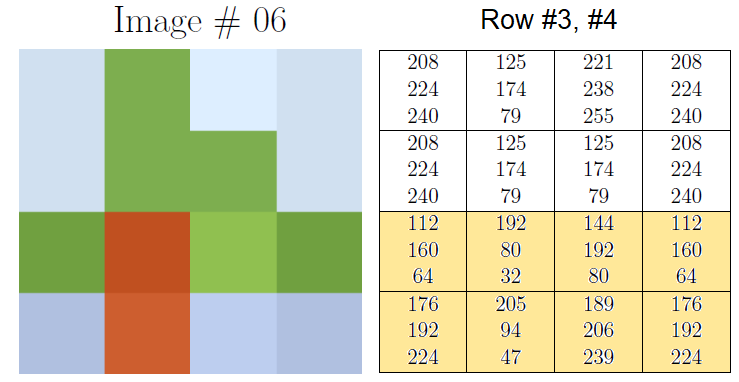
**Task 3: Extract image hidden inside an image**

My picture number is #6 and two rows are 3 and 4 for the Steganography assignment



**ROUND 1**

**First pixel**: row 3, column 1, RGB values: (112, 160, 47). We are going to extract the hidden colour values using conversion to binary.

RED: 250 is 01110000 in binary

The four least significant digits are 0000.

We use these are the leading digits of the hidden colour value: 00000000

We then convert 00000000 to decimal, which gives us 0. And so, the hidden value for red is 0.

GREEN: 160 is 10100000 in binary

The least significant digits are 0000.

The hidden colour value is 00000000

Convert 00000000 to decimal 0. The hidden value for GREEN is 0.

BLUE: 64 is 00100000 in binary

The least significant digits are 0000

The hidden colour value is 00000000

Convert 00000000 to decimal: 0. The hidden value for BLUE is 0.

The hidden colour value is RGB 0,0,0.

**ROUND 2**

**Second pixel**: row 4, column 2, RGB values: (205, 94, 47). We are going to extract the hidden colour values using conversion to hexadecimal.

RED: 205 is CD in hexadecimal

The least significant digit is D.

We use these are the leading digits of the hidden colour value: D0

We then convert D0 to decimal, which gives us 208. And so, the hidden value for red is 208.

GREEN: 94 is 5E in hexadecimal

The least significant digits are E.

The hidden colour value is E0

Convert E0 to decimal 224. The hidden value for GREEN is 224.

BLUE: 47 is 2F in hexadecimal

The least significant digits are F

The hidden colour value is F0

Convert F0 to decimal: 240. The hidden value for BLUE is 240.

The hidden colour value is RGB 208,224,240.

**ROUND 3**

We can use bitwise arithmetic to retrieve the hidden image from the input image, **without** requiring us to convert to binary / hexadecimal. The method described in the explanation section below.

**Third pixel**: row 3, column 3, RGB values: (144, 192, 80). We are going to extract the hidden colour values using bitwise operations.

RED: The decimal equivalent for the 4 least significant bits of the pixel 144 & 15 = 0, which is also the decimal equivalent of the 4 most significant bits of the hidden pixel (where (15)10 = (0F)16).

Hence, the hidden value for red is 0\*16 = 0 (which can be computed with 0 << 4, left shift).

GREEN: The decimal equivalent for the 4 least significant bits of the pixel 192 & 15 = 0, which is also the decimal equivalent of the 4 most significant bits of the hidden pixel.

Hence, the hidden value for GREEN is 0\*16 = 0.

BLUE: The decimal equivalent for the 4 least significant bits of the pixel 80 & 15 = 0, which is also the decimal equivalent of the 4 most significant bits of the hidden pixel.

Hence, the hidden value for BLUE is 0\*16 = 0.

The hidden colour value is RGB 0,0,0.

**Fourth pixel**: row 4, column 3, RGB values: (189, 206, 239). Again, we are going to extract the hidden colour values using bitwise operations.

RED: The decimal equivalent for the 4 least significant bits of the pixel 189 & 15 = 13, which is also the decimal equivalent of the 4 most significant bits of the hidden pixel (where (15)10 = (0F)16).

Hence, the hidden value for red is 13\*16 = 208 (which can be computed with 13 << 4, left shift).

GREEN: The decimal equivalent for the 4 least significant bits of the pixel 206 & 15 = 14, which is also the decimal equivalent of the 4 most significant bits of the hidden pixel.

Hence, the hidden value for GREEN is 14 << 4 = 14\*16 = 224.

BLUE: The decimal equivalent for the 4 least significant bits of the pixel 239 & 15 = 15, which is also the decimal equivalent of the 4 most significant bits of the hidden pixel.

Hence, the hidden value for BLUE is 15 << 4 = 15\*16 = 240.

The hidden colour value is RGB 208,224,240.

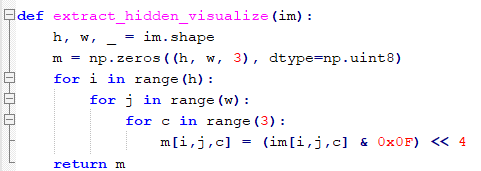
**EXPLANATION OF METHOD USED IN ROUND 3**

For each pixel **p**, expressed as 3-tuple **(pR, pG, pB)** corresponding to 3 colour channels red, green, blue (where each pR, pG, pB value is represented as **decimal**, 8 bit unsigned integer value), do the following to retrieve the hidden pixel **q** from it, again , expressed as 3-tuple **(qR, qG, qB)**:  
  
1. Retrieve the 4 least significant bits from **p** using **logical AND** with (decimal value) 15 (since **(15)10 = (0F)16** or **0x0F**), to extract the 4 most significant bits of the hidden image **q**, for each colour channel:

(pR, pG, pB) = p  
qR = pR & 15  
qG = pG & 15  
qB = pB & 15

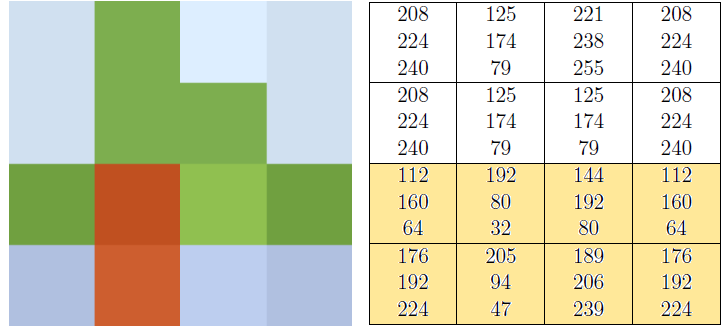
2. Left shift by 4 bits (**bitwise arithmetic**) for each colour channel of the hidden image and combine them as 3-tuple to obtain hidden image **q**:   
qR = qR << 4  
qG = qG << 4  
qB = qB << 4  
q = (qR, qG, qB)

The python code implementing the above is shown below, it accepts the cover image and returns the hidden image.

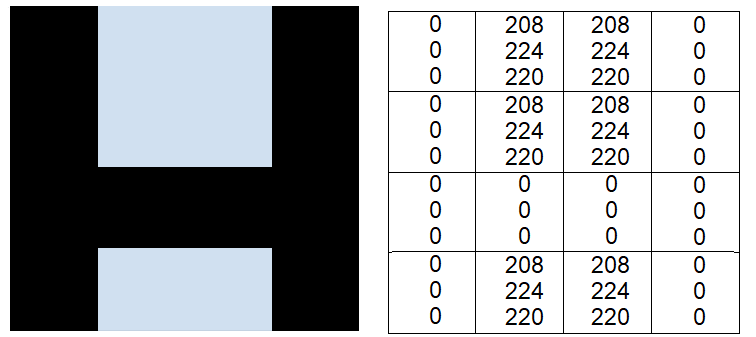


**RESULTS FROM MY IMAGE**

My initial image and colour values:

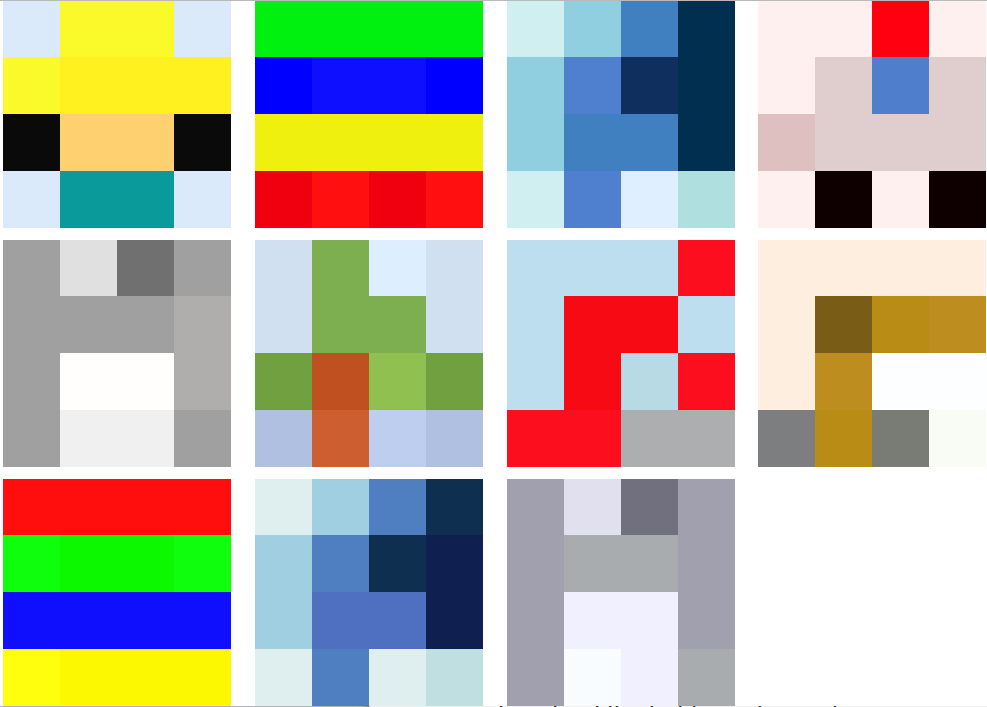


Hidden image colours values and image:



**Corresponding Letter: H**

**RESULTS FROM ALL IMAGES**

The 11 pixel images are shown below: ****

The 11 decoded hidden images are shown below:  


**Letters in order:**

G,R,A,C,E,H,O,P,P,E,R