Chapter 2 Tutorial 2

The purpose of this tutorial is to teach you about Scalars, and Arrays (Vectors and Matrices).

```
% Always clear workspace variables before a new tutorial or program.
clear
```

Edit the code below and update the variable named **`name`** with your name for this tutorial in the code below.

```
name="";
fprintf("Output for Tutorial_02_1 run by %s.\n", name)
```

```
Output for Tutorial_02_1 written by .
```

Scalars

Scalars are simply single values, whether it be a number or a character.

```
radius=6
```

```
radius = 6
```

```
height=12
```

```
height = 12
```

```
width=2
```

```
width = 2
```

Vectors

Vectors, or lists, are one-dimensional arrays and can be in the format of a row or a column vector.

Types

A row vector is made using commas or spaces between values enclosed by square brackets. The comma is optional, just simply a space will suffice however, commas appear more clear than just a space, comma and space makes it even more clear.

A column vector is made using semicolons between values enclosed by square brackets.

```
rowVector=[1, 2, 3, 4]  % A 1x4 row vector
```

```
rowVector = 1×4
     1     2     3     4
```

```
columnVector=[2;4;6;8]  % A 4x1 column vector
```

```
columnVector = 4×1
     2
     4
     6
     8
```

Transposing

A row vector can be converted to a column vector and vise-versa. This is easily done using the transpose operator (apostrophe).

```
rowVector'      % Convert the 1x4 rowVector to a 4x1 column vector
```

```
ans = 4×1
     1
     2
     3
     4
```

```
columnVector'   % Convert 4x1 columnVector to a 1x4 row vector
```

```
ans = 1×4
     2     4     6     8
```

Initializing/Declaring

You've already seen a few examples of declaring vectors. Here are a few more ways vectors can be generated.

```
% Create a row vector using existing vectors and hard coded scalars, notice
% the columnVector is converted to a row vector using the transpose operator.
metaRowVector=[rowVector, columnVector', 0, 5]
```

```
metaRowVector = 1×10
     1     2     3     4     2     4     6     8     0     5
```

Vectors can also be created using the colon operator as we've used before.

```
rowVector2=0:10:100 % Create a row vector using the colon operator
```

```
rowVector2 = 1×11
     0    10    20    30    40    50    60    70    80    90   100
```

```
rowVector3=-10:0    % We can also use negatives and/or decending order
```

```
rowVector3 = 1×11
   -10    -9    -8    -7    -6    -5    -4    -3    -2    -1     0
```

Recall that if you know the starting number, ending number and the increment value, you use colon notation `1:5:100` to declare a vector. The `linspace()` function can be used when the starting number, increment, and total number of values is known. In the case of `linspace()`, you don't know to know the ending number. That is, `linspace()` generates "linearly spaced" data.

```
% Create 6 values, starting at 0 and incrementing my pi/6
piVector=linspace(0,pi/6,6)
```

```
piVector = 1×6
         0    0.1047    0.2094    0.3142    0.4189    0.5236
```

Matrices

A matrix is a collection of values represented as a two-dimensional array. It can be easier to visualize the array during declaration by starting a new line after each semi-colon.

```
% A 3x4 matrix of floating point values
m = [ 3.0, 1.8, 3.6;
      4.6, -2.0, 21.3;
      0.0, -6.1, 12.8;
      2.3, 0.3, -6.1 ]
```

```
m = 4×3
    3.0000    1.8000    3.6000
    4.6000   -2.0000   21.3000
         0   -6.1000   12.8000
    2.3000    0.3000   -6.1000
```

Accessing Matrix and Vector Elements

The function notation `variable(row,col)` can be used to access subscripts, or, in this case a single element value in a matrix at a specific row and column position.

```
% Get the value from matrix at row 2, column 3
m(2,3)
```

```
ans = 42.8000
```

```
fprintf("The value at row 3, column 3 of the matrix m is %4.1f", m(3,3))
```

```
The value at row 3, column 3 of the matrix m is 12.8
```

Since vectors are simply 1-dimensional matrices, the same notation can be used to get the `nth` element in a vector.

```
fprintf("The second element in rowVector2 is %i", rowVector2(2))
```

```
The second element in rowVector2 is 10
```

```
fprintf("The second element in columnVector is %i", columnVector(2))
```

```
The second element in columnVector is 4
```

The previous notation showed examples of accessing a single value from a matrix or vector. We can also access multiple value subscripts of matrices.

```
rowVector2(2:4)  % access the second, through fourth, elements in rowVector2
```

```
ans = 1×3
    10    20    30
```

```
rowVector2(2:2:6) % Fancy, access elements at positions 2, 4, and 6
```

```
ans = 1×3
    10    30    50
```

Editing Matrix and Vector Values

If a value in a matrix or vector needs to be changed, we simply use the parenthesis notation as before when accessing the value, and use the equals sign to set it like any other variable.

```
m(2, 3) % Display the current value of m(2, 3)
```

```
ans = 21.3000
```

```
% Update the value at row 2, column 3 of the matrix m
m(2, 3)=42.8;
m(2, 3) % Display the updated value of m(2, 3)
```

```
ans = 42.8000
```

Additional Notes:

- Only integer values can be used to obtain a subscript of a matrix or vector
- Subscripts may be scalar (single value) or vectors (a collection of values)
- Many programming languages start with the first element at position 0, MATLAB subscripts always start at 1