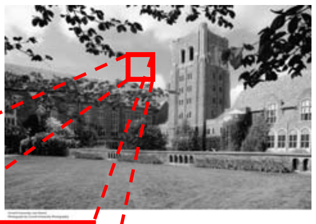
- Previous class:
  - Play with sound files
  - Practice working with vectors
- Now:
  - Play with image files
  - 2-dimensional array—matrix

A picture as a matrix—2-dimensional array

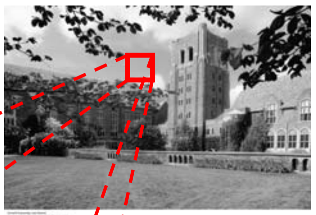
1458-by-2084



```
150   149   152   153   152   155
151   150   153   154   153   156
153   151   155   156   155   158
154   153   156   157   156   159
156   154   158   159   158   161
157   156   159   160   159   162
```

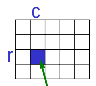
Grayness: a value in [0..255]

0 = black
255 = white



```
150   149   152   153   152   155
151   150   153   154   153   156
153   151   155   156   155   158
154   153   156   157   156   159
156   154   158   159   158   161
157   156   159   160   159   162
```

2-d array: **matrix**



- An array is a named collection of like data organized into rows and columns
- A 2-d array is a table, called a *matrix*
- Two *indices* identify the position of a value in a matrix, e.g.,

  `mat(r,c)`

  refers to component in row r, column c of matrix mat
- Array index starts at 1
- Rectangular: all rows have the same #of columns

Creating a matrix

- Built-in functions: ones, zeros, rand
  - E.g., zeros(2,3) gives a 2-by-3 matrix of 0s
- "Build" a matrix using square brackets, [ ], but the dimension must match up:
  - [x y] puts y to the right of x
  - [x; y] puts y below x
  - [4 0 3; 5 1 9] creates the matrix

4	0	3
5	1	9

  - [4 0 3; ones(1,3)] gives

4	0	3
1	1	1

  - [4 0 3; ones(3,1)] doesn't work

```
% What will M be?
M = [ones(1,3); 1:4]
```

A
```
1  1  1  0
1  2  3  4
```

B
```
1  1  1
1  2  3
```

C *Error – M not created*

What is `[7 0 5]'` ?

- A Same as `[5 0 7]`
- B Same as `[7; 0; 5]`
- C Same as `[5; 0; 7]`

What will A be?

```
A= [1  1]
A= [A'  ones(2,1)]
A= [1  1  1  1;  A  A]
```

- A 3-by-4 matrix
- B 4-by-3 matrix
- C vector of length 12
- D *Error*

Working with a matrix: `size` and individual components

Given a matrix M

2	-1	.5	0	-3
3	8	6	7	7
5	-3	8.5	9	10
52	81	.5	7	2

```
[nr, nc]= size(M)   % nr is #of rows,
                    % nc is #of columns
M(2,4)= 1;
disp(M(3,1));
M(1,nc)= 4;
```

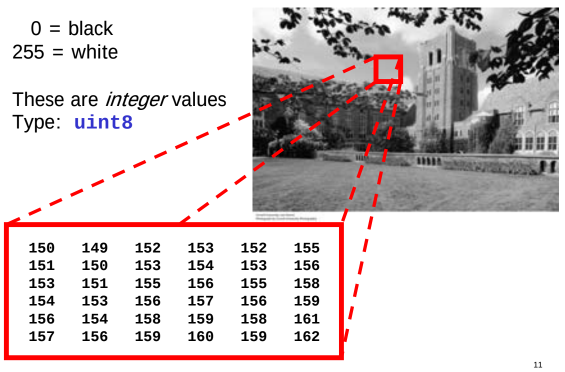
Images can be encoded in different ways

- Common formats include
  - JPEG: Joint Photographic Experts Group
  - GIF: Graphics Interchange Format
- Data are compressed
- We will work with jpeg files:
  - `imread`: read a .jpg file and convert it to a "normal numeric" array that we can work with
  - `imwrite`: write an array into a .jpg file (compressed data)

Grayness: a value in [0..255]

0 = black
255 = white

These are *integer* values
Type: `uint8`

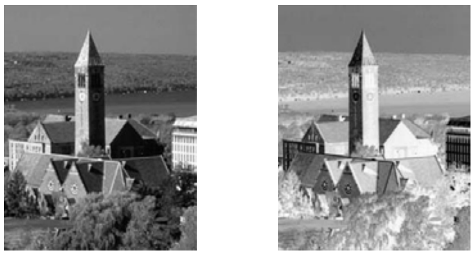


```
150  149  152  153  152  155
151  150  153  154  153  156
153  151  155  156  155  158
154  153  156  157  156  159
156  154  158  159  158  161
157  156  159  160  159  162
```

Let's put a picture in a frame

- Read a grayscale jpeg file into a matrix P
  P = imread('<filename>.jpg');
- See the image represented by P
  imshow(P)
- Change the "edge pixels" into the frame color (grayscale) you want
  …

Problem: produce a negative



13

Problem: produce a negative

- "Negative" is what we say, but all color values are positive numbers!
- Think in terms of the extremes, 0 and 255. Then the "negative" just means the opposite side.
- So 0 is the opposite of 255;

1	…	254;
5	…	250;
30	…	225;
x	…	255-x

14