Break command:

The **break** command will terminate a loop prematurely.

```
n = 10;
for i=1:n
   disp(i^2)
   if (i==5)
     break;
   end
end
```

After execution:

```
1
4
9
16
25
```

Continue command:

The **continue** command goes back to the start of a loop and increments by one unit.

```
n = 5;
for i=1:n
  if (i==3)
    continue;   % when i is 3, the loop goes back to the beginning and increases by 1
  end
  disp(i^2)
end
```

After execution

```
1
4
16
25
```

Nested for loops:

Often you need to put one or more for loops inside another for loop. For every for loop
Example:

```
for i=1:3
  for j=1:2
    disp([i j])
    disp(i+j)
    x(i,j) = i+j;
  end
end
```

After execution:

```
 1   1
 2
 1   2
 3
 2   1
 3
 2   2
 4
 3   1
 4
 3   2
 5
```

A more elaborate example: 2 quizzes are given in a particular class to 5 students. The scores for quiz #1 are stored in the first row and scores for quiz #2 are stored in the second row.

	Student 1	Student 2	Student 3	Student 4	Student 5
Quiz 1	90	66	99	83	98
Quiz 2	93	87	100	89	100

What is the average quiz score?

```
score = [90, 66, 99, 83, 98 ; 93, 87, 100, 89, 100]
[nrow,ncol] = size(score);
sum = 0;
for i=1:nrow
  for j=1:ncol
    sum = sum + score(i,j);
  end
end
average = sum/(nrow*ncol);
disp('Average using nested loops:')
disp(average)
disp('Average using mean() function')
disp( mean( score(:) ) )
```

After execution:

```
score =
   90    66    99    83    98
   93    87   100    89   100
Average using nested loops:
 90.500
Average using mean() function
 90.500
```

INPUT and OUTPUT:

(A) Simple printing to the screen using disp( ):

You have already learned about the **disp( )** function. This function is useful if you want to print something to the screen without fancy formatting.

Example:

```
x = [1,3,5,7,9];
score = 90.2;
disp('The elements in the array x are:')
disp(x)
disp('Your test score is:')
disp(score)
```

After execution:

```
The elements in the array x are:
   1   3   5   7   9
Your test score is:
 90.200
```

If you want to print different types of data (for example, character and floating-point) on the same line, you must convert the floating-point data to a string using the **num2str( )** command.

Example:

```
x = [1,3,5,7,9];
score = 90.2;
disp(['The elements in the array x are: ' num2str(x)])
disp(['Your test score is ' num2str(score)])
```

After execution:

```
The elements in the array x are: 1   3   5   7   9
Your test score is 90.2
```

(B) Fancier printing with the fprintf( ) command.

The general form of the **fprintf**( ) function is the following:

fprintf(string with format commands, variable/array names)

Example:
score = 94.5;
fprintf('Your score on the test was %f out of 100', score)

After execution:
Your score on the test was 94.500000 out of 100

Notice how the score was inserted at the **%f**. The **%** symbol is called a place holder. The **f** means that a floating-point number (a number with a decimal point) will be inserted at the location of the **%** symbol. There are many different types of formats.

%f = fixed-point (decimal) notation
%e = scientific notation
%g = fixed-point or scientific notation with no trailing zeros
%i = integer
%c = single character
%s = string of characters

If you have two or more variables, you can use two or more placeholders.

Example:
score1 = 92.5;
score2 = 81.0;
fprintf('The two scores on the test were %f and %f', score1,score2)
fprintf('The two scores on the test were %g and %f', score1,score2)
fprintf('The two scores on the test were %e and %e', score1,score2)

After execution:
The two scores on the test were 92.500000 and 81.000000The two scores on the test were 92.5 and 81.000000The two scores on the test were 9.2
50000e+001 and 8.100000e+001

Whoops. We tried to print 3 lines and they printed on the same line. We need to tell the program to go down to a new line with **\n**.

Example:

```
score1 = 92.5;
score2 = 81.0;
fprintf('The two scores on the test were %f and %f \n', score1,score2)
fprintf('The two scores on the test were %g and %f \n', score1,score2)
fprintf('The two scores on the test were %e and %e \n', score1,score2)
```

After execution:

```
The two scores on the test were 92.500000 and 81.000000
The two scores on the test were 92.5 and 81.000000
The two scores on the test were 9.250000e+001 and 8.100000e+001
```

You can print off array elements in a similar manner.

Example:

```
score(1) = 92.5;
score(2) = 81.0;
fprintf('The two scores on the test were %f and %f. \n', score)
```

After execution

```
The two scores on the test were 92.500000 and 81.000000.
```

Here are some useful formatting commands
\n = newline
\t = tab
\b = backspace

You can have even more control over the precision of the variables when displayed to the screen. Previously, we used **%f** to denote a decimal format. The general form of this formatting option is the following:

%X.Yf

where **X** is the total number of spaces for displaying the variable's value and **Y** is the amount of numbers to display right of the decimal. Keep in mind that the decimal place and negative sign (if it exists) takes up one space.

%X.Ye = scientific notation format
%Xi = integer
%Xc = single character
%Xs = string of characters

Example:

```
score(1) = 92.5;
score(2) = 81.0;
average = (score(1) + score(2)) / 2;
names = char('Paul' , 'Frank');
fprintf('The two scores on the test were %8.3f and %4.1f \n', score(1), score(2))
fprintf('The two scores on the test were %6.3f and %10.0f \n', score(1), score(2))
fprintf('The two scores on the test were %10.1e and %10.4e \n', score(1), score(2))
fprintf('Hello %10s and %10s \n', names(1,:), names(2,:))
fprintf('%4s received a score of %5.2f', names(1,:), score(1))
```

Output:

```
The two scores on the test were   92.500 and 81.0
The two scores on the test were 92.500 and        81
The two scores on the test were   9.2e+001 and 8.1000e+001
Hello      Paul and      Frank
Paul received a score of 92.50
```

For the first line,

- 8 spaces are allotted for **score(1)**. 3 of the spaces are allotted for numbers to the right of the decimal. The decimal takes 1 space, so 4 spaces are left for numbers to the left of the decimal. Since '92' takes up 2 spaces, there are 2 blank spaces left.
- 4 spaces are allotted for **score(2)**. 1 of the spaces is allotted for numbers to the right of the decimal. The decimal takes 1 space, so 2 spaces are left for numbers to the left of the decimal. Since '92' take up 2 spaces, there are 0 blank spaces left.

For the second line,

- 6 spaces are allotted for **score(1)**, which is the perfect amount. 3 of the spaces are allotted for numbers to the right of the decimal. The decimal takes 1 space.
- Notice that there is no decimal point for **score(2)**. This is due to zero spaces being allotted to numbers after the decimal.

For the third line, we are using the **e** formatting command.

- 10 spaces are allotted for **score(1)**. 1 spaces is allotted for numbers to the right of the decimal. Notice that the '5' is truncated… we didn't give enough space to display the entire number in scientific notation. The decimal point takes up 1 space and the 'e+001' takes up 5 spaces. 1 space is allocated for numbers to the right of the decimal point (this will always be 1 in scientific notation). Therefore, there are 2 blank spaces
- The %10.4e format is straightforward

For the fourth line, we are using the **s** formatting command.

- 10 spaces are allotted for each character string. If there are any spaces left over after all the characters have been placed, they will be added as blank spaces to the left of the character string.

For the fifth line, we are mixing the **s** and **f** formatting commands.