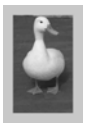
- Previous Lecture:
  - 2-d array examples

- Today's Lecture:
  - Complete matrix example from previous lecture
  - Image processing
    - Type `uint8`
    - Vectorized code for accessing subarrays
- Announcement:
  - Prelim 2 tonight 7:30-9pm in Kennedy Aud.

Available data

`C(i,j)` is what it costs factory i to make product j

`Inv(i,j)` is the inventory in factory i of product j

`PO(j)` is the number of product j's that the client wants

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

Inv

38	5	99	34	42
82	19	83	12	42
51	29	21	56	87

PO

1	0	12	29	5

Who Can Fill the Order?

Inv

38	5	99	34	42	Yes
82	19	83	12	42	No
51	29	21	56	87	Yes

PO

1	0	12	29	5

Wanted: A True/False Function

DO is "true" if factory i can fill the order.

DO is "false" if factory i cannot fill the order.

Encapsulate…

```
function  DO = iCanDo(i,Inv,PO)
% DO is true if factory i can fill
% the purchase order. Otherwise, false

nProd = length(PO);
DO = 1;
for j = 1:nProd
     DO = DO && ( Inv(i,j) >= PO(j) );
end
```

inv

8	9	5	4

PO

7	3	6	3

Encapsulate…

```
function  DO = iCanDo(i,Inv,PO)
% DO is true if factory i can fill
% the purchase order. Otherwise, false
nProd = length(PO);
j = 1;
while j<=nProd && Inv(i,j)>=PO(j)
   j = j+1;
end
DO = _______;
```

Back To Finding the Cheapest

```
iBest = 0; minBill = inf;
for i=1:nFact
    iBill = iCost(i,C,PO);
    if iBill < minBill
        % Found an Improvement
        iBest = i; minBill = iBill;
    end
end
```

Don't bother with this unless there is sufficient inventory.

Back To Finding the Cheapest

```
iBest = 0; minBill = inf;
for i=1:nFact
    if iCanDo(i,Inv,PO)
        iBill = iCost(i,C,PO);
        if iBill < minBill
        % Found an Improvement
            iBest = i; minBill = iBill;
        end
    end
end
```

See `Cheapest.m` for alternative implementation

Accessing a submatrix

`M`

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

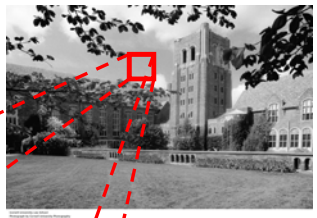
- `M` refers to the whole matrix
- `M(3,5)` refers to one component of `M`
- `M(2:3,3:5)` refers to a submatrix of `M`

row indices

column indices

A picture as a matrix

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
```


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)

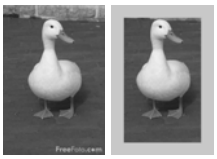
Grayness: a value in [0..255]

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



Things to do:

1. Read `bwduck.jpg` from memory and convert it into an array
2. Show the original picture
3. Assign a gray value (frame color) to the "edge pixels"
4. Show the manipulated picture

Reading a jpeg file and displaying the image

```
% Read jpg image and convert to
% an array P
  P = imread('bwduck.jpg');

% Show the data in array P as
% an image
  imshow(P)
```



```
% Frame a grayscale picture

P= imread('bwduck.jpg');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 200;  % light gray
[nr,nc]= size(P);
for r= 1:nr
  for c= 1:nc
    % At pixel (r,c)




  end
end
imshow(P)
```



```
% Frame a grayscale picture

P= imread('bwduck.jpg');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 200;  % light gray
[nr,nc]= size(P);
for r= 1:nr
  for c= 1:nc
    % At pixel (r,c)
    if  r<=width || r>nr-width || ...
        c<=width || c>nc-width
      P(r,c)= frameColor;
    end
  end
end
imshow(P)
```

Things to consider...
1. What is the type of the values in P?
2. Can we be more efficient?


```
% Frame a grayscale picture

P= imread('bwduck.jpg');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 200;  % light gray
[nr,nc]= size(P);
for r= 1:nr
  for c= 1:nc
    % At pixel (r,c)
    if  r<=width || r>nr-width || ...
        c<=width || c>nc-width
      P(r,c)= frameColor;
    end
  end
end
imshow(P)
```

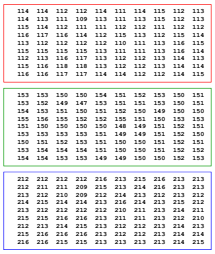
`P(r,c)= uint8(framColor);`

P has type uint8: cannot change just one bin, P(r,c), to type double. So Matlab did the type conversion to uint8 for us behind the scenes.

Things to consider...
1. What is the type of the values in P?
2. Can we be more efficient?

See `pictureFrame*.m`

A color picture is made up of RGB matrices → 3-d array



E.g., color image data is stored in a 3-d array A:

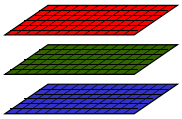
$0 \le A(i,j,1) \le 255$

$0 \le A(i,j,2) \le 255$

$0 \le A(i,j,3) \le 255$

A color picture is made up of RGB matrices → 3-d array

Color image



3-d Array

$0 \le A(i,j,1) \le 255$

$0 \le A(i,j,2) \le 255$

$0 \le A(i,j,3) \le 255$

Operations on images amount to operations on matrices!

Example: Mirror Image

LawSchool.jpg

LawSchoolMirror.jpg

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

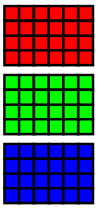
```
% Read jpg image and convert to
% a 3D array A
  A = imread('LawSchool.jpg');

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


A 3-d array as 3 matrices

```
[nr, nc, np] = size(A)  % dimensions of 3-d array A
```

#rows #columns #layers (pages)

```
A(1:nr,1:nc,1)
```



4-by-6 `M1= A(:,:,1)`

4-by-6 `M2= A(:,:,2)`

4-by-6 `M3= A(:,:,3)`


```
%Store mirror image of A in array B

[nr,nc,np]= size(A);
for r= 1:nr
  for c= 1:nc

      B(r,c  )= A(r,nc-c+1  );

  end
end
```

A B


```
%Store mirror image of A in array B

[nr,nc,np]= size(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
```