

MATLAB Workshop 5 - Introduction to Scripts

Objectives: Learn about scripts and input/output to the screen.

MATLAB Features:*simple screen input/output*

Command	Action
<code>disp('text')</code>	displays indicated text
<code>disp(var_name)</code>	displays only value of <code>var_name</code>
<code>var_name = input('request for value ==> ');</code>	displays request message to screen assigns input value to <code>var_name</code> semicolon at end suppresses echo display
<code>str_name = input('request for string ==> ','s');</code>	displays request message to screen assigns input string to <code>str_name</code> semicolon at end suppresses echo display

formatted output display

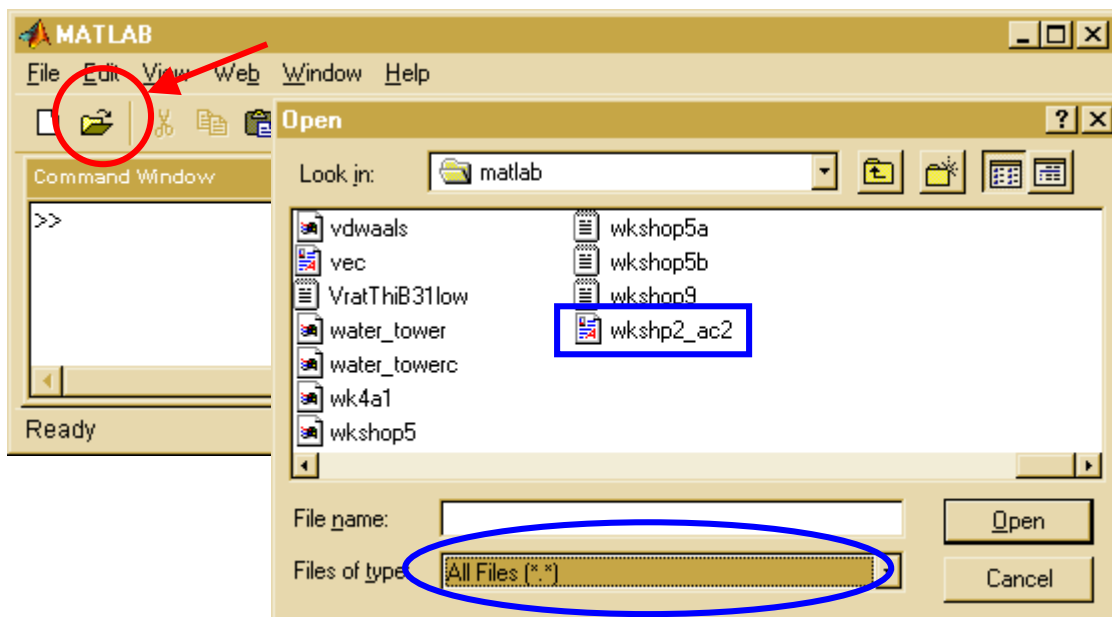
Command	Action
<code>fprintf('control string', values)</code>	displays listed values in order listed at placeholder positions in control string text
Placeholders	
<code>%wd</code>	<code>d</code> : Integer occupying <code>w</code> spaces.
<code>%w.dF</code>	<code>f</code> : Floating point with <code>w</code> spaces, <code>d</code> decimal places
<code>%w.de</code>	<code>e</code> : Exponential display with <code>w</code> spaces, <code>d</code> decimal places
<code>%s</code>	<code>s</code> : String
Carriage control	
<code>\n</code>	new line

- **Basics of scripts in MATLAB**

So far we have been using MATLAB as a (very elaborate) calculator. That is, basically everything that we have done we could also have done on a handheld calculator. If that were all that MATLAB is capable of doing, it would not be a very useful engineering or problem solving tool. The real power of MATLAB lies in the ability to create *scripts* (programs) that can perform the same task many times with minimal revisions.

We have already seen the basics of developing a script in MATLAB in Workshop 2 where we used MATLAB to solve a particular engineering problem. A problem solving approach was used to generate a step-by-step method to find the desired results. Creating a script to do the same involves exactly the same kind of approach. A more detailed description of a problem solving approach that will help you get to the right answers and also to develop scripts that will produce the right answers can be found in Background 1: Top Level Program Design.

A diary file makes a good starting place from which to talk about scripts. We will use the diary file `wkshp2_ac2` that was created as an annotated solution to Activity 2 in Workshop 2. To open this file in the MATLAB Editor (as opposed to the MATLAB Command Window), click on the open file icon in the MATLAB Command Window. This brings up the Open File Window with the current

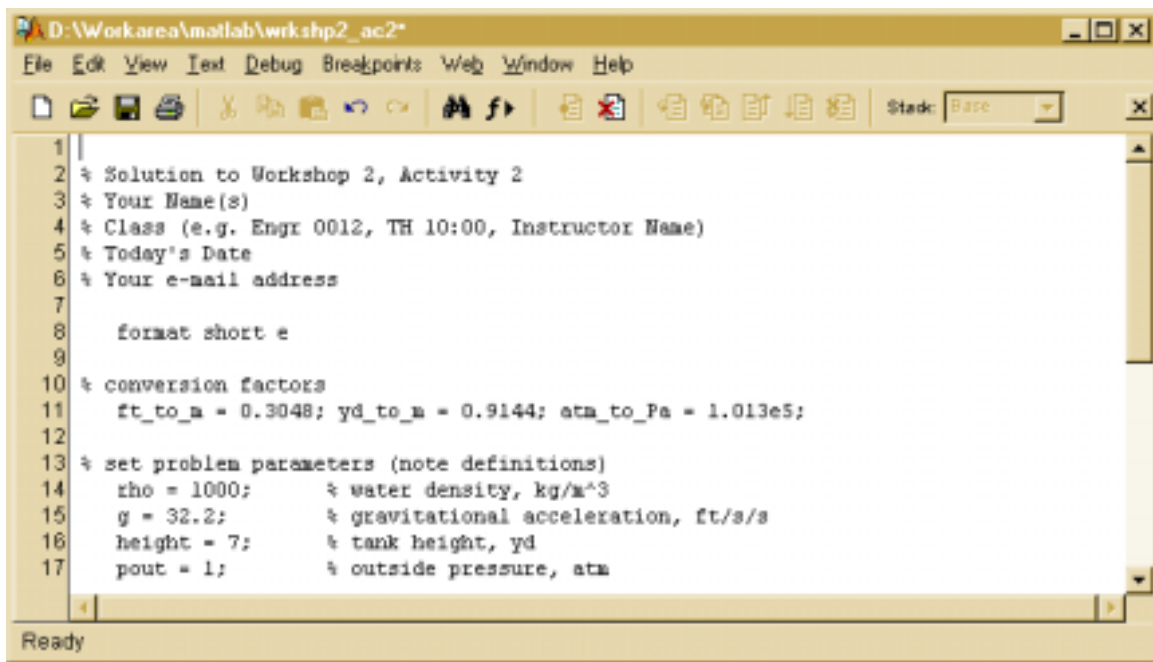


directory contents. If necessary, change the Look in: directory to the one which holds the desired file. Then change files of type: to All Files (*.*) and locate the desired file (`wkshp2_ac2`). Double click on the file to bring it into the MATLAB Editor as shown at the top of the next page.

The MATLAB Editor is a general purpose editor that has some features that are special to MATLAB. The file that has just been imported is simply a text file not a MATLAB script or **m-file**. **m-files** are easily recognized in the MATLAB Editor because they are color-coded. To see this, save the file that has just been opened as an **m-file** (File, Save as, add `.m` extension to the file name). `.m` is a MATLAB specific file extension that tells the MATLAB Command Window and MATLAB Editor that the file contains MATLAB code. Note the shift in screen display; most noticeably, **comments are now displayed in green**.

Scroll down through the file. All of the lines that were entered as part of Workshop 2, Activity 2 should be there. In particular, note the diary is broken into neatly visible sections:

- A header section that provides information about the file and its creators;
- A “constant” parameter section that provides conversion constants;
- A section that provides given values for parameters (variables);



```

1 |
2 % Solution to Workshop 2, Activity 2
3 % Your Name(s)
4 % Class (e.g. Engr 0012, TH 10:00, Instructor Name)
5 % Today's Date
6 % Your e-mail address
7
8     format short e
9
10 % conversion factors
11     ft_to_m = 0.3048; yd_to_m = 0.9144; atm_to_Pa = 1.013e5;
12
13 % set problem parameters (note definitions)
14     rho = 1000;      % water density, kg/m^3
15     g = 32.2;      % gravitational acceleration, ft/s/s
16     height = 7;    % tank height, yd
17     pout = 1;      % outside pressure, atm

```

- A section that converts given values to SI units; and
- A section that calculates and displays the desired result.

A diary file can provide the basis for developing a workable script. In fact, the file is easily modified to a not very useful script. Edit the file to change the first line to **Workshop 5, Activity 1**, and remove the displayed answer and **diary off** command at the end of the file. Save the edited file as **wkshp5_ac1.m**. Now enter

```

>> clear
>> wkshp5_ac1
pabs_b_si =
    1.6412e+005

```

MATLAB will search for a **.m** file with the stated name (which it will find in the current directory), run all of the commands in the file, and respond with the answer. Note that the **.m** extension was not needed in the command. Although the **m-file** you just created works as a MATLAB script, it is not very useful because it merely reiterates work that you have already done. Worse, you have no idea what happened - only the answer appears.

- **Elements of scripts**

All MATLAB scripts in this text will follow the same basic outline and have the same basic elements. In particular, all scripts will

- Start with a comment (header) section that identifies the purpose of the script and provides information about the creator(s) of the script;
- Contain a variable dictionary that identifies all variables used in the script (with units);
- Have an optional **format** command;
- Display the header information to the screen whenever the script is executed;
- Ask the user to enter variables/values in an user-friendly manner;
- Do whatever computations are required (unit conversions, calculations); and
- Display the results in an user-friendly manner.

(1) m-file header section.

All **m-files** will start with a header section that contains information about the purpose of the file and the creators of the file as comments. The generic header section used in this text is

```
% File purpose
%   as many comment lines as needed to clearly describe
% Your Name(s)
% Class (e.g. Engr 0012, TH 10:00, Instructor Name)
% Date created or last updated
% Your e-mail address
```

In addition to providing documentation within the file itself, the header lines are displayed in the Command Window by the command `help filename`. For example, enter

```
>> help wkshp5_ac1
Solution to Workshop 5, Activity 1
Your Name(s)
Class (e.g. Engr 0012, TH 10:00, Instructor Name)
Today's Date
Your e-mail address
```

(2) Variable dictionary.

The next section in your script should be a comment section containing the variable dictionary, separated from the header section by a blank line. The variable dictionary identifies all of the variables used in the script together with units. Add a variable dictionary to `wkshp5_ac1.m` in the Editor by adding these lines following a blank line after the header section.

```
% Variable dictionary
% rho          water density, kg/m^2
% g            gravitational acceleration, ft/s/s
% height       tank height, yd
% pout         outside pressure, atm
% rho_si       water density in SI units, kg/m^2
% g_si         gravitational acceleration in SI units, m/s/s
% height_si    tank height in SI units, m
% pout_si      outside pressure in SI units, Pa
% pabs_b_si    pressure at tank bottom in SI units, Pa
```

The purpose of the variable dictionary is to document all of the variables that you are using. Complete documentation of a program or script is a (required) necessity. It tells you (years later) what you meant when you developed the script. It also tells others who happen to be using your work what you did. Both of these are exceedingly important if revisions or alterations to the script need to be made or if you want to adapt the script for use in another application.

(3) Optional **format** command.

As you have learned, the `format` command can be used to control the screen display of numbers. If you are happy with the default display for the purposes of your script, then no change in format is required.

(4) Display header information.

When you executed `wkshp5_ac1` above, only the answer appeared. This is unacceptable for a program or script. A script should provide information to the user about what it does whenever it is invoked. The `disp` command in MATLAB provides a convenient way to display such information. Add the following section after a blank line following the variable dictionary in `wkshp5_ac1` in the Editor. (Hint: use copy and paste to copy the header section to the new location and modify it appropriately).

```
% display header to screen
clc
disp(' ')
disp('Solution to Workshop 5, Activity 1')
disp('Your Name(s)')
disp('Class (e.g. Engr 0012, TH 10:00, Instructor Name)')
disp('Today''s Date')
disp('Your e-mail address')
disp(' ')
```

The `clc` command clears the Command Window and resets the cursor and command line to the top. This does not affect Workspace or the diary if it is running (all previous commands will still be in the diary). Use of this command clears away any clutter that you may have in the Command Window that could distract from appreciating the results of running your script.

Another example of color coding in the Editor is seen: strings are displayed in dull red. A **string** is a sequence of letters and characters (including numbers). A string is denoted in MATLAB by containing it between two single quotes, `' '`. Thus, each of the arguments in the `disp` command above is a separate string.

- A strings is a sequence of letters, characters, and numbers.
- Strings are identified (declared) by containing the sequence between single quotes, `' '`, in MATLAB.
- Note how a single quote can be displayed in a string.

(5) Gathering information in a user-friendly manner.

Another problem with `wkshp5_ac1` as it stands is that it is *hardwired*. That is, it only works for the values that are declared in the script. What if your instructor changes the tank dimensions? While it is possible to edit the script to enter the new values, it would be more convenient if the script would ask for the values it needs. Then, once the script is operational, it would work for any sets of values your instructor desires (with minimal additional effort on your part).

Getting information from the user in MATLAB is a two-step process. First, the script must ask for the information. Second, the script must grab the information the user enters and make sure that it is assigned to the appropriate variable. MATLAB has a single command that can be used to accomplish both tasks - the `input` command. The general form of the input command is

```
var_name = input( 'request for value ==> ' )
```

An appropriate request is contained in single quotes as the argument to `input`. `input` has variations on the theme as noted here:

The `input` command

- `var_name = input('request for value ==> ')` will get a numeric value and assign it to `var_name`. The input will be displayed as no semicolon was used to suppress display.
- `var_name = input('request for value ==> ');` will get a numeric value and assign it to `var_name`. The input will not be displayed because a semicolon was used to suppress display.
- `str_name = input('request for string ==> ','s')` will get a string and assign it to the string variable `str_name`. The input will be displayed as no semicolon was used to suppress display.
- `str_name = input('request for string ==> ','s');` will get a string and assign it to the string variable `str_name`. The input will not be displayed as a semicolon was used to suppress display.
- **Note** the use of `==>` (with spaces on either side) to highlight where value is entered at keyboard.

Modify the `% set problem parameters` section of `wkshp5_ac1` as follows:

```
% get problem values
rho = input('Please input fluid density in kg/m^3 ==> ');
g = input('Please input grav accel in ft/s/s ==> ');
height = input('Please input tank height in yd ==> ');
pout = input('Please input atm pressure in atm ==> ');
```

This makes the problem about as general as possible in that it works for different fluids in the tank (water versus oil versus gasoline), different locations (Earth versus Mars), different tank dimensions, and different atmospheric conditions. Note that because the variables have been previously defined in the variable dictionary, comments identifying the variable and units are no longer necessary at this point.

Also note that the `input` command string is a request for information that specifies the units that must be associated with the value. A good practice is to use `==>` (with a space between the arrow and the closing single quote) after the request. This identifies where the input value is placed when typed at the keyboard. It also makes your code easier to read and use.

(6) Unit conversions/calculations.

The unit conversion section remains unchanged, other than that the comments identifying the variable names and units can be deleted now that a variable dictionary is in use.

The calculation of pressure at the bottom of the tank remains unchanged, except that a semicolon is placed at the end to suppress display at this time. We want the answer to display, but we want the answer to display in a better annotated manner than is done by the MATLAB default.

(7) Displaying the script results.

We are now ready to display the results of our calculations in a easily readable (user-friendly) manner. To do this, we use the `disp` command. Enter the following section after the calculations in `wkshp5_ac1`

```
% display results
disp(' ');
disp('Pressure at the bottom of the tank, Pa');
disp( pabs_b_si );
```

Annotation identifying the value to follow (with units) comes first. The value is then displayed. Using the `disp` command displays only the value of the variable requested without displaying the variable name. (There is no reason why a user should need to know what you have chosen to use as variable names when only a properly identified correct result is important).

The `disp` command

- `disp('This is a string')` displays the string enclosed in single quotes.
- `disp(var_name)` displays only the value without the variable name.

(8) Running the script.

At this point in time, the contents of the MATLAB Editor should match the contents of the text box at the end of this Workshop. Note the use of comments, blank lines, and indenting to make the script easier to read. Save the contents and run the script in the MATLAB Command Window.

(9) Formatted output display.

While sequential use of the `disp` command provides rudimentary control of the output display, it is not really user-friendly in that the displays are still controlled by MATLAB, take more lines than really necessary, and allow only one type of information, string or value, on the output line. MATLAB also has available a formatted output command, `fprintf`, which has the general form

```
fprintf('control string', value list )
```

where the control string consists of text and placeholders. Placeholders are used to identify where the values in the value list are to be placed in the text and the format for display of the values. For example, the display commands in Section 7 could be replaced with

```
% display results
fprintf('\nPressure at the bottom of the tank, Pa: %10.4f', ...
        pabs_b_si);
```

The control string is contained in single quotes. The value to be displayed is separated from the control string by a comma. The `\n` highlighted in red in the control string is a carriage control that tells MATLAB to start the output on a new line on the screen. The placeholder always starts with a `%` and has the form

```
%w.dtype
```

where `w` is the total number of spaces to use to display the output, `d` is the number of decimal places to display, and `type` is the type of display. In the example above, the value is to be displayed with 4 decimal places using 10 total spaces as a floating point (decimal) number. Other types of display are noted in the table at the right.

Formatted output placeholders	
Placeholder	Meaning
<code>%wd</code>	<code>d</code> : Integer occupying <code>w</code> spaces.
<code>%w.df</code>	<code>f</code> : Floating point with <code>w</code> spaces, <code>d</code> decimal places
<code>%w.de</code>	<code>e</code> : Exponential display with <code>w</code> spaces, <code>d</code> decimal places
<code>%s</code>	<code>s</code> : String

Try replacing the display commands in your script with the formatted display command and rerun your script. Which display do you like better?

- **Exercises:**

1. Create a script corresponding to `wkshp2_ac3`. Name it `wkshp5_ex1.m`.
2. Create a script corresponding to `wkshp2_ac4`. Name it `wkshp5_ex2.m`.
3. Create a script corresponding to `wkshp2_ex1`. Name it `wkshp5_ex3.m`.
4. Create a script corresponding to `wkshp2_ex2`. Name it `wkshp5_ex4.m`.
5. Create a script corresponding to `wkshp2_ex3`. Name it `wkshp5_ex5.m`.
6. Create a script corresponding to `wkshp2_ex4`. Name it `wkshp5_ex6.m`.

Recap: You should have learned

- The basic ideas behind scripts and script organization.
- That MATLAB scripts are saved as **m-files**.
- That all scripts must have a header section.
- That all scripts must have a variable dictionary.
- That all scripts must display a header to the screen when run.
- How to use the `disp` command to display strings and values to the screen.
- How to use the `input` command to get information from the user.
- How to create user-friendly displays.


```
% Solution to Workshop 5, Activity 1
% Your Name(s)
% Class (e.g. Engr 0012, TH 10:00, Instructor Name)
% Today's Date
% Your e-mail address

% Variable dictionary
% rho          water density, kg/m^2
% g            gravitational acceleration, ft/s/s
% height       tank height, yd
% pout         outside pressure, atm
% rho_si       water density in SI units, kg/m^2
% g_si         gravitational acceleration in SI units, m/s/s
% height_si    tank height in SI units, m
% pout_si      outside pressure in SI units, Pa
% pabs_b_si    pressure at tank bottom in SI units, Pa

% set display format
format short e

% display header to screen
clc
disp(' ')
disp('Solution to Workshop 5, Activity 1')
disp('Your Name(s)')
disp('Class (e.g. Engr 0012, TH 10:00, Instructor Name)')
disp('Today's Date')
disp('Your e-mail address')
disp(' ')

% conversion factors
ft_to_m = 0.3048; yd_to_m = 0.9144; atm_to_Pa = 1.013e5;

% get problem values
rho = input('Please input fluid density in kg/m^3 ==> ');
g = input('Please input grav accel in ft/s/s ==> ');
height = input('Please input tank height in yd ==> ');
pout = input('Please input atm pressure in atm ==> ');

% convert all parameters to SI
rho_si = rho;
g_si = ft_to_m*g;
height_si = yd_to_m*height;
pout_si = atm_to_Pa*pout;

% calculate Pabs,b in Pa
pabs_b_si = rho_si*g_si*height_si+pout_si;

% display results
disp(' ')
disp('Pressure at the bottom of the tank, Pa' )
disp(pabs_b_si )
```