1. INTRODUCTION TO MATLAB

1.Basic Syntax

Use of Semicolon (;) in MATLAB: Semicolon (;) indicates end of statement. However, if you want to suppress and hide the MATLAB output for an expression, add a semicolon after the expression.

For example,

```
x = 3;
y = x + 5
```

Adding Comments:

The percent symbol (%) is used for indicating a comment line. For example,

```
x = 9     % assign the value 9 to x
```

You can also write a block of comments using the block comment operators % { and % }. The MATLAB editor includes tools and context menu items to help you add, remove, or change the format of comments.

Commonly used Operators and Special Characters: MATLAB supports the following commonly used operators and special characters:

Operator	Purpose
+	Plus; addition operator.
-	Minus; subtraction operator.
*	Scalar and matrix multiplication operator.
.*	Array multiplication operator.
^	Scalar and matrix exponentiation operator.
.^	Array exponentiation operator.
\	Left-division operator.
/	Right-division operator.
.\	Array left-division operator.
./	Array right-division operator.
:	Colon; generates regularly spaced elements and represents an entire row or column.
( )	Parentheses; encloses function arguments and array indices; overrides precedence.
[ ]	Brackets; enclosures array elements.
.	Decimal point.
…	Ellipsis; line-continuation operator
,	Comma; separates statements and elements in a row
;	Semicolon; separates columns and suppresses display.
%	Percentsign; designates a comment and specifies formatting.
_	Quote sign and transpose operator.
._	Non-conjugated transpose operator.
=	Assignment operator.

Special Variables and Constants:

MATLAB supports the following special variables and constants:

Name	Meaning
ans	Most recent answer.
eps	Accuracy of floating-point precision.
i,j	The imaginary unit √-1.
Inf	Infinity.
NaN	Undefined numerical result (not a number).

pi	The number π

Naming Variables:

- Variable names consist of a letter followed by any number of letters, digits or underscore.
- MATLAB is case-sensitive.
- Variable names can be of any length, however, MATLAB uses only first N characters, where N is given by the function namelengthmax.

Saving Your Work:

The save command is used for saving all the variables in the workspace, as a file with .mat extension, in the current directory.

For example,

```
save myfile
```

You can reload the file anytime later using the load command.

```
load myfile
```

2. Variables

In MATLAB environment, every variable is an array or matrix. You can assign variables in a simple way. For example,

MATLAB will execute the above statement and return the following result:

```
x = 3       % defining x and initializing it with a value
x =
     3
```

It creates a 1-by-1 matrix named x and stores the value 3 in its element. Let us check another example,

```
x = sqrt(16)       % defining x and initializing it with an expression
```

MATLAB will execute the above statement and return the following result:

```
x =
     4
```

Please note that:

- Once a variable is entered into the system, you can refer to it later. Variables must have values before they are used.
- When an expression returns a result that is not assigned to any variable, the system assigns it to a variable named ans, which can be used later.

For example,

```
sqrt(78)
```

MATLAB will execute the above statement and return the following result:

```
ans =
    8.8318
```

You can use this variable ans:

```
9876/ans
```

MATLAB will execute the above statement and return the following result:

```
ans =
   1.1182e+03
```

Let's look at another example:

```
x = 7 * 8;
y = x * 7.89
```

MATLAB will execute the above statement and return the following result:

```
y =
  441.8400
```

Multiple Assignments: You can have multiple assignments on the same line.

For example, `a = 2; b = 7; c = a * b;`

MATLAB will execute the above statement and return the following result:

```
c =
    14
```

Who: The who command displays all the variable names you have used.

```
who
```

MATLAB will execute the above statement and return the following result:

```
Your variables are:
a     ans  b c      x      y
```

The whos command displays little more about the variables:

Clear: The clear command deletes all (or the specified) variable(s) from the memory.

clear x

Long Assignments: Long assignments can be extended to another line by using an ellipses (...).

For example,

```
initial_velocity = 0;

acceleration = 9.8;
time = 20;

final_velocity = initial_velocity ...
    + acceleration * time
```

MATLAB will execute the above statement and return the following result:

```
final_velocity =
   196
```

3. Format Command

The format Command: By default, MATLAB displays numbers with four decimal place values. This is known as short format.

However, if you want more precision, you need to use the format command.

1. **format long:** The format long command displays 16 digits after decimal.

For example:

```
format long
x = 7 + 10/3 + 5 ^ 1.2
```

MATLAB will execute the above statement and return the following result:

```
x =
  17.231981640639408
```

2. **format short:**

```
format short
x = 7 + 10/3 + 5 ^ 1.2
```

MATLAB will execute the above statement and return the following result:

```
x =
   17.2320
```

3. format blank: The format bank command rounds numbers to two decimal places.

For example,

```
format bank
daily_wage =
177.45;
weekly_wage = daily_wage * 6
```

MATLAB will execute the above statement and return the following result:

```
weekly_wage =

       1064.70
```

4. format short e: MATLAB displays large numbers using exponential notation. The format short e command allows displaying in exponential form with four decimal places plus the exponent.

For example,

```
format short e
4.678 * 4.9
```

MATLAB will execute the above statement and return the following result:

```
ans =
   2.2922e+01
```

4. format long e: The format long e command allows displaying in exponential form with four decimal places plus the exponent.

For example,

```
format long e
x = pi
x =
     3.141592653589793e+00
```

5. format rat: MATLAB will execute the above statement and return the following result:

The format rat command gives the closest rational expression resulting from a calculation.

For example,

```
format rat
4.678 * 4.9
```

MATLAB will execute the above statement and return the following result:

```
ans =
2063/90
```

4. Commands

MATLAB is an interactive program for numerical computation and data visualization. You can enter a command by typing it at the MATLAB prompt '>>' on the Command Window.

Commands for Managing a Session: MATLAB provides various commands for managing a session. The following table provides all such commands:

Command	Purpose
clc	Clears command window.
clear	Removes variables from memory.
exist	Checks for existence of file or variable.
global	Declares variables to be global.
help	Searches for a help topic.
lookfor	Searches help entries for a keyword.
quit	Stops MATLAB.
who	Lists current variables.
whos	Lists current variables (long display).

Commands for Working with the System: MATLAB provides various useful commands for working with the system, like saving the current work in the workspace as a file and loading the file later. It also provides various commands for other system-related activities like, displaying date, listing files in the directory, displaying current directory, etc. The following table displays some commonly used system-related commands:

Command	Purpose
cd	Changes current directory.
date	Displays current date.
delete	Deletes a file.
diary	Switches on/off diary file recording.
dir	Lists all files in current directory.
load	Loads workspace variables from a file.
path	Displays search path.
pwd	Displays current directory.
save	Saves workspace variables in a file.
type	Displays contents of a file.
what	Lists all MATLAB files in the current directory.
wklread	Reads .wk1 spreadsheet file.

Input and Output Commands: MATLAB provides the following input and output related commands:

Command	Purpose
disp	Displays contents of an array or string.
fscanf	Read formatted data from a file.
format	Controls screen-display format.
fprintf	Performs formatted writes to screen or file.
input	Displays prompts and waits for input.
;	Suppresses screen printing.

They support the following format codes:

Format Code	Purpose
%s	Format as a string.
%d	Format as an integer.
%f	Format as a floating point value.
%e	Format as a floating point value in scientific notation.
%g	Format in the most compact form: %f or %e.
\n	Insert a new line in the output string.
\t	Insert a tab in the output string.

The format function has the following forms used for numeric display:

Format Function	Display up to
format short	Four decimal digits (default).
format long	16 decimal digits.
format short e	Five digits plus exponent.
format long e	16 digits plus exponents.
format bank	Two decimal digits.
format +	Positive, negative, or zero.
format rat	Rational approximation.
format compact	Suppresses some line feeds.
format loose	Resets to less compact display mode.

Vector, Matrix, and Array Commands: The following table shows various commands used for working with arrays, matrices and vectors:

Command	Purpose
cat	Concatenates arrays.
find	Finds indices of nonzero elements.
length	Computes number of elements.
linspace	Creates regularly spaced vector.
logspace	Creates logarithmically spaced vector.
max	Returns largest element.
min	Returns smallest element.
prod	Product of each column.
reshape	Changes size.
size	Computes array size.
sort	Sorts each column.
sum	Sums each column.
eye	Creates an identity matrix.
ones	Creates an array of ones.
zeros	Creates an array of zeros.
cross	Computes matrix cross products.
dot	Computes matrix dot products.

det	Computes determinant of an array.
inv	Computes inverse of a matrix.
pinv	Computes pseudoinverse of a matrix.
rank	Computes rank of a matrix.
rref	Computes reduced row echelon form.
cell	Creates cell array.
celldisp	Displays cell array.
cellplot	Displays graphical representation of cell array.
num2cell	Converts numeric array to cell array.
deal	Matches input and output lists.
iscell	Identifies cell array.

5. Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. MATLAB is designed to operate primarily on whole matrices and arrays. Therefore, operators in MATLAB work both on scalar and non- scalar data. MATLAB allows the following types of elementary operations: Arithmetic Operators, Relational Operators, Logical Operators, Bitwise Operations, Set Operations

Arithmetic Operators: MATLAB allows two different types of arithmetic operations: Matrix arithmetic operations, Array arithmetic operations.

Matrix arithmetic operations are same as defined in linear algebra. Array operations are executed element by element, both on one-dimensional and multidimensional array.

The matrix operators and array operators are differentiated by the period (.) symbol. However, as the addition and subtraction operation is same for matrices and arrays, the operator is same for both cases. The following table gives brief description of the operators:

Operator	Description
+	Addition or unary plus.
-	Subtraction or unary minus.
*	Matrix multiplication.
.*	Array multiplication.
/	Slash or matrix right division.
./	Array right division.
\	Backslash or matrix left division.
.\	Array left division.
^	Matrix power..
.^	Array power.
'	Matrix transpose.
.'	Array transpose.

Example The following examples show the use of arithmetic operators on scalar data. Create a script file with the following code:

```
a = 10;
b = 20;
c = a + b; d = a - b ;e = a * b ;f = a / b ;g = a \ b; x = 7;
y = 3;
z = x ^ y
```

When you run the file, it produces the following result:

```
c =
    3
    0
d =
```

```
   -10
e = 200
f =
    0.5000
g =
     2
z = 343
```

Relational Operators: Relational operators can also work on both scalar and non-scalar data. Relational operators for arrays perform element-by-element comparisons between two arrays and return a logical array of the same size, with elements set to logical 1 (true) where the relation is true and elements set to logical 0 (false) where it is not.
The following table shows the relational operators available in MATLAB:

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

Example: Create a script file and type the following code:

```
a = 100;

b = 200;

if (a >=
b) max = a
else
max = b

end
```

When you run the file, it produces following result:

```
max =
   200
```

Logical Operators: MATLAB offers two types of logical operators and functions:
Element-wise - These operators operate on corresponding elements of logical arrays.
Short-circuit - These operators operate on scalar and logical expressions.
Element-wise logical operators operate element-by-element on logical arrays. The symbols &, |, and ~ are the logical array operators AND, OR, and NOT.
Short-circuit logical operators allow short-circuiting on logical operations. The symbols && and || are the logical short-circuit operators AND and OR.
Example: Create a script file and type the following code:

```
a = 5;
b = 20;
   if ( a && b)
       disp('Line 1 - Condition is true');
   end
   if ( a || b)
      disp('Line 2 - Condition is true');
```

```
end
% lets change the value of a and
b a = 0;
b = 10;
if ( a && b)
   disp('Line 3 - Condition is
true'); else
   disp('Line 3 - Condition is not true');
end
if (~(a && b))
  disp('Line 4 - Condition is
true'); end
```

When you run the file, it produces following result:

```
Line 1 - Condition is true
Line 2 - Condition is true
Line 3 - Condition is not
true
Line 4 - Condition is true
```

Set Operations: MATLAB provides various functions for set operations, like union, intersection and testing for set membership, etc.

The following table shows some commonly used set operations:

Function	Description
intersect(A,B)	Set intersection of two arrays; returns the values common to both A and B. The values returned are in sorted order.
intersect(A,B,'rows')	Treats each row of A and each row of B as single entities and returns the rows common to both A and B. The rows of the returned matrix are in sorted order.
ismember(A,B)	Returns an array the same size as A, containing 1 (true) where the elements of A are found in B. Elsewhere, it returns 0 (false).
ismember(A,B,'rows')	Treats each row of A and each row of B as single entities and returns a vector containing 1 (true) where the rows of matrix A are also rows of B. Elsewhere, it returns 0 (false).
issorted(A)	Returns logical 1 (true) if the elements of A are in sorted order and logical 0 (false) otherwise. Input A can be a vector or an N-by-1 or 1-by-N cell array of strings. A is considered to be sorted if A and the output of sort(A) are equal.
issorted(A, 'rows')	Returns logical 1 (true) if the rows of two-dimensional matrix A are in sorted order, and logical 0 (false) otherwise. Matrix A is considered to be sorted if A and the output of sortrows(A) are equal.
setdiff(A,B)	Sets difference of two arrays; returns the values in A that are not in B. The values in the returned array are in sorted order.
setdiff(A,B,'rows')	Treats each row of A and each row of B as single entities and returns the rows from A that are not in B. The rows of the returned matrix are in sorted order. The 'rows' option does not support cell arrays.
setxor	Sets exclusive OR of two arrays
union	Sets union of two arrays
unique	Unique values in array

Example: Create a script file and type the following code:

```
a = [7 23 14 15 9 12 8 24 35]
b = [ 2 5 7 8 14 16 25 35 27]
u = union(a, b)
i = intersect(a, b)
s = setdiff(a, b)
```

2. VECTORS

A vector is a one-dimensional array of numbers. MATLAB allows creating two types of vectors: Row vectors, Column vectors.

Creating Vectors:

Row Vectors: Row vectors are created by enclosing the set of elements in square brackets, using space or comma to delimit the elements.

```
r = [7 8 9 10 11]
```

Column Vectors: Column vectors are created by enclosing the set of elements in square brackets, using semicolon to delimit the elements.

```
c = [7; 8; 9; 10; 11]
```

Referencing the Elements of a Vector: You can reference one or more of the elements of a vector in several ways. The ithcomponent of a vector v is referred as v(i).

For example:

```
v = [ 1; 2; 3; 4; 5; 6]; % creating a column vector of 6 elements
v(3)
```

When you reference a vector with a colon, such as v(:), all the components of the vector are listed.

```
v = [ 1; 2; 3; 4; 5; 6]; % creating a column vector of 6 elements
v(:)
```

MATLAB allows you to select a range of elements from a vector.

For example, let us create a row vector rv of 9 elements, then we will reference the elements 3 to 7 by writing rv(3:7) and create a new vector named sub_rv.

```
rv = [1 2 3 4 5 6 7 8 9];
sub_rv = rv(3:7)
```

Vector Operations:

Addition and Subtraction of Vectors: You can add or subtract two vectors. Both the operand vectors must be of same type and have same number of elements.

Example: Create a script file with the following code:

```
A = [7, 11, 15, 23, 9];
B = [2, 5, 13, 16, 20];
C = A + B; D = A - B;
disp(C);
disp(D);
```

Scalar Multiplication of Vectors: When you multiply a vector by a number, this is called the scalar multiplication. Scalar multiplication produces a new vector of same type with each element of the original vector multiplied by the number.

Example: Create a script file with the following code:

```
v = [ 12 34 10 8];
m = 5 * v
```

When you run the file, it displays the following result:

```
m =
60      170     50      40
```

Transpose of a Vector: The transpose operation changes a column vector into a row vector and vice versa. The transpose operation is represented by a single quote (').

Example: Create a script file with the following code:

```
r = [ 1 2 3 4 ];
tr = r';
v = [1;2;3;4];
tv = v';
disp(tr); disp(tv);
```

Appending Vectors: MATLAB allows you to append vectors together to create new vectors. If you have two row vectors r1 and r2 with n and m number of elements, to create a row vector r of n plus m elements, by appending these vectors, you write:

```
r = [r1,r2]
```

You can also create a matrix r by appending these two vectors, the vector r2, will be the second row of the matrix:

```
r = [r1;r2]
```

However, to do this, both the vectors should have same number of elements.

Similarly, you can append two column vectors c1 and c2 with n and m number of elements. To create a column vector c of n plus m elements, by appending these vectors, you write:

```
c = [c1; c2]
```

You can also create a matrix c by appending these two vectors; the vector c2 will be the second column of the matrix:

```
c = [c1, c2]
```

However, to do this, both the vectors should have same number of elements.

Example: Create a script file with the following code:

```
r1 = [ 1 2 3 4 ];
r2 = [5 6 7 8 ];
r = [r1,r2]
rMat = [r1;r2]
c1 = [ 1; 2; 3; 4 ];
c2 = [5; 6; 7; 8 ];
c = [c1; c2]
cMat = [c1,c2]
```

Magnitude of a Vector: Magnitude of a vector v with elements v1, v2, v3, …, vn, is given by the equation: $|v| = \sqrt{(v1^2 + v2^2 + v3^2 + \ldots + vn^2)}$

You need to take the following steps to calculate the magnitude of a vector:

Take the product of the vector with itself, using array multiplication (.*). This produces a vector sv, whose elements are squares of the elements of vector v.

```
sv = v.*v;
```

Use the sum function to get the sum of squares of elements of vector v. This is also called the dot product of vector v. dp= sum(sv);

Use the sqrt function to get the square root of the sum which is also the magnitude of the vector v. mag = sqrt(s);

Vector Dot Product: Dot product of two vectors a = (a1, a2, …, an) and b = (b1, b2, …, bn) is given by: $a.b = \sum(ai.bi)$

Dot product of two vectors a and b is calculated using the dot function.

```
dot(a, b);
```

Example: Create a script file with the following code:

```
v1 = [2 3 4];
v2 = [1 2 3];
dp = dot(v1, v2);
disp('Dot Product:'); disp(dp);
```

When you run the file, it displays the following result:

```
Dot Product:
    20
```

Vectors with Uniformly Spaced Elements: MATLAB allows you to create a vector with uniformly spaced elements. To create a vector v with the first element f, last element l, and the difference between elements is any real number n, we write:

```
v = [f : n : l]
```

Example: Create a script file with the following code:

```
v = [1: 2: 20]; sqv = v.^2; disp(v);disp(sqv);
```

3. Matrix

A matrix is a two-dimensional array of numbers. In MATLAB, you create a matrix by entering elements in each row as comma or space delimited numbers and using semicolons to mark the end of each row.
For example, let us create a 4-by-5 matrix a:

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

MATLAB will execute the above statement and return the following result:

```
a =
     1     2     3     4     5
     2     3     4     5     6
     3     4     5     6     7
     4     5     6     7     8
```

Referencing the Elements of a Matrix: To reference an element in the mth row and nth column, of a matrix mx, we write: mx(m, n);
For example, to refer to the element in the 2nd row and 5th column, of the matrix a, as created in the last section, we type:

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
a(2,5)
```

MATLAB will execute the above statement and return the following result:

```
ans =

     6
```

To reference all the elements in the mth column we type A(:,m).
Let us create a column vector v, from the elements of the 4th row of the matrix a:

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

v = a(:,4)
```

You can also select the elements in the mth through nth columns, for this we write: a(:,m:n)
Let us create a smaller matrix taking the elements from the second and third columns:

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
a(:, 2:3)
```

MATLAB will execute the above statement and return the following result:

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

In the same way, you can create a sub-matrix taking a sub-part of a matrix.

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
a(:, 2:3)
```

In the same way, you can create a sub-matrix taking a sub-part of a matrix. For example, let us create a sub-matrix sa taking the inner subpart of a:

```
3   4   5
4   5   6
```

To do this, write:

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
sa = a(2:3,2:4)
```

Deleting a Row or a Column in a Matrix: You can delete an entire row or column of a matrix by assigning an empty set of square braces [] to that row or column. Basically, [] denotes an empty array.

For example, let us delete the fourth row of a:

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
a( 4 , : ) = []
```

MATLAB will execute the above statement and return the following result:

```
a =
    1   2   3   4   5
    2   3   4   5   6
    3   4   5   6   7
```

Next, let us delete the fifth column of a:

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];
a(: , 5)=[]
```

MATLAB will execute the above statement and return the following result:

```
a =
    1   2   3   4
    2   3   4   5
    3   4   5   6
    4   5   6   7
```

Example: In this example, let us create a 3-by-3 matrix m, then we will copy the second and third rows of this matrix twice to create a 4-by-3 matrix.

Create a script file with the following code:

```
a = [ 1 2 3 ; 4 5 6; 7 8 9];
new_mat = a([2,3,2,3],:)
```

Matrix Operations:

Addition and Subtraction of Matrices:

You can add or subtract matrices. Both the operand matrices must have the same number of rows and columns.

Example: Create a script file with the following code:

```
a = [ 1 2 3 ; 4 5 6; 7 8 9];
b = [ 7 5 6 ; 2 0 8; 5 7 1];
```

```
c = a + b; d = a - b
```

Division (Left, Right) of Matrix: You can divide two matrices using left (\) or right (/) division operators. Both the operand matrices must have the same number of rows and columns.

Example: Create a script file with the following code:

```
a = [ 1 2 3 ; 4 5 6; 7 8 9];
b = [ 7 5 6 ; 2 0 8; 5 7 1];
c = a /
b d = a
\ b
```

Scalar Operations of Matrices : When you add, subtract, multiply or divide a matrix by a number, this is called the scalar operation.

Scalar operations produce a new matrix with same number of rows and columns with each element of the original matrix added to, subtracted from, multiplied by or divided by the number.

Example: Create a script file with the following code:

a = [10 12 23 ; 14 8 6; 27 8 9];
b = 2;
c = a + b ;d = a – b; e = a * b;f = a / b

Transpose of a Matrix: The transpose operation switches the rows and columns in a matrix. It is represented by a single quote(').

Example: Create a script file with the following code:

a = [10 12 23 ; 14 8 6; 27 8 9]
b = a'

Concatenating Matrices: You can concatenate two matrices to create a larger matrix. The pair of square brackets '[]' is the concatenation operator. MATLAB allows two types of concatenations: Horizontal concatenation, Vertical concatenation.

When you concatenate two matrices by separating those using commas, they are just appended horizontally. It is called horizontal concatenation.

Alternatively, if you concatenate two matrices by separating those using semicolons, they are appended vertically. It is called vertical concatenation.

Example: Create a script file with the following code:

```
a = [ 10 12 23 ; 14 8 6; 27 8 9]
b = [ 12 31 45 ; 8 0 -9; 45 2 11]
c = [a, b]
d = [a; b]
```

Matrix Multiplication: Consider two matrices A and B. If A is an m x n matrix and B is an n x p matrix, they could be multiplied together to produce an m x n matrix C. Matrix multiplication is possible only if the number of columns n in A is equal to the number of rows n in B. In matrix multiplication, the elements of the rows in the first matrix are multiplied with corresponding columns in the second matrix.

Each element in the (i, j)th position, in the resulting matrix C, is the summation of the products of elements in ith row of first matrix with the corresponding element in the jth column of the second matrix. Matrix multiplication in MATLAB is performed by using the * operator.

Example: Create a script file with the following code:

```
a = [ 1 2 3; 2 3 4; 1 2 5]
b = [ 2 1 3 ; 5 0 -2; 2 3 -1]
prod = a * b
```

Determinant of a Matrix: Determinant of a matrix is calculated using the det function of MATLAB. Determinant of a matrix A is given by det(A).
Example: Create a script file with the following code:

```
a = [ 1 2 3; 2 3 4; 1 2 5]
det(a)
```

Inverse of a Matrix: The inverse of a matrix A is denoted by A^{-1} such that the following relationship holds: $AA^{-1} = A^{-1A} = I$
The inverse of a matrix does not always exist. If the determinant of the matrix is zero, then the inverse does not exist and the matrix is singular. Inverse of a matrix in MATLAB is calculated using the inv function. Inverse of a matrix A is given by inv(A).
Example: Create a script file and type the following code:

```
a = [ 1 2 3; 2 3 4; 1 2 5]
inv(a)
```

Arrays: All variables of all data types in MATLAB are multidimensional arrays. A vector is a one-dimensional array and a matrix is a two-dimensional array.
We have already discussed vectors and matrices. In this chapter, we will discuss multidimensional arrays. However, before that, let us discuss some special types of arrays.
Special Arrays in MATLAB:
1. **The zeros() function** creates an array of all zeros:

For example: `zeros(5)`
MATLAB will execute the above statement and return the following result:

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

2. **The ones() function** creates an array of all ones:

For example: `ones(4,3)`
MATLAB will execute the above statement and return the following result:

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

3. **The eye() function** creates an identity matrix.
For example:

```
eye(4)
```

MATLAB will execute the above statement and return the following result:

```
1   0   0   0
0   1   0   0
0   0   1   0
0   0   0   1
```

4.**The rand() function** creates an array of uniformly distributed random numbers on (0,1):

For example: `rand(3, 5)`

MATLAB will execute the above statement and return the following result:

```
0.8147    0.9134    0.2785    0.9649    0.9572
0.9058    0.6324    0.5469    0.1576    0.4854
0.1270    0.0975    0.9575    0.9706    0.8003
```

5. **A Magic Square:** A magic square is a square that produces the same sum, when its elements are added row-wise, column-wise or diagonally.

The magic() function creates a magic square array. It takes a singular argument that gives the size of the square. The argument must be a scalar greater than or equal to 3.

```
magic(4)
```

MATLAB will execute the above statement and return the following result:

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

You can use the colon operator to create a vector of indices to select rows, columns or elements of arrays.

The following table describes its use for this purpose (let us have a matrix A):

Format	Purpose
A(:,j)	is the jth column of A.
A(i,:)	is the ith row of A.
A(:,:)	is the equivalent two-dimensional array. For matrices this is the same as A.
A(j:k)	is A(j), A(j+1),...,A(k).
A(:,j:k)	is A(:,j), A(:,j+1),...,A(:,k).
A(:,:,k)	is the kth page of three-dimensional array A.
A(i,j,k,:)	is a vector in four-dimensional array A. The vector includes A(i,j,k,1), A(i,j,k,2), A(i,j,k,3), and so on.
A(:)	is all the elements of A, regarded as a single column. On the left side of an assignment statement, A(:) fills A, preserving its shape from before. In this case, the right side must contain the same number of elements as A.

4. Symbolic Algebra

Solving Basic Algebraic Equations in MATLAB: The solve function is used for solving algebraic equations. In its simplest form, the solve function takes the equation enclosed in quotes as an argument.

For example, let us solve for x in the equation x-5 = 0

syms x

```
solve(x-5==0) or solve(x-5)
```

MATLAB will execute the above statement and return the following result:

```
ans =

  5
```

If the equation involves multiple symbols, then MATLAB by default assumes that you are solving for x, however, the solve command has another form:

```
solve(equation, variable)
```

For example, let us solve the equation $v - u - 3t^2 = 0$, for v. In this case, we should write:

```
syms v u t
solve(v-u-3*t^2)
ans =
 3*t^2 + u
```

Solving Basic Algebraic Equations: The roots command is used for solving algebraic equations in Octave and you can write above examples as follows:

For example, let us solve for x in the equation x-5 = 0

```
roots([1, -5])
```

Solving Quadratic Equations in MATLAB

The solve function can also solve higher order equations. It is often used to solve quadratic equations. The function returns the roots of the equation in an array.

The following example solves the quadratic equation $x^2 - 7x + 12 = 0$. Create a script file and type the following code:

syms x

```
eq = x^2 -7*x + 12 = 0; s = solve(eq);
disp('The first root is: '), disp(s(1));
disp('The second root is: '), disp(s(2));
```

When you run the file, it displays the following result:

```
The first root is: 3
The second root is: 4
```

Solving Quadratic Equations in Octave

The following example solves the quadratic equation $x^2 - 7x + 12 = 0$ in Octave. Create a script file and type the following code:

```
s = roots([1, -7, 12]);
disp('The first root is: '), disp(s(1));
disp('The second root is: '), disp(s(2));
```

When you run the file, it displays the following result:

```
The first root is: 4
The second root is: 3
```

Solving Higher Order Equations in MATLAB: The solve command can also solve higher order equations.
For example, let us solve a cubic equation as $(x-3)^2(x-7) = 0$

```
syms x; solve((x-3)^2*(x-7)==0)
```

MATLAB will execute the above statement and return the following result:

```
ans

= 3

 3

 7
```

In case of higher order equations, roots are long containing many terms. You can get the numerical value of such roots by converting them to double.
The following example solves the fourth order equation $x^4 - 7x^3 + 3x^2 - 5x + 9 = 0$.
Create a script file and type the following code:

```
eq = x^4 - 7*x^3 + 3*x^2 - 5*x + 9 ; s = solve(eq);
disp('The first root is: '), disp(s(1));
disp('The second root is: '), disp(s(2));
disp('The third root is: '), disp(s(3));
disp('The fourth root is: '), disp(s(4));
% converting the roots to double type
disp('Numeric value of first root'), disp(double(s(1)));
disp('Numeric value of second root'), disp(double(s(2)));
disp('Numeric value of third root'), disp(double(s(3)));
disp('Numeric value of fourth root'), disp(double(s(4)));
```

Solving Higher Order Equations :
The following example solves the fourth order equation $x^4 - 7x^3 + 3x^2 - 5x + 9 = 0$.
Create a script file and type the following code:

```
v = [1, -7,  3, -5, 9];
s = roots(v);
% converting the roots to double type
disp('Numeric value of first root'), disp(double(s(1)));
disp('Numeric value of second root'),
disp(double(s(2))); disp('Numeric value of third root'),
disp(double(s(3)));
disp('Numeric value of fourth root'), disp(double(s(4)));
```

When you run the file, it returns the following result:

```
Numeric value of first root 6.6304
Numeric value of second root
-0.34509 + 1.07784i
Numeric value of third root
-0.34509 - 1.07784i
Numeric value of fourth root 1.0598
```

Solving System of Equations in MATLAB: The solve function can also be used to generate solutions of systems of equations involving more than one variables. Let us take up a simple example to demonstrate this use.

1. Let us solve the equations: $5x + 9y = 5$, $3x – 6y = 4$

Create a script file and type the following code:

```
s = solve(5*x + 9*y == 5,3*x - 6*y == 4);
```

```
s.x
s.y
```

When you run the file, it displays the following result:

ans = 22/19

ans = -5/57

2. In same way, you can solve larger linear systems. Consider the following set of equations:
   x + 3y -2z = 5 3x + 5y + 6z = 7 2x + 4y + 3z = 8

Solving System of Equations : We have a little different approach to solve a system of 'n' linear equations in 'n' unknowns. Let us take up a simple example to demonstrate this use.

Let us solve the equations: 5x + 9y = 5, 3x – 6y = 4

Such a system of linear equations can be written as the single matrix equation Ax = b, where A is the coefficient matrix, b is the column vector containing the right- hand side of the linear equations and x is the column vector representing the solution as shown in the below program:

Create a script file and type the following code:

```
A = [5, 9; 3, -6];
b = [5;4];
A \ b
```

When you run the file, it displays the following result:

```
ans =
   1.157895

  -0.087719
```

2. In same way, you can solve larger linear systems as given below: x + 3y -2z = 5
3x + 5y + 6z = 7 2x + 4y + 3z = 8

Expanding and Collecting Equations in MATLAB: The expand and the collect commands expands and collects an equation respectively. The following example demonstrates the concepts: When you work with many symbolic functions, you should declare that your variables are symbolic.

Create a script file and type the following code:

```
syms x %symbolic variable x syms y %symbolic variable x
% expanding equations expand((x-5)*(x+9))
expand((x+2)*(x-3)*(x-5)*(x+7))
expand(sin(2*x)) expand(cos(x+y))

% collecting equations
collect(x^3 *(x-7))
collect(x^4*(x-3)*(x-5))
```

When you run the file, it displays the following result:

ans =

x^2 + 4*x - 45

ans =

x^4 + x^3 - 43*x^2 + 23*x + 210

ans =

2*cos(x)*sin(x)

ans =

cos(x)*cos(y) - sin(x)*sin(y)

ans =

x^4 - 7*x^3

ans =

x^6 - 8*x^5 + 15*x^4

Create a script file and type the following code:

```
 % define symbolic variables
 x = sym ('x');
 y = sym ('y');
 z = sym ('z');
 % expanding equations
 expand((x-5)*(x+9))
 expand((x+2)*(x-3)*(x-5)*(x+7))
 expand(Sin(2*x))
 expand(Cos(x+y))
% collecting equations
collect(x^3 *(x-7))
collect(x^4*(x-3)*(x-5))
```

Factorization and Simplification of Algebraic Expressions: The factor function factorizes an expression and the simplify function simplifies an expression. The following example demonstrates the concept:

Example: Create a script file and type the following code:

```
syms x y
factor(x^3 - y^3);factor(x^2-y^2)
simplify((x^4-16)/(x^2-4))
```

5. M-Files

So far, we have used MATLAB environment as a calculator. However, MATLAB is also a powerful programming language, as well as an interactive computational environment.

The M Files: MATLAB allows writing two kinds of program files:

1. Scripts - script files are program files with .m extension. In these files, you write series of commands, which you want to execute together. Scripts do not accept inputs and do not return any outputs. They operate on data in the workspace.

2. Functions - functions files are also program files with .m extension. Functions can accept inputs and return outputs. Internal variables are local to the function.

You can use the MATLAB editor or any other text editor to create your .m files. A script file contains multiple sequential lines of MATLAB commands and function calls. You can run a script by typing its name at the command line.

1. Scripts

Creating and Running Script File:

To create scripts files, you need to use a text editor. You can open the MATLAB editor in two ways:

- Using the command prompt Using the IDE
- If you are using the command prompt, type edit in the command prompt. This will open the editor. You can directly type edit and then the filename (with .m extension)

```
edit
Or
edit <filename>
```

The above command will create the file in default MATLAB directory. If you want to store all program files in a specific folder, then you will have to provide the entire path. Let us create a folder named progs. Type the following commands at the command prompt(>>):

```
mkdir progs    % create directory progs under default
directory chdir progs       % changing the current directory to
progs
edit prog1.m % creating an m file named prog1.m
```

Alternatively, if you are using the IDE, choose NEW -> Script. This also opens the editor and creates a file named Untitled. You can name and save the file after typing the code. Type the following code in the editor:

```
NoOfStudents = 6000;
TeachingStaff = 150;
NonTeachingStaff = 20;
Total = NoOfStudents + TeachingStaff + NonTeachingStaff;
disp(Total);
```

After creating and saving the file, you can run it in two ways:

- Clicking the Run button on the editor window or
- Just typing the filename (without extension) in the command prompt: >> prog1.

The command window prompt displays the result:

```
6170
```

Example: Create a script file, and type the following code:

```
a = 5; b = 7;
c = a + b
d = c +
sin(b) e = 5
* d
f = exp(-d)
```

When the above code is compiled and executed, it produces the following result:

```
c =
    1
    2
```

```
d =
   12.657
   0
e =
   63.284
   9
f =
   3.1852e-06
```

3. Functions

A function is a group of statements that together perform a task. In MATLAB, functions are defined in separate files. The name of the file and of the function should be the same. Functions operate on variables within their own workspace, which is also called the local workspace, separate from the workspace you access at the MATLAB command prompt which is called the base workspace. Functions can accept more than one input arguments and may return more than one output arguments

Syntax of a function statement is:

```
function [out1,out2, ..., outN] = myfun(in1,in2,in3, ..., inN)
```

Example: The following function named mymax should be written in a file named mymax.m. It takes five numbers as argument and returns the maximum of the numbers.

Create a function file, named mymax.m and type the following code in it:

Example:

```
function max = mymax(n1, n2, n3, n4, n5)

%This function calculates the maximum of the
% five numbers given as
input max =  n1;
if(n2 > max)
   max = n2;
end
if(n3 > max)
   max = n3;
end
if(n4 > max)
   max = n4;
end
if(n5 > max)
    max = n5;
end
```

The first line of a function starts with the keyword function. It gives the name of the function and order of arguments. In our example, the mymax function has five input arguments and one output argument. The comment lines that come right after the function statement provide the help text. These lines are printed when you type:

```
help mymax
```

MATLAB will execute the above statement and return the following result:

This function calculates the maximum of the five numbers given as input

You can call the function as:

```
mymax(34, 78, 89, 23, 11)
```

MATLAB will execute the above statement and return the following result:

```
ans = 89
```

Anonymous Functions: An anonymous function is like an inline function in traditional programming languages, defined within a single MATLAB statement. It consists of a single MATLAB expression and any number of input and output arguments. You can define an anonymous function right at the MATLAB command line or within a function or script. This way you can create simple functions without having to create a file for them. The syntax for creating an anonymous function from an expression is

```
f = @(arglist)expression
```

Example : In this example, we will write an anonymous function named power, which will take two numbers as input and return first number raised to the power of the second number.
Create a script file and type the following code in it:

```
power = @(x, n) x.^n;
result1 = power(7, 3)
result2 = power(49, 0.5)
result3 = power(10, -10)
result4 = power (4.5, 1.5)
```

When you run the file, it displays:

```
result1 = 343
result2 = 7
result3 = 1.0000e-10
result4 = 9.5459
```

Primary and Sub-Functions: Any function other than an anonymous function must be defined within a file. Each function file contains a required primary function that appears first and any number of optional sub-functions that comes after the primary function and used by it. Primary functions can be called from outside of the file that defines them, either from command line or from other functions, but sub-functions cannot be called from command line or other functions, outside the function file. Sub-functions are visible only to the primary function and other sub-functions within the function file that defines them.
Example: Let us write a function named quadratic that would calculate the roots of a quadratic equation. The function would take three inputs, the quadratic co- efficient, the linear co-efficient and the constant term. It would return the roots. The function file quadratic.m will contain the primary function quadratic and the sub-function disc, which calculates the discriminant. Create a function file quadratic.m and type the following code in it:

```
function [x1,x2] = quadratic(a,b,c)
%this function returns the roots of
% a quadratic equation.
% It takes 3 input arguments
% which are the co-efficients of x2, x and the
%constant term
% It returns the roots d =
disc(a,b,c);
x1 = (-b + d) / (2*a);
x2 = (-b - d) /(2*a);
end % end of quadratic
function dis = disc(a,b,c)
%function calculates the discriminant
dis = sqrt(b^2 - 4*a*c);
end % end of sub-function
```

You can call the above function from command prompt as:

```
quadratic(2,4,-4)
```

MATLAB will execute the above statement and will give the following result:

```
ans = 0.7321
```

Nested Functions: You can define functions within the body of another function. These are called nested functions. A nested function contains any or all of the components of any other function. Nested functions are defined within the scope of another function and they share access to the containing function's workspace.
A nested function follows the below syntax:

```
function x = A(p1, p2)
...
B(p2)
   function y = B(p3)
   ...
   end
...
end
```

Example: Let us rewrite the function quadratic, from previous example, however, this time the disc function will be a nested function.
Create a function file quadratic2.m and type the following code in it:

```
function [x1,x2] =
quadratic2(a,b,c)

function disc % nested function
d = sqrt(b^2 - 4*a*c);
end % end of function disc
disc;
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);
end % end of function quadratic2
```

You can call the above function from command prompt as:

```
quadratic2(2,4,-4)
```

MATLAB will execute the above statement and return the following result:

```
ans = 0.7321
```

Private Functions: A private function is a primary function that is visible only to a limited group of other functions. If you do not want to expose the implementation of a function(s), you can create them as private functions. Private functions reside in subfolders with the special name private. They are visible only to functions in the parent folder.
Example: Let us rewrite the quadratic function. This time, however, the disc function calculating the discriminant, will be a private function. Create a subfolder named private in working directory. Store the following function file disc.m in it:

```
function dis = disc(a,b,c)
%function calculates the discriminant
dis = sqrt(b^2 - 4*a*c);
end % end of sub-function
```

Create a function quadratic3.m in your working directory and type the following code in it:

```
function [x1,x2] = quadratic3(a,b,c)
%this function returns the roots of
% a quadratic equation.
% It takes 3 input arguments
% which are the co-efficients of x2, x and the
%constant term
% It returns the roots d = disc(a,b,c);
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a); end % end of quadratic3
```

You can call the above function from command prompt as:

```
quadratic3(2,4,-4)
```

MATLAB will execute the above statement and return the following result:

```
ans = 0.7321
```

Global Variables: Global variables can be shared by more than one function. For this, you need to declare the variable as global in all the functions. If you want to access that variable from the base workspace, then declare the variable at the command line. The global declaration must occur before the variable is actually used in a function. It is a good practice to use capital letters for the names of global variables to distinguish them from other variables.

Example: Let us create a function file named average.m and type the following code in it:

```
function avg =

average(nums) global TOTAL

avg = sum(nums)/TOTAL;

end
```

Create a script file and type the following code in it:

```
global

TOTAL; TOTAL

= 10;

n = [34, 45, 25, 45, 33, 19, 40, 34, 38, 42];

av = average(n)
```

When you run the file, it will display the following result:

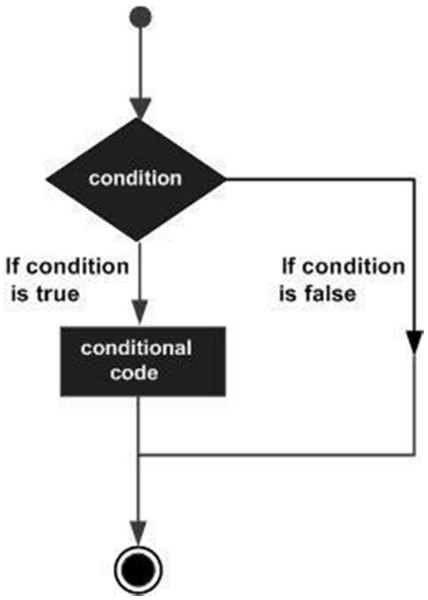
```
av =

   35.5000
```

6. Decision Making

Decision making structures require that the programmer should specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages:



MATLAB provides following types of decision making statements. Click the following links to check their detail:

Statement	Description
if ... end statement	An if ... end statement consists of a boolean expression followed by one or more statements.
if...else...end statement	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
If... elseif...elseif...else...end statements	An if statement can be followed by one (or more) optional elseif... and an else statement, which is very useful to test various conditions.
nested if statements	You can use one if or elseif statement inside another if or elseif statement(s).
switch statement	A switch statement allows a variable to be tested for equality against a list of values.
nested switch statements	You can use one switch statement inside another switch statement(s).

if... end Statement

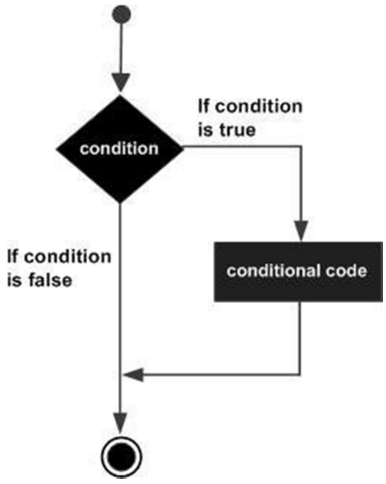
An if ... end statement consists of an if statement and a boolean expression followed by one or more statements. It is delimited by the end statement.

Syntax: The syntax of an if statement in MATLAB is:

```
if <expression>
% statement(s) will execute if the boolean expression is true
<statements
> end
```

If the expression evaluates to true, then the block of code inside the if statement will be executed. If the expression evaluates to false, then the first set of code after the end statement will be executed.

Flow Diagram



Example: Create a script file and type the following code:

```
a = 10;
% check the condition using if statement
   if a < 20
   % if condition is true then print the
       following fprintf('a is less than 20\n'
       );
   end
fprintf('value of a is : %d\n', a);
```

When you run the file, it displays the following result:

```
a is less than 20
value of a is : 10
```

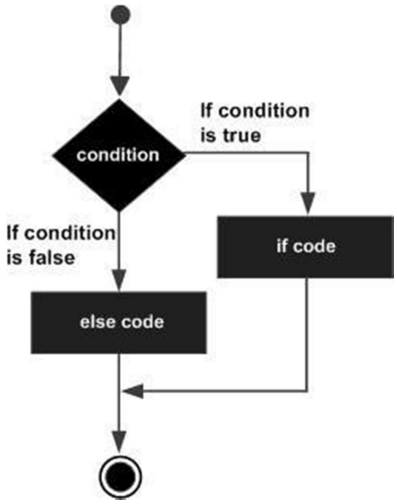
if...else...end Statement: An if statement can be followed by an optional else statement, which executes when the expression is false.

Syntax: The syntax of an if...else statement in MATLAB is:

```
if <expression>
% statement(s) will execute if the boolean expression is true
<statement(s)
> else
<statement(s)>
% statement(s) will execute if the boolean expression is false
end
```

If the boolean expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.

Flow Diagram



Example: Create a script file and type the following code:

```
a = 100;
% check the boolean condition
   if a < 20
       % if condition is true then print the following
      fprintf('a is less than 20\n' );
   else

      % if condition is false then print the following
      fprintf('a is not less than 20\n' );
   end

   fprintf('value of a is : %d\n', a);
```

When the above code is compiled and executed, it produces the following result:

```
a is not less than 20
value of a is : 100
```

if...elseif...elseif...else...end Statements: An if statement can be followed by one (or more) optional elseif... and an else statement, which is very useful to test various conditions. When using if... elseif...else statements, there are few points to keep in mind: An if can have zero or one else's and it must come after any elseif's. An if can have zero to many elseif's and they must come before the else. Once an else if succeeds, none of the remaining elseif's or else's will be tested.

Syntax:

```
if <expression 1>
% Executes when the expression 1 is true
<statement(s)>
elseif <expression 2>
% Executes when the boolean expression 2 is true
<statement(s)>
elseif <expression
3>
% Executes when the boolean expression 3 is true
<statement(s)
> else
% executes when the none of the above condition is true
<statement(s)
> end
```

Example: Create a script file and type the following code in it:

```
a = 100;
%check the boolean condition
   if a == 10
        % if condition is true then print the following
      fprintf('Value of a is 10\n' );
    elseif( a == 20 )
      % if else if condition is true
      fprintf('Value of a is 20\n' );
    elseif a == 30
       % if else if condition is true
      fprintf('Value of a is 30\n'
   ); else
       % if none of the conditions is true '
       fprintf('None of the values are matching\n');
    fprintf('Exact value of a is: %d\n', a ); end
```

When the above code is compiled and executed, it produces the following result:

```
None of the values are matching
Exact value of a is: 100
```

The Nested if Statements: It is always legal in MATLAB to nest if-else statements which means you can use one if or elseif statement inside another if or elseif statement(s).

Syntax: The syntax for a nested if statement is as follows:

```
if <expression 1>
% Executes when the boolean expression 1 is
   true if <expression 2>
      % Executes when the boolean expression 2 is true
  end
  end
```

You can nest elseif...else in the similar way as you have nested if statement.

Example: Create a script file and type the following code in it:

```
a = 100;
b = 200;
    % check the boolean condition
   if( a == 100 )
      % if condition is true then check the following
      if( b == 200 )
        % if condition is true then print the following
        fprintf('Value of a is 100 and b is 200\n' );
      end
   end
   fprintf('Exact value of a is : %d\n', a );

   fprintf('Exact value of b is : %d\n', b );
```

When you run the file, it displays:

```
Value of a is 100 and b is 200
Exact value of a is :
100 Exact value of b is
: 200
```

The switch Statement: A switch block conditionally executes one set of statements from several choices. Each choice is covered by a case statement.

An evaluated switch_expression is a scalar or string.

An evaluated case_expression is a scalar, a string or a cell array of scalars or strings.

The switch block tests each case until one of the cases is true. A case is true when: For numbers, eq(case_expression,switch_expression).
For strings, strcmp(case_expression,switch_expression).
For objects that support the eq function,eq(case_expression,switch_expression).
For a cell array case_expression, at least one of the elements of the cell array matches switch_expression, as defined above for numbers, strings and objects.
When a case is true, MATLAB executes the corresponding statements and then exits the switch block.
The otherwise block is optional and executes only when no case is true.
Syntax: The syntax of switch statement in MATLAB is:

```
switch
   <switch_expression>
   case <case_expression>
     <statements>

   case <case_expression>

     <statements>
     ...
     ...
   otherwise
      <statements>
end
```

Example: Create a script file and type the following code in it:

```
grade = 'B';
   switch(grade
   ) case 'A'
      fprintf('Excellent!\n'
   ); case 'B'
       fprintf('Well done\n'
   ); case 'C'
      fprintf('Well done\n'
   ); case 'D'
      fprintf('You passed\n' );
   case 'F'
     fprintf('Better try again\n' );
   otherwise
     fprintf('Invalid grade\n' );
   end
```

When you run the file, it displays:

```
Well done
Your grade is  B
```

The Nested Switch Statements: It is possible to have a switch as part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.
Syntax: The syntax for a nested switch statement is as follows:

```
switch(ch1)
case 'A'
   fprintf('This A is part of outer switch');
      switch(ch2)
         case 'A'

          fprintf('This A is part of inner switch' );
         case 'B'
         fprintf('This B is part of inner switch'
```

```
      ); end
 case 'B'
 fprintf('This B is part of outer switch' );
 end
```

Example: Create a script file and type the following code in it:

```
 a = 100;
 b = 200;
 switch(a)
      case 100
        fprintf('This is part of outer switch %d\n', a
        ); switch(b)
           case 200
             fprintf('This is part of inner switch %d\n', a );
        end
 end
 fprintf('Exact value of a is : %d\n', a );
fprintf('Exact value of b is : %d\n', b );
```

When you run the file, it displays:

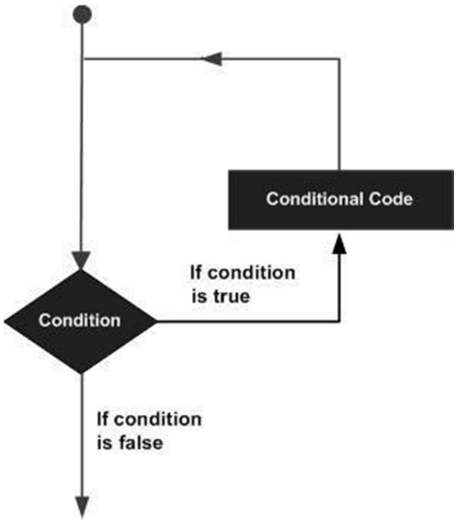
```
 This is part of outer switch
 100 This is part of inner
 switch 100 Exact value of a is
 : 100
 Exact value of b is : 200
```

7. Loop Types

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:



MATLAB provides following types of loops to handle looping requirements. Click the following links to check their detail:

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loops inside any another loop.

The while Loop: The while loop repeatedly executes statements while condition is true.

Syntax: The syntax of a while loop in MATLAB is:

```
while <expression>
   <statements
> end
```

The while loop repeatedly executes program statement(s) as long as the expression remains true. An expression is true when the result is nonempty and contains all nonzero elements (logical or real numeric). Otherwise, the expression is false.

Example: Create a script file and type the following code:

```
a = 10;
% while loop execution
while( a < 20 )
  fprintf('value of a: %d\n',
  a); a = a + 1;
end
```

When you run the file, it displays the following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
```

```
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

The for Loop: A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax: The syntax of a for loop in MATLAB is:

```
for index = values
  <program statements>

        ...
end
```

values has one of the following forms:

Format	Description
initval:endval	increments the index variable from initval to endval by 1, and repeats execution of program statements until index is greater than endval.
initval:step:endval	increments index by the value step on each iteration, or decrements when step is negative.
valArray	creates a column vector index from subsequent columns of arrayvalArray on each iteration. For example, on the first iteration, index = valArray(:,1). The loop executes for a maximum of n times, where n is the number of columns of valArray, given by numel(valArray, 1, :). The input valArray can be of any MATLAB data type, including a string, cell array, or struct.

Example 1: Create a script file and type the following code:

```
for a = 10:20
  fprintf('value of a: %d\n', a); end
```

When you run the file, it displays the following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 2
```

Example 2: Create a script file and type the following code:

```
for a = 1.0: -0.1:0.0
   disp(a) end
```

Example 3: Create a script file and type the following code:

```
for a = [24,18,17,23,28]
disp(a) end
```

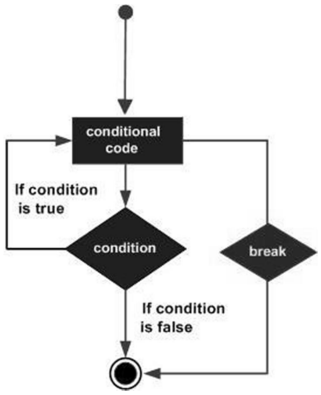
The Nested Loops: MATLAB allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax: The syntax for a nested for loop statement in MATLAB is as follows:
for m = 1:j
for n = 1:k
<statements>;
End
end
The syntax for a nested while loop statement in MATLAB is as follows:
while <expression1>
while <expression2>
<statements>
end
end
Example: Let us use a nested for loop to display all the prime numbers from 1 to 100. Create a script file and type the following code:
for i=2:100
for j=2:100 if(~mod(i,j))
break; % if factor found, not prime
end end
if(j > (i/j))
fprintf('%d is prime\n', i);
end end

Loop Control Statements: Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. MATLAB supports the following control statements. Click the following links to check their detail.

Control Statement	Description
break statement	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

The break Statement: The break statement terminates execution of for or while loop. Statements in the loop that appear after the break statement are not executed.
In nested loops, break exits only from the loop in which it occurs. Control passes to the statement following the end of that loop.
Flow Diagram



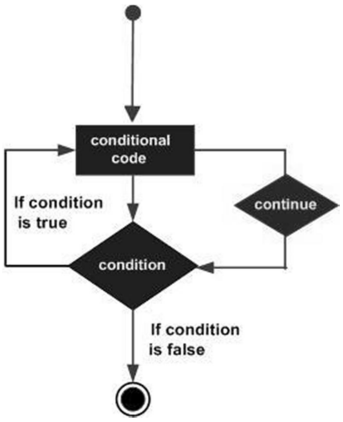
Example: Create a script file and type the following code:

```
a = 10;
% while loop execution
 while (a < 20 )
     fprintf('value of a: %d\n',
     a); a = a+1;
```

```
      if( a > 15)
         % terminate the loop using break statement
          break;
      end
   end
```

The continue Statement: The continue statement is used for passing control to next iteration of for or while loop. The continue statement in MATLAB works somewhat like the break statement. Instead of forcing termination, however, 'continue' forces the next iteration of the loop to take place, skipping any code in between.

Flow Diagram:



Example: Create a script file and type the following code:

```
a = 10;

%while loop execution

while a < 20 if a == 15
     % skip the iteration a = a + 1; continue;
  end

  fprintf('value of a: %d\n', a); a = a + 1;
end
```