4. VARIABLES

In MATLAB environment, every variable is an array or matrix.

You can assign variables in a simple way. For example,

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

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

```
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
```


tutorialspoint
SIMPLYEASYLEARNING

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
```

I have forgotten the Variables!

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:

Variables currently in memory

Type of each variables

Memory allocated to each variable

Whether they are complex variables or not

```
whos
```

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

```
  Name      Size            Bytes  Class     Attributes


  a         1x1                 8  double

  ans       1x1                 8  double

  b         1x1                 8  double

  c         1x1                 8  double

  x         1x1                 8  double

  y         1x1                 8  double
```

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

```
clear x     % it will delete x, won't display anything

clear      % it will delete all variables in the workspace

             %  peacefully and unobtrusively
```

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
```

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.

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
```

Another example,

```
format short

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

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

```
x =

   17.2320
```

The **format bank** command rounds numbers to two decimal places. For example,

tutorialspoint
SIMPLYEASYLEARNING

```
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
```

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
```

The **format long e** command allows displaying in exponential form with four decimal places plus the exponent. For example,

```
format long e

x = pi
```

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

```
x =

     3.141592653589793e+00
```

The **format rat** command gives the closest rational expression resulting from a calculation. For example,

```
format rat
```


tutorialspoint
SIMPLYEASYLEARNING

```
4.678 * 4.9
```

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

```
ans =

   2063/90
```

Creating Vectors

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

Row vectors

Column vectors

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

For example,

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

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

```
r =

  Columns 1 through 4

     7            8            9           10

  Column 5

    11
```

Another example,

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

t = [2, 3, 4, 5, 6];

res = r + t
```

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

tutorialspoint
SIMPLYEASYLEARNING

```
res =

  Columns 1 through 4

     9           11           13           15

  Column 5

    17
```

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]
```

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

```
c =

      7

      8

      9

     10

     11
```

Creating Matrices

A matrix is a two-dimensional array of numbers.

In MATLAB, a matrix is created by entering each row as a sequence of space or comma separated elements, and end of a row is demarcated by a semicolon. For example, let us create a 3-by-3 matrix as:

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

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

```
m =

      1           2           3

      4           5           6
```


tutorialspoint
SIMPLYEASYLEARNING

7	8	9	

tutorialspoint
SIMPLYEASYLEARNING

11. VECTORS

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

Row vectors

Column 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]
```

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

```
r =

  Columns 1 through 4

     7            8            9           10

  Column 5

    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]
```

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

```
c =

      7

      8
```


tutorialspoint
SIMPLYEASYLEARNING

```
     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 i^{th}component 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)
```

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

```
ans =

    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 will execute the above statement and return the following result:

```
ans =

    1

    2

    3

    4

    5

    6
```

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)
```

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

```
sub_rv =

    3     4     5     6     7
```

Vector Operations

In this section, let us discuss the following vector operations:

Addition and Subtraction of Vectors

Scalar Multiplication of Vectors

Transpose of a Vector

Appending Vectors

Magnitude of a Vector

Vector Dot Product

Vectors with Uniformly Spaced Elements

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);
```

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

```
9    16    28    39    29

5     6     2     7   -11
```

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
```

Please note that you can perform all scalar operations on vectors. For example, you can add, subtract and divide a vector with a scalar quantity.

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';
```


tutorialspoint
SIMPLYEASYLEARNING

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

tv = v';

disp(tr); disp(tv);
```

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

```
    1

    2

    3

    4


    1     2     3     4
```

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:

tutorialspoint
SIMPLYEASYLEARNING

```
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]
```

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

```
r =

    1    2    3    4    5    6    7    8

rMat =

    1    2    3    4

    5    6    7    8

c =

    1

    2

    3

    4

    5
```


tutorialspoint
SIMPLYEASYLEARNING

```
    6

    7

    8

cMat =

    1    5

    2    6

    3    7

    4    8
```

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);

Example

Create a script file with the following code:

```
v = [1: 2: 20];

sv = v.* v;     %the vector with elements

               % as square of v's elements

dp = sum(sv);    % sum of squares -- the dot product

mag = sqrt(dp);  % magnitude
```


tutorialspoint
SIMPLYEASYLEARNING

```
disp('Magnitude:'); disp(mag);
```

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

```
Magnitude:

   36.4692
```

Vector Dot Product

Dot product of two vectors a = (a1, a2, …, an) and b = (b1, b2, …, bn) is given by:

a.b = ∑(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:

tutorialspoint
SIMPLYEASYLEARNING

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

Example

Create a script file with the following code:

```
v = [1: 2: 20];

sqv = v.^2;

disp(v);disp(sqv);
```

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

```
1     3     5     7     9    11    13    15    17    19

1     9    25    49    81   121   169   225   289   361
```


tutorialspoint
SIMPLYEASYLEARNING

12. 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 m^{th} row and n^{th} column, of a matrix *mx*, we write:

```
mx(m, n);
```

For example, to refer to the element in the 2^{nd} row and 5^{th} 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
```


tutorialspoint
SIMPLYEASYLEARNING

To reference all the elements in the m^{th} column we type A(:,m).

Let us create a column vector v, from the elements of the 4^{th} 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)
```

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

```
v =

    4

    5

    6

    7
```

You can also select the elements in the m^{th} through n^{th} 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:

```
ans =

    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)
```

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

```
ans =

    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.

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)
```

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

```
sa =

    3     4     5

    4     5     6
```

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],:)
```

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

```
new_mat =

    4    5    6
```


tutorialspoint
SIMPLYEASYLEARNING

```
    7     8     9

    4     5     6

    7     8     9
```

Matrix Operations

In this section, let us discuss the following basic and commonly used matrix operations:

Addition and Subtraction of Matrices

Division of Matrices

Scalar Operations of Matrices

Transpose of a Matrix

Concatenating Matrices

Matrix Multiplication

Determinant of a Matrix

Inverse of a Matrix

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
```

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

```
c =

    8     7     9

    6     5    14
```


tutorialspoint
SIMPLYEASYLEARNING

```
   12    15    10

d =

   -6    -3    -3

    2     5    -2

    2     1     8
```

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
```

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

```
c =


  -0.52542   0.68644   0.66102

  -0.42373   0.94068   1.01695

  -0.32203   1.19492   1.37288


d =


  -3.27778  -1.05556  -4.86111

  -0.11111   0.11111  -0.27778
```


tutorialspoint
SIMPLYEASYLEARNING

```
   3.05556   1.27778   4.30556
```

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
```

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

```
c =

    12    14    25

    16    10     8

    29    10    11

d =

     8    10    21

    12     6     4

    25     6     7

e =

    20    24    46
```


tutorialspoint
SIMPLYEASYLEARNING

```
    28    16    12

    54    16    18

f =

    5.0000    6.0000   11.5000

    7.0000    4.0000    3.0000

   13.5000    4.0000    4.5000
```

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'
```

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

```
a =

    10    12    23

    14     8     6

    27     8     9

b =

    10    14    27

    12     8     8

    23     6     9
```

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]
```

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

```
a =

   10   12   23

   14    8    6

   27    8    9

b =

   12   31   45

    8    0   -9

   45    2   11

c =

   10   12   23   12   31   45

   14    8    6    8    0   -9

   27    8    9   45    2   11

d =
```


tutorialspoint
SIMPLYEASYLEARNING

```
   10    12    23

   14     8     6

   27     8     9

   12    31    45

    8     0    -9

   45     2    11
```

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 i^{th} row of first matrix with the corresponding element in the j^{th} 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
```

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

```
a =

     1     2     3

     2     3     4

     1     2     5
```


tutorialspoint
SIMPLYEASYLEARNING

```
b =

     2     1     3

     5     0    -2

     2     3    -1

prod =

    18    10    -4

    27    14    -4

    22    16    -6
```

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)
```

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

```
a =

     1     2     3

     2     3     4

     1     2     5

ans =

    -2
```

Inverse of a Matrix

The inverse of a matrix A is denoted by A^{-1} such that the following relationship holds:

tutorialspoint
SIMPLYEASYLEARNING

```
AA–1 = A–1A = 1
```

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)
```

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

```
a =

    1    2    3

    2    3    4

    1    2    5

ans =

   -3.5000    2.0000    0.5000

    3.0000   -1.0000   -1.0000

   -0.5000         0    0.5000
```


tutorialspoint
SIMPLYEASYLEARNING

13. 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

In this section, we will discuss some functions that create some special arrays. For all these functions, a single argument creates a square array, double arguments create rectangular array.

The **zeros()** function creates an array of all zeros:

For example:

```
zeros(5)
```

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

```
ans =

     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
```

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:

tutorialspoint
SIMPLYEASYLEARNING

```
ans =

    1    1    1

    1    1    1

    1    1    1

    1    1    1
```

The **eye()** function creates an identity matrix.

For example:

```
eye(4)
```

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

```
ans =

    1    0    0    0

    0    1    0    0

    0    0    1    0

    0    0    0    1
```

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:

```
ans =

   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
```

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:

```
ans =

   16    2    3   13

    5   11   10    8

    9    7    6   12

    4   14   15    1
```

Multidimensional Arrays

An array having more than two dimensions is called a multidimensional array in MATLAB. Multidimensional arrays in MATLAB are an extension of the normal two-dimensional matrix.

Generally to generate a multidimensional array, we first create a two-dimensional array and extend it.

For example, let's create a two-dimensional array a.

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

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

```
a =

    7    9    5

    6    1    9

    4    3    2
```


tutorialspoint
SIMPLYEASYLEARNING

The array *a* is a 3-by-3 array; we can add a third dimension to *a*, by providing the values like:

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

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

```
a(:,:,1) =

    7    9    5

    6    1    9

    4    3    2


a(:,:,2) =

    1    2    3

    4    5    6

    7    8    9
```

We can also create multidimensional arrays using the ones(), zeros() or the rand() functions.

For example,

```
b = rand(4,3,2)
```

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

```
b(:,:,1) =

   0.0344    0.7952    0.6463

   0.4387    0.1869    0.7094

   0.3816    0.4898    0.7547

   0.7655    0.4456    0.2760


b(:,:,2) =

   0.6797    0.4984    0.2238
```


tutorialspoint
SIMPLYEASYLEARNING

```
    0.6551    0.9597    0.7513

    0.1626    0.3404    0.2551

    0.1190    0.5853    0.5060
```

We can also use the **cat()** function to build multidimensional arrays. It concatenates a list of arrays along a specified dimension:

Syntax for the cat() function is:

```
B = cat(dim, A1, A2...)
```

Where,

B is the new array created

A1, *A2*, ... are the arrays to be concatenated

dim is the dimension along which to concatenate the arrays

Example

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

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

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

c = cat(3, a, b, [ 2 3 1; 4 7 8; 3 9 0])
```

When you run the file, it displays:

```
c(:,:,1) =

    9    8    7

    6    5    4

    3    2    1

c(:,:,2) =

    1    2    3

    4    5    6

    7    8    9

c(:,:,3) =
```


tutorialspoint
SIMPLYEASYLEARNING

```
    2     3     1

    4     7     8

    3     9     0
```

Array Functions

MATLAB provides the following functions to sort, rotate, permute, reshape, or shift array contents.

Function	Purpose
length	Length of vector or largest array dimension
ndims	Number of array dimensions
numel	Number of array elements
size	Array dimensions
iscolumn	Determines whether input is column vector
isempty	Determines whether array is empty
ismatrix	Determines whether input is matrix
isrow	Determines whether input is row vector
isscalar	Determines whether input is scalar
isvector	Determines whether input is vector
blkdiag	Constructs block diagonal matrix from input arguments
circshift	Shifts array circularly
ctranspose	Complex conjugate transpose

diag	Diagonal matrices and diagonals of matrix
flipdim	Flips array along specified dimension
fliplr	Flips matrix from left to right
flipud	Flips matrix up to down
ipermute	Inverses permute dimensions of N-D array
permute	Rearranges dimensions of N-D array
repmat	Replicates and tile array
reshape	Reshapes array
rot90	Rotates matrix 90 degrees
shiftdim	Shifts dimensions
issorted	Determines whether set elements are in sorted order
sort	Sorts array elements in ascending or descending order
sortrows	Sorts rows in ascending order
squeeze	Removes singleton dimensions
transpose	Transpose
vectorize	Vectorizes expression

Examples

The following examples illustrate some of the functions mentioned above.

Length, Dimension and Number of elements:

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

tutorialspoint
SIMPLYEASYLEARNING

```
x = [7.1, 3.4, 7.2, 28/4, 3.6, 17, 9.4, 8.9];

length(x)  % length of x vector

y = rand(3, 4, 5, 2);

ndims(y)    % no of dimensions in array y

s = ['Zara', 'Nuha', 'Shamim', 'Riz', 'Shadab'];

numel(s)   % no of elements in s
```

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

```
ans =

     8

ans =

     4

ans =

    23
```

Circular Shifting of the Array Elements:

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

```
a = [1 2 3; 4 5 6; 7 8 9]  % the original array a

b = circshift(a,1)  %  circular shift first dimension values down by 1.

c = circshift(a,[1 -1]) % circular shift first dimension values % down by 1

                        % and second dimension values to the left % by 1.
```

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

```
a =

     1     2     3

     4     5     6

     7     8     9
```


tutorialspoint
SIMPLYEASYLEARNING

```
b =

    7    8    9

    1    2    3

    4    5    6


c =

    8    9    7

    2    3    1

    5    6    4
```

Sorting Arrays

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

```
v = [ 23 45 12 9 5 0 19 17]  % horizonal vector

sort(v)   %sorting v

m = [2 6 4; 5 3 9; 2 0 1]  %  two dimensional array

sort(m, 1)   % sorting m along the row

sort(m, 2)   % sorting m along the column
```

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

```
v =

   23   45   12    9    5    0   19   17

ans =

    0    5    9   12   17   19   23   45

m =

    2    6    4

    5    3    9

    2    0    1
```


tutorialspoint
SIMPLYEASYLEARNING

```
ans =

    2    0    1

    2    3    4

    5    6    9

ans =

    2    4    6

    3    5    9

    0    1    2
```

Cell Array

Cell arrays are arrays of indexed cells where each cell can store an array of a different dimensions and data types.

The **cell** function is used for creating a cell array. Syntax for the cell function is:

```
C = cell(dim)

C = cell(dim1,...,dimN)

D = cell(obj)
```

Where,

C is the cell array;

dim is a scalar integer or vector of integers that specifies the dimensions of cell array C;

dim1, ... , dimN are scalar integers that specify the dimensions of C;

obj is One of the following:

Java array or object

.NET array of type System.String or System.Object

Example

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

```
c = cell(2, 5);

c = {'Red', 'Blue', 'Green', 'Yellow', 'White'; 1 2 3 4 5}
```


tutorialspoint
SIMPLYEASYLEARNING

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

```
c =

    'Red'    'Blue'    'Green'    'Yellow'    'White'

    [  1]    [   2]    [    3]    [    4]    [    5]
```

Accessing Data in Cell Arrays

There are two ways to refer to the elements of a cell array:

Enclosing the indices in first bracket (), to refer to sets of cells

Enclosing the indices in braces {}, to refer to the data within individual cells

When you enclose the indices in first bracket, it refers to the set of cells.

Cell array indices in smooth parentheses refer to sets of cells.

For example:

```
c = {'Red', 'Blue', 'Green', 'Yellow', 'White'; 1 2 3 4 5};

c(1:2,1:2)
```

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

```
ans =

    'Red'    'Blue'

    [  1]    [   2]
```

You can also access the contents of cells by indexing with curly braces.

For example:

```
c = {'Red', 'Blue', 'Green', 'Yellow', 'White'; 1 2 3 4 5};

c{1, 2:4}
```

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

```
ans =

   Blue
```


tutorialspoint
SIMPLYEASYLEARNING

```
ans =

   Green

ans =

   Yellow
```

14. COLON NOTATION

The **colon(:)** is one of the most useful operator in MATLAB. It is used to create vectors, subscript arrays, and specify for iterations.

If you want to create a row vector, containing integers from 1 to 10, you write:

```
1:10
```

MATLAB executes the statement and returns a row vector containing the integers from 1 to 10:

```
ans =

    1    2    3    4    5    6    7    8    9   10
```

If you want to specify an increment value other than one, for example:

```
100: -5: 50
```

MATLAB executes the statement and returns the following result:

```
ans =

   100    95    90    85    80    75    70    65    60    55    50
```

Let us take another example:

```
0:pi/8:pi
```

MATLAB executes the statement and returns the following result:

```
ans =

  Columns 1 through 7

         0    0.3927    0.7854    1.1781    1.5708    1.9635    2.3562

  Columns 8 through 9

    2.7489    3.1416
```


tutorialspoint
SIMPLYEASYLEARNING

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 k^{th} 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.

Example

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

```
A = [1 2 3 4; 4 5 6 7; 7 8 9 10]

A(:,2)      % second column of A

A(:,2:3)    % second and third column of A

A(2:3,2:3)  % second and third rows and second and third columns
```


tutorialspoint
SIMPLYEASYLEARNING

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

```
A =

     1     2     3     4

     4     5     6     7

     7     8     9    10


ans =

     2

     5

     8


ans =

     2     3

     5     6

     8     9


ans =

     5     6

     8     9
```


tutorialspoint
SIMPLYEASYLEARNING