Chapter 1: What is programming and why learn it?

1.1 How to use this book

- *Be careful how you skip through this book.* Unless you are fluent in another language, please don't skip. A lot of important stuff happens even in Chapter 2. We've noticed that people who skip early chapters and exercises tend to get horribly stuck later because they are missing crucial skills.

- *Find a buddy*
  It's great if you can persuade a buddy to learn with you (not your boyfriend if you want the relationship to survive). But you don't want to be watching while the 'better' buddy writes code, see the next tip on gurus.

- *Find a guru*
  Don't hesitate to ask a better programmer to be your guru – like good deeds, one of the hidden expectations of the Matlab brotherhood is that you must help pass on the knowledge. (Besides, the stupid problems that beginners struggle with are secretly quite hilarious.)

  It's even ok to get your guru to write the code for an exercise you are struggling with and explain it to you. Spend as long as you want looking at your guru's code. But then, close the file containing their code, take a 2 minute break away from the computer, and try to re-write the program *without looking at their code*. If you can do that successfully you have encoded what your guru has taught you, and you aren't relying on short term memory.

  For bugs, ask your guru to give you a hint rather then just telling you what the bug is.

- *Don't go programming blind.*
  If you've been staring at the same bug for more than an hour, take a 15 minute break. Preferably, go outside and take a walk. If you've been staring at the same bug for more than four hours then give up for the day. You are likely to find that the bug is easier to solve in the morning.

- *Don't become a crazy-Frito-eating-insommniac*
  It's easy to code till 4 am and then lie half-awake till 6 am dreaming of 'for loops'. Set a time for when you are going to stop programming. Do something else for at least ½ an hour before you go to bed.

- *Remember the 100:10:1 rule*
  A very small simple program takes me about 1 hour to write because I have 10 years of programming experience (actually 20). The same program took me about

10 hours to write when I had 1 year of experience. It took 100 hours to write when I had 1 week of experience. You *will* get faster.

1.2 What is programming?

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

* Computers are very stupid – you have to tell them EXACTLY what to do.
* 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 and faster 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.

The metaphor with language goes further. Like real languages some things are easier to say in one language rather than another (Italian is the language of love etc. etc.). Some programming languages are better for computations, others for graphics. Like any language, programming languages have grammar. 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 very boring and very hard to say the simplest thing, but it will get easier, fast. Finally, like learning languages, with even basic fluency you can do a lot of things you couldn't do otherwise.

1.3 Why program?

The following is a summary of the conversations I've had with students who have been wondering whether or not to learn to program.

"My field has lots of specialized prewritten software packages. Why can't I just use those?"

"You can, but your experiments/analyses will be limited to what these packages can do. So a lot of really creative things aren't possible. What's more, I've noticed in some senior colleagues that spending many years thinking "inside the box" - only designing programs that are within the technical capacities of their software packages - can also limit their ***theoretical*** *creativity."*

"But surely these specialized software programs are way better then anything I can write?"

"Yes and no. These software packages are written by real programmers and are used by lots of people. So they tend to be pretty reliable, and have fewer bugs. But they are definitely not bug free – no program is. If you can program you will have better insight into where the bugs are and will be able pinpoint issues with technical

support personnel much more quickly. With open source software – which is becoming increasingly common - you will be able to fix the bugs yourself."

"I plan to hire a professional programmer when I run my lab."

"This has the advantages of flexibility and not needing to spend a lot of time programming yourself or teaching your students to program. However, good programmers are expensive and they can be hard to find. Computer scientist students are cheap and available, but they are not always good. And unless you yourself can program it's hard to evaluate how good a job they are doing. Even if you hire a programmer, being able to program yourself means you will be better able to describe what you want in a program and what kind of flexibilities you will want in terms of future experiments. Even more importantly, you will know how to bug-check the program carefully once it is written."

"I've tried to learn before, and programming is too hard for me."

"Every year some students in my class ***love*** *the class, find the exercises ridiculously easy and are writing programs that predict the next winner of American Idol by week 7. Yes, some of them are uber-geeks, but others are relatively cool by graduate student standards. Other, equally bright, students hate the class, feel totally inferior, and end up in my office in tears.* ***But … they all learn to program competently in the end if they put their time in****. Like real languages, some people have an 'ear for language' and others don't. I have no ear for language whatsoever and I still speak reasonable French after 1000 hours of high school instruction. I'm not a natural programmer and I'm writing this book. If you find this book really difficult, relax, it doesn't mean you are stupid. Accept that you are not a natural programmer and it's going to take you a while. Or possibly we've written a bad book.*

1.4 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 destroy by poking it with a screwdriver.

1.5 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 – theinstructions 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. You may find that a few 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!

1.6 Getting Started

You need:
* A PC running Windows or a Mac computer running OSX.
* You need to install Matlab on your computer.

Now, start Matlab. A window will appear that’s divided into a number of sub-windows. Close all the sub-windows (Later you may find these other windows helpful, but we will ignore them for now) except the one that called the "command window” which has a little prompt:
>>
This is where you should type your commands. In this book the text in green is what you should type into the command window. The text in blue shows what the command window should spit back out at you.

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

```
str1 =
I have no clue what I am doing
```

Congratulations! You have just done your first bit of programming.