Scan or click here for more resources

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.

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 –

Live Demo

```
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 –

Live Demo

```
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 –

Live Demo

```
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 –

Live Demo

```
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.

Live Demo

```
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.

Live Demo

```
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
```

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

Live Demo

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

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

```
a =

ans(:,:,1) =

   0   0   0
   0   0   0
   0   0   0

ans(:,:,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,

Live Demo

```
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
   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 –

Live Demo

```
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 –

Live Demo

```
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 –

Live Demo

```
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 –

Live Demo

```
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 –

Live Demo

```
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 –

Live Demo

```
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 –

Live Demo

```
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
```

Sparse Matrix Functions

This table display some of the functions most generally used when working with sparse matrices.

Function	Description
full	It converts a sparse matrix to a full matrix.
issparse	It determines if a matrix is sparse.
nnz	It return the number of nonzero matrix elements.
nonzeros	It returns the nonzero elements of a matrix.
nzmax	It returns the amount of storage allocated for nonzero elements.
spalloc	It allocate space for a sparse matrix.
sparse	It create a sparse matrix or converts full to sparse.
speye	It creates a sparse identity matrix.
sprand	It creates a sparse uniformly distributed random matrix.
sprandn	It creates a sparse normally distributed random matrix.
find	It finds indices and values of nonzero elements in a matrix.
spones	It replaces nonzero sparse matrix elements with ones.
spfun	It apply function to nonzero matrix elements
spy	It visualizes sparsity pattern as a plot.

MATLAB - Strings

Creating a character string is quite simple in MATLAB. In fact, we have used it many times. For example, you type the following in the command prompt –

Live Demo

```
my_string = 'Tutorials Point'
```

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

```
my_string = Tutorials Point
```

MATLAB considers all variables as arrays, and strings are considered as character arrays. Let us use the **whos** command to check the variable created above –

```
whos
```

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

```
Name          Size            Bytes  Class     Attributes
my_string     1x16               32  char
```

Interestingly, you can use numeric conversion functions like **uint8** or **uint16** to convert the characters in the string to their numeric codes. The **char** function converts the integer vector back to characters –

Example

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

Live Demo

```
my_string = 'Tutorial''s Point';
str_ascii = uint8(my_string)          % 8-bit ascii values
str_back_to_char= char(str_ascii)
str_16bit = uint16(my_string)         % 16-bit ascii values
str_back_to_char = char(str_16bit)
```

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

```
str_ascii =

   84  117  116  111  114  105   97  108   39  115   32   80  111  105  110  116

str_back_to_char = Tutorial's Point
str_16bit =

   84  117  116  111  114  105   97  108   39  115   32   80  111  105  110  116

str_back_to_char = Tutorial's Point
```

Rectangular Character Array

The strings we have discussed so far are one-dimensional character arrays; however, we need to store more than that. We need to store more dimensional textual data in our program. This is achieved by creating rectangular character arrays.

Simplest way of creating a rectangular character array is by concatenating two or more one-dimensional character arrays, either vertically or horizontally as required.

You can combine strings vertically in either of the following ways –

- Using the MATLAB concatenation operator **[]** and separating each row with a semicolon (;). Please note that in this method each row must contain the same number of characters. For strings with different lengths, you should pad with space characters as needed.
- Using the **char** function. If the strings are of different lengths, char pads the shorter strings with trailing blanks so that each row has the same number of characters.

Example

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

Live Demo

```
doc_profile = ['Zara Ali                                ';...
               'Sr. Surgeon                             ';...
               'R N Tagore Cardiology Research Center']
```

```
doc_profile = char('Zara Ali', 'Sr. Surgeon', ...
                   'RN Tagore Cardiology Research Center')
```

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

```
doc_profile =
Zara Ali
Sr. Surgeon
R N Tagore Cardiology Research Center
doc_profile =
Zara Ali
Sr. Surgeon
RN Tagore Cardiology Research Center
```

You can combine strings horizontally in either of the following ways –

- Using the MATLAB concatenation operator, **[]** and separating the input strings with a comma or a space. This method preserves any trailing spaces in the input arrays.
- Using the string concatenation function, **strcat**. This method removes trailing spaces in the inputs.

Example

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

Live Demo

```
name =      'Zara Ali                                    ';
position = 'Sr. Surgeon                                 ';
worksAt =  'R N Tagore Cardiology Research Center';
profile = [name ', ' position ', ' worksAt]
profile = strcat(name, ', ', position, ', ', worksAt)
```

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

```
profile = Zara Ali        , Sr. Surgeon        , R N Tagore Cardiology Research Center
profile = Zara Ali,Sr. Surgeon,R N Tagore Cardiology Research Center
```

Combining Strings into a Cell Array

From our previous discussion, it is clear that combining strings with different lengths could be a pain as all strings in the array has to be of the same length. We have used blank spaces at the end of strings to equalize their length.

However, a more efficient way to combine the strings is to convert the resulting array into a cell array.

MATLAB cell array can hold different sizes and types of data in an array. Cell arrays provide a more flexible way to store strings of varying length.

The **cellstr** function converts a character array into a cell array of strings.

Example

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

Live Demo

```
name =      'Zara Ali                                    ';
```

```
position = 'Sr. Surgeon                              ';
worksAt =  'R N Tagore Cardiology Research Center';
profile = char(name, position, worksAt);
profile = cellstr(profile);
disp(profile)
```

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

```
{
   [1,1] = Zara Ali
   [2,1] = Sr. Surgeon
   [3,1] = R N Tagore Cardiology Research Center
}
```

String Functions in MATLAB

MATLAB provides numerous string functions creating, combining, parsing, comparing and manipulating strings.

Following table provides brief description of the string functions in MATLAB –

Function	Purpose
Functions for storing text in character arrays, combine character arrays, etc.	
blanks	Create string of blank characters
cellstr	Create cell array of strings from character array
char	Convert to character array (string)
iscellstr	Determine whether input is cell array of strings
ischar	Determine whether item is character array
sprintf	Format data into string
strcat	Concatenate strings horizontally
strjoin	Join strings in cell array into single string
Functions for identifying parts of strings, find and replace substrings	
ischar	Determine whether item is character array
isletter	Array elements that are alphabetic letters
isspace	Array elements that are space characters
isstrprop	Determine whether string is of specified category
sscanf	Read formatted data from string

strfind	Find one string within another
strrep	Find and replace substring
strsplit	Split string at specified delimiter
strtok	Selected parts of string
validatestring	Check validity of text string
symvar	Determine symbolic variables in expression
regexp	Match regular expression (case sensitive)
regexpi	Match regular expression (case insensitive)
regexprep	Replace string using regular expression
regexptranslate	Translate string into regular expression
Functions for string comparison	
strcmp	Compare strings (case sensitive)
strcmpi	Compare strings (case insensitive)
strncmp	Compare first n characters of strings (case sensitive)
strncmpi	Compare first n characters of strings (case insensitive)
Functions for changing string to upper- or lowercase, creating or removing white space	
deblank	Strip trailing blanks from end of string
strtrim	Remove leading and trailing white space from string
lower	Convert string to lowercase
upper	Convert string to uppercase
strjust	Justify character array

Examples

The following examples illustrate some of the above-mentioned string functions –

Formatting Strings

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

Live Demo

```
A = pi*1000*ones(1,5);
sprintf(' %f \n %.2f \n %+.2f \n %12.2f \n %012.2f \n', A)
```

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

```
ans =  3141.592654
   3141.59
   +3141.59
      3141.59
   000003141.59
```

Joining Strings

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

Live Demo

```
%cell array of strings
str_array = {'red','blue','green', 'yellow', 'orange'};
% Join strings in cell array into single string
str1 = strjoin(str_array, "-")
str2 = strjoin(str_array, ",")
```

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

```
str1 = red-blue-green-yellow-orange
str2 = red,blue,green,yellow,orange
```

Finding and Replacing Strings

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

Live Demo

```
students = {'Zara Ali', 'Neha Bhatnagar', ...
            'Monica Malik', 'Madhu Gautam', ...
            'Madhu Sharma', 'Bhawna Sharma',...
            'Nuha Ali', 'Reva Dutta', ...
            'Sunaina Ali', 'Sofia Kabir'};
 % The strrep function searches and replaces sub-string.
new_student = strrep(students(8), 'Reva', 'Poulomi')% Display first names
first_names = strtok(students)
```

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

```
new_student =
{
   [1,1] = Poulomi Dutta
}
first_names =
{
   [1,1] = Zara
   [1,2] = Neha
   [1,3] = Monica
   [1,4] = Madhu
   [1,5] = Madhu
   [1,6] = Bhawna
   [1,7] = Nuha
   [1,8] = Reva
   [1,9] = Sunaina
   [1,10] = Sofia
```

```
}
```

Comparing Strings

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

Live Demo

```
str1 = 'This is test'
str2 = 'This is text'if (strcmp(str1, str2))
   sprintf('%s and %s are equal', str1, str2)else
   sprintf('%s and %s are not equal', str1, str2)end
```

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

```
str1 = This is test
str2 = This is text
ans = This is test and This is text are not equal
```