Introduction to MATLAB

MATLAB (short for MATrix LABoratory) is a special-purpose computer program optimized to perform engineering and scientific calculations. It started life as a program designed to perform matrix mathematics, but over the years it has grown into a flexible computing system capable of solving essentially any technical problem.

The MATLAB program implements the MATLAB programming language and provides a very extensive library of predefined functions to make technical programming tasks easier and more efficient.

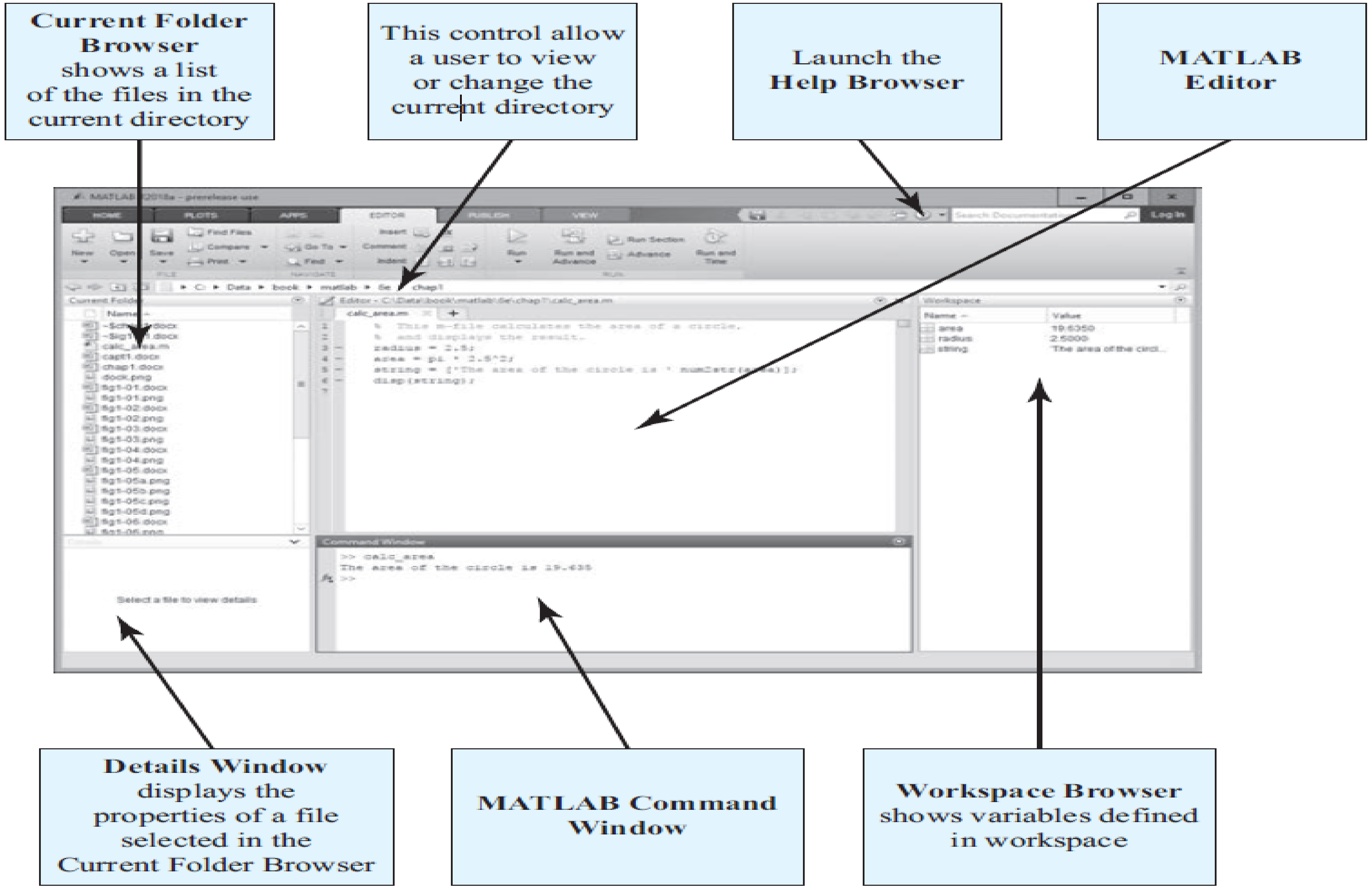
MATLAB is a huge program with an incredibly rich variety of functions. Even the basic version of MATLAB without any toolkits is much richer than other technical programming languages. There are more than 1000 functions in the basic MATLAB product alone, and the toolkits extend this capability with many more functions in various specialties. Furthermore, these functions often solve very complex problems (solving differential equations, inverting matrices, and so forth) in a single step, saving large amounts of time.

**The MATLAB Desktop

When you start MATLAB, a special window called the MATLAB desktop appears. The desktop is a window that contains other windows showing MATLAB data, plus toolbars and a “Toolstrip” or “Ribbon Bar” similar to that used by Windows 10 or Microsoft Office.

By default, most MATLAB tools are “docked” to the desktop, so that they appear inside the desktop window. However, the user can choose to “undock” any or all tools, making them appear in windows separate from the desktop.

The default configuration of the MATLAB desktop is shown in Figure 1. It integrates many tools for managing files, variables, and applications within the MATLAB environment.


Current Folder Browser shows a list of the files in the current directory
This control allow a user to view or change the current directory
Launch the Help Browser
MATLAB Editor
Details Window displays the properties of a file selected in the Current Folder Browser
MATLAB Command Window
Workspace Browser shows variables defined in workspace
Select a file to view details

Editor - C:\Data\book\matlab\6e\chap1\calc_area.m

```
% This m-file calculates the area of a circle,
% and displays the result.
radius = 2.5;
area = pi * 2.5^2;
string = ['The area of the circle is ' num2str(area)];
disp(string);
```

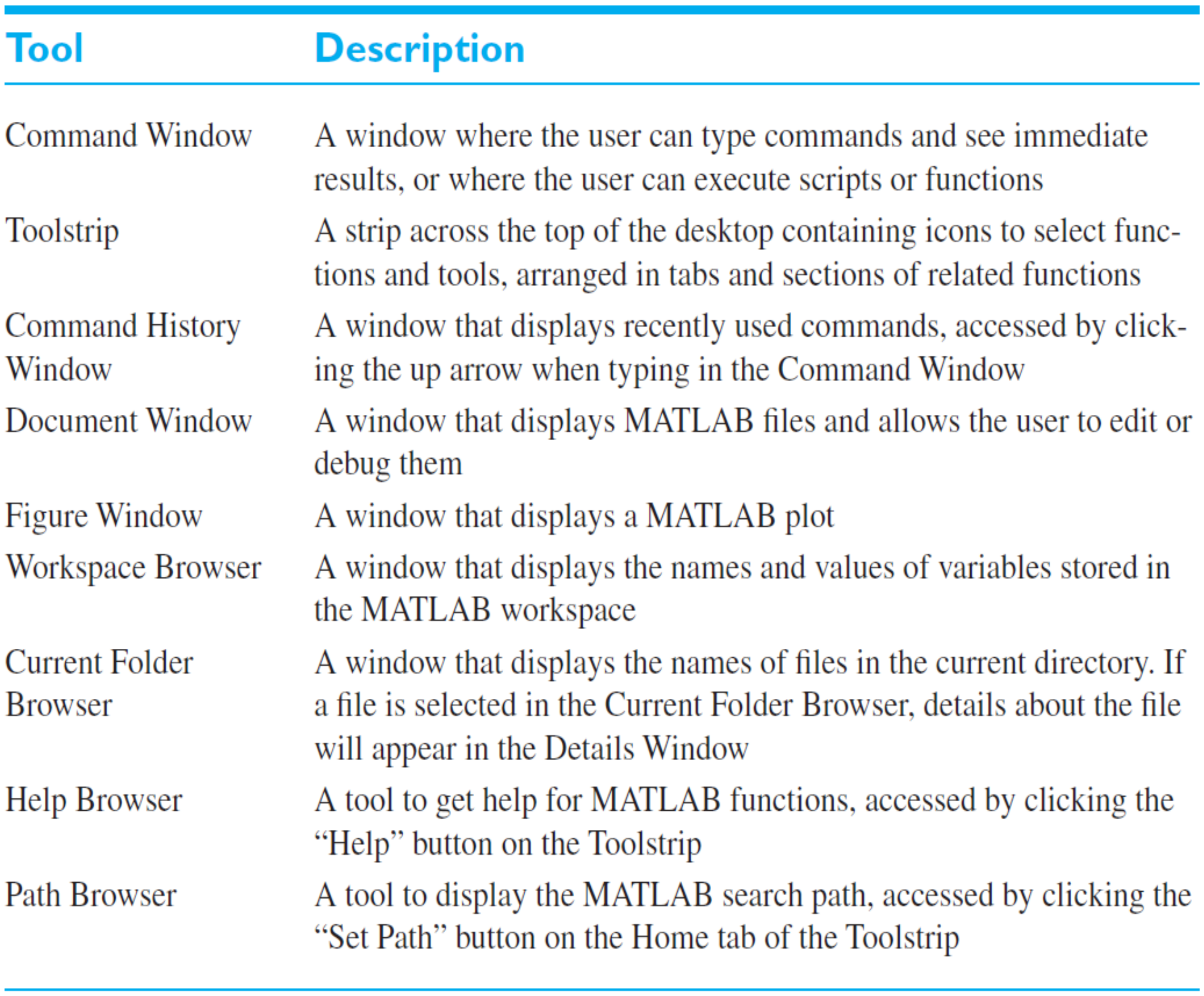
Command Window

```
>> calc_area
The area of the circle is 19.635
>>
```

Workspace

Name	Value
area	19.6350
radius	2.5000
string	'The area of the circl...

Figure 1 The default MATLAB desktop.



Tool	Description
Command Window	A window where the user can type commands and see immediate results, or where the user can execute scripts or functions
Toolstrip	A strip across the top of the desktop containing icons to select functions and tools, arranged in tabs and sections of related functions
Command History Window	A window that displays recently used commands, accessed by clicking the up arrow when typing in the Command Window
Document Window	A window that displays MATLAB files and allows the user to edit or debug them
Figure Window	A window that displays a MATLAB plot
Workspace Browser	A window that displays the names and values of variables stored in the MATLAB workspace
Current Folder Browser	A window that displays the names of files in the current directory. If a file is selected in the Current Folder Browser, details about the file will appear in the Details Window
Help Browser	A tool to get help for MATLAB functions, accessed by clicking the "Help" button on the Toolstrip
Path Browser	A tool to display the MATLAB search path, accessed by clicking the "Set Path" button on the Home tab of the Toolstrip

Table 1: Tools and Windows Included in the MATLAB Desktop

The functions of these tools are summarized in Table 1. We will discuss them.

**The Command Window

The bottom center of the default MATLAB desktop contains the **Command Window**. A user can enter interactive commands at the command prompt (») in the Command Window, and they will be executed on the spot.

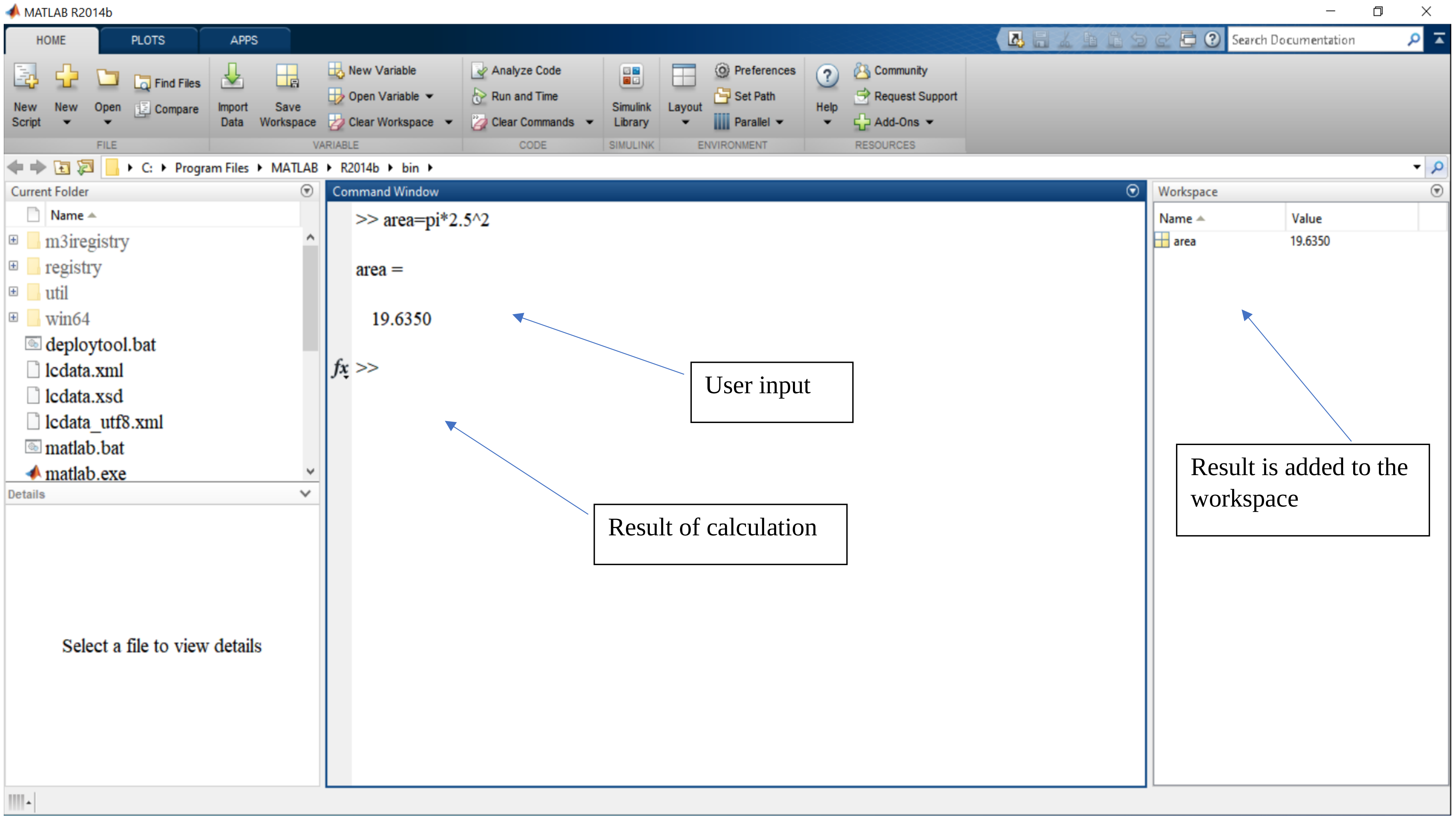
As an example of a simple interactive calculation, suppose that you wanted to calculate the area of a circle with a radius of 2.5 m. The equation for this area of a circle is:

$$A = \pi \ r^2$$

where *r* is the radius of the circle and *A* is the area of the circle. This equation can be evaluated in the MATLAB Command Window by typing:

```
» area = pi * 2.5^2

area =

19.6350
```

where * is the multiplication symbol and ^ is the exponential symbol. MATLAB calculates the answer as soon as the Enter key is pressed, and stores the answer in a variable (really a 1 X 1 array) called area. The contents of the variable are displayed in the Command Window as shown in Figure 2, and the variable can be used in further calculations. (Note that p is predefined in MATLAB, so we can just use pi without first declaring it to be 3.141592 …).



```
>> area=pi*2.5^2

area =

   19.6350

>>
```

Figure 2 : The Command Window appears in the center of the desktop. You enter commands and see responses here.

**The Toolstrip

The Toolstrip (see Figure 1.3) is a bar of tools that appears across the top of the desktop. The controls on the Toolstrip are organized into related categories of functions, first by tabs, and then by groups. For example, the tabs visible in Figure 1.3 are “Home”, “Plots”, “Apps”, “Editor”, and so forth. When one of the tabs is selected, a series of controls grouped into sections is displayed. In the Home tab, the sections are “File”, “Variable”, “Code”, and so forth. With practice, the logical grouping of commands helps the user to quickly locate any desired function. In addition, the upper-right corner of the Toolstrip contains the Quick Access Toolbar, which is where you can customize the interface and display the most commonly used commands and functions at all times. To customize the functions displayed there, right-click on the toolbar and select the Customize option from the popup menu.

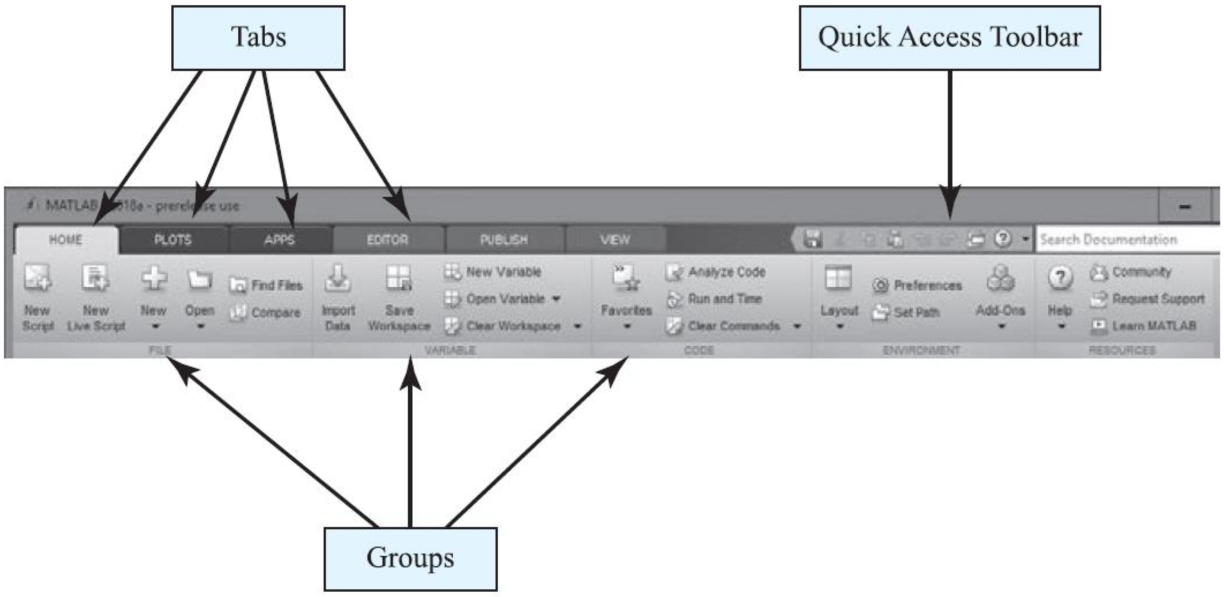
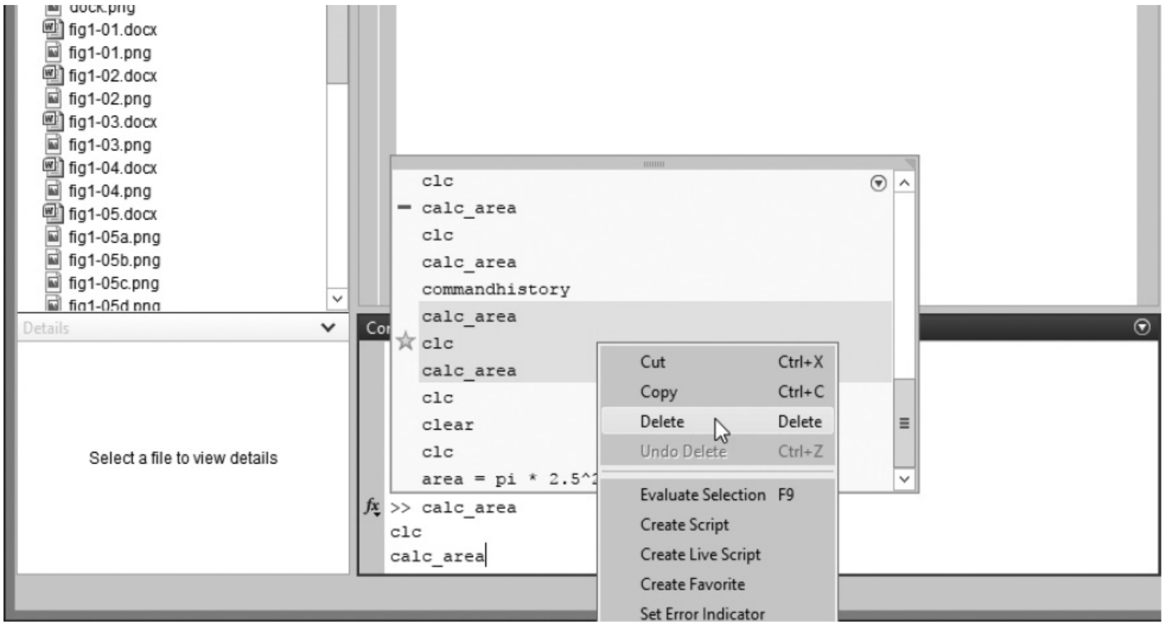


Figure 3 : The Toolstrip

**The Command History Window

The Command History Window displays a list of the commands that a user has previously entered in the Command Window. The list of commands can extend back to previous executions of the program. Commands remain in the list until they are deleted. To display the Command History Window, press the up arrow key while typing in the Command Window. To reexecute any command, simply double-click it with the left mouse button. To delete one or more commands from the Command History Window, select the commands and right-click them with the mouse. A popup menu will be displayed that allows the user to delete the items (see Figure 4).



```
clc
calc_area
clc
calc_area
commandhistory
calc_area
clc
calc_area
clc
clear
clc
area = pi * 2.5^2
```

```
>> calc_area
clc
calc_area
```

Figure 4: The Command History Window, showing three commands being deleted.

**The Document Window

A Document Window (also called an Edit/Debug Window) is used to create new M-files or to modify existing ones. An Edit/Debug Window is created automatically when you create a new M-file or open an existing one. You can create a new M-file with the “New Script” command from the “File” group on the Toolstrip (Figure 5a), or by clicking the New icon and selecting Script from the popup menu (Figure 5b). You can open an existing M-file file with the Open command from the “File” section on the Toolstrip.

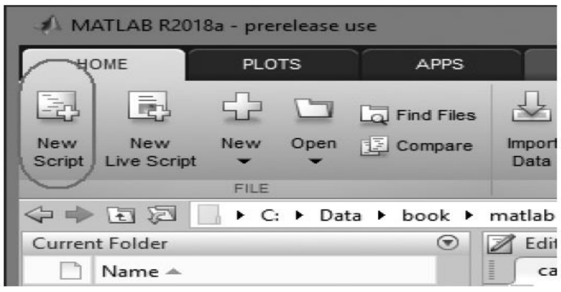
An Edit/Debug Window displaying a simple M-file called calc_area.m is shown in Figure 5. This file calculates the area of a circle given its radius and displays the result. By default, the Edit Window is docked to the desktop, as shown in Figure 5.

The Edit Window is essentially a programming text editor, with the MATLAB language’s features highlighted in different colors. Comments in an M-file file appear in green, variables and numbers appear in black, complete character strings appear in magenta, incomplete character strings appear in red, and language keywords appear in blue.

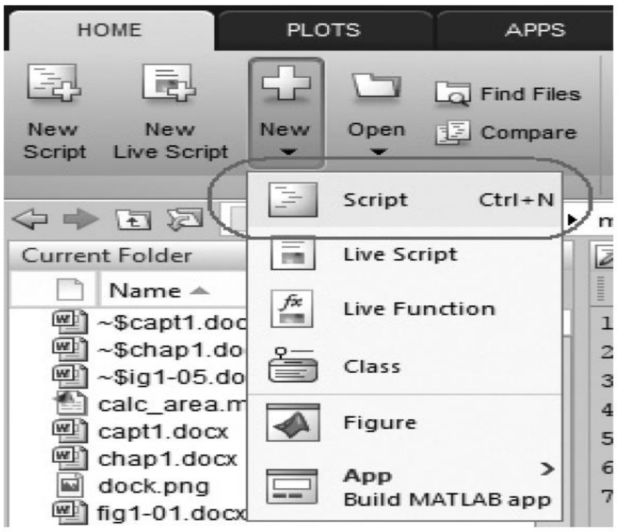
After an M-file is saved, it may be executed by typing its name in the Command Window. For the M-file in Figure 5, the results are:

```
» calc_area

area = 19.635
```

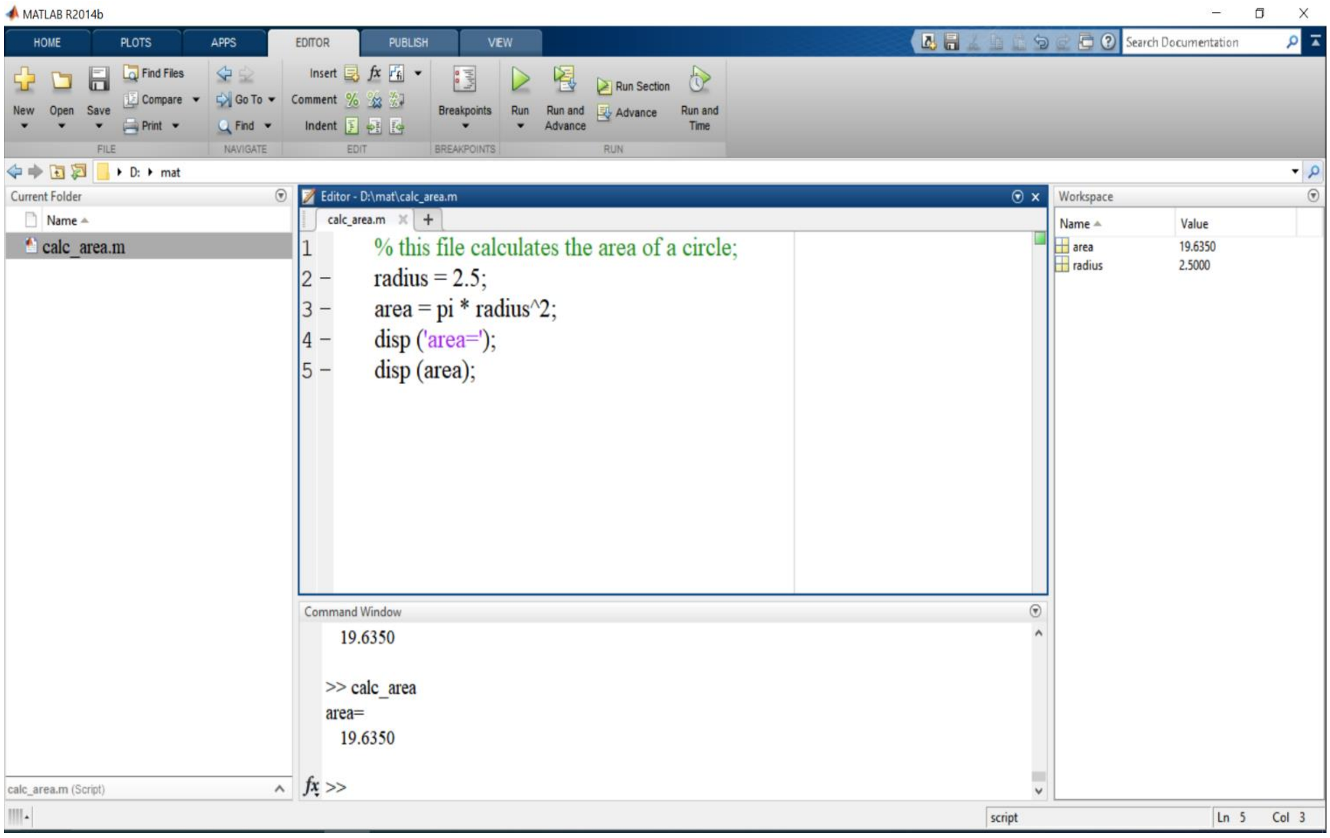
The Edit Window also doubles as a debugger.



(a)



(b)



```
% this file calculates the area of a circle;
radius = 2.5;
area = pi * radius^2;
disp ('area=');
disp (area);
```

```
  19.6350

>> calc_area
area=
  19.6350

fx >>
```

(c)

Figure 5: (a) Creating a new M-file with the "New Script" command. (b) Creating a new M-file with the "New >> Script" selection from the Toolbar. (c) The MATLAB Editor, docked to the MATLAB desktop. (See color insert.)

**Figure Windows

A Figure Window is used to display MATLAB graphics. A figure can be a two- or three-dimensional plot of data, an image, or a GUI. A simple script file that calculates and plots the function sin x is as follows:

% sin_x.m: This M-file calculates and plots the function sin(x) for
0 <= x <= 6.

x = 0:0.1:6

y = sin(x)

plot(x,y)

The resulting plot is shown in Figure 6.

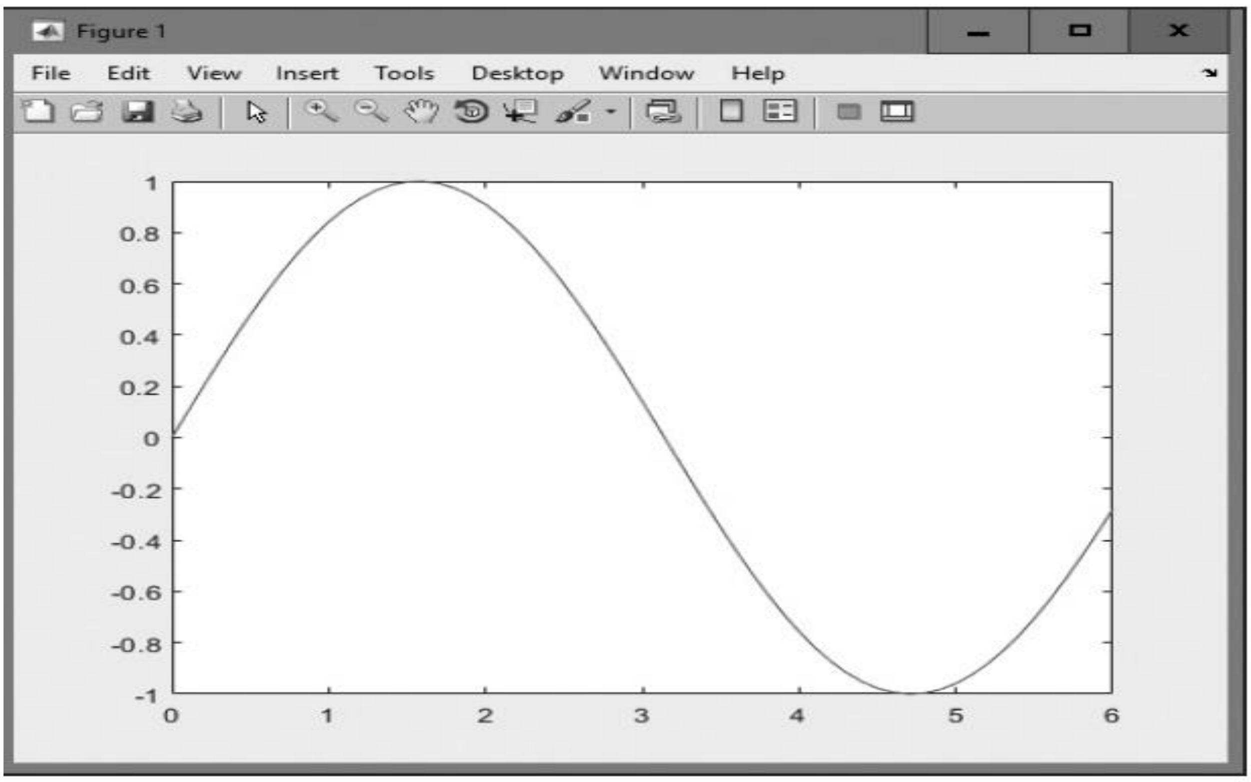
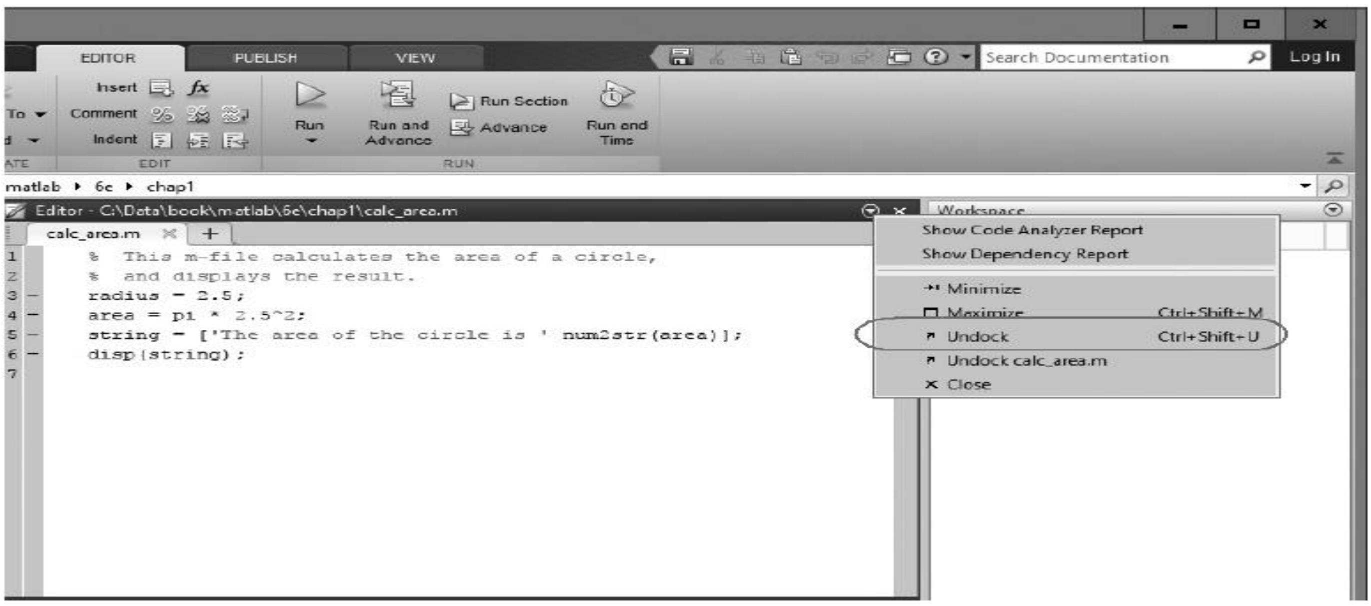


Figure 6 : MATLAB plot of sin x versus x.

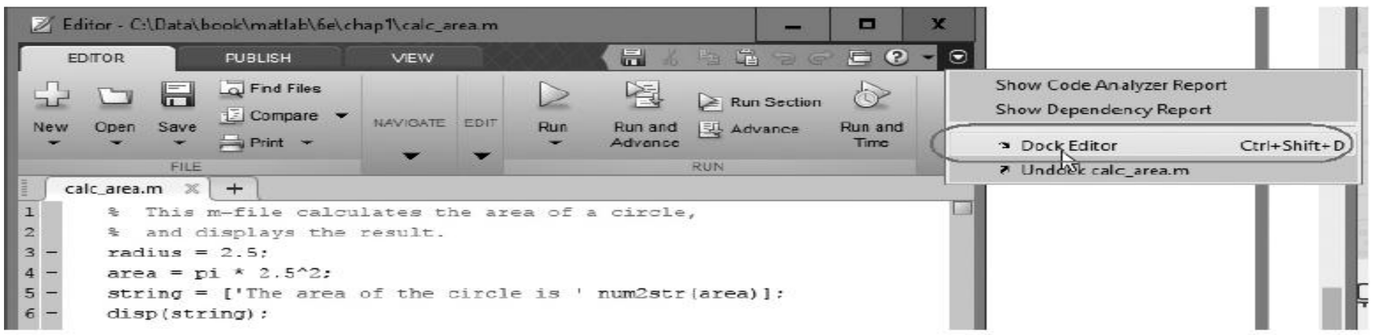
****Docking and Undocking Windows**

MATLAB windows such as the Command Window, the Edit/Debugging Window, and Figure Windows can either be docked to the desktop, or they can be undocked. When a window is docked, it appears as a pane within the MATLAB desktop. When it is undocked, it appears as an independent window on the computer screen separate from the desktop. When a window is docked to the desktop, it can be undocked by selecting the small down arrow in the upper-right corner and selecting the "Undock" option from the popup menu (see Figure 7a). When a window is an independent window, it can be docked to the desktop by selecting the small down arrow in the upper-right corner and selecting the "Dock" option from the popup menu (see Figure 7b).



```
% This m-file calculates the area of a circle,
% and displays the result.
radius = 2.5;
area = pi * 2.5^2;
string = ['The area of the circle is ' num2str(area)];
disp(string);
```

(a)



```
% This m-file calculates the area of a circle,
% and displays the result.
radius = 2.5;
area = pi * 2.5^2;
string = ['The area of the circle is ' num2str(area)];
disp(string);
```

(b)

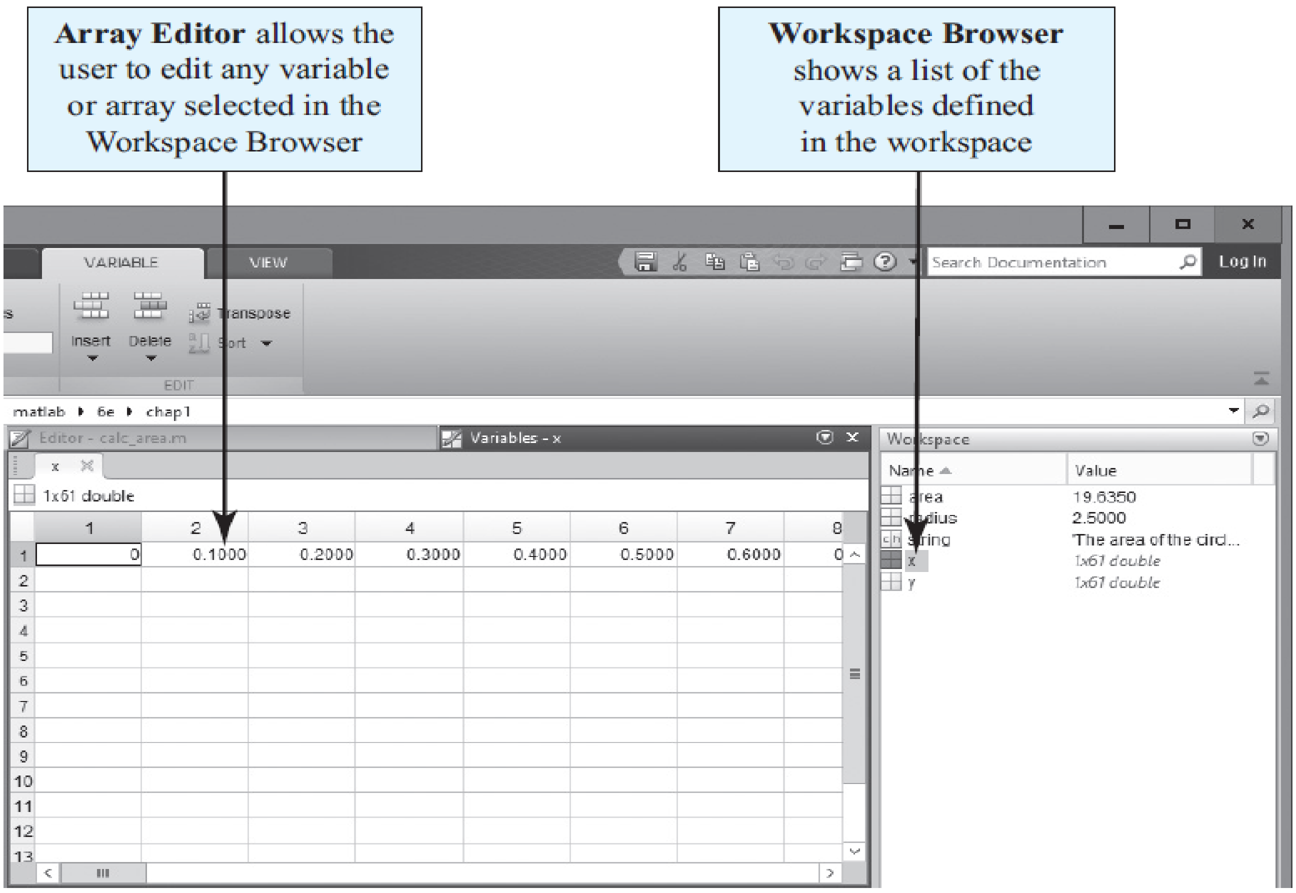
Figure 7 (a) Selecting the "Undock" option from the menu displayed after clicking the small down arrow. (b) Selecting the "Dock" option

** The MATLAB Workspace

A statement like

z = 10

creates a variable named z, stores the value 10 in it, and saves it in a part of computer memory known as the workspace. A workspace is the collection of all the variables and arrays that can be used by MATLAB when a particular command, M-file, or function is executing. All commands executed in the Command Window (and all script files executed from the Command Window) share a common workspace, so they can all share variables. As we will see later, MATLAB functions differ from script files in that each function has its own separate workspace. A list of the variables and arrays in the current workspace can be generated with the who’s command. See figure 8.


Array Editor allows the user to edit any variable or array selected in the Workspace Browser
Workspace Browser shows a list of the variables defined in the workspace
VARIABLE
VIEW
Search Documentation
Log In
Insert
Delete
Transpose
Sort
EDIT
matlab ▸ 6e ▸ chap1
Editor - calc_area.m
Variables - x
Workspace

x

1x61 double

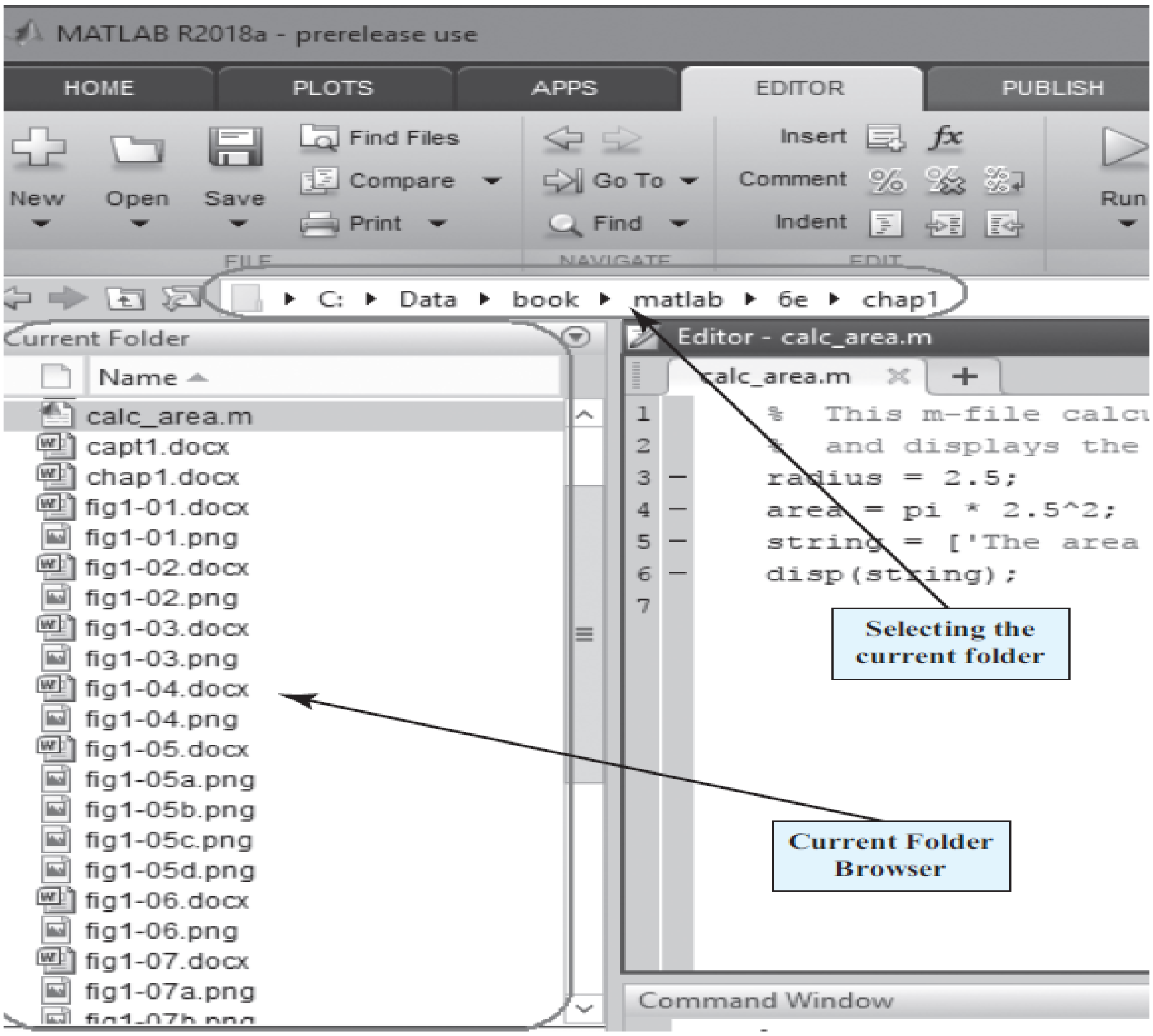
	1	2	3	4	5	6	7	8
1	0	0.1000	0.2000	0.3000	0.4000	0.5000	0.6000	0
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								

Name	Value
area	19.6350
radius	2.5000
string	'The area of the circl...
x	1x61 double
y	1x61 double

Figure 8 The Workspace Browser and Array Editor. The Array Editor is invoked by double-clicking a variable in the Workspace Browser. It allows

**The Current Folder Browser

The Current Folder Browser is displayed on the upper-left side of the desktop. It shows all the files in the currently selected folder, and allows the user to edit or execute any desired file. You can double-click on any M-file to open it in the MATLAB editor, or you can right-click it and select “Run” to execute it. The Current Folder Browser is shown in Figure 9. A toolbar above the browser is used to select the current folder to display.



```
calc_area.m
capt1.docx
chap1.docx
fig1-01.docx
fig1-01.png
fig1-02.docx
fig1-02.png
fig1-03.docx
fig1-03.png
fig1-04.docx
fig1-04.png
fig1-05.docx
fig1-05a.png
fig1-05b.png
fig1-05c.png
fig1-05d.png
fig1-06.docx
fig1-06.png
fig1-07.docx
fig1-07a.png
fig1-07b.png
```

```
%  This m-file calcu
%  and displays the
radius = 2.5;
area = pi * 2.5^2;
string = ['The area
disp(string);
```

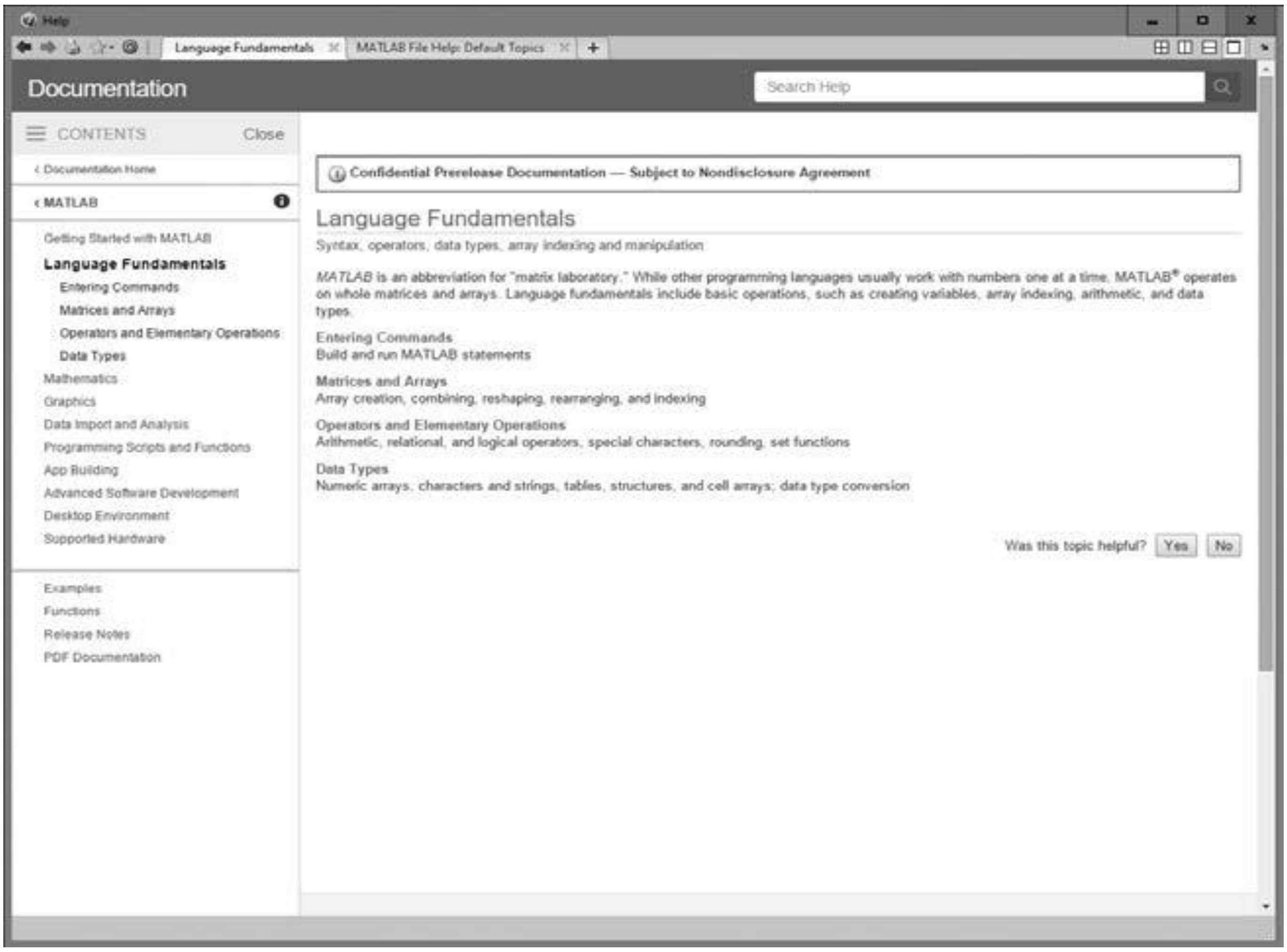
Figure 9 The Current Folder Browser.

** Getting Help

There are three ways to get help in MATLAB. The preferred method is to use the Help Browser. The Help Browser can be started by selecting the icon from the Toolstrip

or by typing **doc** or **helpwin** in the Command Window.

The Help Browser is shown in Figure 10.



Confidential Prerelease Documentation — Subject to Nondisclosure Agreement

Language Fundamentals

Syntax, operators, data types, array indexing and manipulation

MATLAB is an abbreviation for "matrix laboratory." While other programming languages usually work with numbers one at a time, MATLAB® operates on whole matrices and arrays. Language fundamentals include basic operations, such as creating variables, array indexing, arithmetic, and data types.

Entering Commands
Build and run MATLAB statements

Matrices and Arrays
Array creation, combining, reshaping, rearranging, and indexing

Operators and Elementary Operations
Arithmetic, relational, and logical operators, special characters, rounding, set functions

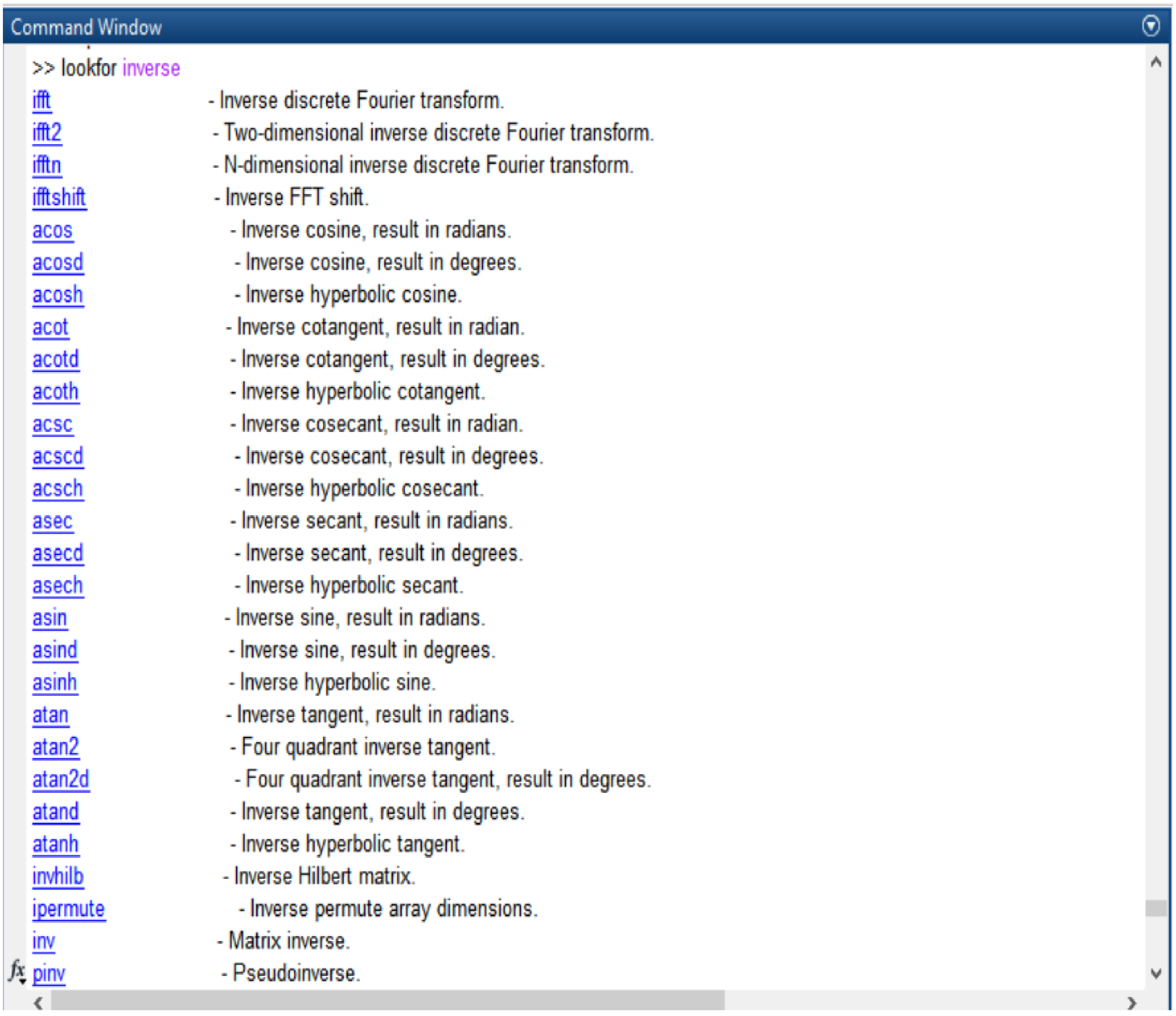
Data Types
Numeric arrays, characters and strings, tables, structures, and cell arrays; data type conversion

Figure 10 The Help Browser.

A user can get help by browsing the MATLAB documentation.

There are also two command-line-oriented ways to get help. The first way is to type help or help followed by a function name in the Command Window. If you just type **help**, MATLAB will display a list of possible help topics in the Command Window. If a specific function or a toolbox name is included, help will be provided for that particular function or toolbox.

The second way to get help is the **lookfor** command. The **lookfor** command differs from the help command in that the help command searches for an exact function name match, while the **lookfor** command searches the quick summary information in each function for a match. This makes **lookfor** slower than help, but it improves the chances of getting back useful information. For example, suppose that you were looking for a function to take the inverse of a matrix. Since MATLAB does not have a function named inverse, the command "help inverse" will produce nothing. On the other hand, the command "**lookfor inverse**" will produce the following results (show in figure 11)



```
>> lookfor inverse
ifft                     - Inverse discrete Fourier transform.
ifft2                    - Two-dimensional inverse discrete Fourier transform.
ifftn                    - N-dimensional inverse discrete Fourier transform.
ifftshift                - Inverse FFT shift.
acos                     - Inverse cosine, result in radians.
acosd                    - Inverse cosine, result in degrees.
acosh                    - Inverse hyperbolic cosine.
acot                     - Inverse cotangent, result in radian.
acotd                    - Inverse cotangent, result in degrees.
acoth                    - Inverse hyperbolic cotangent.
acsc                     - Inverse cosecant, result in radian.
acscd                    - Inverse cosecant, result in degrees.
acsch                    - Inverse hyperbolic cosecant.
asec                     - Inverse secant, result in radians.
asecd                    - Inverse secant, result in degrees.
asech                    - Inverse hyperbolic secant.
asin                     - Inverse sine, result in radians.
asind                    - Inverse sine, result in degrees.
asinh                    - Inverse hyperbolic sine.
atan                     - Inverse tangent, result in radians.
atan2                    - Four quadrant inverse tangent.
atan2d                   - Four quadrant inverse tangent, result in degrees.
atand                    - Inverse tangent, result in degrees.
atanh                    - Inverse hyperbolic tangent.
invhilb                  - Inverse Hilbert matrix.
ipermute                 - Inverse permute array dimensions.
inv                      - Matrix inverse.
pinv                     - Pseudoinverse.
```

Figure 11 lookfor inverse list.

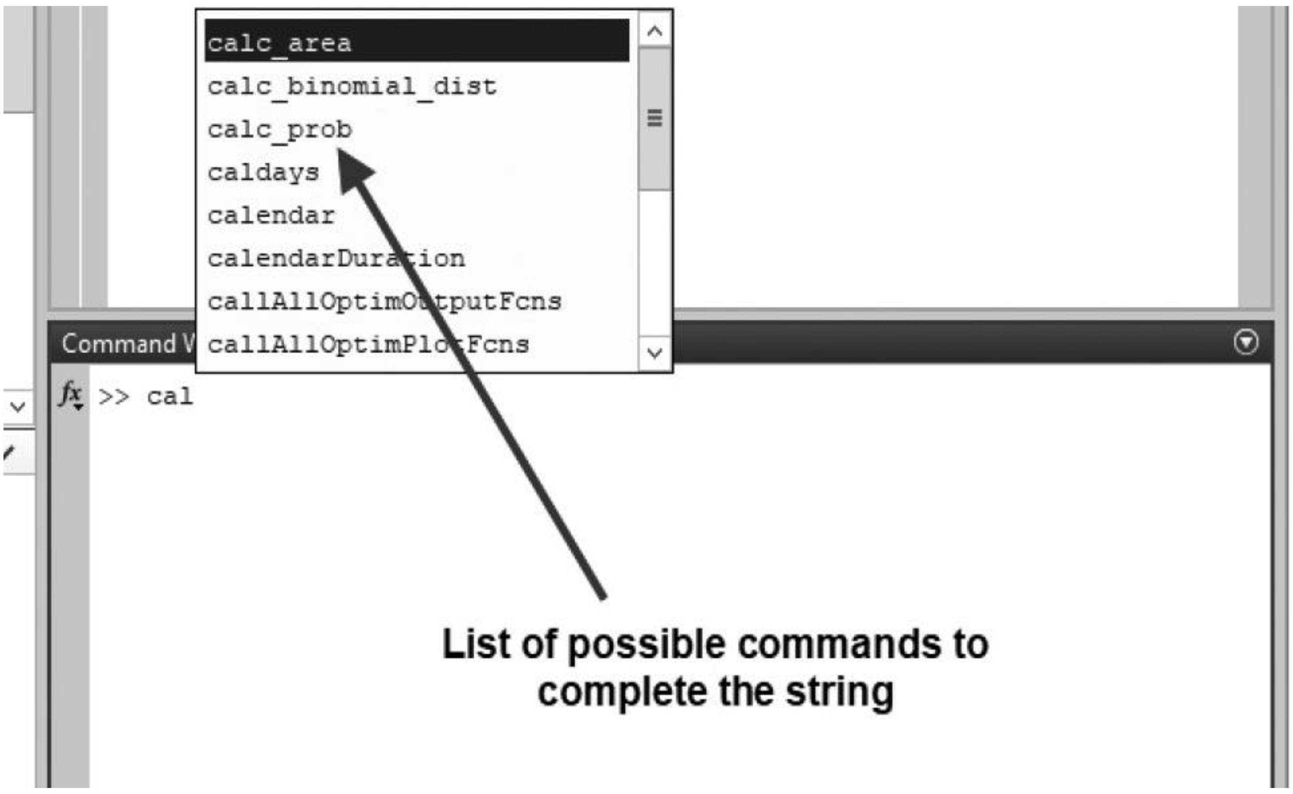
** A Few Important Commands

The contents of the Command Window can be cleared at any time using the **(clc)** command, and the contents of the current Figure Window can be cleared at any time using the **(clf)** command. The variables in the workspace can be cleared with the **(clear)** command.

If an M-file appears to be running for too long, it may contain an infinite loop, and it will never terminate. In this case, the user can regain control by typing (**control-c**).

There is also an auto-complete feature in MATLAB. If a user starts to type a command and then presses the Tab key, a popup list of recently typed commands and MATLAB functions that match the string will be displayed (see Figure 12). The user can complete the command by

selecting one of the items from the list



```
>> cal
calc_area
calc_binomial_dist
calc_prob
caldays
calendar
calendarDuration
callAllOptimOutputFcns
callAllOptimPlotFcns
```

Figure 12 If you type a partial command and then hit the Tab key, MATLAB will pop up a window of suggested commands or functions that match the string.

**The MATLAB Search Path

MATLAB has a search path that it uses to find M-files. MATLAB's M-files are organized in directories on your file system. Many of these directories of M-files are provided along with MATLAB, and users may add others. If a user enters a name at the MATLAB pompt, the MATLAB interpreter attempts to find the name as follows:

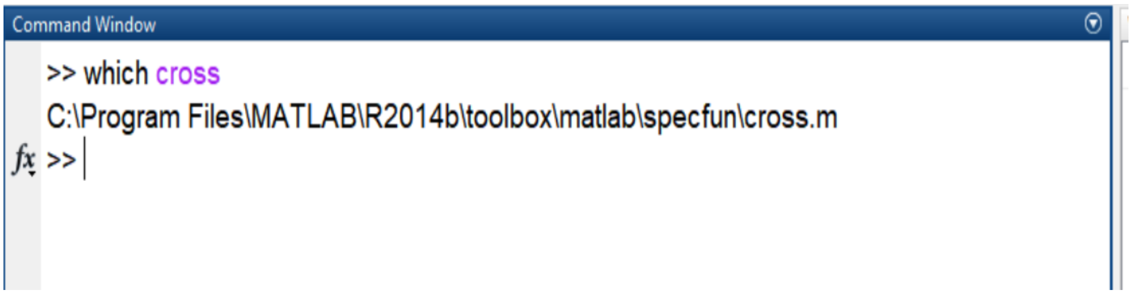
1. It looks for the name as a variable. If it is a variable, MATLAB displays the current contents of the variable.

2. It checks to see if the name is an M-file in the current directory. If it is, MATLAB executes that function or command.

3. It checks to see if the name is an M-file in any directory in the search path. If it is, MATLAB executes that function or command.

Note that MATLAB checks for variable names first, so if you define a variable with the same name as a MATLAB function or command, that function or command becomes inaccessible. This is a common mistake made by novice users.

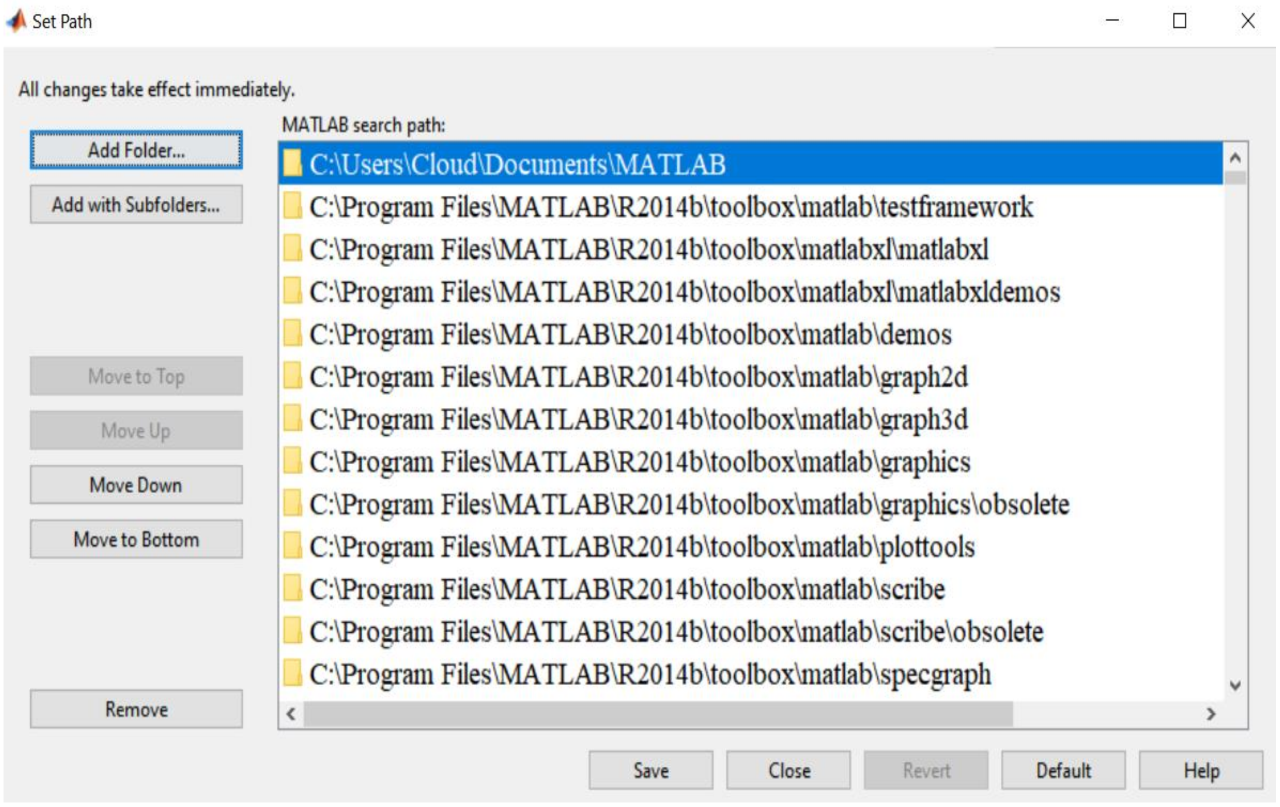
MATLAB includes a special command (which) to help you find out just which version of a file is being executed and where it is located. This can be useful in finding filename conflicts. The format of this command is which function name, where function name is the name of the function that you are trying to locate. For example, the cross-product function cross.m can be located as follows:



```
>> which cross
C:\Program Files\MATLAB\R2014b\toolbox\matlab\specfun\cross.m
>>
```

Figure 13 command window which cross

The MATLAB search path can be examined and modified at any time by selecting the “Set Path” tool from the Environment section of the Home tab on the Toolstrip, or by typing pathtool in the Command Window. The Path Tool is shown in Figure 14. It allows you to add, delete, or change the order of directories in the path.



```
C:\Users\Cloud\Documents\MATLAB
C:\Program Files\MATLAB\R2014b\toolbox\matlab\testframework
C:\Program Files\MATLAB\R2014b\toolbox\matlabxl\matlabxl
C:\Program Files\MATLAB\R2014b\toolbox\matlabxl\matlabxldemos
C:\Program Files\MATLAB\R2014b\toolbox\matlab\demos
C:\Program Files\MATLAB\R2014b\toolbox\matlab\graph2d
C:\Program Files\MATLAB\R2014b\toolbox\matlab\graph3d
C:\Program Files\MATLAB\R2014b\toolbox\matlab\graphics
C:\Program Files\MATLAB\R2014b\toolbox\matlab\graphics\obsolete
C:\Program Files\MATLAB\R2014b\toolbox\matlab\plottools
C:\Program Files\MATLAB\R2014b\toolbox\matlab\scribe
C:\Program Files\MATLAB\R2014b\toolbox\matlab\scribe\obsolete
C:\Program Files\MATLAB\R2014b\toolbox\matlab\specgraph
```

Figure 14 The Path Tool.

**Using MATLAB as a Calculator

In its simplest form, MATLAB can be used as a calculator to perform mathematical calculations. The calculations to be performed are typed directly into the Command Window, using the symbols +, -, *, /, and ^ for addition, subtraction, multiplication, division, and exponentiation, respectively. After an expression is typed, the results of the expression will be automatically calculated and displayed. If an equal sign is used in the expression, then the result of the calculation is saved in the variable name to the left of the equal sign.

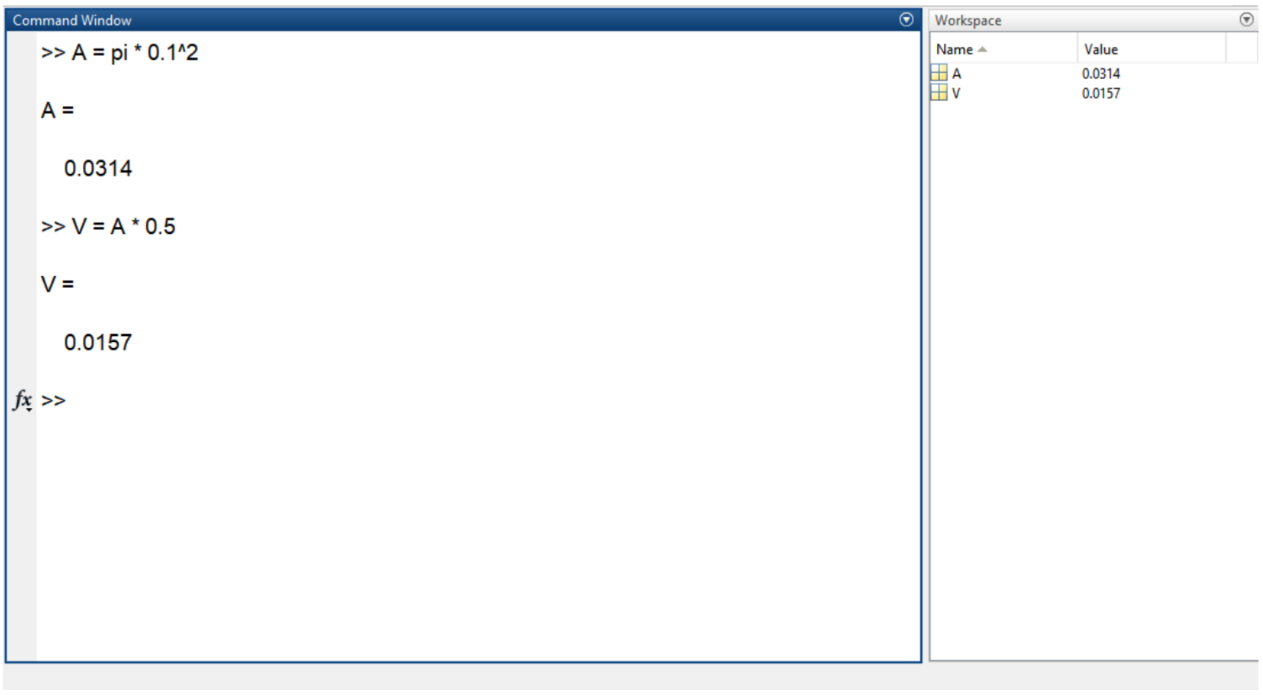
For example, suppose that we would like to calculate the volume of a cylinder of radius r and length l. The area of the circle at the base of the cylinder is given by the equation

$$A = \pi r^2$$

and the total volume of the cylinder will be

$$V = Al$$

If the radius of the cylinder is 0.1 m and the length is 0.5 m, then the volume of the cylinder can be found using the following MATLAB statements :



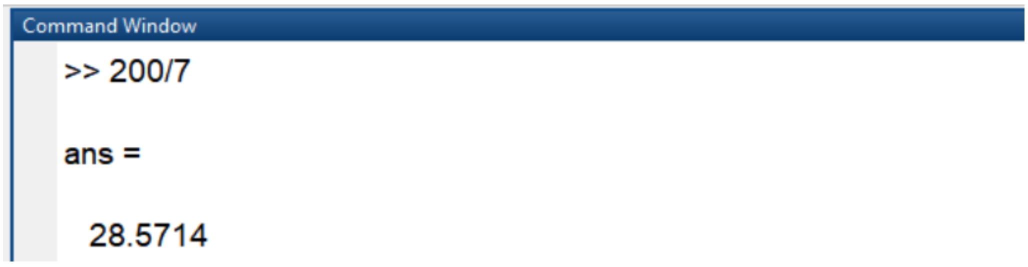
```
>> A = pi * 0.1^2

A =

  0.0314

>> V = A * 0.5

V =

  0.0157

fx >>
```

Name	Value
A	0.0314
V	0.0157

Note that pi is predefined to be the value 3.141592 … .

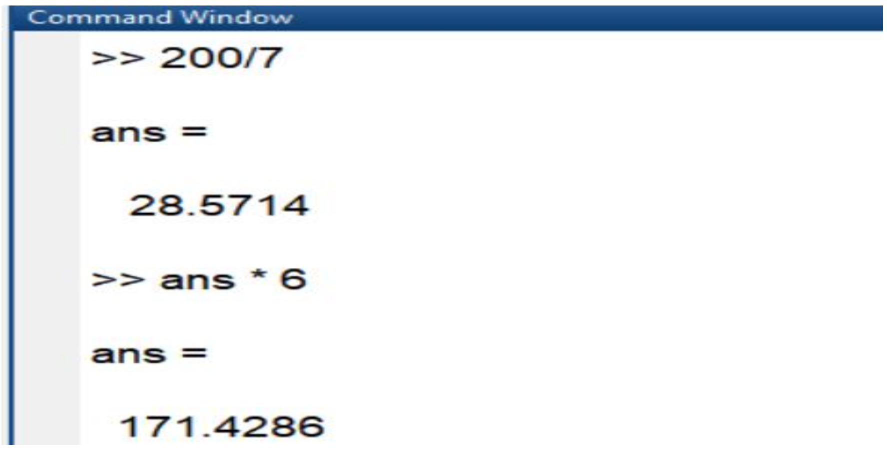
When the first expression is typed, the area at the base of the cylinder is calculated, stored in variable A, and displayed to the user. When the second expression is typed, the volume of the cylinder is calculated, stored in variable V, and displayed to the user. Note that the value stored in A was saved by MATLAB and reused when we calculated V.

If an expression without an equal sign is typed into the Command Window, MATLAB will evaluate it, store the result in a special variable called ans, and display the result.



```
>> 200/7

ans =

  28.5714
```

The value in ans can be used in later calculations, but be careful! Every time a new expression without an equal sign is evaluated, the value saved in ans will be overwritten.



```
>> 200/7

ans =

  28.5714

>> ans * 6

ans =

  171.4286
```

The value stored in ans is now 171.4286, not 28.5714.

If you want to save a calculated value and reuse it later, be sure to assign it to a specific name instead of using the default name ans.

** **Note**

The following summary lists all of the MATLAB special symbols, commands, and functions described

Special Symbols

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation

**Note

Predefined MATLAB functions can also be used in the calculations. A few common ones are given in Table. They can be combined with the basic addition, subtraction, multiplication, division, and exponentiation to evaluate mathematical equations.

Function	Description
Mathematical Functions	
`abs(x)`	Calculates the absolute value $\lvert x\rvert$.
`acos(x)`	Calculates $\cos^{-1}x$ (results in radians).
`asin(x)`	Calculates $\sin^{-1}x$ (results in radians).
`atan(x)`	Calculates $\tan^{-1}x$ (results in radians).
`cos(x)`	Calculates $\cos x$, with x in radians.
`log10(x)`	Calculates the logarithm to the base 10 $\log_{10}x$.
`sin(x)`	Calculates $\sin x$, with x in radians.
`sqrt(x)`	Calculates the square root of x.
`tan(x)`	Calculates $\tan x$, with x in radians.

For example, from basic trigonometry we know that the square root of the sine of an angle squared plus the cosine of the angle squared will always add up to 1:

$$\sqrt{(\sin\theta)^2 + (\cos\theta)^2} = 1$$

We can evaluate the expression $\sqrt{(\sin\theta)^2 + (\cos\theta)^2}$ for the case of $\theta = \frac{\pi}{2}$:

```
» sqrt( (sin(pi/2))^2 + (cos(pi/2))^2 )
ans =
     1
```

As expected, the result is 1.0.

**MATLAB Script Files

In the previous examples , we have executed MATLAB commands by typing them directly into the Command Window and observing the results in the Command Window. While this works, it is a very poor way to perform complex calculations.

For example, suppose that an engineer wanted to make a series of calculations where the results of some calculations depended on the values derived from previous calculations. This could be done by typing each equation in by hand, but there are three disadvantages to this approach:

A MATLAB script file is a much better solution for performing series of calculations and reusing those calculations later. A script file is a file containing a series of MATLAB commands or equations, exactly as they would have been typed into the Command Window.

Script files are also called M-files, because the filename has a file extension of “.m”.

H.W:- Suppose that for a project we wanted to calculate the following values:

1. The area of the circle of radius r

2. The circumference of a circle of radius r

3. The volume of a sphere of radius r

4. The surface area of a sphere of radius r

We will write a single script file that calculates all four values for a given input radius, and we will test the script using a radius of 5 m.

**Variables and Arrays

The basic unit of data in any MATLAB program is the array. An array is a collection of data values organized into rows and columns and known by a single name. Individual data values within an array are accessed by including the name of the array followed by subscripts in parentheses that identify the row and column of the particular value. see Figure 1.

Arrays can be classified as either vectors or matrix. The term “vector” is usually used to describe an array with only one dimension, while the term “matrix” is usually used to describe an array with two or more dimensions. In this text, we will use the term “vector” when discussing one-dimensional arrays, and the term “matrix” when discussing arrays with two or more dimensions.

The size of an array is specified by the number of rows and the number of columns in the array, with the number of rows mentioned first. The total number of elements in the array will be the product of the number of rows and the number of columns. For example, here are some arrays and sizes.

Array	Size
$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$	This is a 3×2 matrix containing 6 elements.
$b = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$	This is a 1×4 array containing 4 elements; it is known as a **row vector**.
$c = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	This is a 3×1 array containing 3 elements; it is known as a **column vector**.

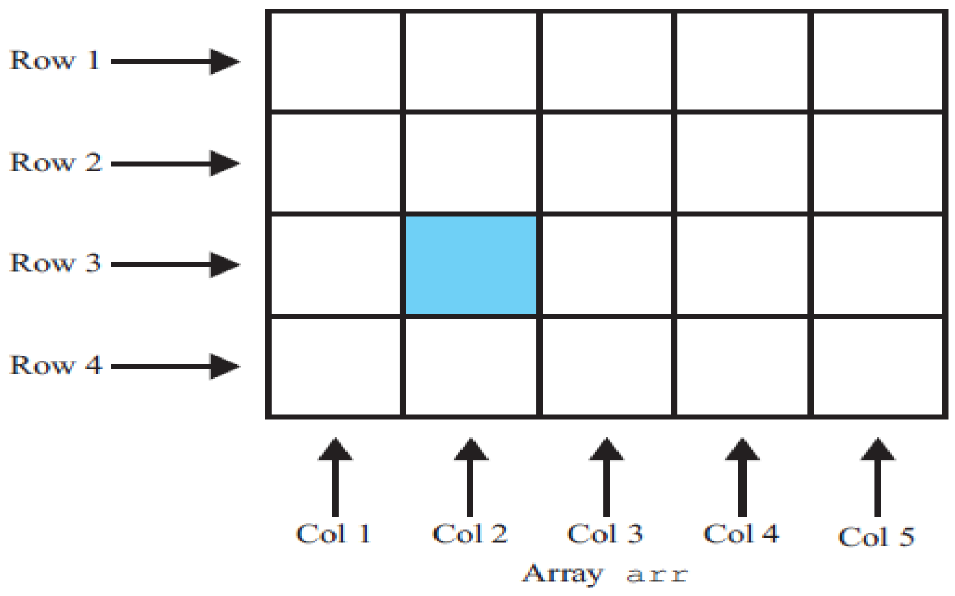


Figure 1: An array is a collection of data values organized into rows and columns.

Individual elements in an array are addressed by the array name followed by the row and column of the particular element. If the array is a row or column vector, then only one subscript is required. For example, in the preceding arrays, a(2, 1) is 3 and c(2) is 2.

A MATLAB variable is a region of memory containing an array; the array is known by a user-specified name. The contents of the array may be used or modified at any time by including its name in an appropriate MATLAB command. MATLAB variable names must begin with a letter, followed by any combination of letters, numbers, and the underscore (_) character. Only the first 63 characters are significant; if more than 63 are used, the remaining characters will be ignored. If two variables are declared with names that only differ in the 64th character, MATLAB will treat them as the same variable. MATLAB will issue a warning if it has to truncate a long variable name to 63 characters.

***Note:** When writing a program, it is important to pick meaningful names for the variables. Meaningful names make a program much easier to read and to maintain. Names such as day, month, and year are clear even to a person seeing a program for the first time. Since spaces cannot be used in MATLAB variable names, underscore characters can be substituted to create meaningful names. For example, exchange rate might become (**exchange_rate).**

The most common types of MATLAB variables are double and char. Variables of type double consist of scalars or arrays of 64-bit double-precision floating-point numbers. They can hold real, imaginary, or complex values. The real and imaginary components of each variable can be positive or negative numbers

A variable of type double is automatically created whenever a numerical value is assigned to a variable name. The numerical values assigned to double variables can be real, imaginary, or complex. A real value is just a number. For example, the following statement assigns the real value 10.5 to the double variable var:

var = 10.5

An imaginary number is defined by appending the letter i or j to a number.1 For example, 10i and –4j are both imaginary values. The following statement assigns the imaginary value 4i to the double variable var:

var = 4i

A complex value has both a real and an imaginary component. It is created by adding a real and an imaginary number together. For example, the following statement assigns the complex value 10 1 10i to variable var:

var = 10 + 10i

Variables of type char consist of scalars or arrays of 16-bit values, each representing a single character. Arrays of this type are called character arrays. They are automatically created whenever a single character or a string of characters is assigned to a variable name. For example, the following statement creates a variable of type char whose name is comment, and stores the specified string in it. After the statement is executed, comment will be a 1 X 26 character array.

comment = 'This is a character string'

**Initializing Variables in Assignment Statements

Simple examples of initializing variables with assignment statements include

var = 40i;

var2 = var / 5;

x = 1; y = 2;

array = [1 2 3 4];

The first example creates a scalar variable of type double and stores the imaginary number 40i in it. The second example creates a scalar variable and stores the result of the expression var/5 in it. The third example creates a variable and stores a 4-element row vector in it. The third example shows that multiple assignment statements can be placed on a single line, provided that they are separated by semicolons.

Note that if any of the variables had already existed when the statements were executed, then their old contents would have been lost.

The last example shows that variables can also be initialized with arrays of data. Such arrays are constructed using brackets ([ ]) and semicolons. All of the elements of an array are listed in row order. In other words, the values in each row are listed from left to right, with the topmost row first and the bottommost row last. Individual values within a row are separated by blank spaces or commas, and the rows themselves are separated by semicolons or new lines. The following expressions are all legal arrays that can be used to initialize a variable:

`[3.4]`	This expression creates a 1 × 1 array (a scalar) containing the value 3.4. The brackets are not required in this case.
`[1.0 2.0 3.0]`	This expression creates a 1 × 3 array containing the row vector $[1 \quad 2 \quad 3]$.
`[1.0; 2.0; 3.0]`	This expression creates a 3 × 1 array containing the column vector $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$.
`[1, 2, 3; 4, 5, 6]`	This expression creates a 2 × 3 array containing the matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$.

`[1, 2, 3`
 ` 4, 5, 6]`
	This expression creates a 2 × 3 array containing the matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$. The end of the first line terminates the first row.

`[]`	This expression creates an **empty array**, which contains no rows and no columns. (Note that this is not the same as an array containing zeros.)

****Note:** The number of elements in every row of an array must be the same, and the number of elements in every column must be the same. An expression such as

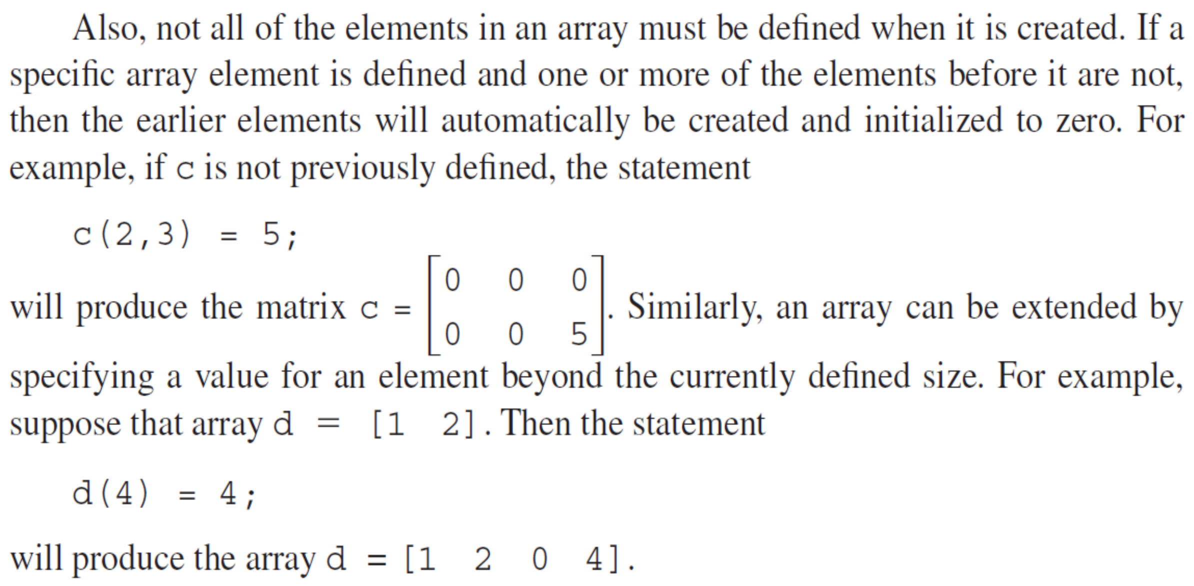
[1 2 3; 4 5];

is illegal because row 1 has three elements while row 2 has only two elements.

**Note :The expressions used to initialize arrays can include algebraic operations and all or portions of previously defined arrays. For example, the assignment statements

```
a = [0 1+7];
b = [a(2) 7 a];
```

will define an array `a = [0   8 ]` and an array `b = [8   7   0   8 ]`.



Also, not all of the elements in an array must be defined when it is created. If a specific array element is defined and one or more of the elements before it are not, then the earlier elements will automatically be created and initialized to zero. For example, if c is not previously defined, the statement

```
c(2,3) = 5;
```

will produce the matrix c = $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 5 \end{bmatrix}$. Similarly, an array can be extended by specifying a value for an element beyond the currently defined size. For example, suppose that array `d = [1 2]`. Then the statement

```
d(4) = 4;
```

will produce the array `d = [1 2 0 4]`.

**Initializing with Shortcut Expressions

It is easy to create small arrays by explicitly listing each term in the array, but what happens when the array contains hundreds or even thousands of elements? It is not practical to write out each element in the array separately.

MATLAB provides a special shortcut notation for these circumstances using the colon operator. The colon operator specifies a whole series of values by specifying the first value in the series, the stepping increment, and the last value in the series. The general form of a colon operator is

first:incr:last

where first is the first value in the series, incr is the stepping increment, and last is the last value in the series.

For example, the expression 1:2:10 is a shortcut for a 1 3 5 row vector containing the values 1, 3, 5, 7, and 9. The next value in the series would be 11, which is greater than 10, so the series terminates at 9.

```
» x = 1:2:10

x =

1 3 5 7 9
```

Shortcut expressions can be combined with the **transpose operator** (`'`) to initialize column vectors and more complex matrices. The transpose operator swaps the row and columns of any array that it is applied to. Thus the expression

```
f = [1:4]';
```

generates a 4-element row vector [1 2 3 4] and then transposes it into the 4-element column vector `f` = $\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$. Similarly, the expressions

```
g = 1:4;
h = [g' g'];
```

will produce the matrix `h` = $\begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \\ 4 & 4 \end{bmatrix}$.

**Initializing with Built-In Functions

Arrays can also be initialized using built-in MATLAB functions. For example, the function zeros can be used to create an all-zero array of any desired size. There are several forms of the zeros function. If the function has a single scalar argument, it will produce a square array using the single argument as both the number of rows and the number of columns. If the function has two scalar arguments, the first argument will be the number of rows, and the second argument will be the number of columns. Since the size function returns two values containing the number of rows and columns in an array, it can be combined with the zeros function to generate an array of zeros that is the same size as another array. Some examples using the zeros function follow:

```
a = zeros(2);
b = zeros(2,3);
c = [1 2; 3 4];
d = zeros(size(c));
```

These statements generate the following arrays:

$$a = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad b = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$c = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Similarly, the ones function can be used to generate arrays containing all ones, and the eye function can be used to generate arrays containing identity matrices, in which all on-diagonal elements are one, while all off-diagonal elements are zero. The next Table. contains a list of common MATLAB functions useful for initializing variables.

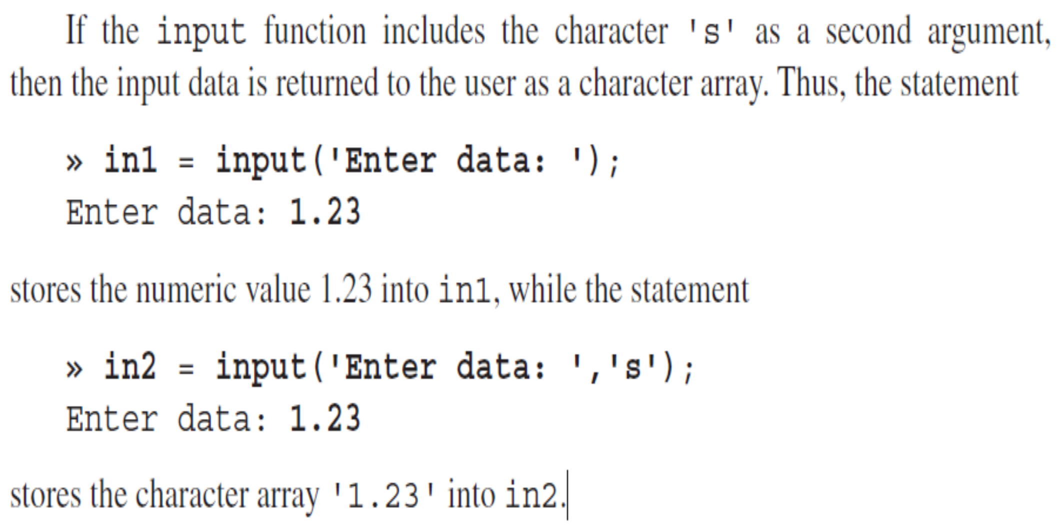
Function	Purpose
`zeros(n)`	Generates an `n` × `n` matrix of zeros.
`zeros(m,n)`	Generates an `m` × `n` matrix of zeros.
`zeros(size(arr))`	Generates a matrix of zeros of the same size as `arr`.
`ones(n)`	Generates an `n` × `n` matrix of ones.
`ones(m,n)`	Generates an `m` × `n` matrix of ones.
`ones(size(arr))`	Generates a matrix of ones of the same size as `arr`.
`eye(n)`	Generates an `n` × `n` identity matrix.
`eye(m,n)`	Generates an `m` × `n` identity matrix.
`length(arr)`	Returns the length of a vector, or the longest dimension of a two-dimensional array.
`numel(arr)`	Returns the total number of elements in an array, which is the product of the number of rows times the number of columns.
`size(arr)`	Returns two values specifying the number of rows and columns in `arr`.

**Initializing Variables with Keyboard Input

It is also possible to prompt a user and initialize a variable with data that he or she types directly at the keyboard. This option allows a script file to prompt a user for input data values while it is executing. The input function displays a prompt string in the Command Window and then waits for the user to type in a response. For example, consider the following statement:

my_val = input('Enter an input value:');

When this statement is executed, MATLAB prints out the string 'Enter an input value:', and then waits for the user to respond



If the `input` function includes the character `'s'` as a second argument, then the input data is returned to the user as a character array. Thus, the statement

```
» in1 = input('Enter data: ');
Enter data: 1.23
```

stores the numeric value 1.23 into `in1`, while the statement

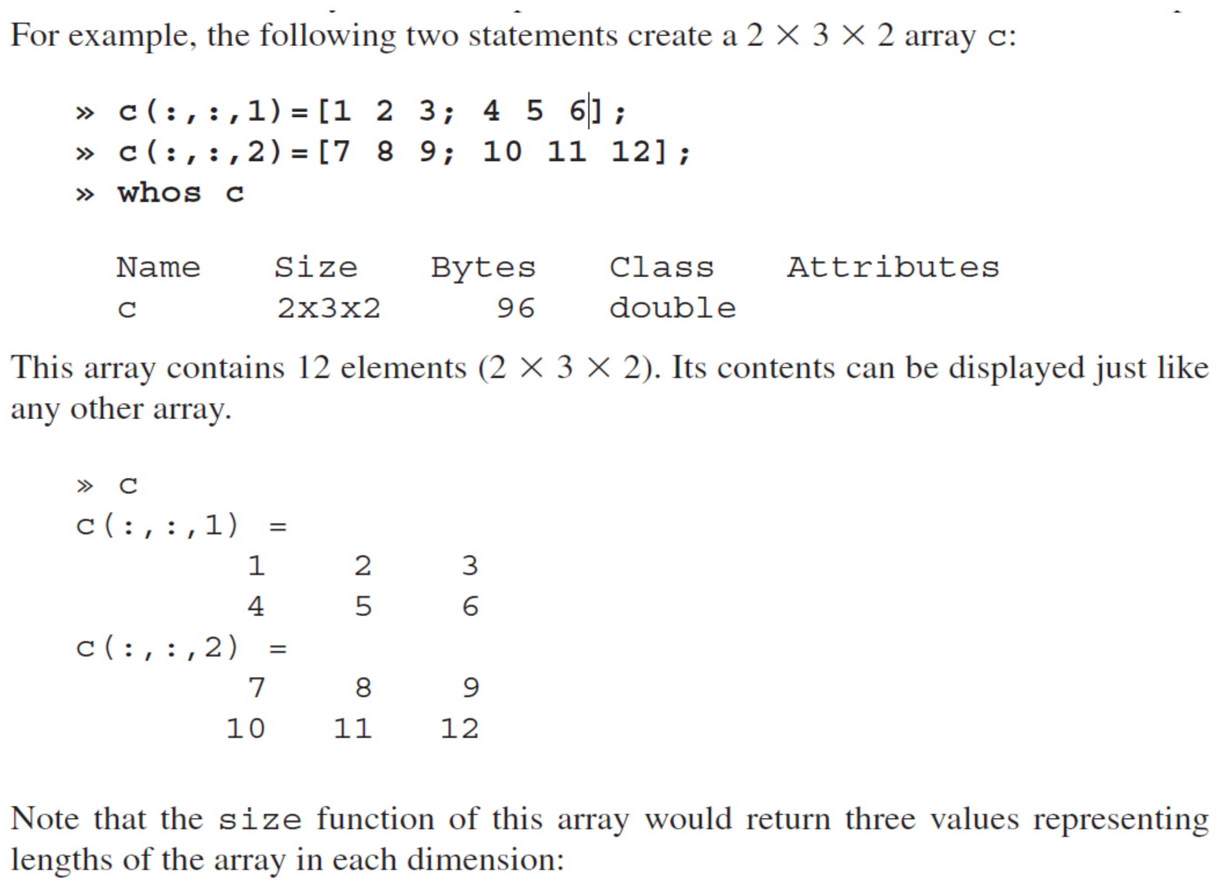
```
» in2 = input('Enter data: ','s');
Enter data: 1.23
```

stores the character array `'1.23'` into `in2`.

**Multidimensional Arrays

As we have seen, MATLAB arrays can have one or more dimensions. One-dimensional arrays can be visualized as a series of values laid out in a row or column, with a single subscript used to select the individual array elements. Such arrays are useful to describe data that is a function of one independent variable, such as a series of temperature measurements made at fixed intervals of time.

Some types of data are functions of more than one independent variable. For example, we might wish to measure the temperature at five different locations at four different times. In this case, our 20 measurements could logically be grouped into five different columns of four measurements each, with a separate column for each location .



For example, the following two statements create a 2 × 3 × 2 array c:

```
» c(:,:,1)=[1 2 3; 4 5 6];
» c(:,:,2)=[7 8 9; 10 11 12];
» whos c

  Name      Size       Bytes    Class     Attributes
  c         2x3x2         96    double
```

This array contains 12 elements (2 × 3 × 2). Its contents can be displayed just like any other array.

```
» c
c(:,:,1) =
        1     2     3
        4     5     6
c(:,:,2) =
        7     8     9
       10    11    12
```

Note that the size function of this array would return three values representing lengths of the array in each dimension:

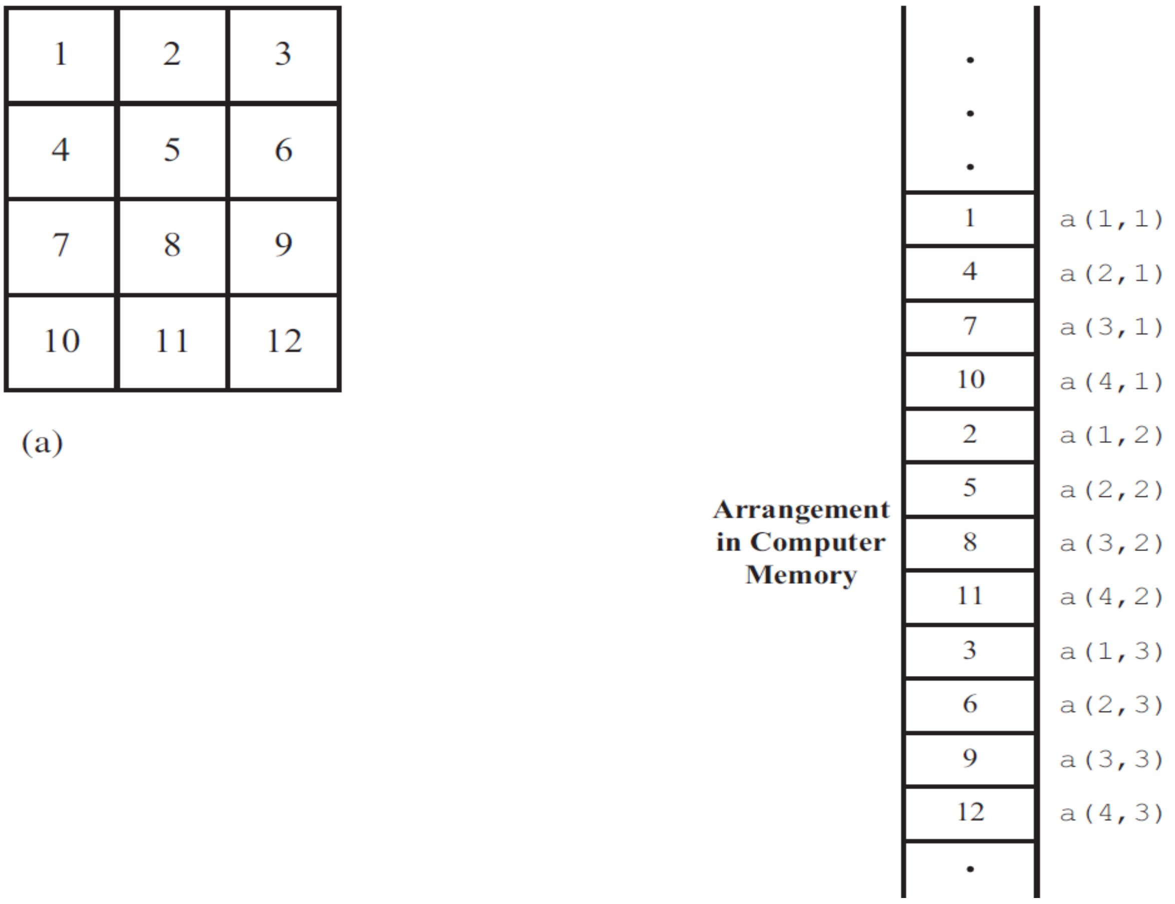
```
» size(c)

ans =

2 3 2
```

and the numel function would return the total number of elements in the array:

```
» numel(c)

ans =

12
```

**Storing Multidimensional Arrays in Memory

MATLAB always allocates array elements in column major order. That is, MATLAB allocates the first column in memory, then the second, then the third, and so forth until all of the columns have been allocated. Figure



**Accessing Multidimensional Arrays with One Dimension

One of MATLAB’s property is that it will permit a user or programmer to treat a multidimensional array as though it were a one-dimensional array whose length is equal to the number of elements in the multidimensional array. If a multidimensional array is addressed with a single dimension, then the elements will be accessed in the order in which they were allocated in memory.

```
» a = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
a =
1 2 3
4 5 6
7 8 9
10 11 12
```

Then the value of a(5) will be 2.

**Subarrays

It is possible to select and use subsets of MATLAB arrays as though they were separate arrays. To select a portion of an array, just include a list of all of the elements to be selected in the parentheses after the array name. For example, suppose array arr1 is defined as follows:

```
arr1 = [1.1  -2.2  3.3   -4.4   5.5];
```

Then arr1(3) is just 3, arr1([1 4]) is the array [1.1 -4.4], and arr1(1:2:5) is the array [1.1 3.3 5.5].

For a two-dimensional array, a colon can be used in a subscript to select all of the values of that subscript. For example, suppose

```
arr2 = [1 2 3; -2 -3 -4; 3 4 5];
```

This statement would create an array arr2 containing the values $\begin{bmatrix} 1 & 2 & 3 \\ -2 & -3 & -4 \\ 3 & 4 & 5 \end{bmatrix}$

With this definition, the subarray arr2(1,:) would be [1 2 3], and the subarray

arr2(:,1:2:3) would be $\begin{bmatrix} 1 & 3 \\ -2 & -4 \\ 3 & 5 \end{bmatrix}$.

**The end Function

MATLAB includes a special function named end that is very useful for creating array subscripts. When used in an array subscript, end *returns the highest value taken on by that subscript*. For example, suppose that array arr3 is defined as follows:

```
arr3 = [1 2 3 4 5 6 7 8];
```

Then arr3(5:end) would be the array [5 6 7 8], and array(end) would be the value 8.

The value returned by end is always the *highest value* of a given subscript. If end appears in different subscripts, it can return *different* values within the same expression. For example, suppose that the 3 × 4 array arr4 is defined as follows:

```
arr4 = [1 2 3 4; 5 6 7 8; 9 10 11 12];
```

Then the expression arr4(2:end,2:end) would return the array $\begin{bmatrix} 6 & 7 & 8 \\ 10 & 11 & 12 \end{bmatrix}$.

Note It is also possible to use subarrays to update only some of the values in an array, as long as the shape (the number of rows and columns) of the values being assigned matches the shape of the subarray.

```
» arr4 = [1 2 3 4; 5 6 7 8; 9 10 11 12]
arr4 =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

Then the following assignment statement is legal, since the expressions on both sides of the equal sign have the same shape (2 × 2):

```
» arr4(1:2,[1 4]) = [20 21; 22 23]
arr4 =
    20     2     3    21
    22     6     7    23
     9    10    11    12
```

```
arr4 = [1 2 3 4; 5 6 7 8; 9 10 11 12];
```

Then the following expression assigns the value 1 to four elements of the array.

```
» arr4(1:2,1:2) = 1
arr4 =
     1     1     3     4
     1     1     7     8
     9    10    11    12
```