Output in MATLAB: disp

The easiest way to display the results of calculations is to use `disp`:

```
help disp
```



```
 DISP Display array.
 DISP(X) displays the array, without printing the array name.  In
 all other ways it's the same as leaving the semicolon off an
 expression except that empty arrays don't display.

 If X is a string, the text is displayed.

 See also INT2STR, NUM2STR, SPRINTF, RATS, FORMAT.
```


but it's restrictive in that it takes only a matrix as input and only uses default format to display matrix entries

Strings

Strings are row vectors of characters; use single quotes always!


```
clc; s = ' Avogadro''s number        % why 2 '' ?
disp(s)     % disp works for strings as well as arrays
```



```
s =

 Avogadro's number

 Avogadro's number
```

Arrays of strings

```
s2 = ['This is a long and boring string that is' ...
    ' in fact so long, it must be written on two lines'];
disp(s2)
```



```
This is a long and boring string that is in fact so long, it must be
```


To mix strings and variables, need to convert variable values to strings

```
NAv = 6.023e23;
disp([s  '  ' num2str(NAv,4)])  % an array of 3 strings
```

```
 Avogadro's number  6.023e+23
```

Example of disp

```
clc;
x=1:5; disp(x);
y=x*pi;disp(y);
disp(['The answer is  ' num2str(sqrt(5))])
```


Here we create an array of strings
and need the function num2str to convert the number sqrt(5) to a string

```
     1     2     3     4     5

    3.1416    6.2832    9.4248   12.5664   15.7080

The answer is  2.2361
```

long format

The default format prints about 4 figures after the decimal point

```
clc;disp(y)
```

```
    3.1416    6.2832    9.4248   12.5664   15.7080
```


If you want more figures displayed, use format long

```
format long ;
disp(y)
```

```
  Columns 1 through 3

   3.14159265358979   6.28318530717958   9.42477796076938

  Columns 4 through 5

  12.56637061435917  15.70796326794897
```

Default format

This doesn't change any computations, just the display.

To go back, use format short or just format (the default)


```
format short
disp(y)
```



```
    3.1416    6.2832    9.4248   12.5664   15.7080
```

Scientific format

If the numbers are likely to be large or small, use scientific format or e format

```
format
x = [4/3 1.2345e-6]
disp(x)

format short e
disp(x)
```



```
x =

    1.3333    0.0000

    1.3333    0.0000

   1.3333e+00   1.2345e-06
```

or format long e


```
format long e
disp(x)
```



```
1.333333333333333e+00   1.234500000000000e-06
```

Formatted output: fprintf

To have more control over output format, use `fprintf` (stolen from C) means 'formatted print to file' .
The related command `sprintf` formats in the same way but saves the result as a string variable.

The simplest version :


```
x=[45 pi pi];
fprintf('\n %10d %6.4f %12.7e\n',x)
```



```
        45 3.1416 3.1415927e+00
```

```
fprintf('format string',list of values to be printed)
```

The format string specifies how each value is to be printed (to screen). It includes: a conversion specification for each printed value, plus possibly escape sequences to represent nonprinting characters. You can include any text you want in the format string.

Format string

A conversion specification has the form:

- %wd for integers (d for decimal) where w = field width. If w < number of digits, it is ignored. w = minimum number of spaces used to print the integer value
- %w.pf or %w.pe for float variables where p is the number of digits after the decimal point. f is for fixed point format , e for exponential format. If p is not given, it defaults to 6.

Common escape sequences are :
\n newline
\t tab
" to print '
%% to print %

Use d format for integers


```
life = 42;
fprintf('The answer to the meaning of life is %4d\n',life);
```



```
The answer to the meaning of life is   42
```

Making a table of output

Armed with fprintf we can now make a nice output table

```
format
out = NewtonFuncOutvec(2,4)  % uses disp

y = NewtonFuncFormattedTable(2,4);

out =

    2.0000    2.2500    2.2361    2.2361    2.2361


  n     iterate    |xn-x_nm1|     residual
 1     2.25000000   2.500000e-01   6.250000e-02
 2     2.23611111   1.388889e-02   1.929012e-04
 3     2.23606798   4.313320e-05   1.860471e-09
 4     2.23606798   4.160139e-10   8.881784e-16
```

Functions

We use functions to break up a programming task into smaller chunks, and to re-use code with the same purpose but acting on different data.

We usually have a 'main' program (which can be a MATLAB function or a script file) that sets up the problem, reads in data if required, calls other functions to do the calculations, then postprocesses the data for output e.g. plotting.

Let's run the following program

```
swapMain;
```

then understand it using the debugger.

8 7 9

8 9 7

Function scope

When designing programs, you need to understand which variables are visible by which functions. This is called variable scope.

In working at the command prompt >> you are creating variables in the base workspace. Any script file also uses the base workspace, hence the danger if everything uses the same workspace.


```
clear
A = rand(50); b = size(A); y = sum(b);
whos
```

```
  Name        Size                Bytes  Class     Attributes

  A          50x50                20000  double
  b           1x2                    16  double
  y           1x1                     8  double
```

Local variables

```
swapMain
whos
```

```
     8     7
     8     9
```



```
  Name         Size                Bytes  Class     Attributes

  A           50x50                20000  double
  b             1x2                   16  double
  y             1x1                    8  double
```

Any variables created inside a function are local to that function and are stored in a workspace local to that function. They are not seen by the base workspace and disappear when the function exits/returns.
Moral: pass information in and out of functions with argument lists

End of Lecture 2