Coding Tutorial

Derrick Hasterok

February 1, 2005

1 Beginning to Code

Starting to write a program can be difficult for a beginner because of the large vocabulary one must first amass. Once this hurdle is overcome, coding quickly comes much more easily and the hard problems come in designing efficient algorithms. This document is to help with some of the vocabulary and ease the learning curve. The examples are emphasised using Matlab.

Before putting pencil to paper or keystrokes to a script, begin with writing *pseudocode*. Pseudocode is basically an outline to help in writing code. Once the program is done, the pseudocode becomes the major comments within the code. Pseudocode is a very useful exercise for the beginner. An experienced programmer often does not write pseudocode for small scripts and it becomes increasingly terse for larger applications, but can still be useful to remind the author of the scripts ultimate goal.

Suppose you wanted to write a program which finds all of the prime numbers between 1 and 100. The pseudocode might read something like this:

```
Goal: Find all prime numbers between 1 and N.
Algorithm: A number is prime if it is not divisible by any integer
up to its square root.

set value of N to be an integer #

loop n from 1 to N
    set flag prime true
    loop i from 2 to square root of n
       if n/i is integer
           set flag prime false
           break i loop
    end i loop
    if prime true
        add to list
end n loop

print prime number list
```

This pseudocode introduces several very freqently used methods in a code. The code resulting from the pseudocode is:

```
 1: % This program finds prime numbers between 1 and N
 2: % checking divisibility by integers up to the
 3: % square root of the number.
 4:
 5: N = 100;       % Maximum value to test
 6:
 7: pnums = [];    % Start with empty list
 8:
 9: % Loop to test all numbers 1 to N
10: for n = 1:N
11:     flag = 1;  % Set flag primes true
12:
13:     % Test divisibility of n of integers from 1 to sqrt(n)
14:     for i = 2:floor(sqrt(n))
15:         if mod(n,i) == 0
16:             flag = 0;  % Set flag primes false
17:             break      % Break from i loop
18:         end
19:     end  % i loop
20:
21:     % If prime, add to list
22:     if flag == 1
23:         pnums = [pnums; n];
24:     end
25: end  % n loop
26:
27: % Print list
28: pnums
```

Notice how closely the comments resemble the original psuedocode.

Programming in most languages use top down methods of programming, which means that the flow of the program starts at the top and progresses line by line until the end of the program is reached. There are a few commands which can break or interupt the flow somewhat, but they are often necessary. Loops *for* and *while*, *if/elseif/else* statements and *goto* and the like, all break the line by line flow somewhat. When a loop is reached, the flow treats the loop almost as if it is a small program itself, executing the statements within the loop in sequence until the end of the loop is reached. Once the end of the loop is reached, it jumps back up to the top and begins executing the loop statements again, until the end condition is reached. See the document *Introduction to Programming* linked to on the website for discussions of other useful tips when beginning to program such as why to indent and comment the code.

2 Why '=' Is Not '=='

Unlike humans who can understand math and determine the difference between equating something and comparing two things, a computer has to be told. When starting to code, this can cause confusion. Writing $a = 5$ is not the same as $a == 5$. The first statement $a = 5$ is an assignment, seting the value of the variable a to be the number 5. The second statement is a comparision, saying "Is a equal to the number 5?" If a is equal to 5, then $a == 5$ returns *true* (1), if a is not equal to 5, then it returns *false* (0). Assignment expressions are used most often in scientific programming and change are used to set the value of variables. Comparison statements have many uses, but used most often as a test to execute (or not) a section of code. Comparisons are used in *if* statements, *while* loops, *switch/case* and to set the value of flags.

This is a very important concept to understand. Try doing a simple test on the Matlab command line. For example, set a variable `a` equal 10. Then set the variable `b` equal to `a`. Then try `b == a`. What is the result? Now set `b` to be the value 9. Now try `b == a`. What is the result now? Why?

```
EDU>> a = 10
a =
    10

EDU>> b = a
b =
    10

EDU>> b == a
ans =
     1

EDU>> b = 9
b =
     9

EDU>> b == a
ans =
     0
```

When `b` is set equal to `a`, that is an assignment, using "=". When `b` is "==" to `a`, this is a test, and the computer returns *true* or *false*.

3 Looping

3.1 *for* Loops

There are a couple different ways to loop, the most common of which is the *for* loop. Loops are used to do a particular procedure a given number of times. *For* loops are written in the following way:

```
for n in A to B
    statement1
    statement2
    ...
end
```

A *for* loop is initiated by telling it how many times to execute. This is the statement `n in A to B`. `A to B` need not be integers, but must be increasing. If `A to B` is not increasing, the *for* loop will not execute the statements (`statement1, statement2, ...`). A more concrete example of a for loop would be a simply incrementing a number:

```
1: a = 0;
2: for i = 1:100
3:    a = a + 1;
4: end
5: a
```

This loop simply starts with `a = 0` and adds 1 to the value of `a` from 1 to 100 times. The program flow starts at line one and reaches the loop at line 2 which continues to step 4. Once the program reaches step 4, it jumps back to line 2, increments the counter `i` and continues again to line 4. Once `i` reaches the value of 100, the loop is terminated and the program then will move on to line 5.

3.2 *while* Loops

A *while* loops greatest advantage is that it is designed for cases when the number of times a loop needs to be executed is indefinite. This is the major difference between a *while* and a *for* loop. A *while* loop is generally executed in the following way:

```
while (boolean expression)
    statement1
    statement2
    ...
end
```

The boolean expression can be any type of comparison. When *true*, the loop executes until the test returns a *false*. For example a *while* loop may be used to iteratively solve an infinite series until the difference between the previous and

current sum is less than some tolerance. Below is an example of a *while* loop solving the same problem as shown for *for* loops above.

```
1: a = 0;
2: while a < 100
3:     a = a + 1;
4: end
5: a
```

The program will begin at line 1, then reach the while loop at line 2 and execute the while loop until `a` is no longer <100. When `a` is not <100 (`a = 100` in this case), the loop will terminate and the program will jump to and execute line 5.

4 Commands that Interupt the Flow, *continue, break* and *return*