# NOISY

- Analyzing encrypted WAV files with Python


- Ajay Balguri
- Kanaka Durga Grandhi
- Anvitha Kanukuntla
- Naveen Kumar

# OVERVIEW

Main agenda is to decrypt the hidden message from a WAV file.

Tools used

- scipy
- Matplotlib
- numpy.

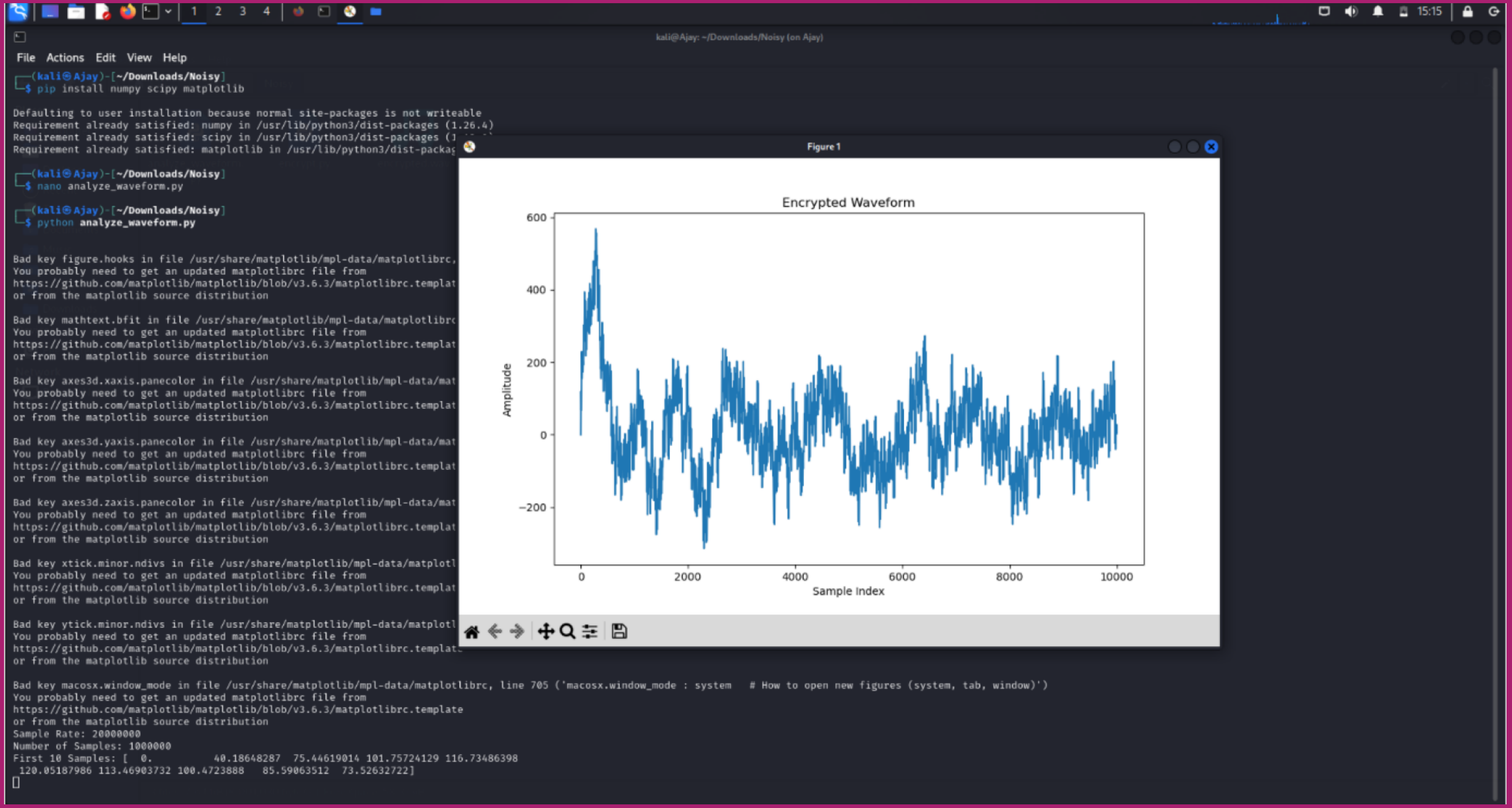
# TOOLS AND PYTHON LIBRARY

The **scipy** Python library is a comprehensive ecosystem for scientific and technical computing. It provides a variety of modules and functions that facilitate tasks in mathematics, science, and engineering.

**Matplotlib** is a widely used Python library for creating static, interactive, and animated visualizations. It provides tools to generate plots, graphs, and charts to help visualize data effectively.

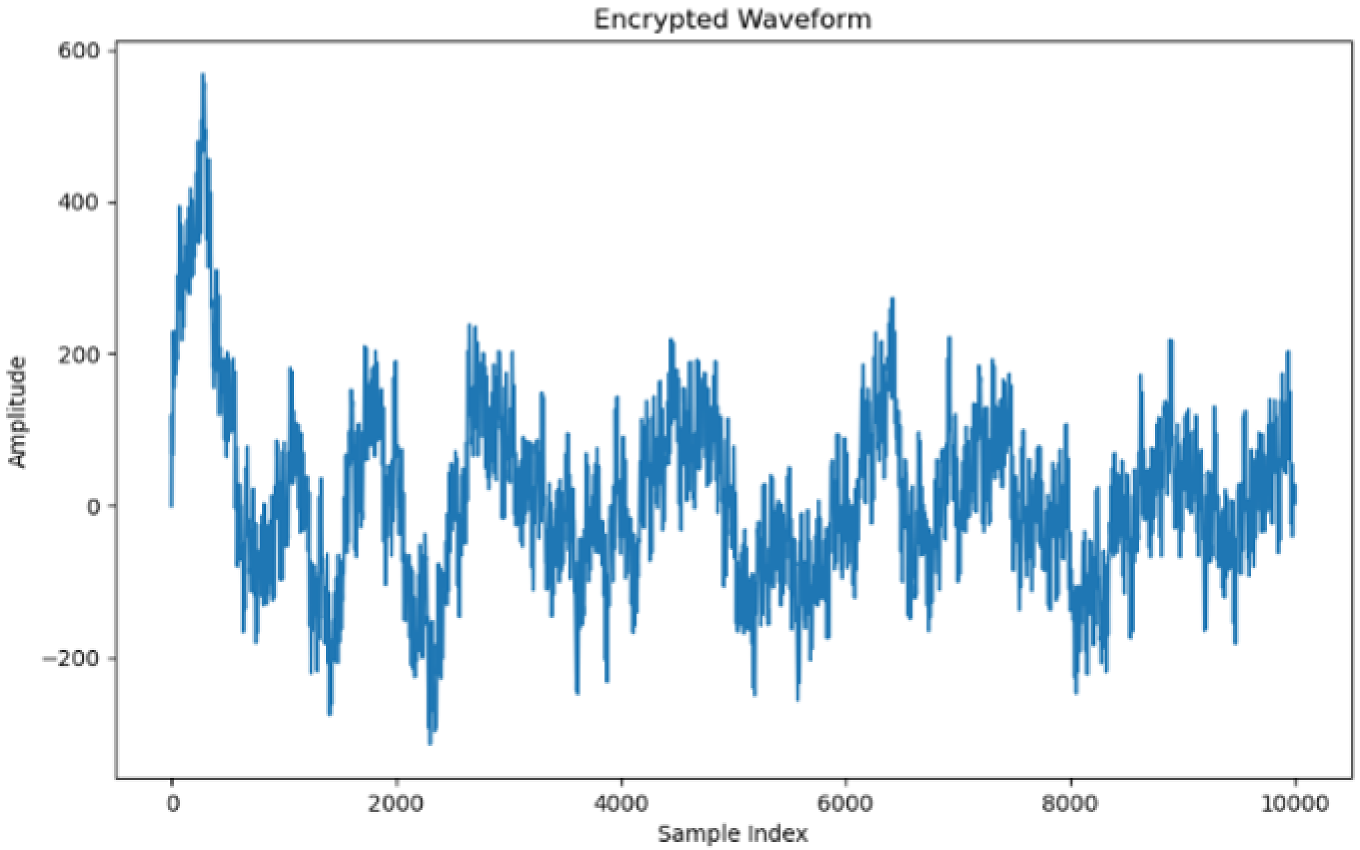
**NumPy** is a powerful Python library primarily used for numerical computing. It provides support for working with large, multi-dimensional arrays and matrices, along with a vast collection of mathematical functions to operate on these arrays efficiently.

# ANALYZING THE WAV FILE



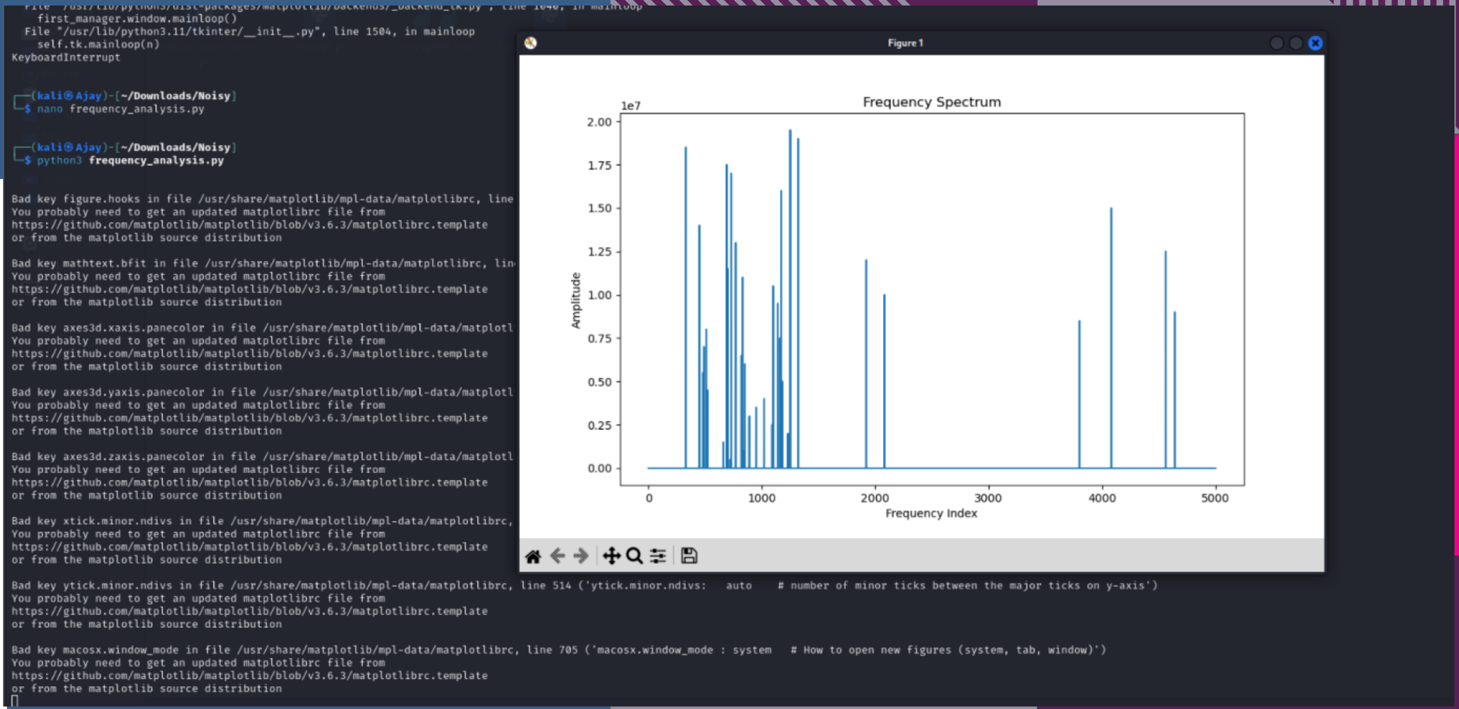
# VISUALIZING THE WAVEFORM

```
plt.figure(figsize=(10, 6))
plt.plot(data[:10000])  # Plot the first 10,000 samples for visualization
plt.title("Encrypted Waveform")
plt.xlabel("Sample Index")
plt.ylabel("Amplitude")
plt.show()
```



# PERFORMING FFT ON DATA

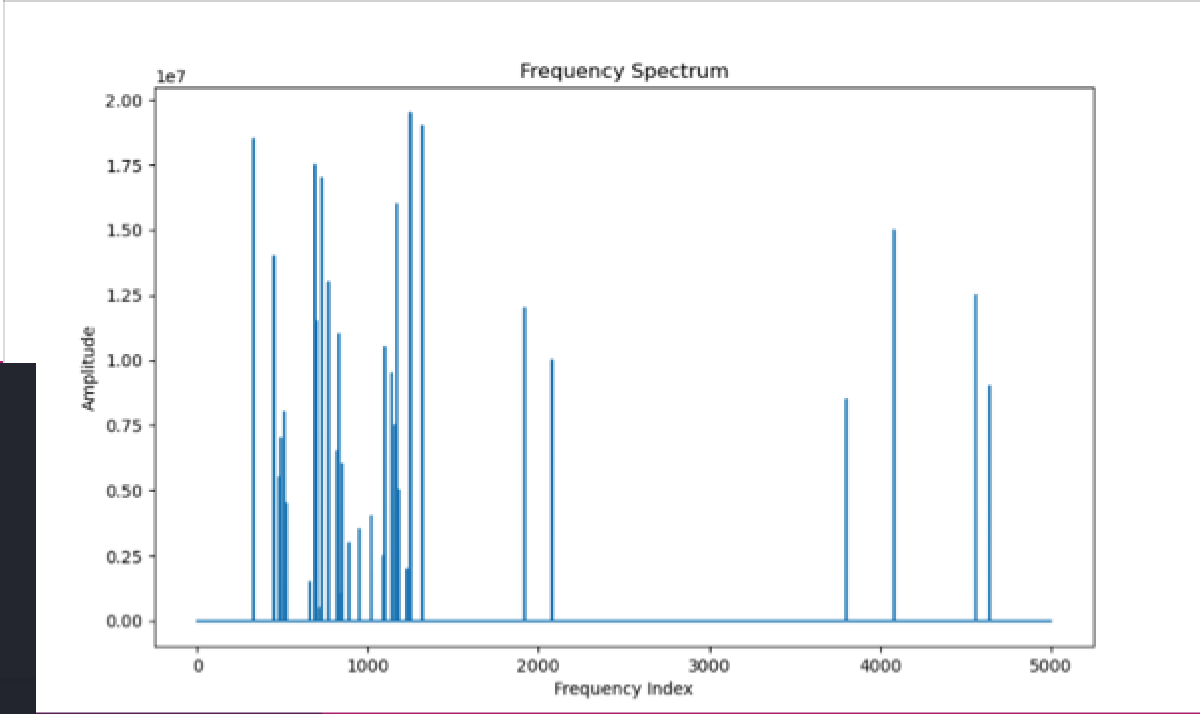
```
import numpy as np

from scipy.fftpack import fft

from scipy.io.wavfile import read


# Step 1: Load the encrypted WAV file

rate, data = read("encrypted.wav")


# Step 2: Perform FFT on the data

frequencies = np.abs(fft(data))
```



# VISUALIZING FREQUENCY SPECTRUM

```
plt.figure(figsize=(10, 6))

plt.plot(frequencies[:5000])  # Plot the first 5000 frequency components

plt.title("Frequency Spectrum")

plt.xlabel("Frequency Index")

plt.ylabel("Amplitude")

plt.show()
```

```
┌──(kali㉿Ajay)-[~/Downloads/Noisy]
└─$ nano frequency_analysis2.py

┌──(kali㉿Ajay)-[~/Downloads/Noisy]
└─$ python3 frequency_analysis2.py
Dominant Frequencies (Hz): [1.99750e+07 2.50000e+04 1.99736e+07 2.64000e+04 6.60000e+03 1.99934e+07
 1.45920e+06 1.85408e+07 1.38000e+04 1.99862e+07]
Corresponding Amplitudes: [19500000.0000001  19500000.0000001  18999999.9999999  18999999.9999999
 18499999.99999996 18499999.99999996 17999999.99999994 17999999.99999994
 17500000.00000085 17500000.00000085]

┌──(kali㉿Ajay)-[~/Downloads/Noisy]
└─$ 
```



# EXTRACTING DOMINANT FREQUENCIES

```
# Find the indices of the 10 most significant frequencies
dominant_indices = np.argsort(-frequencies)[:10]  # Top 10 frequencies
dominant_amplitudes = frequencies[dominant_indices]  # Get their amplitudes


# Convert indices to frequencies
dominant_frequencies = dominant_indices * rate / len(data)  # Frequency in Hz


# Print results
print("Dominant Frequencies (Hz):", dominant_frequencies)
print("Corresponding Amplitudes:", dominant_amplitudes)
```

# DECRYPTING THE MESSAGE

**Character Mapping Logic:**

1. **Frequency Formula**:

$$f = (i + 1) \cdot \sin(2\pi x \cdot \text{multiplier})$$

   - $i + 1$: Index of the character.
   - $\text{multiplier}$: Proportional to the character count.

2. **Decode Each Character**:
   - Divide the frequency by $\text{rate}/\text{len}(\text{data})$.
   - Map the result to characters.

# DISCRETE SINE TRANSFORM (DST):

The mathematical formula for DST of a sequence $x_n$ is:

$$X_k = \sum_{n=1}^{N} x_n \sin\left(\frac{\pi k(n+1)}{N+1}\right), \quad k = 1, 2, \ldots, N$$

Here:

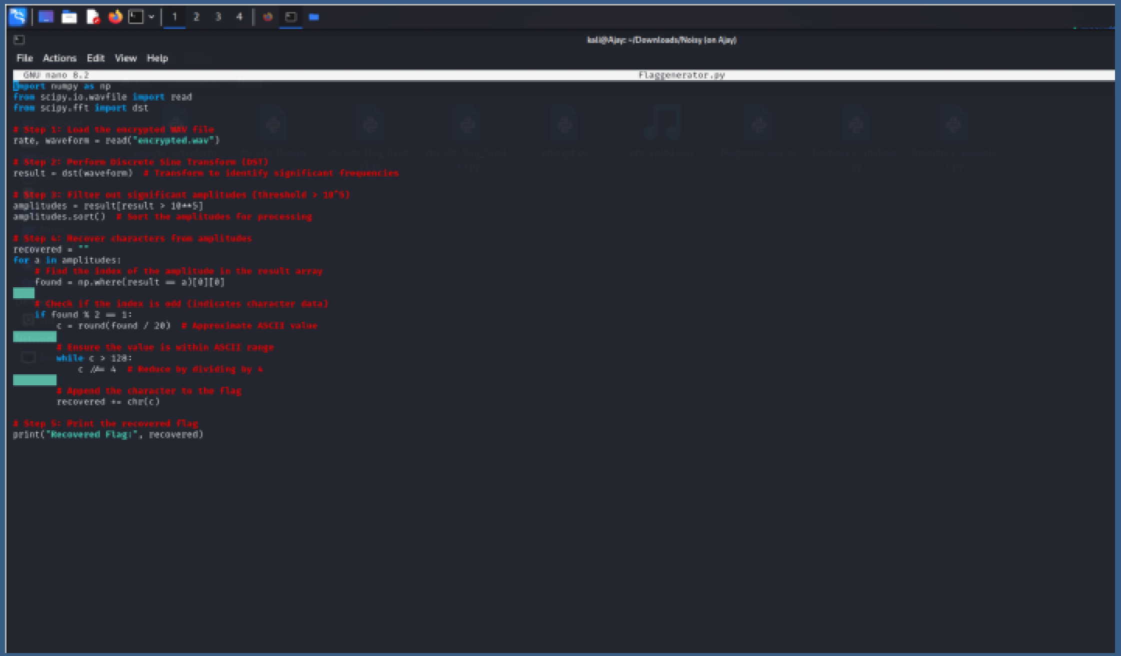
- $x_n$ is the input time-domain signal (amplitude values).
- $X_k$ is the frequency-domain representation.
- $N$ is the length of the input signal.

- dst(final_waveform[1]) performs a Discrete Sine Transform on the waveform data
- The DST is similar to the Discrete Fourier Transform (DFT) but uses sine functions instead of exponential functions. It converts the time-domain signal into its frequency-domain representation.
- The result, result, is a sequence of frequency-domain coefficients.

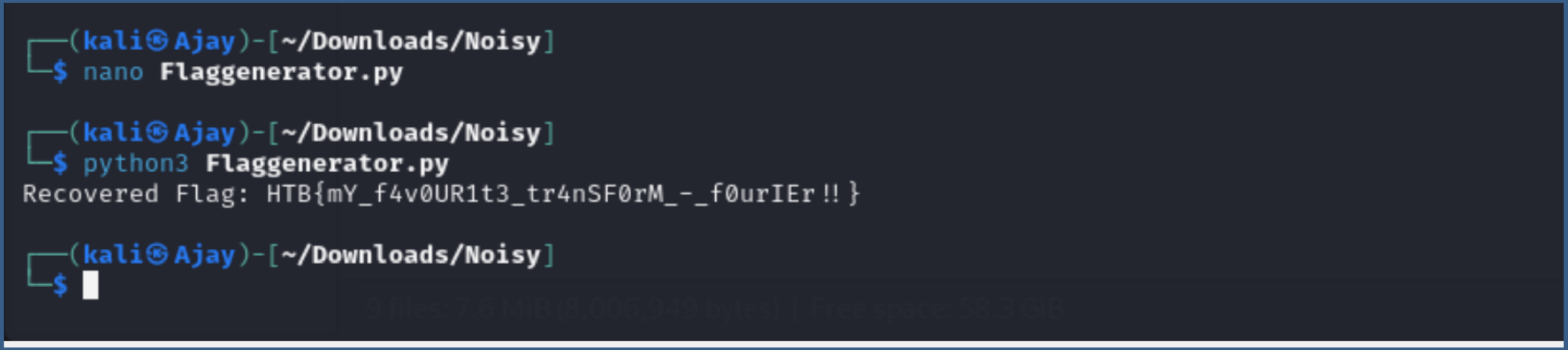
# CONCLUSION

- First we have loaded the waveform and then visualized it.
- We then have used fast fourier transform on the data.
- Now visualize the frequency spectrum and extract dominant frequency.
- Now map the result to characters and we finally will have the key.

# FLAG




HTB{mY_f4v0UR1t3_tr4nSF0rM_-_f0urIEr!!}



# THANK YOU

Group 7