Contents

SIMPLE INPUT & OUTPUT

So far we have learned how to perform basic calculations and create variables to store values. However, all variables created thus far have been hard-coded a value to setting the variable equal to some value using the equals sign inside the *command window* or in an M-File. It is often desirable to allow your program to read in values during execution, and then use these values to perform calculations inside your code. Additionally, we want to be able to display the results of our calculations in a concise and meaningful way. Two simple input and output commands commonly used in MATLAB are discussed in the following sections.

Printing to the Screen (*Command Window*) Using the `disp()` Command

If you want to print out the value(s) of a variable that are currently stored in memory to the screen, you simply can type the variable name at the command line (without using a semicolon to suppress the output). The variable name and its value(s) will then be displayed in the *command window*.

```
>> x = 5

x =

     5
```

The same method can be used for variables that are character strings or other data types.

```
>> z = 'hello'

z =

hello
```

You can also write character and numerical data on the same line, but it will not print out that way. (Don't forget to enclose all character data in single quotes!)

```
>> 'The value of x is: ', x

ans =

The value of x is:


x =

     5
```

This does not look very good and is probably not the output we were looking for. We would like to have more control with the output so that we can present data in a clear and meaningful way. Specifically, we would like to get rid of that annoying `x =`, `z =`, and `ans =` after every statement is executed.

The `disp()` command is useful if you want to print something to the screen without fancy formatting. For example, let's define a variable called `score`, using a semicolon at the end of the line to suppress the output to the *command window*. When you terminate a line with a semicolon, it prevents any output from being printed to the *command window*. However, the variable is still defined (stored in memory) so that we can access and use it later.

```
>> score = 90.2;
>> disp('Your score is: ')
Your score is:
>> disp(score)
   90.2000
```

Here, we defined a variable and suppressed the output with a semicolon, but the variable was still stored in memory so that we can use the `disp()` command to output the value in a more clear way. However, our output is still spread over multiple lines. If you want to print different types of data (e.g., character and floating-point) on the same line using a single `disp()` command, you must convert the floating-point data to a string using the `num2str()` command. This is done as follows:

```
>> score = 90.2;
>> disp(['Your test score is: ', num2str(score)])
Your test score is: 90.2
```

Now the output is clear, neat, and labeled. It is important that you appropriately label all output from your code so that the user (anyone who uses your code, which could be you, your boss, a co-worker, the instructor of MAE10, etc.) knows exactly what the value they are seeing represents. In general, this means labeling the output (e.g., saying that it is a test score) and including units whenever applicable (there are no units for a test score). The importance of including units cannot be overemphasized in engineering. Remember, a number missing units can be easily misinterpreted. Notice also the use of

brackets `[]` around everything you want printed to the screen. Including brackets in addition to parenthesis is required when you include more than one argument in the `disp()` command. Here are some important points to remember when using the `disp()` command:

- The use of brackets is **required** when the `disp()` command has more than one argument (i.e., more than one variable, a variable and a character string, etc.). This is shown in the example above, where we display the character string `Your test score is:` and the variable `score` using a single `disp()` command. If you do not use the brackets when they are required, MATLAB will produce an error.
- Using brackets is **optional** when the `disp()` command has only one argument (e.g., just one variable, just a character string, etc.). The example at the top of the page shows the `disp()` command with just one argument at a time. No brackets are used, since they are not required. However, we could still use brackets even if there is only one argument since they are optional in this case – this would not produce an error.
- You **must** use `num2str()` (on the number) when printing both a number(s) and a character string(s) in a single `disp()` command. Not doing so will not produce an error, but it will produce unexpected output. This is because floating point data (numbers) must be converted to character strings when printing both data types in a single `disp()` command.
    - It is never necessary to use `num2str()` on a character string.
    - It is not necessary to use `num2str()` when printing only numbers in the `disp()` command (since you are not mixing variable types).

Based on these set of rules you should realize one thing: Technically it is safest to **always** use brackets in the `disp()` command if you are in doubt.

Getting Input From the User Using the `input()` Command

Sometimes you want a program to read in data provided by the user and then use that data to perform some calculations. For example, the user may input some number of hours and the program converts that number to minutes and seconds. To read in data from the screen (*command window*) that is input by the user, use the `input()` command in the following way:

```
variable = input('some instructions/prompt for the user')
```

Note that the prompt that is given to the user in the `input()` command must be enclosed in single quotes just like all other character strings in MATLAB. Let's look at a sample M-File that uses the input command to allow the user to input any number of hours, and displays the equivalent number of minutes and seconds.

In `myfile.m`:

```
hours = input('Enter the number of hours: ') % MATLAB will wait for a value
minutes = hours * 60;
seconds = minutes * 60;
disp(['This is ' num2str(minutes) ' minutes'])
disp(['This is ' num2str(seconds) ' seconds'])
```

Sample output:

```
>> myfile
Enter the number of hours: 5    (MATLAB waits here until I type a value and press enter)

hours =

     5

This is 300 minutes
This is 18000 seconds
```

When the `input()` command is executed MATAB will pause the execution of the code until the user types a value and presses enter. This value is then stored in the variable to the left of the equals sign (`hours`) and the code proceeds with execution. If we wanted to prevent the variable `hours` from being repeated back to use after the user inputs a value, we can terminate the `input()` command with a semicolon.

Inputting character data is a little trickier. If you use the same syntax for inputting character data as you use for inputting numerical data, the user must put single quotes around the text they enter so that MATLAB recognizes the text as a character string.

In `myfile.m`:

```
name = input('What is your name? ');
disp(['You have entered: ', name])
```

Sample output:

```
>> myfile
What is your name? 'Peter'      (I had to use single quotes when entering 'Peter')
You have entered: Peter
```

If you don't want the user to have to enter character data in single quotes, you can use a slightly different form of the `input()` command designed specifically for inputting character data. The only change to the `input()` command is that we include a comma and the letter `s` in single quotes after the prompt. For example:

In `myfile.m`:

```
% The 's' tells MATLAB that the input is a character string (not a #)
name = input('What is your name? ','s');
disp(['You have entered: ' name])
```

At command line:

```
>> myfile
What is your name? Peter
You have entered: Peter
```

Including the `'s'` tells MATLAB that the input from the user will be a character string, so the user does not need to enclose the input in single quotes. We will discuss how to enter in multiple pieces of information at once later.

To summarize, here are some key points for the `input()` command:

- Be sure to always have a variable to the left-hand side of the equals sign when using the `input()` command – this is the variable that will store (save in memory) any value(s) that are input so that you can use them later.
- Be sure to enclose your prompt for the `input()` command in single quotes (since it is a character string).
- When inputting numbers, **do not** include the `'s'` in the `input()` command.
- When inputting character strings, **always** include the `'s'` in the `input()` command.
- You can prevent any data that is input from being repeated to the user (e.g., re-printed in the *command window*) by terminating the `input()` command with a semicolon as in the above examples. Once again, using a semicolon only prevents the information from being printed in the *command window* – the variable is still stored in memory.

While the `input()` command may seem simple, it is actually very important conceptually. Remember that a computer is just a machine that reads, stores, and manipulates data to provide the desired output. The only way a computer can read data is through some form of input from the user – your computer's mouse and keyboard are input devices, for example. The mic on your phone is an input device – if you give it a verbal command, it reads this input, manipulates it (e.g., by converting the frequency of your voice into words for a search engine), and provides the desired output (such as the weather in Irvine or directions to your destination). While in MATLAB you will primarily be inputting numbers or simple character strings, the idea is the same. You provide an input and a set of instructions (your code), and MATLAB provides you with an output (the result of your calculations).