

## Chapter 1

### **Topic 1 – What is programming?**

Programming is telling a computer what to do. There are only 2 tricky things

- 1) Computers are very stupid – you have to tell them exactly what to do
- 2) Computers don't speak English. Any programming language is a compromise between the computer's native language (0010001) and your native language (English). High level programming languages are closer to English, low level languages are closer to computer-ese. The closer a language is to English, the easier it tends to be to program, but the slower it is for the computer to interpret it and the more constrained it is on what it can do. Low level languages tend to be hard to program but very fast to run, and less constrained. Matlab is a mid- to high level language.

Like any language, programming languages have grammar. Like real languages some things are easier to say in one language rather than another (Italian is the language of love etc. etc.). Some languages are better for computations, others for graphics. Unlike people who speak real languages (with the exception of the French) computers are very fussy about grammatical errors. Like learning a real language, at first it will be hard to say the simplest thing, but it will get easier, fast.

#### **Hardware**

Hardware is the physical presence of the computer. The monitor, the hard drive, the CPU (central processing unit, i.e. the “brain”). Hardware is anything you can damage by poking it with a screwdriver.

#### **Software**

Software = programs. Programs are instructions to your computer to behave in a particular way. So a software program like Microsoft Office gets all machines to behave in a particular way – the instructions are slightly different for different computers (different hardware), but the program makes all computers behave (almost) the same.

All software is written in a programming language. Some programs, (like Matlab) are there to help you write new programs. Matlab will run on Macs, PC, and UNIX. But some of the commands only run on some computers.

**NOTE** – these days it is almost impossible to damage your computer by writing a program. The computer usually makes it very hard for you to do anything that will damage it. In this book you won't be using any commands that can do any permanent damage. So crash your computer hard. Have fun!

## Getting Started

You need:

1. A PC running Windows or a Mac computer running OSX.

**WARNING:** This book is not designed to be compatible with Macs running Mac OS9 or earlier. Most of the book will work with Mac OS9 anyway, but if you are using Mac OS9 then where we differentiate in the code between PCs and Mac, pretend you are a PC.

2. You need to install Matlab on your computer. The student version is fine. You may run into license manager problems installing Matlab Mac OSX since it's really confusing and badly designed – just call Mathworks' technical support number and they will sort you out.

Now, start Matlab. A window will appear that's divided into a number of sub-windows. Close all the sub-windows except the one that called the "command window" which has a little prompt `>>`. Later you may find these other windows helpful.

OK – so where there's a line with a `>>` that means I want you to type the text at the prompt in the command window. Where there isn't the `>>` then the text should match what the command window is spitting out at you.

```
>>str1='I have no clue  
what I am doing'  
You just told the computer to create a list  
of letters 'I have no idea what I'm doing'  
and to name that list of letters str1. str1 is  
a variable - The single quotes tell the  
computer that str1 is a list of letters (not  
numbers, more on that later). Any list of  
letters is called a string. Now type:
```

```
>>who
```

This command asks your computer to give you a list of all the variables you have. At the moment the only variable you have is str1.

Typing the name of a variable asks your computer to tell you what is contained within that variable.

```
>>str1
```

The computer should show you what's in str1

```
str1 =
```

```
I have no clue what I am doing
```

Using `disp` also displays what a variable is, but only shows the contents, instead of repeating the name of the variable

```
>>disp(str1)
```

Text that looks like this is stuff that is happening in the command window. Either stuff you are typing in at the prompt, or stuff that the command window is spitting back out.

Text that looks like this is me telling you what's happening in Matlab for a particular command.

Text that looks like this is me giving you a general overview.

## Chapter 1 - 3 - 4/18/2007

So all the computer does is display the contents of the variable as follows:

```
I have no clue what I am doing
```

```
>>str2='Is it all going to be boring?';
```

```
>>str2='Is it all going to be boring?'
```

The first time you typed this you added a semi-colon at the end. The second time you didn't. The semi-colon tells the computer whether or not you want it to display the output of each command.

```
>>who
```

Now you have both str1 and str2

```
>>str1='I still have no clue what I am doing'
```

Now you are re-defining str1 by making it represent a slightly different list of letters

```
>>str1
```

See - the list of letters contained within str1 has changed

```
>>str1(3)
```

You've asked the computer to display the third letter in str1. This is called *indexing* or *subscripting*. 3 is an index (or subscript) into the third character in str1. Now try:

```
>>str(6)
```

You can see from this that the computer is counting spaces

```
>>mixstr=str1;
```

Now you've created a new variable called mixstr. You've told the computer to make mixstr the same as str1

```
>>mixstr
```

See, they are exactly the same

```
>>str
```

```
>>mixstr(3)=str1(1);
```

Now you are telling the computer to make the 3<sup>rd</sup> letter in mixstr the same as the 1<sup>st</sup> letter in str1

```
>>mixstr
```

```
>>mixstr(1)=str1(3);
```

Now make the 1st letter in mixstr the same as the 3rd letter in str1

```
>>mixstr
```

You should now have:

```
'h Iave no idea what I am doing'
```

You can also create lists of numbers. These are called *arrays* or *vectors*. Here's four different ways of creating a vector list that goes from 2 to 9 in steps of 1. A *variable* is a generic term that can be used to describe a *character* (a single letter), a *string* (a string of characters), a *double* (a single number, more on that later), a *vector*, or a *matrix* (a two or more dimensional set of numbers, more on that later too), as well as some other funky things we'll get to near the end of this book.

```
>>array1=[2 3 4 5 6 7 8 9]
```

```
>>array1=linspace(2, 9, 8)
```

Here you are saying you want a list of 8 numbers that are evenly spaced between 2 and 9. You can imagine that this command would be useful if you had collected 8 pieces of data evenly spaced between two and 9 seconds.

```
>>array1=2:1:9
```

Here you are saying that you want a list of numbers that goes from 2 to 9 with a step-size between each number of 1. You can imagine that this command would be useful if you collected data that went from 2 seconds, to 9 seconds, and had collected data every second.

```
>>array1=2:9
```

Matlab assumes a default step-size of 1, so you can simply skip it for this particular way of creating vectors.

Here are the three ways of creating a list of numbers that goes from 1 to 17 in steps of two.

```
>>array2=[1 3 5 7 9 11 13 15 17]
```

```
>>array2=linspace(1, 17, 9)
```

```
>>array2=1:2:17
```

```
>>array3=5:10
```

Remember, there is a convention that if you don't specify the step size, then the list of numbers goes up in steps of 1

You can also index vectors. 2 indexes the second integer in array3

```
>>array3(2)
```

You can also index more than one number in an array or string.

```
>>array3(2:4)
```

```
>>disp(array3)
```

disp can also be used to display numbers

By now you should be getting a little irritated with having to type in every command one at a time. We are therefore going to create a **program** – a program is simply a document containing a sequence of commands.

In Matlab programs are written in documents called **m-files**. So now we are going to put the commands you just did in a m-file.

## Creating an m-file

Make sure the command window is at the front. Now go to the menu bar and choose File->New->M-file.

You'll get a blank document in a new editor window.

Every program begins with a few lines of documentation. This is called a **header**. Good headers contain the following information

- 1) The name of the program
- 2) A description of what it does
- 3) Who wrote it, and when

```
% MixStrings.m
%
% Replaces 'this is gibberish' with a
% string of xxxxx's
%
% written by IF & GMB 4/2005

% because there is a empty line with no % above it,
% this line isn't part of the header
```

OK, type this header into your m-file.

Make sure every new line begins with a %. The % tells the computer to ignore that line – these comments aren't for the computer, they're for you. Commented text will probably show up as being green.

Headers are important. You may think that you will remember the programs you write – but trust me, you won't! Getting in the habit of having good up-to-date headers is like flossing – it's boring but it will save you a lot of pain in the long run.

Now we need to save the file. Make a folder called “MatlabClass” somewhere. Create a subfolder called “Misc”. **Don't** put these folders inside the Matlab application folder or you will lose everything if you reinstall Matlab.

Save the file as MixStrings.m (the same as the header) in the “Misc” folder.

Now go back to the command window and type

```
>>help MixStrings  
You will almost certainly get an error message:
```

```
MixStrings.m not found.
```

When the computer says that a file is “not found” that means the computer can't find an m-file that has that particular name. The reason the computer can't find the file even though you saved it in the folder ‘Misc’ is because the computer is only allowed to look for files in certain places.

One place that the computer always looks for files is the *current directory* or *working directory*. In fact this is the first place that the computer looks. When Matlab opens it automatically links to a particular folder (the default setting is made by Matlab). If you don't tell it otherwise it will save files to that folder. You can see what folder Matlab thinks is the current directory using the *print working directory* command:

```
>>pwd
```

The other places that the computer can find files are in folders that are in Matlab's *search path*. This *path* is simply a list of folders that the computer is allowed to look in whenever it is trying to find a file. Again Matlab comes with a default set of paths. You can get a list of the current folders in the path very easily:

```
>>path
```

To tell the computer where to look, you *set the path* ...

### **Setting the path via the menu bar**

Make sure the command window is at the front, and go to:

File->Set Path in the menu bar. A pop-up window will appear  
Choose “Add With Subfolders”, choose the “Misc” folder, and click  
OK. Then choose Save and Close.

Now type  
>> help MixStrings  
You should see the following information.

```
MixStrings.m
```

```
Replaces 'this is gibberish' with a
string of xxxxx's
```

written by IF & GMB 4/2005

Hopefully this will look a little familiar? It's the header you wrote.

The command `help` tells the computer to display the header you wrote. That's why you always need to write headers for every m-file that describe clearly what the program does and how to use it. The header will also help you find the right program when you can't remember its name – I'll show you how that works in just a second.

Try typing the following and read the headers.

```
>>help round
>>help pause
>>help length
>>help path
>>help addpath
>>doc round
```

`doc` is like `help` but provides more information.

But to use `help` or `doc` you need to know the name of the command you are looking for.

The solution to this problem is `lookfor`

```
>>lookfor gibberish
```

`lookfor` finds all the m-files that contain a particular word in their headers. For example, try:

```
>>lookfor string
>>lookfor random
```

An important thing to remember about setting your path is that if there are two files with the same name, and your path allows the computer to see both of them you won't get a warning, the computer will just use the first one it finds!

You can find out which file the computer is using by typing

```
>>which MixStrings
```

Matlab should spit out something like the following depending on where you put `MixStrings.m` on your computer.

```
C:\WORK\Matlab class &
book\MatlabBookOrganization\MatlabClass\Misc\MixStrings.m
```

You only have one version of `MixStrings`. If you had multiple versions Matlab would print out the path for each version that was within the path. The one at the top of the list is the version that Matlab will use by default at that current moment.

But be careful – which version that is used depends on what directory is the current working directory for Matlab. If the current directory changes it is possible that the version of `MixStrings` that is used will also change. This can lead to some really weird **bugs** (a bug is any time a program doesn't do what you want it to do and you have no idea why). One really confusing thing that can happen if more than one file of the same name is in the path is that you can make changes to a file, but the changes don't affect what the computer does. What's happening is that the computer is not actually using the file that you are changing – it is using a different file of the same name somewhere else in the path.

So you need to be careful about two things:

1) Don't be sloppy about having multiple m-files with the same name sitting in different directories

2) Be careful about your path.

Make sure that old directories with out-of-date files in them aren't still in your path. One good technique for path management is not to simply put folders in your default path so they are permanently in the Matlab path. Instead, when you write a program you simply add the paths you will need for that program at the beginning of the program using Matlab commands (instead of the Menu bar). This technique lets you have different paths depending on which programs you are running, instead of having one mega-path. Using this technique to manage your path minimizes the probability of having out-of-date folders in your path.

## Changing the path within Matlab

Play with the following commands and use them to move to your MatlabClass/Misc directory.

```
>>pwd
```

This tells you the current directory

```
>>cd ..
```

Moves up one directory

```
>>ls
```

Lists the files in that directory. You can move to any of the files listed in that directory:

```
>>cd Misc
```

Moves to folder Misc (provided you are in a folder that contained the subfolder Misc). You should be able use these commands to navigate to your Misc folder. Then you can add Misc to your path as follows:

```
>>addpath(pwd)
```

What you are doing here is telling the computer to add the folder you are currently in (pwd) to the path.

Remember how in my computer I used which to find that MixStrings lived in the folder

C:\WORK\Matlab class & book\MatlabBookOrganization\class programs\Misc

Well I can now use that information at the beginning of MixStrings to make sure that the folder Misc is in my path by just adding the line

addpath('C:\WORK\Matlab class & book\MatlabBookOrganization\class programs\Misc') at the beginning of the program

Back to MixStrings.m

OK, back to work. Go back to your MixStrings m-file in the editor window and add the following lines of code. You can delete the commented lines that didn't belong properly to your header. Remember to save

MixStrings when you've finished changing it.

Green is comments  
Purple is text that is part of a string  
Blue is commands that start and end loops

```
% MixStrings.m
%
% Replaces 'this is gibberish' with a
% string of xxxxx's
%
% written by IF & GMB 4/2005
```

```
clear all
```

```
addpath('C:\WORK\Matlab class & book\MatlabBookOrganization\class programs\Misc')
```

```
str='this is gibberish'
```

```
for i=1:17
    str(i)='x';
    disp(str)
    pause
end;

disp('all done!')
```

Now go back to your command window and type:

```
>>MixStrings
```

This tells Matlab to run the program MixStrings.m. Then keep hitting the space bar (or any other key) until “all done!” appears.

*So what actually was going on just then?*

```
11     clear all
12
13     addpath('C:\WORK\Matlab class & book\MatlabBookOrganization\class
14     programs\Misc')
15
16     str='this is gibberish'
17
18
19     for i=1:17
20         str(i)='x';
21         disp(str)
22         pause
23     end;
24
25     disp('all done!')
```

Line 1. Erase all the variables currently in Matlab’s memory

Line 2. Define a string that contains the letters ‘this is gibberish’

Lines 3-8. This is called a *for* loop. The variable *i* won’t simply create an array that goes from 1 to 17. Instead it *steps* from 1 to 17 in steps of 1. So the first time around the loop *i*=1, the second time round the loop *i*=2, and so on up to *i*=17. Each time *i* increases in value Matlab will perform the commands that are inside the loop (these should be indented (tabbed) to make your code easier to read). Each of the 17 times the program repeats the commands inside the loop, *i* will be a different number.

Line 4. Gradually replace *str* with ‘x’ as *i* goes from 1 to 17

Line 5. Display what *i* is (*i* is a number that gradually steps from 1 to 17)

Line 6. Display what *str* is (‘this is gibberish’ will gradually be replaced by ‘xxxxx ...’, as *i* increases from 1 to 17)

Line 7. Wait for you to press any key.

Line 8. *end* tells the program that the *loop* is over

Line 9. Display ‘all done!’ at the command window.

## Variable Types

```
>>str='this is a string'
>>vect=[1 2 10 -4 5]
>>whos
```

You should see something like this:

Name	Size	Bytes	Class
str	1x16	32	char array
vect	1x5	40	double array

Grand total is 21 elements using 72 bytes

`whos` is like `who` but it has a little more information. As well as listing what variable are present in the computer's memory `whos` will all tell you what size the variable is.

*What do the Bytes and Class columns represent?*

The "Bytes" column tells you how much memory the variable is taking up. The "Class" column tells you that `str` is a list of characters (letters) and `vect` is a list of *doubles*. (If you are used to programming in another language you will appreciate the fact Matlab does not require you to specify what a class a variable is.)

*What on earth is a double?*

To represent a variable the computer needs to store information about what that number is. The more precision you represent the number with, the more space it takes in the computer memory to store that number. Computers work in a binary code where a *bit* represents the ability to take 2 different values (usually taken to be 0 or 1). A double is a number that is represented using  $2^{64}$  different bits. 64 bits can represent up to  $1.8^{19}$  different values, so this allows for a big range of numbers and a lot of precision. In these days of powerful computers Matlab uses doubles as the default.

So if you look again at `str` and `vect` you will notice the somewhat curious fact that there are `vect` is using up more Bytes than `str` even though `vect` only has 5 numbers in it, whereas `str` is a list of 16 letters. This is because it doesn't take a lot of memory to define a character (There are only 28 letters in the alphabet, plus you have capitals and punctuation. But even so, you only have just over 100 of alternatives to differentiate between). In contrast, when you represent numbers you are representing them with a lot of precision ( $1.8^{19}$ ). Of course the particular numbers that are in `vect` could be represented easily with less precision, but by default Matlab uses high levels of precision for numbers unless you tell it otherwise. We'll talk about that more later.

Now I'm going to show you something tricky:

```
>>clear all
>>a=4
>>b='4'
>>whos
```

Name	Size	Bytes	Class
a	1x1	8	double array
b	1x1	2	char array

Grand total is 2 elements using 10 bytes

`a` represents the number 4, and `b` represented the *character* '4'. As far as Matlab is concerned they are not the same thing. Sometimes you may want to change a number into a character. For example, if you want to include a number in a string.

Here's an example of doing that:

```
>>killednum=num2str(2500);
>>leftystr='the number of left-handed people killed annually by
right-handed products is ';
```

## Chapter 1 - 10 - 4/18/2007

Note the space at the end of this string

```
>>killedleftystr=[leftystr, killednum]
```

We used the square brackets to tell Matlab to combine the strings separated by commas into a single string

You can also skip the steps of creating variables along the way

```
>> killedleftystr=['the number of left-handed people killed  
annually by right-handed products is ' num2str(2500)]
```

Here's a second example:

```
>>secsnum=13;
```

```
>>beginchickenstr='The longest recorded chicken flight is ';
```

```
>>endchickenstr=' secs'
```

```
>>chickensecstr=[beginchickenstr num2str(secsnum) endchickenstr]
```

You can also go the other way, and change a string into a number

```
>>marriedstr='35% of people using personal ads for dating are  
already married';
```

```
>>baddates=str2num(marriedstr(1:2))
```