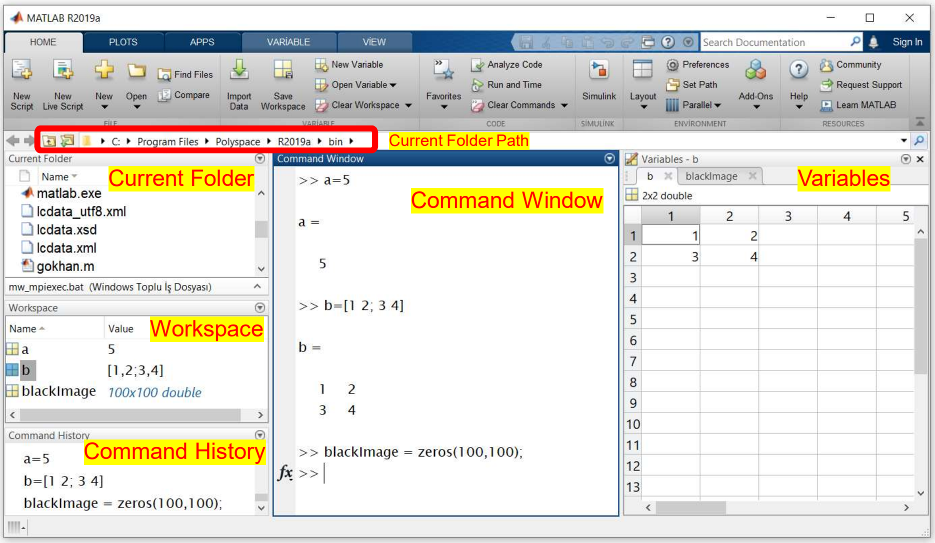
EE409 – Digital Image Processing

2021-2022 Fall

2. Introduction to MATLAB for Image Processing

by: Dr. Gökhan Koray GÜLTEKİN

MATLAB Environment



C: ▸ Program Files ▸ Polyspace ▸ R2019a ▸ bin ▸

Current Folder

Name
- matlab.exe
- lcdata_utf8.xml
- lcdata.xsd
- lcdata.xml
- gokhan.m

mw_mpiexec.bat (Windows Toplu İş Dosyası)

Workspace

Name	Value
a	5
b	[1,2;3,4]
blackImage	100x100 double

Command History

```
a=5
b=[1 2; 3 4]
blackImage = zeros(100,100);
```

Command Window

```
>> a=5

a =

     5

>> b=[1 2; 3 4]

b =

     1     2
     3     4

>> blackImage = zeros(100,100);
fx >> 
```

Variables - b

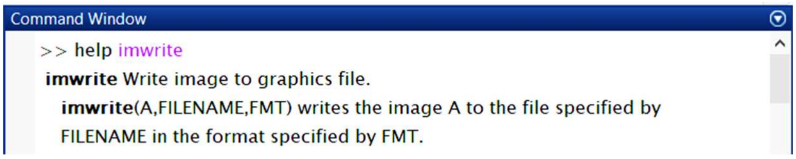
b | blackImage

2x2 double

	1	2	3	4	5
1	1	2			
2	3	4			
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					

MATLAB and images

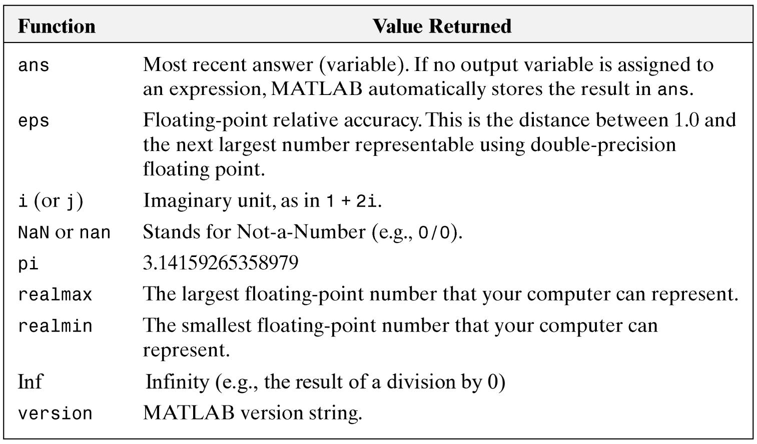
- The help in MATLAB is very good, use it!
- To learn the use of a function (e.g. imwrite) you can type ***help imwrite*** in the command window:



```
>> help imwrite
 imwrite Write image to graphics file.
   imwrite(A,FILENAME,FMT) writes the image A to the file specified by
   FILENAME in the format specified by FMT.
```

- An image in MATLAB is treated as a matrix (2D or 3D)
- Every matrix element describes the brightness/color of a pixel
- All the operators in MATLAB defined on matrices can be used on images: +, -, *, /, ^, sqrt, sin, cos etc.

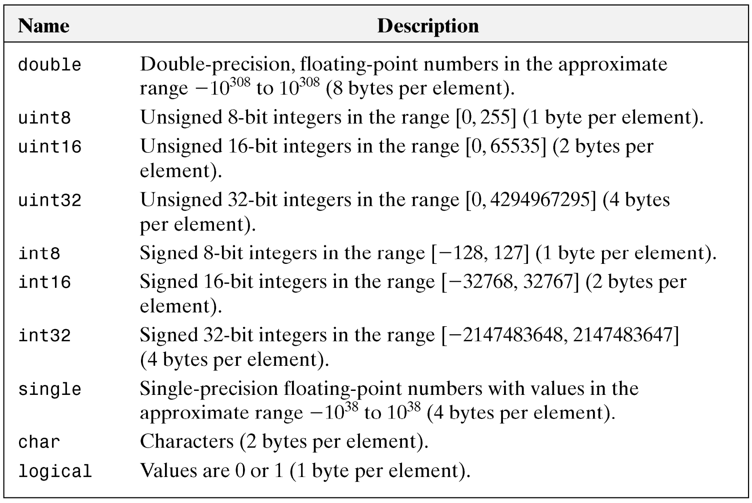
Matlab Predefined Variables and Constants



Function	Value Returned
`ans`	Most recent answer (variable). If no output variable is assigned to an expression, MATLAB automatically stores the result in `ans`.
`eps`	Floating-point relative accuracy. This is the distance between 1.0 and the next largest number representable using double-precision floating point.
`i` (or `j`)	Imaginary unit, as in `1 + 2i`.
`NaN` or `nan`	Stands for Not-a-Number (e.g., `0/0`).
`pi`	3.14159265358979
`realmax`	The largest floating-point number that your computer can represent.
`realmin`	The smallest floating-point number that your computer can represent.
Inf	Infinity (e.g., the result of a division by 0)
`version`	MATLAB version string.

TABLE 2.10 Some important variables and constants.

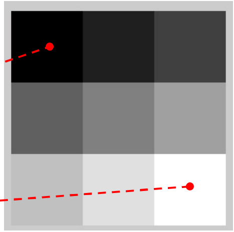
Data Classes



Name	Description
`double`	Double-precision, floating-point numbers in the approximate range -10^{308} to 10^{308} (8 bytes per element).
`uint8`	Unsigned 8-bit integers in the range [0, 255] (1 byte per element).
`uint16`	Unsigned 16-bit integers in the range [0, 65535] (2 bytes per element).
`uint32`	Unsigned 32-bit integers in the range [0, 4294967295] (4 bytes per element).
`int8`	Signed 8-bit integers in the range [−128, 127] (1 byte per element).
`int16`	Signed 16-bit integers in the range [−32768, 32767] (2 bytes per element).
`int32`	Signed 32-bit integers in the range [−2147483648, 2147483647] (4 bytes per element).
`single`	Single-precision floating-point numbers with values in the approximate range -10^{38} to 10^{38} (4 bytes per element).
`char`	Characters (2 bytes per element).
`logical`	Values are 0 or 1 (1 byte per element).

TABLE 2.2 Data classes. The first eight entries are referred to as *numeric* classes; the ninth entry is the *character* class, and the last entry is of class *logical.*

Images and Matrices



- How to build a matrix (or image)?

```
>> A = [ 1 2 3; 4 5 6; 7 8 9 ];
 A = 1    2    3
     4    5    6
     7    8    9
```

>> imshow(A,**[ ]**); shows image A using **automatic pixel range**,
(i.e. highest value is white, lowest value is black)

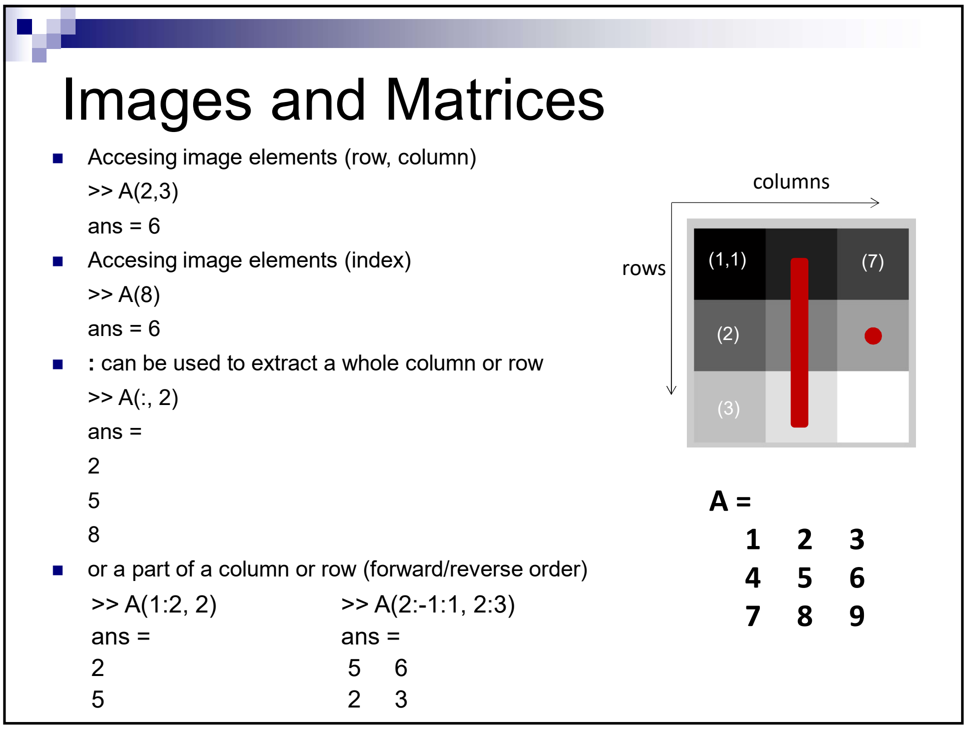
>> imshow(A); shows image A **according to the variable type of A**,
(If A is "uint8" then 255->White, If A is "double" then 1-> white)

3x3 <u>black</u> image of type "double":

```
>> B = zeros(3,4)
   B =  0    0    0
        0    0    0
        0    0    0
```

3x3 <u>white</u> image of type "double":

```
>> C = ones(3,3)
   C = 1    1    1
       1    1    1
       1    1    1
```



Images and Matrices

- Accesing image elements (row, column)

```
>> A(2,3)
ans = 6
```

- Accesing image elements (index)

```
>> A(8)
ans = 6
```

- **:** can be used to extract a whole column or row

```
>> A(:, 2)
ans =
2
5
8
```

- or a part of a column or row (forward/reverse order)

```
>> A(1:2, 2)          >> A(2:-1:1, 2:3)
ans =                 ans =
2                      5    6
5                      2    3
```

A =

```
  1   2   3
  4   5   6
  7   8   9
```

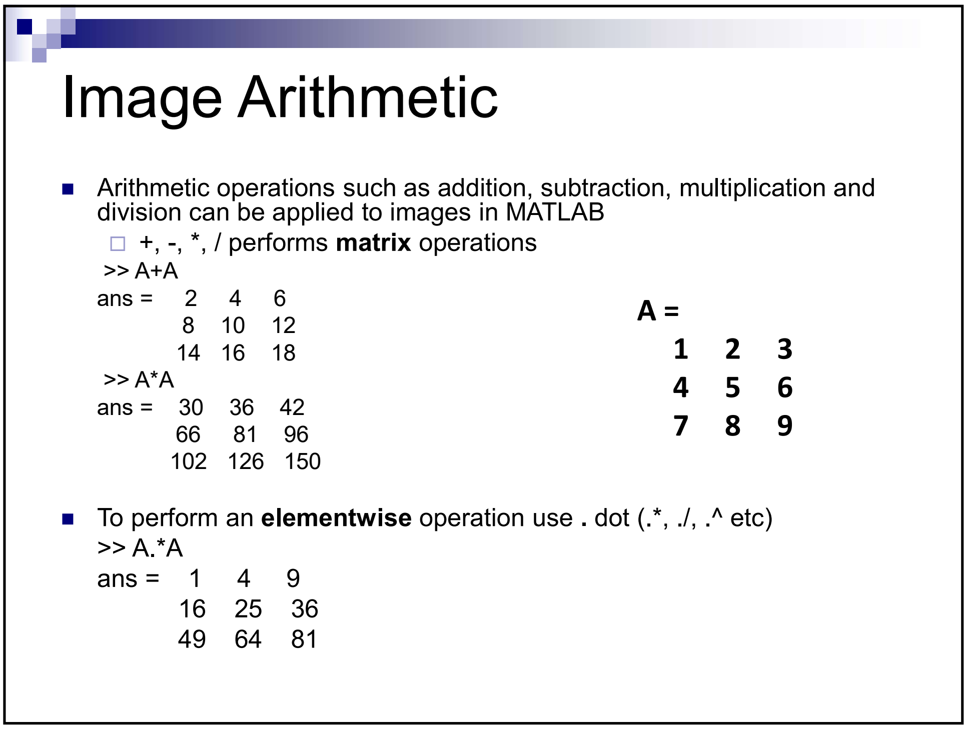


Image Arithmetic

- Arithmetic operations such as addition, subtraction, multiplication and division can be applied to images in MATLAB
  - □ +, -, *, / performs **matrix** operations

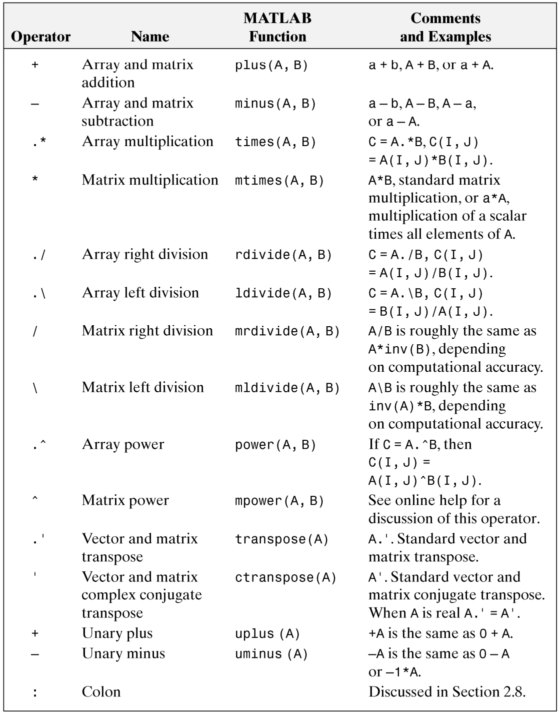
```
 >> A+A
ans =    2    4    6
         8   10   12
        14   16   18
 >> A*A
ans =   30   36   42
        66   81   96
       102  126  150
```

A =

```
  1   2   3
  4   5   6
  7   8   9
```

- To perform an **elementwise** operation use **.** dot (.*, ./, .^ etc)

```
>> A.*A
ans =   1    4    9
       16   25   36
       49   64   81
```



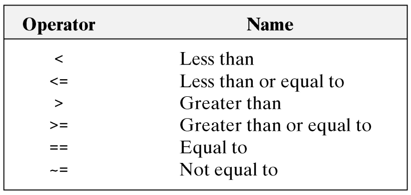
Operator	Name	MATLAB Function	Comments and Examples
+	Array and matrix addition	`plus(A, B)`	a + b, A + B, or a + A.
–	Array and matrix subtraction	`minus(A, B)`	a – b, A – B, A – a, or a – A.
.*	Array multiplication	`times(A, B)`	C = A.*B, C(I, J) = A(I, J)*B(I, J).
*	Matrix multiplication	`mtimes(A, B)`	A*B, standard matrix multiplication, or a*A, multiplication of a scalar times all elements of A.
./	Array right division	`rdivide(A, B)`	C = A./B, C(I, J) = A(I, J)/B(I, J).
.\	Array left division	`ldivide(A, B)`	C = A.\B, C(I, J) = B(I, J)/A(I, J).
/	Matrix right division	`mrdivide(A, B)`	A/B is roughly the same as A*inv(B), depending on computational accuracy.
\	Matrix left division	`mldivide(A, B)`	A\B is roughly the same as inv(A)*B, depending on computational accuracy.
.^	Array power	`power(A, B)`	If C = A.^B, then C(I, J) = A(I, J)^B(I, J).
^	Matrix power	`mpower(A, B)`	See online help for a discussion of this operator.
.'	Vector and matrix transpose	`transpose(A)`	A.'. Standard vector and matrix transpose.
'	Vector and matrix complex conjugate transpose	`ctranspose(A)`	A'. Standard vector and matrix conjugate transpose. When A is real A.' = A'.
+	Unary plus	`uplus (A)`	+A is the same as 0 + A.
–	Unary minus	`uminus (A)`	–A is the same as 0 – A or –1*A.
:	Colon		Discussed in Section 2.8.

TABLE 2.4 Array and matrix arithmetic operators. Computations involving these operators can be implemented using the operators themselves, as in A + B, or using the MATLAB functions shown, as in plus (A, B). The examples shown for arrays use matrices to simplify the notation, but they are easily extendable to higher dimensions.

Relational and Logical Operators

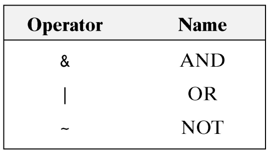
```
A =
     1   2   3
     4   5   6
     7   8   9
```

TABLE 2.6 Relational operators.



Operator	Name
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

TABLE 2.7 Logical operators.



Operator	Name
&	AND
|	OR
~	NOT

```
>> A==5
        ans =
        3×3 logical array
              0    0    0
              0    1    0
              0    0    0
>> A<5
        ans = 1    1    1
              1    0    0
              0    0    0
>> A~=5
        ans = 1    1    1
              1    0    1
              1    1    1

>> result = A>=3 & A<7
        result =
              0    0    1
              1    1    1
              0    0    0
```

Logical Conditions

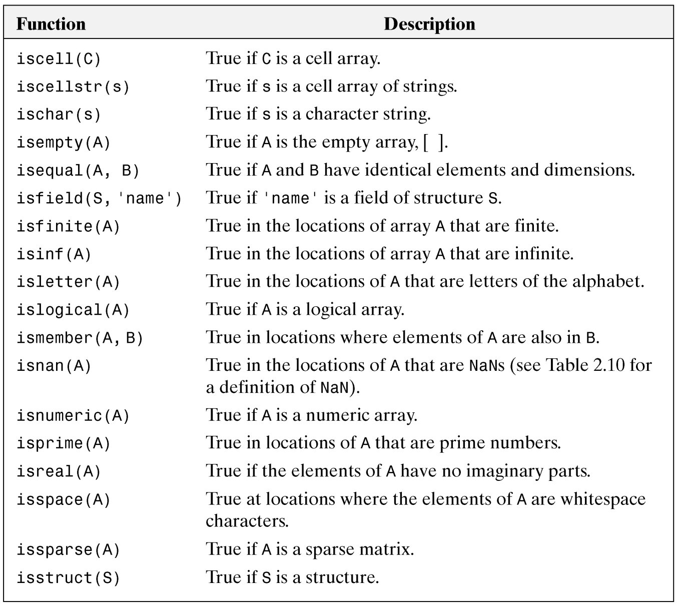
- find('condition') - Returns indexes of A's elements that satisfies the condition.

```
>> [row col] = find(A==7)
row =   3
col =   1
>> [row col] = find(A>7)
row =   3
        3
col =   2
        3
>> indx = find(A<5)
indx =  1
        2
        4
        7
```

A =

```
1   2   3
4   5   6
7   8   9
```

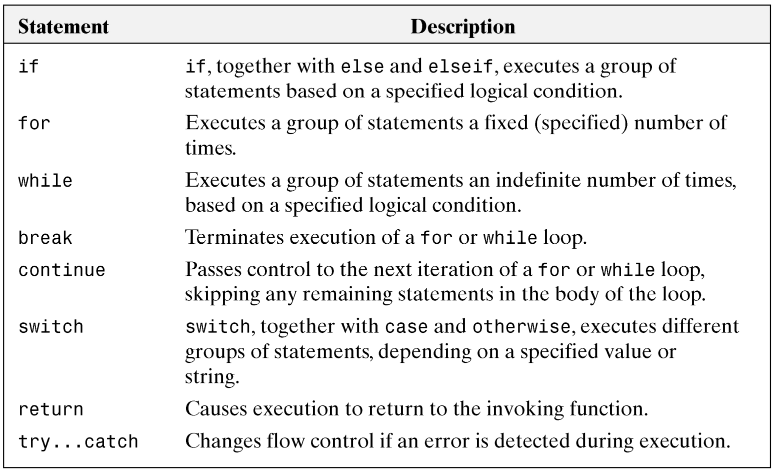
Conditional Test Functions



Function	Description
`iscell(C)`	True if `C` is a cell array.
`iscellstr(s)`	True if `s` is a cell array of strings.
`ischar(s)`	True if `s` is a character string.
`isempty(A)`	True if `A` is the empty array, [ ].
`isequal(A, B)`	True if `A` and `B` have identical elements and dimensions.
`isfield(S, 'name')`	True if `'name'` is a field of structure `S`.
`isfinite(A)`	True in the locations of array `A` that are finite.
`isinf(A)`	True in the locations of array `A` that are infinite.
`isletter(A)`	True in the locations of `A` that are letters of the alphabet.
`islogical(A)`	True if `A` is a logical array.
`ismember(A, B)`	True in locations where elements of `A` are also in `B`.
`isnan(A)`	True in the locations of `A` that are `NaN`s (see Table 2.10 for a definition of `NaN`).
`isnumeric(A)`	True if `A` is a numeric array.
`isprime(A)`	True in locations of `A` that are prime numbers.
`isreal(A)`	True if the elements of `A` have no imaginary parts.
`isspace(A)`	True at locations where the elements of `A` are whitespace characters.
`issparse(A)`	True if `A` is a sparse matrix.
`isstruct(S)`	True if `S` is a structure.

TABLE 2.9
Some functions that return a logical 1 or a logical 0 depending on whether the value or condition in their arguments are `true` or `false`. See online help for a complete list.

Flow Control



Statement	Description
`if`	`if`, together with `else` and `elseif`, executes a group of statements based on a specified logical condition.
`for`	Executes a group of statements a fixed (specified) number of times.
`while`	Executes a group of statements an indefinite number of times, based on a specified logical condition.
`break`	Terminates execution of a `for` or `while` loop.
`continue`	Passes control to the next iteration of a `for` or `while` loop, skipping any remaining statements in the body of the loop.
`switch`	`switch`, together with `case` and `otherwise`, executes different groups of statements, depending on a specified value or string.
`return`	Causes execution to return to the invoking function.
`try...catch`	Changes flow control if an error is detected during execution.

TABLE 2.11 Flow control statements.

Flow Control

- Flow control in MATLAB

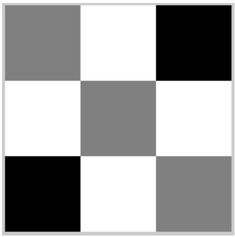
  - for loops

```
A = zeros(3);
for row = 1 : 3
     for col = 1 : 3
            if row == col
                   A(row, col) = 1;
            elseif abs(row - col) == 1
                   A(row, col) = 2;
            else
                   A(row, col) = 0;
            end
     end
end
```

```
A =

   1   2   0
   2   1   2
   0   2   1
```

```
>> imshow(A,[ ]);
```



Flow Control

- while, expression, statements, end

```
indx=1;
while A(indx)<6
     A(indx)=0;
     indx=indx+1;
 end
```

```
A =
    0    2    3
    0    5    6
    7    8    9
```

```
A =
   1   2   3
   4   5   6
   7   8   9
```

Working with M-Files

- M-files can be ***scripts*** that simply execute a series of MATLAB statements, or they can be ***functions*** that also accept input arguments and produce output.
- MATLAB functions:
  - □ Are useful for extending the MATLAB language for your application.
  - □ Can accept input arguments and return output arguments.
  - □ Store variables in a workspace internal to the function.

Scripts and Functions

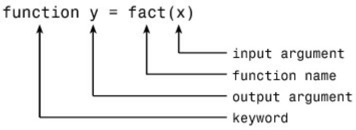
There are two kinds of M-files(.m):

- **Scripts**, do not accept input arguments or return output arguments. They operate on data in the workspace. Equivalent to typing into the command window. What we have written up to now were all scripts.
- **Functions**, can accept input arguments and return output arguments. Internal variables are local to the function.

Working with M-Files

- Create a new empty m-file

```
function A=test(I)
[row col]=size(I);
for r=1:row
        for c=1:col
                    if r==c
                                A(r, c)=1;
                    elseif abs(r-c)==1
                                A(r, c)=2;
                    else
                                A(r, c)=0;
                    end
        end
end
```



function y = fact(x)

input argument

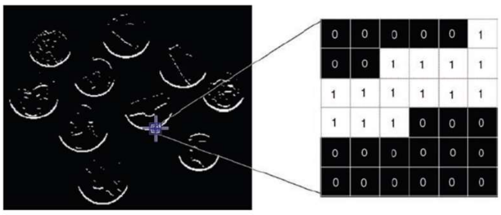
function name

output argument

keyword

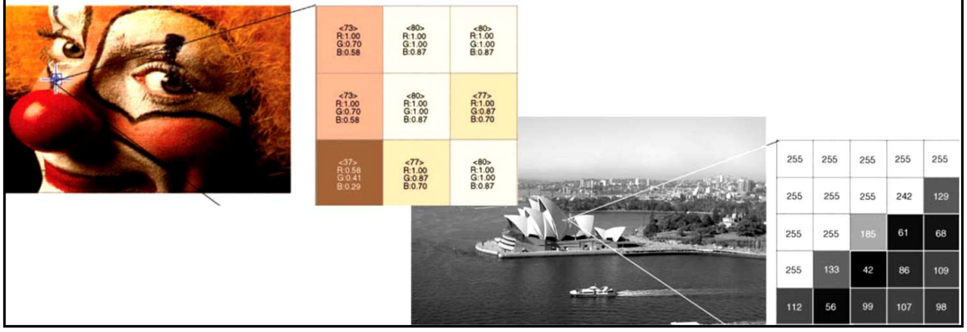
Images in MATLAB

- **Binary** images : {**0**,**1**}
- **Intensity** images : **[0,255] uint8** or **[0,1] double**
- **RGB** images : **m**-by-**n**-by-**3**
- Multidimensional images **m**-by-**n**-by-**p** (p is the number of layers)



0	0	0	0	0	1
0	0	1	1	1	1
1	1	1	1	1	1
1	1	1	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

FIGURE 2.2 A binary image and the pixel values in a 6 × 6 neighborhood. Original image: courtesy of MathWorks.



<73> R:1.00 G:0.70 B:0.58	<80> R:1.00 G:1.00 B:0.87	<80> R:1.00 G:1.00 B:0.87
<73> R:1.00 G:0.70 B:0.58	<80> R:1.00 G:1.00 B:0.87	<77> R:1.00 G:0.87 B:0.70
<37> R:0.58 G:0.41 B:0.29	<77> R:1.00 G:0.87 B:0.70	<80> R:1.00 G:1.00 B:0.87

255	255	255	255	255
255	255	255	242	129
255	255	185	61	68
255	133	42	86	109
112	56	99	107	98

Loading and displaying images

```
>> f = imread('peppers.png');   % load image
```

Matrix with image data

image filename as a string

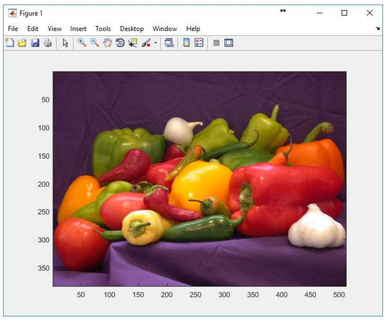
```
>> imshow(f)   % display image
>> whos f

Name         Size              Bytes      Class

f            384x512x3         589824     uint8
Grand total is 589824 elements using 589824 bytes
```

Dimensions of f

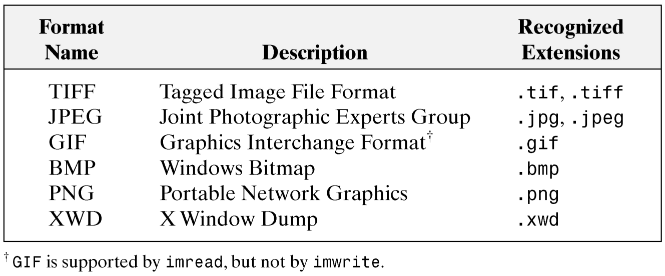
Uint8 is not appropriate for arithmetic operations! Why?



```
>> size(f)
ans =
   384   512     3
```

```
>> [row, col, chan] = size(f)
row =
   384
col =
   512
chan =
     3
```

Image File Types



Format Name	Description	Recognized Extensions
TIFF	Tagged Image File Format	.tif, .tiff
JPEG	Joint Photographic Experts Group	.jpg, .jpeg
GIF	Graphics Interchange Format[†]	.gif
BMP	Windows Bitmap	.bmp
PNG	Portable Network Graphics	.png
XWD	X Window Dump	.xwd

[†] GIF is supported by imread, but not by imwrite.

TABLE 2.1 Some of the image/graphics formats supported by imread and imwrite, starting with MATLAB 6.5. Earlier versions support a subset of these formats. See online help for a complete list of supported formats.

We can also indicate the full path of the image file to be read using:

```
>> f = imread('C:\images\apple.jpg');        % load image from path
```

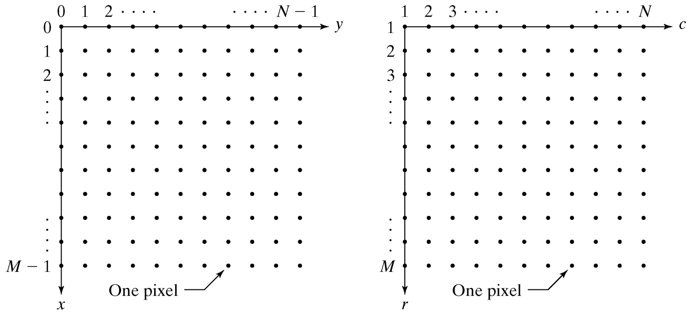
Image File Information

- Listing the file information of an image file:

```
>> imfinfo office_5.jpg
ans =
        Filename: 'C:\Program Files\MATLAB\R2014a\toolbox\images\...'
        FileModDate: '25-Sep-2013 19:12:04'
        FileSize: 146638
        Format: 'jpg'
        FormatVersion: ''
        Width: 903
        Height: 600
        BitDepth: 24
        ColorType: 'truecolor'
        FormatSignature: ''
        NumberOfSamples: 3
        CodingMethod: 'Huffman'
        CodingProcess: 'Sequential'
```

Image Coordinates

- Assume an image with M rows and N columns
- Coordinate conventions may vary mostly among the following two ways (In Matlab we use the one on the right)
- The coordinates are given as (row,column)
- The origin is at (1,1)



a b

FIGURE 2.1 Coordinate conventions used (a) in many image processing books, and (b) in the Image Processing Toolbox.

Finding Image Size

- Function '***size***' gives the **row**, **column** and **channel** dimentions of an image.
- Assume **f** is a 512x384 color(RGB) image

```
>> size(f)
   384   512     3
>> [M, N, C] = size(f);
   M=384
   N=512
   C=3
```

Displaying Images

- Opening a new figure window:

  >> figure;

- Images are displayed in MATLAB using function **imshow**, which has the basic syntax:

  >> imshow(f)

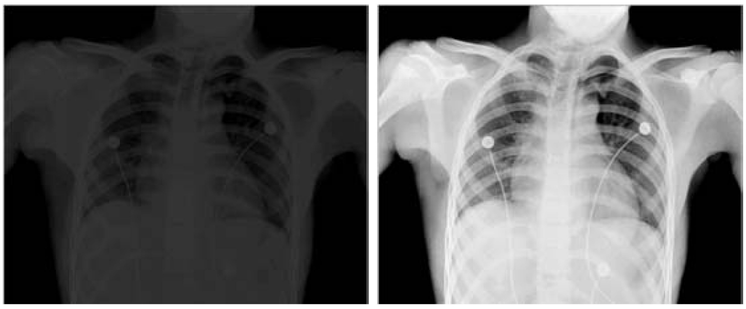
  shows image f according to the **variable type** of **f**

  (If **f** is "**uint8**" then **255→White**, If **f** is "**double**" then **1→White**)

  >> imshow(f,[low high])

- Displays as **black** all values less than or equal to '***low***' and as **white** all values greater than or equal to '***high***'

Displaying Images



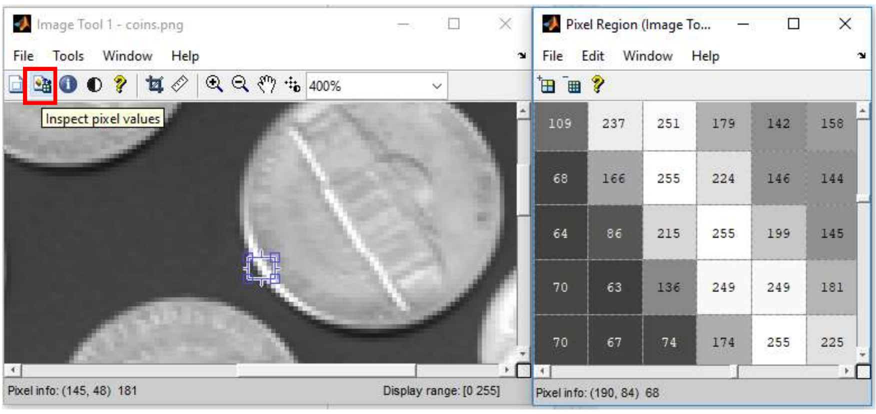
a b

FIGURE 2.3 (a) An image, h, with low dynamic range. (b) Result of scaling by using `imshow(h,[])`. (Original image courtesy of Dr. David R. Pickens, Dept. of Radiology & Radiological Sciences, Vanderbilt University Medical Center.)

>> imshow(f,[])

- □ Sets variable '*low*' to the **minimum** value of 'f' and '*high*' to its **maximum**

Displaying Images



109	237	251	179	142	158
68	166	255	224	146	144
64	86	215	255	199	145
70	63	136	249	249	181
70	67	74	174	255	225

Pixel info: (145, 48) 181

Display range: [0 255]

Pixel info: (190, 84) 68

```
>> imtool(f);
```

- □ Displays an image and contains a number of associated tools that can be used to explore the image contents

Displaying Images

- **subplot(r,c,n)** creates a figure with **r** x **c** grids and selects the **n**$^{\text{th}}$ cell so that imshow() displays the image on that cell

```
>> subplot(1,2,1);       % subplot(121);
>> imshow(image1);
>> subplot(1,2,2);       % subplot(122);
>> imshow(image2);
>> title('figure title');     % add title to figure
```

image1

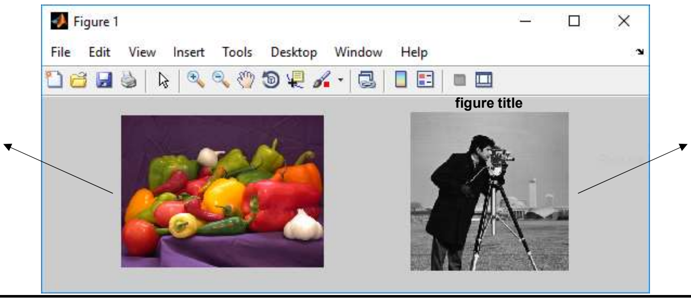
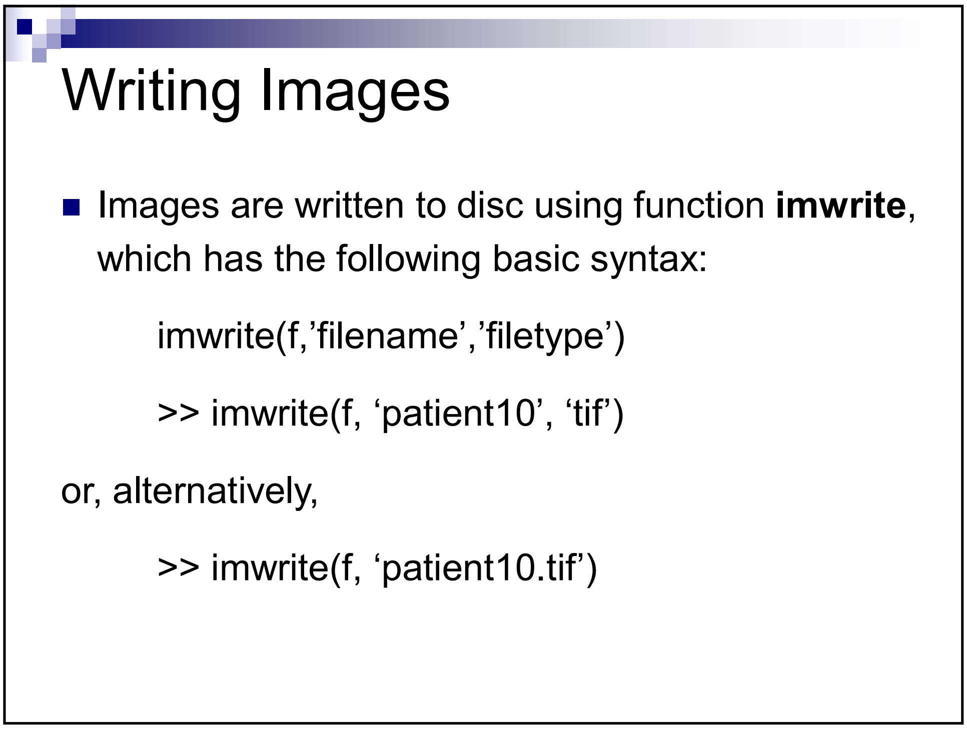


image2



Writing Images

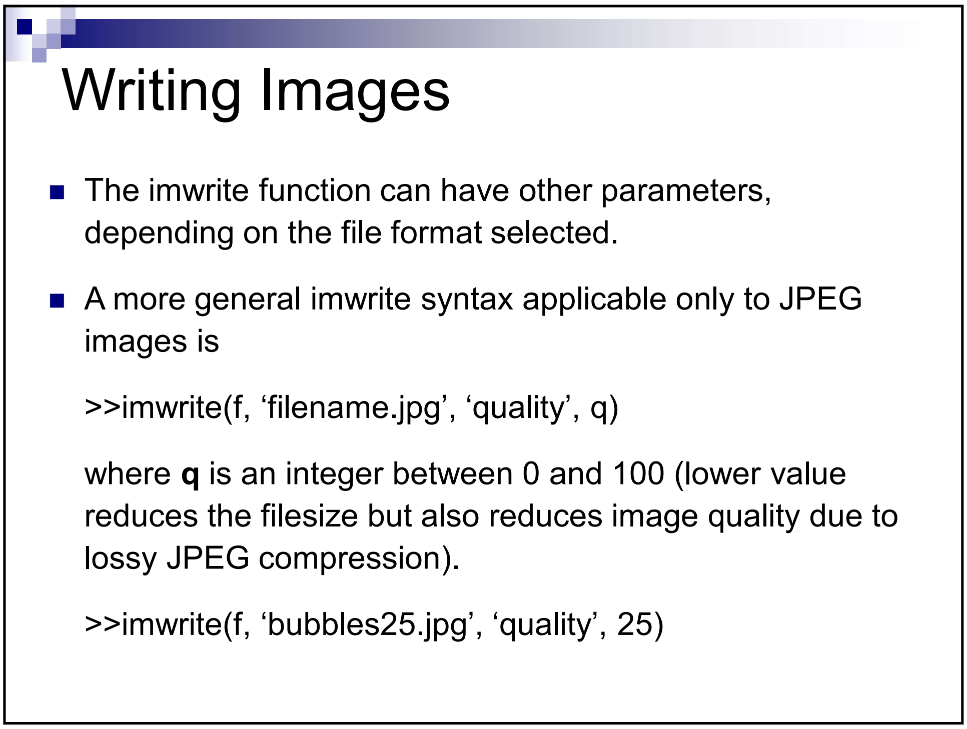
- Images are written to disc using function **imwrite**, which has the following basic syntax:

```
imwrite(f,’filename’,’filetype’)

>> imwrite(f, ‘patient10’, ‘tif’)
```

or, alternatively,

```
>> imwrite(f, ‘patient10.tif’)
```



Writing Images

- The imwrite function can have other parameters, depending on the file format selected.
- A more general imwrite syntax applicable only to JPEG images is

  ```
  >>imwrite(f, ‘filename.jpg’, ‘quality’, q)
  ```

  where **q** is an integer between 0 and 100 (lower value reduces the filesize but also reduces image quality due to lossy JPEG compression).

  ```
  >>imwrite(f, ‘bubbles25.jpg’, ‘quality’, 25)
  ```

Useful functions for manipulating images

- Convert **color** image **f** to **grayscale**:

```
>> fgray = rgb2gray(f);
```

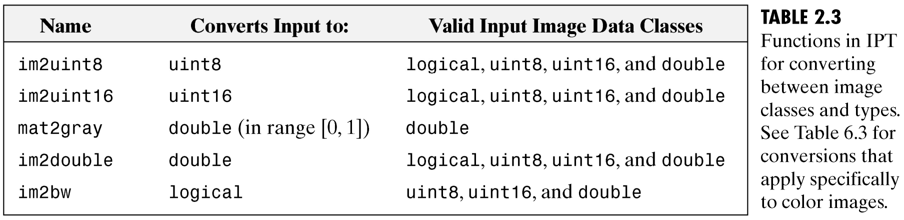
- **Resize** image

```
>> fsmall = imresize(f,[100 100], 'bilinear');
```

- **Rotate** image

```
>> f45 = imrotate(f,45);   % rotates image 45 degrees
```

Converting Image Types

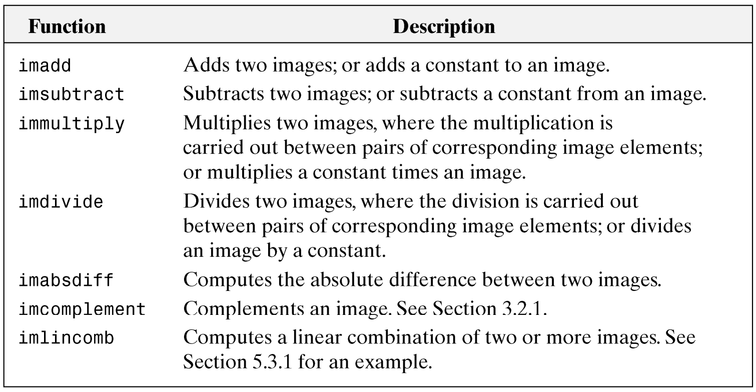


Name	Converts Input to:	Valid Input Image Data Classes
`im2uint8`	`uint8`	`logical`, `uint8`, `uint16`, and `double`
`im2uint16`	`uint16`	`logical`, `uint8`, `uint16`, and `double`
`mat2gray`	`double` (in range [0, 1])	`double`
`im2double`	`double`	`logical`, `uint8`, `uint16`, and `double`
`im2bw`	`logical`	`uint8`, `uint16`, and `double`

TABLE 2.3 Functions in IPT for converting between image classes and types. See Table 6.3 for conversions that apply specifically to color images.

- Use Matlab **help** to learn the usage of the above functions (e.g. write "help im2double" in command window)

Image Arithmetic Functions



Function	Description
`imadd`	Adds two images; or adds a constant to an image.
`imsubtract`	Subtracts two images; or subtracts a constant from an image.
`immultiply`	Multiplies two images, where the multiplication is carried out between pairs of corresponding image elements; or multiplies a constant times an image.
`imdivide`	Divides two images, where the division is carried out between pairs of corresponding image elements; or divides an image by a constant.
`imabsdiff`	Computes the absolute difference between two images.
`imcomplement`	Complements an image. See Section 3.2.1.
`imlincomb`	Computes a linear combination of two or more images. See Section 5.3.1 for an example.

TABLE 2.5 The image arithmetic functions supported by IPT.

- You can also use arithmetic operators +,-,*,/, etc. but above functions execute faster in CPU.

Saving Your Workspace Variables

- `save mysession`

  % creates mysession.mat with all variables
- `save mysession a b`

  % save only variables a and b
- `clear all`

  % clear all variables
- `clear a b`

  % clear variables a and b
- `load mysession`

  % load session

Questions ?