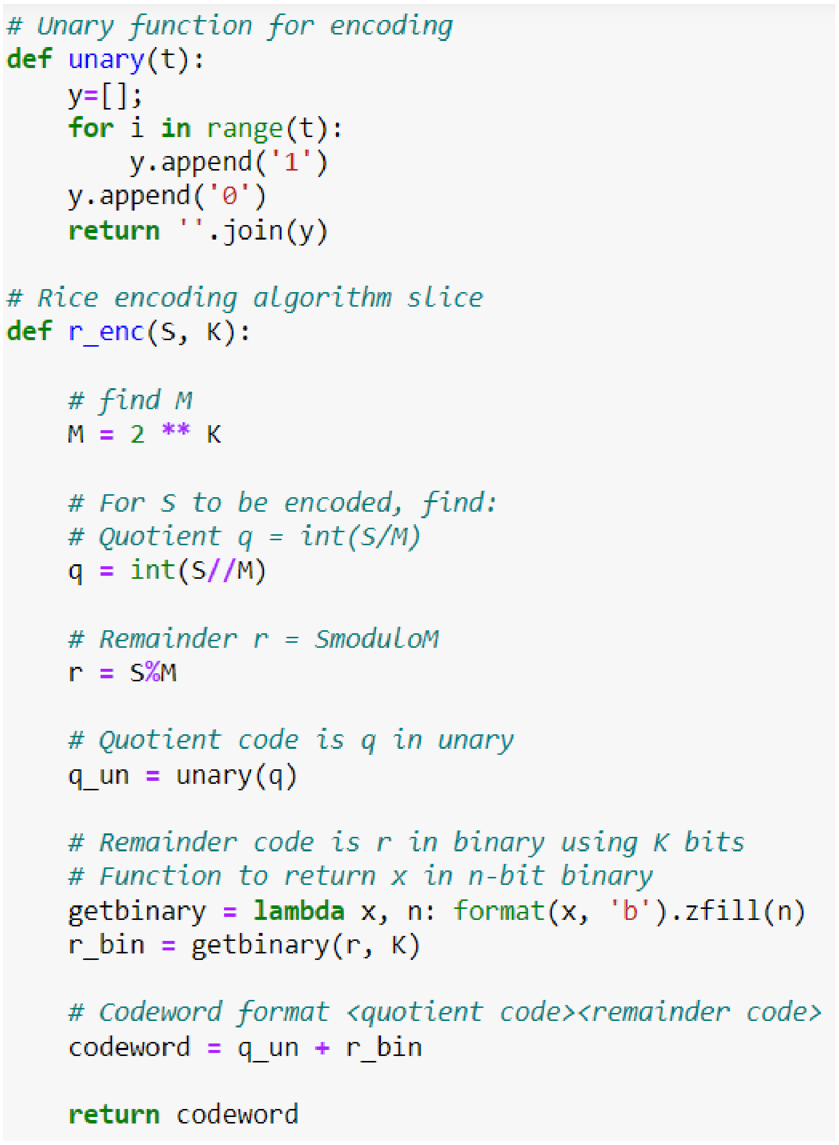
Exercise 2

Introduction

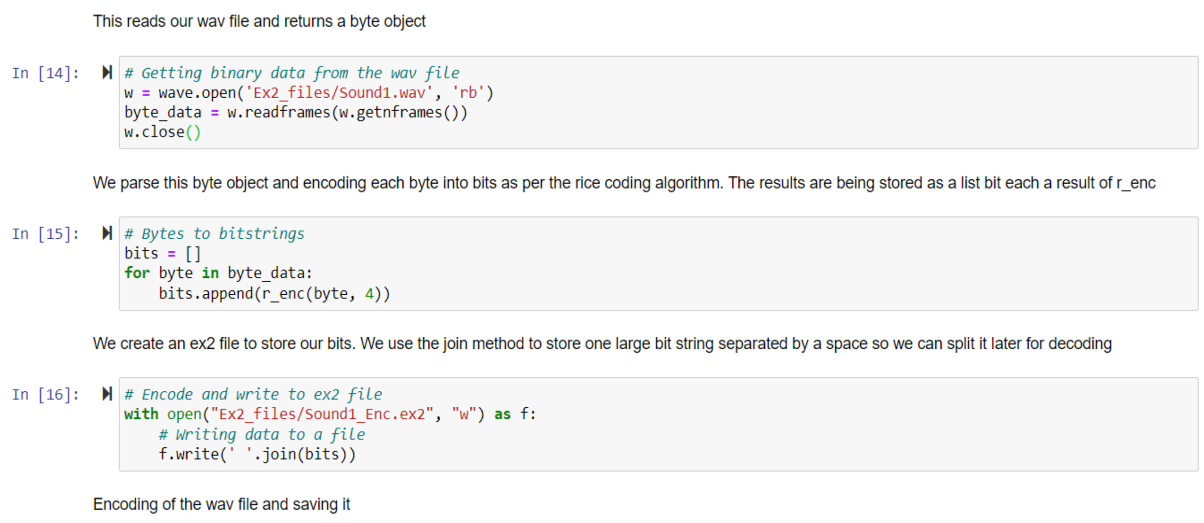
The program reads a wav file as a byte object, where each byte is an integer ranging from 0-255. The rice encoding algorithm, which I implemented, is then used to compress the file. The compressed file is written to an ex2 file in text format. To restore the original file, the program uses the same ex2 file and a decoding algorithm.

In the first cell of the program, after importing the required modules, I define the rice encoding algorithm. This algorithm is based on a previous exercise. To perform rice encoding, the r_enc function takes an integer S as input and returns an encoded string of binary, using K as a parameter and a unary helper function.



```python
# Unary function for encoding
def unary(t):
    y=[];
    for i in range(t):
        y.append('1')
    y.append('0')
    return ''.join(y)

# Rice encoding algorithm slice
def r_enc(S, K):

    # find M
    M = 2 ** K

    # For S to be encoded, find:
    # Quotient q = int(S/M)
    q = int(S//M)

    # Remainder r = SmoduloM
    r = S%M

    # Quotient code is q in unary
    q_un = unary(q)

    # Remainder code is r in binary using K bits
    # Function to return x in n-bit binary
    getbinary = lambda x, n: format(x, 'b').zfill(n)
    r_bin = getbinary(r, K)

    # Codeword format <quotient code><remainder code>
    codeword = q_un + r_bin

    return codeword
```

We then open the wav file and read it as a byte object and store it within the byte_data and then compress each byte using the rice encoder after specifying a K value and making a list of strings named ‘bits’.



This reads our wav file and returns a byte object

```
In [14]:  # Getting binary data from the wav file
          w = wave.open('Ex2_files/Sound1.wav', 'rb')
          byte_data = w.readframes(w.getnframes())
          w.close()
```

We parse this byte object and encoding each byte into bits as per the rice coding algorithm. The results are being stored as a list bit each a result of r_enc

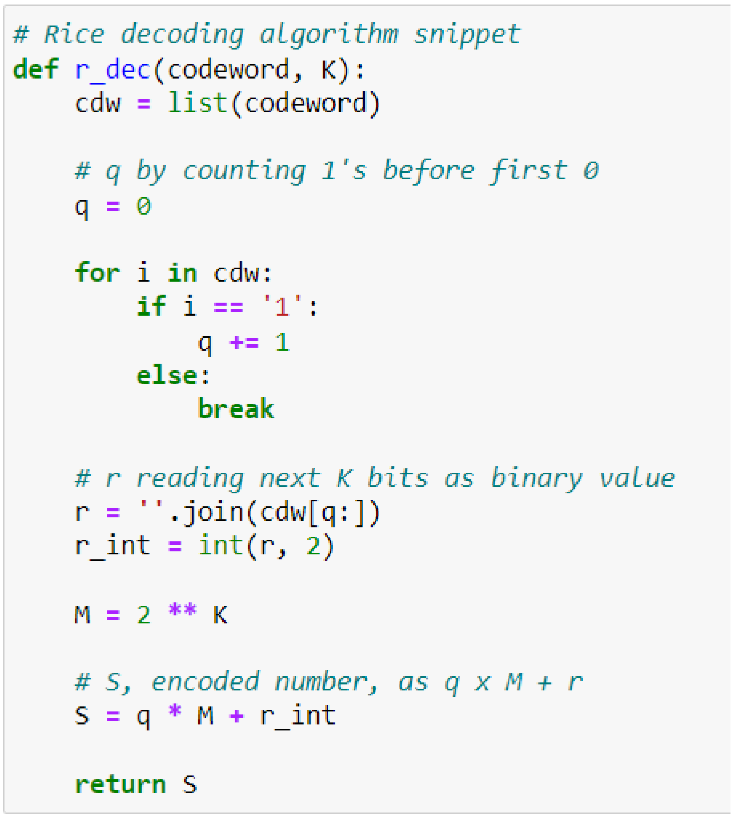
```
In [15]:  # Bytes to bitstrings
          bits = []
          for byte in byte_data:
              bits.append(r_enc(byte, 4))
```

We create an ex2 file to store our bits. We use the join method to store one large bit string separated by a space so we can split it later for decoding

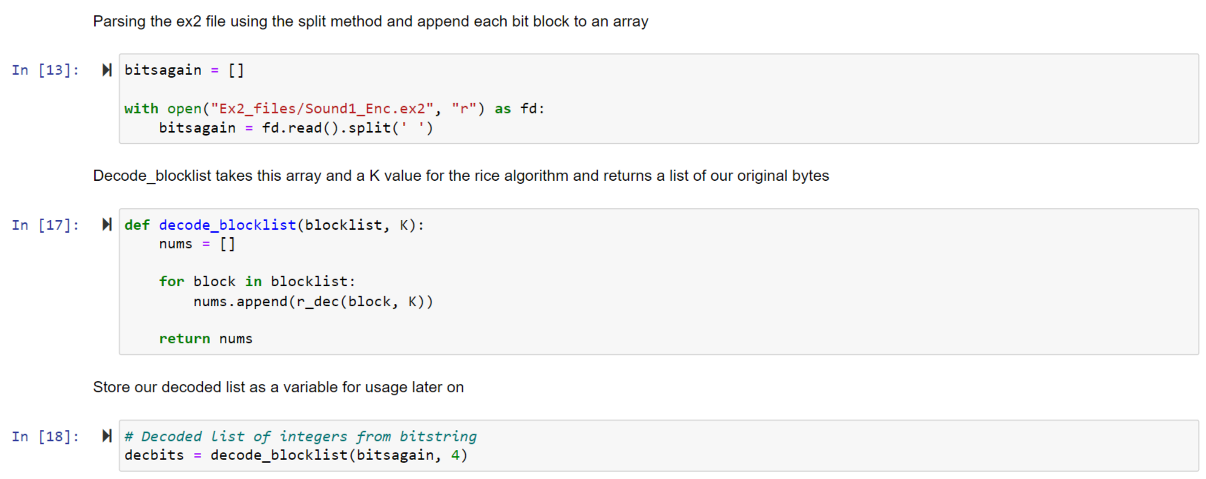
```
In [16]:  # Encode and write to ex2 file
          with open("Ex2_files/Sound1_Enc.ex2", "w") as f:
              # Writing data to a file
              f.write(' '.join(bits))
```

Encoding of the wav file and saving it

The r_dec function then takes the codeword and K value to return the decoded integer.



```
# Rice decoding algorithm snippet
def r_dec(codeword, K):
    cdw = list(codeword)

    # q by counting 1's before first 0
    q = 0

    for i in cdw:
        if i == '1':
            q += 1
        else:
            break

    # r reading next K bits as binary value
    r = ''.join(cdw[q:])
    r_int = int(r, 2)

    M = 2 ** K

    # S, encoded number, as q x M + r
    S = q * M + r_int

    return S
```



Parsing the ex2 file using the split method and append each bit block to an array

```
In [13]:  bitsagain = []

          with open("Ex2_files/Sound1_Enc.ex2", "r") as fd:
              bitsagain = fd.read().split(' ')
```

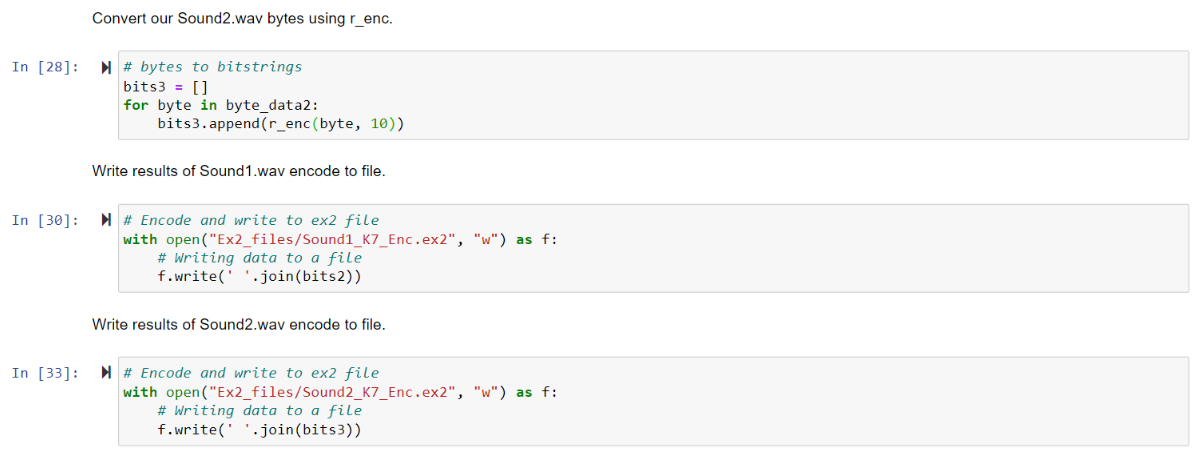
Decode_blocklist takes this array and a K value for the rice algorithm and returns a list of our original bytes

```
In [17]:  def decode_blocklist(blocklist, K):
              nums = []

              for block in blocklist:
                  nums.append(r_dec(block, K))

              return nums
```

Store our decoded list as a variable for usage later on

```
In [18]:  # Decoded list of integers from bitstring
          decbits = decode_blocklist(bitsagain, 4)
```

We then read our encoded file again and defining and using the decode_blocklist and r_dec functions to get the list of integers again. We then convert those integers back to bytes and we then obtain what appears to be our original byte object.



Convert our Sound2.wav bytes using r_enc.

```
In [28]:  # bytes to bitstrings
          bits3 = []
          for byte in byte_data2:
              bits3.append(r_enc(byte, 10))
```

Write results of Sound1.wav encode to file.

```
In [30]:  # Encode and write to ex2 file
          with open("Ex2_files/Sound1_K7_Enc.ex2", "w") as f:
              # Writing data to a file
              f.write(' '.join(bits2))
```

Write results of Sound2.wav encode to file.

```
In [33]:  # Encode and write to ex2 file
          with open("Ex2_files/Sound2_K7_Enc.ex2", "w") as f:
              # Writing data to a file
              f.write(' '.join(bits3))
```

We then check the equivalence of the 1st wav object called ‘byte_data’ and the bytes_again object. We then write it back to the wav file with the enc_dec extension.

The encoded files can then be found in the Ex2_file folder when required. Results of the compression with 2 and 4 for values of K are as follows:

	Initial size	Rice (K = 2)	Rice (K = 4)	Compression (K = 2)	Compression (K = 4)
Sound1.wav	1 MB	13.2 MB	33.8 MB	-1220%	-3280%
Sound2.wav	1.01 MB	13.4 MB	35.0 MB	-1226%	-3365%

The resulting compressions took up more space, as the rice encoding algorithm uses more space to store binary values for larger numbers. It becomes especially inefficient for K=2

Coursera Link:

https://hub.labs.coursera.org:443/connect/sharedlycoqram?forceRefresh=false