**Input**

Input into a MATLAB program is achieved using the **input** function. A character vector, which is the prompt, is passed, and the user’s input should be stored in a variable. For example, to read in a number:

>> favnum = input('Please enter your fave num: ')

Please enter your fave num: 33

favnum =

33

The user could also be prompted for a vector, but would have to know the proper syntax.

>> urvec = input('Enter a vector: ')

Enter a vector: [4 8 2 9]

urvec =

4 8 2 9

>> urvec = input('Enter a vector: ')

Enter a vector: 4:5:25

urvec =

4 9 14 19 24

If the user is to enter a character or a character vector, a second argument is passed to the **input** function, 's'.

>> favpunct = input('Enter your fave punctuation mark: ', 's')

Enter your fave punctuation mark: !

favpunct =

'!'

>> icolor = input('What is your eye color: ', 's')

What is your eye color: brown

icolor =

'brown'

It is not possible to read directly into a string variable. However, a character vector can be converted to a string.

>> stricolor = string(icolor)

stricolor =

"brown"

The **input** function can only read in one thing at a time. Multiple desired inputs means multiple calls to the **input** function. (Note that a vector is considered one thing.)

**Output**

MATLAB has two basic output functions: **disp** and **fprintf**. The **disp** function is a quick way of displaying an expression, but it does not allow any formatting. It does, however, always print a newline character at the end. The **fprintf** function allows custom formatting of the output. Vectors and matrices are best displayed using **disp**.

Here are some examples using **disp**:

>> disp(4e2)

400

>> disp('Hi there!')

Hi there!

>> disp("Hello")

Hello

>> disp(3:7)

3 4 5 6 7

>> disp(eye(3))

1 0 0

0 1 0

0 0 1

The **fprintf** function allows formatting. For example, the following prints the number 33 in a sentence. Two arguments are passed to the **fprintf** function: a ***format string***, and the number 33. The %d in the format string is a ***place holder***, meaning that whatever comes after the format string fills in in that location. The newline character \n at the end moves the cursor down.

>> fprintf('The number is %d.\n', 33)

The number is 33.

>>

Without the newline character, the next prompt would end up on the same line as what was printed. It’s still a prompt, but it looks messy!

>> fprintf('The number is %d.', 33)

The number is 33.>> 5 - 2

ans =

3

The basic place holders are %d for integers (decimal integers), %f for floats, %c for single characters, and %s for character vectors or strings. The format string can be either a character vector or a string.

>> fprintf("The character is '%c'!!\n", 'x')

The character is 'x'!!

>> fprintf("The word is %s!!\n", 'fun')

The word is fun!!

>> fprintf('The word is "%s"!!\n', "fun")

The word is "fun"!!

>> fprintf('The number is %f, I think.\n', 11.11)

The number is 11.110000, I think.

The field width, number of decimal places (for floats) and justification can be specified.

The field width can be specified by putting an integer in between the % and the character in the place holder. For example, to print an integer in a field width of 5:

>> fprintf('The number is %5d!!\n', 33)

The number is 33!!

A negative integer will left-justify instead of right.

>> fprintf('The number is %-5d!!\n', 33)

The number is 33 !!

For floats, the number of decimal places can be specified. For example, a place holder of %6.2f prints in a field width of 6 altogether, including the decimal point and 2 decimal places, so in the format xxx.xx.

>> fprintf('The real number is %6.2f!!\n', 12.3)

The real number is 12.30!!

The field width does not need to be specified. If just the number of decimal places is specified, the field width will be set according to how wide the actual number is. For example,

>> fprintf('The cost was $%.2f, wow.\n', 123.456)

The cost was $123.46, wow.

If more than one expression is to be printed, there will be multiple place holders in the format string.

>> fprintf('Int: %d, Char: %c\n', 33, 'x')

Int: 33, Char: x

**Scripts**

We have seen that commands, statements, and expressions can be entered interactively one at a time in the Command Window. It is also possible to group them together into a script, and then have MATLAB execute the statements in the script sequentially. From the Home tab, you can click on the “New Script” icon, which will bring up a new Editor window. Once the statements have been entered in the Editor, from the Editor tab, you can click on Save to save the file. Code files including simple scripts are stored in files with the extension of .m on the name.

For example, the following was entered in the Editor, and saved in a file named ‘myscript1.m’.

myscript1.m

% This script calculates and prints the area of a rectangle

disp('Enter the length and width of a rectangle in inches.')

rlength = input('Enter the length: ');

rwidth = input('Enter the width: ');

fprintf('The area is %.2f\n', rlength \* rwidth)

The first line in the file is a comment. Comments are anything from a % to the end of that line.

To execute the script, type the name of the code file (without the .m) at the prompt in the Command Window.

>> myscript1

Enter the length and width of a rectangle in inches.

Enter the length: 4.2

Enter the width: 3.3

The area is 13.86

To view the script from the Command Window, the **type** command can be used.

>> type myscript1

% This script calculates and prints the area of a rectangle

disp('Enter the length and width of a rectangle in inches.')

rlength = input('Enter the length: ');

rwidth = input('Enter the width: ');

fprintf('The area is %.2f\n', rlength \* rwidth)

**Selection Statements**

There are several selection statements in MATLAB, including the **if** statement, **if-else** statement (with optional **elseif** clause), and the **switch** statement.

The **if** statement chooses whether an action is executed or not. The general form is:

if condition

action

end

The condition is a Boolean expression that is either true or false. If the condition is true, the action is executed – otherwise it is not executed. The action can be any number of statements up to the key word **end**. The action should be indented to make it easy to see.

For example, if a number should be positive, an if statement would change it if it is negative.

if num < 0

num = abs(num);

end

The **if-else** statement chooses between two actions. The general form is:

if condition

action1

else

action2

end

Again, the condition is a Boolean expression. If the value of the condition is true, the first action (action1) is executed. If instead the condition is false, the second action (action2) is executed. The actions are naturally bracketed by the reserved words **else** and **end**. One, and only one, of the actions will be executed. Which one depends on the value of the condition.

For example,

rint = randi([1, 10]);

if rint <= 5

fprintf('Lower half\n')

else

fprintf('Upper half\n')

end

If multiple actions are desired based on multiple conditions, nested **if-else** statements can be used with the optional **elseif** clause. The general form is:

if condition1

action1

elseif condition2

action2

elseif condition3

action3

else % can have many more elseif

actionn

end

The **switch** statement is used when you want to test a variable to see whether it is equal to different values, with different actions for each. For example, if you have a variable x and want to see whether it is equal to a, b, or c, the code could be written using an **if-else** statement, or a **switch** statement. The algorithms would be:

if x == a

action1

elseif x == b

action2

elseif x == c

action3

else

action4

end

switch x

case a

action1

case b

action2

case c

action3

otherwise

action4

end

The **switch** statement has **case** labels; the value of the variable is compared to the values on the case labels. If none of them are equal, the **otherwise** clause handles “none of the above”.

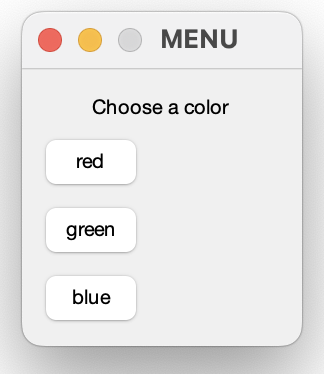
The **switch** statement does not add any power to the language, but may be easier to read.

One application of the **switch** statement is to choose different actions from a menu. MATLAB has a **menu** function that graphically presents the user with options.

For example, the following **menu** function asks the user to choose a color.

>> choice = menu('Choose a color', 'red', 'green', 'blue')

This brings up a box in that looks like this:



The first character vector passed to menu is a prompt that appears at the top. The others are labels that go on buttons. If the user pushes the first button (labeled ‘red’), the value returned from the **menu** function is 1, if the user pushes the second button the value returned is 2, if the user pushes the third button the value is 3, and if the user pushes the red circle, the value returned is 0. Once the value is returned and stored in the variable choice, a **switch** statement could then do different actions based on the user’s choice.

choice = menu('Choose a color', 'red', 'green', 'blue')

switch(choice)

case 1

fprintf('Your chose red!')

case 2

fprintf('You chose green!')

case 3

fprintf('You chose blue!')

otherwise

fprintf('Sadly, you made no choice.')

end

Instead of the **otherwise** clause, in this example a case label of 0 could be used.

**Simple Plots**

There are many simple plot types in MATLAB. Creating and annotating a plot is generally an iterative process, until the plot looks the way you want it to. For this reason, it is usually easiest to write a script to create a plot rather than doing it one line at a time in the Command Window.

To begin, create vectors storing the data points. Plots can be created programmatically, as in the following. This creates an x vector ranging from -pi to 2pi, using **linspace** which will by default create 100 elements. The y vector is sin(x). Both assignments are suppressed. Then, the plot function plots the vectors as points using the color blue and the marker ‘\*’. The plot is then annotated with labels on the x and y axes and a title on top.

x = linspace(-pi, 2\*pi);

y = sin(x);

plot(x,y, 'b\*')

xlabel('x')

ylabel('sin(x)')

title('sin(x) from -pi to 2pi')

This brings up a Figure Window that contains the following plot.



The plot can be modified from the Figure Window, for example under Insert you can insert x and y axis labels and a title directly on the plot.

It is also possible to create plots without using the functions. You can choose variables in the Workspace Window, and then under the Plots tab you can click on a plot type.