ATP164
First Step in AutoLISP Programming

Segment 2

Date: January 15, 2007
Instructor: Mohmed Zuber Shaikh
Level: Beginning
Category: AutoLISP

Web: www.AUGI.com

Introduction

In first segment, we explored programming fundamentals. Every programming language is equipped with pre defined functions to make programr's life easy. In this segment we will be dealing with String handling functions, Data type conversion functions, built-in functions for controlling the user input, list manipulation functions and other useful functions. This segment will also cover the methods and functions for providing logical flow to your program. At the end of this segment, you will be in position to write your own functions.

OTHER PREDEFINED FUNCTIONS

3.1 Handling of string in AutoLISP :

AutoLISP is basically a graphics mode output program tool but text also forms a part of drawings and diagrams, so handling the string is also a important part of AutoLISP programming.

String is nothing but a sequence of alphanumeric characters. AutoLISP provides many string manipulation functions. We have already covered the function getstring, used for accepting string input from the user. There is one more function called read-line, which can be employed to get the user input as line. Read-line is primarily used to read the file. We shall discuss the same in detail at a later stage.

3.2 String Handling Functions :

1. strcase

(strcase <string> [<flag>])

This function takes a string and depending on the <flag>, it returns the same sequence of alphabetical characters converted to upper or lower case.

Example:

(strcase "Bharuch")	returns	BHARUCH
(strcase "gUjArat")	returns	GUJARAT

<string> is the first argument whose case is to be changed. If optional second argument is not provided, <string> is converted to upper case. If optional parameter <flag> is supplied and evaluates to non-nil then the returning string is in lower case. If <flag> is evaluated to nil string is converted to uppercase.

Example:

(strcase "Bharuch" T) returns bharuch
(strcase "GUJARAT" 1) returns Gujarat

Let us assume that your program needs the name of a company as one of the input. You want to be sure that name of the company always appears in Capital letters, then you can use strcase function before getstring call. This way whatever case is used for entering company name is sure to get converted into Capital before storage in variable name.

(setq name (strcase (getstring "\n Enter name of company:")))

2. strcat

(strcat <string1> <string2> ...)

This function concatenates the strings supplied as arguments. Concatenating means joining the strings and space with another string or simple space.

Example:

(strcat "Year" "2000") returns "Year2000"
(strcat "Mr." "Gandhi") returns "Mr.Gandhi"
(strcat "Mr." " " "Gandhi") returns "Mr. Gandhi"

The joining of strings is not limited to strings and spaces but any variable that contains the string value can also be joined.

(setq city "London")
(setq name "Mohmed Zuber")
(setq year 2004)

(strcat name "was in" city "in year" (itoa year))

check out the result, you know where I was in year 2004.

2. strlen

(strlen <string>...)

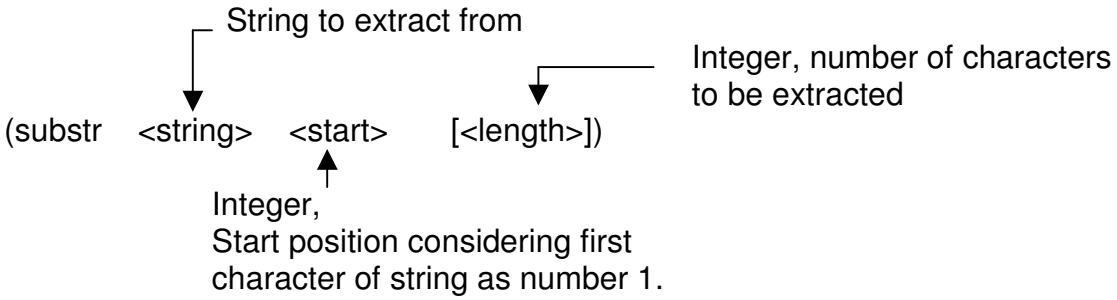
This function counts the total characters of the string and returns the number as an integer. Space is also counted as a character.

Example:

```
(setq  name  "AutoCAD")
(strlen  name)                  returns      7
(strlen  "movie" "titanic")     returns      12
```

The strlen can take a variable number of parameters and returns the sum of all their lengths.

3. substr



This function allows us to extract the some portion of string to form a new substring. If length is not provided, the substring continues to the end of <string>.

Example:

```
(substr  "AutoCAD"  5)      returns      "CAD"
(substr  "AutoLISP" 1  4)   returns      "Auto"
```

Following example demonstrate the use of STRLEN and SUBSTR. As we have not discussed While and if condition so far, you may find it little difficult to understand. Some functions used in this program are yet to be discussed, but are surely explained at some location in this literature. After loading this program in AutoCAD, a user defined function called TUD will be available for use. Write TUD on AutoCAD command prompt and you are ready to write text in Step UP or Step Down style.

txtupdown.LSP

```
(DEFUN C:TUD (/ txt step txtht stpt txtlength HV cnt exch)

        (setq txt (getstring T "\n Enter Text : "))

        (initget 1 "u d")
        (setq step (getkword "\n Step Up or Down <u/d> :"))
        (initget 7)
        (setq txtht (getreal "\n Enter Text Height : "))
        (initget 1)
        (setq stpt (getpoint "\n Select point for text writing : "))
        (setq txtlength (strlen txt))
        (setq HV (* txtht 2))
        (setq cnt 1)
        (while (<= cnt txtlength)
               (setq exch (substr txt cnt 1))
               (command "text" stpt txtht 0 exch)
                 (if (= step "u")
                    (setq stpt (list (+(car stpt)hv)(+(cadr stpt)hv)))
                    (setq stpt (list (+(car stpt)hv)(-(cadr stpt)hv)))
                 )
               (setq cnt (+ cnt 1))
        )
        (princ)
)
```

3.3 Data Type Conversion Functions:

When your program accepts the numeric data as a string and you wish to use it for mathematical operations, you need a conversion function. If you are reading the data from a file, all data is considered string, but to utilize the data some conversion functions are required to convert it.

1. atoi

(atoi <string>)

This function returns the conversion of string to an integer.
atoi - is called ASCII to integer.

Example:

(atoi "223")	returns	223
(atoi "3.38")	returns	3

```
command: (setq  Telno  "226960")
"226960"

command: (/  Telno  2)
 ; error: bad argument type: numberp: "226960"

command: (atoi  Telno)
226960

command: (/  Telno  2)
113480
```

4. atof

(atof <string>)

This function returns the conversion of a string in to a real.
atof - is called ascii to float.

Example:

```
(atof  "3.38")          returns     3.38
(atof  "3")             returns     3.0
```

5. angtos

(angtos <angle> [<mode> [<precision>]])

This function converts angles expressed in radian to a string according to the settings of <mode>, <precision> and the AutoCAD DIMZIN dimensioning variable.

The <mode> supported are ...

Mode	Editing format
0	Degree
1	Degree/minutes/seconds
2	Gradians

3	Radians
4	Surveyor's units

The <precision> should be expressed in the integer which selects the number of decimal places of precision desired.

Example:

command: (setq ang1 1.5708)
1.5708

(angtos ang1 0 0) returns "90"
(angtos ang1 0 2) returns "90.00"

4. itoa

(itoa <int>)

This function returns the conversion of an integer to a string.
itoa - is called integer to ascii.

Example:

(itoa 78) returns "78"
(setq *no* 210) returns 210
(setq *no* (itoa *no*)) returns "210"

Now if you try to do any mathematical operation on *'no'* variable you will get a bad argument because it is not a number.

6. rtos

(rtos <number> [<mode> [<precision>]])

This function returns a string of <number> expressed in real numbers considering <mode>, <precision> and DIMZIN dimensioning variable.

mode values supported are...

RTOS Mode	Editing format
1	Scientific
2	Decimal
3	Engineering
4	Architectural
5	Arbitrary fraction units

Example:

```
(setq  num  15.5)
(rtos  num  1  4)        returns    "1.5500E+01"
(rtos  num  2  2)        returns    "15.50"
(rtos  num  3  2)        returns    "1'-3.50""
(rtos  num  4  2)        returns    "1'-3 1/2""
(rtos  num  5  2)        returns    "1/ 1/2 "
```

Please note that if you skip the mode and precision arguments, the current values of LUNITS and LUPREC will be used respectively.

6. fix function converts a number from real to integer.

7. float function converts a number from integer to real.

8. Ascii

(ascii <string>)

This function returns the conversion of the first character of string into its ASCII character code in integers.

```
(ascii  "a")              returns    97
(ascii  "Bharuch")        returns    66
```

9. chr

(chr <ascii number>)

This function returns the conversion of the ascii code in integers to equivalent characters.

```
(chr  97)                 returns    "a"
```

3.4 Controlling The User Input:

Specific task oriented programs are designed for accepting a fixed set of data in a prescribed sequence. There is the chance of data entry errors and so validation is a part of all well written programs.

Suppose your assignment is to design a program for a gear drawing. You may wish to ask the user the total number of gear teeth required, if the user responds with zero or a negative number, there must be a check in your program to disallow such invalid entries. Such control of user input, or the validation of data, can be carried out in AutoLISP using the INITGAT function in combination with the "get" family of functions.

1. initget

(initget [<bits>] [<string>])

This function presets the options for use by the next GETxxx family of functions.

```
(initget 1)
(setq pt1(getpoint "\nEnter starting point:"))
```

if you include (initget 1) before calling getpoint function in Rect.lsp program. Null input will not be allowed in response to Enter starting point :. If the user simply presses enter without giving a valid input AutoCAD keeps reissuing "Enter starting point :" until a valid point is entered.
Let's see the bit values honored by AutoCAD.

INITGET bits	Meaning	Valid for …
1	Null value not allowed	Except Getstring and Getvar all Getxxx family functions.
2	Zero value not allowed	Getint, Getreal, Getdist, Getangle
4	Negative value not allowed	Getint, Getreal, Getdist
8	Do not check limits even if LIMCHECK is on	Getpoint, Getcorner

16, 32 and 64 bit values are not of much importance at this stage. The bits can be used in the combination of values to validate the data. Suppose you wish to restrict the user input for null, zero and negative, then you can write...

```
(initget  (+ 1  2  4))
```

OR

```
(initget  7)
```

… and of course you have to include the initget statement before calling the getint, getreal and getdist.

2. getkword

```
(getkword  [<prompt>])
```

The getkword function requests a keyword from the user. INITGET will have to be used to set valid keywords. At getkword's command, AutoCAD will compare the user input with the preset keyword initialized using the INITGET function, as a string. If the matching keyword is available, AutoCAD returns it as a string, otherwise the prompt will be repeated.

Example:

```
(initget  1  "Yes  No"))
(setq ans (getkword "Do you wish to continue...(Yes OR No)?"))
```

In response to "Do you wish to continue... "if the user enters "maybe" , it will not be accepted and AutoCAD will ask to try again. Only "yes" OR "y" OR "ye" OR "YES" OR "NO" OR "no" OR "N" OR "n" OR combination of letters it contains, shall be the acceptable user entry.

3.5 List Manipulating Functions :

As we have already discussed, the manipulation of lists is the heart of programming. We shall experiment with the manipulating functions on the list...

```
(setq  Lst1 '(a b c d e f g h i j ))
```

1. nth

```
(nth  <number>  <list>)
```

Earlier we saw that first element of a list can be accessed through car, second with cadr, third with caddr and so on...but for accessing 20th element of a list or if we wish to access the all elements of list one by one, then nth function would be the only solution. **nth** function returns the "nth" element of <list>, where <number> is the number of elements to return.

PLEASE NOTE THAT ZERO [0] IS THE FIRST ELEMENT OF LIST.

Example:

```
(setq zeroelt  (nth  0  Lst1))        returns        a
(setq sixelt     (nth  5  Lst1))      returns        f
```

2. Length

```
(length  <list>)
```

This function returns an integer value equivalent to the total number of elements in a list.

```
(length  Lst1)                        returns        10
```

3. Last

```
(last  <list>)
```

This function returns the last element of the list if the list is not nil.

```
(last  Lst1)                          returns        j
```

4. Append

(append <expression>...)

This function accepts any number of lists as arguments and ties them together as a single list. The function is very useful for reading data from files and storing all the data as elements of a single list.

Example:
(setq Lst2 '(k l m n o p q r s t))
(setq Lst1 (append Lst1 lst2))
returns (A B C D E F G H I J K L M N O P Q R S T)

7. Apply

(apply <function> <list>)

This function applies the function or operator of AutoLISP, specified as <function> on list specified as <list>.

Example:

(setq Lst3 '(10 100 1000))
(apply '+ Lst3) returns 1110

8. cons

(cons < element to be added <list>)
as first element >

This function accepts the atom as the first argument and the list as second argument. It returns the list with addition of that element to the beginning of the list.

Example:

(cons 'z Lst1) returns (Z A B C D E F G H I J)

(cons '(z) Lst1) returns ((Z) A B C D E F G H I J)

9. Member

(member <expression> <list>)

This function searches the list for an occurrence of an expression and returns the reminder of list starting with the first occurrence of expression.

(member 'e Lst1) returns (E F G H I J)

(member 'v Lst1) returns nil

10. Reverse

(reverse <list>)

This function returns the list with all the elements reversed.

(reverse Lst1) returns (J I H G F E D C B A)

3.6 Other Useful Functions :

1. angle

(angle <point1> <point2>)

This function returns the angle in radian of a straight line between two UCS points supplied as arguments. The angle is measured from the X axis of the current construction plane. If 3D points are supplied as arguments, points are projected in to the current construction plane to measure the angle in radian.

Example:

(angle '(2 3) '(6 3)) returns 3.14139

Suppose you wish to find out the included angle <pt2,pt1,pt3 in Rect.lsp program, you can write the code..

(setq ang1 (angle pt1 pt3))

2. Distance

(distance <point1> <point2>)

This function returns the distance between the two points supplied as arguments. Distance function is capable of supplying 3D distance if arguments are 3D points.

Example:

(distance '(3 4) '(5 4)) returns 2.0

(distance '(3 4 4) '(5 4 8)) returns 4.47214

3. Inters

(inters pt1 pt2 pt3 pt4 <onseg>)

The inters function examines two lines and returns the point where they intersect or nil if they do not intersect.

pt1, pt2 and pt3, pt4 are the end points of first and second line respectively.

For example :

```
(setq pt1  '(1.0 7.0))
(setq pt2  '(6.0 7.0))

(setq pt3  '(4.0 1.0))
(setq pt4  '(4.0 8.0))

(setq pt5  '(2.0 1.0))
(setq pt6  '(2.0 4.0))
```

(inters pt1 pt2 pt3 pt4) returns (4.0 7.0)

(inters pt1 pt2 pt5 pt6) returns nil

If an optional argument <onseg> is present and is nil, inters returns the point where they will intersect each other when they are extended.

(inters pt1 pt2 pt5 pt6 nil) returns (2.0 7.0)

4. Getvar

(getvar <system variable name>)

5. Setvar

(setvar <system variable name>)

You can access AutoCAD's system variables with AutoLISP's getvar and setvar functions. Getvar is a function, not a command, Setvar is the name of AutoLISP function and also the name of the AutoCAD command.

The best practice for programming in AutoLISP is to first get the system variables value in the program and store them in dummy variables and then setting the system variables per the requirement of your program, and at the end of the program return the value of the system variables using the dummy variables which you set in the beginning. This practice ensures the programs portability from one computer to other, without damaging the settings of others computers.

The following example highlights this practice. AutoLISP code is written in small letters, so I've used CAPITALS to indicate my comments for you!

1. COMMENTS RELATED TO PROGRAM
2. ANNOUNCEMENT OF UTILITY AND DATA INPUT
3. USER DEFINED FUNCTIONS
4. CAPTURING OF SYSTEM VARIABLES STARTS NOW .....

```
(setq dmyasz (getvar "dimasz")
(setq dmytxt (getvar "dimtxt")
 |
AND MANY MORE PLEASE REFER THE SYSTEM VARIABLE TABLE
 |
```

5. SET THE SYSTEM VARIABLES AS PER YOUR REQUIREMENTS

```
(setvar "dimasz" 0.10)
(setvar "dimtxt" 0.12)
```

6. WRITE YOUR PROGRAM CODE HERE
7. RESET THE SYSTEM VARIABLES CAPTURED EARLIER.

```
(setvar "dimasz" dmyasz)
(setvar "dimtxt" dmytxt)
```

8. PROGRAM ENDS HERE.

Following system variables may interest you....

(setvar "cmdecho" 0)	code does not prompt
(setvar "EXPERT" 3)	suppress prompts issued by linetype
(setvar "mirrtext" 0)	reflects text if nonzero.
(setvar "dimasz" 0.10)	for changing arrow size.
(setvar "dimtxt" 0.12)	for changing text size.
(setvar "dimse1" 1)	for suppressing dimension line
(setvar "dimse2" 1)	for suppressing dimension line
(setvar "lunits" 2)	units mode
(setvar "luprec" 3)	linear units decimal places.

4. Osnap

(osnap <point> <AutoCAD Osnap mode as string>)

This function allows you to capture the 3D point of a drawing entity on applying AutoCAD object snapping values in string on that entity. Say, your program draws a line between two points, pt1 and pt2. You wish to find out the midpoint of the line joining pt1 and pt2 points. Your object is to draw more entities from the midpoint. You can use osnap to find out the mid point.

```
(setq pt1 (getpoint "Enter First Point :"))
(setq pt2 (getpoint "Enter Second Point :"))

(command "line" pt1 pt2 "")

(setq mpt1pt2 (osnap pt1 "mid"))
```

The coordinates of the midpoint of the line joining points pt1 and pt2 will be stored in mpt1pt2 variable.

if you write..

```
(command "line" mpt1pt2 "@3<45" "")
```

there will be a 3 unit long line aligned at 45 degree and starting from mid of line joining pt1 and pt2. You can also apply more than one snap identifiers as a second argument to the osnap function.

```
(setq pt3 (osnap pt1 "midp,center"))
```

5. Polar

(polar <point> <angle> <distance>)

This function returns the UCS point at angle <angle> and distance <distance> from the point <point>.

The following example will clarify this. A circle is drawn with point1 as the centre, having a radius r, and you wish to find out the coordinates of a point on the circumference of this circle forming 33 degree at the center, then...

```
(setq point1 '(2 3))
(setq r 3.0)
```

```
(command "circle"  point1  r)
(setq pt_on_cir (polar  point1 0.575959  r))
(command "line" pt_on_cir point1 "")
```

6. Exit
   Quit

The (exit) or (quit) function forces the current application to quit. Both function display error massage and return you to AutoCAD command prompt.

7. Read

```
(read  string)
```

This function is used to convert strings to lists or symbols, integers or real. The function returns the first atom or list obtained from string.

Example :

```
(read  "3")                 returns an integer 3.
(read "(1 2 3)")            returns list ( 1 2 3 ).

(set  (read  "cnt" ) 10)     returns  10 and the  variable
                             cnt is assigned  a  value  10.
```

Suppose we need to define 20 variables v1 to v20 which house integers for some load calculations. You can either use 20 setq lines for defining v1 to v20 variables or use the following functions to accept 20 values from user. This will be something like the array concept of other languages.

```
(DEFUN  arr(/ cnt)
        (setq cnt 1)
        (REPEAT 20
                (set (read (strcat "v" (itoa cnt)))
                        (getint "\n Enter Integer Value : ")
        )
        (setq cnt (+ cnt 1))
        )
)
```

On loading, this function waits for the user to key in an integer. Whatever the user keys in will be stored in the variable v1 during the first iteration, v2 during second and so on.

8. Type

(type <item>)

This function returns the type of <item>. Following are the details of response.

INT	-	integers
REAL	-	floating point numbers
STR	-	strings
FILE	-	file descriptors
SYM	-	symbols or variables
LIST	-	lists and user functions
SUBR	-	AutoLISP predefined functions or subroutines
PICKSET	-	selection set
ENAME	-	entity names
PAGETB	-	function paging table.

Branching and Decision Making

Making AutoLISP more Logical

We have noticed in our earlier programs that if no options or no repetitions of certain statements are required, the statements are executed sequentially in the order of their appearance. Though in actual programming we face situations where we may have to change the order of execution of statements based on certain facts or conditions, even repetition of certain statements required to extract the best probable result. This involves imparting "decision making" which can be achieved through statements like, if, while and repeat.

AutoLISP allows you to decide between one task and another based on some existing conditions using control structure functions, which also control the flow of logic. You might think of this ability as a way for a program to make decisions by testing whether certain conditions are true or false. Another important requirement of problem solving is repetitive computation. Many repetitive, tedious and time consuming tasks can be automated using AutoLISP.

4.1 AutoLISP Predicates

Predicate is a function which tests its parameters and returns a nil or some other non-nil value. Predicate functions end with **p** [like minusp] with some exceptions.

Function Returns True IF Predicates	
=	two numeric or string values are equal
/=	two numeric or string values are NOT equal
eq	two values are exactly the same
equal	two values are the same
<	a numeric value is less than another
>	a numeric value is greater than another
<=	a numeric value is less than or equal to another
>=	a numeric value is greater than or equal to another
atom	object is atom [decide between atom and list]
boundp	Variable has a value bound to it
listp	object is list [decide between list and atom]
minusp	a numeric value is negative
numberp	an object is a number, real or integer
zerop	an object evaluates to zero

Logical Operators

and	all of several expressions or atoms return non-nil
not	a symbol is nil
nul	a list is nil
or	one of several expressions or atoms returns non-nil

(= atom atom)

The '=' function takes a variable number of parameters and checks for the equality of all the parameters. If all the parameters are equal, then this function returns a T or else a nil.

This function is used to check for equality when the parameters happen to be atoms and not lists.

Example:

```
command: (= (* 3 4) (+ 6 6))
T
command: (= "walmi" "Walmi")
nil
command: (= '(1 2) '(1 2))
nil
```

EQ

Eq mainly operates on lists. In AutoLISP this is used primarily to determine whether two variables are bound to the same list or not.

Try out...

```
command: (setq a '(2 4))
(2 4)

command: (setq b '(2 4))
(2 4)
command: (setq c b)
(2 4)
command: (eq a b)
nil
command: (eq b c)
T
```

EQUAL

(EQUAL expr1 expr2 [fuzz])

```
command: (setq  m  3.14286)
3.14286
command: (setq  n  (/  22.0 7))
3.14286
command: (equal  m  n)
nil
command: (equal  m  n  0.00001)
T
```

ATOM

This predicate function returns T if the parameter passed on to it is an atom, and nil if it is a list.

```
command: (atom  2)
T
command: (atom  '(3 5))
nil
```

Predicates ending with p are self-explanatory. Try them out on your own.

AND

Predicates could be glued together by this predicate function. The and function returns a true only if all the parameters evaluate to a TRUE. This function evaluates its parameters till it encounters a parameter that evaluates to a nil. If it doesn't it returns a T.

Syntax:
(and expr...)

```
Command: (and  (< 3 4) (> 5 3))
T
Command: (and   2  4)
T
Command: (and   1  nil)
nil
```

NOT

Predicates could be combined using the not function. The not function takes just a parameter, and if the parameter evaluates to a nil, then this function will return a T.

Syntax:
(not item)

Command: (not nil)
T
Command: (not 1)
nil
Command: (not (= 3 4))
T

OR

The or like and takes a variable number of parameters and returns T if either of its parameters return a value other than nil.

Command: (or 1 nil)
T
Command: (or 5 6)
T

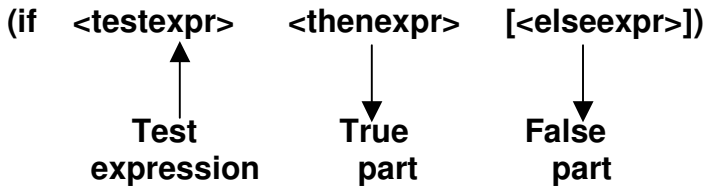
4.2 Decision making with IF statement :

The if statement is common to almost all programming languages and it is a powerful decision making statement with very simple syntax.

The roots of the if statement are taken from English language usage. A usual instruction for a person who is preparing for traveling is, IF GUJARAT EXPRESS TRAIN IS NOT ON TIME, CATCH THE BUS. So, it is a conditional branching of the movement of the traveler. If Gujarat express is not on time then and only then the traveler takes the bus. The same logic of if can be applied to computers to control the flow of execution of statements. If function allows the computer to evaluate the expression first and then, depending on whether the value of the test expression is TRUE or FALSE, it transfers the control to a particular statement.

[I] IF Function

The IF function conditionally evaluates expressions.



The control is branched to the true part when the condition evaluates to a true, or else, it branches to the false part.

Example:

```
(setq  a  (getint "Enter First Number  : "))
(setq  b  (getint "Enter Second Number : "))
          (if  ( = a b)
              (prompt  "This is true")
              (prompt  "This is false")
           ); closing of if
(prompt "\n Program complete")
```

if you enter both values the same "This is true" i.e first statement shall be evaluated. On entering both different numbers, computer shall skip the first statement and "This is false" shall be executed. The statements after closing parenthesis of if is executed in sequence, hence "Program complete" is displayed always irrespective of true or false condition.

Take a closer look at examples....

1.
```
(if (= 2  (+ 1 1))
  (prompt "Yes")
  (prompt  "No")
        )
```
Returns "Yes"

2.
```
(if (= 2  (+ 4 6))
  (prompt "Yes")
        )
```
Returns nil

You must have noted here that part of the statement is not provided because if the condition fails we don't wish to display or execute anything.

3.
```
(if
  (and (= a b) (> a c))
  (prompt "Both are true")
  (prompt "One is  false")
```

```
    )
4.  (setq no (getint "\n Enter Number : "))
        (if (minusp no)
      (prompt "Value entered is negative")
    )
```

Try Yourself....

5.	(if 1 10 20)	return	?
6.	(if "bar" 12 24)	return	?
7.	(if nil 20 30)	return	?
8.	(if nil 50)	return	?
9.	(if nil 50 (* 20 2))	return	?

Expression bunching
Progn

Many times programmers need to evaluate several expressions depending upon the result of predicate, Progn allows it.

```
(progn
            (expression1)
            (expression2)
                  |
                  |
            (expressionN)
)
```

Start your group of expressions with (progn. End the group with a closing parenthesis. Everything within the group is considered to be a single expression, or, in this case, a single expression even though the number of expressions within the group can be unlimited. You can also use (progn) for multiple expressions.

```
(if
   (= a b)
   (progn
     (prompt  "a is equal to b")
     (prompt  "b is equal to a")
     (prompt  "They are both equal to each other")
 ); end of progn
   (prompt  "a is not equal to b"); if condition fails
                                  ; we wish to display
                                  ; only one statement
                                  ; hence progn is not
```

```
                                        ; required.
            ); end of if condition


;Program to find out the perfect length of belt between two pulleys.
;29-Jul-99 7:36 PM


(setq S  (getstring "Enter type of drive 'OPEN'/ 'CROSS' (O/C):"))
(setq r1 (getreal "Enter radius of small pulley (r1):"))
(setq r2 (getreal "Enter radius of big pulley (r2):"))
(setq c  (getreal "Distance between two pulleys (c):"))

(IF (= S "o")
   (setq L (+ (* pi(+ r1 r2)) (* 2 c) (/ (expt (- r2 r1)) c)))
)

(if (= S "c")
   (setq L (+ (* pi(+ r1 r2)) (* 2 c) (/ (expt (+ r2 r1)) c)))
)

(prin1 L)
(princ)
```

4.3 ITERATION

There is one aspect, in which computers overshadow the brains of humans and that is capability of repeating the work. The repeating or looping capabilities enable us to develop concise programs containing repetitive processes.

4.4 While Function

While is a entry controlled loop consists of control statement <testexpr> and body of the loop <statements>. The test expression tests for conditions and if the conditions are satisfied, control passed on to loop statements again. If the conditions are not satisfied, then the loop statements are not executed at all.

(while <testexpr> <expr>...)

This function evaluates <testexpr> and, if NOT NIL, evaluates the other <expr>s and then evaluates <testexpr> again. This continues until <testexpr> is NIL.

Example:

```
(setq cnt 1)
(while (< cnt 5)
        (princ "\n ABC")
        (setq cnt (+ cnt 1))
)
(prompt "\n Program complete")
(princ)
```

As soon as cnt variable attains value 5, the while loop will terminate.

In other words the two expressions:

```
(princ "\n ABC")
(setq cnt (+ cnt 1))
```

…will be executed 4 times. During every iteration, the value of cnt is incremented. This code will hence, print "ABC" 4 times. Each time on a new line due to control character "\n". On exit, the program continue with the statements immediately at the end of while loop.

The test conditions with while shall have to be selected with due consideration and care, If you write (while (<= cnt 5) in above example, your loop shall repeated one more time to print ABC 5 times.

If you write (while T) in above example., printing of ABC continue unless you press ctrl C. The reason is T being a non nil value always pass control back to loop statements and hence it will be infinite loop and loop body is executed over and over again.

Program :

```
(setq lst1 '(a b c d e f g h i j ))
(setq cnt 0)
(setq len (length lst1))

(while  (<= cnt len)
        (setq ele_of_list (nth cnt lst1))
        (prin1 ele_of_list) (prin1  )
        (if (= cnt len)
           (princ)
           (prin1 cnt)
```

```
            )
            (setq cnt (+ cnt 1))
            (princ "\n")
        )
        (prompt "\n Program complete")
        (princ)
```

4.5 Repeat Function

Another function that performs iteration is the repeat function. Repeat works in a similar way to while but instead of using a predicate to determine whether to evaluate its arguments, Repeat uses an integer value to determine the number of times to perform an evaluation.

The Syntax is:

```
(repeat n
(expression1)
(expression2)
(expression3)
)
```

n can be integer OR a variable representing an integer.

Example:

```
(setq stpt1 '(3 9))
(setq cnt 1)
(repeat 11
            (command "text" "c" stpt1 "0.30" "0" cnt)
            (setq stpt1 (list (car stpt1)(-(cadr stpt1)0.5)))
            (setq cnt (+ cnt 1))
)
```

Let us try to evaluate the above code in a way similar to AutoLISP interpreter. As soon as you load above code the first line of code i.e. (setq stpt1 '(3 9)) shall be evaluated and co-ordinates (3 9) shall be stored as location point in the variable stpt1. [Please try to name the variable in some meaningful way so that program can be altered at later stage easily. In our case stpt1 stands for starting point 1.]

The second line to evaluate is (setq cnt 1) , where 1 is stored in variable cnt. The variable cnt stands for counter, if you can effectively use the counters in program the logical solution to the problem would be easy. Now the third line i.e. (REPEAT 11 , it clearly indicates that

whatever code nested in (REPEAT .....) parenthesis has to repeat itself 11 times. You can assigned an integer variable instead of 11. The fourth line shall write the first counter value [i.e. 1] at point stpt1. The fifth line shall shift the location of point stpt1 vertically to the magnitude of 0.5 unit. Now the Sixth line plays the magic, it adds 1 [one] in the last value of variable cnt. The assigned value of cnt was 1 , hence after first iteration the value in cnt would be...

```
(setq  cnt (+  cnt  1))
           +    1   1                    2 [two].
```

like that iteration continues up to 11 and value of stpt1 would go on changing and so as value of cnt. Can you guess the result .....?

Example:

```
; fun.lsp
(setq pt1 (getpoint "\nEnter insertion point for circle : "))
(setq r   (getdist  "\nEnter circle radius : "))
(command "circle" pt1 r)
(setq d1 (getdist "\nEnter inside dia of Donut : "))
(setq d2 (getdist "\nEnter outer dia of Donut : "))
(setq i (polar pt1 0 r))
(command "donut" d1 d2 i "")
(setq n (getint "\nNo. of Repetition : "))
(setq a (getint "\nEnter angle of rotation : "))

(REPEAT n
      (command "rotate" "l" "" pt1 a)
      (REDRAW)
)
```

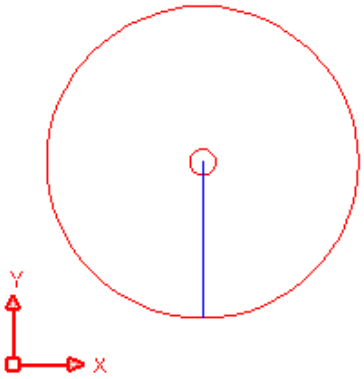
How can AutoLISP make design processes faster? Let's take a real life example.

SPIRAL STAIR

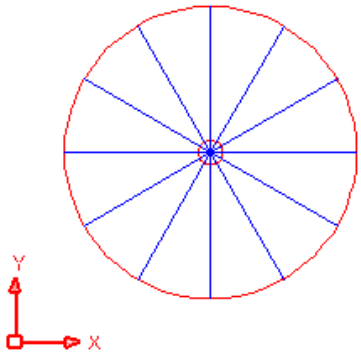
Spiral stairs are basically rotating around cylindrical support. The steps of the stair are designed in such a way that tread is minimum towards center support and moving up as per rise in circular angular pattern.

In a plan view, thecenter support is represented by circles and the outer circle represents the end of the width of the step. A line connecting the center point and quadrant at 270 is drawn and a polar array is performed on it to divide circle in 12 parts or parts required as per your design. The island between inner circle, outer circle and two consecutive array lines will form the shape of step.

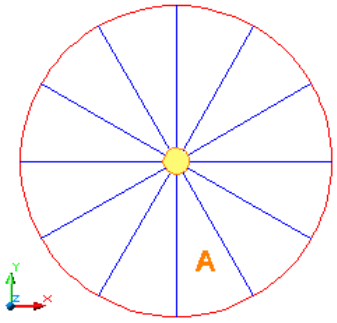
1. Draw 2D circles and line as shown



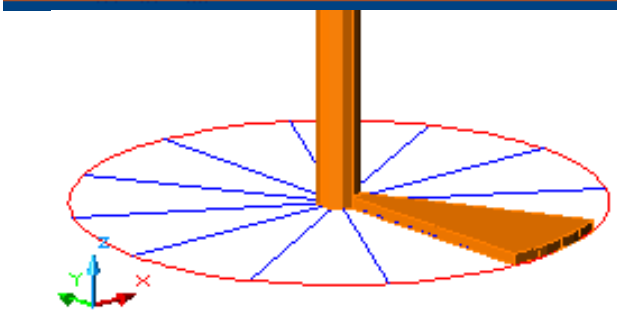
2. Perform polar array on line



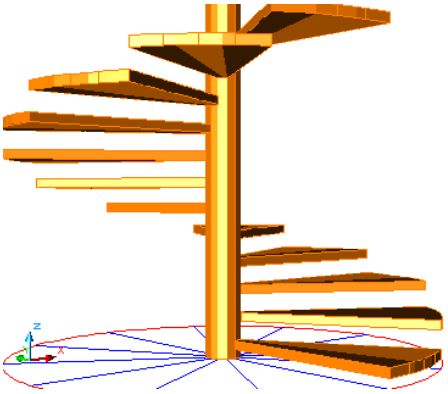
3. Create BOUNDARY by picking point at A.



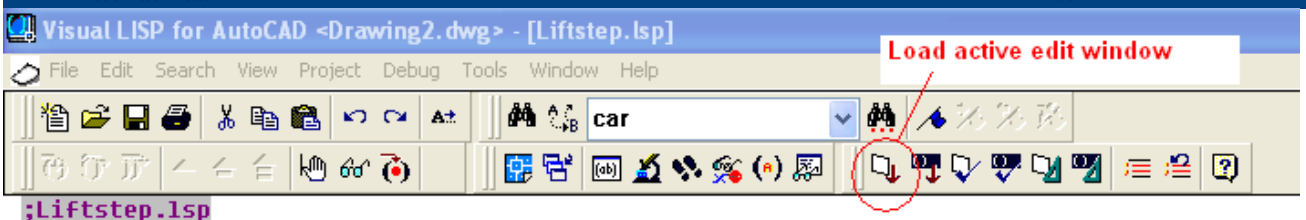
4. EXTRUDE this shape and also EXTRUDE center circle representing circular column. Result will be as shown in figure when viewed using VPOINT.



As you can see in the last figure, extruded shape is clearly visible resting on ground. Move this step in Z direction as per requirement of rise. For creating other steps you can repeat steps narrated above OR create polar array of first step and lift each step as per requirement of rise.



You will have to use move 20 times to lift 20 spiral steps in upward (Z) direction. Every step will be lifted with new value for second point of displacement in move command. Oh! I don't want to do doctorate on MOVE command. Is there any way available to do it fast and in simplified way? Yes. Use this AutoLISP program to make your life easy.




```
;Liftstep.lsp
;Program for Lifting of spiral steps in Z direction.
;User will have to specify RISE of a step.(Once Only) and then go on selecting steps in
;upword motion. The steps will be lifted as requirement of Rise

(DEFUN C:Lift (/ cnt rise mrise)

(setq cnt 1)
(setq rise (getreal "Step RISE value:"))

  (while T
       (setq mrise (* rise cnt))
       (command "move" pause "" "0,0,0" (strcat "0,0," (rtos mrise 2 3)))
       (setq cnt (+ cnt 1))
  )
)
```

This program can be written in NOTEPAD or Visual LISP editor by invoking VLIDE command at command prompt. Save your program with .lsp extension. Use (load “liftstep”) on AutoCAD prompt to add new command LIFT in AutoCAD. While drafting you can write LIFT on AutoCAD prompt to use this program. As a user of this program, you need to supply step Rise value once and then simply go on selecting steps in logical sequence. This program will lift steps in Z direction.

If you are using Visual LISP editor as shown in figure, Press “Load active edit window” button to load this program in AutoCAD.

Command: LIFT
Step RISE value: 0.15
move
Select objects: *Select First Step*
Select objects: *Select Second Step*

The Select objects prompt will continue, unless you press Esc key.

User Defined Function

LISP is an acronym for LISt Processing, which indicates that LISP works with lists. AutoLISP programming is significant customization option in AutoCAD. Repetitive and difficult tasks can be automated by creating small but powerful AutoLISP programs. One of the primary advantages of LISP is that it allows hierarchical program layering. The program does this by developing subroutines to perform primary functions.

Functions in AutoLISP are either :

Predefined
OR
User Defined

Predefined functions are functions that come with AutoCAD Software written by Autodesk Inc. USA. The predefined functions are faster and this can be attributed to the fact that they are compiled and stored in machine language. Setq is one such predefined function.

User defined functions are of course, functions that users write. These functions are written to a standard ASCII file using a word processor. AutoLISP routines are written in to a file with extension ". LSP". A single file may contain many subroutines and they have to be loaded in to the computer's memory to be usable.

5.1 DEFUN

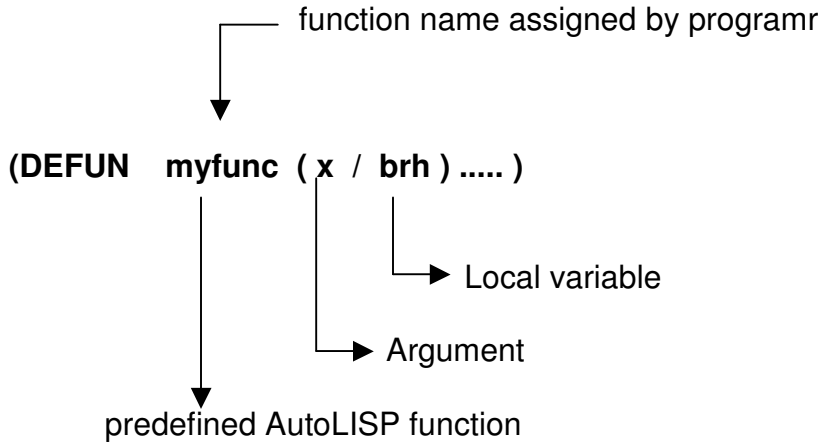
You can create your own functions with the help of a predefined AutoLISP functions called DEFUN [DEfine FUNction].

Syntax of DEFUN is ...

(DEFUN <sym> <argument list> <expr>...)

DEFUN defines a function with the name <sym>. It has an <argument list>, which may be void or contain global and/or local variables. The <expression> is the procedure that is to be processed when the function is called. The function begins with a left parenthesis and ends with right parenthesis.

Example :



The DEFUN function itself return the name of the function being defined. When the function so defined is invoked, its arguments will be evaluated and bound to the argument symbols. Local variables are symbols available only to your function. The use of local variables ensures that the variables in your functions do not make conflict with other functions variables. Local variable do not remain available after the calling function has completed its task.

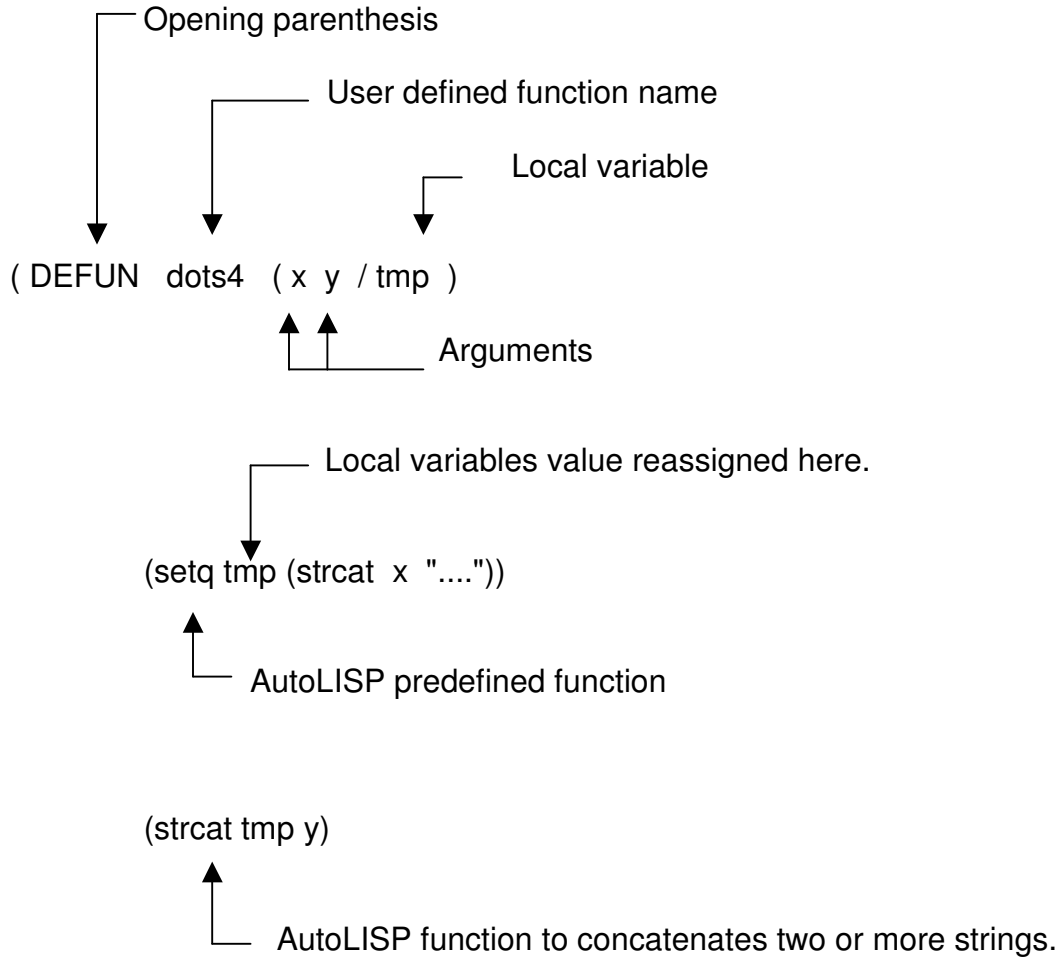
All functions loaded within a drawing can access *global variable*. Global variables may retain their value after the program that defined them completes. *Local variables* retain their value only as long as the function that defined them is running. After the function finishes running, the local variable values are automatically discarded, and the system reclaims the memory space the variable used. This is known as automatic garbage collection, and is a feature of most LISP development environments, such as VLISP. Local variables use memory more efficiently than global variables. Another big advantage is that local variables make it easier to debug and maintain your applications. With global variables, you are never sure when or in which function the variable's value might be modified; with local variables you don't have as far to trace. You usually end up with fewer side effects.

for example.....

```
(DEFUN   dots4   ( x  y  / tmp  )
                 (setq tmp (strcat  x  "...."))
                 (strcat tmp y)
)
```

returns DOTS4

```
(dots4   "CADD"  "Bharuch")   returns   CADD....Bharuch
(dots4   "CADD"   "LISP")      returns   CADD....LISP
```



) ;closing parenthesis

Predefined or Userdefined, all functions follows certain rules, such as....

a. functions are always written in standard ASCII characters.

b. upper or lower case writing is not important.

c. functions are enclosed in balanced parenthesis, and evaluated from left to right.

d. there must be at least one space separating the function name and its parameters. Other spaces are ignored.

e. functions can be nested to any level.

f. function returns the value of the last expression evaluated.

I think some examples of the above rules will clarify the idea of creating user defined functions.

5.2 User Defined Functions

[1]. Define a function to calculate the hypotenuse of a right angle triangle using the pythagorean theorem $c = \sqrt{(a2 + b2)}$

```
(defun  hypot (/ a  b)
                (setq  a  (getdist  "Base length"))
                (setq  b  (getdist  "Rise"))
                (setq  c  (sqrt (+ (* a a) (* b b))))
)
```

[2]. function for displaying 10 slide files having a name SL1, SL2, SL3 and so on till SL10. Slides are displayed at the interval of 1000 milliseconds.

```
(defun ss(/ oldcmd cnt)
                (setq oldcmd (getvar "cmdecho"))
                (setvar "cmdecho" 0)
                (setq cnt 1)
                (while (<= cnt 10)
                        (command "vslide" (strcat "SL" (itoa cnt)))
```

```
                    (command "delay" "1000")
                    (setq cnt (+ cnt 1))
                )
                (setvar "cmdecho" oldcmd)
    )
```

[3]. fprint usage : (fprint "filename.ext")

This function prints an ASCII text file on the screen.

```
(defun fprint (s / c i)
    (setq i (open s "r"))
    (if i
        (progn
            (while   (setq c (read-line i))
                (princ c)
                (princ "\n")
            )
        )
        (princ (strcat "cannot open file"  s ))
    )
    (princ)
)
```

5.3 Adding commands to AutoCAD

You can add new commands to AutoCAD by using DEFUN to define functions implementing those commands. If you want to frame a function to use as AutoCAD command, please adhere to following rules.

a. The function must have "C:" as the immediate prefix to name of the user defined function. As for example "C:MYFUNC", where all letters are upper case.

b. The function must be defined with a nil argument.

Let us write a few utility functions which would improve efficiency, while working in AutoCAD.

You can store such functions in **UTIL.LSP.** At the time of actual use of such functions, you can load util.lsp file and all user defined functions written in this file will be available to you.

[1]. MARK

This user defined function places the encircled number or character at the desired location. Such type of function can be very useful in marking the node numbers/characters for contour, electrical circuits and PCBs.

```
(DEFUN C:MARK (/ ch stno ud cnt flag stpt stch)
        (initget 1 "c n")
        (setq ch(getkword "\n Choose Charactor or Number (c/n) :"))
        (if  (or (= ch "n") (= ch "N"))
        (progn
        (initget 5)
        (setq stno(getint "\n Enter starting number :"))
      (initget 5)
        (setq txtht (getdist "\n Text Height:"))
        (initget 1 "I D" )
        (setq ud (getkword "\n Decide order of number, Increment or Decrement (I/D) :"))
        (setq cnt stno)

        (setq flag 1)
        (while (= flag 1)
          (setq stpt (getpoint "\n pick the point :"))

           (if (< cnt 100)
              (command "circle" stpt (* txtht 1.25))
              (command "circle" stpt (* txtht 1.75))
           )

            (command "text" "m" stpt txtht 0 cnt)
          (if (or (= ud "I")(= ud "i"))
            (setq cnt(+ cnt 1))
            (setq cnt(- cnt 1))
          )

      )
        )
      (progn
      (setq stch(getstring "\n Enter starting charactor :"))
        (initget 5)
        (setq txtht (getdist "\n Text Height:"))
        (initget 1 "I D" )
        (setq ud (getkword "\n Decide order of character, Increment or Decrement (I/D) :"))
```

```
(setq cnt (ascii stch))
(setq flag 1)
(while (= flag 1)
   (setq stpt (getpoint "\n pick the point :"))

      (command "circle" stpt txtht)

      (command "text" "m" stpt txtht 0 (chr cnt))

   (if (or (= ud "I")(= ud "i"))
      (setq cnt(+ cnt 1))
      (setq cnt(- cnt 1))
   )
(if (or (= cnt 123) (= cnt 91))
   (setq cnt (- cnt 26))
   )
   (if (or (= cnt 96) (= cnt 64))
         (setq cnt (+ cnt 26))
   )
)

)
)
         ); end of defun
```

[2]. Creating Mirror copy of current file at same path location

```
;mysave.lsp
;saving mirror image of current file
;When this command is invoked in AutoCAD, it saves current file and create new
;duplicate file at same path but prefixing the file name with "Mirror".
;For example Bridge.dwg will be saved as MirrorBridge.dwg

;Program By: Mohmed Zuber Shaikh

(DEFUN C:MYSAVE (/ path mrr flnm newname original)
(command "_.qsave")
(setvar "filedia" 0)
(setq path (getvar "DWGPREFIX"))
(setq mrr "Mirror")
(setq flnm (getvar "DWGNAME"))
(setq original (strcat path flnm))
(setq newname (strcat path mrr flnm))
```

```
  (if (= newname nil)
     (command "_.saveas" "" newname)
     (command "_.saveas" "" newname "y")
  )

(command "_.saveas" "" original "y")
(setvar "filedia" 1)
)
```

5.4 Automatic Function Definition

Each time an AutoCAD drawing editor session begins, AutoLISP loads the file "ACADxxxxdoc.LSP". You can put the definitions of commonly used functions in this file, and they will be defined automatically each time you begin editing a drawing. Automatically defined doesn't mean automatically being executed. Four x followed by ACAD indicates the change you have suppose to make as per your AutoCAD software version. i.e. for AutoCAD 2006, the file is ACAD2006doc.lsp For pre AutoCAD 2006 versions, this file can be ACADxxxx.lsp or something else for very old versions. Please determine the sequence according to your version. In our case if you include the code of MARK & MYSAVE in ACAD2006doc.LSP file, you can use them as if it is an AutoCAD command.

My Dear participants, in next segment we will try to explore File handling operations which includes reading and writing of file through program. The Segment_3 will include a sample Road profile program to appreciate the power of AutoLISP.

I know – When I am teaching, I am learning more.

Remember that this material is only a portion of the class; support is always available online in the private course forum. I encourage you to visit the course forum and ask any questions that you may have about this segment or simply join in the discussion. The ATP Mantra is: the only stupid question is the one you don't ask. Thanks again for attending this course!