Q1

Code Explanation

- I'm using python to write this assignment
- The function `YCoCg_to_YUV(YCoCg=[])`
    - The input format for this function is a list
    - The function first convert the input YCoCg color to RGB color by the matrix given in lecture notes
    - Then convert the RGB color to the required YUV color format
    - Finally put the Y, U, and V to a list called `YUV`, and return this list as output

The output of the sample inputs

```
input: Y Co Cg = [105, 20, 45]
output: Y U V = [117, -77, -37]

input: Y Co Cg  = [149, 69, 61]
output: Y U V = [172, -153, -15]

input: Y Co Cg  = [108, 68, 7]
output: Y U V = [122, -89, 47]
```

Q2

Code Explanation

- The function `halftone_printing(picture=[[]])`
    - The input format for this function is a 2D list
    - The function first create a 2D list (called `result`) with doubled length in both dimensions as an output list. Since here we use 2x2 dithering matrix, the halftone printing method prints the picture 4 times as the original picture.
    - Then read the input 2D list one element at a time, and calculate the intensity level for each bit by the formula in notes: $intensity = bitValue/(256/5)$
    - After the calculation, the intensity is compared with the dither matrix.
        - If the intensity > the dither matrix entry, assign the corresponding entry as 1
        - otherwise, assign the corresponding entry as 0
    - Finally, print the `result` list
- The function `ordered_dithering(picture=[[]])`
    - The input format for this function is a 2D list
    - The function first create a 2D list (called `result`) with the same size as `picture` since ordered dithering method keep the same size as the original picture.
    - Then read the input 2D list one element at a time, and calculate the intensity level for each bit by the formula in notes: $intensity = bitValue/(256/5)$
    - Then comparing each intensity level to the corresponding dither matrix entry
        - If the intensity > the dither matrix entry, assign the corresponding entry as 1
        - otherwise, assign the corresponding entry as 0

  - Finally print the `result` list

The output of the sample inputs

```
Sample 1:
input:
[[32, 25, 165, 231],
[157, 63, 79, 86],
[231, 36, 168, 132],
[15, 125, 218, 87]]

output for halftone_printing:
[0, 0, 0, 0, 1, 0, 1, 1]
[0, 0, 0, 0, 1, 1, 1, 1]
[1, 0, 1, 0, 1, 0, 1, 0]
[1, 1, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 1, 0, 1, 0]
[1, 1, 0, 0, 1, 1, 0, 1]
[0, 0, 1, 0, 1, 1, 1, 0]
[0, 0, 0, 1, 1, 1, 0, 0]

output for ordered_dithering:
[0, 0, 1, 1]
[1, 0, 0, 0]
[1, 0, 1, 0]
[0, 1, 1, 0]



Sample 2:
input:
[[95, 249, 7, 216, 60, 48, 210, 149],
[65, 168, 169, 36, 152, 46, 116, 192],
[219, 222, 250, 210, 88, 16, 96, 34],
[105, 56, 175, 249, 11, 108, 72, 215],
[114, 134, 4, 200, 229, 91, 103, 238],
[81, 17, 147, 146, 204, 20, 77, 36],
[51, 59, 112, 99, 103, 17, 46, 245],
[38, 151, 218, 143, 209, 165, 152, 198]]

output for halftone_printing:
[1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0]
[0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1]
[1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0]
[0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0]
[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1]
[0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1]
[1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1]
[0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1]
[1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1]
[0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0]
[0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1]
```

```
output for ordered_dithering:
[1, 1, 0, 1, 1, 0, 1, 0]
[0, 1, 1, 0, 0, 0, 0, 1]
[1, 1, 1, 1, 1, 0, 1, 0]
[0, 0, 1, 1, 0, 1, 0, 1]
[1, 0, 0, 0, 1, 0, 1, 1]
[0, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 1, 0, 1, 0, 0, 1]
[0, 1, 1, 1, 1, 1, 0, 1]
```