In [24]:
```
# module : to interact with the operating system
import os

# module :  for high level operations on files and collection of fi
les
# helps in automating the copying and removal of files and director
ies
import shutil

# making the setup
def setup():
    # directory to store intermediate frames to make video file
    inframes = './inframes'

    # directory to disseminate video file into frames at a specific
rate
    outframes = './outframes'

    # if inframes does not exist, create it
    if not os.path.exists(inframes):
        os.makedirs(inframes)

    # if outframes does not exist, create it
    if not os.path.exists(outframes):
        os.makedirs(outframes)

#     # delete previous content of inframes, if there
#     shutil.rmtree(inframes)

#     # delete previous content of outframe, if there
#     shutil.rmtree(outframes)
```

In [15]:
```
# module : gzip support for files
import gzip

# module :  for high level operations on files and collection of fi
les
# helps in automating the copying and removal of files and director
ies
import shutil

# contextlib module :  provides utilities for common tasks involvin
g the 'with' statement
from contextlib import ExitStack

# module : to interact with the operating system
import os

# function : compress a file using gzip compression
def compress_file(filepath):
#   os.path.basename() : removes the leading path information of th
e file and leaves only
#   with the actual filename from the complete path
#   /home/user/file.txt -> file.txt
    print("Compressing " + os.path.basename(filepath) + "...")

#   creates a stack of files so that we can define operations one a
fter the other
#   helpful in removing nested 'with' statements
    with ExitStack() as stack:
        f_in = stack.enter_context(open(filepath, 'rb'))
        f_out = stack.enter_context(gzip.open(filepath + ".gz", 'wb
'))
        shutil.copyfileobj(f_in, f_out)

#       deleting unused objects
        del f_in
        del f_out

    print("Successfully compressed " + os.path.basename(filepath))
```

In [16]:
```
# bitstring module :  for the creation and analysis of binary data
from bitstring import BitArray

# function : get binary representation of a file
def get_bitarray(filepath):
    print("Converting " + os.path.basename(filepath) + " to binary
form...")

#   stores the hexdump of the file in a bitstring.BitArray object
    bitarray = BitArray(filename = filepath)
#     print("Type: " ,type(bitarray))
#     print("Hexdump: ", bitarray)

    print("Successfully converted " + os.path.basename(filepath) +
" to binary form")

#   returns the binary dump of file
#     print("Bindump: ", bitarray.bin)
    return bitarray.bin
```

In [17]:
```
# module : python imaging library : support for image processing
from PIL import Image

# module : numerical python : fast and efficient processing for arr
ays
import numpy as np

# function : generate frames from binstring.BitArray object
def generate_frames(bitarray):
#   RESOLUTION = (HEIGHT, WIDTH) : resolution of the video
    RESOLUTION = (480, 854)

    print("Generating frames...")

    index = 0
    frame_num = 0
    while(index < len(bitarray)):
#       generating a numpy array with the bitarray[index : index +
resolution] slice
#       with data type as int
        pixels = np.fromiter(bitarray[index : index + (RESOLUTION
[0] * RESOLUTION[1])], dtype = np.int)

#       creating a new instance of 1-bit pixel image with the speci
fied resolution and
#       with 1 pixel per byte. tuple denotes (width, height)
        image = Image.new("1", (RESOLUTION[1], RESOLUTION[0]))

        image.putdata(pixels)
        image.save("./inframes/" + "frame_" + str(frame_num) + ".pn
g")
#         print("Generated frame: " + str(frame_num))

        del pixels
        del image
        frame_num += 1
        index += (RESOLUTION[0] * RESOLUTION[1])

    print("Successfully generated all frames")
```

In [18]:
```
# module : video processing support for python
import ffmpeg

# function : generate video file from frames
def generate_video():
#     FRAMERATE = 24
    print("Generating video file...")

    (
        ffmpeg
        .input('./inframes/frame_%d.png')
        .filter('fps', fps=25, round='up')
        .output('output.mp4')
        .run()
    )
#     os.system('ffmpeg -framerate 24 -i ./inframes/frame_%d.png ou
tput.mp4')

    print("Successfully generated video file")
```

In [19]:
```
import ffmpeg

def convert_video_to_frames(videopath):
    print("Converting video file to respective frames...")

    (
        ffmpeg
        .input('output.mp4')
        .filter('fps', fps=25, round='down')
        .output('./outframes/frame_%d.png')
        .run()
    )
#     os.system('ffmpeg -i ' + videopath + ' -r 24 ./outframes/fram
e_%d.png')

    print("Successfully generated all frames")
```

In [20]:
```
def convert_image_to_bits(imagepath):
    image = Image.open(imagepath)
    width, height = image.size
    bits = ""
    pixels = image.load()
    del image

    for j in range(height):
        for i in range(width):
            pixel = pixels[i, j]
            pixel_bin_rep = "0"

#           if white difference is smaller then black difference, t
hen
#           pixel_bin_rep must be "1"
            if (abs(pixel[0] - 255) < abs(pixel[0] - 0)
            and abs(pixel[1] - 255) < abs(pixel[1] - 0)
            and abs(pixel[2] - 255) < abs(pixel[2] - 0)):
                pixel_bin_rep = "1"

            bits += str(pixel_bin_rep)
    del pixels
    return bits
```

In [21]:
```
# module : unix style pathname pattern expansion
import glob

def get_bits_from_video(videopath):
    print("Getting bits from video file...")

    bits = ""
    convert_video_to_frames(videopath)

    for image in sorted(glob.glob("./outframes/*.png")):
        bits += convert_image_to_bits(image)

    print("Successfully retrieved bits from video file")

    return bits
```

In [26]:
```
from bitstring import Bits, BitArray

def get_file_from_bits(bits, filepath):
    print("Generating gzip of original file from bits...")

    bitstring = Bits(bin = bits)
    bitstring = BitArray(bitstring)


    with open(filepath, 'wb') as outfile:
        bitstring.tofile(outfile)

    del bitstring

    print("Successfully retrieved the zipped archive")
```

In [28]:
```
setup()
compress_file('hello.txt')
bitarray = get_bitarray('hello.txt.gz')
generate_frames(bitarray)
generate_video()
bits = get_bits_from_video('output.mp4')
get_file_from_bits(bits, 'hello_new.txt.gz')
```

```
Compressing hello.txt...
Successfully compressed hello.txt
Converting hello.txt.gz to binary form...
Successfully converted hello.txt.gz to binary form
Generating frames...
Successfully generated all frames
Generating video file...
Successfully generated video file
Getting bits from video file...
Converting video file to respective frames...
Successfully generated all frames
Successfully retrieved bits from video file
Generating gzip of original file from bits...
Successfully retrieved the zipped archive
```

In [ ]: