# ****Input****

# The ****input function**** is a MATLAB feature that allows for a program to include user inputted information. When the ****input**** function is ran, a customized prompt will be given to the user and the value that is typed will be assigned to a [variable](http://matlab.enge.vt.edu/variable.html).

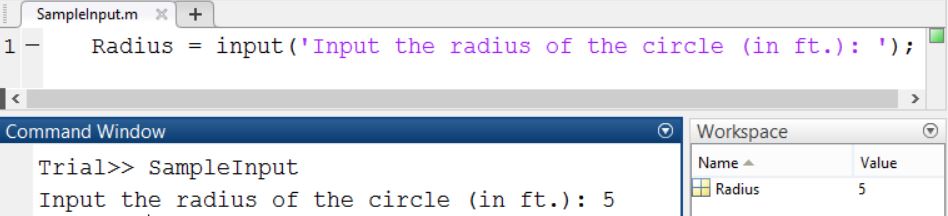
|  |
| --- |
| **Contents**  [1. How-to in MATLAB](http://matlab.enge.vt.edu/input.html#howtoinmatlab)  [1.1 Scalar Values](http://matlab.enge.vt.edu/input.html#scalar)  [1.2 Vector Values](http://matlab.enge.vt.edu/input.html#vector) |

How-to in MATLAB

**Scalar Values**

Assigning variables using the input function is similar to [assigning variables regularly](http://matlab.enge.vt.edu/scalar.html#howtoinmatlab). In this case, the value assigned to the variable is whatever is typed into the command window by the user.

In the following screenshot, the [script](http://matlab.enge.vt.edu/script.html) **SampleInput.m** assigns the value of the variable Radius using the input function with a customized prompt. After the script is run, when 5 is typed into the [Command Window](http://matlab.enge.vt.edu/programlayout.html#commandwindow), Radius is assigned that value in the [Workspace](http://matlab.enge.vt.edu/programlayout.html#workspace).



# ****disp and display****

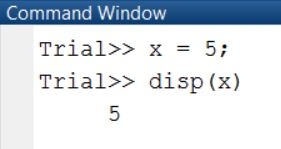
# There a two basic functions to output the values of [variables](http://matlab.enge.vt.edu/variable.html), ****disp**** and ****display****. ****disp**** and ****display**** can be also used to display simple numbers and text and are not exclusively used with variables.

|  |
| --- |
| **Contents**  [1. disp](http://matlab.enge.vt.edu/display.html#disp)  [2. display](http://matlab.enge.vt.edu/display.html#display) |

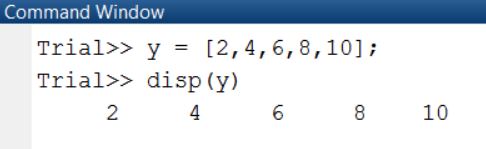
disp

**disp** is the simpler version of the two. It will only output the value of variable, and not the variable name.

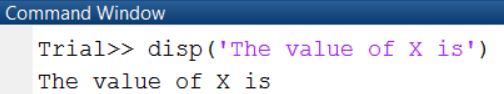
In the following screenshot, the value of the [scalar](http://matlab.enge.vt.edu/scalar.html) variable x is outputted using the **disp** function.



**disp** can also be used to display vector values. The following screenshot shows the output of the [vector](http://matlab.enge.vt.edu/vector.html) variable y.

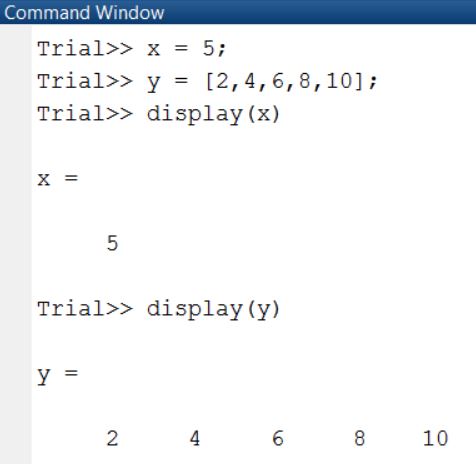


If you want to display text using **disp**, place the text inside apostrophes.



display

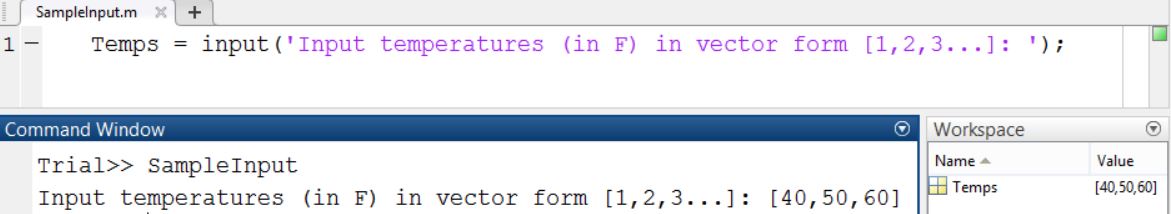
**display** will output both the variable name and the value of the variable. Using the above [scalar](http://matlab.enge.vt.edu/scalar.html) and [vector](http://matlab.enge.vt.edu/vector.html) examples, the output using the **display** function would be as follows.



**Vector Values**

Similarly, the input function can be used to input [vectors](http://matlab.enge.vt.edu/vector.html). In order to operate properly, the user must input the values using brackets.

In the following screenshot, the [script](http://matlab.enge.vt.edu/script.html) **SampleInput.m** assigns the value of the variable Temps using the input function with a customized prompt. After the script is run, when 40, 50, and 50 are typed into the [Command Window](http://matlab.enge.vt.edu/programlayout.html#commandwindow), Temps is assigned those values in the [Workspace](http://matlab.enge.vt.edu/programlayout.html#workspace).



**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.

[Live Demo](http://tpcg.io/LmINmJ)

r = [7 8 9 10 11]

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

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.

[Live Demo](http://tpcg.io/KypbfL)

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

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

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 ith component of a vector v is referred as v(i). For example −

[Live Demo](http://tpcg.io/9o0sjC)

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.

[Live Demo](http://tpcg.io/DOEIeM)

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*.

[Live Demo](http://tpcg.io/JGNvrB)

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

# MATLAB - Addition & 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

# MATLAB - 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.

# MATLAB - 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);

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

1

2

3

4

1 2 3 4

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 −

[Live Demo](http://tpcg.io/QYrfjm)

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 =

Columns 1 through 7:

1 2 3 4 5 6 7

Column 8:

8

rMat =

1 2 3 4

5 6 7 8

c =

1

2

3

4

5

6

7

8

cMat =

1 5

2 6

3 7

4 8

# MATLAB - Magnitude of a Vector

Magnitude of a vector v with elements v1, v2, v3, …, vn, is given by the equation −

|v| = √(v12 + v22 + v32 + … + vn2)

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 = [2: 2: 10]; v = [2 4 6 8 10]

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

disp('Magnitude:');

disp(mag);

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

Magnitude:

36.469

=======================================

# MATLAB - 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 −

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

Additional inputs on Vectors

## Indexing Vectors

Let's start with the simple case of a vector and a single subscript. The vector is:

v = [16 5 9 4 2 11 7 14];

The subscript can be a single value:

v(3) % Extract the third element

ans =

9

Or the subscript can itself be another vector:

v([1 5 6]) % Extract the first, fifth, and sixth elements

ans =

16 2 11

The colon notation in MATLAB provides an easy way to extract a range of elements from v:

v(3:7) % Extract the third through the seventh elements

ans =

9 4 2 11 7

Swap the two halves of v to make a new vector:

v2 = v([5:8 1:4]) % Extract and swap the halves of v

v2 =

2 11 7 14 16 5 9 4

The special end operator is an easy shorthand way to refer to the last element of v:

v(end) % Extract the last element

ans =

14

The end operator can be used in a range:

v(5:end) % Extract the fifth through the last elements

ans =

2 11 7 14

You can even do arithmetic using end:

v(2:end-1) % Extract the second through the next-to-last elements

ans =

5 9 4 2 11 7

Combine the colon operator and end to achieve a variety of effects, such as extracting every k-th element or flipping the entire vector:

v(1:2:end) % Extract all the odd elements

ans =

16 9 2 7

v(end:-1:1) % Reverse the order of elements

ans =

14 7 11 2 4 9 5 16

By using an indexing expression on the left side of the equal sign, you can *replace* certain elements of the vector:

v([2 3 4]) = [10 15 20] % Replace some elements of v

v =

16 10 15 20 2 11 7 14

Usually the number of elements on the right must be the same as the number of elements referred to by the indexing expression on the left. You can always, however, use a *scalar* on the right side:

v = [16 5 9 4 2 11 7 14];

v([2 3]) = 30 % Replace second and third elements by 30

v =

16 30 30 20 2 11 7 14

This form of indexed assignment is called *scalar expansion*.

## Indexing Matrices with Two Subscripts

Now consider indexing into a matrix. We'll use a magic square for our experiments:

A = magic(4)

A =

16 2 3 13

5 11 10 8

9 7 6 12

4 14 15 1

Most often, indexing in matrices is done using two subscripts—one for the rows and one for the columns. The simplest form just picks out a single element:

A(2,4) % Extract the element in row 2, column 4

ans =

8

More generally, one or both of the row and column subscripts can be vectors:

A(2:4,1:2)

ans =

5 11

9 7

4 14

# MATLAB - 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.

# Array Indexing

Open in MATLAB Online

[View MATLAB Command](matlab:openExample('matlab/MatrixIndexingExample'))

In MATLAB®, there are three primary approaches to accessing array elements based on their location (index) in the array. These approaches are indexing by position, linear indexing, and logical indexing.

### Indexing with Element Positions

The most common way is to explicitly specify the indices of the elements. For example, to access a single element of a matrix, specify the row number followed by the column number of the element.

A = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]

A = 4×4

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

e = A(3,2)

e = 10

e is the element in the 3,2 position (third row, second column) of A.

You can also reference multiple elements at a time by specifying their indices in a vector. For example, access the first and third elements of the second row of A.

r = A(2,[1 3])

r = 1×2

5 7

To access elements in a range of rows or columns, use the [colon](https://in.mathworks.com/help/matlab/ref/colon.html). For example, access the elements in the first through third row and the second through fourth column of A.

A = 1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

r = A(1:3,2:4)

r = 3×3

2 3 4

6 7 8

10 11 12

An alternative way to compute r is to use the keyword end to specify the second column through the last column. This approach lets you specify the last column without knowing exactly how many columns are in A.

r = A(1:3,2:end)

r = 3×3

2 3 4

6 7 8

10 11 12

If you want to access all of the rows or columns, use the colon operator by itself. For example, return the entire third column of A.

r = A(:,3)

r = 4×1

3

7

11

15

## 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 −

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

## =====================================================================

[View MATLAB Command](matlab:openExample('matlab/ArrayIndexingGSExample'))

Every variable in MATLAB® is an array that can hold many numbers. When you want to access selected elements of an array, use indexing.

For example, consider the 4-by-4 matrix A:

A = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]

A = 4×4

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

There are two ways to refer to a particular element in an array. The most common way is to specify row and column subscripts, such as

A(4,2)

ans = 14

Less common, but sometimes useful, is to use a single subscript that traverses down each column in order:

A(8)

ans = 14

Using a single subscript to refer to a particular element in an array is called linear indexing.

If you try to refer to elements outside an array on the right side of an assignment statement, MATLAB throws an error.

test = A(4,5)

Index in position 2 exceeds array bounds (must not exceed 4).

However, on the left side of an assignment statement, you can specify elements outside the current dimensions. The size of the array increases to accommodate the newcomers.

A(4,5) = 17

A = 4×5

1 2 3 4 0

5 6 7 8 0

9 10 11 12 0

13 14 15 16 17

To refer to multiple elements of an array, use the colon operator, which allows you to specify a range of the form start:end. For example, list the elements in the first three rows and the second column of A:

A(1:3,2)

ans = 3×1

2

6

10

The colon alone, without start or end values, specifies all of the elements in that dimension. For example, select all the columns in the third row of A:

A(3,:)

ans = 1×5

9 10 11 12 0

The colon operator also allows you to create an equally spaced vector of values using the more general form start:step:end.

B = 0:10:100

B = 1×11

0 10 20 30 40 50 60 70 80 90 100

If you omit the middle step, as in start:end, MATLAB uses the default step value of 1.

## Indexing Matrices with Two Subscripts

Now consider indexing into a matrix. We'll use a magic square for our experiments:

A = magic(4)

A =

16 2 3 13

5 11 10 8

9 7 6 12

4 14 15 1

Most often, indexing in matrices is done using two subscripts—one for the rows and one for the columns. The simplest form just picks out a single element:

A(2,4) % Extract the element in row 2, column 4

ans =

8

More generally, one or both of the row and column subscripts can be vectors:

A(2:4,1:2)

ans =

5 11

9 7

4 14

A single : in a subscript position is shorthand notation for 1:end and is often used to select entire rows or columns:

**A(3,:) % Extract third row**

**ans =**

**9 7 6 12**

**A(:,end) % Extract last column**

**ans =**

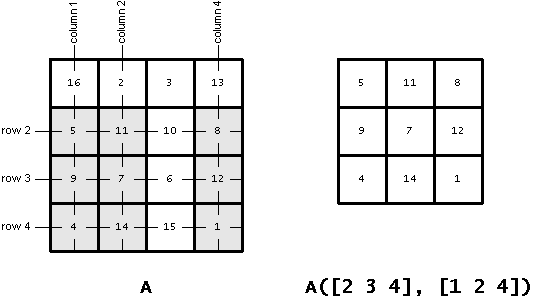
**13**

**8**

**12**

**1**

There is often confusion over how to select scattered elements from a matrix. For example, suppose you want to extract the (2,1), (3,2), and (4,4) elements from A. The expression A([2 3 4], [1 2 4]) won't do what you want. This diagram illustrates how two-subscript indexing works:



Extracting scattered elements from a matrix requires a different style of indexing, and that brings us to our next topic.

## ==========================================================

## Linear Indexing

What does this expression A(14) do?

When you index into the matrix A using only one subscript, MATLAB treats A as if its elements were strung out in a long column vector, by going down the columns consecutively, as in:

16

5

9

...

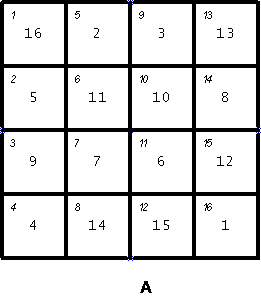
8

12

1

The expression A(14) simply extracts the 14th element of the implicit column vector. Indexing into a matrix with a single subscript in this way is often called *linear indexing*.

Here are the elements of the matrix A along with their linear indices:



The linear index of each element is shown in the upper left.

From the diagram you can see that A(14) is the same as A(2,4).

The single subscript can be a vector containing more than one linear index, as in:

A([6 12 15])

ans =

11 15 12

Consider again the problem of extracting just the (2,1), (3,2), and (4,4) elements of A. You can use linear indexing to extract those elements:

A([2 7 16])

ans =

5 7 1

That's easy to see for this example, but how do you compute linear indices in general? MATLAB provides a function called sub2ind that converts from row and column subscripts to linear indices. You can use it to extract the desired elements this way:

idx = sub2ind(size(A), [2 3 4], [1 2 4])

ans =

2 7 16

A(idx)

ans =

5 7 1

### Indexing with a Single Index

Another method for accessing elements of an array is to use only a single index, regardless of the size or dimensions of the array. This method is known as linear indexing. While MATLAB displays arrays according to their defined sizes and shapes, they are actually stored in memory as a single column of elements. A good way to visualize this concept is with a matrix. While the following array is displayed as a 3-by-3 matrix, MATLAB stores it as a single column made up of the columns of A appended one after the other. The stored vector contains the sequence of elements 12, 45, 33, 36, 29, 25, 91, 48, 11, and can be displayed using a single colon.

A = [12 36 91; 45 29 48; 33 25 11]

A = 3×3

12 36 91

45 29 48

33 25 11

Alinear = A(:)

Alinear = 9×1

12

45

33

36

29

25

91

48

11

For example, the 3,2 element of A is 25, and you can access it using the syntax A(3,2). You can also access this element using the syntax A(6), since 25 is sixth element of the stored vector sequence.

e = A(3,2)

e = 25

elinear = A(6)

elinear = 25

While linear indexing can be less intuitive visually, it can be powerful for performing certain computations that are not dependent on the size or shape of the array. For example, you can easily sum all of the elements of A without having to provide a second argument to the sum function.

s = sum(A(:))

s = 330

The [sub2ind](https://in.mathworks.com/help/matlab/ref/sub2ind.html) and [ind2sub](https://in.mathworks.com/help/matlab/ref/ind2sub.html) functions help to convert between original array indices and their linear version. For example, compute the linear index of the 3,2 element of A.

=============================

linearidx = sub2ind(size(A),3,2)

linearidx = 6

Convert from the linear index back to its row and column form.

[row,col] = ind2sub(size(A),6)

row = 3

col = 2

### Indexing with Logical Values

Using true and false logical indicators is another useful way to index into arrays, particularly when working with conditional statements. For example, say you want to know if the elements of a matrix A are less than the corresponding elements of another matrix B. The less-than operator returns a logical array whose elements are 1 when an element in A is smaller than the

corresponding element in B.

A = [1 2 6; 4 3 6]

A = 2×3

1 2 6

4 3 6

B = [0 3 7; 3 7 5]

B = 2×3

0 3 7

3 7 5

ind = A<B

ind = 2x3 logical array

0 1 1

0 1 0

Now that you know the locations of the elements meeting the condition, you can inspect the individual values using ind as the index array. MATLAB matches the locations of the value 1 in ind to the corresponding elements of A and B, and lists their values in a column vector.

Avals = A(ind)

Avals = 3×1

2

3

6

Bvals = B(ind)

Bvals = 3×1

3

7

7

## =======================================================FN

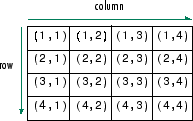
## Multidimensional Arrays

# Multidimensional Arrays

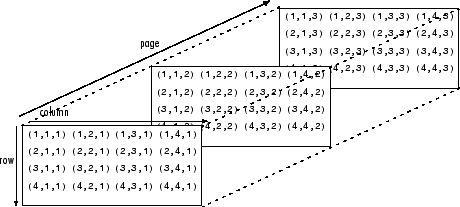
Open in MATLAB Online

[View MATLAB Command](matlab:openExample('matlab/nddemo'))

A multidimensional array in MATLAB® is an array with more than two dimensions. In a matrix, the two dimensions are represented by rows and columns.



Each element is defined by two subscripts, the row index and the column index. Multidimensional arrays are an extension of 2-D matrices and use additional subscripts for indexing. A 3-D array, for example, uses three subscripts. The first two are just like a matrix, but the third dimension represents pages or sheets of elements.



### Creating Multidimensional Arrays

You can create a multidimensional array by creating a 2-D matrix first, and then extending it. For example, first define a 3-by-3 matrix as the first page in a 3-D array.

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

A = 3×3

1 2 3

4 5 6

7 8 9

Now add a second page. To do this, assign another 3-by-3 matrix to the index value 2 in the third dimension. The syntax A(:,:,2) uses a colon in the first and second dimensions to include all rows and all columns from the right-hand side of the assignment.

A(:,:,2) = [10 11 12; 13 14 15; 16 17 18]

A =

A(:,:,1) =

1 2 3

4 5 6

7 8 9

A(:,:,2) =

10 11 12

13 14 15

16 17 18

The [cat](https://in.mathworks.com/help/matlab/ref/double.cat.html) function can be a useful tool for building multidimensional arrays. For example, create a new 3-D array B by concatenating A with a third page. The first argument indicates which dimension to concatenate along.

B = cat(3,A,[3 2 1; 0 9 8; 5 3 7])

B =

B(:,:,1) =

1 2 3

4 5 6

7 8 9

B(:,:,2) =

10 11 12

13 14 15

16 17 18

B(:,:,3) =

3 2 1

0 9 8

5 3 7

Another way to quickly expand a multidimensional array is by assigning a single element to an entire page. For example, add a fourth page to B that contains all zeros.

B(:,:,4) = 0

B =

B(:,:,1) =

1 2 3

4 5 6

7 8 9

B(:,:,2) =

10 11 12

13 14 15

16 17 18

B(:,:,3) =

3 2 1

0 9 8

5 3 7

B(:,:,4) =

0 0 0

0 0 0

0 0 0

### Accessing Elements

To access elements in a multidimensional array, use integer subscripts just as you would for vectors and matrices. For example, find the 1,2,2 element of A, which is in the first row, second column, and second page of A.

A

A =

A(:,:,1) =

1 2 3

4 5 6

7 8 9

A(:,:,2) =

10 11 12

13 14 15

16 17 18

elA = A(1,2,2)

elA = 11

Use the index vector [1 3] in the second dimension to access only the first and last columns of each page of A.

C = A(:,[1 3],:)

C =

C(:,:,1) =

1 3

4 6

7 9

C(:,:,2) =

10 12

13 15

16 18

To find the second and third rows of each page, use the colon operator to create your index vector.

D = A(2:3,:,:)

D =

D(:,:,1) =

4 5 6

7 8 9

D(:,:,2) =

13 14 15

16 17 18

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) =

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 −

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

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] % horizontal 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

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}

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

c =

{

[1,1] = Red

[2,1] = 1

[1,2] = Blue

[2,2] = 2

[1,3] = Green

[2,3] = 3

[1,4] = Yellow

[2,4] = 4

[1,5] = White

[2,5] = 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 =

{

[1,1] = Red

[2,1] = 1

[1,2] = Blue

[2,2] = 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

ans = Green

ans = Yellow

EXERCISE : 1

Understanding fundamentals

EXERCISE : 2

Vectors :

1. Create a vector from 1 to 10.

Extract and 3rd and 5th element from it.

1. Extract the elements from 2nd to 9th positions from the vector

1. Write a script file for the following :

A = [10, 14, 10, 20, 20];

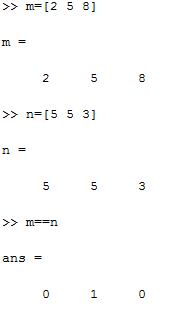
B = [2, 4, 2, 6, 2];

Apply Subtraction and division operators and store it in different variables.

1. Create 2 Vectors of size 6 elements

and apply logical operators ( < and > ) to know if they are equal.

Example :



1. Create a vector x=[1,2,4,6,8,3,9,10,12]
2. Replace the first 2 elements with maximum value of the vector.
3. Reshape this vector in to a 3X3 matrix
4. Use the MATLAB’s factorial function and for finding factorial of a Number and a vector.

Give inputs of your own choice.

1. Create a Matrix A - 4 X 4 Matrix of 4 pages ; and display that pagewise.
2. Apply linear indexing for the given matrix , Display the indices.

14 56 7

23 5 6

2 4 8

1. Generate the following row vector b= [ 1,2,3… 10]

Find the transpose of it.

1. If x=[1 4 ; 8 3]

Find (a) inverse of the matrix x

(b) diagonal of x

© sum of each column and sum of whole matrix x

(d) the transpose of x