5. 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**.

In this section, we will provide lists of commonly used general MATLAB commands.

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.

The **fscanf** and **fprintf** commands behave like C scanf and printf functions. 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.

Plotting Commands

MATLAB provides numerous commands for plotting graphs. The following table shows some of the commonly used commands for plotting:

Command	Purpose
axis	Sets axis limits.
fplot	Intelligent plotting of functions.
grid	Displays gridlines.
plot	Generates xy plot.
print	Prints plot or saves plot to a file.
title	Puts text at top of plot.
xlabel	Adds text label to x-axis.
ylabel	Adds text label to y-axis.
axes	Creates axes objects.

close	Closes the current plot.
close all	Closes all plots.
figure	Opens a new figure window.
gtext	Enables label placement by mouse.
hold	Freezes current plot.
legend	Legend placement by mouse.
refresh	Redraws current figure window.
set	Specifies properties of objects such as axes.
subplot	Creates plots in sub windows.
text	Places string in figure.
bar	Creates bar chart.
loglog	Creates log-log plot.
polar	Creates polar plot.
semilogx	Creates semi log plot. (logarithmic abscissa).
semilogy	Creates semi log plot. (logarithmic ordinate).
stairs	Creates stairs plot.
stem	Creates stem plot.

7. DATA TYPES

MATLAB does not require any type declaration or dimension statements. Whenever MATLAB encounters a new variable name, it creates the variable and allocates appropriate memory space.

If the variable already exists, then MATLAB replaces the original content with new content and allocates new storage space, where necessary.

For example,

```
Total = 42
```

The above statement creates a 1-by-1 matrix named 'Total' and stores the value 42 in it.

Data Types Available in MATLAB

MATLAB provides 15 fundamental data types. Every data type stores data that is in the form of a matrix or array. The size of this matrix or array is a minimum of 0-by-0 and this can grow up to a matrix or array of any size.

The following table shows the most commonly used data types in MATLAB:

Data Type	Description
int8	8-bit signed integer
uint8	8-bit unsigned integer
int16	16-bit signed integer
uint16	16-bit unsigned integer
int32	32-bit signed integer
uint32	32-bit unsigned integer
int64	64-bit signed integer

uint64	64-bit unsigned integer
single	single precision numerical data
double	double precision numerical data
logical	logical values of 1 or 0, represent true and false respectively
char	character data (strings are stored as vector of characters)
cell array	array of indexed cells, each capable of storing an array of a different dimension and data type
structure	C-like structures, each structure having named fields capable of storing an array of a different dimension and data type
function handle	pointer to a function
user classes	objects constructed from a user-defined class
java classes	objects constructed from a Java class

Example

Create a script file with the following code:

```
str = 'Hello World!'

n = 2345

d = double(n)

un = uint32(789.50)

rn = 5678.92347

c = int32(rn)
```

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

```
str =

Hello World!

n =

   2345

d =

   2345

un =

   790

rn =

   5.6789e+03

c =

   5679
```

Data Type Conversion

MATLAB provides various functions for converting a value from one data type to another. The following table shows the data type conversion functions:

Function	Purpose
Char	Convert to character array (string)
int2str	Convert integer data to string
mat2str	Convert matrix to string
num2str	Convert number to string
str2double	Convert string to double-precision value

str2num	Convert string to number
native2unicode	Convert numeric bytes to Unicode characters
unicode2native	Convert Unicode characters to numeric bytes
base2dec	Convert base N number string to decimal number
bin2dec	Convert binary number string to decimal number
dec2base	Convert decimal to base N number in string
dec2bin	Convert decimal to binary number in string
dec2hex	Convert decimal to hexadecimal number in string
hex2dec	Convert hexadecimal number string to decimal number
hex2num	Convert hexadecimal number string to double-precision number
num2hex	Convert singles and doubles to IEEE hexadecimal strings
cell2mat	Convert cell array to numeric array
cell2struct	Convert cell array to structure array
cellstr	Create cell array of strings from character array
mat2cell	Convert array to cell array with potentially different sized cells
num2cell	Convert array to cell array with consistently sized cells
struct2cell	Convert structure to cell array

Determination of Data Types

MATLAB provides various functions for identifying data type of a variable.

Following table provides the functions for determining the data type of a variable:

Function	Purpose
is	Detect state
isa	Determine if input is object of specified class
iscell	Determine whether input is cell array
iscellstr	Determine whether input is cell array of strings
ischar	Determine whether item is character array
isfield	Determine whether input is structure array field
isfloat	Determine if input is floating-point array
ishghandle	True for Handle Graphics object handles
isinteger	Determine if input is integer array
isjava	Determine if input is Java object
islogical	Determine if input is logical array
isnumeric	Determine if input is numeric array
isobject	Determine if input is MATLAB object
isreal	Check if input is real array
isscalar	Determine whether input is scalar

isstr	Determine whether input is character array
isstruct	Determine whether input is structure array
isvector	Determine whether input is vector
class	Determine class of object
validateattributes	Check validity of array
whos	List variables in workspace, with sizes and types

Example

Create a script file with the following code:

```
x = 3

isinteger(x)

isfloat(x)

isvector(x)

isscalar(x)

isnumeric(x)


x = 23.54

isinteger(x)

isfloat(x)

isvector(x)

isscalar(x)

isnumeric(x)


x = [1 2 3]
```

```
isinteger(x)

isfloat(x)

isvector(x)

isscalar(x)


x = 'Hello'

isinteger(x)

isfloat(x)

isvector(x)

isscalar(x)

isnumeric(x)
```

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

```
x =

     3

ans =

     0

ans =

     1

ans =

     1

ans =

     1

ans =

     1

x =

   23.5400
```

```
ans =

     0

ans =

     1

ans =

     1

ans =

     1

ans =

     1

x =

     1     2     3

ans =

     0

ans =

     1

ans =

     1

ans =

     0

x =

Hello

ans =

     0

ans =

     0
```

```
ans =

     1

ans =

     0

ans =

     0
```

8. 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. A+B adds the values stored in variables A and B. A and B must have the same size, unless one is a scalar. A scalar can be added to a matrix of any size.
-	Subtraction or unary minus. A-B subtracts the value of B from A. A and B must have the same size, unless one is a scalar. A scalar can be subtracted from a matrix of any size.

*	Matrix multiplication. C = A*B is the linear algebraic product of the matrices A and B. More precisely, $C(i,j) = \sum_{k=1}^{n} A(i,k)B(k,j)$ For non-scalar A and B, the number of columns of A must be equal to the number of rows of B. A scalar can multiply a matrix of any size.
.*	Array multiplication. A.*B is the element-by-element product of the arrays A and B. A and B must have the same size, unless one of them is a scalar.
/	Slash or matrix right division. B/A is roughly the same as B*inv(A). More precisely, B/A = (A'\B')'.
./	Array right division. A./B is the matrix with elements A(i,j)/B(i,j). A and B must have the same size, unless one of them is a scalar.
\	Backslash or matrix left division. If A is a square matrix, A\B is roughly the same as inv(A)*B, except it is computed in a different way. If A is an n-by-n matrix and B is a column vector with n components, or a matrix with several such columns, then X = A\B is the solution to the equation *AX* = *B*. A warning message is displayed if A is badly scaled or nearly singular.
.\	Array left division. A.\B is the matrix with elements B(i,j)/A(i,j). A and B must have the same size, unless one of them is a scalar.
^	Matrix power. X^p is X to the power p, if p is a scalar. If p is an integer, the power is computed by repeated squaring. If the integer is negative, X is inverted first. For other values of p, the calculation involves eigenvalues and eigenvectors, such that if [V,D] = eig(X), then X^p = V*D.^p/V.
.^	Array power. A.^B is the matrix with elements A(i,j) to the B(i,j) power. A and B must have the same size, unless one of them is a scalar.

'	Matrix transpose. A' is the linear algebraic transpose of A. For complex matrices, this is the complex conjugate transpose.
.'	Array transpose. A.' is the array transpose of A. For complex matrices, this does not involve conjugation.

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 =

    30

d =

   -10

e =

   200

f =

    0.5000
```

```
g =

     2

z =

   343
```

Functions for Arithmetic Operations

Apart from the above-mentioned arithmetic operators, MATLAB provides the following commands/functions used for similar purpose:

Function	Description
uplus(a)	Unary plus; increments by the amount a
plus (a,b)	Plus; returns a + b
uminus(a)	Unary minus; decrements by the amount a
minus(a, b)	Minus; returns a - b
times(a, b)	Array multiply; returns a.*b
mtimes(a, b)	Matrix multiplication; returns a* b
rdivide(a, b)	Right array division; returns a ./ b
ldivide(a, b)	Left array division; returns a.\ b
mrdivide(A, B)	Solve systems of linear equations $xA = B$ for x
mldivide(A, B)	Solve systems of linear equations $Ax = B$ for x
power(a, b)	Array power; returns a.^b
mpower(a, b)	Matrix power; returns a ^ b

cumprod(A)	Cumulative product; returns an array of the same size as the array A containing the cumulative product. If A is a vector, then cumprod(A) returns a vector containing the cumulative product of the elements of A. If A is a matrix, then cumprod(A) returns a matrix containing the cumulative products for each column of A. If A is a multidimensional array, then cumprod(A) acts along the first non-singleton dimension.
cumprod(A, dim)	Returns the cumulative product along dimension *dim*.
cumsum(A)	Cumulative sum; returns an array A containing the cumulative sum. If A is a vector, then cumsum(A) returns a vector containing the cumulative sum of the elements of A. If A is a matrix, then cumsum(A) returns a matrix containing the cumulative sums for each column of A. If A is a multidimensional array, then cumsum(A) acts along the first non-singleton dimension.
cumsum(A, dim)	Returns the cumulative sum of the elements along dimension *dim*.
diff(X)	Differences and approximate derivatives; calculates differences between adjacent elements of X. If X is a vector, then diff(X) returns a vector, one element shorter than X, of differences between adjacent elements: [X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)] If X is a matrix, then diff(X) returns a matrix of row differences: [X(2:m,:)-X(1:m-1,:)]
diff(X,n)	Applies *diff* recursively n times, resulting in the nth difference.

idivide(A, B, 'ceil')	Fractional quotients are rounded toward infinity to the nearest integers.
mod (X,Y)	Modulus after division; returns X - n.*Y where n = floor(X./Y). If Y is not an integer and the quotient X./Y is within round off error of an integer, then n is that integer. The inputs X and Y must be real arrays of the same size, or real scalars (provided Y ~=0). Please note: mod(X,0) is X mod(X,X) is 0 mod(X,Y) for X~=Y and Y~=0 has the same sign as Y
rem (X,Y)	Remainder after division; returns X - n.*Y where n = fix(X./Y). If Y is not an integer and the quotient X./Y is within round off error of an integer, then n is that integer. The inputs X and Y must be real arrays of the same size, or real scalars (provided Y ~=0). Please note that: rem(X,0) is NaN rem(X,X) for X~=0 is 0 rem(X,Y) for X~=Y and Y~=0 has the same sign as X.
round(X)	Round to nearest integer; rounds the elements of X to the nearest integers. Positive elements with a fractional part of 0.5 round up to the nearest positive integer. Negative elements with a fractional part of -0.5 round down to the nearest negative integer.

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
```

Apart from the above-mentioned relational operators, MATLAB provides the following commands/functions used for the same purpose:

Function	Description
eq(a, b)	Tests whether a is equal to b
ge(a, b)	Tests whether a is greater than or equal to b
gt(a, b)	Tests whether a is greater than b
le(a, b)	Tests whether a is less than or equal to b
lt(a, b)	Tests whether a is less than b
ne(a, b)	Tests whether a is not equal to b
isequal	Tests arrays for equality
isequaln	Tests arrays for equality, treating NaN values as equal

Example

Create a script file and type the following code:

```
% comparing two values

a = 100;

b = 200;

if (ge(a,b))

max = a

else

max = b

end

% comparing two different values

a = 340;

b = 520;
```

```
if (le(a, b))

disp(' a is either less than or equal to b')

else

disp(' a is greater than b')

end
```

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

```
max =

   200

 a is either less than or equal to b
```

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
```

Functions for Logical Operations

Apart from the above-mentioned logical operators, MATLAB provides the following commands or functions used for the same purpose:

Function	Description
and(A, B)	Finds logical AND of array or scalar inputs; performs a logical AND of all input arrays A, B, etc. and returns an array containing elements set to either logical 1 (true) or logical 0 (false). An

	element of the output array is set to 1 if all input arrays contain a nonzero element at that same array location. Otherwise, that element is set to 0.
not(A)	Finds logical NOT of array or scalar input; performs a logical NOT of input array A and returns an array containing elements set to either logical 1 (true) or logical 0 (false). An element of the output array is set to 1 if the input array contains a zero value element at that same array location. Otherwise, that element is set to 0.
or(A, B)	Finds logical OR of array or scalar inputs; performs a logical OR of all input arrays A, B, etc. and returns an array containing elements set to either logical 1 (true) or logical 0 (false). An element of the output array is set to 1 if any input arrays contain a nonzero element at that same array location. Otherwise, that element is set to 0.
xor(A, B)	Logical exclusive-OR; performs an exclusive OR operation on the corresponding elements of arrays A and B. The resulting element C(i,j,...) is logical true (1) if A(i,j,...) or B(i,j,...), but not both, is nonzero.
all(A)	Determine if all array elements of array A are nonzero or true. If A is a vector, all(A) returns logical 1 (true) if all the elements are nonzero and returns logical 0 (false) if one or more elements are zero. If A is a nonempty matrix, all(A) treats the columns of A as vectors, returning a row vector of logical 1's and 0's. If A is an empty 0-by-0 matrix, all(A) returns logical 1 (true). If A is a multidimensional array, all(A) acts along the first non-singleton dimension and returns an array of logical values. The size of this dimension

	reduces to 1 while the sizes of all other dimensions remain the same.
all(A, dim)	Tests along the dimension of A specified by scalar *dim*.
any(A)	Determine if any array elements are nonzero; tests whether any of the elements along various dimensions of an array is a nonzero number or is logical 1 (true). The *any* function ignores entries that are NaN (Not a Number). If A is a vector, any(A) returns logical 1 (true) if any of the elements of A is a nonzero number or is logical 1 (true), and returns logical 0 (false) if all the elements are zero. If A is a nonempty matrix, any(A) treats the columns of A as vectors, returning a row vector of logical 1's and 0's. If A is an empty 0-by-0 matrix, any(A) returns logical 0 (false). If A is a multidimensional array, any(A) acts along the first non-singleton dimension and returns an array of logical values. The size of this dimension reduces to 1 while the sizes of all other dimensions remain the same.
any(A,dim)	Tests along the dimension of A specified by scalar *dim*.
False	Logical 0 (false)
false(n)	is an n-by-n matrix of logical zeros
false(m, n)	is an m-by-n matrix of logical zeros.
false(m, n, p, ...)	is an m-by-n-by-p-by-... array of logical zeros.

false(size(A))	is an array of logical zeros that is the same size as array A.
false(...,'like',p)	is an array of logical zeros of the same data type and sparsity as the logical array p.
ind = find(X)	Find indices and values of nonzero elements; locates all nonzero elements of array X, and returns the linear indices of those elements in a vector. If X is a row vector, then the returned vector is a row vector; otherwise, it returns a column vector. If X contains no nonzero elements or is an empty array, then an empty array is returned.
ind = find(X, k) **ind = find(X, k, 'first')**	Returns at most the first k indices corresponding to the nonzero entries of X. k must be a positive integer, but it can be of any numeric data type.
ind = find(X, k, 'last')	returns at most the last k indices corresponding to the nonzero entries of X.
[row,col] = find(X, ...)	Returns the row and column indices of the nonzero entries in the matrix X. This syntax is especially useful when working with sparse matrices. If X is an N-dimensional array with N > 2, col contains linear indices for the columns.
[row,col,v] = find(X, ...)	Returns a column or row vector v of the nonzero entries in X, as well as row and column indices. If X is a logical expression, then v is a logical array. Output v contains the non-zero elements of the logical array obtained by evaluating the expression X.
islogical(A)	Determine if input is logical array; returns true if A is a logical array and false otherwise. It also returns true if A is an instance of a class that is derived from the logical class.

logical(A)	Convert numeric values to logical; returns an array that can be used for logical indexing or logical tests.
True	Logical 1 (true)
true(n)	is an n-by-n matrix of logical ones.
true(m, n)	is an m-by-n matrix of logical ones.
true(m, n, p, ...)	is an m-by-n-by-p-by-... array of logical ones.
true(size(A))	is an array of logical ones that is the same size as array A.
true(...,'like', p)	is an array of logical ones of the same data type and sparsity as the logical array p.

Example

Create a script file and type the following code:

```
a = 60; % 60 = 0011 1100

b = 13; % 13 = 0000 1101

c = bitand(a, b)      % 12 = 0000 1100

c = bitor(a, b)       % 61 = 0011 1101

c = bitxor(a, b)      % 49 = 0011 0001

c = bitshift(a, 2)    % 240 = 1111 0000 */

c = bitshift(a,-2)    % 15 = 0000 1111 */
```

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

```
c =

    12

c =
```

```
    61

c =

    49

c =

   240

c =

    15
```

Bitwise Operations

Bitwise operators work on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows:

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume if A = 60; and B = 13; Now in binary format they will be as follows:

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

MATLAB provides various functions for bit-wise operations like 'bitwise and', 'bitwise or' and 'bitwise not' operations, shift operation, etc.

The following table shows the commonly used bitwise operations:

Function	Purpose
bitand(a, b)	Bit-wise AND of integers *a* and *b*
bitcmp(a)	Bit-wise complement of *a*
bitget(a,pos)	Get bit at specified position *pos*, in the integer array *a*
bitor(a, b)	Bit-wise OR of integers *a* and *b*
bitset(a, pos)	Set bit at specific location *pos* of *a*
bitshift(a, k)	Returns *a* shifted to the left by *k* bits, equivalent to multiplying by 2^k. Negative values of k correspond to shifting bits right or dividing by $2^{|k|}$ and rounding to the nearest integer towards negative infinite. Any overflow bits are truncated.
bitxor(a, b)	Bit-wise XOR of integers *a* and *b*
swapbytes	Swap byte ordering

Example

Create a script file and type the following code:

```
a = 60; % 60 = 0011 1100

b = 13; % 13 = 0000 1101

c = bitand(a, b)      % 12 = 0000 1100

c = bitor(a, b)       % 61 = 0011 1101

c = bitxor(a, b)      % 49 = 0011 0001

c = bitshift(a, 2)    % 240 = 1111 0000 */

c = bitshift(a,-2)    % 15 = 0000 1111 */
```

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

```
c =

    12

c =

    61

c =

    49

c =

   240

c =

    15
```

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)
```

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

```
a =

     7    23    14    15     9    12     8    24    35

b =

     2     5     7     8    14    16    25    35    27

u =

  Columns 1 through 11

     2     5     7     8     9    12    14    15    16    23    24

  Columns 12 through 14

    25    27    35

i =

     7     8    14    35

s =

     9    12    15    23    24
```