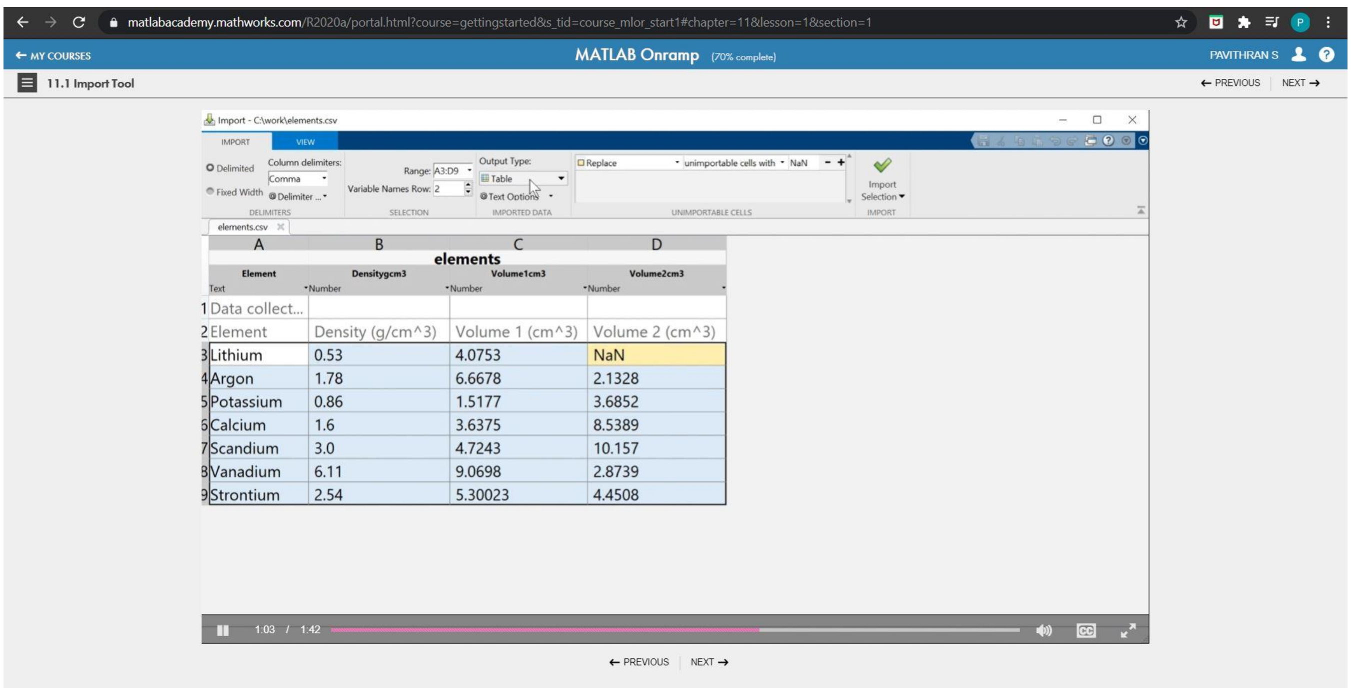
DAILY ASSESSMENT FORMAT

Date:	**08 JULY 2020**	**Name:**	**PAVITHRAN S**
Course:	**MATLAB**	**USN:**	**4AL17EC068**
Topic:	**MATLAB**	**Semester & Section:**	**6TH B**
Github Repository:	**Pavithran**		

FORENOON SESSION DETAILS
Image of session



Report – Report can be typed or hand written for up to two pages.

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 −

```
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
```

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 =

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,

```
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 −

```
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
```

Date:		**Name:**	**PAVITHRAN S**
Course:		**USN:**	**4AL17EC068**
Topic:		**Semester & Section:**	**6^{TH} B**
Github	**Pavithran**		

Repository:			

AFTERNOON SESSION DETAILS
Image of session

Report – Report can be typed or hand written for up to two pages.