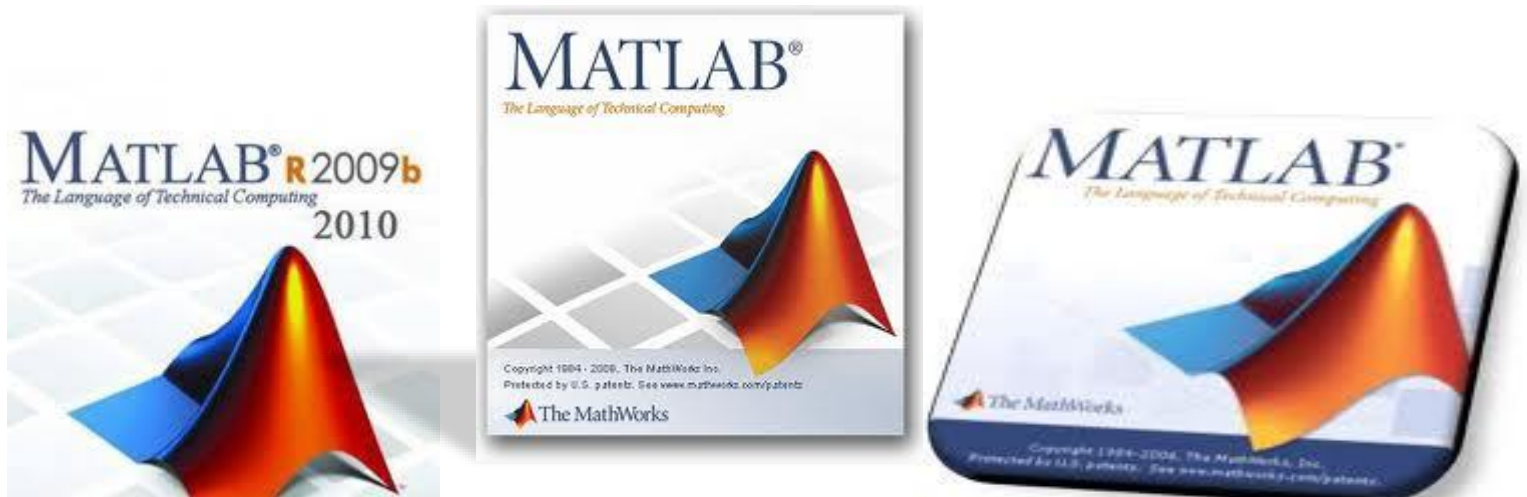


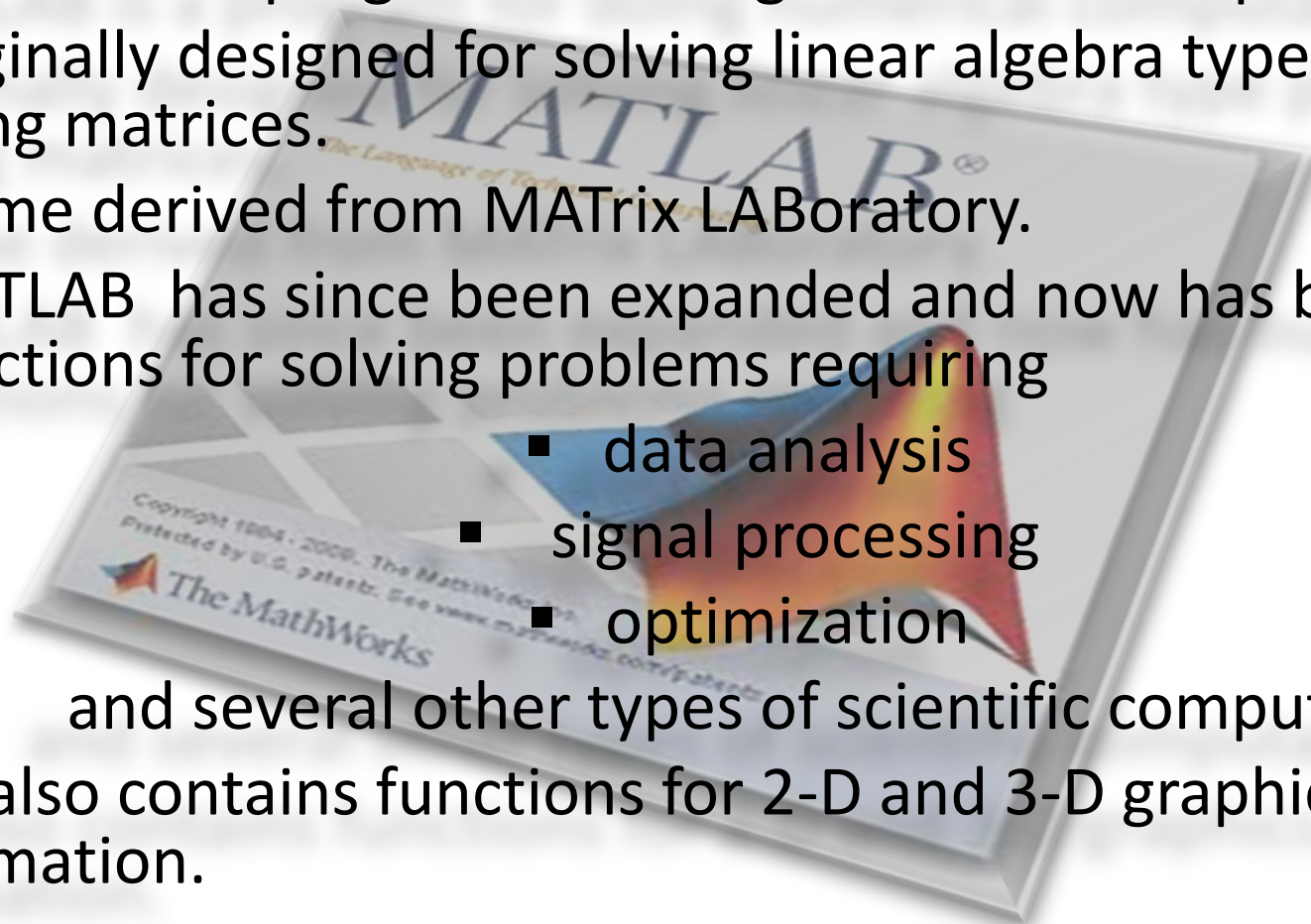
# INTRODUCTION TO MATLAB



By  
Murshida ck  
TKMCE

# Introduction

- MATLAB is a program for doing numerical computation.
- originally designed for solving linear algebra type problems using matrices.
- name derived from MATrix LABoratory.
- MATLAB has since been expanded and now has built-in functions for solving problems requiring
  - data analysis
  - signal processing
  - optimization
- and several other types of scientific computations.
- It also contains functions for 2-D and 3-D graphics and animation.



HOME PLOTS APPS Search Documentation Log In

New Script New Live Script New Open Compare Import Data Save Workspace Clear Workspace Analyze Code Run and Time Clear Commands Simulink Layout Parallel Add-Ons Help Community Request Support Learn MATLAB

FILE VARIABLE CODE SIMULINK ENVIRONMENT RESOURCES

C:\Program Files\MATLAB\R2018a\bin

- Current Folder
- Name
  - worker.bat
  - mw\_mpiexec.bat
  - mexutils.pm
  - mexsetup.pm
  - mexext.bat
  - mex.pl
  - mex.bat
  - mcc.bat
  - mbuild.bat
  - matlab.exe
  - lcdata\_utf8.xml
  - lcdata.xsd
  - lcdata.xml
  - deploytool.bat
  - win64
  - win32
  - util

Command Window

```
fx >>
```

Workspace

Name	Value
------	-------

Details

Select a file to view details

- The MATLAB environment is command oriented somewhat like UNIX. A prompt appears on the screen and a MATLAB statement can be entered. When the <ENTER> key is pressed, the statement is executed, and another prompt appears.
- If a statement is terminated with a semicolon (;), no results will be displayed. Otherwise results will appear before the next prompt.



# Desktop Tools

- **Command Window**

- type commands

- **Workspace**

- view program variables

- `clear` to clear

- double click on a variable to see it in the Array Editor

- **Command History**

- view past commands

- save a whole session using `diary`

MATLAB R2018a

HOME PLOTS APPS

Search Documentation Log In

New Script New Live Script New Open Compare Import Data Save Workspace Clear Workspace Favorites Analyze Code Run and Time Clear Commands Simulink Layout Parallel Add-Ons Help Community Request Support Learn MATLAB

FILE VARIABLE CODE SIMULINK ENVIRONMENT RESOURCES

C:\Program Files\MATLAB\R2018a\bin

Current Folder

**Command Window**

**Current Directory**

- worker.bat
- mw\_mpiexec.bat
- mexutils.pm
- mexsetup.pm
- mexent.bat
- mex.pl
- mex.bat
- mcc.bat
- mbuild.bat
- matlab.exe
- lcdata\_utf8.xml
- lcdata.xsd
- lcdata.xml
- deploytool.bat

win64

win32

util

Details

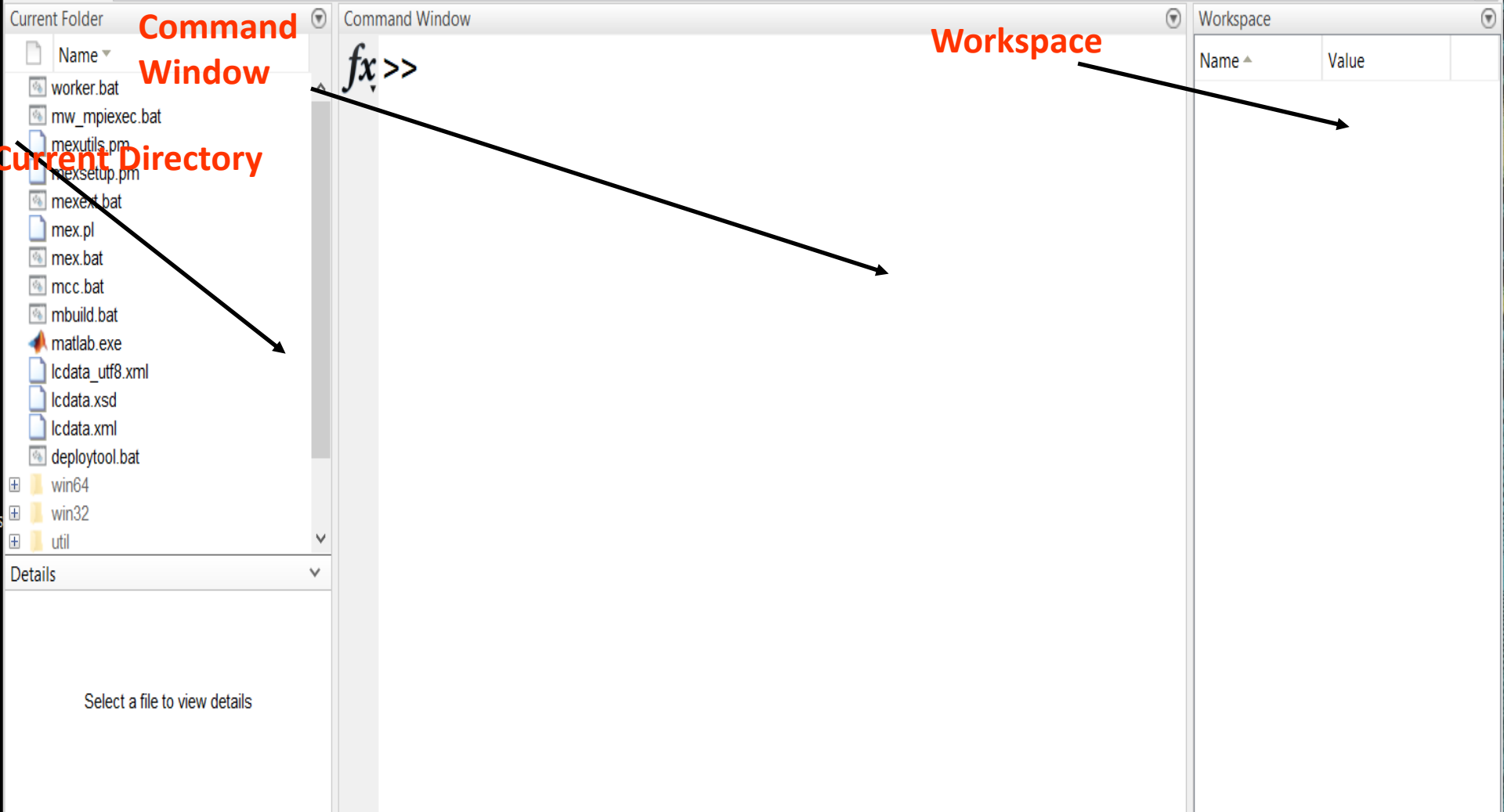
Select a file to view details

Command Window

```
fx>>
```

**Workspace**

Name	Value
------	-------



- **Figure Window**

- contains output from graphic commands

- **Help Window**

- provides help information

- **Editor Window**

- creates and debugs script and function files

- **Current directory Window**

- shows files in current directory

- **Launch Pad Window**

- provides access to tools, demos and documentation

# COMMAND WINDOW

- `>>` type code
- Press **enter**
- Command executed and output display
- **semicolon(;**)  
     output not displayed
- **Ellipsis(...)** if a command is too long to fit in one line
- Command can continue line after line up to **4096 characters.**



- Matlab **case sensitive**
- % -comment
- **clc** -clear screen
- ↑ -recall previously typed commands
- ↓ -move down to previously typed commands

# Arithmetic Operations With Scalars

<u>Operation</u>	<u>Symbol</u>	<u>Example</u>
Addition	+	$5+3$
Subtraction	-	$5-3$
Multiplication	*	$5*3$
Right division	/	$5/3$
Left division	\	$5\backslash 3=3/5$
Exponentiation	^	$5^3=125$

# text from a MATLAB screen

» %To get started, type one of these commands:

» a=5;

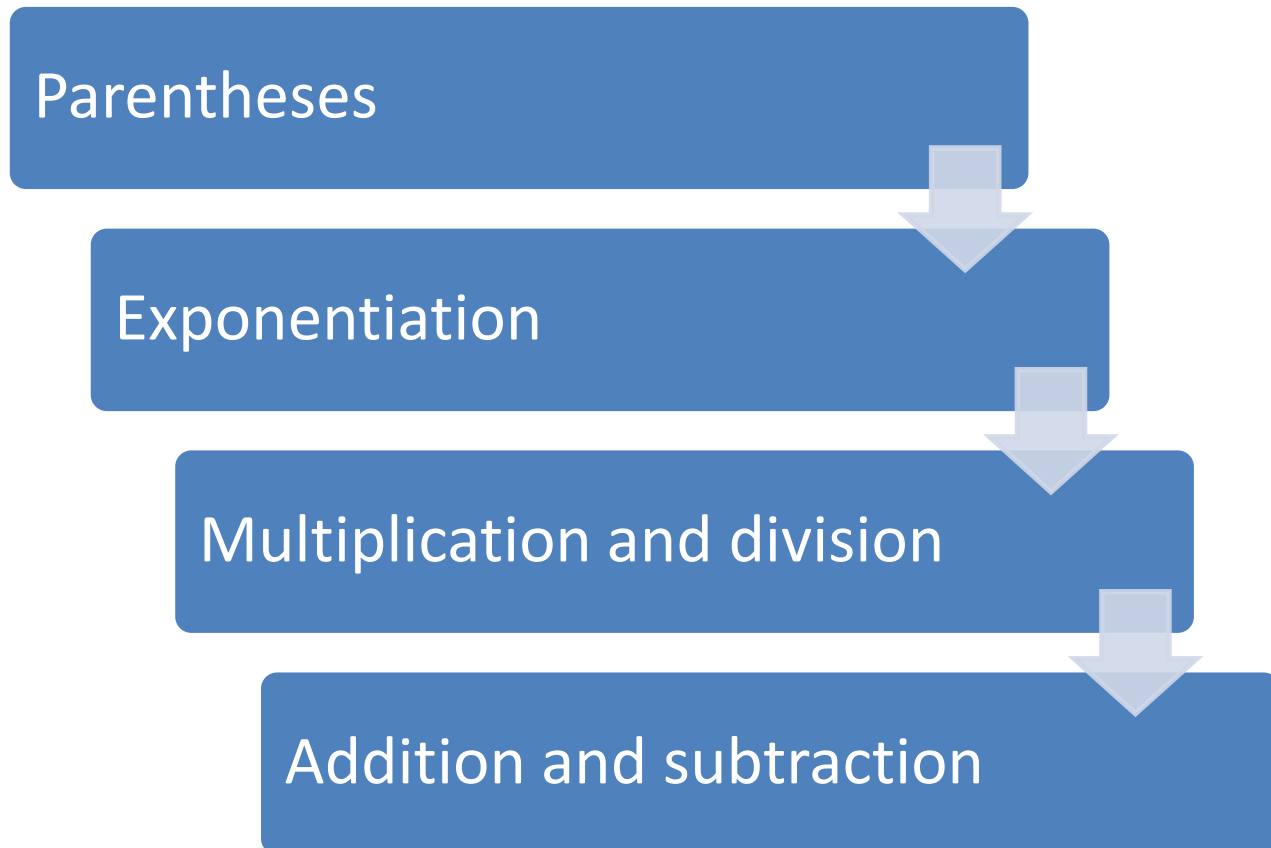
» b=a/2

b =

2.5000

»

# Order of Precedence



# Display Formats

- User can control format in which MATLAB displays o/p on screen.
- **format** command– To change o/p format
- Default format for numerical values **-short** (fixed point with 4 decimal digits)

Command	Description
format short	Fixed-point with 4 decimal digits
format long	Fixed-point with 14 decimal digits
format bank	2 decimal digits
format compact	Eliminates empty lines
format loose	Adds empty lines

# Elementary Math functions

Function	Description
sqrt (x)	Square root
exp (x)	Exponential ( $e^x$ )
abs (x)	Absolute value
log (x)	Natural logarithm Base e logarithm
Log10(x)	Base 10 logarithm
factorial(x)	Factorial function $x!$

# Trigonometric math functions

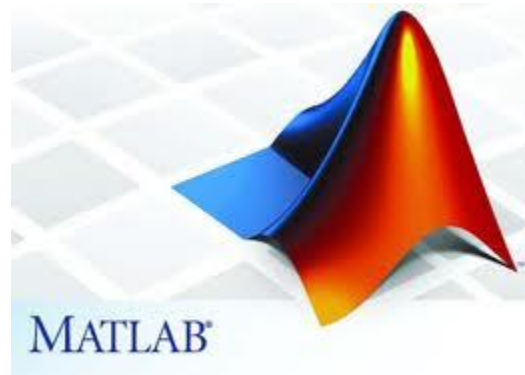
$\sin(x)$ ,  $\cos(x)$ ,  $\tan(x)$ ,  $\cot(x)$

## Rounding functions

Function	Description
<code>round(x)</code>	Round to the nearest integer
<code>fix(x)</code>	Round towards zero
<code>ceil(x)</code>	Round towards infinity
<code>floor(x)</code>	Round towards minus infinity
<code>rem(x,y)</code>	Returns remainder after x is divided by y
<code>Sign(x)</code>	Signum function



# Variables (Arrays) and Operators



# Defining scalar variables

The variable is a name made of a letter or a combination of several letters that are assigned a numerical value

- actually the name of a memory location
- assignment operator '='

Variable\_name = a numerical value or a computable expression

```
>>x=15
```

```
>>x=3*x-12
```

- When new variable is created matlab assigns appropriate memory space where assigned value can be stored
- When variable is used stored data is used
- If assigned new value content of memory is replaced

```
>>ABB=72;  
>>ABB=9;  
>>ABB  
ABB=  
9
```

# Rules about variable names

- Variable names are case sensitive.
- Variable names can contain up to 63 characters (as of MATLAB 6.5 and newer).
- Variable names must start with a letter followed by letters, digits, and underscores.
- Must begin with a letter.
- Avoid using names of built-in functions for variable.

# Predefined variables

variable	description
ans	Value of last expression
eps	Smallest difference between 2 numbers
i	$\sqrt{-1}$
inf	Infinity
j	Same as i
NaN	Not a number
pi	The number $\pi$

# Some Useful MATLAB commands

- `who` List known variables
- `whos` List known variables plus their size
- `help` `>> help sqrt` Help on using `sqrt`
- `clear` Clear all variables from work space
- `clear x y` Clear variables `x` and `y` from work space
- `clc` Clear the command window

# Array

List of numbers arranged in row and/or columns.

- Simplest array
  - 1D array
  - usually to represent vectors.
- Complex array
  - 2D array
  - represent matrixes

# Creating vector from a known list of numbers

```
Variable_name = [type vector elements]
```

- Row vector-type elements with space or comma
- Column vector-type elements with semicolon(;) or press Enter key after each element



# Creating vector with constant spacing

Variable\_name = [m:q:n] or

Variable\_name = m:q:n

First term      spacing      last term



Variable\_name = linspace(xi,xf,n)

First element      last element      no of elements (when omitted default value 100)



# Creating 2D array(matrix)

- Matrix are used in science & eng to describe many physical quantities.

```
Variable_name = [1st row elements;2nd row  
elements;.....;last row elements]
```

- All rows must have same number of elements

# *Zeros , ones and eye* commands

- `zeros(m,n)`

mxn matrix of 0's.

- `ones(m,n)`

mxn matrix of 1's.

- `eye(n)`

nxn identity matrix

# Transpose operator

- Type single quote (') following variable to be transposed.

```
>> aa = [3 8 1]
```

```
aa=
```

```
    3    8    1
```

```
>> bb=aa'
```

```
bb=
```

```
    3
```

```
    8
```

```
    1
```

# Array addressing

- Vector named  $ve$

$ve(k)$

element in position  $k$

- Matrix  $ma$

$ma(k,p)$

refers to element in row  $k$  & column  $p$

# Using a colon : in addressing arrays

- $va(:)$**  -all elements of vector  $va$
- $va(m:n)$**  -all elements  $m$  through  $n$  of vector  $va$
- $A(:,n)$**  -elements in all rows of column  $n$  of matrix  $A$
- $A(n,:)$**  -elements in all columns of row  $n$  of matrix  $A$
- $A(:,m:n)$**  -elements of all rows between columns  $m$  and  $n$
- $A(m:n,:)$**  -elements in all columns between rows  $m$  and  $n$

# Adding elements to existing variables

- by assigning values to the elements.

```
>>DF = 1 2 3 4
```

```
DF =
```

```
1 2 3 4
```

```
>>DF(5:10)=10:5:35
```

```
DF =
```

```
1 2 3 4 10 15 20 25 30 35
```

```
>>AD = [5 7 2 ]
```

```
AD =
```

```
5 7 2
```

```
>>AD(8) = 4
```

```
AD=
```

```
5 7 2 0 0 0 0 4
```

Adding elements  
to a vector

Matlab add zeros  
between last  
original element  
and new  
element

```
>>RE = [3 8 1 ];
```

```
>>GT = 4:3:16;
```

```
>>KNH = [RE GT]
```

```
KNH =
```

```
3 8 1 4 7 10 13 16
```

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

```
E=
```

```
1 2 3 4
```

```
5 6 7 8
```

```
>>E(3,:) = [6:4:9]
```

```
E=
```

```
1 2 3 4
```

```
5 6 7 8
```

```
6 7 8 9
```

Appending

Adding elements  
to a matrix

# Deleting Elements

By assigning nothing to these elements.

```
>>kt=[10 8 6 21 9]
```

```
kt=
```

```
10 8 6 21 9
```

```
>>kt(4) = []
```

```
Kt=
```

```
10 8 6 9
```

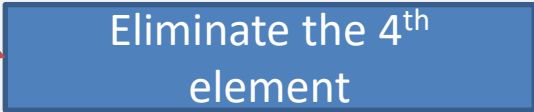
```
>>mtr = [5 56 75;23 54 12;64 12 76]
```

```
mtr=
```

```
5 56 75
```

```
23 54 12
```

```
64 12 76
```



Eliminate the 4<sup>th</sup>  
element

```
>>mtr(:,2:3) = []
```

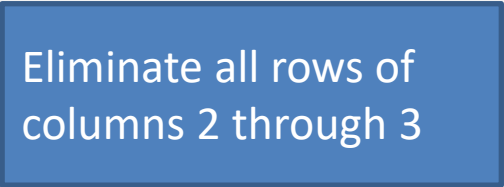
```
mtr=
```

```
5
```

```
23
```

```
64
```

```
>>
```



Eliminate all rows of  
columns 2 through 3



# Building function for handling arrays

Function	Description	Example
<code>length(A)</code>	returns number of elements in vector A	<pre>&gt;&gt;A=[ 5 9 2 4] &gt;&gt;length(A) ans =     4</pre>
<code>size(A)</code>	returns a row vector of [m,n]	<pre>&gt;&gt;A=[6 1 4 ;5 9 8 ] A=     6 1 4     5 9 8 &gt;&gt;size(A) ans =     2 3</pre>
<code>reshape(A,m,n)</code>	rearrange r x s matrix to a m x n matrix	<pre>&gt;&gt;B=reshape(A,3,2) B=     6 5     1 9     4 8</pre>

# contd...

Function	Description	Example
<code>diag(v)</code>	When <code>v</code> is a vector ,creates a square matrix with the elements of <code>v</code> in the diagonal.	<pre>&gt;&gt;v = [7 4 2]; &gt;&gt;A =diag(v) A= 7 0 0 0 4 0 0 0 2</pre>
<code>diag(A)</code>	When <code>A</code> is a matrix , creates a vector from the diagonal elements of <code>A</code> .	<pre>&gt;&gt;A= [1 2 3;4 5 6;7 8 9] A= 1 2 3 4 5 6 7 8 9 &gt;&gt;vec=diag(A) Vec= 1 5 9</pre>

# Strings And String As Variables

- In single quotes.
- Text on screen change to purple when 1<sup>st</sup> single quote is typed then turns into maroon when string is typed.
- Used in
  - o/p commands to display text messages.
  - in formatting commands of plots.(labels to axes , title and plots).
  - as i/p arguments for some functions.


```
>>Name = 'murshida'  
Name =  
murshida  
>>Name(2:8) = 'idhya '  
Name =  
midhya
```

# Character Strings

---

```
>> hi = ' hello';  
>> class = 'MATLAB';  
>> hi  
hi =  
    hello  
>> class  
class =  
MATLAB  
>> greetings = [hi class]  
greetings =  
    helloMATLAB  
>> vgreetings = [hi;class]  
vgreetings =  
    hello  
MATLAB
```

concatenation with **blank**  
or with “,”



semi-colon: join **vertically**



# Strings placed as matrix

---

- Done by typing ';' or by pressing 'Enter' key at the end of each row.
- Each row must be placed as strings.(ie enclosed in single quotes).
- Each row with words of equal size.add space for making words of equal size.

```
>>a=['abcd';'efg ']
```

```
a=
```

```
abcd
```

```
efg
```

```
>>a=['ab cd';'ef gh']
```

```
a=
```

```
ab cd
```

```
ef gh
```

# Built-in function **char**

- Create an array with rows that have same number of characters from an input of characters which are not of same length.
- MATLAB makes length of all rows equal to the longest line by adding spaces to the short line.

```
Variable_name = char( ' string 1 ' , ' string 2 ' , 'string 3 ' )
```

```
>>info = char('student name:', 'john smith', 'grade:', 'A+')
```

```
Info =
```

```
student name:
```

```
john smith
```

```
grade:
```

```
A+
```

```
>>X='156'
```

```
X=
```

```
156
```

```
>>Y=156
```

```
Y=
```

```
156
```

```
>>
```



both X & Y look the same .But X cannot be used for mathematical calculation.

# Mathematical Operations With Arrays



# Addition and subtraction

```
>>vectA = [8 5 4]; vectB = [10 2 7];
```

```
>>vectC = vectA + vectB
```

```
vectC =
```

```
18 7 11
```

```
>>A = [ 5 -3  8;9 2 10]
```

```
A =
```

```
5 -3  8
```

```
9  2 10
```

```
>>B = [10 7 4;-11 15 1]
```

```
B =
```

```
10  7  4
```

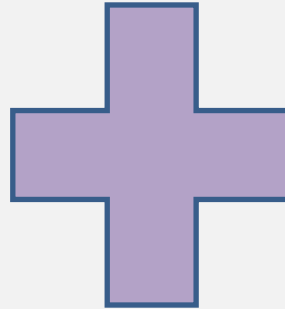
```
-11 15  1
```

```
>>A - B
```

```
ans=
```

```
-5 -10  4
```

```
20 -13  9
```



# Array multiplication

```
>>F = [1 3; 5 7]
```

```
F=
```

```
1 3
```

```
5 7
```

```
>>G = [ 4 2; 1 6 ]
```

```
G=
```

```
4 2
```

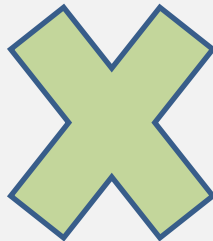
```
1 6
```

```
>>F*G
```

```
ans =
```

```
7 20
```

```
27 52
```



```
>>b=3
```

```
b=
```

```
3
```

```
>>b*F
```

```
ans =
```

```
3 9
```

```
15 21
```

# Array division

- **Determinants**

$$|A|$$

in matlab use `det(A)`

- **Identity matrix**

$$AI = IA = A$$

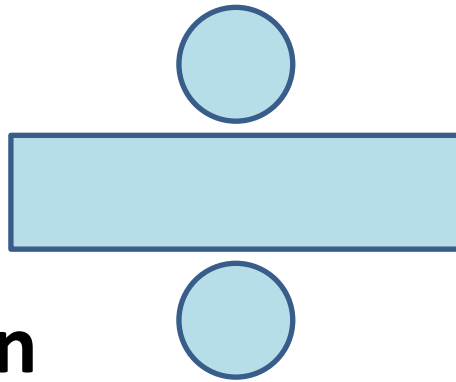
In matlab use `eye(A)`

- **Inverse of a matrix**

$$BA = AB = I$$

In matlab use `A^-1` or `inv(A)`

Contd..



➤ **Left division**

to solve matrix eqn  $AX = B$

$$X = A \setminus B$$

What happens: **B is divided by A**

➤ **Right division**

to solve matrix eqn  $XC = D$

$$X = D / C$$

What happens: **D is divided by A**

## example

Q. Solve three linear equations using

matrix  $4x - 2y + 6z = 8$

$$2x + 8y + 2z = 4$$

$$6x + 10y + 3z = 0$$

Eqns are of the form  $AX = B$  or  $XC = D$

$$\begin{bmatrix} 4 & -2 & 6 \\ 2 & 8 & 2 \\ 6 & 10 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ 4 \\ 0 \end{bmatrix}$$

or

$$\begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} 4 & 2 & 6 \\ -2 & 8 & 10 \\ 6 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 8 & 4 & 0 \end{bmatrix}$$

# Operators (Element by Element)

- $\cdot^*$  element-by-element multiplication
- $\cdot/$  element-by-element right division
- $\cdot\backslash$  element-by-element left division
- $\cdot^{\wedge}$  element-by-element exponentiation

# element-by-element operations

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

```
A=
```

```
2 6 3
```

```
5 8 4
```

```
>>B = [ 1 4 10; 3 2 7]
```

```
B=
```

```
1 4 10
```

```
3 2 7
```

```
>>A .*B
```

```
ans=
```

```
2 24 30
```

```
15 16 28
```

```
>>C = A/B
```

```
C=
```

```
2.0000 1.5000 0.3000
```

```
1.6667 4.0000 0.5714
```

```
>>B .*3
```

```
ans=
```

```
1 64 1000
```

```
27 8 343
```

```
>>A * B
```

```
???Error using ==>
```


```
Inner matrix dimensions must  
agree.
```

# Built-in function for analyzing arrays



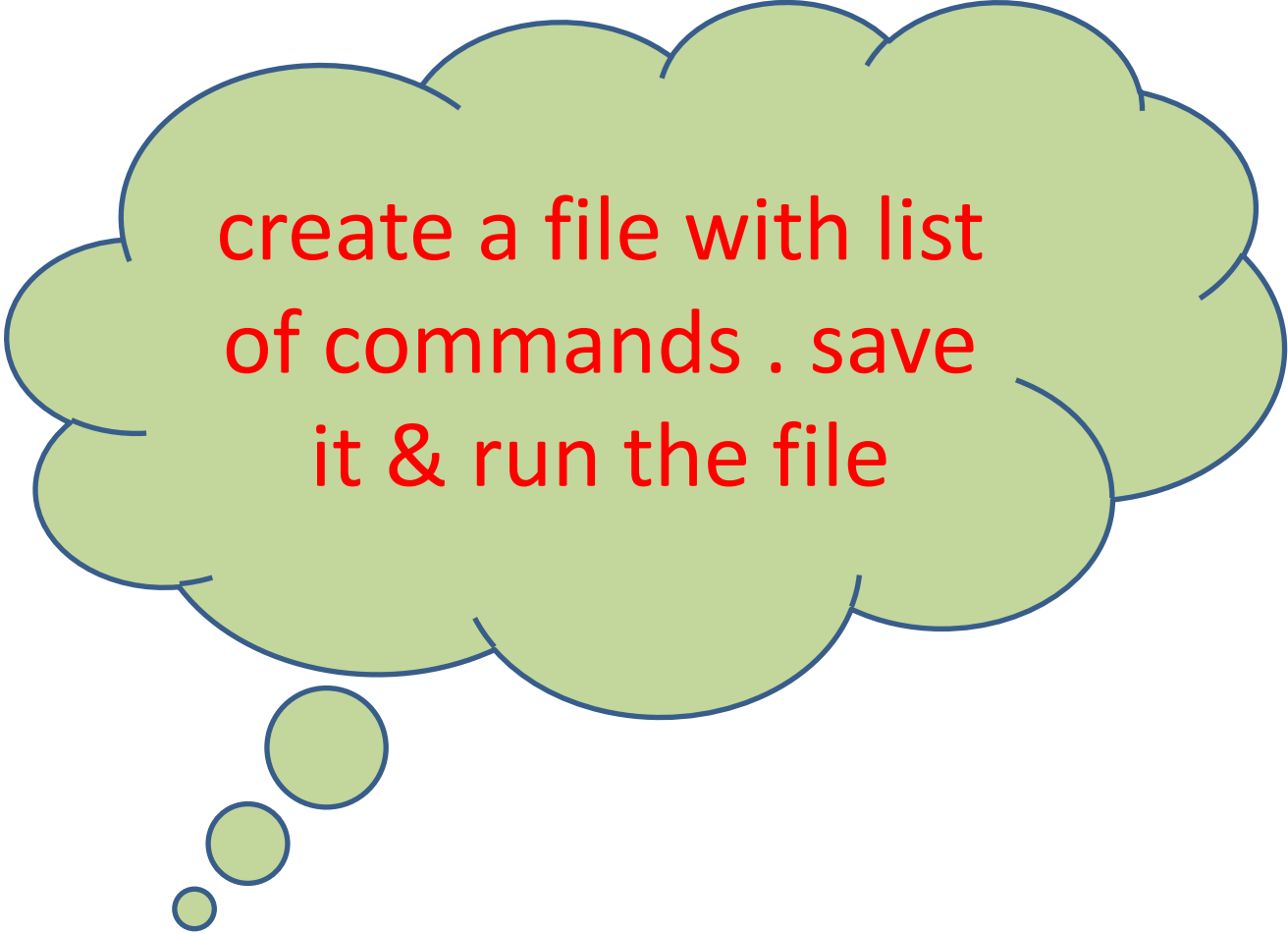
Function	Description	Example
mean(A)	mean value of the elements of the vector A.	<pre>&gt;&gt; A=[5 9 2 4]; &gt;&gt; mean(A) ans =     5</pre>
C=max(A)	C= is the largest element in vector A.( If A is a matrix, C =row vector containing the largest element of each column of A.)	<pre>&gt;&gt; A=[5 9 2 4 11 6 7 11 0 1]; &gt;&gt; C=max(A) C=     11</pre>
[d,n]=max(A)	d =largest element in vector A, n =position of the element ( the first if several have the max value ).	<pre>&gt;&gt; [d,n]=max(A) d =     11 n =     5</pre>
min(A)	Same as max(A) , but for smallest element.	<pre>&gt;&gt; A=[5 9 2 4]; &gt;&gt; min(A) ans =     2</pre>
[d,n]=min(A)	Same as [d,n]= max(A), but for the smallest element .	
Std(A)	standard deviation of the element of the vector A.	<pre>&gt;&gt; A=[5 9 2 4]; &gt;&gt; std(A) Ans=     2.9439</pre>

Function	Description	Example
dot (A)	scalar (dot) product of two vectors a and b. The vectors can each be row or column vector.	<pre>&gt;&gt;a=[1 2 3]; &gt;&gt;b=[ 3 4 5]; &gt;&gt;dot(A) Ans=     26</pre>
cross(A)	cross product of two vectors a and b , (a*b). The vectors must have 3 element.	<pre>&gt;&gt;a=[1 3 2]; &gt;&gt;b=[2 4 1]; &gt;&gt;cross(a,b) Ans=     -5 3 -2</pre>
sum(A)	sum of the elements of the vector A.	<pre>&gt;&gt;A=[5 9 2 4]; &gt;&gt;sum(A) ans=     20</pre>
sort(A)	arranges the elements of the vector A in ascending order.	<pre>&gt;&gt;A=[5 9 2 4 ]; &gt;&gt;sort(A) ans=     2 4 5 9</pre>
Median(A)	median value of the elements of the vector A.	<pre>&gt;&gt;A=[5 9 2 4]; &gt;&gt;median(A) ans=     4.5000</pre>



## Limitations of command window

- commands cannot be saved and executed again.
- not interactive.
- for change or correction all commands are to be entered & executed again.



create a file with list  
of commands . save  
it & run the file

# Script files

HOME FLOYS ARTS EDITOR PUBLISH VIEW

FILE NAVIGATE EDIT BREAKPOINTS RUN

Insert fx fi Comment % % Indent Breakpoints Run Run and Advance Run and Time

Create new document (Ctrl+N)

Go To Find

C:\Program Files\MATLAB\R2018a\bin

- Current Folder
- worker.bat
  - mw\_mpiexec.bat
  - mexutils.pm
  - mexsetup.pm
  - mexext.bat
  - mex.pl
  - mex.bat
  - mcc.bat
  - mbuild.bat
  - matlab.exe
  - lcdata\_utf8.xml
  - lcdata.xsd
  - lcdata.xml
  - deploytool.bat
  - win64
  - win32
  - util

Editor - Untitled

Untitled x +

```
1
```

Command Window

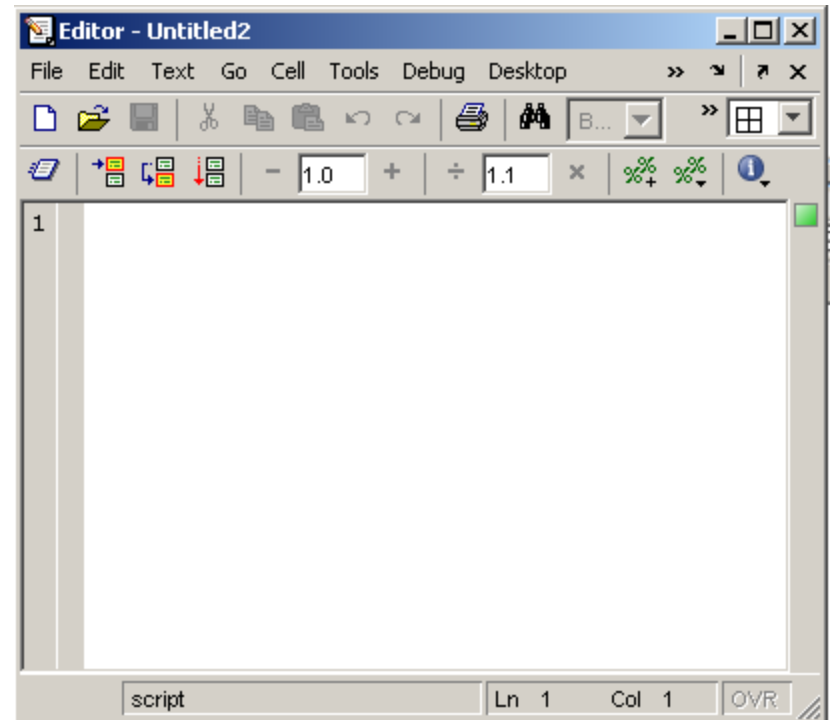
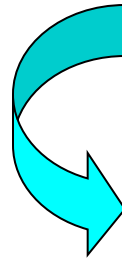
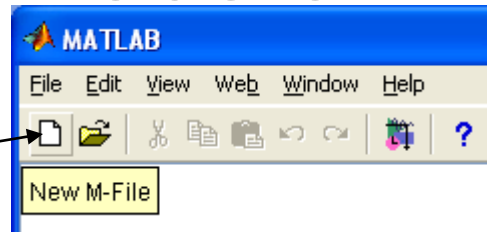
```
fx >>
```

Details

Select a file to view details

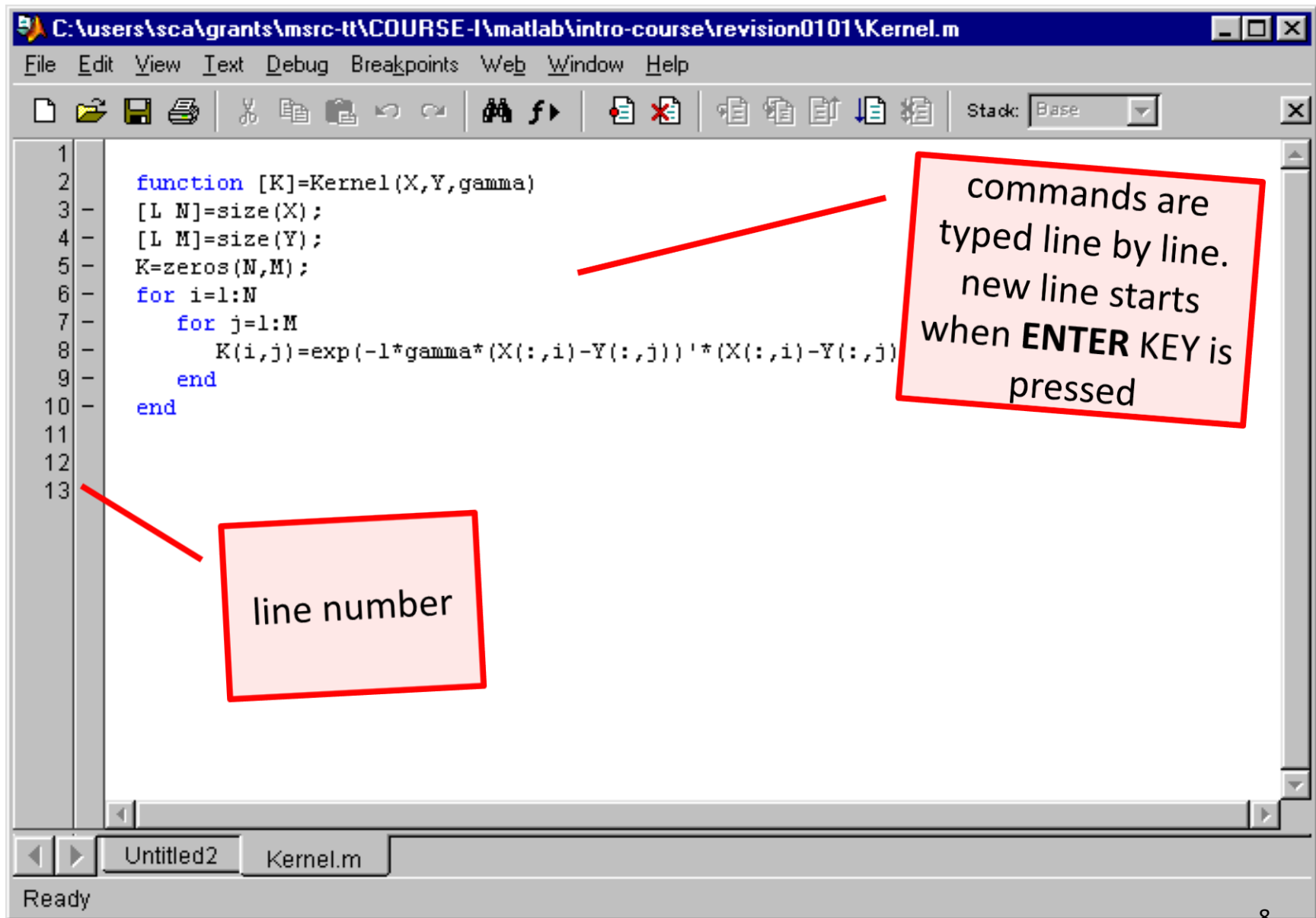
# Use of M-File

Click to create a new M-File



- Extension “.m”
- A text file containing script or function or program to run

# m-file Editor Window



# SAVING A SCRIPT FILE

FILE

```
graph TD; A[FILE] --> B[SAVE As..]; B --> C[enter name of file with .m extension];
```

SAVE As..

enter name of file  
with .m extension



# Running a script file






typing its name in  
command window  
& pressing **Enter**



clicking **Run** icon  
on Editor window

# Input to a script file

- 1.** Variable is defined & assigned value in the script file. 
- 2.** Variable is defined & assigned value in the command window. 
- 3.** Variable is defined in script file, but a specific value is entered in the command window when the script file is executed. 



```
%this script file calculates the avg points scored in 3
%games
game1=75;
game2=93;
game3=68;
avg_point=(game1+game2+game3)/3
```

```
>>ex1
```

```
avg_points=
  78.6667
```



```
%this script file calculates the avg points scored in 3
%games .
%values of game1,game2 & game3 is done in
%command window.
avg_point=(game1+game2+game3)/3
```

```
>>game1=75;
>>game2=90;
>>game3=68
>>ex2
avg_points=
  78.6667
```



```
%this script file calculates the avg points scored in 3
%games .
%values of game1,game2 & game3 is assigned using
%input command.
game1=input('enter points scored in game1 : ');
game2=input('enter points scored in game2 : ');
game3=input('enter points scored in game3 : ');
avg_point=(game1+game2+game3)/3
```



```
>>ex2
```

```
enter points scored in game1 : 75
```

```
enter points scored in game2 : 93
```

```
enter points scored in game3 : 68
```

```
avg_points=
```

```
78.6667
```



# Initializing with Keyboard Input

- The **input** function displays a prompt string in the Command Window and then waits for the user to respond.

```
my_val = input( 'Enter an input value: ' );
```

```
in1 = input( 'Enter data: ' );
```

```
in2 = input( 'Enter data: ', 's' );
```



## Output Commands

- ***disp*** command

displays o/p on the screen

```
disp(name of a variable)
```

or

```
disp('text as string')
```

- ***fprintf*** command

to display o/p(text & data) on the screen or save it to a file

```
fprint('text typed in as string')
```

# *Scripts and Functions*

- There are two kinds of M-files:
  - **Scripts**, which do not accept input arguments or return output arguments. They operate on data in the workspace.
  - **Functions**, which can accept input arguments and return output arguments. Internal variables are local to the function.



# if/elseif/else Statement

```
>> A = 2; B = 3;
>> if A > B
    'A is bigger'
elseif A < B
    'B is bigger'
elseif A == B
    'A equals B'
else
    error('Something odd is happening')
end
ans =
B is bigger
```

# switch Statement

```
>> n = 8
```

```
n =
```

```
8
```

```
>> switch(rem(n,3))
```

```
    case 0
```

```
        m = 'no remainder'
```

```
    case 1
```

```
        m = 'the remainder is one'
```

```
    case 2
```

```
        m = 'the remainder is two'
```

```
    otherwise
```

```
        error('not possible')
```

```
    end
```

```
m =
```

```
the remainder is two
```

# For Loop

```
>> for i = 2:5
    for j = 3:6
        a(i,j) = (i + j)^2
    end
end
```

```
>> a
```

```
a =
```

0	0	0	0	0	0
0	0	25	36	49	64
0	0	36	49	64	81
0	0	49	64	81	100
0	0	64	81	100	121

# while Loop

```
>> b = 4; a = 2.1; count = 0;
>> while b - a > 0.01
    a = a + 0.001;
    count = count + 1;
end
>> count
count =
    1891
```

# Common OS Commands

- `ls` / `dir` provide a directory listing of the current directory
- `pwd` shows the current directory

# ***Algorithm***

The word “**algorithm**” relates to the name of the mathematician Al-Khwarizmi, which means a procedure or a technique. Software Engineer commonly uses an algorithm for planning and solving problems. An algorithm is a sequence of steps to solve a particular problem or an algorithm is an ordered set of unambiguous steps that produce a result and terminate in a finite time.

## **The algorithm has the following characteristics**

- **Input:** An algorithm may or may not require input
- **Output:** Each algorithm is expected to produce at least one result
- **Definiteness:** Each instruction must be clear and unambiguous.
- **Finiteness:** If the instructions of an algorithm are executed, the algorithm should terminate after a finite number of steps.

# The algorithm and flowchart include the following three types of control structures.

1. **Sequence:** In the sequence structure, statements are placed one after the other and the execution takes place starting from up to down.
2. **Branching (Selection):** In branch control, there is a condition and according to a condition, a decision of either TRUE or FALSE is achieved. In the case of TRUE, one of the two branches is explored; but in the case of the FALSE condition, the other alternative is taken. Generally, the 'IF-THEN' is used to represent branch control.
3. **Loop (Repetition):** The Loop or Repetition allows a statement(s) to be executed repeatedly based on certain loop conditions e.g. WHILE, FOR loops.

# Advantages of algorithm

- It is a step-wise representation of a solution to a given problem, which makes it easy to understand.
- An algorithm uses a definite procedure.
- It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.
- Every step in an algorithm has its own logical sequence so it is easy to debug.



# HOW TO WRITE ALGORITHMS

- Step 1 (Define your algorithms input):** Many algorithms take in data to be processed, e.g. to calculate the area of rectangle input may be the rectangle height and rectangle width.
- Step 2 (Define the variables):** The algorithm's variables allow you to use it for more than one place. We can define two variables for rectangle height and rectangle width as HEIGHT and WIDTH (or H & W). We should use meaningful variable names e.g. instead of using H & W use HEIGHT and WIDTH as a variable name.
- Step 3 (Outline the algorithm's operations):** Use the input variable for computation purposes, e.g. to find the area of the rectangle multiply the HEIGHT and WIDTH variables and store the value in a new variable (say) AREA. An algorithm's operations can take the form of multiple steps and even branches, depending on the value of the input variables.
- Step 4 (Output the results of your algorithm's operations):** In the case of the area of the rectangle output will be the value stored in variable AREA. if the input variables described a rectangle with a HEIGHT of 2 and a WIDTH of 3, the algorithm would output the value of 6.





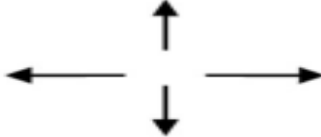




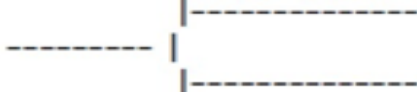
# Flow Chart

The first design of a flowchart goes back to 1945 which was designed by John Von Neumann. Unlike an algorithm, Flowchart uses different symbols to design a solution to a problem. It is another commonly used programming tool. By looking at a Flow chart one can understand the operations and sequence of operations performed in a system. A flowchart is often considered a blueprint of a design used for solving a specific problem.

## **Advantages of flowchart:**

- The flowchart is an excellent way of communicating the logic of a program.
- Easy and efficient to analyze the problem using a flowchart.
- During the program development cycle, the flowchart plays the role of a blueprint, which makes the program development process easier.
- After the successful development of a program, it needs continuous timely maintenance during its operation. The flowchart makes program or system maintenance easier.
- It is easy to convert the flowchart into any programming language code.

**Flowchart** is a diagrammatic /Graphical representation of a sequence of steps to solve a problem. To draw a flowchart following standard symbols are used:

Symbol Name	Symbol	function
Oval		Used to represent the start and end of a flowchart
Parallelogram		Used for input and output operation
Rectangle		Processing: Used for arithmetic operations and data-manipulations
Diamond		Decision making. Used to represent the operation in which there are two/three alternatives, true and false, etc
Arrows		Flow line Used to indicate the flow of logic by connecting symbols
Circle		Page Connector
		Off Page Connector
		Predefined Process /Function Used to represent a group of statements performing one processing task.
		Preprocessor
		Comments

# Mathematical Operators

Operator	Meaning	Example
+	Addition	$A + B$
-	Subtraction	$A - B$
*	Multiplication	$A * B$
/	Division	$A / B$
^	Power	$A^3$ for $A^3$
%	Reminder	$A \% B$

# Relational Operators

Operator	Meaning	Example
<	Less than	$A < B$
<=	Less than or equal to	$A <= B$
= or ==	Equal to	$A = B$
# or !=	Not equal to	$A \# B$ or $A != B$
>	Greater than	$A > B$
>=	Greater than or equal to	$A >= B$

# Logical Operators

Operator	Example	Meaning
AND	$A < B$ AND $B < C$	Result is True if both $A < B$ and $B < C$ are true else false
OR	$A < B$ OR $B < C$	Result is True if either $A < B$ or $B < C$ are true else false
NOT	NOT ( $A > B$ )	Result is True if $A > B$ is false else true

# Selection Control Statements

Selection Control	Example	Meaning
IF ( Condition ) Then ... ENDIF	IF ( X > 10 ) THEN Y=Y+5 ENDIF	If condition X>10 is True execute the statement between THEN and ENDIF
IF ( Condition ) Then ... ELSE ..... ENDIF	IF ( X > 10 ) THEN Y=Y+5 ELSE Y=Y+8 Z=Z+3 ENDIF	If condition X>10 is True execute the statement between THEN and ELSE otherwise execute the statements between ELSE and ENDIF

# Loop control Statements

Selection Control	Example	Meaning
WHILE (Condition) DO .. .. ENDDO	WHILE ( X < 10) DO print x x=x+1 ENDDO	Execute the loop as long as the condition is TRUE
DO .... ... UNTILL (Condition)	DO print x x=x+1 UNTILL ( X >10)	Execute the loop as long as the condition is false



# Examples of Algorithms and Flow Charts



## Example1: Algorithm & Flowchart to find the sum of two numbers

### Algorithm

Step-1 Start

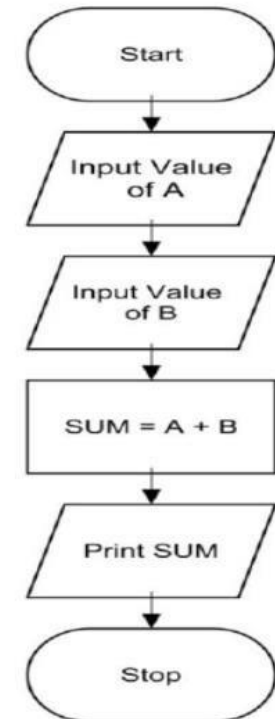
Step-2 Input first numbers say A

Step-3 Input second number say B

Step-4  $SUM = A + B$

Step-5 Display SUM

Step-6 Stop



# Another Method

## Algorithm

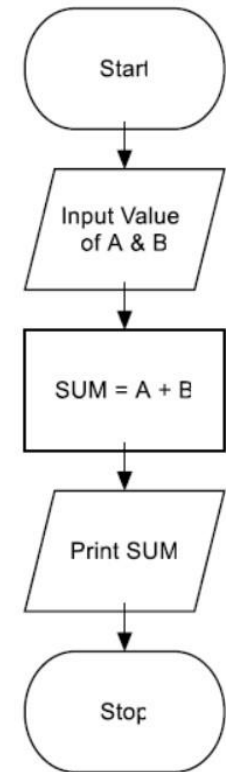
Step-1 Start

Step-2 Input two numbers say A & B

Step-3  $SUM = A + B$

Step-4 Display SUM

Step-5 Stop



# Example2: Algorithm & Flowchart to convert temperature from Celsius to Fahrenheit

C : temperature in Celsius  
F : temperature Fahrenheit

## Algorithm

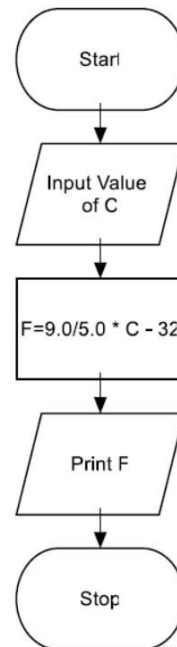
Step-1 Start

Step-2 Input temperature in Celsius say C

Step-3  $F = (9.0/5.0 \times C) + 32$

Step-4 Display Temperature in Fahrenheit F

Step-5 Stop



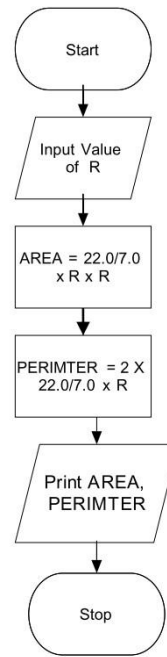
- **Exercise 1: Write an algorithm and flow chart to convert temperature from Fahrenheit to Celsius.**

# Example3: Algorithm & Flowchart to find the Area and Perimeter of the Circle

R : Radius of Circle  
AREA : Area of Circle  
PERIMETER : Perimeter of Circle

## Algorithm

- Step-1 Start
- Step-2 Input Radius of Circle say R
- Step-3 Area =  $22.0/7.0 \times R \times R$
- Step-4 PERIMETER =  $2 \times 22.0/7.0 \times R$
- Step-5 Display AREA, PERIMETER
- Step-6 Stop



- Exercise 2: Write an algorithm and flow chart to find the Area and Perimeter of the Rectangle.
- Exercise 3: Write an algorithm and flow chart to find the Area and Perimeter of a Square.
- Exercise 4: Write an algorithm and flow chart to find the Area and Perimeter of the Triangle

# Example4: Algorithm and flow chart to find the average of three numbers.

## Algorithm

Step1 : Start

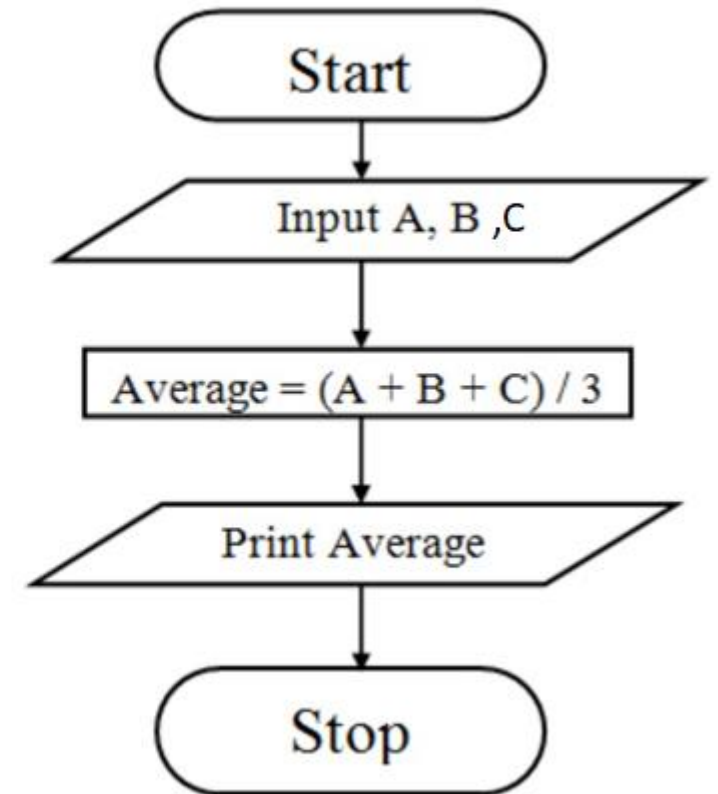
Step 2 : Enter Three Numbers A, B and C

Step 3 : Compute Average =  $(A+B+C)/3$

Step 4 : Print Average

Step 5 : Stop

## Flow Chart



# Example 5: Algorithm to find the larger of two numbers.

## Algorithm

Step1: Start

Step 2: Enter two numbers A and B

Step 3: Check if A is greater than B if yes go to Step 4 else go to Step 5

Step 4: Print A is greater than B

Step 5: Check if B is greater than A if yes go to Step 6 else go to Step 7

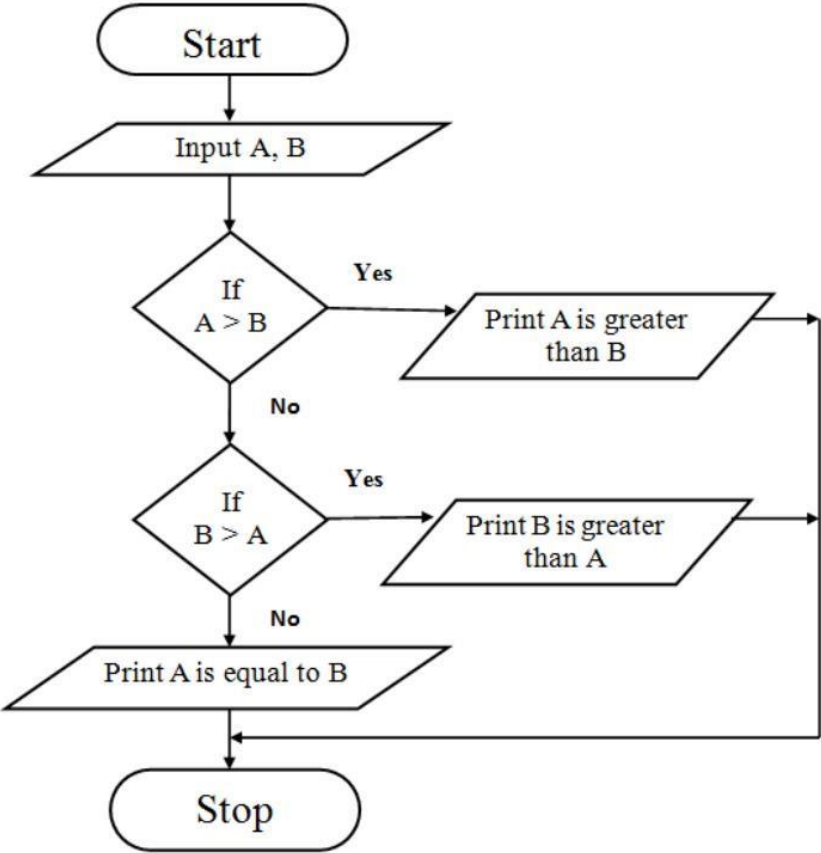
Step 6: Print B is greater than A

Step 7: Print A is equal to B

Step 8: Stop

# Example 5: Flow chart to find the larger of two numbers.

Flow Chart





# Example 6: Algorithm to find the factorial of a number.

## Algorithm

**Step 1: Start**

**Step 2: Read N**

**Step 3: [Initialize all  
counters] Set FACT= 1, i  
= 1**

**Step 4: Compute Fact =  
Fact \* I Increment i**

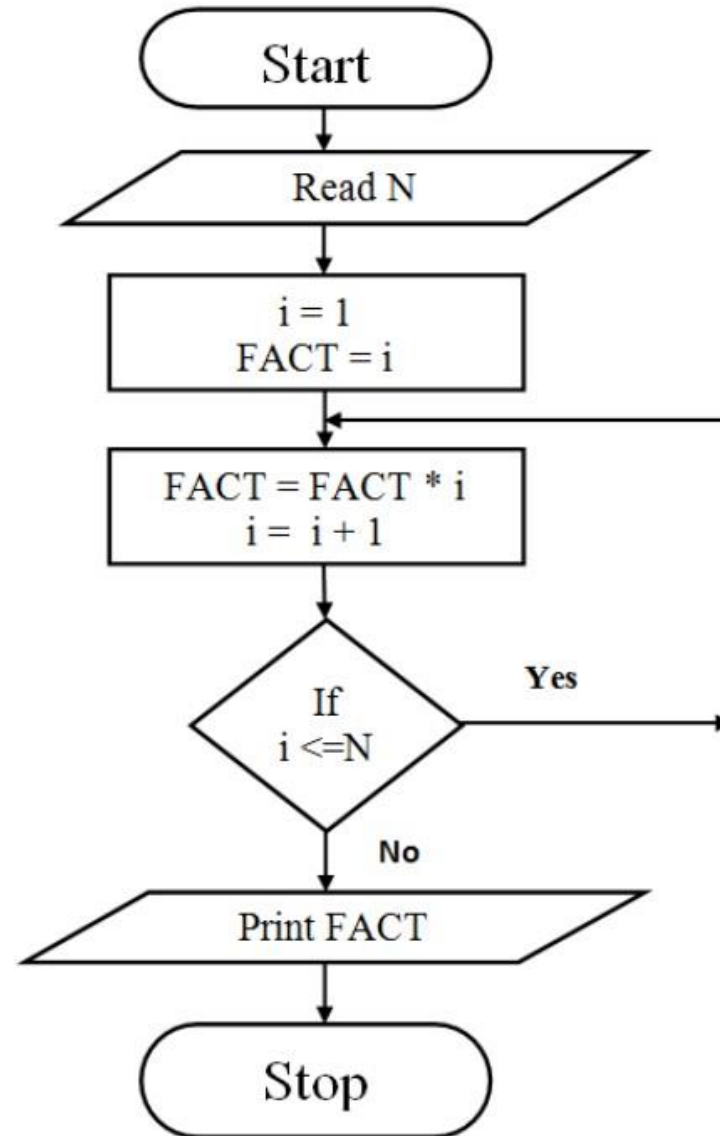
**Step 5: Check if  $i \leq N$  if  
true repeat step 4 if false  
go to step 6**

**Step 6: Print fact**

**Step 7: Stop**

Example 6:  
Flow chart to  
find the  
factorial of a  
number.

Flow Chart



Example 7:  
Algorithm to  
find the  
largest of  
three

**Algorithm**

Step-1 Start

Step-2 Read three numbers say num1,num2, num3

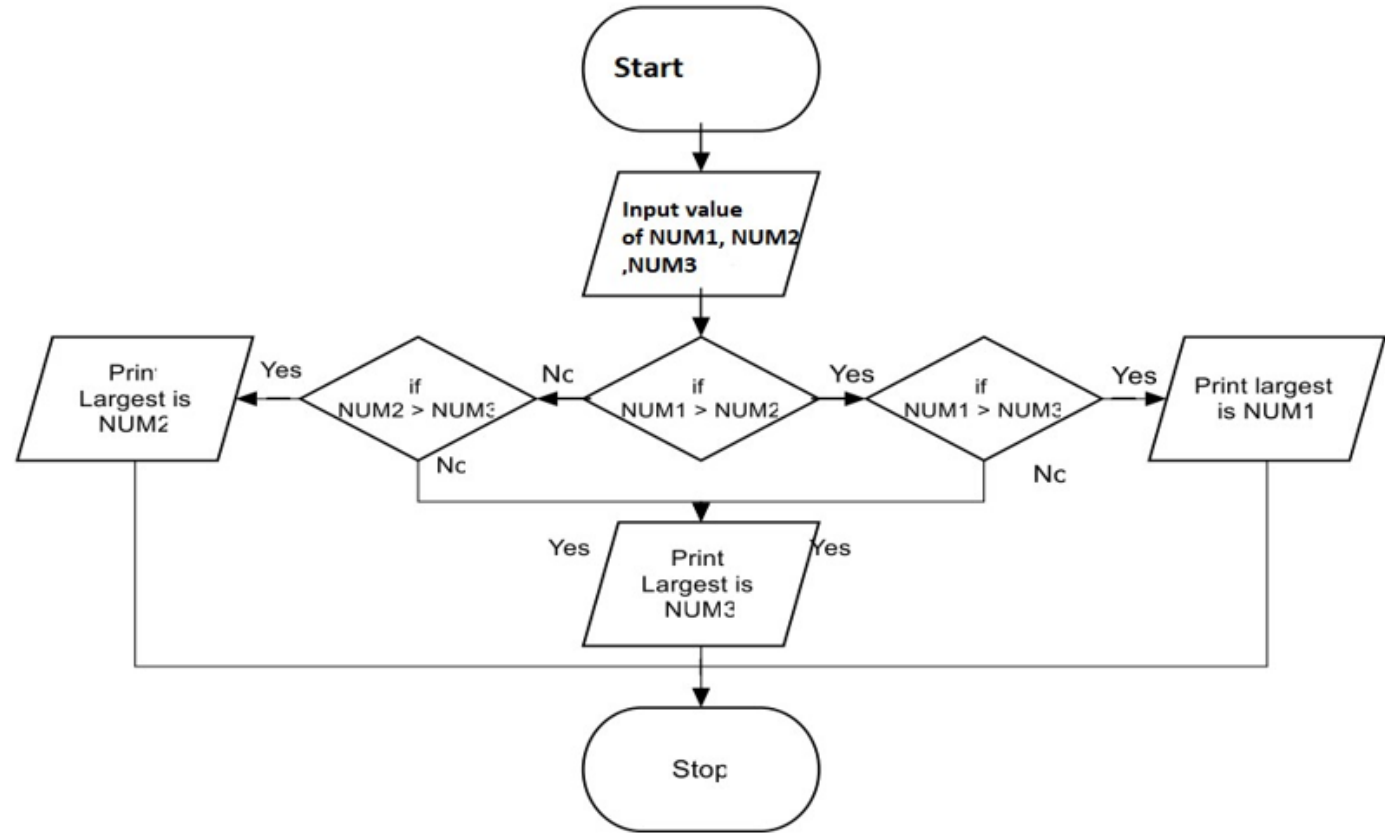
Step-3 if num1>num2 then go to step-5

Step-4 IF num2>num3 THEN  
    print num2 is largest  
ELSE  
    print num3 is largest  
ENDIF  
GO TO Step-6

Step-5 IF num1>num3 THEN  
    print num1 is largest  
ELSE  
    print num3 is largest  
ENDIF

Step-6 Stop

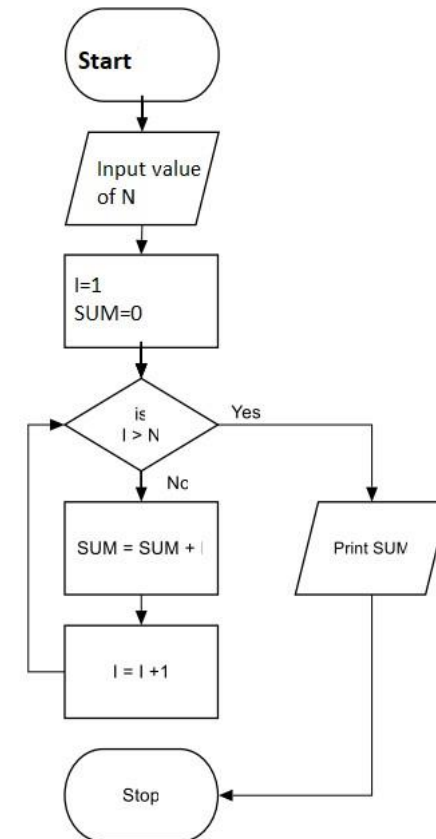
Example 7:  
Flowchart to  
find the  
largest of  
three



# Example 8: Algorithm & Flowchart to find the sum of series $1+2+3+\dots+N$ .

## Algorithm

- Step-1 Start
- Step-2 Input Value of N
- Step-3  $I = 1, \text{SUM} = 0$
- Step-4 IF ( $I > N$ ) THEN  
GO TO Step-8  
ENDIF
- Step-5  $\text{SUM} = \text{SUM} + I$
- Step-6  $I = I + 1$
- Step-7 Go to step-4
- Step-8 Display value of SUM
- Step-9 Stop





# Example to Programming in MATLAB



Handwritten mathematical notes on a chalkboard background, including:

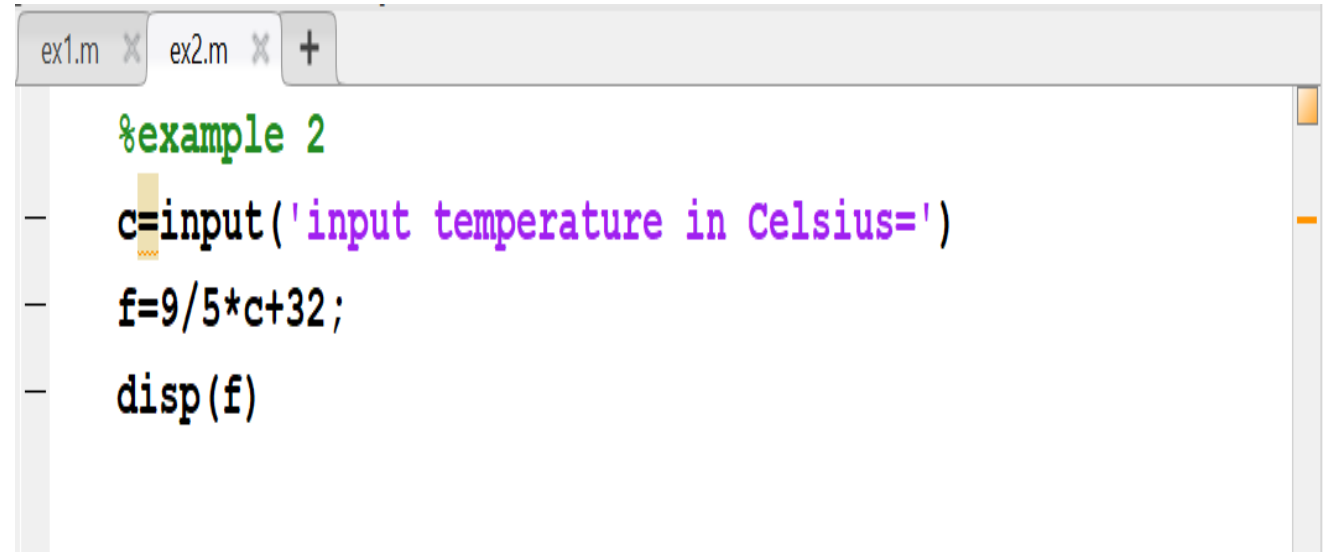
- $D(x) = -2 + 3 + 4 \cdot 31447$
- $\sqrt{a^2 + b^2} = x^2 \cdot \pi x$
- $x^2 + y^2 = ab + 4c$
- $A \cap B, \frac{24+x}{y} + \frac{a^2 + b^2}{c} + \frac{1}{x^2}$
- $mem = 384 + n^{20}$
- $x = 9.22$
- $\sum_{x=2}^{n=14} N_{30} \cdot x - \frac{1}{2} [984 + x^2 + p]$
- Matrix:  $\begin{bmatrix} 010112 \\ 010002 \\ 200110 \\ 011002 \end{bmatrix}$
- Graphs of functions and geometric shapes.

# Example 1: Program to find the sum of two numbers

```
ex1.m x +
1  % example 1
2  a=input('the value of a=')
3  b=input('the value of b=')
4  sum=a+b;
5  disp(sum)
```



## Example 2: Program to convert temperature from Celsius to Fahrenheit



```
ex1.m x ex2.m x +
%example 2
c=input('input temperature in Celsius=')
f=9/5*c+32;
disp(f)
```



Example 3:  
program to  
find the Area  
and  
Perimeter of  
the Circle

```
ex1.m x ex2.m x ex3.m x +
1 % example 3
2 r=input('input the rudius of circle=')
3 area=22/7*r;
4 disp(area)
```



Example 4:  
program to  
find the  
average of  
three  
numbers.

```
ex1.m x ex2.m x ex3.m x ex4.m x +
1  % example 4
2  a=input('the value of a=')
3  b=input('the value of b=')
4  c=input('the value of c=')
5  ave=(a+b+c)/3;
6  disp(ave)
```

Example 5:  
Program to  
find the  
larger of two  
numbers.

```
ex1.m x ex2.m x ex3.m x ex4.m x ex5.m x Untitled3* x +
1      % example 5
2      a=input('the value of a=')
3      b=input('the value of b=')
4      if a>b
5          'the larger number a='
6          disp(a)
7      else
8          'the larger number b='
9          disp(b)
10     end
```

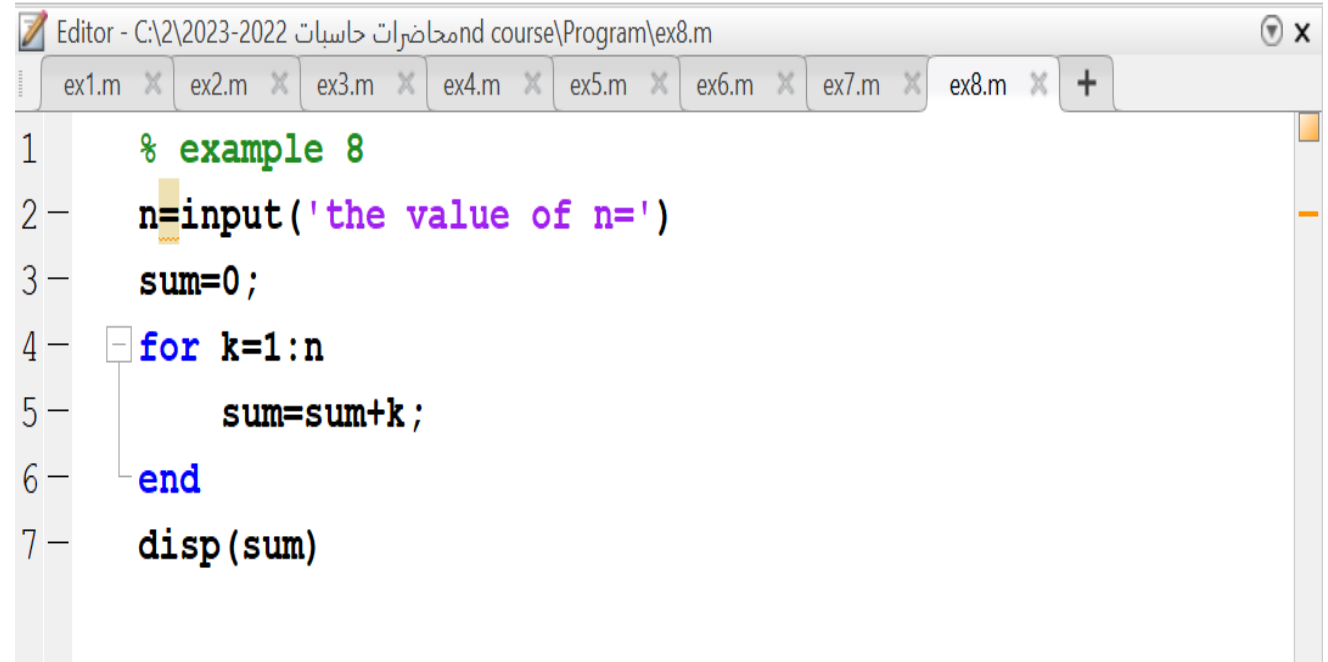
# Example 6: Program to find the factorial of a number

```
Editor - C:\2\2023-2022 محاضرات حاسبات and course\Program\ex6.m
ex1.m x ex2.m x ex3.m x ex4.m x ex5.m x ex6.m x +
1  % example 6
2  n=input('the value of n=')
3  fac=1;
4  for k=1:n
5      fac=fac*k;
6  end
7  disp(fac)
```

# Example 7: Program to find the largest of three

```
Editor - C:\2023-2022 محاضرات حسابيات\course\Program\ex7.m
ex1.m x ex2.m x ex3.m x ex4.m x ex5.m x ex6.m x ex7.m x +
1 % example 7
2 a=input('the value of a=')
3 b=input('the value of b=')
4 c=input('the value of c=')
5 if a>b
6     if a>c
7         ' the largest number a='
8         disp(a)
9     else
10        ' the largest number c='
11        disp(c)
12    end
13 else
14     if b>c
15         ' the largest number b='
16         disp(b)
17     else
18         ' the largest number c='
19         disp(c)
20    end
21 end
```

Example 8:  
Program to  
find the sum  
of series  
 $1+2+3+\dots+N$ .



```
Editor - C:\2023-2022 محاضرات حاسبات and course\Program\ex8.m
ex1.m x ex2.m x ex3.m x ex4.m x ex5.m x ex6.m x ex7.m x ex8.m x +
1 % example 8
2 n=input('the value of n=')
3 sum=0;
4 for k=1:n
5     sum=sum+k;
6 end
7 disp(sum)
```

# **Selection Statements**

# Relational Expressions

- These expressions can use both *relational operators*, which relate two expressions of compatible types, and *logical operators*, which operate on logical operands.
- The relational operators in MATLAB are:

Operator Meaning

<b>&gt;</b>	<b>greater than</b>
<b>&lt;</b>	<b>less than</b>
<b>&gt;=</b>	<b>greater than or equals</b>
<b>&lt;=</b>	<b>less than or equals</b>
<b>==</b>	<b>equality</b>
<b>~=</b>	<b>inequality</b>



All concepts should be familiar, although the operators used may be different from those used in other programming languages, or in mathematics classes. In particular, it is important to note that the operator for equality is two consecutive equal signs, not a single equal sign (recall that the single equal sign is the assignment operator).

For numerical operands, the use of these operators is straightforward. For example,  $3 < 5$  means “3 less than 5,” which is conceptually a true expression. However, in MATLAB, as in many programming languages, *logical true* is represented by integer 1, and *logical false* is represented by integer 0. So, the expression  $3 < 5$  has the value 1 in MATLAB. Displaying the result of expressions like this in the Command Window demonstrates the values of the expressions.

```
>> 3 < 5
```

```
ans =
```

```
1
```

```
>> 9 < 2
```

```
ans =
```

```
0
```

However, in the Workspace Window, the value shown for the result of these expressions would be true or false. The type of result is **logical**. Mathematical operations could be performed on the resulting 1 or 0.

```
>> 5 < 7
```

```
ans =
```

```
1
```

```
>> ans + 3
```

```
ans =
```

```
4
```

Comparing characters, for example 'a' < 'c', is also possible. Characters are compared using their ASCII equivalent values. So, 'a' < 'c' is conceptually a true expression, because the character 'a' comes before the character 'c'.

- `>> 'a' < 'c'`
- `ans =`
- `1`
- The logical operators are:

Operator	Meaning
<b>  </b>	<b>or for scalars</b>
<b>&amp;&amp;</b>	<b>and for scalars</b>
<b>~</b>	<b>not</b>

All logical operators operate on logical or Boolean operands. The **not** operator is a unary operator; the others are binary. The **not** operator will take a Boolean expression, which is conceptually true or false, and give the opposite value. For example,  $\sim(3 < 5)$  is conceptually false since  $(3 < 5)$  is true. The **or** operator has two Boolean expressions as operands. The result is true if either or both of the operands are true, and false only if both operands are false. The **and** operator also operates on two Boolean operands. The result of an **and** expression is true only if both operands are true; it is false if either or both are false. In addition to these logical operators, MATLAB also has a function **xor**, which is the exclusive or function. It returns logical true if one (and only one) of the arguments is true. For example, in the following only the first argument is true, so the result is true:

```
>> xor(3 < 5, 'a' > 'c')
ans =
     1
```

In this example, both arguments are true so the result is false:

```
>> xor(3 < 5, 'a' < 'c')
ans =
     0
```

Given the logical values of true and false in variables  $x$  and  $y$ , the **truth table** (see Table 3.1) shows how the logical operators work for all combinations. Note that the logical operators are commutative (e.g.,  $x \parallel y$  is the same as  $y \parallel x$ ).

**Table 3.1** Truth Table for Logical Operators

$x$	$y$	$\sim x$	$x \parallel y$	$x \&\& y$	$xor(x,y)$
true	true	false	true	true	false
true	false	false	true	false	true
false	false	true	false	false	false

Examples



Example 1: Write a program to evaluate the following function  $f(x) = \frac{6}{x}$

```
x=input('x=');  
if x~=0  
    z=6/x  
else  
    disp('there is no solution')  
end
```

Example 2: Write a program to evaluate the following function

$$f(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{if } x < 0 \end{cases}$$



Example 3: Write a program to find the area of rectangle using the **function** code.

Example 4: Write a program to find a number of odd and even in a vector x using function code.

Example 5: Write a program to evaluate the following series using function code:  $z = 1 + x + x^2 + \dots + x^n = \sum_{k=0}^n x^k$ , for  $n=10$

Example 6: Write a program to calculate the following equation:

$$w = \sum_{i=1}^{10} i^2 + \prod_{k=1}^{15} 2k - \sum_{j=1}^5 \prod_{r=0}^8 j^r$$

Example 7: Write a program to calculate the following equation:

$$w = \sum_{j=1}^5 \prod_{\substack{r=0 \\ j \neq r}}^8 j^r$$