**STEGANOGRAPHY TOOLS**

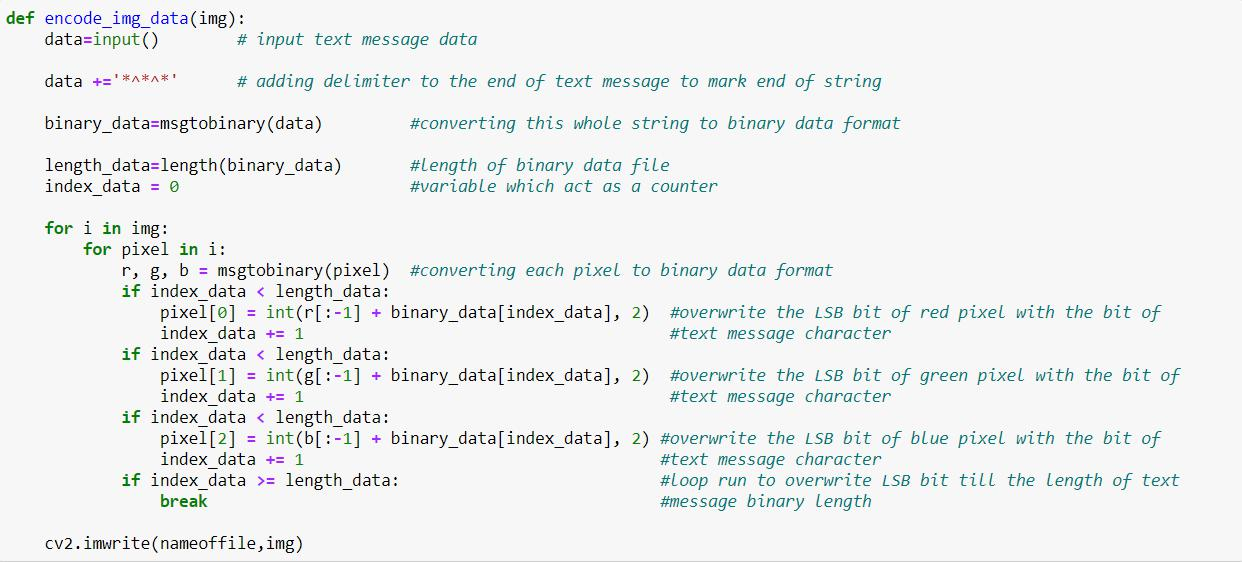
**ALGORITHM AND IMPLEMENTATION**

**Image Steganography**

ENCODE:-

Using **Modified LSB Algorithm** where we overwrite the LSB bit of actual image with the bit of text

message character. At the end of the text message we **push a delimiter to the message string** as a checkpoint useful in the decoding function. We encode data in order of Red, then Green and then Blue pixel for the entire message.



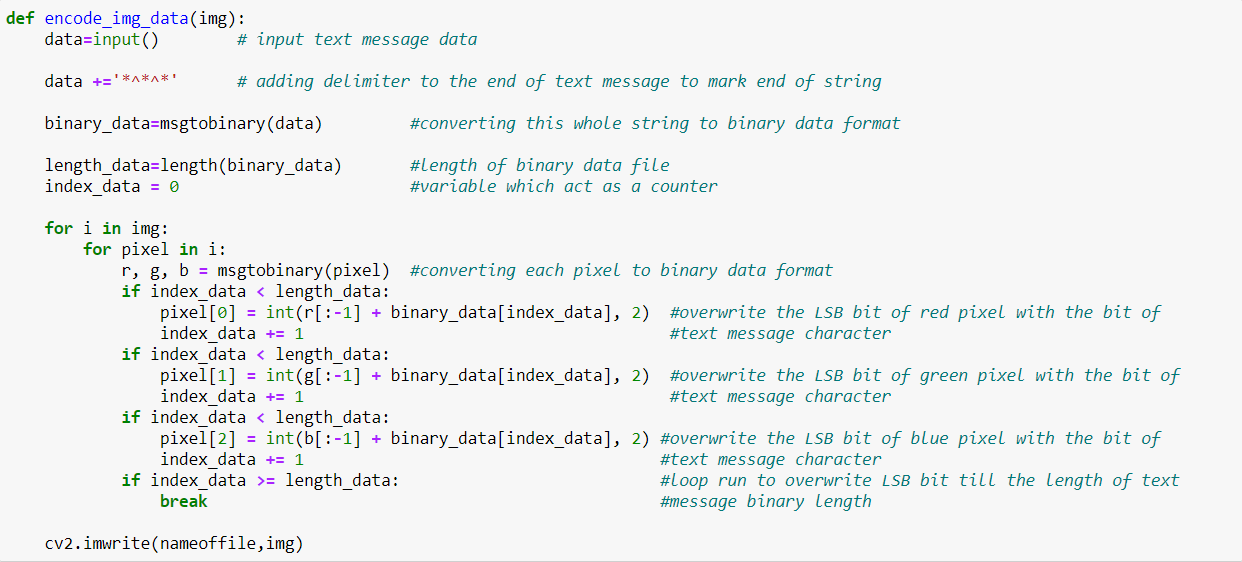


Image Steganography Encoding Algorithm

DECODE:-

In the decode part, we take **all the LSB bits of each pixel** until we get a checkpoint/delimiter and then **we split them by 8 bits** and convert them to characters data type and **print the string (i.e., the secret text message) without delimiter.**

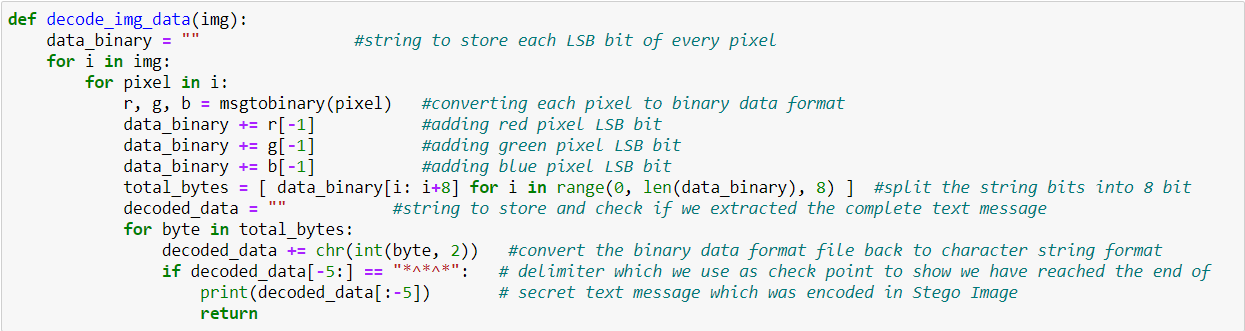


Image Steganography Decoding Algorithm

**Text Steganography**

ENCODE:-

ZWCs- In Unicode, there are specific zero-width characters (ZWC) that are used to control special entities such as Zero Width Non Joiner (e.g., ZWNJ separates two letters in special languages) and POP directional, which have no written symbol or width in digital text . In social media, if it utilizes the Unicode standard in order to process digital texts in different languages, then the ZWCs show invisible written symbols; otherwise, they might generate unconventional symbols . We used four ZWCs for hiding the Secret message through the cover text.

The embedding algorithm contains following stages:-

Secret Message (SM): This can be secret or confidential information.

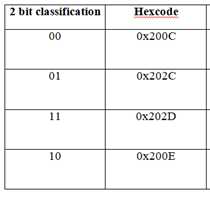
Cover Text (CT): This is an innocent text that can be any type of meaningful text.

For every character of the secret message :-

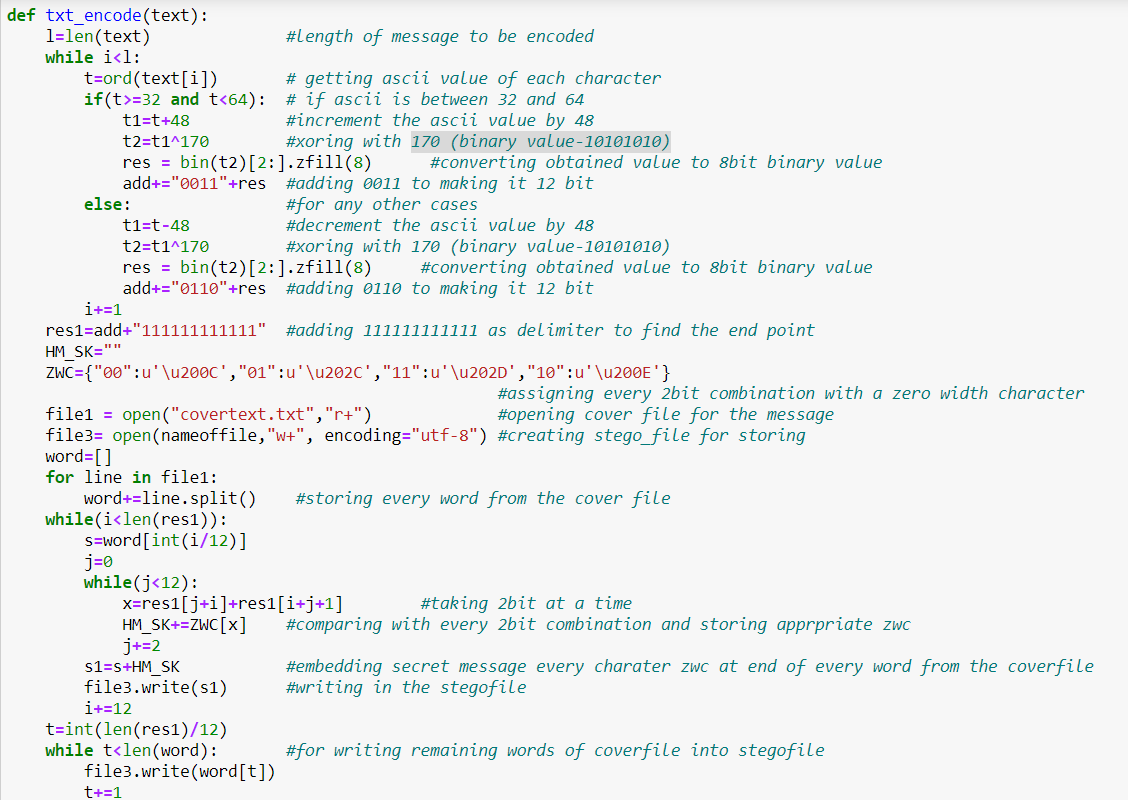
* We get its ascii value and it is incremented or decremented based on if ascii value between 32 and 64 , it is incremented by 48(ascii value for 0) else it is decremented by 48
* Then xor the the obtained value with 170(binary equivalent-10101010)
* Convert the obtained number from first two step to its binary equivalent then add "0011" if it earlier belonged to ascii value between 32 and 64 else add "0110" making it 12 bit for each character.

With the final binary equivalent we also 111111111111 as delimiter to find the end of message

Now from 12 bit representing each character every 2 bit is replaced with equivalent ZWCs according to the table. Each character is hidden after a word in the cover text.



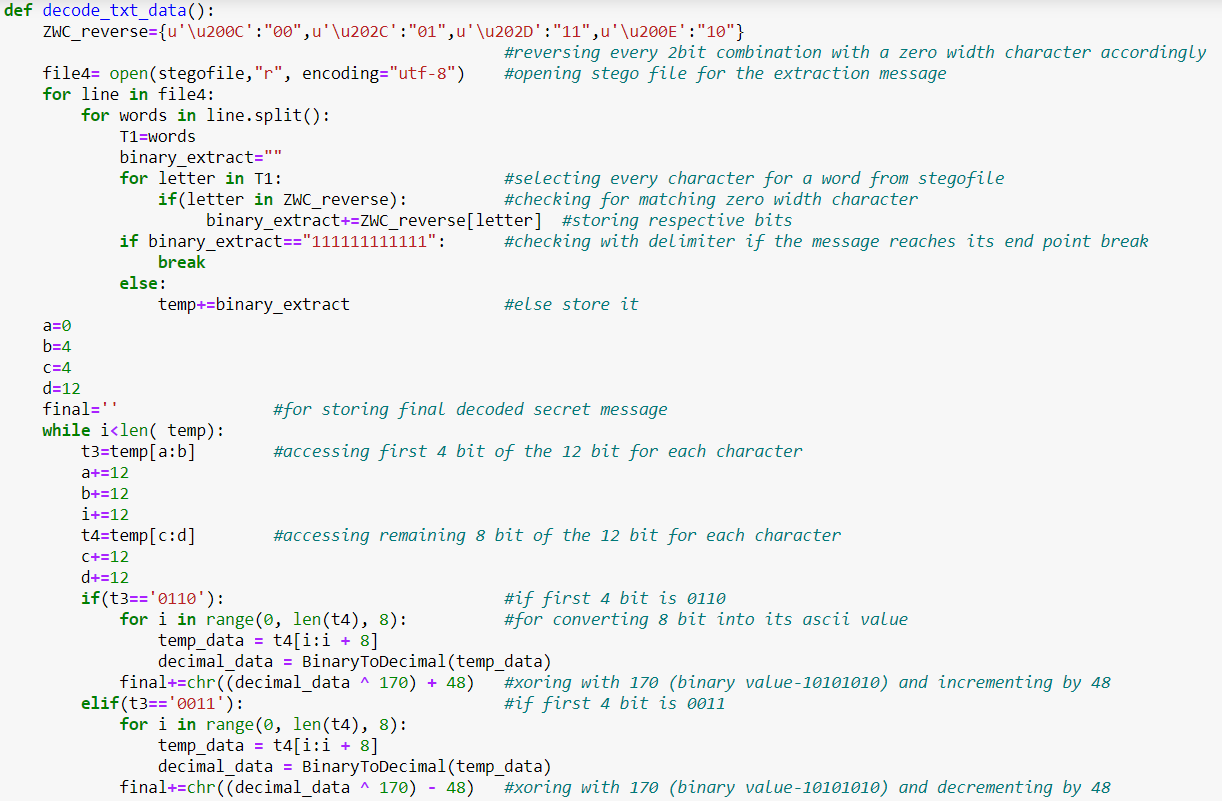
Zero Width Character Table



Text Steganography Encoding Algorithm

DECODE:-

After receiving a stegofile , the extraction algorithm discovers the contractual 2-bit of each ZWCs , every 12 bit from end of the word in the stego file and then the binary equivalent is completely extracted and the delimiter discussed above helps us in getting to the end point. Now we divide the 12 bit into two parts first 4 bit and another 8bit on which we do the xor operation with 170(binary value 10101010). Now according to the first 4bit if its "0110" we increment it by 48 else we decrement by 48. At last we convert the ascii value into its equivalent character to get the final hidden message from the stego file.

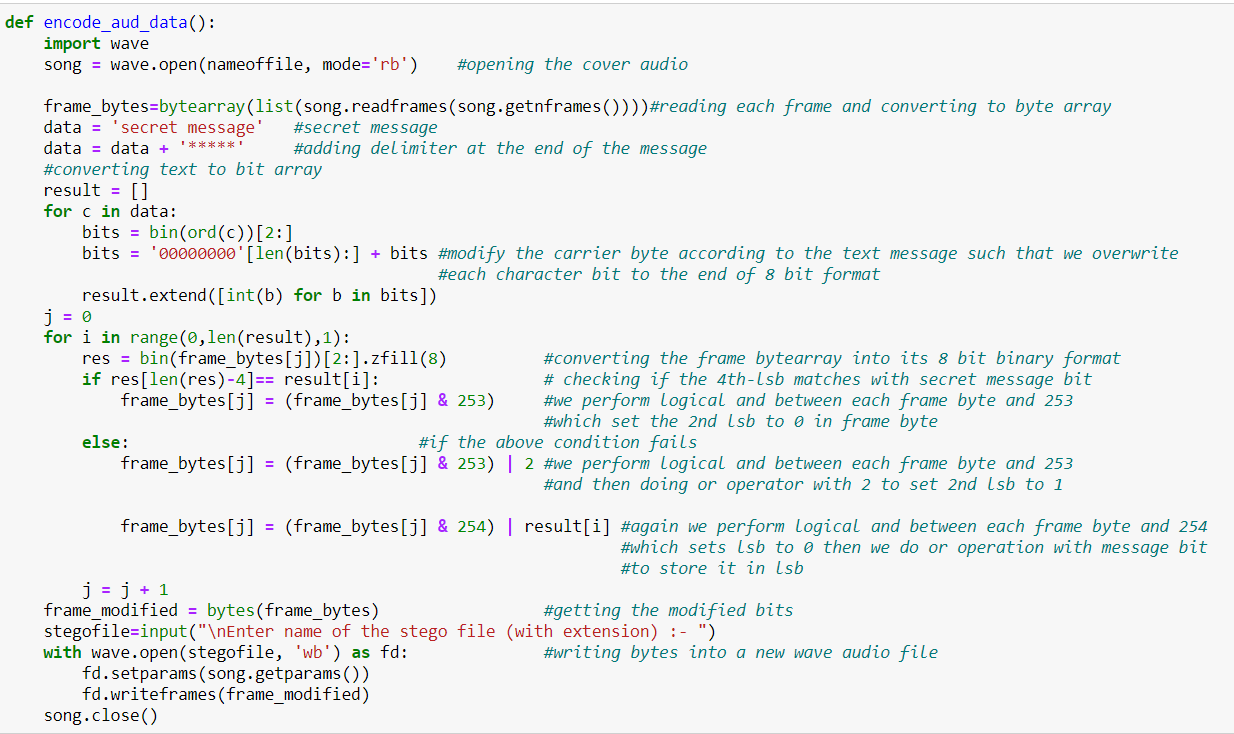


Text Steganography Decoding Algorithm

**Audio Steganography**

ENCODE:-

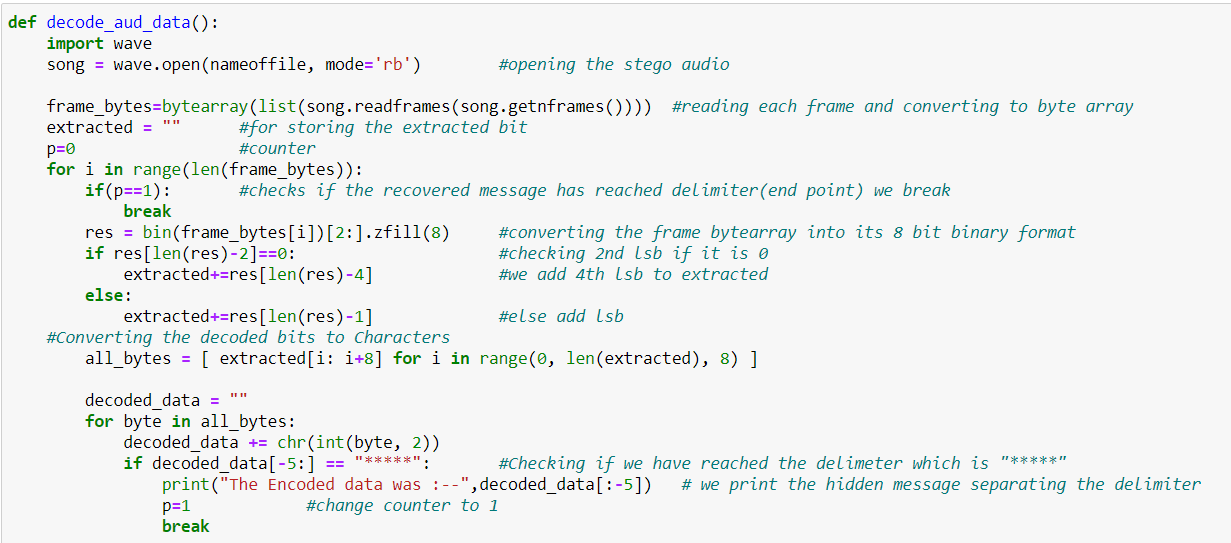
We will be using Cover Audio as a Cover file to encode the given text. Wave module is used to read the audio file. Firstly we convert our secret message to its binary equivalent and added delimiter '\*\*\*\*\*' to the end of the message. For encoding we have modified the LSB Algorithm, for that we take each frame byte of the converting it to 8 bit format then check for the 4th LSB and see if it matches with the secret message bit. If yes, change the 2nd LSB to 0 using the logical AND operator between each frame byte and 253(11111101). Else we change the 2nd LSB to 1 using logical AND operation with 253 and then logical OR to change it to 1 and now add secret message bit in LSB for achieving that use logical AND operation between each frame byte of carrier audio and a binary number of 254 (11111110). Then logical OR operation between modified carrier byte and the next bit (0 or 1) from the secret message which resets the LSB of the carrier byte.

****

Audio Steganography Encoding Algorithm

DECODE:-

We start the extraction process by reading each frame and converting it to a byte array. After that we check the 2nd LSB if it is 0 or 1. If the bit is 1 we store the LSB of the frame byte else we store the 4th LSB, we keep this process until we reach the delimiter and then we break from the loop, then convert the message into characters and print it.

****

Audio Steganography Decoding Algorithm

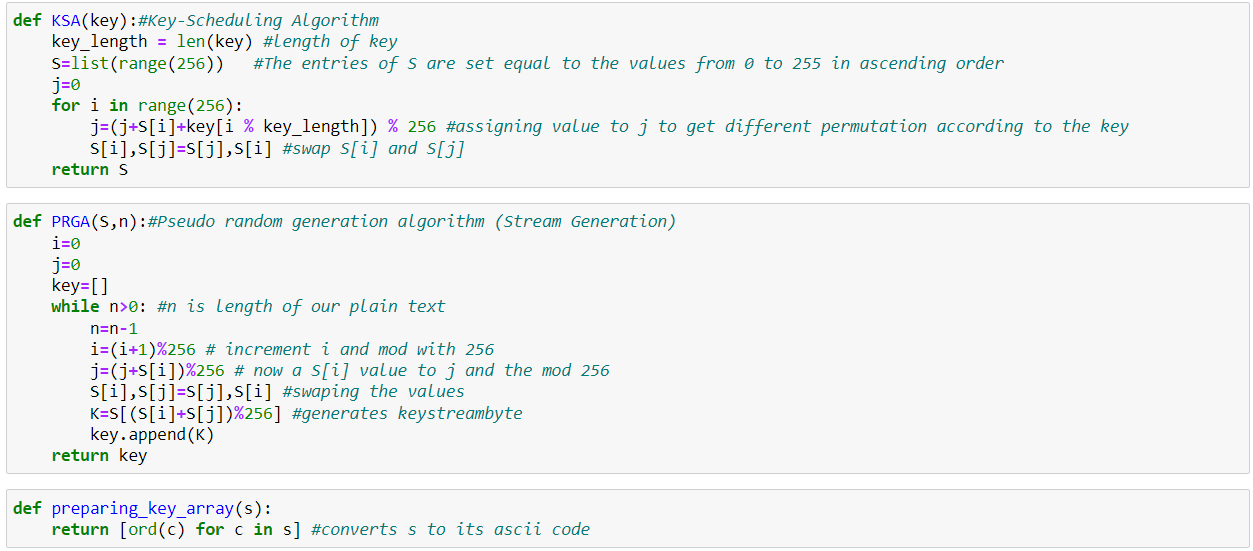
**Video Steganography**

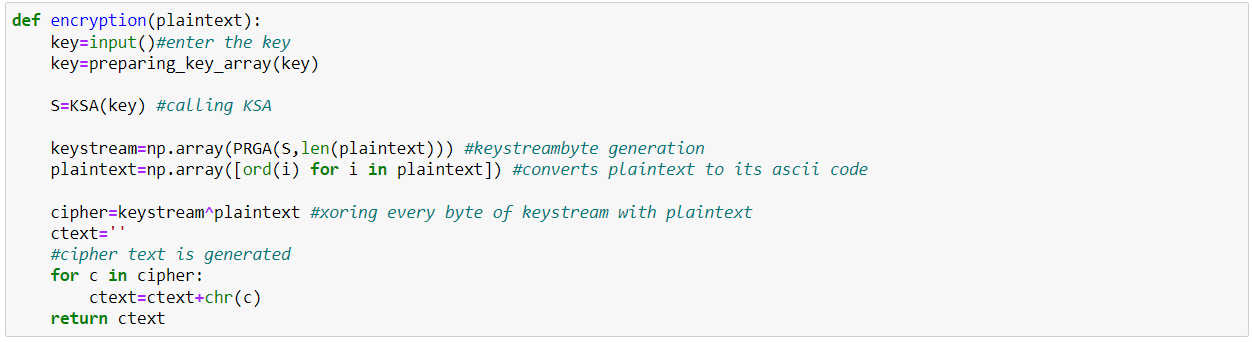
ENCODE:-

In video steganography we have used a combination of cryptography and Steganography. We encode the message through two parts:-

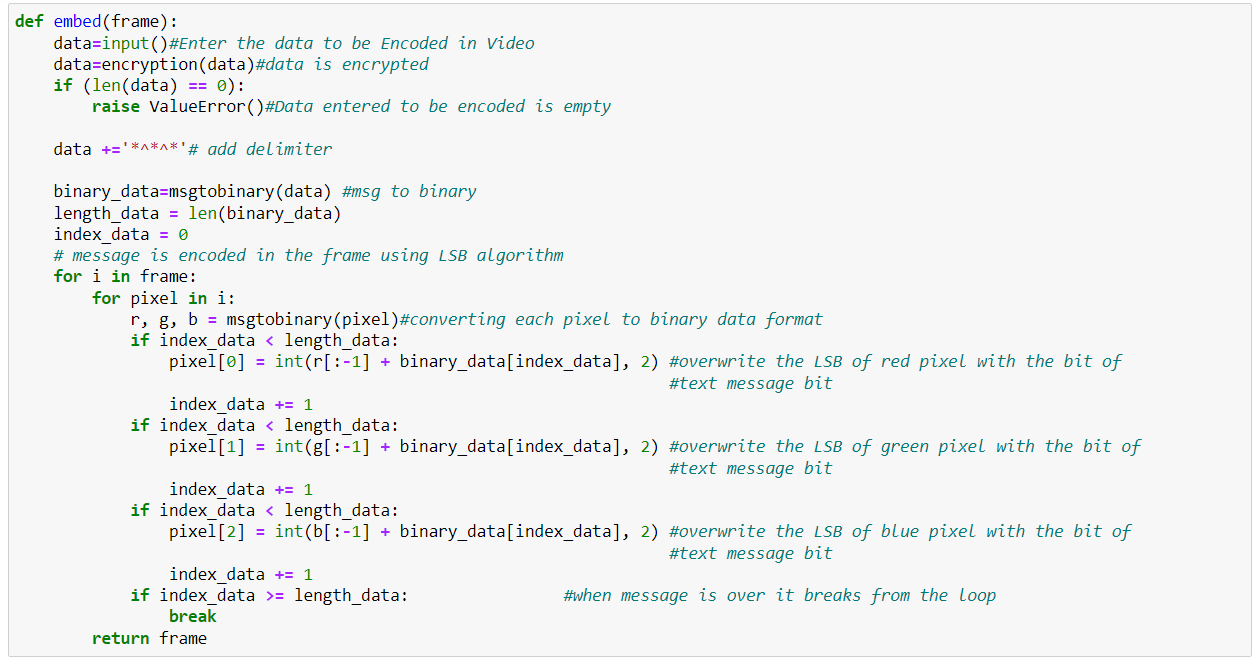
* We convert plaintext to cipher text for doing so we have used RC4 Encryption Algorithm. RC4 is a stream cipher and variable-length key algorithm. This algorithm encrypts one byte at a time. It has two major parts for encryption and decryption:-
* KSA(Key-Scheduling Algorithm)- A list S of length 256 is made and the entries of S are set equal to the values from 0 to 255 in ascending order. We ask the user for a key and convert it to its equivalent ascii code. S[] is a permutation of 0,1,2....255, now a variable j is assigned as j=(j+S[i]+key[i%key\_length) mod 256 and swap S(i) with S(j) and accordingly we get new permutation for the whole keystream according to the key.
* PRGA(Pseudo random generation Algorithm (Stream Generation)) - Now we take input length of plaintext and initiate a loop to generate a keystream byte of equal length. For this we initiate i=0, j=0 now increment i by 1 and mod with 256. Now we add S[i] to j and mod it with 256 ,again swap the values. At the last step take store keystreambytes which matches as S[(S[i]+S[j]) mod 256] to finally get a key stream of length same as plaintext.

Now we xor the plaintext with keystream to get the final cipher.





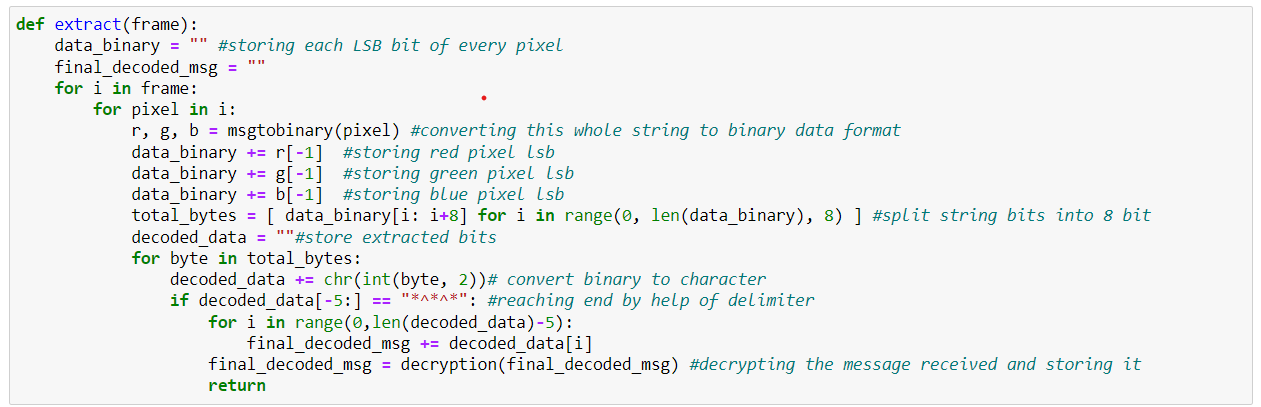
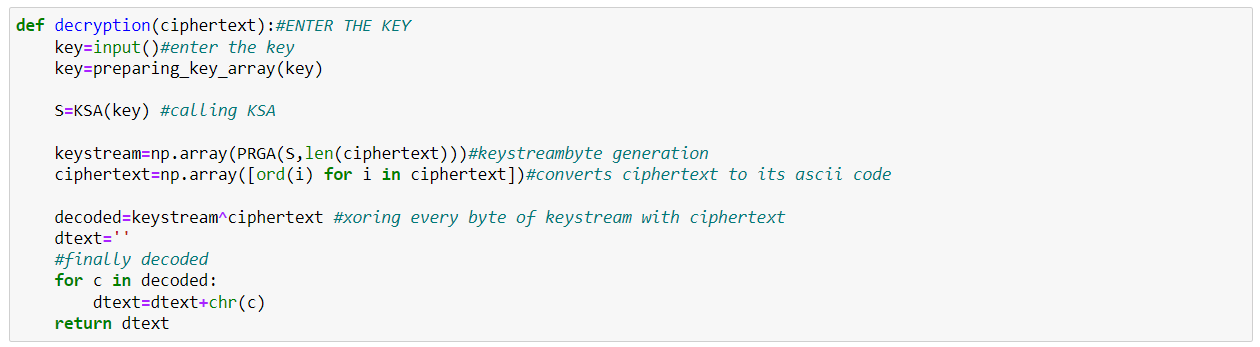
* Now for the Steganography part we will be using Modified LSB Algorithm where we overwrite the LSB bits of the selected frame (given by the user) from the cover video, with the bit of text message character. At the end of the text message we add a delimiter to the message string as an endpoint which comes useful in the decoding function. We encode data in order of Red, then Green and then Blue pixel for the entire message of the selected frame.



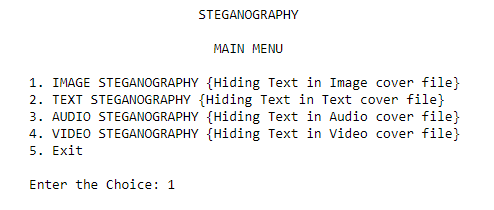
Video Steganography Encoding Algorithm

DECODE:-

In decode part In the decode part, we take the encoded frame from the stego video, in the frame each pixels last LSB is stored until we get to the delimiter as we reach there we split them by 8 bits and convert them to characters data type now we go to the decryption process where we do the same as encode, make Keystream with help of secret key and using KSA and PRGA and finally xoring with the obtained data from the frame with keystream to get the final decoded secret message

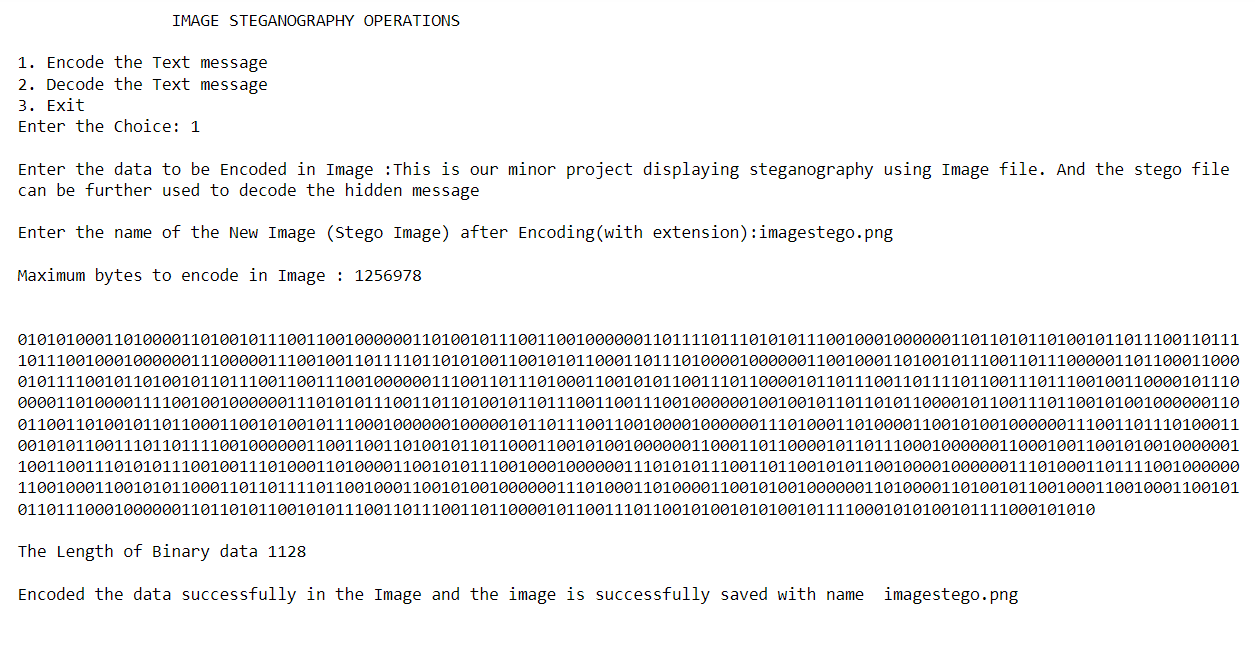


**RESULTS AND OUTPUTS**

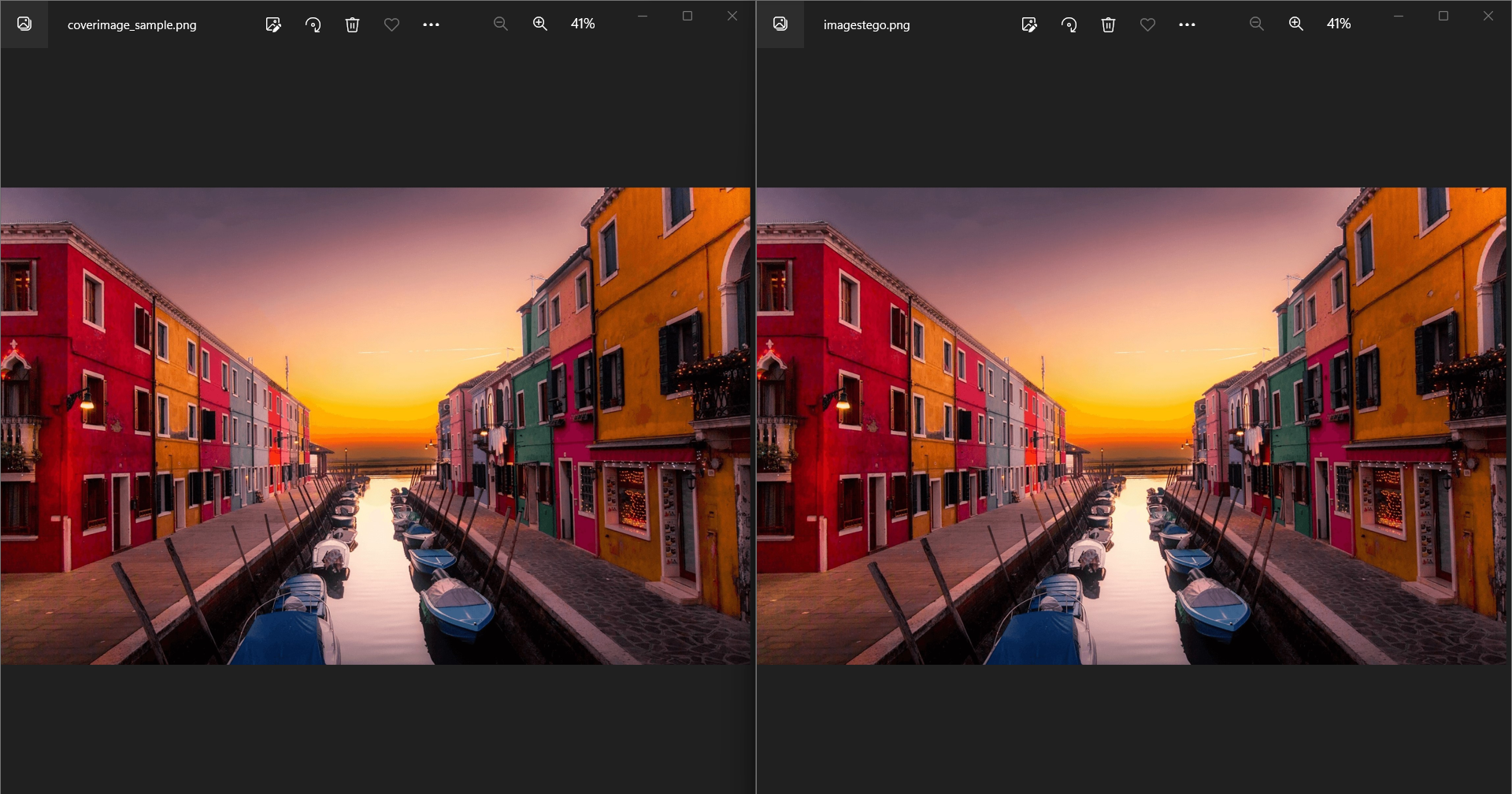


**Image Steganography**

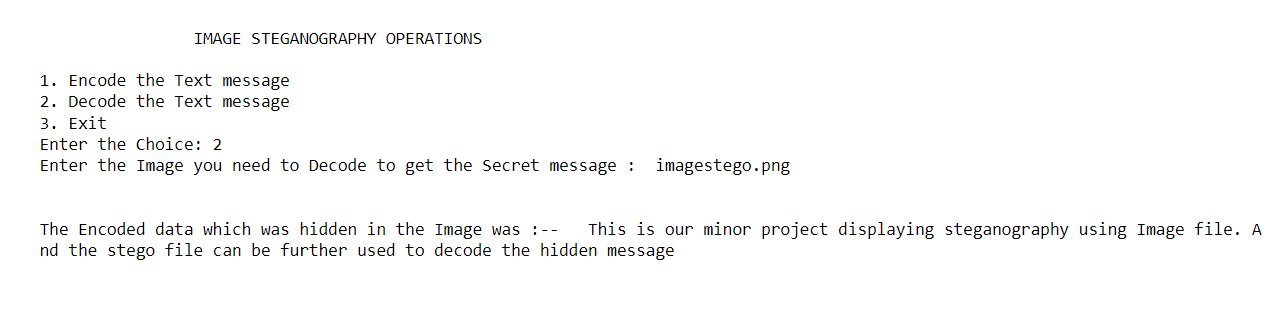
* Encoding the message in image file



* Preview of Cover image and Stego image:

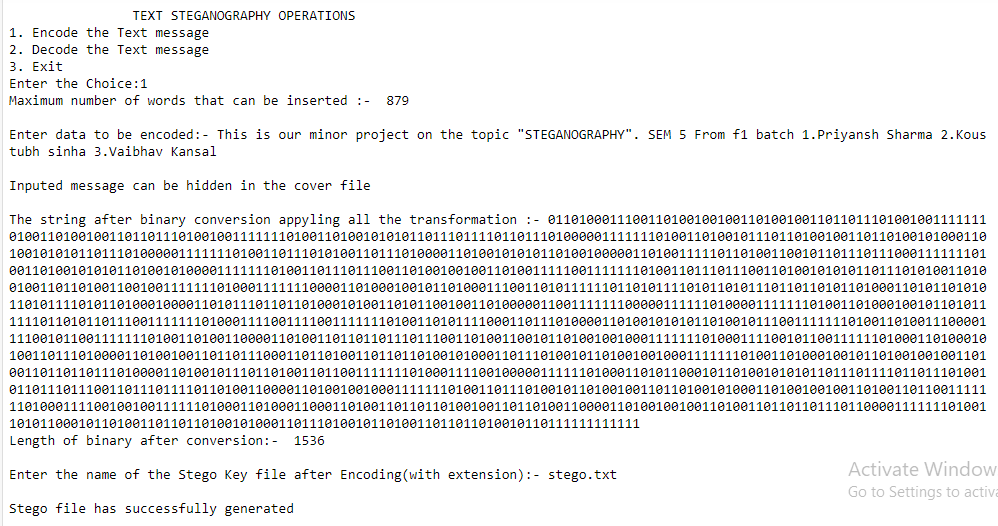


* Decoding the message from Image file:

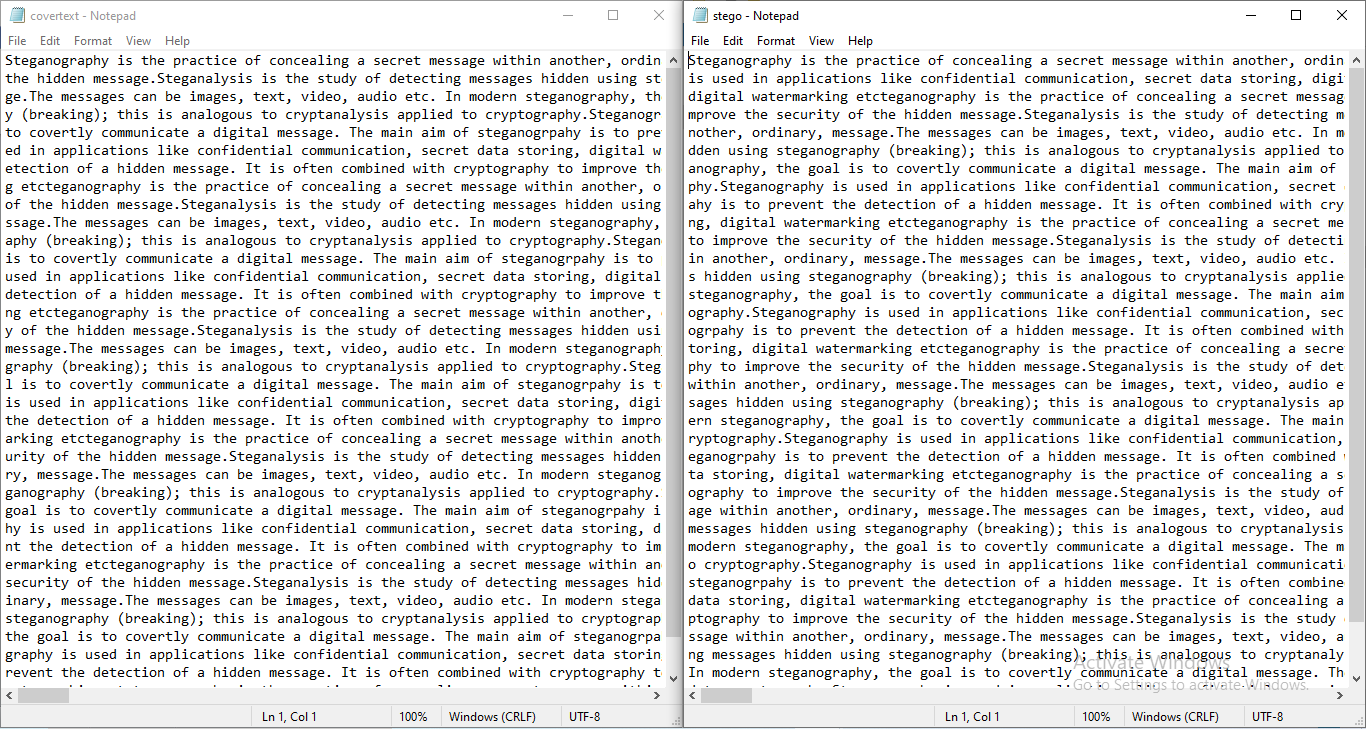


**Text Steganography**

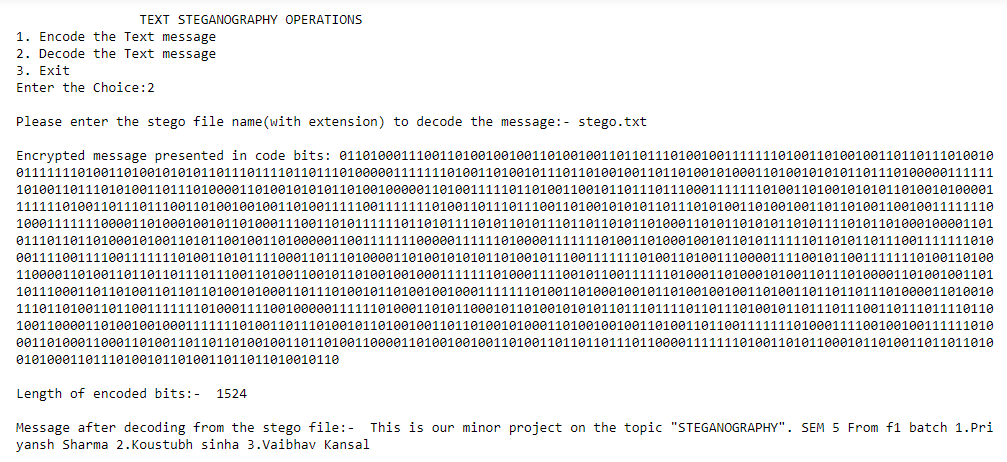
* Encoding the message in text file:



* Preview of Cover Text file and Stego Text file:

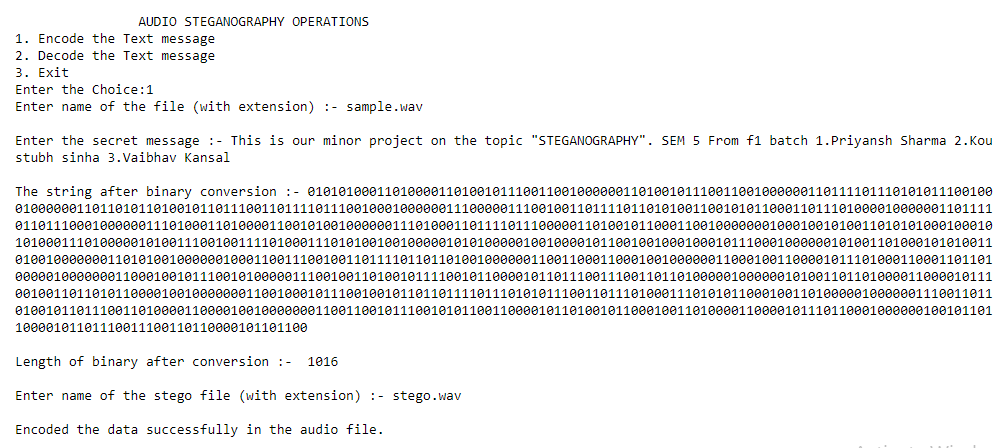


* Decoding the message from Text file:

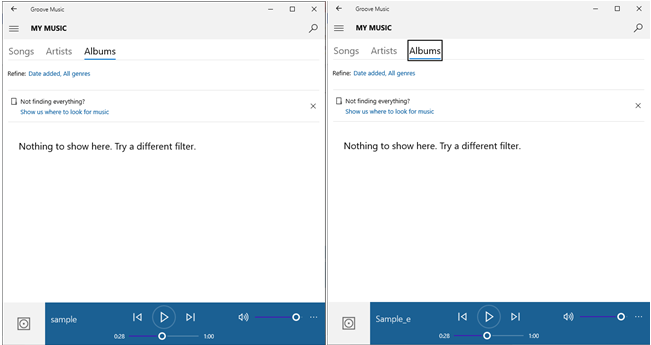


**Audio Steganography:**

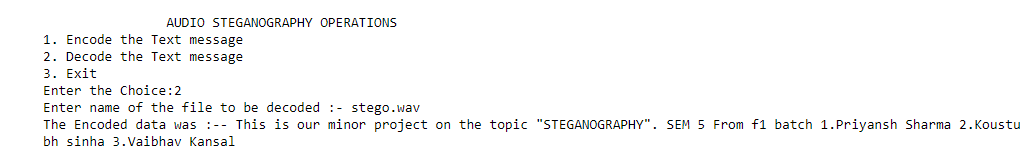
* Encoding the message in Audio file:



* Preview of Cover Audio file and Stego Audio file:

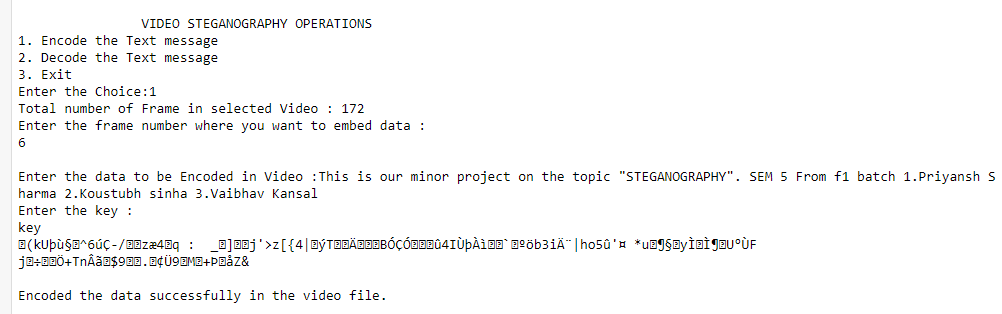
****

* Decoding the message from Audio file:

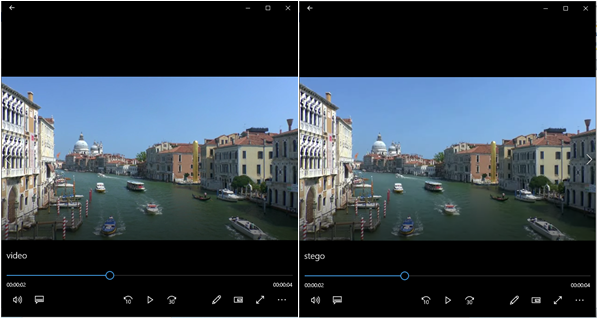


**Video Steganography:**

* Encoding the message in Video file:



* Preview of Cover Video file and Stego Video file:



* Decoding the message from Video file:

