```
def change( p ):
    """ change takes in a pixel (an [R,G,B] list)
        and returns a new pixel to take its place!
    """
    red = p[0]
    green = p[1]
    blue = p[2]
    return [ 255-red, 255-green, 255-blue ]


def invert(fname = 'in.png'):
    """Run this function to read the in.png image,
       change it, and write the result to out.png.
    """
    Im_pix = getRGB(fname)  # read in the in.png image
    print("The first two pixels of the first row are", Im_pix[0][0:2])
    #
    # Remember that Im_pix is a list (the image)
    # of lists (each row) of lists (each pixel is [R,G,B])
    #
    New_pix = copy.deepcopy(Im_pix)
    [numCols, numRows] = getWH(New_pix)
    for rowInd in range(numRows):
        for colInd in range(numCols):
            New_pix[rowInd][colInd] = change(New_pix[rowInd][colInd])
    # now, save to the file 'out.png'
    saveRGB(New_pix, 'out.png')

# test it out!
#invert('olin.png')
```

Invert that works as intended. Straight forward

```
def newChange( p ):
    """ change takes in a pixel (an [R,G,B] list)
        and returns a new pixel to take its place!
    """
    red = p[0]
    green = p[1]
    blue = p[2]
    lum = int((red*.21) + (green*.71) + (blue*.07))
    return [ lum, lum, lum ]

def greyscale(fname):
    Im_pix = getRGB(fname)  # read in the in.png image
    print("The first two pixels of the first row are", Im_pix[0][0:2])
    New_pix = copy.deepcopy(Im_pix)
    [numCols, numRows] = getWH(New_pix)
    for rowInd in range(numRows):
        for colInd in range(numCols):
            New_pix[rowInd][colInd] = newChange(New_pix[rowInd][colInd])
    # now, save to the file 'out.png'
    saveRGB(New_pix, 'out.png')

#greyscale('spam.png')
```

This wasn’t too bad

```
def binarize(fname, thresh):
    greyscale('spam.png')
    Im_pix = getRGB(fname)  # read in the in.png image
    print("The first two pixels of the first row are", Im_pix[0][0:2])
    New_pix = copy.deepcopy(Im_pix)
    [numCols, numRows] = getWH(New_pix)
    for rowInd in range(numRows):
        for colInd in range(numCols):
            if thresh <= threshCompare(New_pix[rowInd][colInd])[0]:
                New_pix[rowInd][colInd] = threshChangeWhite()
            if thresh >= threshCompare(New_pix[rowInd][colInd])[0]:
                New_pix[rowInd][colInd] = threshChangeBlack()

            if thresh <= threshCompare(New_pix[rowInd][colInd])[1]:
                New_pix[rowInd][colInd] = threshChangeWhite()
            if thresh >= threshCompare(New_pix[rowInd][colInd])[1]:
                New_pix[rowInd][colInd] = threshChangeBlack()

            if thresh <= threshCompare(New_pix[rowInd][colInd])[2]:
                New_pix[rowInd][colInd] = threshChangeWhite()
            if thresh >= threshCompare(New_pix[rowInd][colInd])[2]:
                New_pix[rowInd][colInd] = threshChangeBlack()

    # now, save to the file 'out.png'
    saveRGB(New_pix, 'out.png')

#binarize('out.png', 100)
```

This one is tricky and to be honest, to save time, im moving forward. It works with the out.png but I couldn’t figure out how to pass the original SPAM picture on it and use the other functions in conjunction.

```
def scaleBy2(fname):
    Im_pix = getRGB(fname)  # read in the in.png image
    print("The first two pixels of the first row are", Im_pix[0][0:2])
    New_pix = copy.deepcopy(Im_pix)
    [numCols, numRows] = getWH(New_pix)
    for rowInd in range(numRows):
        for colInd in range(numCols):
            if rowInd%2 != 0:
                New_pix[rowInd][colInd] = New_pix[rowInd-1][colInd]
            if colInd%2 != 0:
                New_pix[rowInd][colInd] = New_pix[rowInd][colInd-1]
    # now, save to the file 'out.png'
    saveRGB(New_pix, 'out.png')

#scaleBy2('spam.png')
```

This was tricky but I think it works as intended.

```python
def compress(S):
    binStr = []
    counterList = []
    counter = 0
    counter2 = 0
    starterList = []
    while S != '':
        counter = 0
        starter = S[0]
        starterList.append(starter)
        for j in S:
            if counter == 127:
                break
            if starter == j:
                counter += 1
                S = S[1:]
            else:
                break
        counterList.append(counter)

    while counterList != []:
        counter2 = 0
        bTen = counterList[0]
        binList = []
        while counter2 != 7:
            counter2 += 1
            if bTen%2 == 0:
                binList.insert(0,'0')
                bTen = bTen//2
            else:
                binList.insert(0,'1')
                bTen = bTen//2

        binList.insert(0,starterList[0])
        starterList = starterList[1:]
        binStr.append(''.join(binList))
        counterList = counterList[1:]

    return ''.join(binStr)
```

```python
def uncompress(S):
    totalList = []
    total = 0
    counter = 0
    outputStr = []
    while S != '':
        oneOrZero = S[0]
        counter = 0
        while counter != 8:
            totalList.append(S[0])
            S = S[1:]
            counter += 1
        totalList = totalList[1:]

        counter2 = 0
        total = 0
        for i in range(len(totalList)):
            if totalList == []:
                break
            if totalList[-1] == '1':
                total += (2**i)
                counter2 += 1
            totalList = totalList[:-1]

        outputStr.append(str(oneOrZero*total))

    return ''.join(outputStr)
```

These beasts. Was fun to figure out but took a long time.

David E Reynolds, Peter A Morales
Computer Science I (CSCI-1550-LN01) 2022FA

by the way ... you still seem to be resisting making helper functions (just sayin')

```
def compressedBinaryIm(s, cols, rows):
    S = uncompress(s)
    PX = []
    for row in range(rows):
        ROW = []
        for col in range(cols):
            c = int(S[row*cols + col])*255
            px = [ c, c, c ]
            ROW.append( px )
        PX.append( ROW )
    saveRGB( PX, 'binary.png' )
```

This does the job, but the image is all messed up. It has to do with the other functions that I couldn't get to synergize.

Overall, this took a long time to finish as 1 HW problem. The extra credit is tempting but I'm against the clock here. Moving on to the other assignments!