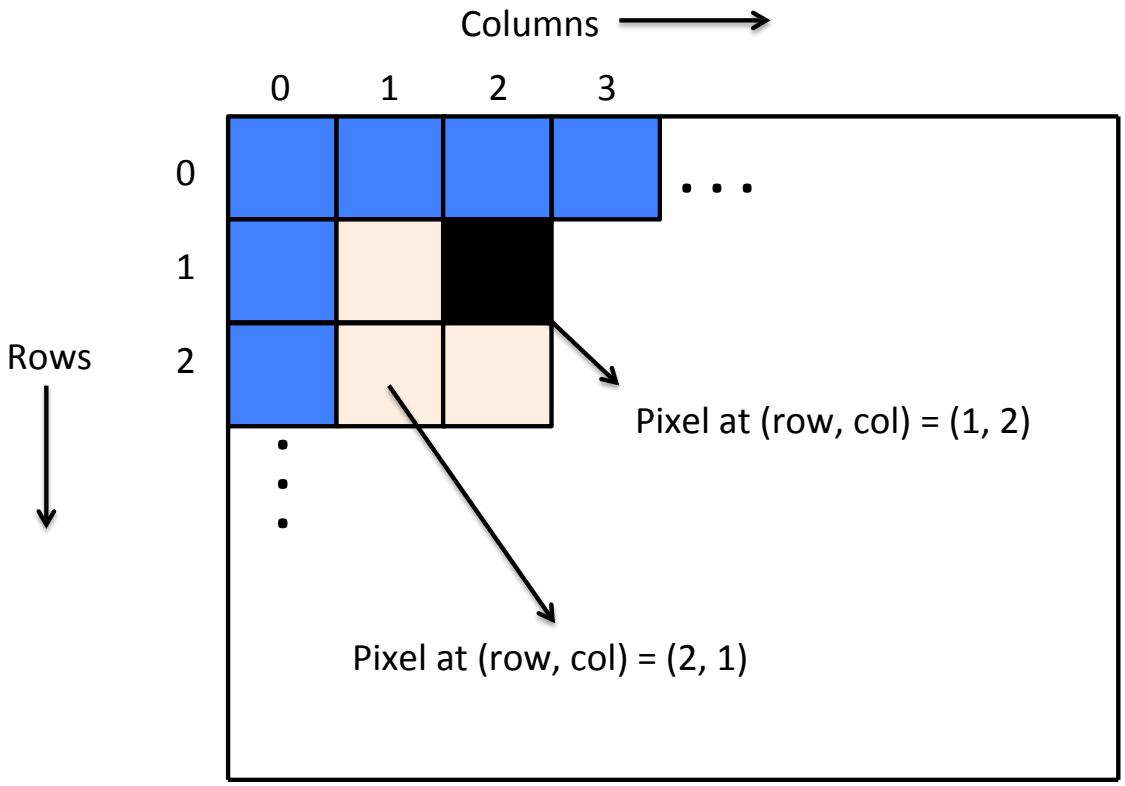
# Introduction to Digital Images

Digital images are all around us. Here we will pull aside the curtain a little, and discover how images are represented inside a computer. Let's begin with an image of some yellow flowers:

This image looks natural and smooth to our eyes; however, notice that the image gets "blocky" if we zoom in 10x on the flower in the upper left corner, as shown in the image below.

Digital images are thus composed of many square *pixels* (short for "picture elements"), each of which displays a single, pure color. A typical web-sized image is 800 pixels high by 600 pixels wide and contains $800 \times 600 = 480,000$ pixels in all, or about 0.5 megapixels

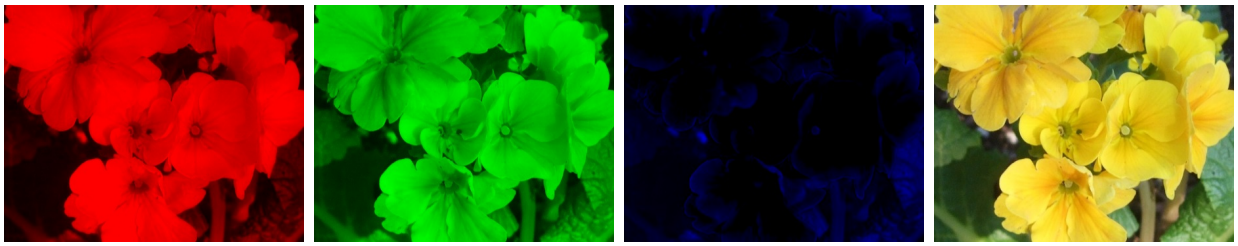
("mega" ≈ one million). Camera phones on the market today produce images that contain between 8 and 15 megapixels — a greater number of pixels corresponds to an image with higher *resolution*, i.e., a greater ability to capture fine details. The pixels are arranged in a grid, with each pixel uniquely identified by its row and column positions within this grid, as shown in the figure below:




Now that we understand how the pixels are laid out, how do we represent the color of a pixel? While there are many commonly used schemes, one of the most popular ones is the Red-Green-Blue (RGB) system. Under this scheme, every color is represented as the mixture of varying proportions of pure red, green and blue lights. Each of the red, green and blue light levels is encoded as an integer in the range $0 \ldots 255$, with 0 meaning a complete absence of light and 255 meaning maximum light. So, for example, $(R = 255, G = 100, B = 0)$ is the color created by mixing the maximum amount of red, a medium amount of green, and no blue at all — the result is a shade of orange. Similarly, $(R = 150, G = 50, B = 240)$ produces purple. What is the color $(R = 0, G = 0, B = 0)$? How about $(R = 255, G = 255, B = 255)$?

You may be wondering — what's so special about the number 255? Why do computers use 256 different levels for each of the three colors? Why not fewer? Or more? It is estimated that the human eye is capable of perceiving roughly 10 million distinct colors. By employing 256 distinct levels for each of red, green and blue, our computers can produce $256 \times 256 \times 256 \approx 16$ million different colors — enough to match and surpass human visual ability[1].

Here's a more holistic view of the color mixing process in action: we can divide the original image into three color "channel" images, one for each of red, green and blue. The red channel shows only the intensities of red in each pixel, with green and blue set to 0. The green and blue channel images similarly show only the distribution of green and blue values in the image, respectively. Mixing these three channels together recovers the original image of the flowers. Notice that the yellow flowers are bright in both the red and green channels. The green leaves pretty much only appear in the green channel. The blue channel is mostly near 0 everywhere, as the original image simply does not contain much blue.



Finally, let's look at the problem of how to store image information within a Python program. Consider the color channel images from above — under the hood, these would correspond to tables of numbers like those shown below, where each table entry would be some integer in the range $0 \ldots 255$.

| | | | |
|---|---|---|---|
| 120 | 140 | 190 | 91 |
| 14 | 52 | 160 | 89 |
| 71 | 78 | 243 | 232 |

| | | | |
|---|---|---|---|
| 220 | 240 | 90 | 191 |
| 140 | 252 | 160 | 79 |
| 171 | 178 | 213 | 32 |

| | | | |
|---|---|---|---|
| 20 | 40 | 19 | 29 |
| 14 | 2 | 60 | 9 |
| 7 | 8 | 24 | 32 |

Focusing on the red channel data, we could take the entire first row and build a list containing just those data values and store the result in a variable like so:

[1] Though not sufficient to fool butterflies or pigeons, that are estimated to be able to perceive 10 *billion* distinct colors!

```
red_row_1 = [120, 140, 190, 91]
```

Similarly, we could create separate lists for each of the other rows:

```
red_row_2 = [14, 52, 160, 89]
red_row_3 = [71, 78, 243, 232]
```

But this approach will not scale well to real-world images that often contain hundreds, if not thousands, of rows of pixels — it would be tedious to create the large number of individual variables that would be required to hold all these separate lists. However, we can solve this problem by *nesting* our lists, i.e., by building a "master" list whose elements are lists themselves, corresponding to each row in our image. In our example, we could store the entire red channel data in a single variable as follows:

```
red_channel = [ [120, 140, 190,  91],
                [ 14,  52, 160,  89],
                [ 71,  78, 243, 232] ]
```

Similarly, we could create separate nested lists to hold the green and blue channel data. Section 10.24 of the course textbook delves into a little more detail about nested lists — read this section and complete the activities contained in it. Then, work through the `image_demo.py` tutorial.