* This program is used to compress the image to smaller size with lower quality. It is the final project of COSI 175a, Brandeis University.
* May.1 2017
* Author: Ziyu Qiu (Credit: This program has been discussed with Yansen Sheng and Ziqi Wang, but the code was not shared and it is written independently by Ziyu Qiu.)

$1 HOW TO USE THE PROGRAM

1. Run the program in terminal with "python go.py"
2. The program will prompt you to choose the mission
   Enter 1 for Compress
   Enter 2 for Decompress
3. For compressing an image, the program will prompt you to enter the name of image file, simply enter the name with suffix.(Make sure the go.py and the image are in the same folder)
   You will then be able to choose the image quality after compression (As described in prompt, enter L for LOW, M for MEDIUM, H for HIGH).
   You should then decide whether you want to slice the image into 8*8 tiles or 16*16 tiles by entering 8 or 16.
   After the compression is done, you will have a chance to start a new mission (by entering "N"), or quit the program (by entering "Y") as the answer to the question "Are you all set?"
4. For decompressing an image, you need to enter the file name with suffix (Make sure the go.py and the image are in the same folder) and that's it.

$2 HOW THE PROGRAM WORKS

Generally the program is divided into 4 parts: **SET UP, ENCODE, DECODE, PURPOSE**.

It should have been arranged in a better order and group, but the nature of python limits that the methods must be defined before it is called, so I have to do it in this way.

The **PURPOSE** section is the control part of this program. It prompts user input to decide whether to encode or decode and call the functions accordingly.

The **SET UP** part stores most of the functions used by encode and decode. It is put together for better understanding because most of them appear in pair with its reverse function (like adjust & re_adjust, dct & reverse_DCT, and so on). Each parameter and return result is explained clearly in comment section before the method.

The **ENCODE** part is the core of this program. After getting the inputs from users, it will read the image, convert it to gray-scale images and turn them into matrix. I did the adjustment (subtract 128 from each pixel) in this step to save troublesome. Then the two for loops iterate over the height and length and crop the image to 8*8 or 16*16 tiles. If the image is not exactly the multiple of 8 or 16, it will pad the empty space. Whenever the small tile (sub matrix) is get from the original matrix, 2D-DCT is performed on the sub image (I get the DCT matrix in

advance to save time because it doesn't need to access cos and pi from numpy every single time). Then quantize the sub image by dividing luminance quantization matrix (it is get by dividing the suggested luminance quantization matrix by a unique factor decided by quality. That factor is decided empirically with previous tests. The factor is 1 for LOW quality with PSNR of ~25, 10 for MEDIUM quality with PSNR of ~28, 90 for HIGH quality with PSNR of ~32). After transformation, get the DC(save by difference with previous one) of each sub matrix and zigzag the AC's. Do the Huffman coding for DC's and run length-coding and Huffman coding for AC's. The Huffman coding is done with a dictionary which contains category, length and code(0 and 1's are swopped to represent negative numbers), and DC and AC share the same dictionary. (The dictionary is from slides and online slides, this limits my PSNR from increasing to some extent). Basically for each DC, it decides the absolute value of it and determine the length its going to use to represent as the first part, and the binary representation of the DC as the second part. For AC, it's basically the same except there's one more part about how many zero's there are beforehand. DC and AC codes are stored in two separate strings. After finish processing the image, the binary string is stored into one file and compressed by three lossless compression method (gzip, bzip2 and lzma). The string is composed in the following way:

side(0 stands for 8, and 1 stands for 16) + quality(used to decide the factor, '00' means factor =1, '01' means factor = 10, '10' means factor = 90) + image width + image height +DC_output + separator (20 consecutive 1's) + AC output

The **DECODE** part is mainly the reverse version of ENCODE part from the last step to the first step. It decodes the side, quality, image width, image height, and then start decoding DC's using the dictionary. When it hits the separator(20 consecutive 1's), it start decoding the AC's and do the reverse transition (reverse zigzag, reverse quantization, reverse dct and reverse adjust). The decode_dc and decode_ac share the decode_common because they share the same dictionary.