| | |
|---|---|
| الكلية | العلوم |
| القسم | الرياضيات |
| المادة باللغة الانجليزية | Programming (MATLAB) |
| المادة باللغة العربية | البرمجة بلغة (ماتلاب) |
| المرحلة الدراسية | الثانية |
| اسم التدريسي | صفوت عبدالقادر حمد |
| عنوان المحاضرة باللغة الانجليزية | Input and Output |
| عنوان المحاضرة باللغة العربية | الادخال والاخراج |
| رقم المحاضرة | التاسعة |
| المصادر والمراجع | MATLAB<br>A Practical Introduction to Programming and Problem Solving |
| | MATLAB<br>The Language of Technical Computing |
| | MATLAB numerical computing |

**University Of Anbar**

**College Of Science**

**Math Department**

# Programing Language (MATLAB)


Lecture 9

Input and Output

**By:**

**Safwat A. Hamad**

**For the 2nd stage Math Department**

# 1. Introduction

The previous script would be much more useful if it were more general; for example, if the value of the radius could be read from an external source rather than being assigned in the script. Also, it would be better to have the script print the output in a nice, informative way. Statements that accomplish these tasks are called input/output statements, or I/O for short. Although, for simplicity, examples of input and output statements will be shown here in the Command Window, these statements will make the most sense in scripts.

# 2. Input Function

Input statements read in values from the default or standard input device. In most systems, the default input device is the keyboard, so the input statement reads in values that have been entered by the user, or the person who is running the script. To let the user know what he or she is supposed to enter, the script must first prompt the user for the specified values.

The simplest input function in MATLAB is called input. The input function is used in an assignment statement. To call it, a character vector is passed that is the prompt that will appear on the screen, and whatever the user types will be stored in the variable named on the left of the assignment statement. For ease of reading the prompt, it is useful to put a colon and then a space after the prompt. For example,

```
>> rad = input('Enter the radius: ')
Enter the radius: 5
rad =
5
```

In this case, the prompt was printed and then the user entered 5. If character or character vector input is desired, **'s'** must be added as a second argument to the input function:

```
>> letter = input('Enter a char: ','s')
```

```
Enter a char: g
letter =
          'g'
```

If the user enters only spaces or tabs before hitting the Enter key, they are ignored and an empty array is stored in the variable:

```
>> mychar = input('Enter a character: ', 's')
Enter a character:
mychar =
          0×0 empty char array
```

However, if blank spaces are entered before other characters, they are included in the variable. In the next example, the user hit the space bar four times before entering "go." The length function returns the number of characters in the variable.

```
>> mystr = input('Enter a word: ', 's')
Enter a word: go
mystr =
          ' go'
>> length(mystr)
ans =
      6
```

It is also possible for the user to type quotation marks around the entry rather than including the second argument 's' in the call to the input function.

```
>> name = input('Enter your name: ')
Enter your name: 'Stormy'
name =
'Stormy'
```

In this case, the user entered a character vector, but it is also possible to enter a string:

```
>> name = input('Enter your name: ')
Enter your name: "Stormy"
name =
```

```
      "Stormy"
```

However, this assumes that the user would know to do this, so it is better to signify that character input is desired in the input function itself. Also, if the **'s'** is specified and the user enters quotation marks, these would become part of the variable.

```
>> name = input('Enter your name: ','s')
Enter your name: 'Stormy'
name =
      ''Stormy''
>> length(name)
ans =
      8
```

Note what happens if character input has not been specified, but the user enters a letter rather than a number.

```
>> num = input('Enter a number: ')
Enter a number: t
      Error using input
Unrecognized function or variable 't'.
Enter a number: 3
num =
      3
```

MATLAB gave an error message and repeated the prompt. However, if t is the name of a variable, MATLAB will take its value as the input.

```
>> t = 11;
>> num = input('Enter a number: ')
Enter a number: t
num =
      11
```

Separate input statements are necessary if more than one input is desired. For example,

```
>> x = input('Enter the x coordinate: ');
>> y = input('Enter the y coordinate: ');
```

Normally, in a script the results from input statements are suppressed with a semicolon at the end of the assignment statements.

It is also possible to enter a vector. The user can enter any valid vector, using any valid syntax such as square brackets, the colon operator, or functions such as ***linspace***.

```
>> v = input('Enter a vector: ')
Enter a vector: [3 8 22]
v =
      3   8   22
```

## 3. Output Statements: (disp) and (fprintf)

Output statements display character arrays or strings and/or the results of expressions and can allow for formatting, or customizing how they are displayed. The simplest output function in MATLAB is ***disp***, which is used to display the result of an expression or a string or character array without assigning any value to the default variable ans. However, disp does not allow formatting. For example,

```
>> disp('Hello')
Hello
>> disp(4^3)
64
```

Formatted output can be printed to the screen using the ***fprintf*** function. For example,

```
>> fprintf('The value is %d, for sure!\n',4^3)
The value is 64, for sure!
>>
```

To the ***fprintf*** function, first a string or character vector (called the format specifier) is passed that contains any text to be printed, as well as formatting information for the expressions to be printed. The format specifier can be either a

string or a character vector; historically (before R2016b), it was always a character vector, so that is the way they will typically be shown, both in this book and in the MATLAB documentation. In this example, the %d is an example of format information.

Note: The format specifier in ***fprintf*** can be either a character vector or a string, but the prompt in an input statement must be a character vector. The %d is sometimes called a place holder because it specifies where the value of the expression that is after the format specifier is to be printed. The character in the place holder is called the conversion character, and it specifies the type of value that is being printed. There are others, but what follows is a list of the simple place holders:

```
%d integer (it stands for decimal integer)
%f float (real number)
%c character (one character)
%s string of characters
```

Don't confuse the **%** in the place holder with the symbol used to designate a comment. The character '\n' at the end of the format specifier is a special character called the newline character; what happens when it is printed is that the output that follows moves down to the next line.

Note that the newline character can also be used in the prompt in the input statement; for example:

```
>> x = input('Enter the \nx coordinate: ');
Enter the
x coordinate: 4
```

However, the newline is the ONLY formatting character allowed in the prompt in input. To print two values, there would be two place holders in the format specifier, and two expressions after the format specifier. The expressions fill in for the place holders in sequence.

```
>> fprintf('The int is %d and the char is %c\n',33 - 2, 'x')
    The int is 31 and the char is x
```

A field width can also be included in the place holder in ***fprintf***, which specifies how many characters total are to be used in printing. For example, floats, the number of decimal places can also be specified; for example, %6.2f means a field width of 6 (including the decimal point and the two decimal places) with 2 decimal places. For floats, just the number of decimal places can also be specified; for example, %.3f indicates 3 decimal places, regardless of the field width.

```
>> fprintf('The float is %2f\n', 4.9)
    The float is 4.90
```

Note that if the field width is wider than necessary, leading blanks are printed, and if more decimal places are specified than necessary, trailing **zeros** are printed.

## 4. Printing Vectors and Matrices

For a vector, if a conversion character and the newline character are in the format specifier, it will print in a column regardless of whether the vector itself is a row vector or a column vector.

```
>> vec = 2:5;
>> fprintf('%d\n', vec)
2
3
4
5
```

Without the newline character, it would print in a row, but the next prompt would appear on the same line:

```
>> fprintf('%d', vec)
    2345>>
```

However, in a script, a separate newline character could be printed to avoid this problem. It is also much better to separate the numbers with spaces.

```
printvec.m
% This demonstrates printing a vector
vec = 2:5;
fprintf('%d ',vec)
fprintf('\n')
>> printvec
2 3 4 5
```

If the number of elements in the vector is known, that many conversion characters can be specified and then the newline:

```
>> fprintf('%d %d %d %d\n', vec)
     2 3 4 5
```

This is not very general, however, and is therefore not preferable. For matrices, MATLAB unwinds the matrix column by column. For example, consider the following (2*3) matrix:

```
>>mat = [5 9 8; 4 1 10]
mat =
     5 9 8
     4 1 10
```

Specifying one conversion character and then the newline character will print the elements from the matrix in one column. The first values printed are from the first column, then the second column, and so on.

```
>> fprintf('%d\n', mat)
5
4
9
1
8
10
```

If three of the %d conversion characters are specified, the ***fprintf*** will print three numbers across on each line of output, but again the matrix is unwound column-by-

column. It again prints first the two numbers from the first column (across on the first row of output), then the first value from the second column, and so on.

```
>> fprintf('%d %d %d\n', mat)
        5 4 9
        1 8 10
```

## 5. Scripts With Input And Output

Putting all of this together now, we can implement the algorithm from the beginning of this chapter. The following script calculates and prints the area of a circle. It first prompts the user for a radius, reads in the radius, and then calculates and prints the area of the circle based on this radius.

circleIO.m

```
% This script calculates the area of a circle
% It prompts the user for the radius
% Prompt the user for the radius and calculate
% the area based on that radius
fprintf('Note: the units will be inches.\n')
radius = input('Please enter the radius: ');
area = pi * (radius^2);
% Print all variables in a sentence format
fprintf('For a circle with a radius of %.2f inches,\n',...
radius)
fprintf('the area is %.2f inches squared\n',area)
```

Executing the script produces the following output:

```
>> circleIO
Note: the units will be inches.
Please enter the radius: 3.9
For a circle with a radius of 3.90 inches,
the area is 47.78 inches squared
```

Note that the output from the first two assignment statements (including the input) is suppressed by putting semicolons at the end. That is usually done in scripts, so that the exact format of what is displayed by the program is controlled by the ***fprintf*** functions.

---

---

---

---