Formatting:

%f = fixed-point, or decimal
%e = scientific notation
%g = fixed-point or exponential with no trailing zeros
%i = integer
%c = single character
%s = string of characters

The general form of the **fprintf**( ) function is the following:

fprintf(string with format commands, variable and array names)

Example:

```
score = [33 44 55];
fprintf('Score 1 is %f. \n' , score(1))
fprintf('Score 2 is %f and score 3 is %f. \n' , score(2) , score(3) )
```

Output:

```
Score 1 is 33.000000.
Score 2 is 44.000000 and score 3 is 55.000000.
```

Recycling format commands:

If you do not provide enough format commands, old format commands will be recycled.

Example:

```
score = [33 44 55];
for i=1:numel(score)
  fprintf('Score #%i is %e. \n' , i, score(i) )
end
```

Output:

```
Score #1 is 3.300000e+001.
Score #2 is 4.400000e+001.
Score #3 is 5.500000e+001.
```

Controlling precision of data:

The general form of this formatting option is the following:

%X.Yf

where **X** is the total number of spaces for displaying the variable's value and **Y** is the amount of numbers to display right of the decimal. Keep in mind that the decimal point takes up one space.

%X.Ye = scientific notation format
%Xi = integer
%Xc = single character
%Xs = string of characters

Example:

```
score = [33 44 55];
fprintf('Score 1 is %7.3f. \n' , score(1))
fprintf('Score 2 is %5.1f and score 3 is %12.3f. \n' , score(2) , score(3) )
```

Output:

```
Score 1 is  33.000.
Score 2 is  44.0 and score 3 is       55.000.
```

Multidimensional arrays:

If you try to print multiple arrays you may run into trouble. Using the function $y = x^2$ I would like to print off all **x**-values in the first column, all the corresponding **y**-values in the second column.

M-file:

```
x = (0:1:5);
y = x.^2;
fprintf('%4.1f  %6.1f \n' , x , y)
```

Output:

```
 0.0    1.0
 2.0    3.0
 4.0    5.0
 0.0    1.0
 4.0    9.0
16.0   25.0
```

Whoops. This didn't work. All the **x** values printed first, then the **y** values. Instead, we combine **x** and **y** into a single array and then print that. Remember, multi-dimensional arrays will be printed column by column.

M-file:

```
x = (0:1:5);
y = x.^2;
tablexy = [x;y];
fprintf('%4.1f  %6.1f \n' , tablexy)
```

Output:

```
 0.0    0.0
 1.0    1.0
 2.0    4.0
 3.0    9.0
 4.0   16.0
 5.0   25.0
```

Writing to a file:

```
x = (0:1:5);
y = x.^2;
tablexy = [x;y];
file1 = fopen('funfile.txt', 'w')
fprintf(file1, '%4.1f  %6.1f \n' , tablexy)
fclose(file1)
```

Inside 'funfile.txt':

```
 0.0    0.0
 1.0    1.0
 2.0    4.0
 3.0    9.0
 4.0   16.0
 5.0   25.0
```

Reading from a file:

Using the code from the previous example, let's open up funfile.txt, read in the data, and display the data on the screen.

```
% From the previous example:
x = (0:1:5);
y = x.^2;
tablexy = [x;y];
file1 = fopen('funfile.txt', 'w');
fprintf(file1, '%4.1f  %6.1f \n' , tablexy)
fclose(file1);

% now we read in the data
file2 = fopen('funfile.txt');
% This will read the x and y values until the file is over
% (2 columns and an infinite number of rows... unless end of file is reached)
% [2,6] would do the same thing as [2,inf] in this case.
A = fscanf(file2, '%f' , [2, inf]);
fclose(file2);

% Notice that the data is transposed when stored in A.
% fscanf( ) reads data in COLUMN ORDER, similar to how fprintf( ) writes in column order.
disp(A)

% However, this format is perfect for fprintf since it prints column by column
fprintf('%7.2f %7.2f \n', A)
```