Digital Images: Row-Column Format

A digital image consists of pixels. Each pixel is a numeric value that characterizes the color content at that point. The point is the physical location of where light which passed thru a lens, fell on the image sensor.

The physical location is often referenced by the row and column on the image sensor. One intuitive (*but naïve*) way is to characterize the digital image as an array of pixels.

Example Recall that one starts count with "0" and that arrays are denoted by (row, col).

	0	1	2	3
0	A	B	C	D
1	E	F	G	H
2	I	J	K	L

Suppose we have an image sensor that yielded a 3x4 digital image Img. One sees that pixel location $Img(1,3)$ has the value H.

For small digital images, array data structures are fine. But for larger ones, the row-column format data structure with a pointer makes computations quicker.

0	1	2	3	4	5	6	7	8	9	10	11
A	B	C	D	E	F	G	H	I	J	K	L

```
*(Img.Data + (R * Img.Cols) + C)
```

`Img.Data:` pixel values (e.g. A thru L)
`R`: pixel's row (e.g. 1)
`C`: pixel's column (e.g. 3)
`Img.Cols`: number of columns in the image (e.g. 4)

In this example, we see $Img(1,3)$ would be `*(Img.Data + (1 * 4) + 3) = *(Img.Data + 7)`

Since the pointer points to the start of the row-column vector, the value is "H"

Often in image processing, one sees the row-column format in a nested `for-loop`

```
void Img_threshold(struct Image *In, struct Image *Out) {

 long i, j;
 int val, thresholdValue;
 unsigned char *tmp;

 thresholdValue = 50;

 for(i=0; i<In->Rows; ++i) {
    for(j=0; j<In->Cols; ++j) {
        val = *(In->Data + i*In->Rows + j);
        if(val < thresholdValue) {
            val = 0;
        }
        else {
            val = 255;
        }
        tmp = Out->Data + i*Out->Rows + j;
        *tmp = (unsigned char)val;
    };
 };
} // end Img_threshold
```

In this sample code, the image's pixel value is stored in a variable called `val`. One sees the value is compared to a threshold value. This function generated a thresholded image.

```
struct Image {
    int Rows;             // image's number of rows
    int Cols;             // image's number of columns
    unsigned char *Data;  // pointer to image data
}; // end of struct Image
```

One also sees that a (pointer to a) struct variable is used. This is also common in image processing. The structure often contains information about the image like the number of rows, columns, and the pixel data itself

Digital image files come in many versions e.g. RAW, BMP, PGM, JPG, and GIF. The former three are uncompressed and hence are large file sizes. The latter two are compressed to save memory space.

RAW files are the “purest” ones and popular with photographers; the light and color characteristics falling onto the image sensor are unaltered. The photographer can then customize the image and the relative values of adjacent pixels as they see fit.

For grey-scale RAW images, each pixel is one byte (value of 0 to 255). Thus a 256x256 greyscale image is 65536 bytes.

For color RAW images, each pixel is represented by 3 bytes (RGB: red, green and blue). Thus a 256x256 greyscale image is 65536*3 = 196608 bytes.