Previous class:
- Play with image (jpeg) files
- 2-dimensional array—matrix

Now:
- Color images—3-dimensional array
- Multi-media project

Problem: produce a negative

A color picture is made up of RGB matrices

<table>
<thead>
<tr>
<th>Color image</th>
<th>3-d Array</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="RGB Matrices" /></td>
<td><img src="image2.png" alt="3-d Array" /></td>
</tr>
</tbody>
</table>

Operations on images amount to operations on matrices—good way to practice matrix manipulation!

Grayness: a value in [0..255]

0 = black
255 = white

These are integer values
Type: uint8

<table>
<thead>
<tr>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>149</td>
<td>152</td>
<td>153</td>
<td>152</td>
<td>155</td>
</tr>
<tr>
<td>151</td>
<td>150</td>
<td>153</td>
<td>154</td>
<td>153</td>
<td>156</td>
</tr>
<tr>
<td>153</td>
<td>151</td>
<td>155</td>
<td>156</td>
<td>155</td>
<td>158</td>
</tr>
<tr>
<td>154</td>
<td>153</td>
<td>156</td>
<td>157</td>
<td>156</td>
<td>159</td>
</tr>
<tr>
<td>156</td>
<td>154</td>
<td>158</td>
<td>159</td>
<td>158</td>
<td>161</td>
</tr>
<tr>
<td>157</td>
<td>156</td>
<td>159</td>
<td>160</td>
<td>159</td>
<td>162</td>
</tr>
</tbody>
</table>

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 \( \ldots \) 254;
  - 5 \( \ldots \) 250;
  - 30 \( \ldots \) 225;
  - \( x \) \( \ldots \) 255-\( x \)

Example: Mirror Image

![LawSchool.jpg](image3.png) ![LawSchoolMirror.jpg](image4.png)
**Solution Framework**

1. Read *LawSchool.jpg* from memory and convert it into an array.
2. Manipulate the Array.
3. Convert the array to a jpg file and write it to memory.

**Reading and writing jpg files**

```matlab
% Read jpg image and convert to a 3D array A
A = imread('LawSchool.jpg');

% Write 3D array B to memory as a jpg image
imwrite(B,'LawSchoolMirror.jpg')
```

**A 3-d array as 3 matrices**

- 4-by-6 \( M_1 = A(:,:,1) \)
- 4-by-6 \( M_2 = A(:,:,2) \)
- 4-by-6 \( M_3 = A(:,:,3) \)

```matlab
% Make B, a mirror image of A
[nr, nc, np] = size(A);  % dimensions of 3-d array A
for r = 1:nr
    for c = 1:nc
        B(r, c) = A(r, nc-c+1);
    end
end
```

```matlab
% Make B, a mirror image of A
[nr, nc, np] = size(A);  % dimensions of 3-d array A
for r = 1:nr
    for c = 1:nc
        for p = 1:np
            B(r, c, p) = A(r, nc-c+1, p);
        end
    end
end
```

```matlab
% Make B, a mirror image of A
[nr, nc, np] = size(A);  % dimensions of 3-d array A
for r = 1:nr
    for c = 1:nc
        B(r, c) = A(r, nc-c+1);
    end
end
```
% Make mirror image of A -- the whole thing
A= imread('LawSchool.jpg');
[nr,nc,np]= size(A);
B= zeros(nr,nc,np);
B= uint8(B); % Type for image color values
for r= 1:nr
  for c= 1:nc
    for p= 1:np
      B(r,c,p)= A(r,nc-c+1,p);
    end
  end
end
imwrite(B) % Show 3-d array data as an image
imwrite(B,'LawSchoolMirror.jpg')

Consider a single matrix (just one layer)
[nr,nc,np] = size(A);
for c= 1:nc
  B( :,c  ) = A( :,nc+1-c  );
end

The colon says "all indices in this dimension." In this case it says "all rows."

Vectorized code simplifies things...
Work with a whole column at a time
A = imread('LawSchool.jpg')
[nr,nc,np] = size(A);
for c= 1:nc
  B(:,c,1) = A(:,nc+1-c,1)
  B(:,c,2) = A(:,nc+1-c,2)
  B(:,c,3) = A(:,nc+1-c,3)
end
imwrite(B,'LawSchoolMirror.jpg')

Turn the white duck yellow!
■ The duck’s body and the image’s background show some contrast. However, neither the duck’s body nor the background has a uniform color
■ Are the RGB values different enough for us to write a “rule” in the program to tell between the duck and the background?
■ Check out the RGB values!

Vectorized code to create a mirror image
A = imread('LawSchool.jpg')
[nr,nc,np] = size(A);
for c= 1:nc
  B(:,c,1) = A(:,nc+1-c,1)
  B(:,c,2) = A(:,nc+1-c,2)
  B(:,c,3) = A(:,nc+1-c,3)
end
imwrite(B,'LawSchoolMirror.jpg')

Extracting subarrays and tiling
■ Accessing a submatrix: M(:,::)
■ Accessing a subarray (3-d): P(:,:,::)
■ Concatenate horizontally: [ PL  PR ]
■ Concatenate vertically: [ PT;  PB ]