A Tutorial for Matlab®

fprintf

fprintf

Matlab® provides many methods to output data, of which Table 1 shows the most used.

Table 1. Output Methods

Method	*Result*	*Description*
x	x = ValueOfX	By not adding a semi-colon, Matlab outputs the variable x and its value to the console
disp(x)	ValueOfX	Matlab only displays the value of the variable on the console
sprintf('%g is the value', x)	ValueOfX is the value	Matlab outputs the value of the variable x to the console in the format described
fprintf(ID, '%g is the value', x)	ValueOfX is the value	Same as sprintf except Matlab outputs the to the ID instead of the console

It is clear that by displaying to an ID of the user's choice while also permitting formatting, the most robust of these methods is fprintf. For this reason, fprintf will be the only output formatting method described. This tutorial begins with the output ID, continues to discuss outputting values, then covers basic formatting and concludes with advanced formatting.

Output Identification

The fprintf function requires a numerical output ID for its first argument. There are a couple options for getting an ID. If output to a particular file is desired, the output ID could be taken from an fopen statement (to write to file: for Unix use 'w' ; for pc use 'wt'):

```
ID = fopen('z:\\Output.txt','wt'); %open Output.txt and wt, write (PC)
fprintf(ID, 'Look! I wrote to a file!'); %write something to it
fclose(ID); %close the output file
```

But fprintf also allows user input for default ID's. If the user wants to test the code in the console before outputting, set the ID to 1. Now it does the same as sprintf except it allows the user to easily switch the ID to a file should desired output location be changed:

```
ID = 1; %The ID of the console
fprintf(ID, 'Look! I wrote to the screen!'); %write something to it
%fclose(ID); %close the output file – comment out since not applicable
```

NOTE: This tutorial will use ID variable = 1; use either file or console output as convenient.

Outputting Variables

Just as a value has a variable to call it, fprintf uses a conversion variable. The % specifies the beginning of a conversion into a string. A letter ends the conversion sequence and defines the output notation. Table 2 shows the most common *conversion characters*:

Table 2. Conversion Characters

Method	*fprintf(ID,'Method',exp(1))*	*Description*
%f	2.718282	Fixed decimal point
%e or %E	2.718282e+000 or 2.718282E+000	Scientific notation. e or E specifies capitalization

Table 2. Conversion Characters cont'd

Method	*fprintf(ID,'Method',exp(1))*	*Description*
%g or %G	2.71828 or 2.71828	Cross between %f and %e without unnecessary zeros
%s	2.718282e+000	String of characters (numbers behave as %e in v6.5)

For a complete list of Conversion Characters, type `doc fprintf` in the Matlab console

Output with fprintf requires the function, itself, with the arguments of the output ID, conversion character and value. Example 1, below, illustrates the ease of outputting variables

Example 1. Ease of Output

M-File	*Console*

`ID = 1;`
`x = 1234.5678;`
`fprintf(ID,'%f',x);`
	`1234.567800`

BASIC FORMATTING

Sometimes the ability to output a single value is adequate, but the ability to describe values and the ability to output entire matrices and arrays are more useful. Consider fprintf:

```
fprintf(ID, 'Format text and conversion characters', variable matrix);
```

The second argument is not limited to conversion characters. In fact, it doesn't need to have one at all! Consider the classic Hello World programming question in Example 2:

Example 2. "Hello World"

M-File	*Console*
`ID = 1;` `fprintf(ID,'Hello World!');`	`Hello World!`

This illustrates the ability to describe the values outputted; try Example 3 for such output:

Example 3. Describing the Output

M-File	*Console*

`ID = 1;`
`x = pi*(1/2)^2;`
`fprintf(ID,'The area of a 1 diameter circle is %g',x);`
	`The area of a 1 diameter circle is 0.785398`

As the usefulness of descriptions is quickly realized, the convenience of quotation marks, include line breaks, etc. is wanted. This is done with escape characters like those in Table 3:

Table 3. Escape Characters

Method	*Result*	*Description*
\n		Creates a new line (similar to pressing the enter key)
\\	\	Two single backward slashes output a single backward slash
''	'	Two single quotation marks output a single quotation mark
%%	%	Two single percent signs output a single percent sign.

For a complete list of Escape Characters, type `doc fprintf` in the Matlab console

Example 4 shows the implementation of some of the escape characters:

Example 4. Escape Characters

M-File	*Console*

`ID = 1;`
`x = 97.5;`
`fprintf(ID,'It ''works'' %g %% of the time\n',x);`
	`It 'works' 97.5 % of the time`

Having now acquired the ability to describe a value for a single value, try utilizing the whole variable matrix. But first, start small with a list. For each value to be read out, there must be a corresponding conversion character (think parallel structure from English class). Example 5 demonstrates this parallel structure in a list of 3 terms:

Example 5. Parallel Structure of List Output

M-File	*Console*

`ID = 1;`
`x1 = 10;`
`x2 = 2;`
`x3 = x1-x2;`
`fprintf(ID,'Difference of %g and %g is %g \n',[x1 x2 x3]);`
	`Difference of 10 and 2 is 8`

This is not just a list but is similar to an array. The array is treated like the list, the next conversion character corresponds with the next array item as in Example 6. Note how multiple types of conversion characters may be used in the same fprintf statement.

Example 6. Arrays within the Variable Matrix

M-File	*Console*

`ID = 1;`
`string = 'integers';`
`array = [1 2 3 4];`
`fprintf(ID,'%g, %g, %g and %g are %s \n', [array string]);`
	`1, 2, 3 and 4 are integers`

An array is simply a column matrix. For a multi-column matrix, fprintf displays each column individually per row. Transpose the matrix to get the format as shown in Example 7:

Example 7. Outputting a Matrix

M-File	*Console*

`ID = 1;`
`matrix = [1 2 3; 4 5 6; 7 8 9];`
`fprintf(ID,'+---+---+---+\n');`
`fprintf(ID,'| %g | %g | %g |\n+---+---+---+\n', matrix');`
	`+---+---+---+`
	`| 1 | 2 | 3 |`
	`+---+---+---+`
	`| 4 | 5 | 6 |`
	`+---+---+---+`
	`| 7 | 8 | 9 |`
	`+---+---+---+`

ADVANCED FORMATTING

This is taking basic formatting and making it look good. This means proper padding, proper alignment, and proper precision. These are accomplished via the conversion method.

Figure 1 illustrates the complete character conversion method:

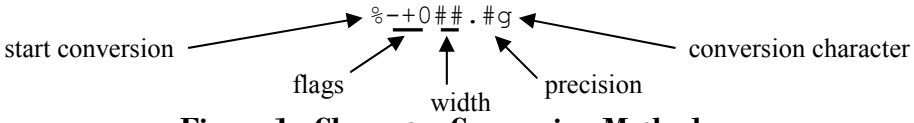

Figure 1. Character Conversion Method

The flags, width, and precision are optional and may be used alone or in tandem. For now, begin with the width. The width is the *minimum* size (in characters) of the string output for a value including exponential and negative signs, and decimals. This means if X is a space:

```
%4g for 1     would result in XXX1
%4g for 1000  would result in 1000
%4g for 10000 would result in 10000
```

It can be seen that consideration of the magnitude of the value is important so that the output lines up. Another part of alignment is selecting the justification. By default output is right justified. The - flag used in figure 1 left justifies it. In order to see a difference the width must be specified and must exceed the size of the string output for the value.

```
%-4g for 1     would result in 1XXX
%-4g for 1000  would result in 1000
%-4g for 10000 would result in 10000
```

The use of X's here is to illustrate a space, which is the default padding. A zero may also pad by placing a 0 flag before the width.

```
%04g for 1 would result in 0001
```

Sometimes it is desirable to always have the sign displayed. A + flag is used to do this.

```
%+04g for 10 would result in +010
%+04g for -1 would result in -001
```

The precision is the number of decimal places printed. This number of decimals is added to the string output size, so don't neglect the size of the decimal in the width specification.

```
%05.2f for 15.02 would result in 15.02
%05.2f for 1.003 would result in 01.00
```

That is all there is to advanced formatting except counting out the spaces to make sure they all line up. Try Example 8 below. It is color coded to denote the origin of each character.

Example 7. Example 5 Redone to Accommodate Different String Output Sizes

M-File	*Console*

`ID = 1;`
`matrix = [1.12 2.27 3.14; 4.45`
`5.56 6.67; 17.90 8.45 109.90];`
 	
`fprintf(ID,'+--------+--------`
`+--------+\n');`
 	
`fprintf(ID,'| %6.2f | %6.2f |`
`%6.2f |\n+--------+--------`
`+--------+\n', matrix');`
	`+--------+--------+--------+`
	`|  1.12 |  2.27 |  3.14 |`
	`+--------+--------+--------+`
	`|  4.45 |  5.56 |  6.67 |`
	`+--------+--------+--------+`
	`| 17.90 |  8.45 | 109.90 |`
	`+--------+--------+--------+`

