

MATLAB

A Tool for Scientific Computing

Dr. Mahesh Goyani

Assistant Professor

Department of Computer Engineering

Government Engineering College, Modasa

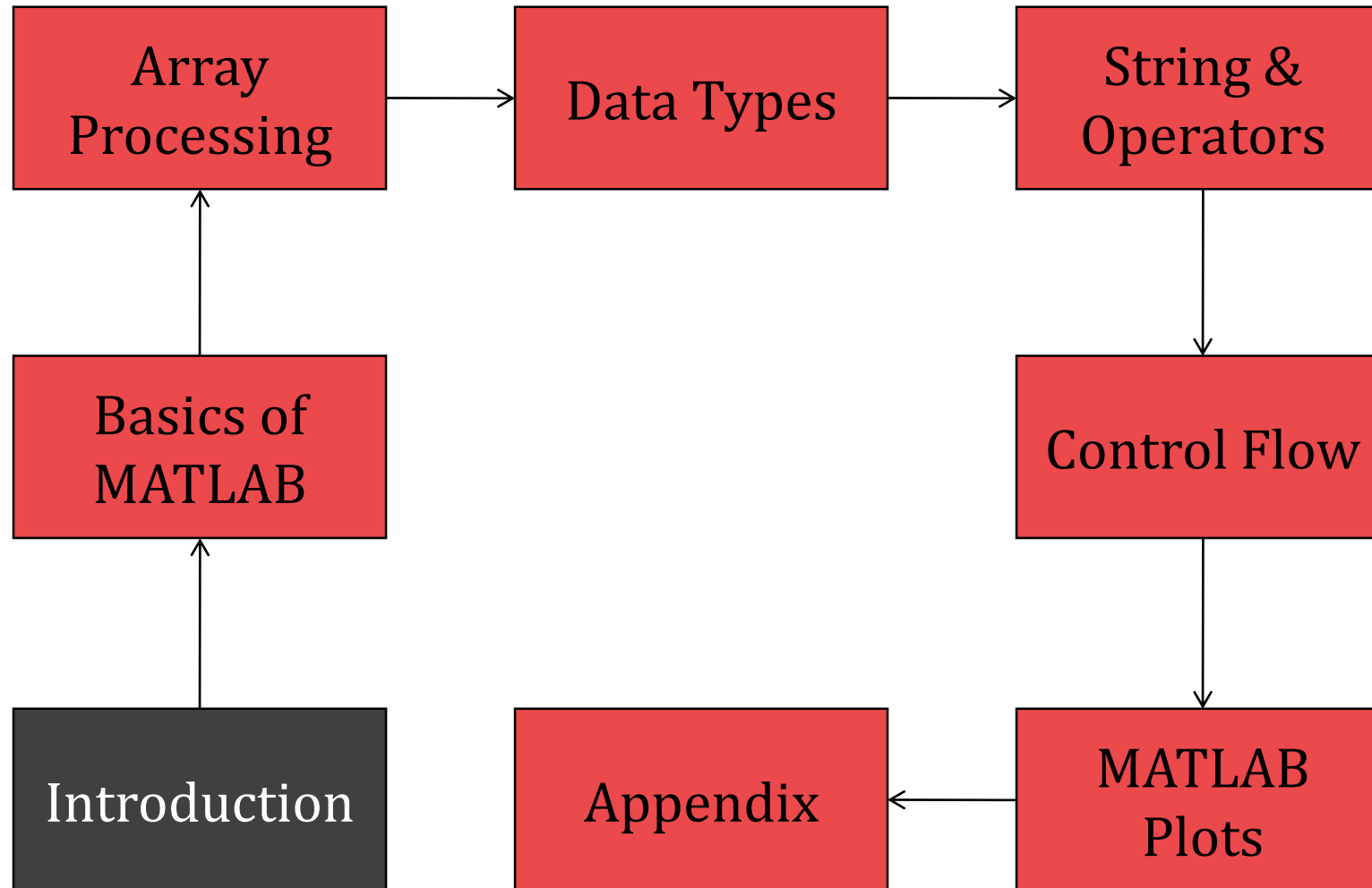
E-Mail: mgoyani@gmail.com

URL: www.maheshgoyani.in

PREREQUISITE

- ◇ Basics of programming language
- ◇ Coding skill in any of the following languages:
 - ◇ C
 - ◇ C++
 - ◇ JAVA
- ◇ And...
 - ◇ *Obviously A Good Logic !! 😊*

OPERATIONAL PIPELINE



MATLAB

- ❖ MATLAB stands for **MAT**rix **LAB**oratory
- ❖ Developed by Cleve Moler in 1970's to help students to learn linear algebra.
- ❖ MathWorks Inc company founded in 1984 for commercial development
- ❖ MATLAB is interpreted – code based system
- ❖ Basic data element of MATLAB is an *Array / Matrix* that does not require dimensioning.
- ❖ Add-on toolboxes for extra power.

- ◆ Used to model, analyze and simulate dynamic systems using block diagrams.
- ◆ Fully integrated with MATLAB , easy and fast to learn and flexible.
- ◆ It has comprehensive block library which can be used to simulate linear, non-linear or discrete systems – excellent research tools.
- ◆ C codes can be generated from Simulink models for embedded applications and rapid prototyping of control systems.

TOOLBOXES

◇ Collections of functions to solve problems from several application fields.

◇ Digital Signal Processing Toolbox

◇ Image Toolbox

◇ Wavelet Toolbox

◇ Neural Network Toolbox

◇ Fuzzy Logic Toolbox

◇ Control Toolbox

◇ Multibody Simulation Toolbox

◇ And many others...

WHY MATLAB?

- ◆ Add-on toolboxes (60+ Toolboxes) for extra power.
- ◆ It frees you from coding in high-level languages
- ◆ Extensive HELP documentation
- ◆ Provides professional looking graphs
- ◆ Easy vector and matrix manipulation
- ◆ *Implicit loops for arrays*
- ◆ MATLAB M-files are completely portable across a wide range of platforms.

PROS & CONS

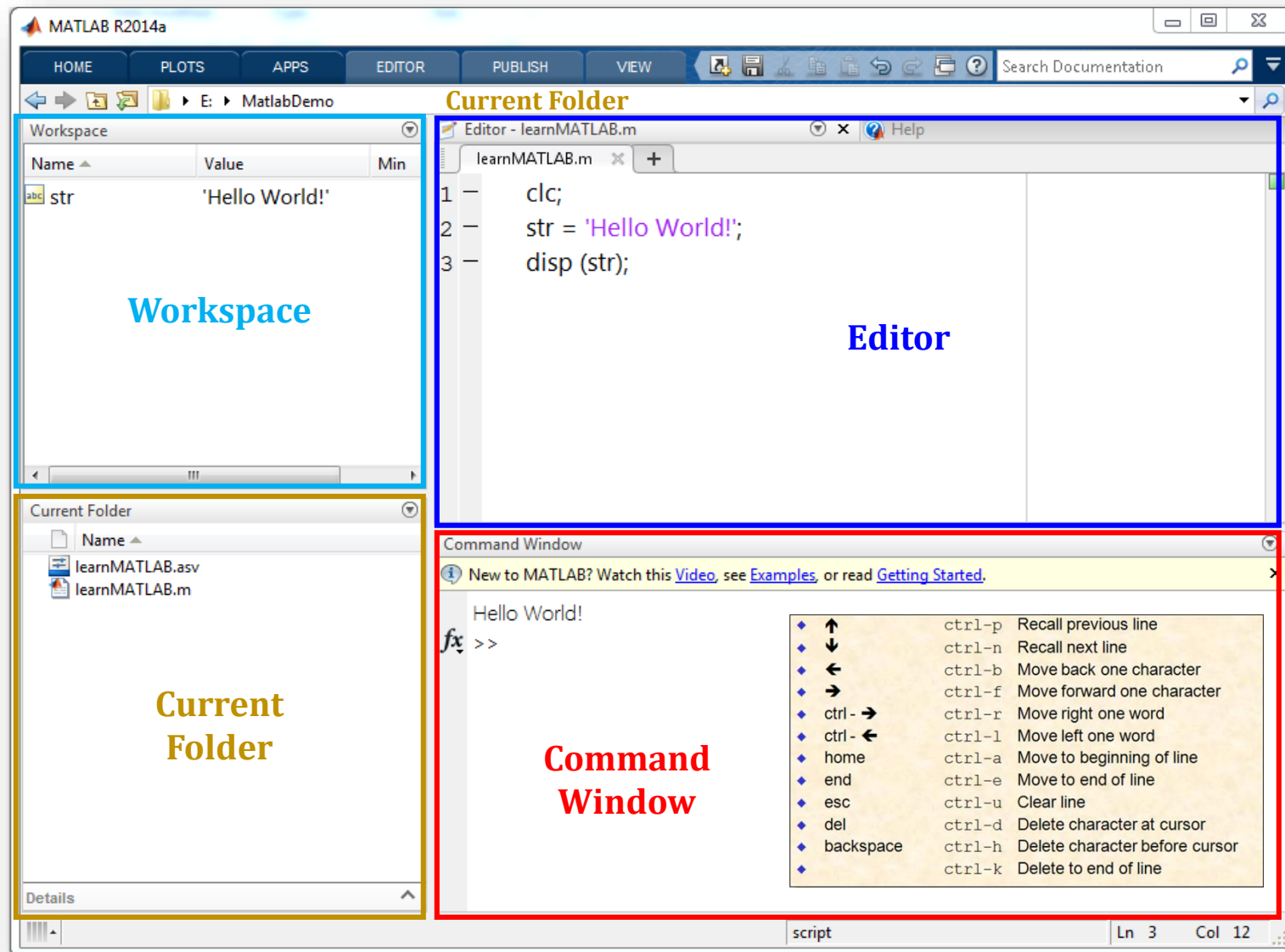
Advantages

- ◆ Ease of use
- ◆ Platform independent
- ◆ Predefined functions
- ◆ Parallelism
- ◆ Professional Plotting

Disadvantages

- ◆ Can be slow
- ◆ Not geared to the Web
- ◆ Expensive
- ◆ Not designed for large scale system development

MATLAB INTERFACE



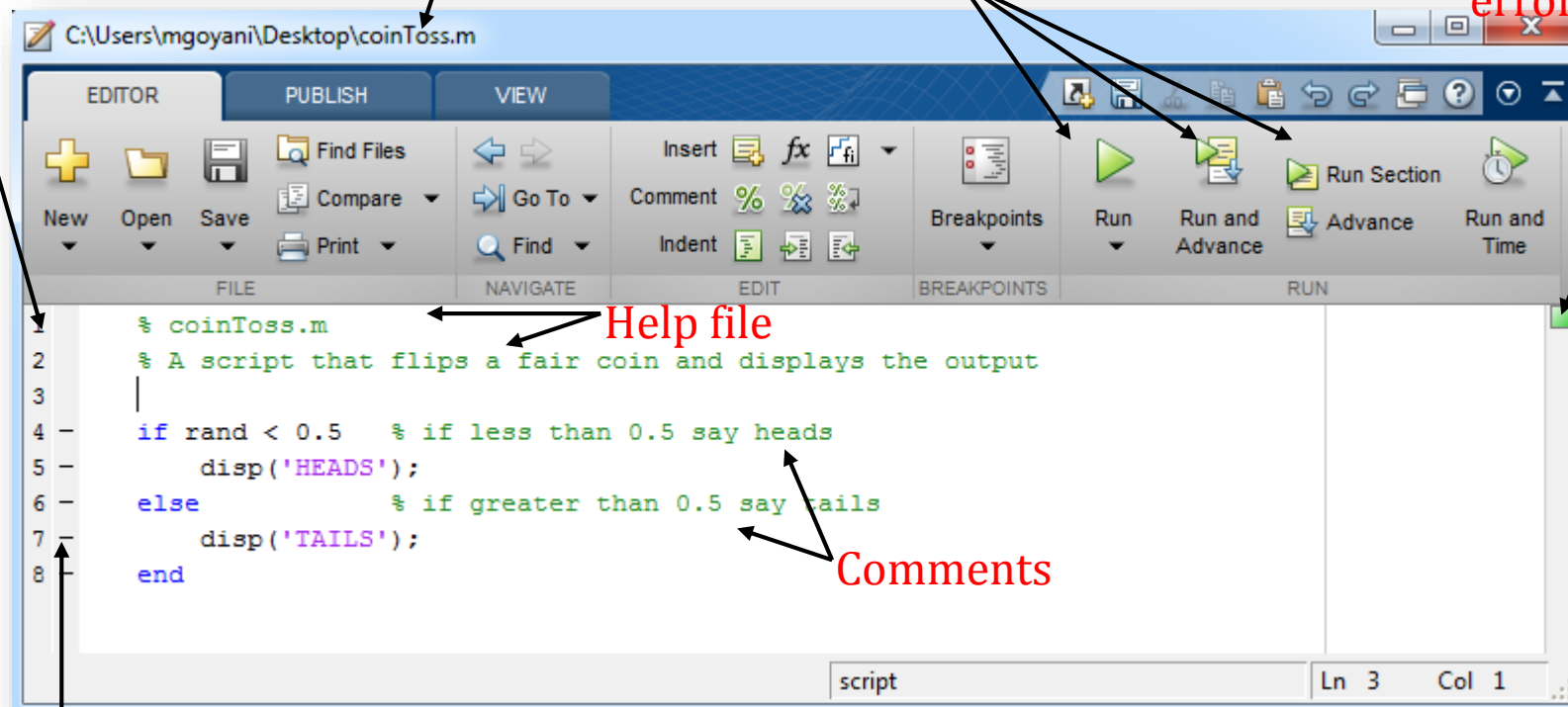
EDITOR

Line numbers

m-file path

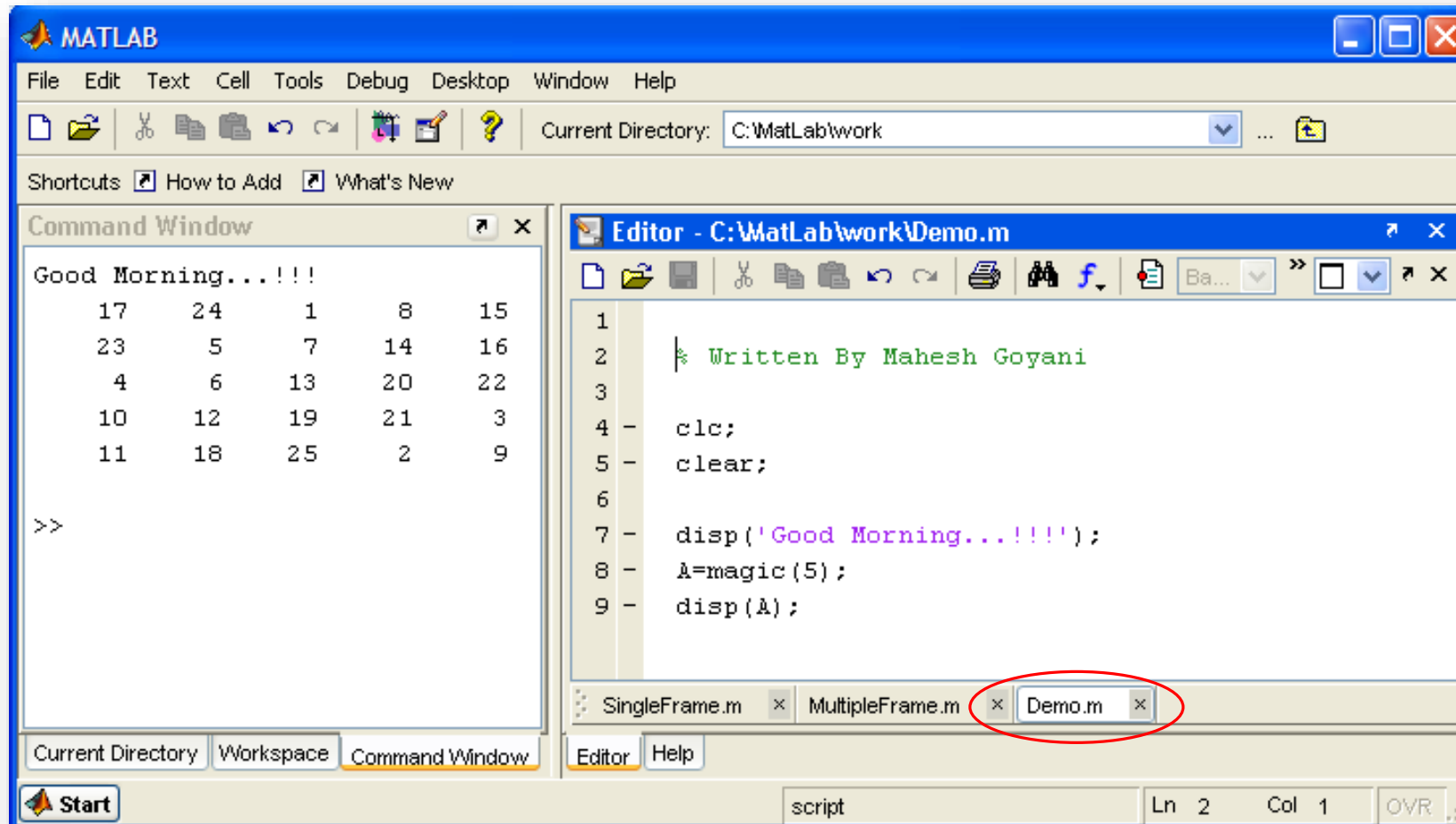
Debugging tools

Real-time
error check



Possible breakpoints

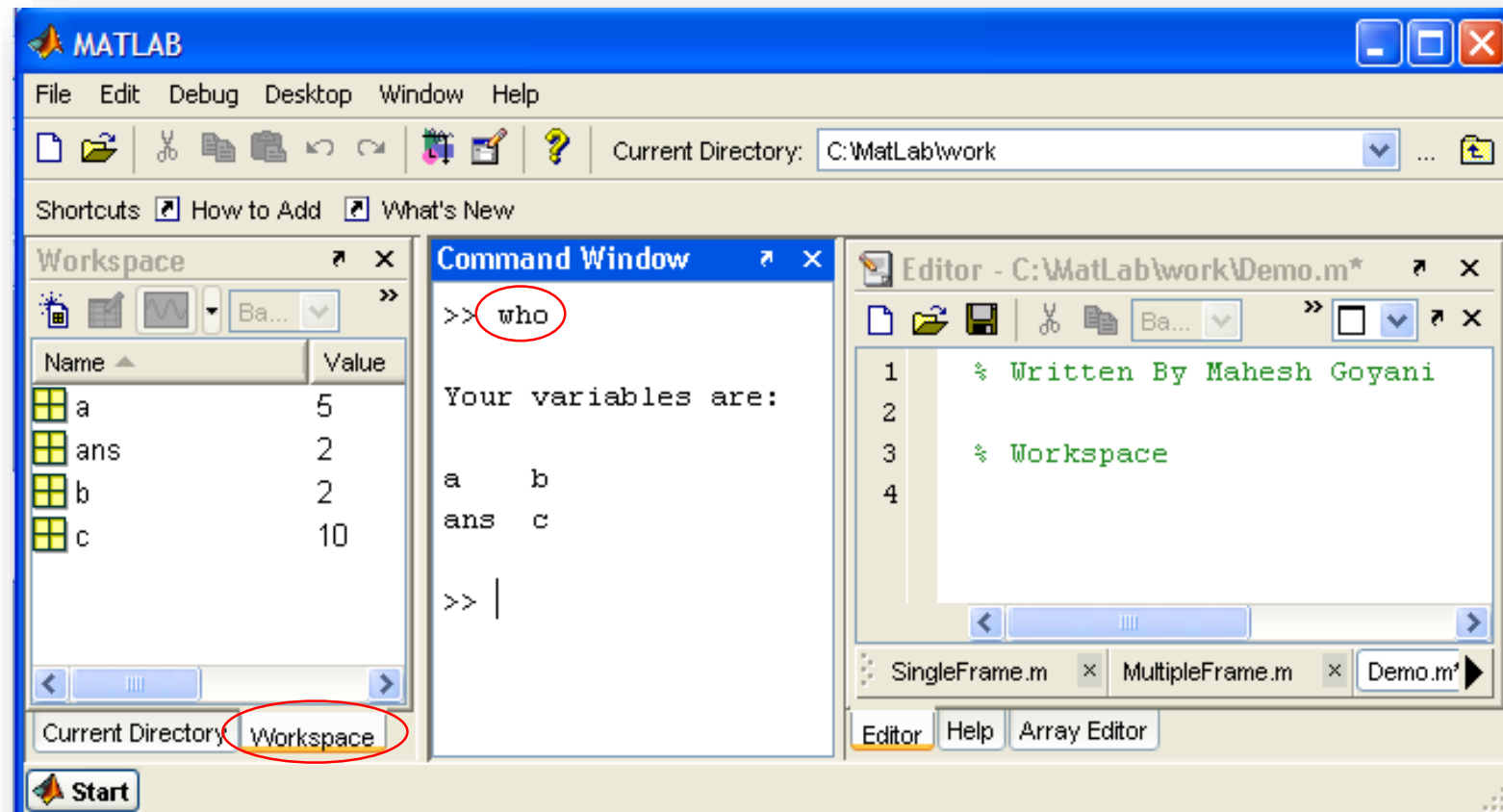
COMMAND WINDOW



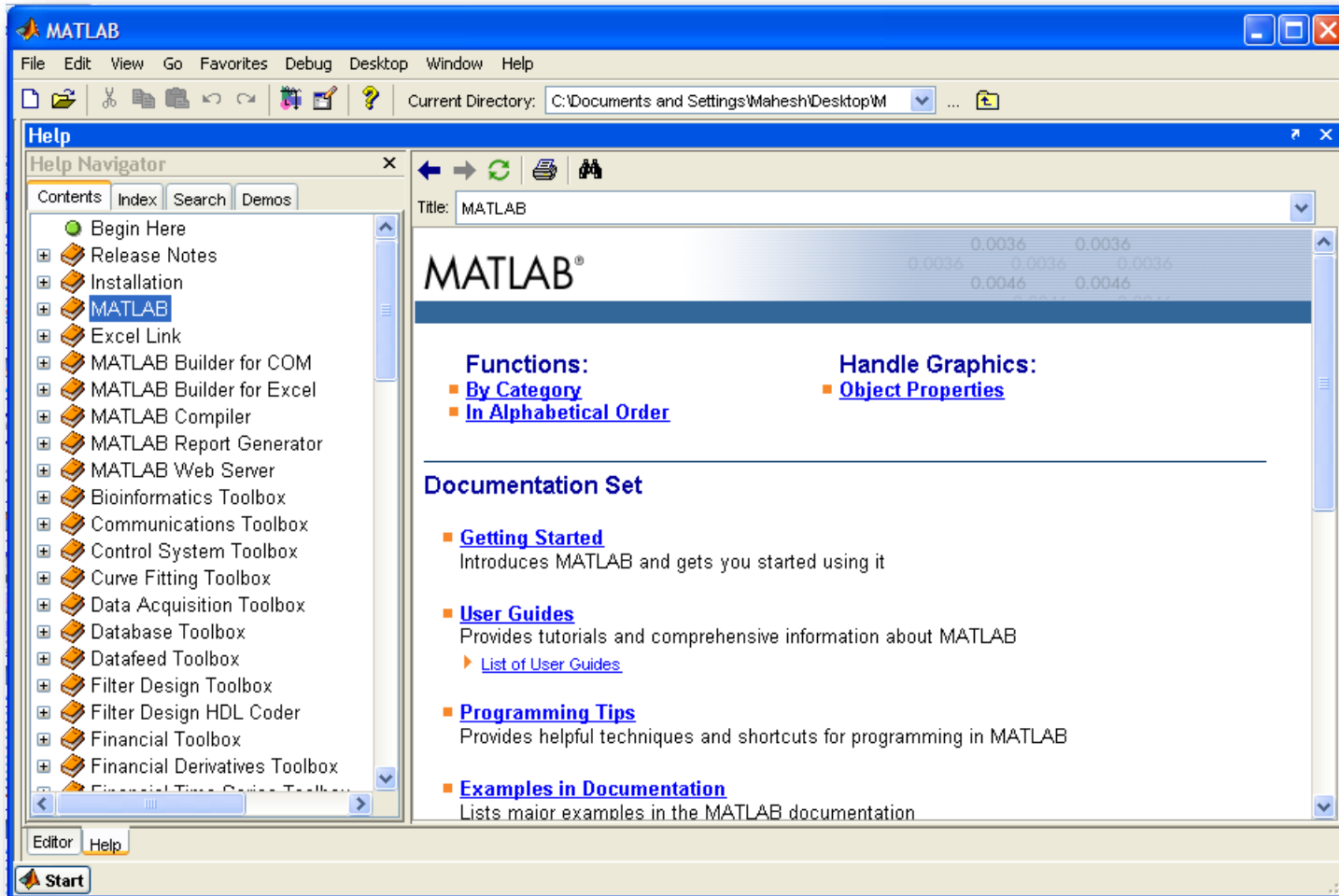
Press **F5** to Run

WORKSPACE

- ❖ Variables are saved in workspace
- ❖ Command *who* retrieves the variables available in workspace

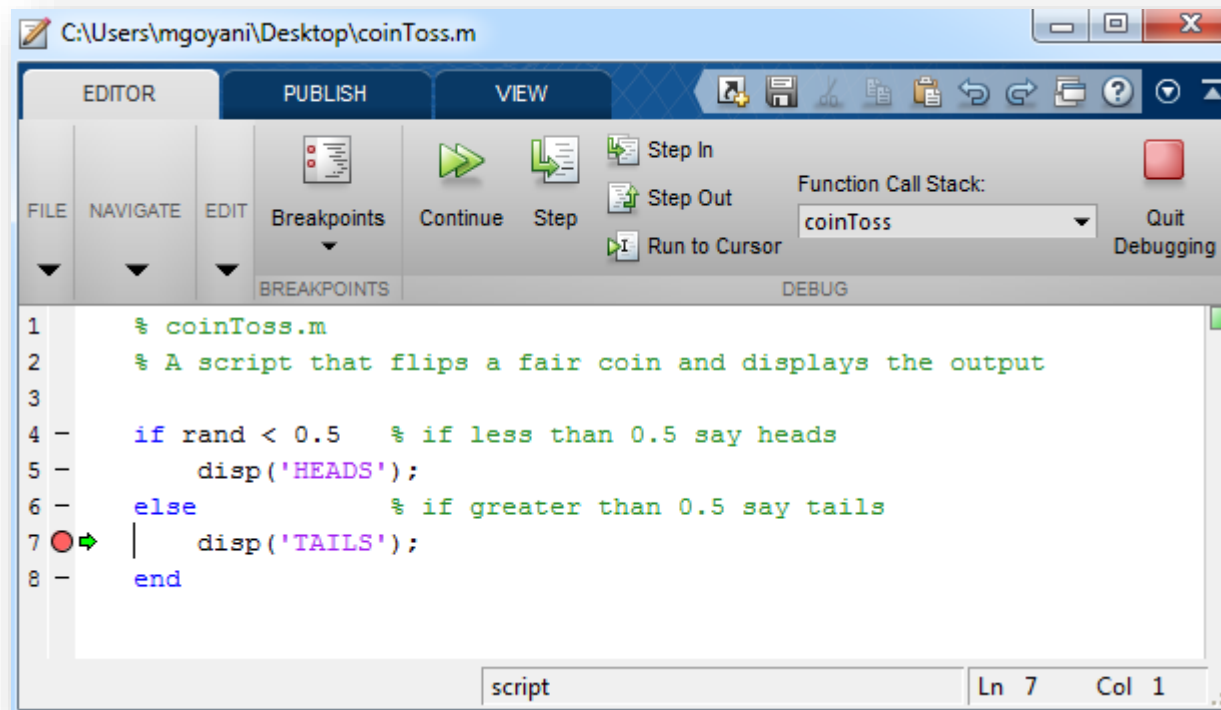


HELP

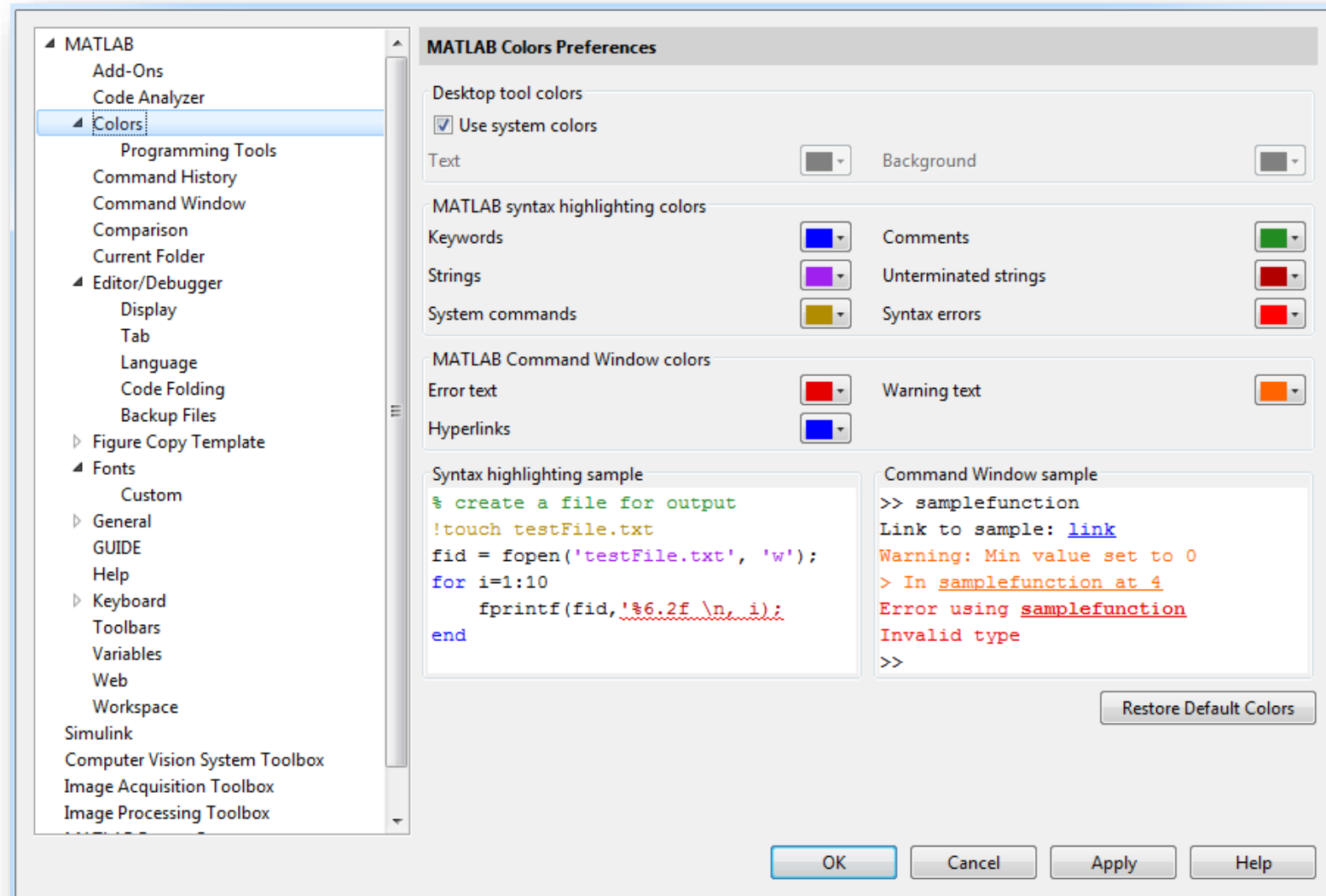


- ◆ help command (>>help)
- ◆ lookfor command (>>lookfor)
- ◆ Help Browser (>>doc)
- ◆ helpwin command (>>helpwin)
- ◆ Search Engine
- ◆ Printable Documents
- ◆ “Matlabroot\help\pdf_doc\”
- ◆ Link to The MathWorks

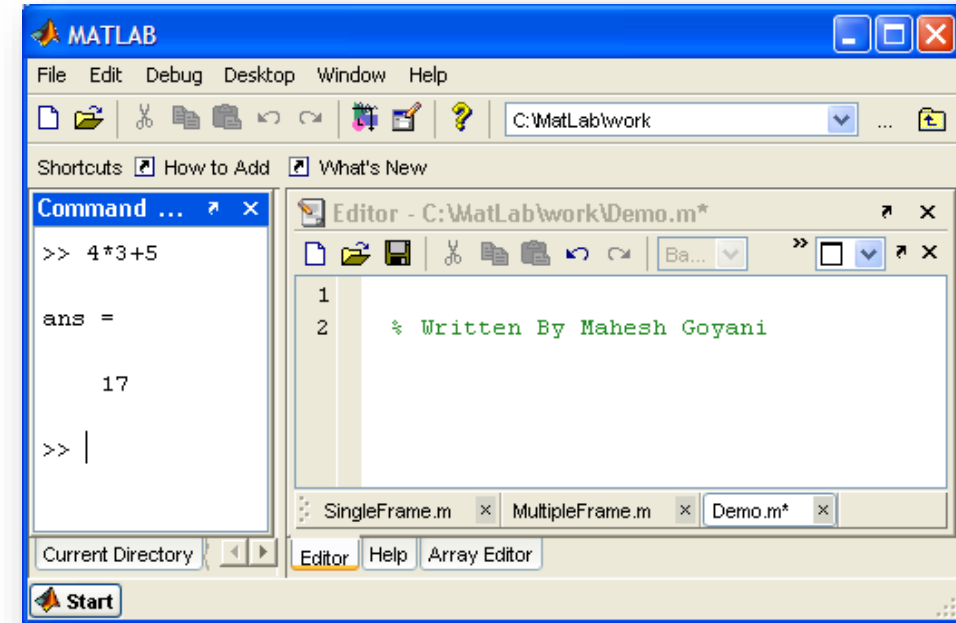
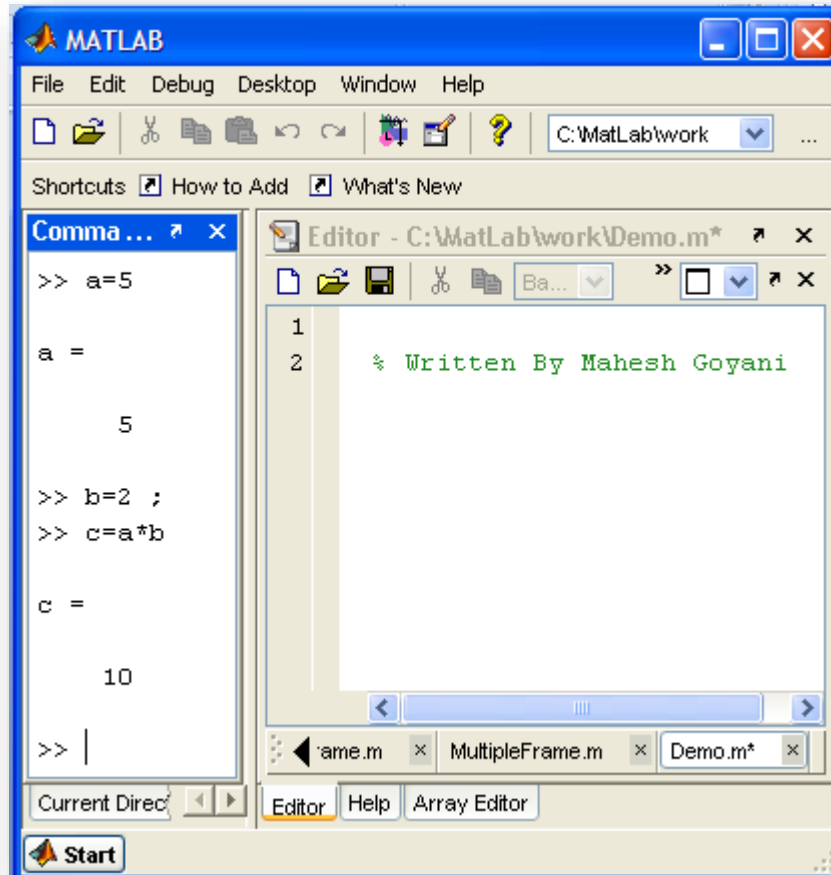
VISUAL DEBUGGING



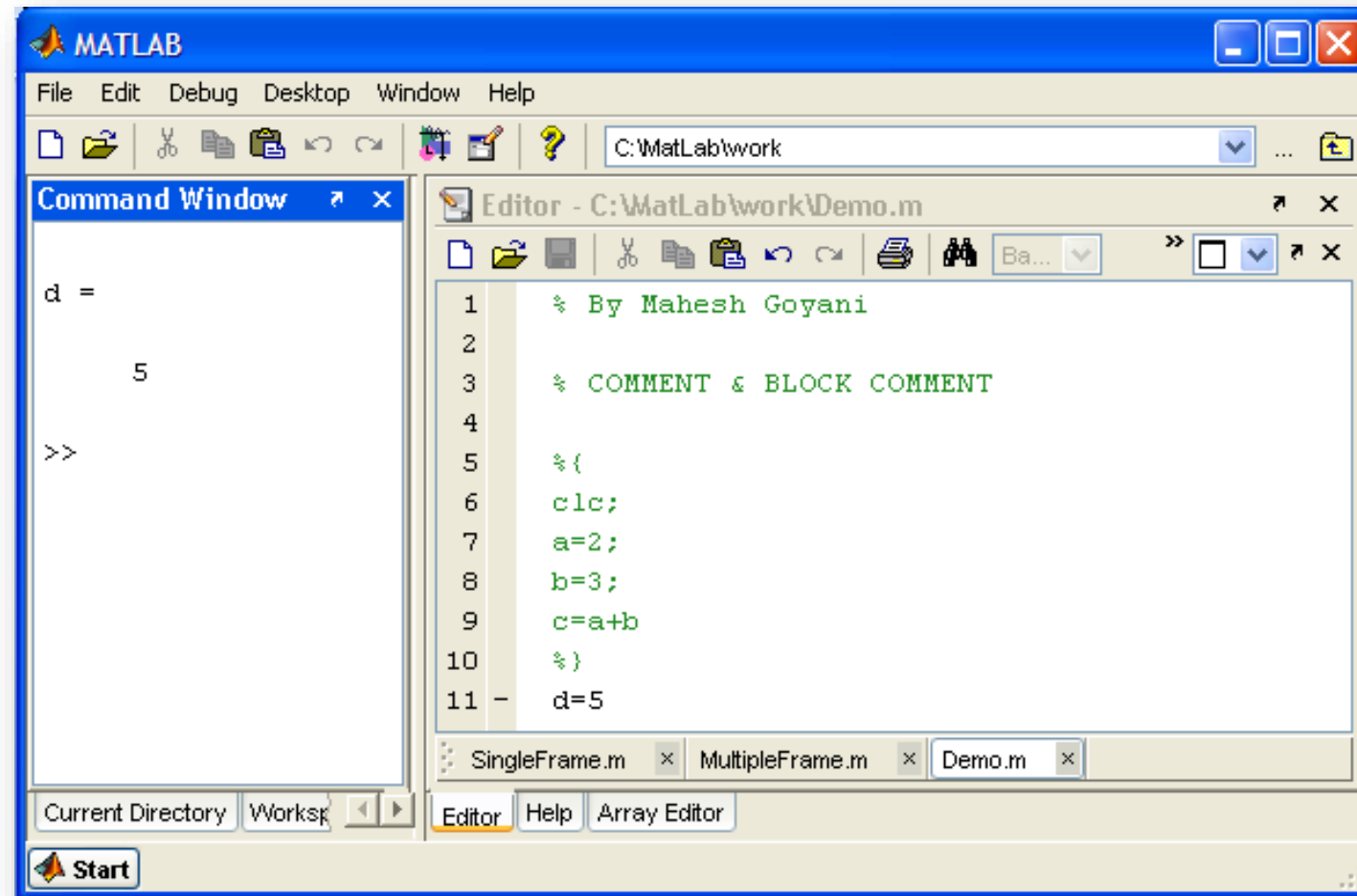
CUSTOMIZATION



BASIC OPERATION



COMMENTS



MANAGING WORKSPACE

◆ MATLAB allocates contiguous memory to variables.

◆ `clc`

◆ `clear a b c*`

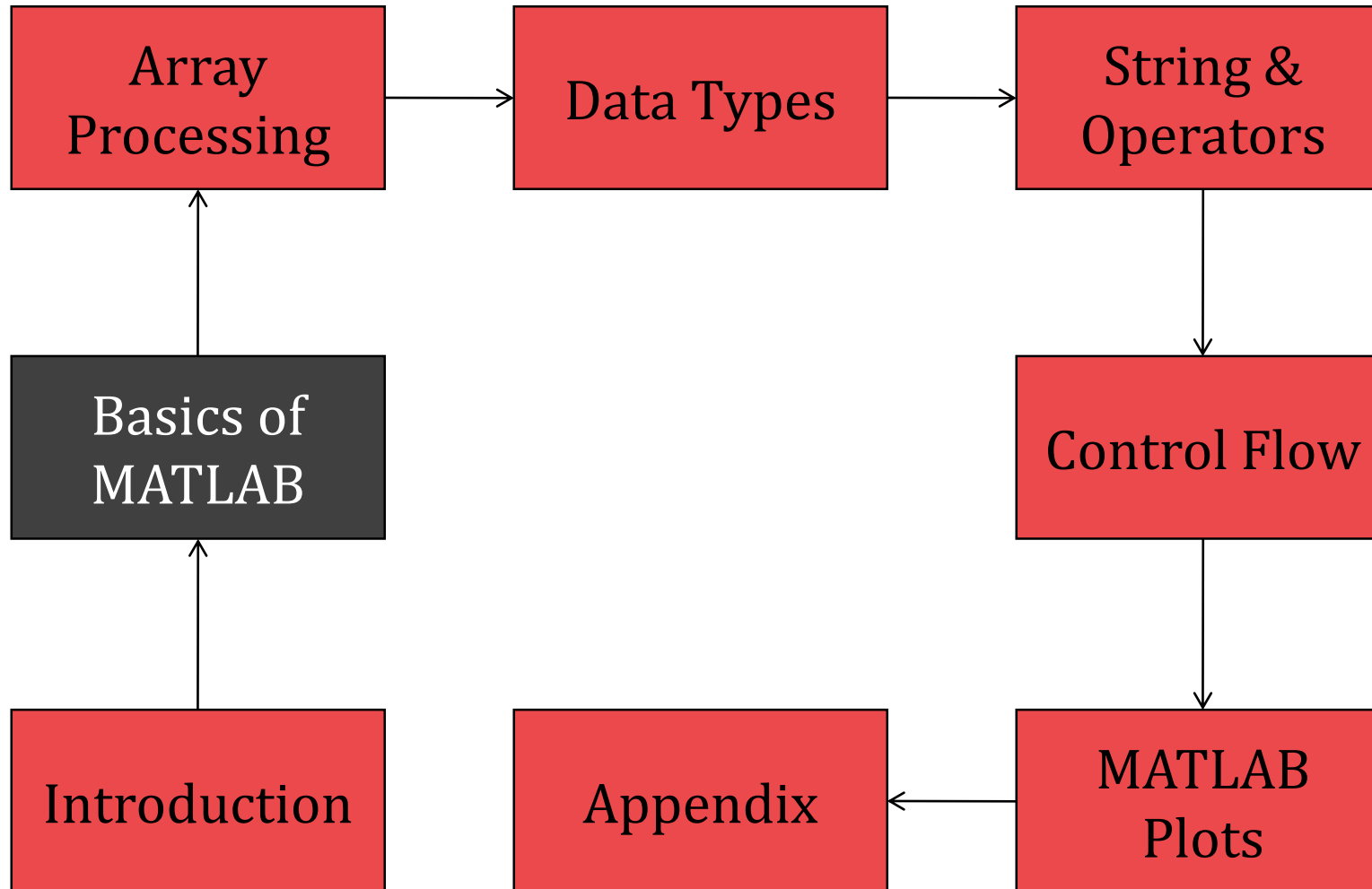
◆ `who`

◆ `whos`

◆ `save`

◆ `pack`

OPERATIONAL PIPELINE

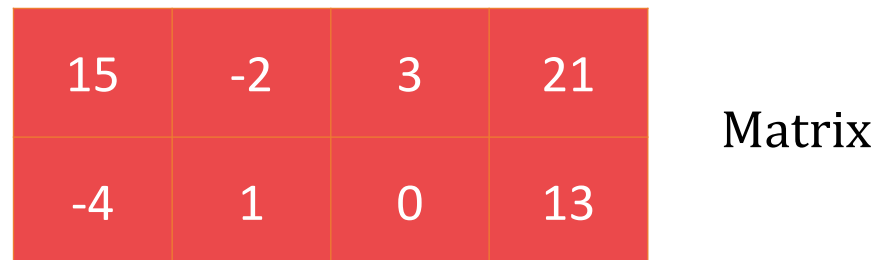
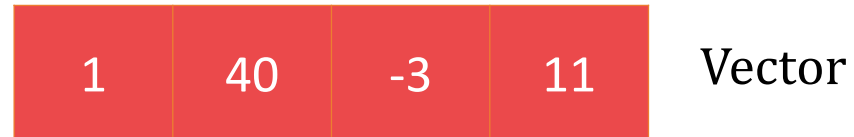


RULES TO DECLARE VARIABLES

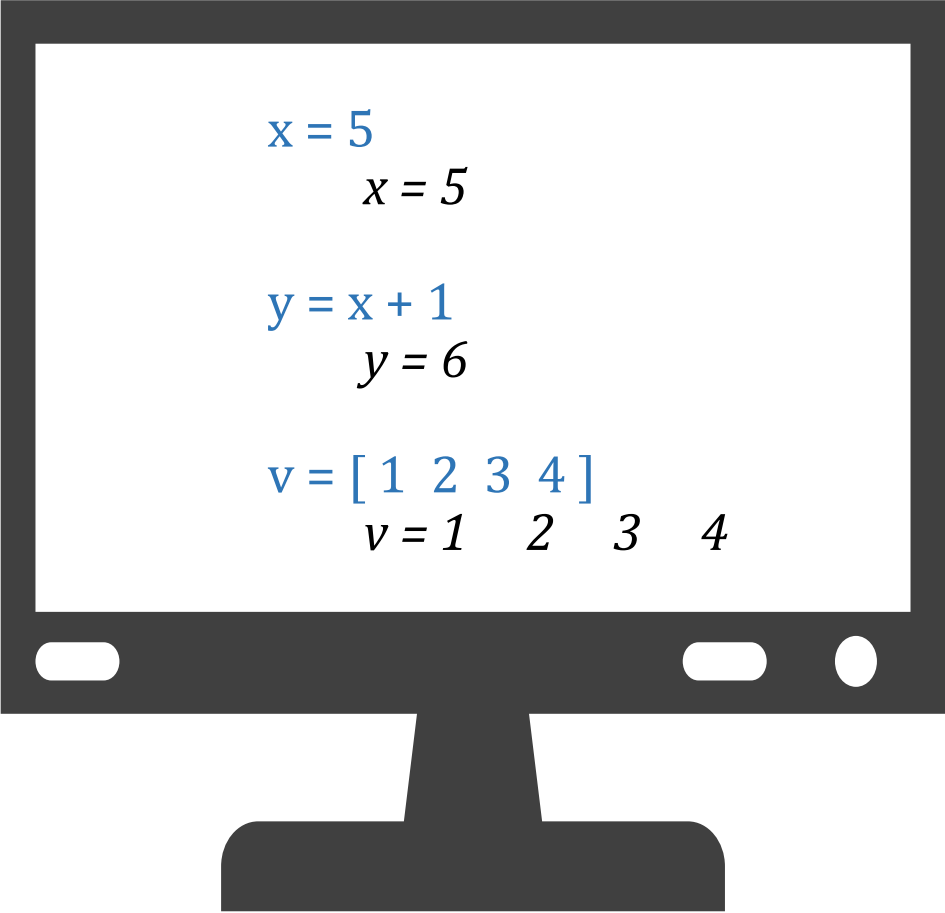
- ❖ Loosely Typed : Declaration is not required
- ❖ Case Sensitive
- ❖ Maximum Length : 63 Characters
- ❖ Must Start With Character
- ❖ Special characters are not allowed
- ❖ Special Variables are not allowed
- ❖ Keywords are not allowed

FUNDAMENTAL UNITS

- ◇ The fundamental unit of data is an **array**



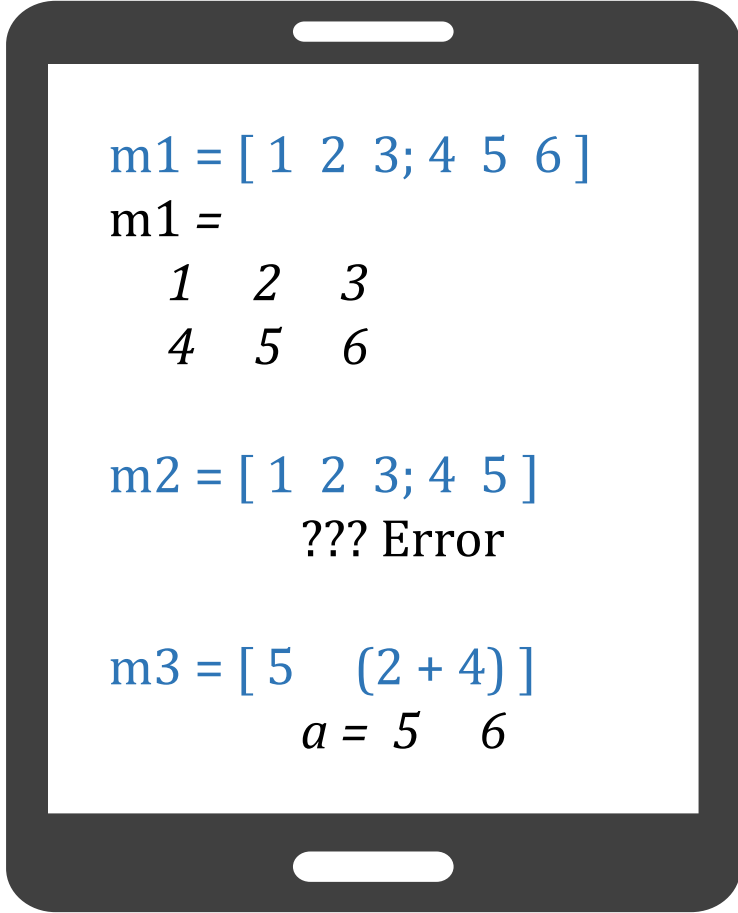
INITIALIZATION USING ASSIGNMENT



```
x = 5  
x = 5
```

```
y = x + 1  
y = 6
```

```
v = [ 1 2 3 4 ]  
v = 1 2 3 4
```



```
m1 = [ 1 2 3; 4 5 6 ]
```

```
m1 =
```

```
1 2 3  
4 5 6
```

```
m2 = [ 1 2 3; 4 5 ]
```

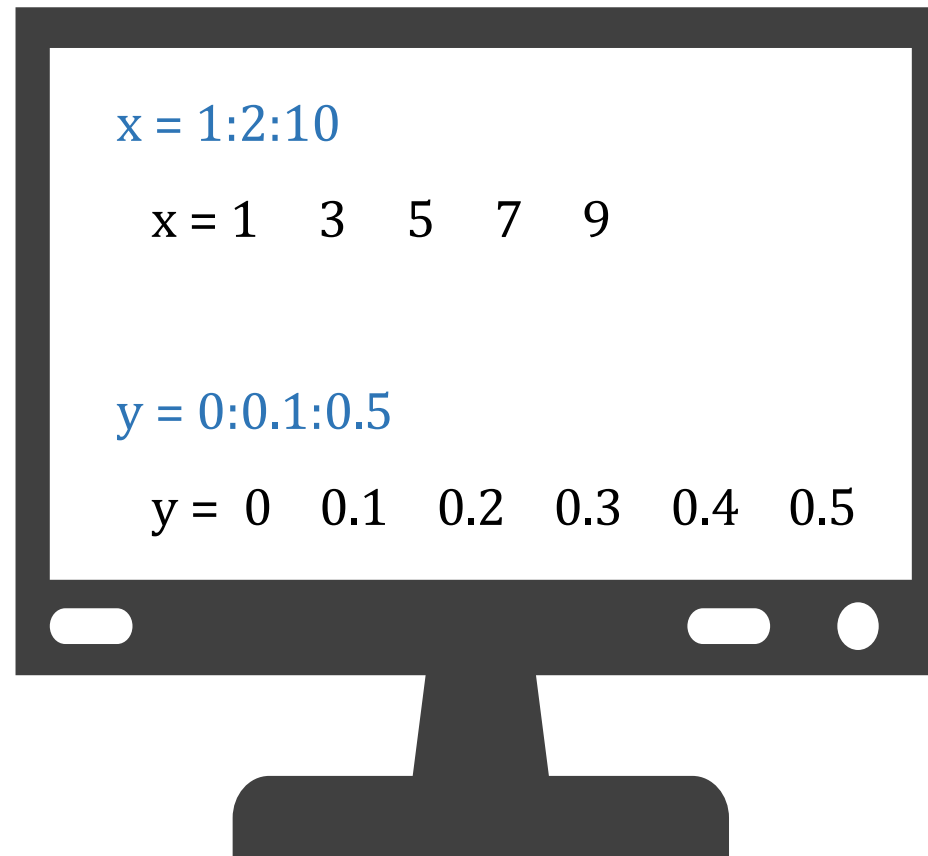
```
??? Error
```

```
m3 = [ 5 (2 + 4) ]
```

```
a = 5 6
```

INITIALIZATION USING **SHORTCUT STATEMENT**

◇ Colon Operator → **First : Increment : Last**



INITIALIZATION USING BUILT IN FUNCTIONS

```
x = zeros(2)
```

```
x =
```

```
0  0  
0  0
```

```
z = zeros(2,3)
```

```
z =
```

```
0  0  0  
0  0  0
```

```
t = zeros( size(z) )
```

```
t =
```

```
0  0  0  
0  0  0
```

```
ones(),  
size(),  
length()
```


INITIALIZATION USING **USER INPUT**

```
value = input( 'Enter an input value:→ ' )
```

```
>> Enter an input value:→ 1.25
```

```
value =
```

```
1.2500
```

NUMBER FORMATS

◇ Changing the data format

◇ value = 12.345678901234567

◇ *format short* → 12.3457

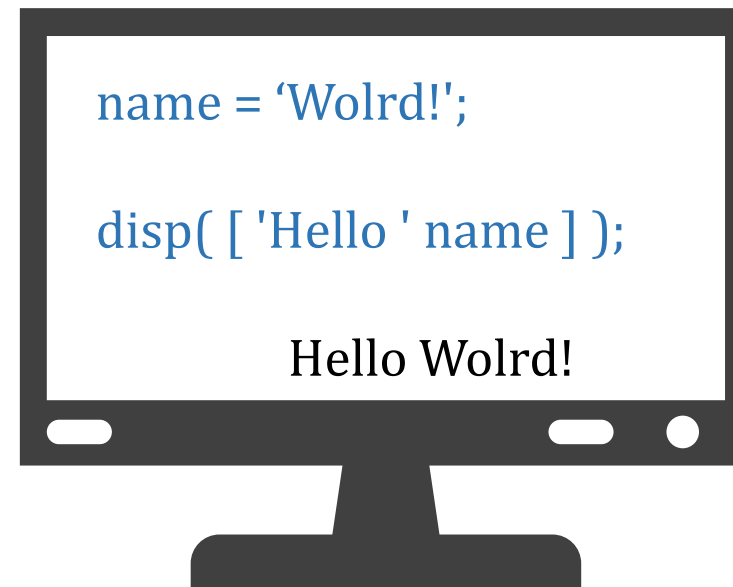
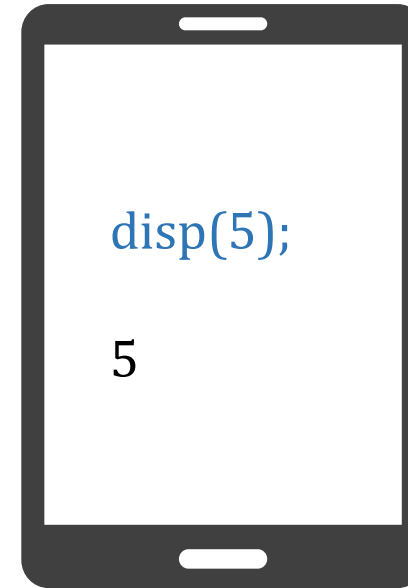
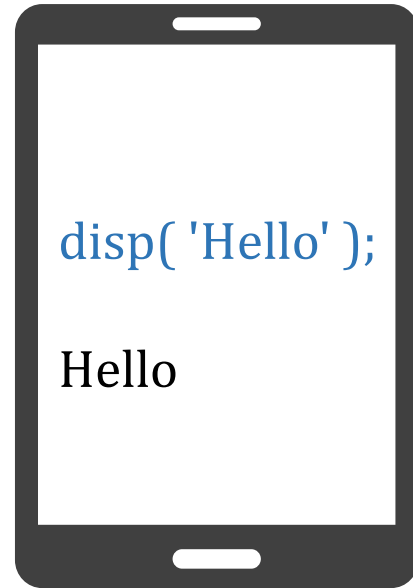
◇ *format long* → 12.34567890123457

◇ *format short e* → 1.2346e+001

◇ *format long e* → 1.234567890123457e+001

◇ *format rat* → 1000/81

DISPLAY DATA



DISPLAY DATA WITH FORMAT

◆ The *fprintf(format, data)* function:

- ◆ *%d* *integer*
- ◆ *%f* *floating point format*
- ◆ *%e* *exponential format*
- ◆ *\n* *new line character*
- ◆ *\t* *tab character*

OUTPUT FORMATTING

```
fprintf( 'Result is %d', 3 );
```

Result is 3

```
fprintf( 'Area of a circle with radius %d  
is %f', 3, pi*3^2 );
```

Area of a circle with radius 3 is

28.274334

```
x = 5;
```

```
fprintf( 'x = %3d', x );
```

x = 5

```
x = pi;
```

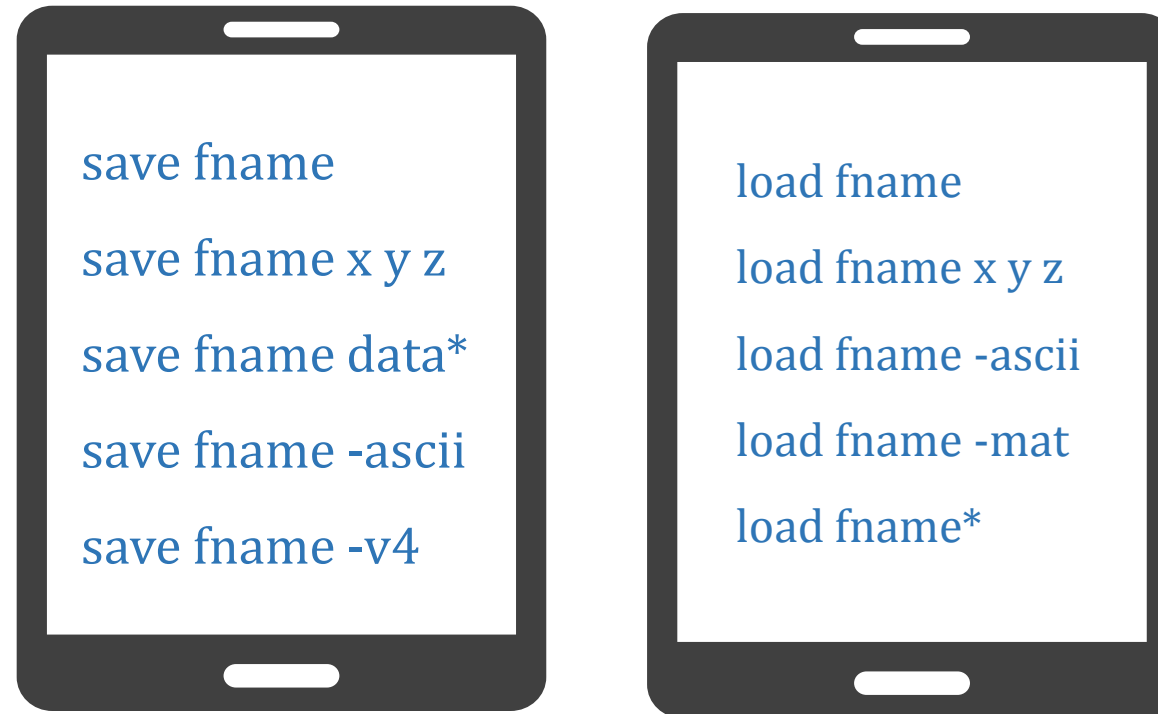
```
fprintf( 'x = %0.2f', x );
```

x = 3.14

```
fprintf( 'x = %6.2f', x );
```

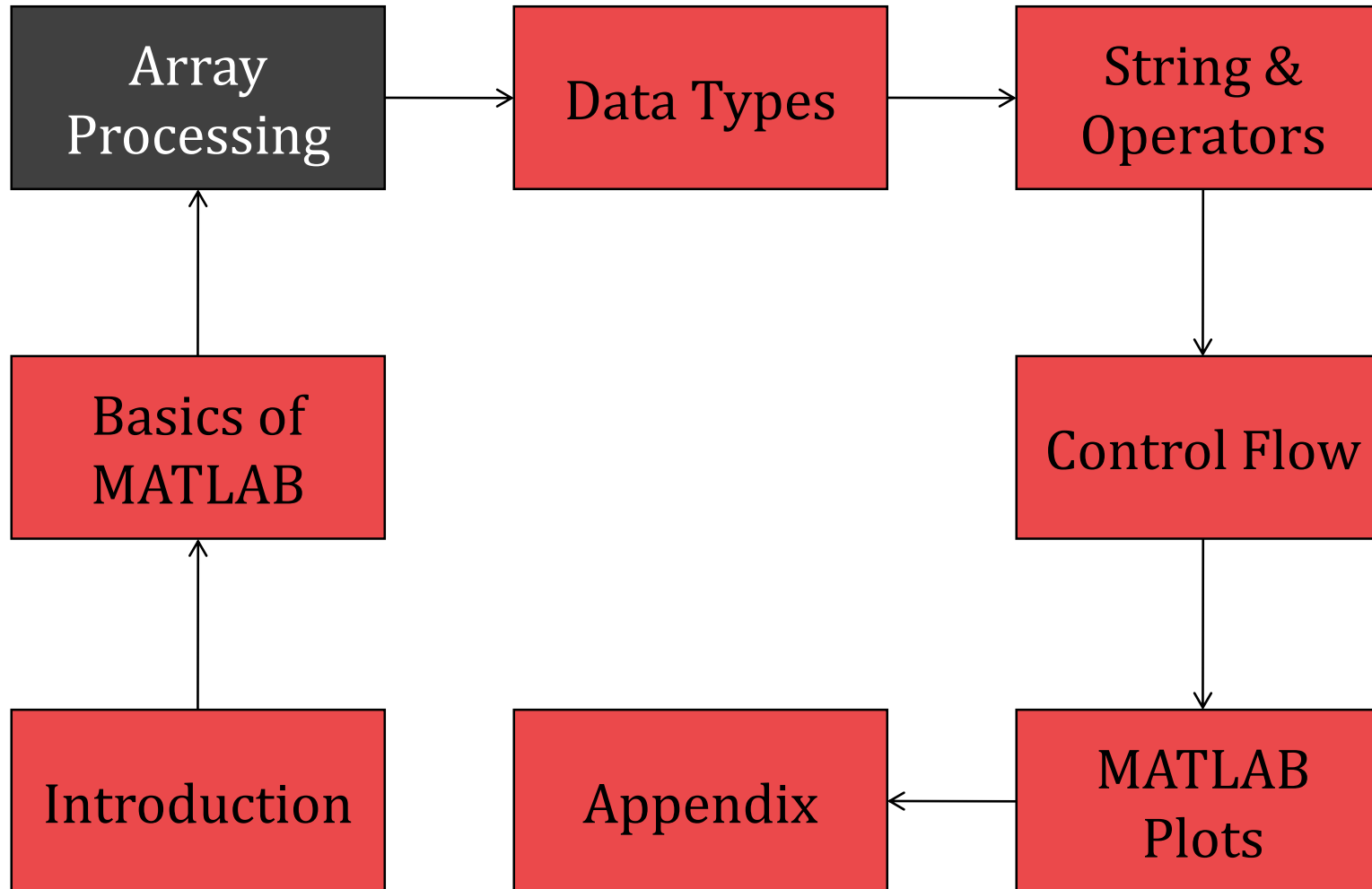
x = 3.14

FILE I/O: LOW LEVEL ROUTINES



- ◇ csvread - Comma-separated ASCII files
- ◇ dlmread - General ASCII delimited files
- ◇ textread - Read formatted data

OPERATIONAL PIPELINE



ARRAY VS. MATRIX OPERATION

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix}$$

$$A.*B = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & a_{13}b_{13} \\ a_{21}b_{21} & a_{22}b_{22} & a_{13}b_{13} \end{bmatrix}$$

$$A./B = \begin{bmatrix} a_{11}/b_{11} & a_{12}/b_{12} & a_{13}/b_{13} \\ a_{21}/b_{21} & a_{22}/b_{22} & a_{13}/b_{13} \end{bmatrix}$$

$$A.^n = \begin{bmatrix} a_{11}^n & a_{12}^n & a_{13}^n \\ a_{21}^n & a_{22}^n & a_{23}^n \end{bmatrix}$$

```
>> M = [1 2; 3 4]
```

```
M =
```

```
1 2  
3 4
```

```
>> M^2
```

```
ans =
```

```
7 10  
15 22
```

```
>> M.^2
```

```
ans =
```

```
1 4  
9 16
```


BASICS OF MATRIX

$a = [1 \ 2];$ $\longrightarrow$ 

$b = [3, \ 4];$ $\longrightarrow$ 

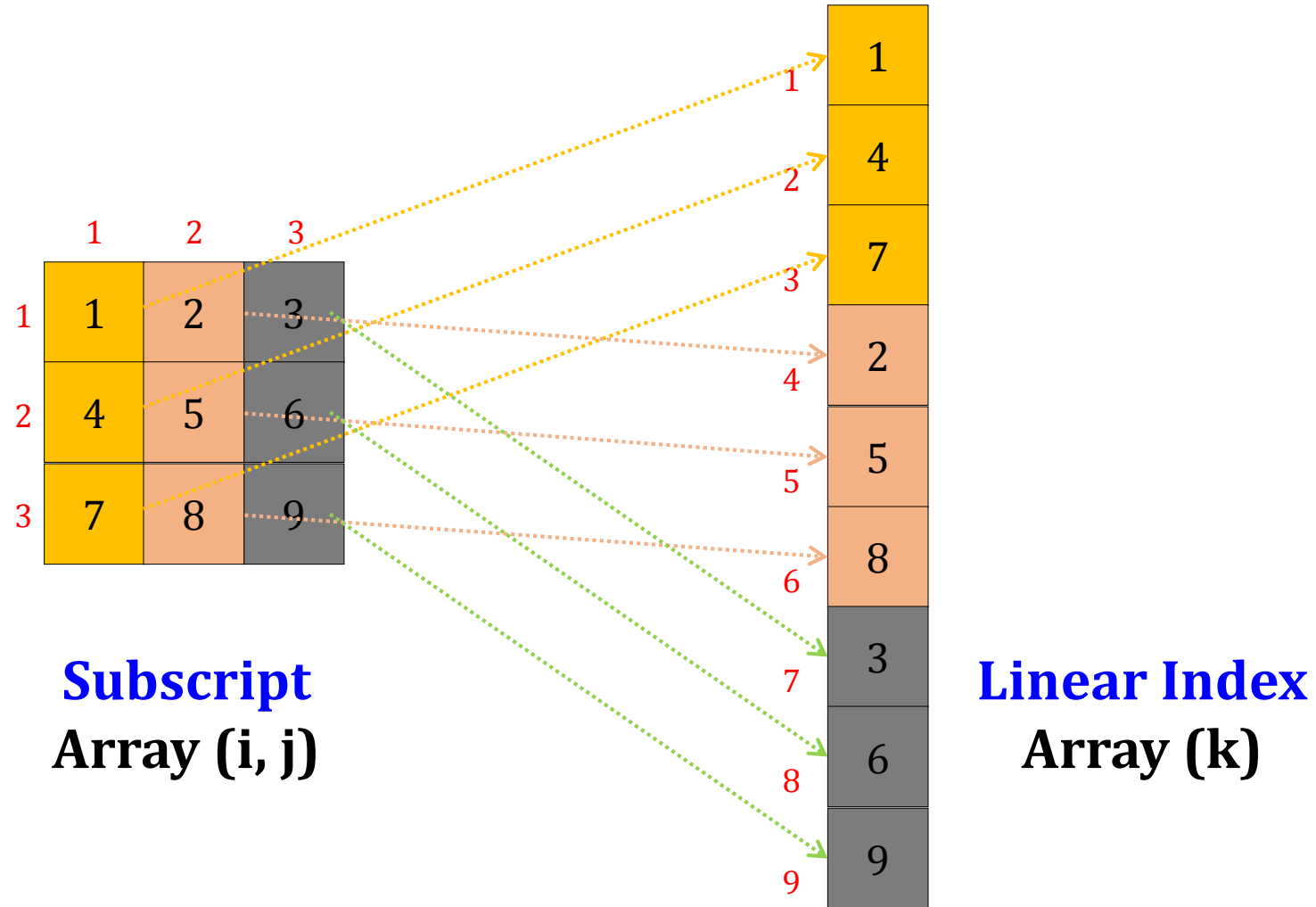
$c = [5; \ 6];$ $\longrightarrow$ 

$d = [a; \ b];$ $\longrightarrow$ 

$e = [d \ c];$ $\longrightarrow$ 

$f = [[e \ e]; [a \ b \ a]];$ $\longrightarrow$ 

ARRAY INDEXING



ARRAY INDEXING

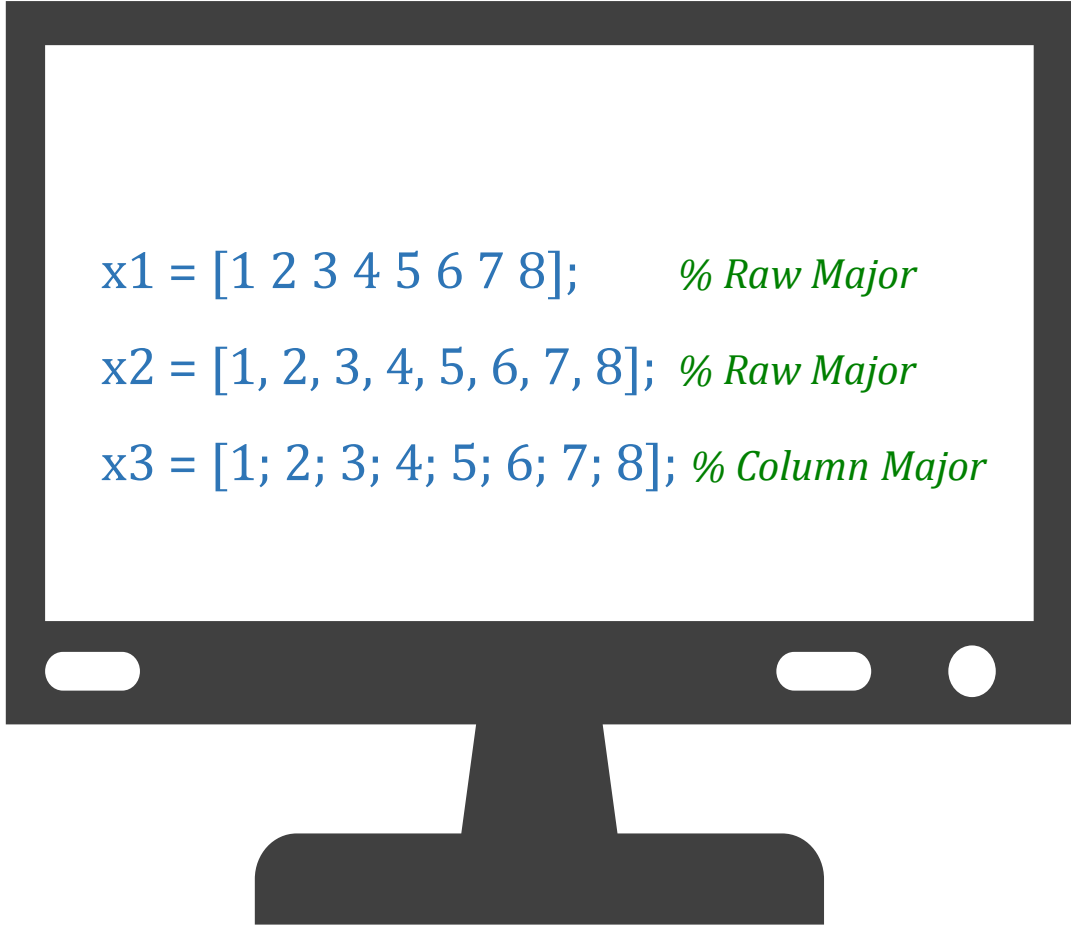
A =

	1	2	3	4	5
1	4 ¹	10 ⁶	1 ¹¹	6 ¹⁶	2 ²¹
2	8 ²	1.2 ⁷	9 ¹²	4 ¹⁷	25 ²²
3	7 ³	5 ⁸	7 ¹³	1 ¹⁸	11 ²³
4	0 ⁴	0.5 ⁹	4 ¹⁴	5 ¹⁹	56 ²⁴
5	23 ⁵	83 ¹⁰	13 ¹⁵	0 ²⁰	10 ²⁵

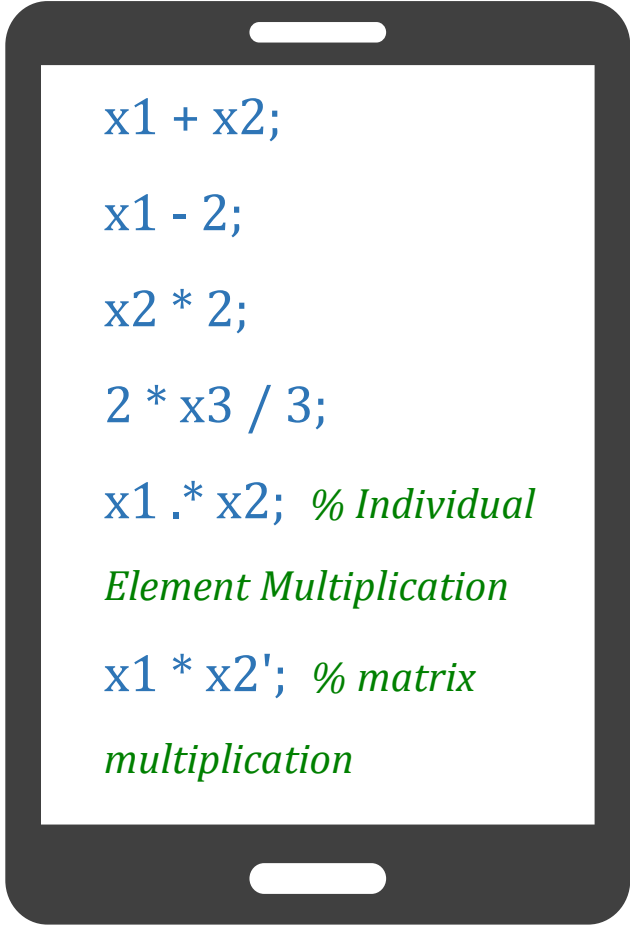
Annotations:

- $A(3, 1)$ (blue) and $A(3)$ (red) point to the element 7³.
- $A(1:5, 5)$ (blue), $A(:, 5)$ (blue), and $A(21:25)$ (red) point to the column containing 2²¹, 25²², 11²³, 56²⁴, and 10²⁵.
- $A(1:\text{end}, \text{end})$ (blue) and $A(:, \text{end})$ (blue) point to the element 2²¹.
- $A(21:\text{end})$ (red) points to the element 10²⁵.
- $A(4:5, 2:3)$ (blue) and $A([9\ 14; 10\ 15])$ (red) point to the submatrix containing 0.5⁹, 4¹⁴, 83¹⁰, and 13¹⁵.

ARRAY BASICS



```
x1 = [1 2 3 4 5 6 7 8];    % Row Major  
x2 = [1, 2, 3, 4, 5, 6, 7, 8]; % Row Major  
x3 = [1; 2; 3; 4; 5; 6; 7; 8]; % Column Major
```



```
x1 + x2;  
x1 - 2;  
x2 * 2;  
2 * x3 / 3;  
x1 .* x2; % Individual  
Element Multiplication  
x1 * x2'; % matrix  
multiplication
```

ARRAY INITIALIZATION AND USE

```
X = linspace(0.1*pi, pi, 10)
```

```
X = [0.1 : 0.1 : 1]*pi
```

```
X = (0.1 : 0.1 : 1)*pi;
```

```
Y = sin(X);           % Apply sin function on each element
```

```
X(1:5);               % Read first five element
```

```
X(5:end);             % Read 5th to last element
```

```
X(3:2:10);            % Read 3rd, 5th, 7th and 9th element
```

```
X(5:-1:1);            % Read 5th, 3rd and 1st element
```

```
X([2 9]);              % Read 2nd and 9th element
```

ARRAY BASICS

```
A = [1:5];
```

```
B = [6:10];
```

```
X = [A B];
```

```
Y = [X; B];
```

```
X = abs(A) > 2 % Return logical array of 0 & 1
```

```
A(X) % Grab the logically true value
```

```
Reshape(A, 2, 3)
```

ARRAY INDEXING

- ◆ *Use () parentheses to specify index*
- ◆ *Colon operator (:) specifies range / ALL*
- ◆ *[] to create matrix of index subscripts*
- ◆ *'end' specifies maximum index value*
- ◆ *size(A) gives dimensions of array*

STANDARD ARRAYS

```
a=ones(5);
```

```
a=ones(5,3);
```

```
a=zeros(5);
```

```
a=zeros(5,3);
```

```
a=eye(5);
```

```
a=eye(5,3);
```

```
True(2,3);
```

```
False(2,3);
```

```
a=rand(5);
```

```
a=rand(5,3);
```

```
a=randn(5);
```

```
a=randn(5,3);
```

```
Magic(5 )
```

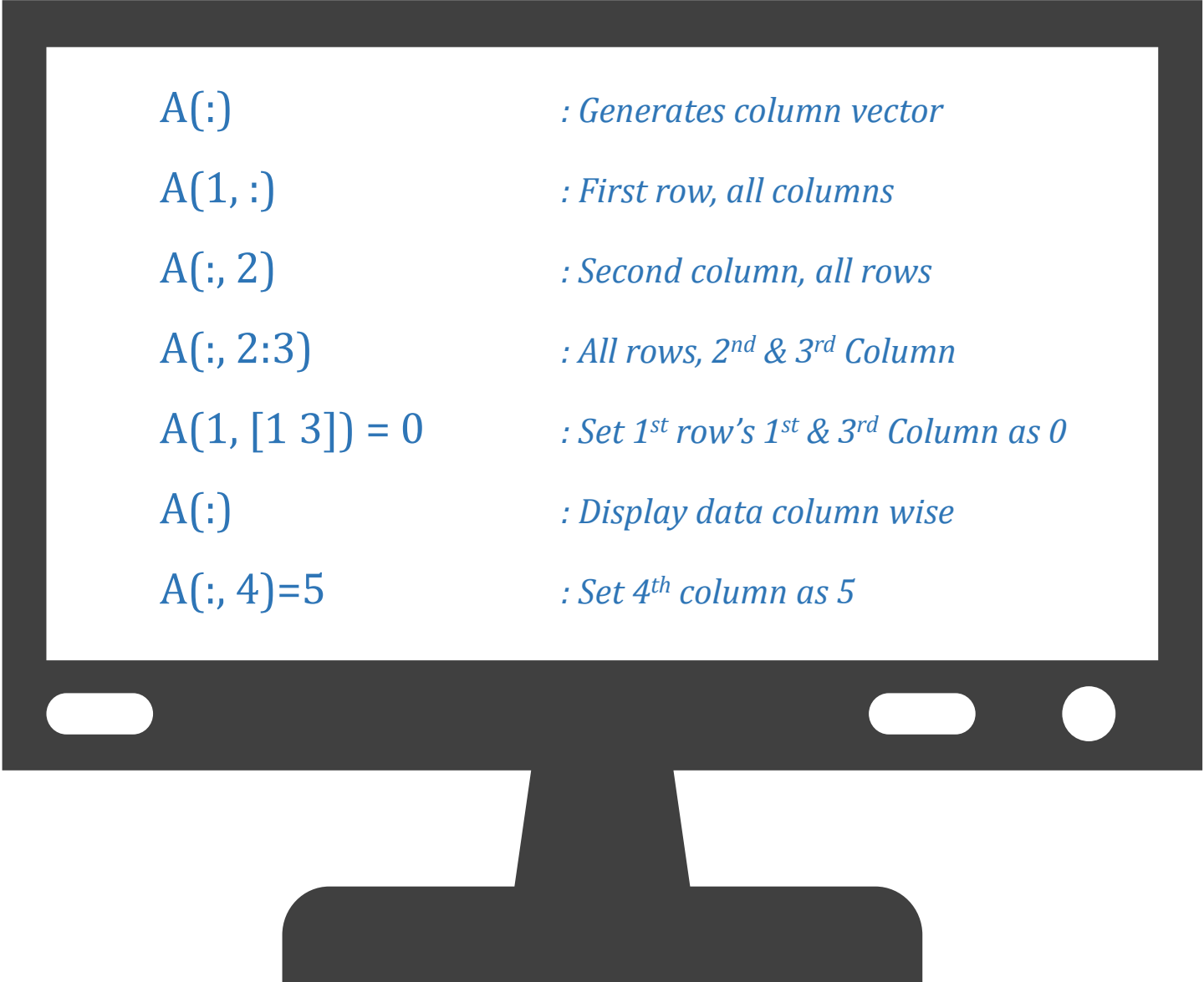
```
b=[1 2 3 4 5];
```

```
a=diag(b);
```

```
a=diag(b,2);
```

```
a=diag(b,-2);
```


ARRAY MANIPULATION



The image shows a computer monitor with a dark grey frame and a white screen. The screen displays seven lines of MATLAB array manipulation syntax, each followed by a descriptive comment in italics. The monitor has a black stand and three white oval-shaped buttons on the bottom bezel.

$A(:)$	<i>: Generates column vector</i>
$A(1, :)$	<i>: First row, all columns</i>
$A(:, 2)$	<i>: Second column, all rows</i>
$A(:, 2:3)$	<i>: All rows, 2nd & 3rd Column</i>
$A(1, [1\ 3]) = 0$	<i>: Set 1st row's 1st & 3rd Column as 0</i>
$A(:)$	<i>: Display data column wise</i>
$A(:, 4)=5$	<i>: Set 4th column as 5</i>

ARRAY FUNCTIONS

`A=[randperm(8); randperm(8); randperm(8)]`

`[data, index] = sort(X, 'ascend')`

`sort(A(:, 4))` : Sort 4th column

`find(A > 2)` : Return Index number on Logical true

`max(A)` : Return Max value from all column

`max(A(:, 1))` : Return Max value from 1st Column

`[a b] = max(A)` : Return max value with index

`diag(A)` : Extract diagonal values

`triu(A)` : Upper triangular part

`tril(A)` : Lower triangular part

`repmat(A, 2)` : replicate matrix A by twice

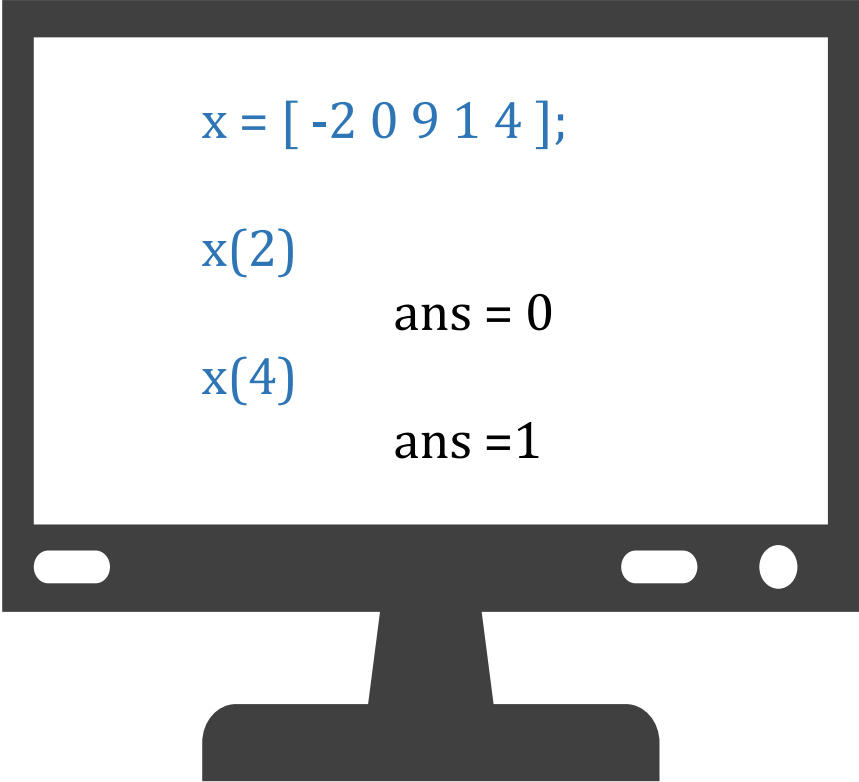
`size(A)` : return dimension of an Array

`lenth(A)` : return no. of Columns

`numel(A)` : return total no. of elements in A

ARRAY INDEXING

- ◆ Array indices start from **1**



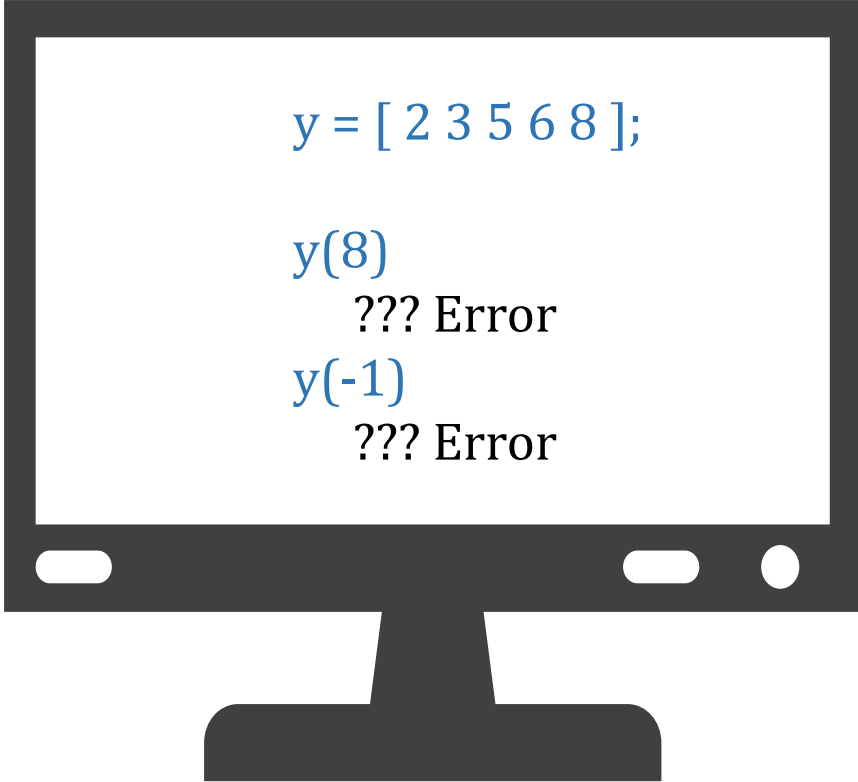
```
x = [ -2 0 9 1 4 ];
```

```
x(2)
```

```
ans = 0
```

```
x(4)
```

```
ans = 1
```



```
y = [ 2 3 5 6 8 ];
```

```
y(8)
```

```
??? Error
```

```
y(-1)
```

```
??? Error
```

ARRAY INDEXING

```
y = [ 1 2 3; 4 5 6 ];
```

```
y(1, 2)  
ans = 2
```

```
y(2, 3)  
ans = 6
```

```
y(2)  
ans = 4
```

```
y(1, 2:end)  
ans = 2 3
```

```
y(1, :)  
ans = 1 2 3
```

```
y(:, 2)  
ans = 2  
5
```

```
y(2, 1:2)  
ans = 4 5
```

```
y(:, 2:end)  
ans = 2 3  
5 6
```

ARRAY INDEXING

$y = [1\ 2\ 3; 4\ 5\ 6];$

$y(1, 2) = -5$

$y =$

1	-5	3
4	5	6

$y(2, 1) = 0$

$y =$

1	-5	3
0	5	6

$y(1, :) = [4\ -1\ 9]$

$y =$

4	-1	9
0	5	6

$y(:, 2) = [3; 2]$

$y =$

4	3	9
0	2	6

ARRAY INDEXING

$z = [1 \ 2 \ 3; 4 \ 5 \ 6; 7 \ 8 \ 9];$

$z(3, :) = 0$

$z =$

1	2	3
4	5	6
0	0	0

$z(:, 1) = -2$

$z =$

-2	2	3
-2	5	6
-2	0	0

$z(2, :) = [1 \ 5]$

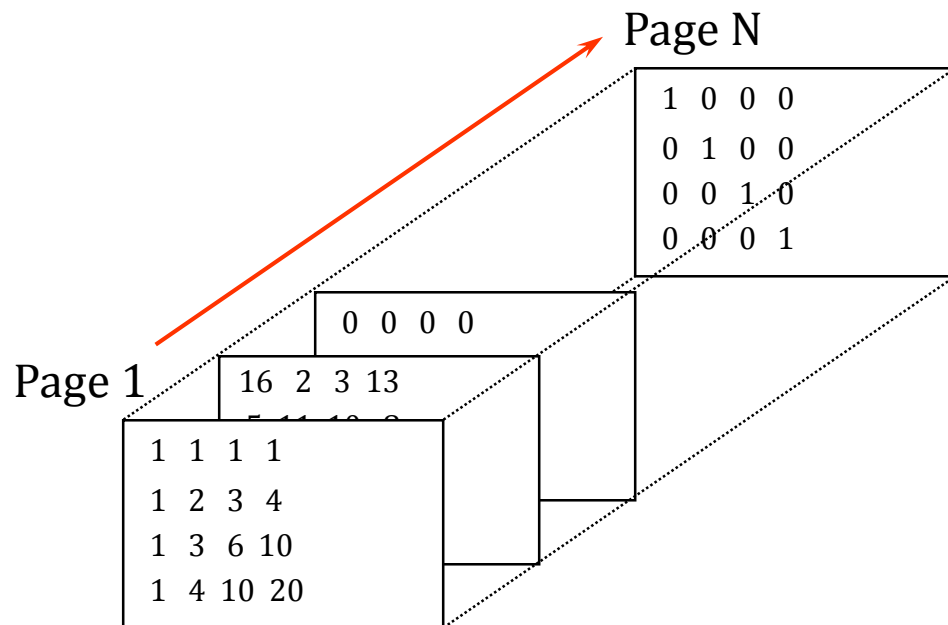
??? Error

$z(:, 3) = [3 \ 9]$

??? Error

MULTIDIMENSIONAL ARRAY

- ◇ Generally Third dimension is referred as page, and higher dimensions have no generic name
- ◇ Each page contains 2D Array



```
A = pascal(4);
```

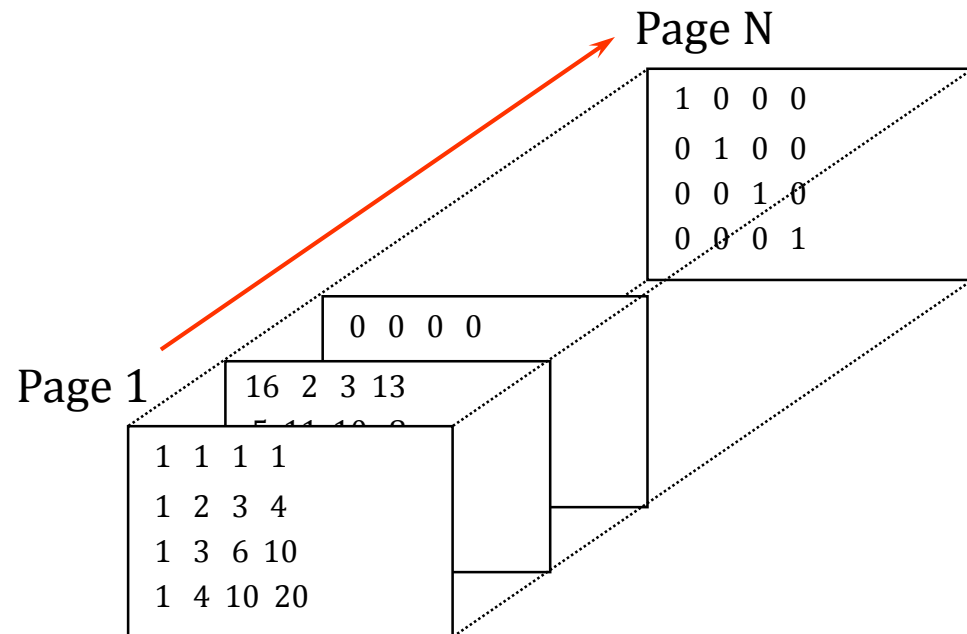
```
A(:, :, 2) = magic(4)
```

```
A(:, :, 3) = zeros(4)
```

```
.
```

```
A(:, :, N) = eye(4)
```

DISPLAYING MULTIDIMENSIONAL ARRAY



$A(:, :, 1) =$

1	1	1	1
1	2	3	4
1	3	6	10
1	4	10	20

$A(:, :, : 2) =$

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

$A(2, :, 1) =$

1	2	3	4
---	---	---	---

$A(:, 1, 2) =$

16
5
9
4

3D ARRAY MANIPULATION

```
A = zeros(2, 3)
```

```
A(:, :, 2) = ones(2, 3)
```

```
A(:, :, 3) = 4
```

```
A(:, :, 1) =
```

```
0  0  0  
0  0  0
```

```
A(:, :, 2) =
```

```
1  1  1  
1  1  1
```

```
A(:, :, 3) =
```

```
4  4  4  
4  4  4
```

```
B = [A(:, :, 1) A(:, :, 2) A(:, :, 3)]
```

```
>> B = reshape(B, 2, 9)
```

```
0  0  0  1  1  1  4  4  4  
0  0  0  1  1  1  4  4  4
```

3D ARRAY MANIPULATION

```
>> A=ones(2,3)
```

A =

1	1	1
1	1	1

```
repmat(A,[2,1,2])
```

ans(:, :, 1) =

1	1	1
1	1	1
1	1	1
1	1	1

ans(:, :, 2) =

1	1	1
1	1	1
1	1	1
1	1	1

3D ARRAY INDEXING

```
>> D = ones(2, 3, 3)
```

```
D(:, :, 1) =
```

1	1	1
1	1	1

```
D(:, :, 2) =
```

1	1	1
1	1	1

```
D(:, :, 3) =
```

1	1	1
1	1	1

```
>> sub2ind(size(D), 2, 1, 2)
```

```
ans = 8
```

```
>> sub2ind(size(D), 1, 3, 3)
```

```
ans = 17
```

```
>> [r c p]=ind2sub(size(D), 18)
```

```
ans = 2 2 3
```

CELL ARRAYS

- ❖ It is a special array of arrays. Each element of cell array may point to a scalar, an array, or another cell array.

```
>> C = cell(2, 3);  
>> M = magic(2);  
>> a = 1:3;  
>> b = [4; 5; 6];  
>> s = 'Hello';  
>> C{1,1} = M; C{1,2} = a;  
>> C{2,1} = b; C{2,2} = s;  
>> C{1,3} = {1};
```

```
C =      [2x2 double]   [1x3 double]   {1x1 cell}  
        [2x1 double]   'This is a string.' []  
  
>> C{1,1} % prints contents of a specific cell element  
      1   3  
      4   2  
  
>> C(1,:) % prints first row of C; not the content  
Related utilities: iscell, cell2mat
```

STRUCTURES

```
>> name(1).last = 'Smith';  name(2).last = 'Hess';  
>> name(1).first = 'Mary';  name(2).first = 'Robert';  
>> name(1).sex = 'female';  name(2).sex = 'male';  
>> name(1).age = 45;        name(2).age = 50;  
>> name(2)  
    last: 'Hess'  
    first: 'Robert'  
    sex: 'male'  
    age: 50
```

```
>> name = struct('last',{Smith,"Hess"},  
    'first',{Mary,"Robert"},...  
    ('sex',{female,"male"}, 'age',{45,50}));
```

Related utilities: isstruct, fieldnames,
getfield, isfield



Thank You!

 Dr. Mahesh Goyani

 mgoyani@gmail.com

 www.maheshgoyani.in