Chapter 2 Tutorial 3

The purpose of this tutorial is to teach you about capturing output with the diary function. At the same time we'll cover some more matrix functions.

```
% 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_3 run by %s.\n", name)
```

```
Output for Tutorial_02_3 written by .
```

Diary

The `diary` command echoes all output sent to the command window into a file of your choosing. If the file does not exist, it will be created. If the file does exist, the diary will be continued from the end of the current file contents.

Note: The diary file will be created, however, due to the nature of matlab live scripts, nothing is printed to the command window so the file will be empty. In a normal script, everything from `diary <filename>` to `diary off` would be captured in the diary file.

```
fprintf("Output will also be sent to the file Tutorial_02_3_Output.txt")
```

```
Output will also be sent to the file Tutorial_02_3_Output.txt
```

```
diary Tutorial_02_3_Output.txt
```

More Ways to Create Matrices

Create a matrix manually

```
matrixA=[1 2 3 4; 5 6 7 8; 9 10 11 12]  % Again, we can use spaces or commas for row va
```

```
matrixA = 3×4
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

Create a matrix filled with ones

```
matrixB=ones(2,3)   % Create a 2 row, 3 column matrix of ones
```

```
matrixB = 2×3
     1     1     1
     1     1     1
```

```
matrixC=ones(4)     % Create a 4x4 matrix of ones
```

```
matrixC = 4×4
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

Create a matrix filled with zeros

```
matrixD=zeros(3,2)  % Create a 3 row, 2 column matrix of zeros
```

```
matrixD = 3×2
     0     0
     0     0
     0     0
```

Create the identity matrix; *more on this later in matrix arithmetic*

```
identityMatrix=eye(3)   % Create a 3 row, 3 column identity matrix
```

```
identityMatrix = 3×3
     1     0     0
     0     1     0
     0     0     1
```

More Ways to Access Matrix Elements

Access a single value subscript

```
matrixA(2,3)    % Get the value of matrixA element at row 2, column 3
```

```
ans = 7
```

Access an entire row or rows subscript, think of the : operator as representing *all*. We want *all* column values in row 3.

```
matrixA(3,:)    % Get the third row of matrixA
```

```
ans = 1×4
     9    10    11    12
```

Access an entire column or columns subscript. We want *all* row values in column 2

```
matrixA(:,2)    % Get the second column of matrixA
```

```
ans = 3×1
     2
     6
    10
```

Access the first and third columns in matrixA

```
matrixA(:,[1,3])    % By manually creating a vector with the column numbers we want
```

```
ans = 3×2
     1     3
     5     7
     9    11
```

```
matrixA(:,1:2:3)    % Not a great example but dynamically creating the vector works to
```

```
ans = 3×2
     1     3
     5     7
     9    11
```

Access columns 2 through 4 of matrixA

```
matrixA(:,2:4)
```

```
ans = 3×3
     2     3     4
     6     7     8
    10    11    12
```

Example

Let's say we want to create a trig table showing the Sine, Cosine, and Tangent values for a set of angles. Here's how we might go about doing that.

```
% What angles do we want to see the values for?
angles=[0:30:360]   % Angles 0 to 360 in increments of 30
```

```
angles = 1×13
     0    30    60    90   120   150   180   210   240   270   300   330   360
```

```

% Get the sine values
sineValues=sind(angles)     % sind for the sine using degrees
```

```
sineValues = 1×13
         0    0.5000    0.8660    1.0000    0.8660    0.5000         0   -0.5000 ⋯
```

```

% Get the cosine values
cosineValues=cosd(angles)   % cosd for the cosine using degrees
```

```
cosineValues = 1×13
    1.0000    0.8660    0.5000         0   -0.5000   -0.8660   -1.0000   -0.8660 ⋯
```

```

% Get the tangent values
tangentValues=tand(angles)  % tand for the tangent using degrees
```

```
tangentValues = 1×13
         0    0.5774    1.7321       Inf   -1.7321   -0.5774         0    0.5774 ⋯
```

```

% Put all of the vectors into a table.
% Notice we're transposing rows to columns using the transpose ' operator.
trigTable=[angles', sineValues', cosineValues', tangentValues'];

% Display the trig table (disp is a nicer way of displaying a matrix)
disp(' Angle(Deg)   Sine     Cosine    Tangent')
```

```
 Angle(Deg)   Sine     Cosine    Tangent
```

```
disp(trigTable)
```

```
         0         0    1.0000         0
   30.0000    0.5000    0.8660    0.5774
   60.0000    0.8660    0.5000    1.7321
   90.0000    1.0000         0       Inf
  120.0000    0.8660   -0.5000   -1.7321
  150.0000    0.5000   -0.8660   -0.5774
  180.0000         0   -1.0000         0
  210.0000   -0.5000   -0.8660    0.5774
  240.0000   -0.8660   -0.5000    1.7321
  270.0000   -1.0000         0      -Inf
  300.0000   -0.8660    0.5000   -1.7321
  330.0000   -0.5000    0.8660   -0.5774
  360.0000         0    1.0000         0
```

TIP: Once you better understand software programming, you can start to "nest" things, here is an example of a much more concise way of solving that same example problem above. Notice that, since we don't need to save the results of the trig functions, we don't have to put them in a variable first, we can compute and transpose them right in place.

```
% What angles do we want to see the values for?
angles=[0:30:360];   % Angles 0 to 360 in increments of 30

% Display the trig table (disp is a nicer way of displaying a matrix)
disp(' Angle(Deg)   Sine     Cosine    Tangent')
```

```
 Angle(Deg)   Sine     Cosine    Tangent
```

```
disp([angles', sind(angles)', cosd(angles)', tand(angles)'])
```

```
         0         0    1.0000         0
   30.0000    0.5000    0.8660    0.5774
   60.0000    0.8660    0.5000    1.7321
   90.0000    1.0000         0       Inf
  120.0000    0.8660   -0.5000   -1.7321
  150.0000    0.5000   -0.8660   -0.5774
  180.0000         0   -1.0000         0
  210.0000   -0.5000   -0.8660    0.5774
  240.0000   -0.8660   -0.5000    1.7321
  270.0000   -1.0000         0      -Inf
  300.0000   -0.8660    0.5000   -1.7321
  330.0000   -0.5000    0.8660   -0.5774
  360.0000         0    1.0000         0
```

Diary

When you've finished with your diary output, it's important to turn it back off.

```
diary off
```

Additional Notes:

- Don't forget to turn the `diary` off at the end of your program.