

“Looking at it I think the short sync word is FF BB FF BB FF BB and the long FF AA FF AA FF AA FF AA FF AA FF AA FF AA”



Either way we will both get the same data from either file. So I think the sync words are defined now for certain.

So with those sync words I've tried to struct the data. I kept getting errors and I did a bit of research and found that. The synchronization words like ('\xFF\xAA\xFF\xAA\xFF\xAA\xFF\xAA\xFF\xAA\xFF\xAA\xFF\xAA') are hexadecimal bytes. We can use these words to search for synchronization patterns within the binary data, but they cannot be directly used with struct for parsing.

Kept trying, kept getting errors so i went back to hexit and just checked the bytes at the start and end of each data packet that were not FF AA FF etc

**My code**

import re

import struct

import json

import base64

# Define your long and short synchronization words as bytes

long\_sync\_word = b'\xBB\xFF\xBB\xFF\xBB\xFF\xBB'

short\_sync\_word = b'\x00\x48\x43'

# Open and read the binary file

with open('non\_jpeg\_challenge.bin', 'rb') as binary\_file:

binary\_data = binary\_file.read()

# Initialize a list to store the 215-byte data packets

data\_packets = []

# Find the positions of long and short synchronization words

long\_sync\_positions = [m.start() for m in re.finditer(re.escape(long\_sync\_word), binary\_data)]

short\_sync\_positions = [m.start() for m in re.finditer(re.escape(short\_sync\_word), binary\_data)]

# Iterate through the synchronization words to extract data packets

for i in range(len(long\_sync\_positions)):

data\_packet\_start = long\_sync\_positions[i] + len(long\_sync\_word)

data\_packet\_end = short\_sync\_positions[i]

# Extract the binary data for the data packet

data\_packet\_bytes = binary\_data[data\_packet\_start:data\_packet\_end]

# Now you can parse the data\_packet\_bytes using your chosen format (struct or other methods).

print(data\_packet\_bytes) # Print data for demonstration

# Data extraction and processing are completed

# Extract the binary data for the data packet

data\_packet\_bytes = binary\_data[data\_packet\_start:data\_packet\_end]

# Encode the data\_packet\_bytes to Base64

data\_packet\_base64 = base64.b64encode(data\_packet\_bytes).decode('utf-8')

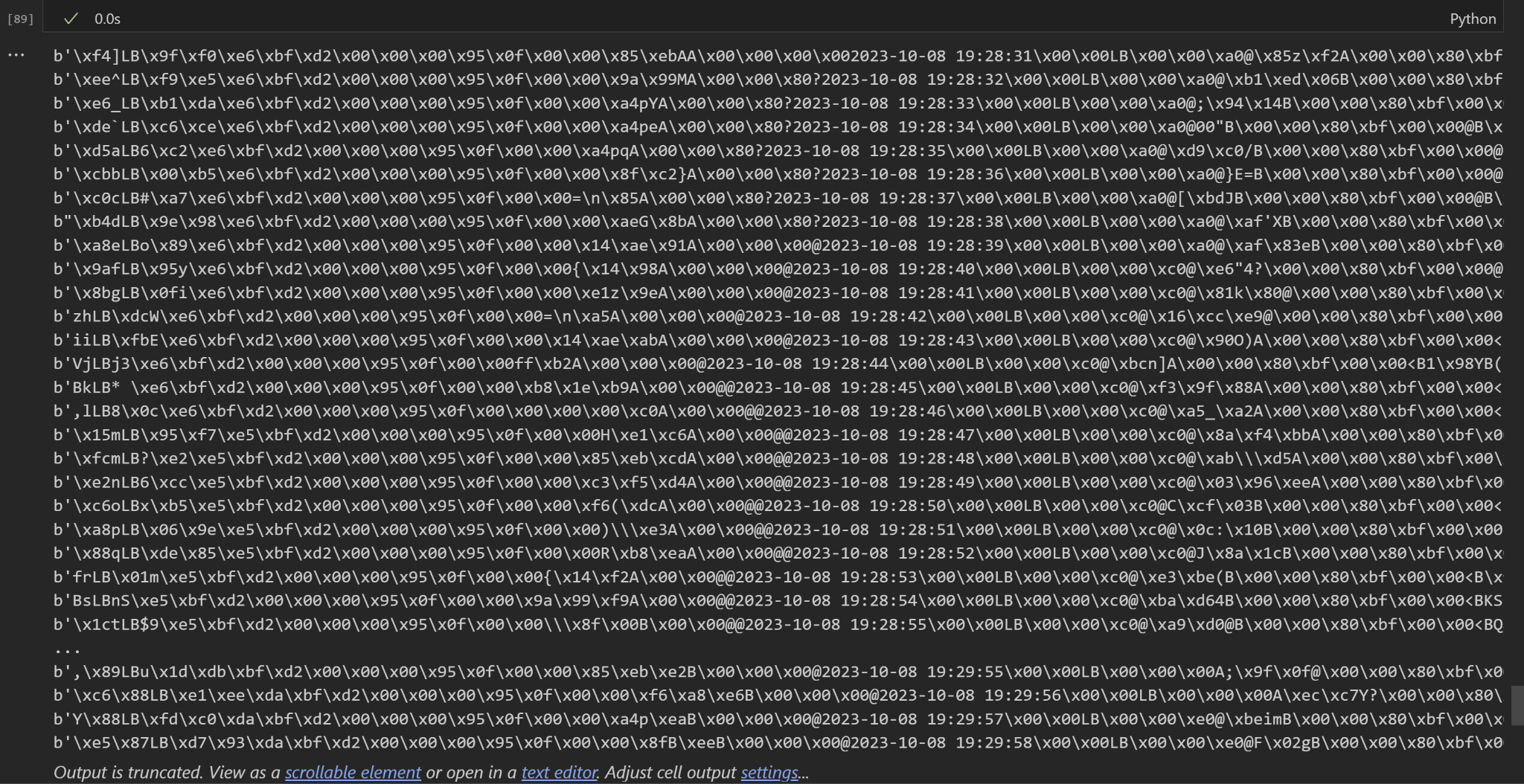
data\_packets.append(data\_packet\_base64)

# Save the extracted data to a JSON file

with open('extracted\_data.json', 'w') as json\_file:

json.dump(data\_packets, json\_file)

# Data extraction and processing are completed  
  
**I got this output**

****

So it looks like it has split and now we have all the separate data packets?

Is this what you think the end goal of this would be?

I also went and copied and pasted the exact piece of code with what we thought were the sync words and I still got errors and it wasn't able to use the struct element. So it's definite that those sync words cannot be used when structing.

**long\_sync\_word = b'\xBB\xFF\xBB\xFF\xBB\xFF\xBB'**

**short\_sync\_word = b'\x00\x48\x43'**

So i was then able to extract this and save it to a JSON

I did a count to see how many data packets had been extracted and I got 88.

I know in the brief it said the data towards the end didn’t transfer properly so that could explain the missing 2 data packets.

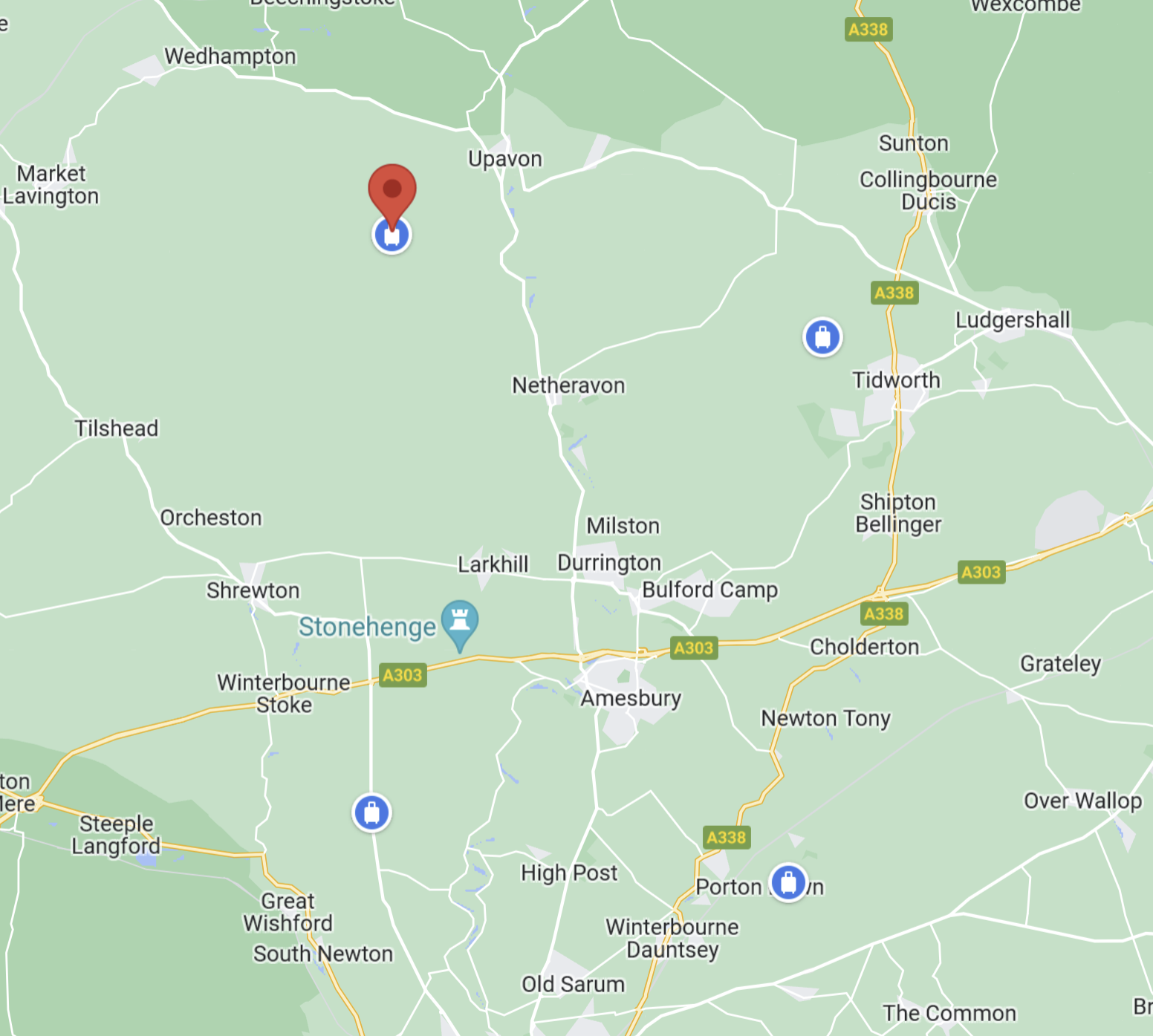
**Image analysis**

Waypoint 1 :https://www.google.com/maps/place/51%C2%B015'11.1%22N+1%C2%B041'30.7%22W/@51.2530828,-1.7021608,15z/data=!4m4!3m3!8m2!3d51.2530833!4d-1.6918611?entry=ttu

Waypoint 2: https://www.google.com/maps/place/51%C2%B008'35.0%22N+1%C2%B051'31.4%22W/@51.1430551,-1.8690219,15z/data=!4m4!3m3!8m2!3d51.1430556!4d-1.8587222?entry=ttu

Waypoint 3: https://www.google.com/maps/place/51%C2%B007'35.8%22N+1%C2%B042'18.2%22W/@51.1266106,-1.7153553,15z/data=!4m4!3m3!8m2!3d51.1266111!4d-1.7050556?entry=ttu

Waypoint 4: <https://www.google.com/maps/place/51%C2%B016'37.0%22N+1%C2%B051'04.3%22W/@51.2769439,-1.8614941,15z/data=!3m1!4b1!4m4!3m3!8m2!3d51.2769444!4d-1.8511944?entry=ttu>





I bring the waypoints up as my colleague is a drone flyer and said there there is usually 6 waypoints. Maybe that is hidden? It would tell us where the drone came from/was going next?

What i think is next

I had a look at session 4 notebook and i think that’s where we start next, like you had mentioned.

Mostly just the second half of the document is useful for us

Just replacing the names/sync words until we get a half decent output 🤣