**Loops**

In general in programming, there are two kinds of loops: counted loops, that repeat an action a specified number of times, and conditional loops, that repeat an action until something happens or as long as a condition is true. In MATLAB, the **for** loop is the counted loop, and **while** is the conditional loop.

**For loop:**

The general form of a **for** loop is

for loopvar = range

action

end

where *loopvar* is the loop variable, *range* is the range through which it iterates, and action is the set of statements that will be repeated. Frequently the colon operator is used to specify the range. For example, a loop to repeat 5 times would have the form:

for i = 1:5

action

end

This specifies that the loop variable *i* iterates through the integers from 1 to 5, so the action is repeated 5 times.

Sometimes the loop variable is used in the action, and sometimes it is not (it is just used to specify how many times to repeat the action). For example, the following uses the value of i:

for i = 3:-1 :1

disp(i)

end

3

2

1

The following does not use the loop variable in the action:

for i = 1:4

fprintf('!')

end

!!!!

The loop variable can be used to index into a vector variable.

vec = [33 11 2 5];

for i = 1:length(vec)

fprintf('%d: %d\n', i, vec(i))

end

1: 33

2: 11

3: 2

4: 5

If a vector is to be created in a loop, it is best practice to ***preallocate*** the vector first if the eventual length of the vector is known. Vectors can be extended, but that is very inefficient. It is common to preallocate a vector to all zeros. For example:

>> vec = zeros(1,3);

>> for i = 1:3

vec(i) = input('Enter a number: ');

end

Enter a number: 4

Enter a number: 9

Enter a number: 2

>> vec

vec =

4 9 2

**While loops**

The conditional loop in MATLAB is the **while** loop. The general form is:

while condition

action

end

The condition is a Boolean expression. The action is repeated as long as the condition is true. The indentation of the action is not necessary, but makes it easier to read. The action is any number of statements up to the reserved word **end**.

For example, the following loops to prompt the user for a number greater than 50, until the user does this.

num = input('Enter a number > 50: ');

while num <= 50

num = input('Enter a number > 50: ');

end

fprintf('Thanks, you entered %.1f\n', num)

Enter a number > 50: 33

Enter a number > 50: -8

Enter a number > 50: 52

Thanks, you entered 52.0

**Nested loops**

A nested loop is one loop inside the action of another loop. For example, a nested **for** loop to print might look like this:

for i = 1:3

for j = 1:5

fprintf('\*')

end

fprintf('\n')

end

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

In the following example, the script prompts the user for 3 negative numbers to store in a vector, each time looping to error-check until the user enters a negative number.

createnvec.m

negvec = zeros(1,3);

for i = 1:3

num = input('Enter a negative number: ');

while num >= 0

num = input('Enter a negative number: ');

end

negvec(i) = num;

end

disp(negvec)

Here is an example of executing the script:

>> createnvec

Enter a negative number: -4

Enter a negative number: 33

Enter a negative number: 2

Enter a negative number: -7

Enter a negative number: -11

-4 -7 -11

Nested loops can also be used to perform an operation on every element in a matrix. One loop would be over the rows of the matrix, and the other loop over the columns.

For example, given a matrix variable *mat*, we will write code to find the overall sum of the numbers in the matrix, the sum of each row, and the sum of each column (without using the **sum** function).

matsums.m

mat = [1 3 5; 2 6 3]

[r c] = size(mat);

% Calculate the overall sum

mysum = 0;

for i = 1:r

for j = 1:c

mysum = mysum + mat(i,j);

end

end

fprintf('The overall sum is %d.\n\n', mysum)

% Calculate the sum for each row

for i = 1:r

mysum = 0;

for j = 1:c

mysum = mysum + mat(i,j);

end

fprintf('Sum for row %d: %d\n', i, mysum)

end

fprintf('\n')

% Calculate the sum for each column

for j = 1:c

mysum = 0;

for i = 1:r

mysum = mysum + mat(i,j);

end

fprintf('Sum for column %d: %d\n', j, mysum)

end

Executing this would result in:

>> matsums

mat =

1 3 5

2 6 3

The overall sum is 20.

Sum for row 1: 9

Sum for row 2: 11

Sum for column 1: 3

Sum for column 2: 9

Sum for column 3: 8

There are several things to note in this code:

* When a result is calculated for every row, the outer loop has to be over the rows
* When a result is calculated for every column, the outer loop has to be over the columns
* When an overall result is calculated, it does not matter whether the outer loop is over the rows or columns
* The matrix is always indexed with the row index first, then the column index, regardless of the order of the loops
* For an overall sum, the running sum variable is initialized to 0 before the nested loop
* For a sum of every row, the running sum variable must be initialized to 0 for every row
* For a sum of every column, the running sum variable must be initialized to 0 for every column

**Vectorizing Code**

Since MATLAB is written to work with vectors and matrices, it is never necessary to use loops when performing an operation on every element in a vector or matrix, or to call a function on every element in a vector or matrix.

For example, to add 5 to every element in a vector we could loop as follows:

>> vec = [4 9 2];

>> for i = 1:length(vec)

vec(i) = vec(i) + 5;

end

>> vec

vec =

9 14 7

The vectorized code would be instead:

>> vec = [4 9 2];

>> vec = vec + 5

vec =

9 14 7

Similarly, to multiply every element in a matrix by 2, we could use a nested loop as follows:

>> mat = [3:5; 2 11 16]

mat =

3 4 5

2 11 16

>> [r c] = size(mat);

>> for i = 1:r

for j = 1:c

mat(i,j) = mat(i,j) \* 2;

end

end

>> mat

mat =

6 8 10

4 22 32

The vectorized code is:

>> mat = [3:5; 2 11 16]

mat =

3 4 5

2 11 16

>> mat = mat \* 2

mat =

6 8 10

4 22 32