Learning Objectives

After completing this chapter, you will be able to:
✓ Use basic AutoLISP functions on the command line.
✓ Locate, load, and run existing AutoLISP programs.
✓ Define new AutoCAD commands.
✓ Write AutoLISP programs using the Visual LISP Editor.

What Is AutoLISP?

AutoLISP is a derivative, or dialect, of the LISP programming language. LISP, which is an acronym for list processing, is a high-level computer programming language used in artificial intelligence (AI) systems. In this reference, the term high-level does not mean complex, rather it means powerful. As a matter of fact, many AutoCAD users refer to AutoLISP as the “nonprogrammer’s language” because it is easy to understand.

AutoLISP is specially designed by Autodesk to work with AutoCAD. It is a flexible language that allows the programmer to create custom commands and functions that can greatly increase productivity and drawing efficiency.

Knowing the basics of AutoLISP gives you a better understanding of how AutoCAD works. By learning just a few simple functions, you can create new commands that make a significant difference in your daily productivity. Read through this chapter slowly while you are at a computer. Type all of the examples and exercises as you read them. This is the best way to get a feel for AutoLISP.

AutoLISP and AutoCAD

AutoLISP can be used in several ways. It is a built-in feature of AutoCAD and, therefore, available on the command line. When AutoLISP commands and functions are issued inside of parentheses on the command line, the AutoLISP interpreter automatically evaluates the entry and carries out the specified tasks. AutoLISP functions can also be incorporated into AutoCAD as buttons, menu items, and tablet menu picks. In addition, AutoLISP command and function definitions can be saved in a file and
then loaded into AutoCAD when needed. Items that are frequently used can be placed in the acad2011.lsp file, which is automatically loaded for the first drawing (by default) when AutoCAD starts. The acad2011doc.lsp file should contain functions that are to be available in all concurrent drawings during a session.

AutoCAD also provides an integrated development environment for editing called Visual LISP. Visual LISP and the Visual LISP Editor offer powerful features designed specifically for writing and editing AutoLISP programs.

The benefits of using AutoLISP are endless. A person with a basic understanding of AutoLISP can create new commands and functions to automate many routine tasks. After working through this chapter, you will be able to add greater capabilities to your menu macros. You can also enter simple AutoLISP expressions on the command line. More experienced programmers can create powerful programs that quickly complete very complex design requirements. Examples of possible new functions that might be designed using AutoLISP include:

- Automatic line breaks when inserting schematic symbols.
- Automatic creation of shapes with associated text objects.
- Parametric design applications that create geometry based on numeric entry.

**AutoLISP Basics**

As stated earlier, LISP stands for list processing, which indicates that AutoLISP processes lists. In the LISP language, a list can be defined as any number of data enclosed in parentheses. Each item in a list must be separated from other items by a space.

When any entry is made on the command line, it is checked to see if the first character is a parenthesis. The opening parenthesis tells AutoCAD that an AutoLISP expression is being entered. AutoCAD then sends the expression to the AutoLISP interpreter for evaluation. The initial input can be supplied as direct keyboard entry or even a menu macro. The format for an AutoLISP expression, called syntax, is:

```
(FunctionName AnyRequiredData...)
```

The first item in the AutoLISP expression is a function name. A function in AutoLISP is similar to a command in AutoCAD. Some functions require additional information. For example, the addition function requires numeric data:

```
Command: (+ 2 4)
6
Command:
```

Any required or optional data for a function are called the arguments. Some functions use no arguments; others may require one or more. When entering an AutoLISP expression, it is important to close it using a closing parenthesis prior to pressing `[Enter]`. When you press `[Enter]`, the AutoLISP interpreter checks to see that the number of opening and closing parentheses match. If they do not, you are prompted:

```
(>)
```

The `>` prompt indicates that you are missing one closing parenthesis. In this example, all that is necessary is to enter the single missing parenthesis and the function is complete.

```
(> )
6
Command:
```
When the AutoLISP interpreter evaluates an AutoLISP expression, it returns a value. An expression entered on the command line instructs the system to return its value to the command line, such as 6 in the previous example. If a different prompt is active, the returned value is used as input for that prompt. For example, this next sequence uses the result of adding two numbers as the input at the Specify radius of circle or [Diameter]: prompt. Checking the CIRCLERAD system variable verifies that the value returned by AutoLISP was in fact applied to the circle radius.

Command: C or CIRCLE.
Specify center point for circle or [3P/2P/Ttr (tan tan radius)]: (pick a point)
Specify radius of circle or [Diameter]: (+ 14.25 3.0).
17.25
Command: CIRCLERAD.
Enter new value for CIRCLERAD <17.2500>: .

**Basic AutoLISP Functions**

The best way to get started learning AutoLISP is to enter a few functions on the command line and see what they do. The following discussion includes basic AutoLISP functions that are part of the foundation for all AutoLISP programs. Practice using the functions as you read. Then, begin using them in menus and macros. At first, these functions and expressions will be entered on the command line. Later in this chapter, and in Chapter 28, you will learn about creating and using AutoLISP program files.

**AutoLISP Math Functions**

AutoLISP provides many different mathematical operators for performing calculations. All real number calculations in AutoLISP are accurate to 15 decimal places. AutoLISP distinguishes between real numbers and integers, handling each data type differently. **Real numbers** are numbers with a decimal point, such as 1.25, 7.0, and −0.438. **Integers** are whole numbers without a decimal point, such as 3, 91, and −115. If a mathematical expression has only integer arguments, the result is returned as an integer. If at least one real number is used, the result is returned as a real number. The following symbols are used for the four basic math functions.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition; returns the sum of all supplied number arguments.</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction; subtracts the sum of the second through the last number arguments from the first number argument and returns the result.</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication; returns the product of all supplied number arguments.</td>
</tr>
<tr>
<td>/</td>
<td>Division; divides the first number argument by the product of the second through the last number arguments.</td>
</tr>
</tbody>
</table>
**NOTE**

*Real numbers* technically include integers and fractions, as well as decimal numbers. However, in applications involving AutoLISP and throughout this discussion, real numbers are always classified as those that have a decimal part.

The following examples illustrate AutoLISP math expressions entered on the command line. As you practice entering these expressions, use the following procedure.

1. Start with an open parenthesis.
2. Separate each item in the expression with a space.
3. End the expression with a closing parenthesis.

Using these steps, enter the following expressions on the command line. If you get lost at any time or do not return to the Command: prompt when expected, press the [Esc] key to cancel the AutoLISP entry.

```
Command: (+ 6 2).\n8

Command: (+ 6.0 2).\n8.0

Command: (– 15 9).\n6

Command: (* 4 6).\n24

Command: (/ 12 3).\n4

Command: (/ 12 3.2).\n3.75

Command: (/ 19 10).\n1

An “incorrect” answer is returned in the last example. The result of dividing 19 by 10 is 1.9. When only integers are supplied as arguments, the result is returned as an integer. If the result were rounded, it would round to 2. However, the returned result is simply the integer portion of the actual answer. The result is not rounded, it is truncated. To get the correct result in division expressions such as the one above, specify at least one of the arguments as a real number.

```
Command: (/ 19.0 10).\n1.9

When entering real numbers between 1 and –1, you must include the leading zero. If the zero is not entered, you will get an error message:

```
Command: (+ .5 16).\n; error: misplaced dot on input
Command:

The correct entry is:

```
Command: (+ 0.5 16).\n16.5
```

**Exercise 27-1**

Complete the exercise on the student website.

www.g-wlearning.com/CAD
Nested Expressions

The term *nested* refers to an expression that is used as part of another expression. For example, to add 15 to the product of 3.75 and 2.125, you can nest the multiplication expression within the AutoLISP addition expression. Notice the two closing parentheses:

Command: `(+ 15 (* 3.75 2.125))
22.9688

Nested expressions are evaluated from the deepest nested level outward. In the previous expression, the multiplication operation is evaluated first and the result is applied to the addition operation. Here are some examples of nested expressions:

Command: `(+ 24 (* 5 4))
44
Command: `(* 12 (/ 60 20))
36
Command: `(/ 39 (* 1.6 11))
2.21591

Significant Digits

AutoLISP performs all mathematical calculations to 15 decimal places, but only displays six significant digits. For example, take a close look at this expression:

Command: `(+ 15 (* 3.75 2.125))
22.9688

The actual result is 22.96875, but AutoLISP displays only six significant digits on the command line and rounds the number for display only. This is true for large and small numbers alike. The next example shows how AutoLISP uses exponential notation to display larger numbers using only six digits:

Command: `(* 1000 1575.25)
1.57525e+006

This final example uses a numeric printing function (real to string) set to show eight decimal places in order to indicate the number is not actually rounded and that no precision is lost:

Command: `(rtos (+ 15 (* 3.75 2.125)) 2 8)
"22.96875000"

Exercise 27-2

Complete the exercise on the student website.
www.g-wlearning.com/CAD

Variables

All programming languages make use of *variables* to temporarily store information. The variable name can be used in an expression anywhere in the program. When AutoLISP encounters a variable in an expression, it uses the value of the variable to evaluate the expression. An AutoLISP variable name cannot be made up of numeric characters only, nor can it contain any of the following characters.

- Open parenthesis (`(`)
- Close parenthesis (`)`)
• Period (.)
• Apostrophe (‘)
• Quotation marks (“”)
• Semicolon (;)

The **setq** AutoLISP function is used to set variable values. A **setq** expression requires a variable name and value as arguments. The following example shows an expression that creates a variable named **A** and assigns it a value of 5.

Command: `(setq A 5)`

5

If you try to use an illegal variable name, an error message is returned. The following example tries to create a variable named **2** with an assigned value of 7. Since **2** is not a valid variable name, an error message is returned.

Command: `(setq 2 7)`

; error: syntax error

Once a valid variable name has been assigned a value, the variable can be used in subsequent AutoLISP expressions or even directly accessed on the command line. To access a variable value on the command line, precede the variable name with an exclamation mark (!). For example:

Command: C or CIRCLE

Specify center point for circle or [3P/2P/Ttr (tan tan radius)]: (pick a point)

Specify radius of circle or [Diameter] <current>: !A

5

Command:

To use the value of a variable in any expression, simply include the variable in the appropriate location. The following sequence sets and uses a series of variables.

Command: `(setq A 5)`

5

Command: `(setq B (– A 1))`

4

Command: `(setq C (– A B))`

1

Command: `(setq D (* (+ A B) 2))`

18

Look closely at the example illustrated in Figure 27-1. Find the three separate expressions inside of parentheses. AutoLISP evaluates expression three first. The result is applied to expression two, which is then evaluated. The result of expression two is applied to expression one. The final evaluation determines the value of the variable **D**.

![Figure 27-1](image-url)
PROFESSIONAL TIP

When working with AutoLISP on the command line, use AutoCAD’s command line editing features to your best advantage. Remember that you can use the up and down arrow keys to display previously entered lines of code. Additionally, you can use the left and right arrow keys to position the cursor to delete or insert text within a line. You may also find it convenient to turn off dynamic input while entering AutoLISP expressions on the command line.

Exercise 27-3

Complete the exercise on the student website.
www.g-wlearning.com/CAD

AutoLISP Program Files

Entering AutoLISP expressions on the command line is suitable for applications that are simple or unique. However, when more complex expressions are required or when the expressions you are using may be needed again, it is best to save them in an AutoLISP program file. This can be easily accomplished using the Visual LISP Editor provided with AutoCAD. AutoLISP programs can be more effectively developed using the Visual LISP Editor. Creating AutoLISP program files is discussed in the following sections.

A very common feature found in most AutoLISP programs is a function definition. A function definition is a collection of AutoLISP expressions that performs any number of tasks. The function is assigned a name that is used to activate the expressions. Some function definitions create new AutoCAD command names that can be entered at the command line.

Once written and saved, an AutoLISP program can be loaded and used whenever it is needed. By default, AutoCAD automatically loads the acad2011.lsp file, if it is located in the support file search path, when you first begin a drawing session. The acad2011doc.lsp file is loaded with each drawing that is opened. Any new AutoLISP commands or functions that you define in this file will be available in every drawing during a session.

An AutoLISP program file must be a “plain” text file. If you choose to edit your AutoLISP files with a word processing program such as Microsoft Word or WordPerfect, be sure to save the files as “text only.” Word processing files use special codes that AutoLISP cannot understand. However, it is recommended that you use the Visual LISP Editor because it has tools specifically designed for use in writing and editing AutoLISP programs.

Introduction to the Visual LISP Editor

The Visual LISP Editor provides powerful editing features. The editor is an integrated development environment (IDE) that features AutoLISP development tools not available in standard text editing programs. The interactive nature of the Visual LISP Editor simplifies the task of creating AutoLISP program files. This section provides only a brief introduction to Visual LISP.
The **VLIDE** command is used to open the **Visual LISP Editor**. When the **Visual LISP Editor** is first displayed, it appears as shown in Figure 27-2. The windows within the editor can be minimized or maximized. The editor itself can be temporarily closed to return to AutoCAD as necessary.

To create a new AutoLISP program using the **Visual LISP Editor**, pick the **New file** button, select **New File** from the editor’s **File** pull-down menu, or press **[Ctrl]+[N]**. This opens a window for an untitled document on the **Visual LISP Editor** desktop. See Figure 27-3. The windows and features in the **Visual LISP Editor** are:

- **Desktop**. This is the main area of the editor window. It is similar to the main program window in AutoCAD and can be used to relocate toolbars or windowed components, such as the **Visual LISP Console** or a text editor window.

- **Text editor window**. Text editor windows are used to write and edit AutoLISP programs. Different windows can be used to create new files or view existing programs. The **Visual LISP Editor** provides interactive feedback as you enter material to help you avoid errors.

- **Visual LISP Console window**. This window provides several functions. You can use it to enter any AutoLISP expression to immediately see the results. You can also enter any AutoLISP variable to determine its value. Visual LISP commands can also be entered from this window. The text can then be copied from the window to a text editor window.

- **Trace window**. This window is minimized when you first display the **Visual LISP Editor**. It records a history of the functions within your program and can be used to trace values when developing or debugging a program.

- **Status bar**. This area at the bottom of the **Visual LISP Editor** is similar to the status bar in AutoCAD's main program window. It provides feedback regarding the status of the current window or application being used.
Several visual aids are provided to help identify functions as you enter text. For example, as you construct the expressions that make up your program, a color-coding system provides immediate feedback as you type. Text for any unrecognized items, such as a user variable or a portion of a function, is shown in black. For example, as you enter the `(setq)` function, the text is shown in black until you have entered the letters `set`. Because AutoLISP recognizes the text entry as the valid function `(set)`, a function not covered in this book, the color of the text is changed to blue. When you have entered the full `(setq)` function name, the text remains blue because AutoLISP also recognizes this function name. The color coding can be very useful. If you enter a function name and the text does not turn blue, you know that you have made an incorrect entry. The default color coding system used in the Visual LISP Editor appears in the following chart.

<table>
<thead>
<tr>
<th>AutoLISP Text Elements</th>
<th>Associated Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-in functions and protected symbols</td>
<td>Blue</td>
</tr>
<tr>
<td>Text strings</td>
<td>Magenta</td>
</tr>
<tr>
<td>Integers</td>
<td>Green</td>
</tr>
<tr>
<td>Real numbers</td>
<td>Teal</td>
</tr>
<tr>
<td>Comments</td>
<td>Purple on a gray background</td>
</tr>
<tr>
<td>Parentheses</td>
<td>Red</td>
</tr>
<tr>
<td>Unrecognized items</td>
<td>Black</td>
</tr>
</tbody>
</table>

Another valuable visual aid provided by the Visual LISP Editor is instant parenthesis matching. When you enter a closing parenthesis in an expression, the cursor jumps to the opening parenthesis and then returns to the current position. If the closing parenthesis does not have a match, the cursor does not jump. This helps indicate that a matching parenthesis is needed.
Once you have entered one or more expressions in the Visual LISP Editor, you can save the file and then test the results in the Visual LISP Console window. Or, you can return to AutoCAD to test your results.

**Defining New AutoCAD Commands**

In this section, you will use several of the built-in AutoLISP functions to create a new AutoCAD command. The \texttt{(defun)} AutoLISP function (define function) is used for this. The syntax for this function is:

\begin{verbatim}
(defun FunctionName (ArgumentList) (Expression) ...) 
\end{verbatim}

The function name can be any alphanumeric name and is subject to the same conditions as for any variable name assigned with the \texttt{(setq)} function. If you prefix the function name with \texttt{C:}, the name can be entered on the command line in AutoCAD.

You must include an argument list in every function definition, even if the list is empty. The argument list is used to declare local variables and, in more advanced applications, to indicate which arguments are required by a function. For many applications, the argument list is simply left empty.

Any number of expressions can be included in a function definition. All of the expressions contained in the definition are evaluated when the function name is called.

A very powerful, yet simple, application for a function definition is to create a shortcut command similar to one of the command aliases in the acad.pgp file. However, a shortcut command defined using AutoLISP can specify command options and even multiple commands to use. Remember that the command aliases defined in the acad.pgp file can only launch a single command; they cannot specify any command options.

This first example shows the definition for a new function named \texttt{ZX}. The function issues the \texttt{ZOOM} command and performs the \texttt{Previous} option.

\begin{verbatim}
(defun C:ZX () (command "ZOOM" "PREVIOUS") )
\end{verbatim}

To see this function work, enter the definition at the command line as:

\begin{verbatim}
Command: (defun C:ZX () (command "zoom" "previous")) \j 
C:ZX
\end{verbatim}

Notice that the new function name is returned by the \texttt{(defun)} function. The \texttt{C:} prefix indicates that it can be entered at the Command: prompt.

\begin{verbatim}
Command: ZX. \j 
zoom Specify corner of window, enter a scale factor (nX or nXP), or [All/Extents/Window/Previous/Object] <real time>: previous Command: nil Command: 
\end{verbatim}
When activated, defined functions return the value of the last expression evaluated in the definition. Since the `(command)` function always returns a value of “nil,” this is returned when using the `ZX` function. The “nil” value has no effect. You can suppress it if you do not want it to appear each time you use a defined function. To suppress the value, add the `(princ)` function using no arguments to the end of the definition:

```
(defun C:ZX ()
   (command "ZOOM" "PREVIOUS")
   (princ)
)
```

Entering a function definition on the command line is an inconvenient way to define custom functions. By storing such definitions in a file, they can be loaded whenever needed.

**PROFESSIONAL TIP**

When defining new command names, keep in mind that most AutoCAD users are one-handed typists because the other hand is used for the pointing device. For example, when deciding on the name for a function that performs a `ZOOM Previous`, it may be easier for the user to type the `[Z]+[X]` key combination rather than `[Z]+[P]`. The `[Z]` and `[P]` keys are in opposite corners of the keyboard.

Creating Your First AutoLISP Program

As discussed earlier, a typical use for an AutoLISP program file is to hold function definitions. An AutoLISP program file can contain a single function or it can contain several. Many AutoLISP programs are created to perform a single group of specific tasks. Other AutoLISP programs hold a large number of function definitions, all of which become available when the program file is loaded. One common application for the `acad2011doc.lsp` file is to create a series of function definitions for shortcut commands used to speed up routine drafting tasks.

Writing the Program

To create your first AutoLISP program, open the Visual LISP Editor or another text editing application and start a new document (file). Begin by entering two function definitions into the program. The first is the `ZX` function from the previous example. The second defines a command named `FC` (fillet corner) that sets the fillet radius to 0 and allows the `FILLET` command to continue.

```
(defun C:ZX()
   (command "ZOOM" "PREVIOUS")
   (princ)
)
```

```
(defun C:FC()
   (command "FILLET" "RADIUS" 0 "FILLET" "MULTIPLE")
   (princ)
)
```

When using the Visual LISP Editor, the final closing parenthesis is not automatically “flush left.” You will need to delete the spaces added. It is a good habit to place the final closing parenthesis flush left to help keep your program organized.
Adding the appropriate documentation to your program files is recommended. When a semicolon (:) is encountered in a program (except when it is part of a text string), any information on the line to the right of the semicolon is ignored. This allows you to place comments and documentation in your AutoLISP programs. The example below shows the appropriate documentation for this program file, called myfirst.lsp.

```lisp
; MyFirst.lsp
; by A. Novice

;C:ZX – To key ZOOM Previous command.
(defun C:ZX ()
  (command "ZOOM" "PREVIOUS")
  (princ)
)

;C:FC – Fillet Corner: Sets fillet radius to 0 and
;allows FILLET command to continue.
(defun C:FC ()
  (command "FILLET" "RADIUS" 0 "FILLET")
  (princ)
)

After entering these functions and comments into the new LISP program, save the file as myfirst.lsp in the AutoCAD or your user’s \Support folder.

**Loading the Program**

The APPLOAD command is used to load applications, such as AutoLISP program files, into AutoCAD. Once the command is selected in AutoCAD, the Load/Unload Applications dialog box appears, Figure 27-4. A list of currently loaded applications appears in the Loaded Applications tab. To load an application file, select it in the file-selection window near the top of the dialog box. You can highlight any number of files.
in the list. Picking the **Load** button loads the currently selected application file(s). If the **Add to History** check box is activated, the loaded file(s) will be added to the list in the **History list** tab. This tab provides convenient access to saved files during subsequent `APPLLOAD` sessions and keeps you from having to search for frequently used files every time they are needed. Picking the **Unload** button removes any highlighted files from the **History list** tab or **Loaded Applications** tab.

You can also load an AutoLISP file by highlighting the file in Windows Explorer and dragging and dropping it into the AutoCAD drawing area. This method is extremely convenient if Windows Explorer is open.

The `(load)` function allows you to load an AutoLISP program file on the command line. This function requires an AutoLISP file name as its argument and requires that the name be enclosed in quotation marks. To load the `myfirst.lsp` file using the `(load)` function, the following sequence is used.

```
Command: (load "myfirst")
```

When the file has a `.lsp` file extension, it is not necessary to include the extension in the `(load)` expression. Therefore, you should use the standard `.lsp` file extension for all AutoLISP files you create. If you are loading an AutoLISP file that does not use a `.lsp` file extension, the actual extension must be included in the file name argument.

When an AutoLISP program file is loaded and no errors are encountered, the result of evaluating the last expression in the file is returned to the screen. In the example above, the last expression in the file is the definition for the `FC` function, so that function name is returned.

The `(load)` function locates AutoLISP files residing in the support file search path. To load a file that exists elsewhere, the path name must also be specified. In the following example, the `myfirst.lsp` file is stored in the `C:\My Documents\AutoLISP` folder.

```
Command: (load "c:/my documents/autolisp/myfirst")
```

Notice that backslashes are not used in the path specification. In an AutoLISP text string, the backslash is used to specify special characters. For example, the string `\n` indicates a new line. When specifying file paths, you can use either forward slashes, as shown above, or double backslashes (`\\`). Therefore, in the example above, the file to load could also have been specified as `c:\my documents\autolisp\myfirst`.

If you frequently load files that are in a folder not found in the current support file search path, it may be helpful to include the folder in the path. This is done using the **Options** dialog box. After displaying this dialog box, pick the **Files** tab and select **Support File Search Path**. Then, pick the **Add...** button and enter the desired folder. Refer to Chapter 21 for detailed information.

As previously indicated, when you have defined one or more functions that you want to have available in all editing sessions, the definitions can be placed in the `acad2011doc.lsp` file. If the `acad2011doc.lsp` file already exists on your system, consult your system administrator or instructor prior to editing this file. The `acad2011.lsp` and `acad2011doc.lsp` files are often used by third-party applications. Changing them or accidentally redefining existing commands or functions may render certain features unusable.

---

**Exercise 27-4**

Complete the exercise on the student website.

www.g-wlearning.com/CAD
Creating Specialized Functions

Although shortcut commands are a powerful use for AutoLISP, it can also be used to create highly specialized functions. Input can specify how the program should function in any given situation. This aspect of AutoLISP allows you to customize AutoCAD to meet the specific needs of your industry or department.

AutoLISP is a versatile tool, offering many different functions for effectively working with numeric data, text data, data files, and AutoCAD drawing objects. This section introduces several basic AutoLISP functions, including some that are used for acquiring input from the user.

Providing for User Input

AutoLISP can prompt the user for various types of data. Depending on its function, a program may need numeric input, text input, or specification of a coordinate location. The \texttt{(getpoint)} function prompts for a point entry and pauses the program until the point is entered. For example:

\texttt{Command: (setq PT1 (getpoint))}\n
After you press [Enter], AutoLISP waits for a value to be input for the \texttt{(getpoint)} function and then stores the value in the \texttt{PT1} variable. A prompt can be added to the original expression to clarify what is required:

\texttt{Command: (setq PT1 (getpoint "Enter a point: "))}\n
Now, pick a point on screen. The coordinates for the selected point are assigned to \texttt{PT1} and displayed on the command line. If you know the coordinates, you can enter them at the keyboard.

The following example shows how closely AutoCAD and AutoLISP work together. First, define the two variables \texttt{PT1} and \texttt{PT2} as shown below. Then, enter the \texttt{LINE} command and use AutoLISP notation to return the values of \texttt{PT1} and \texttt{PT2} as the endpoints of the line.

\texttt{Command: (setq PT1 (getpoint "From point: "))}\n\texttt{From point: 2,2}\n\texttt{(2.0 2.0 0.0)}\n\texttt{Command: (setq PT2 (getpoint "To point: "))}\n\texttt{To point: 6.25,2}\n\texttt{(6.25 2.0 0.0)}\n\texttt{Command: LINE}\n\texttt{Specify first point: !PT1}\n\texttt{(2.0 2.0 0.0)}\n\texttt{Specify next point or [Undo]: !PT2}\n\texttt{(6.25 2.0 0.0)}

The following is a sample function definition named \texttt{1LINE} that uses expressions similar to those given in the previous example. This function will draw a line object based on user input.

\texttt{(defun C:1LINE ()}\n\texttt{(setq PNT1 (getpoint "From point: "))}\n\texttt{(setq PNT2 (getpoint "To point: "))}\n\texttt{(command "LINE" PNT1 PNT2 ")")}\n\texttt{(princ)}\n\texttt{)}
The **(command)** function is used to call AutoCAD commands from within AutoLISP. Typically, an AutoCAD command that is “started” within a program is also “ended” within the program. The pair of quotation marks near the end of the fourth line is equivalent to pressing [Enter] after the second point entry, thus ending the **LINE** command.

When developing AutoLISP routines, you may need to assign the length of a line or the distance between two points to a variable. The **(getdist)** function allows you to assign a distance to a variable. A command line prompt is not added to the following example; the “second point” prompt is automatic.

```
Command: (setq LGTH (getdist)).
(pick the first point)
Specify second point: (pick the second point)
distance
```

Use object snaps as needed or enter absolute coordinates. After the second point is specified, the distance value is shown and assigned to the variable `LGTH`. You can confirm the setting by retrieving the value of the variable.

```
Command: !LGTH.
distance
```

The **(distance)** function is similar to the **(getdist)** function. However, the **(distance)** function does not require picking two points. Instead, it measures the distance between two existing points. This function can be used to display a distance or to assign a distance to a variable.

```
Command: (distance PT1 PT2).  
distance between `PT1` and `PT2`
Command: (setq D1 (distance PT1 PT2)).  
distance between `PT1` and `PT2`
```

The first example returns the distance between the previously defined points `PT1` and `PT2`. The second example displays the distance and applies it to the variable `D1`.

**PROFESSIONAL TIP**

The sample AutoLISP expressions in this section are entered at the command line. However, AutoLISP expressions are more effective as part of a saved program file.

**Exercise 27-5**

Complete the exercise on the student website.
www.g-wlearning.com/CAD

**Assigning Text Values to AutoLISP Applications**

Values assigned to AutoLISP variables do not have to be numeric. In some applications, you may need to assign a word or line of text to a variable. To do so, use the **(setq)** function and enclose the word(s) in quotation marks.

```
Command: (setq W "What next?").
"What next?"
```
You can also assign a word or line of text to a variable with the \texttt{(getstring)} function. This function is similar to the \texttt{(getpoint)} function in that the user must enter a value.

Command: \texttt{(setq E (getstring))}\texttt{.1}

Nothing is displayed on the command line because the optional prompt was not specified. AutoLISP is waiting for a “string” of characters. You can enter as many characters (numbers and letters) as needed. Once you press [Enter] or the spacebar, the string is entered and displayed. To allow spaces in the response, place the letter \texttt{T}, without quotation marks, after the \texttt{(getstring)} function:

Command: \texttt{(setq E (getstring T))}\texttt{.1}

\texttt{HI THERE}\texttt{.1}

"HI THERE"

Command:

The \texttt{(prompt)} function can be used to simply display a message. The resulting message has no variable value. AutoLISP indicates this by printing \texttt{nil} after the prompt.

Command: \texttt{(prompt \textbf{"Select an object: ")}.1}

Select an object: \texttt{nil}

You can use prompts in AutoLISP programs to provide information or to prompt the user.

\textbf{CAUTION}

The symbol \texttt{T} is a built-in AutoLISP constant defined as a \textit{protected symbol}. However, it is possible to change its value using the \texttt{(setq)} function. Do not change its value. Be certain not to use the variable name \texttt{T} for any of your own variables or other functions referencing this symbol may not properly function. If it is accidentally changed, you can reset the value of \texttt{T} using the following expression.

Command: \texttt{(setq T 'T)}\texttt{.1}

\textbf{Exercise 27-6}

Complete the exercise on the student website.\texttt{

\texttt{www.g-wlearning.com/CAD}}

\textbf{Basic AutoLISP Review}

Before applying the functions you have learned to an AutoLISP program, take a few minutes to review the following list. These functions are used in the next chapter, which discusses more advanced AutoLISP applications.

- \texttt{(+), (–), (‘), (/).} These are the basic math functions used in AutoLISP. They must be entered as the first part of an expression. For example, \texttt{(+ 6 8)}.

- \texttt{(setq).} The \texttt{(setq)} function allows a value to be assigned to a variable. For example, the expression \texttt{(setq CITY "San Francisco")} sets the value San Francisco to the variable \texttt{CITY}.

- \texttt{!}. An exclamation point entered before a variable on the command line returns the value of the variable. For example, \texttt{!CITY} returns the value San Francisco for the above expression.
• **(getpoint)**. This function allows you to define a point location by entering coordinates at the keyboard or using the pointing device. The resulting value can be applied to a variable. For example, the expression `(setq A (getpoint))` assigns a point to the variable A.

• **(getdist)**. This function returns the distance between two points entered at the keyboard or picked on screen. The value can be applied to a variable and a prompt can be used. For example, the expression `(setq D2 (getdist "Pick two points:"))` allows you to determine a distance and assign it to the variable D2.

• **(distance)**. This function returns a distance between two existing points. For example, the expression `(distance P1 P2)` returns the distance between the defined points P1 and P2. The distance can also be assigned to a variable. For example, `(setq D (distance P1 P2))`.

• **(getstring)**. This function returns a word or string of characters entered by the user. The resulting text can be assigned to a variable. For example, the expression `(getstring)` waits for a string of characters and displays the string when [Enter] or the spacebar is pressed. Spaces are allowed in the text string if T follows the `(getstring)` function. For example, the expression `(setq TXT (getstring T "Enter text:"))` assigns the text entered, which can contain spaces, to the variable TXT.

• **(prompt)**. Messages or prompts can be issued in a program using the **(prompt)** function. For example, the expression `(prompt "Select an entity:" )` prints the Select an entity: prompt.

**PROFESSIONAL TIP**

Design your AutoLISP programs to closely resemble the AutoCAD interface. For example, it is easier for the user to read “back-to-back” prompts when the prompts appear on separate lines. Use the `\n` string to specify a new line for a prompt. For example:

```
(setq PT2 (getpoint "\nTo point:"))
```

**Exercise 27-7**

Complete the exercise on the student website.

www.g-wlearning.com/CAD
Chapter Test

Answer the following questions. Write your answers on a separate sheet of paper or complete the electronic chapter test on the student website.

www.g-wlearning.com/CAD

1. What is the standard extension used for AutoLISP program files?
2. A comment is indicated in an AutoLISP file with a(n) _____.
3. When in the drawing area, what are three ways to load the contents of the AutoLISP file named chgtext.lsp that is saved in the support file search path?
4. Define the terms integer and real number as related to AutoLISP.
5. Write expressions in the proper AutoLISP format for the following arithmetic functions.
   A. 23 + 54
   B. 12.45 + 6.28
   C. 56 – 34
   D. 23.004 – 7.008
   E. 16 × 4.6
   F. 725 × 10.30
   G. 45 ÷ 23
   H. 147 ÷ 29.6
   I. 53 + (12 × 3.8)
   J. 567 ÷ (34 – 14)
6. Explain the purpose of the (setq) function.
7. Write the proper AutoLISP notation to assign the value of (67 – 34.5) to the variable NUM1.
8. What does the (getpoint) function allow you to do?
9. Which AutoLISP functions allow you to find the distance between two points? Describe the difference between the two functions.
10. Explain the purpose of the (getstring) function.
11. Write the proper AutoLISP notation for assigning the string This is a test: to the variable TXT.
12. How do you allow spaces in a string of text when using the (getstring) function?
13. Write the proper notation for using the PLINe command in an AutoLISP expression.
14. Which prefix must you enter before a function name in an expression to indicate it is accessible at the Command: prompt?
15. What is a function definition?
17. Which AutoLISP function is used to create new AutoCAD commands?
18. What is the purpose of the Visual LISP Editor?
19. When entering text in the Visual LISP Editor, which color indicates that you have entered a built-in function or a protected symbol?
20. Explain the purpose of the \n text string in AutoLISP.
Drawing Problems

Write AutoLISP programs for the following problems. Use the Visual LISP Editor. Save the files as P27-(problem number) with the .lsp extension.

1. Write an AutoLISP program to draw a rectangle. Use the (getpoint) function to set the opposite corners of the rectangle. Follow these guidelines:
   A. Set P1 as the first corner.
   B. Set P3 as the opposite corner.
   C. Use the RECTANG command to draw the rectangle using P1 and P3 in place of picking corners.
   D. The users should not be able to change any options in the RECTANG command.

2. Create an AutoLISP program to draw a square. Follow these guidelines:
   A. Set a variable for the length of one side.
   B. Set the variable P1 as the lower-left corner of the square.
   C. Use the RECTANG command to draw the square.

3. Revise the program in problem 2 to draw a square with filleted corners.
   A. The fillet radius should be equal to 1/4 of the side length.
   B. After the fillet rectangle is drawn, reset the fillet radius so the next rectangle drawn with the RECTANG command does not automatically have fillets. Note: Use ^C to cancel a command that you do not want to complete. For example:
      (command "LINE" PT1 ^C))

4. Use the program in problem 3 to create a new command that draws a square with thick lines.
   A. The line thickness should be a percentage of the fillet radius (between 5% and 10%).
   B. Reset the line thickness so that the next rectangle drawn with the RECTANG command does not automatically have thick lines. Note: Use ^C to cancel a command that you do not want to complete. For example:
      (command "LINE" PT1 ^C)

5. Write an AutoLISP program that allows the user to draw parallel rectangles.
   A. Provide a prompt that asks the user to enter an offset distance for a second rectangle to be placed inside of the first rectangle.
   B. Use the OFFSET command to allow the user to draw the parallel rectangle inside the original without entering an offset distance.