This practical completes the introduction to basic programming in Matlab. By the end of this session, you will know enough Matlab to be able to write useful programs. The following topics will be covered in this practical:

- conditionals (revision)
- while loops
- for loops

The practical ends with a selection of tasks that allow you to determine whether you know sufficient Matlab to be able to successfully complete the forthcoming practicals.

Before you begin, make a new folder for this practical in U:\MATLAB named practical_M3. Put your scripts and functions into this folder. From now on you should do this automatically for every practical.

1. **Conditionals and relational operators revisited**

Conditionals and relational operators will be used a lot in future practicals so it worth revising them (use your notes from Practical 2 to help with Task 1; remember electronic copies of the material from previous practical classes are available via CamTools).

### Task 1

The general quadratic equation may be written as

\[ ax^2 + bx + c = 0. \]

Its roots are the value(s) of \( x \) for which the equation is satisfied. These roots may or may not be real: this is determined by considering the sign of the “discriminant”. The following classification applies:

- \( b^2 - 4ac > 0 \) there are two real roots;
- \( b^2 - 4ac = 0 \) there is one real root that is repeated;
- \( b^2 - 4ac < 0 \) there are no real roots.

(a) Write a function `quadraticSoln(a, b, c)` which takes the parameters \( a, b, c \) as defined above and prints out a classification of the roots of the associated quadratic (i.e. the number of real roots that it has).

(b) Use this function to classify the following quadratics

(i) \( 5x^2 - 10x + 5 = 0 \)
(ii) \( 25x^2 + 14x + 2 = 0 \)
(iii) \( x^2 + 7x + 8 = 0 \)

(c) Extend the function to actually print out the real root(s) in those cases where this is appropriate, calculating them using the quadratic formula:

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \]
2. Loops

Loops allow us to repeat sections of code. There are two main types of loop: *for* loops and *while* loops.

2.1 While loops

While loops repeat a certain block of code, called the **body** of the loop, for as long as a particular **condition** is true. While loops have the following format.

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>while &lt;condition&gt;</td>
<td>Relational statement &lt;condition&gt; may be true (1) or false (0).</td>
</tr>
<tr>
<td>&lt;body&gt;</td>
<td>If &lt;condition&gt; is true then run the &lt;body&gt; of the code.</td>
</tr>
<tr>
<td>end</td>
<td>end tells Matlab to go back and check &lt;condition&gt; again.</td>
</tr>
<tr>
<td>&lt;rest of code&gt;</td>
<td>Once &lt;condition&gt; is false then move onto &lt;rest of code&gt;.</td>
</tr>
</tbody>
</table>

Now read through the following Matlab code.

```matlab
%% while loop
countdown = 5;
while countdown > 0             % checks condition: either 0 or 1
    disp(countdown)            % if condition is 1 goes here
    countdown = countdown - 1;
end
disp('Blast Off!')            % Once condition is 0 goes here
```

What output do you expect? (Note that the command *disp* prints a value to the screen).

Type the code into a script file to check your answer.

**WARNING!** It is essential that there is a statement in the body of the *while* loop that updates the value checked in the conditional statement (e.g. the line updating the variable *countdown* in the above). If there is no such line, then if the condition is true once, then it will be true forever, and the program would get stuck in an infinite loop. If you do make a mistake and write a program which gets stuck in an infinite loop **press Ctrl-C to halt the program.**

**Task 2**

Use a *while* loop to find the largest prime number that is less than one thousand, using the built-in function *isprime(x)*.

*Hint: You can check how *isprime* works by typing *isprime(4) and isprime(7) into the Command Window.*
### 2.2 For loops

**For loops** are similar to **while** loops but use the values in a vector to control how many times the loop runs. They work by executing the loop for each value in a given vector (from the first value to the last). The format of a **for** loop is

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>for i = &lt;vector&gt;</td>
<td>List of the values of the variable i to use within the for loop.</td>
</tr>
<tr>
<td>&lt;body&gt;</td>
<td>Code to be run for each value of i.</td>
</tr>
<tr>
<td>end</td>
<td>end tells Matlab to go back and to run loop for next value of i.</td>
</tr>
<tr>
<td>&lt;rest of code&gt;</td>
<td>Once all the elements of vector have been used as a value of i, move onto &lt;rest of code&gt;.</td>
</tr>
</tbody>
</table>

Now read through the following Matlab code.

```matlab
%% for loop
for i = 5:-1:1
    disp(i)
end
disp('Blast Off!')
```

What output do you expect?

Type the code into a script file to check your answer.

---

**Task 3**

Use a **for** loop to print out the square of all the even numbers from 2 to 20. Note the function `disp([x,y])` prints out the variables `x` and `y` on the same line.

---

### 2.3 Should a for or a while loop be used?

Generally, **for** loops should be used if you know in advance how many times the loop will repeat. If you can’t determine in advance how many times a loop will be repeated, then a **while** loop should be used instead.

An example of this distinction is given by the following:

- **Problem 1**: Find all prime numbers less than 25.
- **Problem 2**: Starting from 1 find the 25th prime number.

Problem 1 always requires checking the first 25 numbers so would require a **for** loop.

Problem 2 requires checking all integers starting from 1 to see if they are prime until you reach the 25th prime number. In this case it is unclear beforehand how many numbers need to be checked so a **while** loop should be used (using the number of primes found so far in the condition at the top of the loop).

During most of these practicals you will be using **for** loops, so if in doubt try a **for** loop first.

---

**NEED HELP? ASK A DEMONSTRATOR!**
2.4 Nesting if statements within loops

Suppose we are set Problem 1: find all prime numbers less than 25. We have already decided that using a for loop is most appropriate.

So we might write

```matlab
%% Problem 1: finding all primes less than 25
for X = 1:25
    disp([X,isprime(X)]);
end
```

What output would you expect?

Type the code into a script file to check your answer.

The above code answers the question, but really we only want to display the prime numbers. To do this we can insert an if statement into our loop which will only display X if it is a prime.

```matlab
%% Problem 1: finding all primes less than 25
for X = 1:25
    IsXPrime = isprime(X);
    if IsXPrime == 1
        disp(X);
    end
end
```

What output would you expect from the modified code?

Type the code into a script file to check your answer.

One might like to add other features into your program to improve the output. For example we might want to output the number of primes less than 25 and to store all the primes in a vector.

```matlab
%% Problem 1: finding all primes less than 25
PrimeList = []; % creates empty vector to store primes
for X = 1:25
    IsXPrime = isprime(X);
    if IsXPrime == 1
        PrimeList = [PrimeList X]; % appends X to list
    end
end
disp([num2str(PrimeList), ' are the primes less than 25'])
NumPrimes = length(PrimeList);
disp(['There are ', num2str(NumPrimes), ' in total'])
```

Notice this code uses the functions length and num2str. Use doc to investigate what they do.

What output would you expect from the modified code?

Type the code into a script file to check your answer.

NEED HELP? ASK A DEMONSTRATOR!
The following tasks are designed to give you more practice of writing loops and using conditional statements. You may not have time to complete all the tasks given below during the practical but remember you can use Matlab on your College computers. It is important that you are comfortable with using for and while loops, if statements and writing simple functions as you will be expected to use them in future practicals.

**Task 4**
Write a for loop to plot the function \( y = e^{\beta t} \) for values of \( \beta \) ranging in steps of 0.1 from 0.1 to 2 for values of \( t \) ranging from 0 to 1.

**Task 5**
Using the `isprime` function, determine how many prime numbers there are between 100 and 200.

**Task 6**
Use the function `isprime` to find 25th prime number (starting at 1).

**Task 7**
Using a for loop write a function `fact(n)` that gives the factorial of a positive integer \( n \). The factorial of a number, written \( n! \) is the product of all the numbers from 1 to \( n \), i.e. \( 1 \times 2 \times 3 \times ... \times n \). Use your function to find 7!

**Task 8 (Harder: optional)**
The first two elements of the Fibonacci sequence are 1,1. Subsequent elements are obtained by adding the previous two elements i.e.

\[
f_n = f_{n-1} + f_{n-2} \quad \text{for} \quad n \geq 3
\]

The first five Fibonacci numbers are 1,1,2,3,5. Use a for loop to write a script that finds the 30th Fibonacci number.

*Hints:*
1. The for loop should go from \( i = 3 \) to \( i = 30 \).
2. To find the \( i \)th element of a vector called `vectorname`, type `vectorname(i)`

**Task 9 (Harder: optional)**
One way of defining the exponential function is using the following so-called “Taylor series” (do not worry if you have not seen this before: it will be taught in the Lent term)

\[
e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + ...
\]

(a) Write a function `myExp(x, n)` which calculates \( e^x \) using the first \( n \) terms of the series above.
(b) What is the error when calculating \( e^2 \) using \( n = 5 \)?
(c) How many terms are required to get an error of less than 0.001?