Managing What your LISP Routines Manage
IT18197-L

Craig Black

Instructor of Mechanical Design, Fox Valley Technical College, Appleton WI, United States

Facebook: Craig Black

Twitter: craigpblack

Craig Black

- Working with Autodesk products since 1985
- Teacher at Fox Valley Technical College since 1990
- No idea how many classes I have taught at AU
- Co-author of AutoCAD and Its Applications – Advanced
- Technical editor of AutoCAD Platform Customization: AutoLISP by Lee Ambrosius
- Working part time as a programmer/analyst for KI furniture manufacturer in Green Bay, WI
- Enjoys golfing and cheering for the Green Bay Packers

Green Bay Packers Hall of Fame

Lab Assistants

- Alex Lepeska
- John Jordan
- Lee Ambrosius

Download the data set to the desktop

I will go look over a student's shoulder and talk you through this!

This will be our first attempt at working together and seeing how well you listen to me (and I to you!) ☺

Class summary

- Writing AutoLISP code is one thing, troubleshooting it, and restoring the system “back to normal” when crashes occur while troubleshooting can be a headache. Take this class and take away your headaches. Learn how to quickly get things back to normal. Learn how to use lists to your advantage – I mean, LISP does stand for List Processing! Learn how to create your own error routine to set things back the way they were before your fancy new program crashed and burned. Crashing and burning is part of the process of getting a new time saving routine off the ground – this class will make those crashes less painful. This session features AutoCAD. AIA Approved

Key learning objectives

At the end of this class, you will be able to:

- Write shortcut routines that put things back to how they were
- Use lists to store system variables that the program will modify
- Create your own custom error routines
- Understand more of the subtle nuances of how LISP works

This class is meant for those with intermediate knowledge of AutoLISP. Some of you may be entry level, others may be advanced. Keep in mind that every paragraph, every slide doesn't have to be a big eye-opener… if you take away "a little something", these 90 minutes will be a success!

View from My Bedroom Window…

Learning Objective:
Shortcut to get back to where you were…

(before you screwed things up…)

FavSet.lsp

FavSet.LSP

```
(defun c:FAVSET ()
  (setvar "CMDECHO" 0)                  ; turns off (command) prompting
  (setvar "CLAYER" "OBJECT")            ; sets current layer to OBJECT
  (setvar "OSMODE" 39)                  ; sets object snaps to END, MID, CEN, INT
  (setvar "POLARANG" (/ pi 12))         ; sets polar tracking angle to 15 degrees
  (setvar "PICKBOX" 4)                  ; sets pickbox size to 4 pixels
  (setvar "AUNITS" 0)                   ; sets angular unit type to decimal degrees
  (setvar "AUPREC" 2)                   ; sets angular unit precision to 2 places
  (setvar "LUNITS" 2)                   ; sets linear unit type to decimal
  (setvar "LUPREC" 4)                   ; sets linear unit precision to 4 places
  (setvar "FILLETRAD" 0)                ; sets fillet radius to 0
  (setvar "LTSCALE" 0.5)                ; sets linetype scaling to .5
  (setvar "AUTOSNAP" 63)                ; turns on all AutoSnap features
  (setvar "TEXTSTYLE" "Regular")        ; sets the current textstyle to Regular
  (command "dimstyle" "r" "DEC-2PL")    ; sets the current dimstyle to DEC-2PL
  (princ)
)
```

Exercise No. 1

- Open the Test Settings.dwg file
- Open the Visual LISP Editor
- Start a new program file
- Type the above lines of code shown in Figure 1
- Save the file as FAVSET.lsp
- Press the “Load active edit window” button
- Switch to AutoCAD
- Change the current layer to something other than Object
- Change the running object snaps to QUAdrant, PERpendicular, and NODe
- Type PICKBOX and set it to 25
- Turn off Polar Tracking and Object Snap Tracking
- Scan the screen and mentally note what you have just changed
- Type FAVSET at the Command: prompt
- Scan the screen and note that things have been set back to “normal”

```
(defun c:FAVSET ()
  (setvar "CMDECHO" 0)
  (setvar "CLAYER" "OBJECT")
  (setvar "OSMODE" 39)
  (setvar "POLARANG" (/ pi 12))
  (setvar "PICKBOX" 4)
  (setvar "AUNITS" 0)
  (setvar "AUPREC" 2)
  (setvar "LUNITS" 2)
  (setvar "LUPREC" 4)
  (setvar "FILLETRAD" 0)
  (setvar "LTSCALE" 0.5)
  (setvar "AUTOSNAP" 63)
  (setvar "TEXTSTYLE" "Regular")
  (command "dimstyle" "r" "DEC-2PL")
  (princ)
)
```

Interview at the Hall of Fame

Learning Objective:
Use Lists to Store System Variables that the Program will Modify

(It is called *LIS*t *P*rocessing for a reason!)

The Old Way

3

```
(defun C:LISTSTORE (/              |CMDECHO|     |CLAYER|      |OSMODE|
                    |POLARANG|     |PICKBOX|     |AUNITS|      |AUPREC|
                    |LUNITS|       |LUPREC|      |FILLETRAD|   |LTSCALE|
                    |AUTOSNAP|     |TEXTSTYLE|   |DIMSTYLE|    PT1
                    PT2           PT3
                   )
```

1

```
  (setq |CMDECHO| (getvar "CMDECHO"))
  (setq |CLAYER| (getvar "CLAYER"))
  (setq |OSMODE| (getvar "OSMODE"))
  (setq |POLARANG| (getvar "POLARANG"))
  (setq |PICKBOX| (getvar "PICKBOX"))
  (setq |AUNITS| (getvar "AUNITS"))
  (setq |AUPREC| (getvar "AUPREC"))
  (setq |LUNITS| (getvar "LUNITS"))
  (setq |LUPREC| (getvar "LUPREC"))
  (setq |FILLETRAD| (getvar "FILLETRAD"))
  (setq |LTSCALE| (getvar "LTSCALE"))
  (setq |AUTOSNAP| (getvar "AUTOSNAP"))
  (setq |TEXTSTYLE| (getvar "TEXTSTYLE"))
  (setq |DIMSTYLE| (getvar "DIMSTYLE"))

  (setvar "CLAYER" "HIDDEN")
  (prompt (strcat "\nCurrent layer was changed to "
                  (getvar "CLAYER")
                  "."
          )
  )
  (setvar "OSMODE" 0)
  (prompt "\nRunning object snaps were turned off.")
  (setvar "AUTOSNAP" 0)
  (prompt "\nPolar and object snap tracking were turned off.")
  (setq PT1 (getpoint "\nPick a first point: "))
  (setq PT2 (getpoint "\nPick a second point: "))
  (setq PT3 (getpoint "\nPick a third point: "))
  (command "line" PT1 PT2 PT3 "Close")
```

2

```
  (setvar "CMDECHO" |CMDECHO|)
  (setvar "CLAYER" |CLAYER|)
  (setvar "OSMODE" |OSMODE|)
  (setvar "POLARANG" |POLARANG|)
  (setvar "PICKBOX" |PICKBOX|)
  (setvar "AUNITS" |AUNITS|)
  (setvar "AUPREC" |AUPREC|)
  (setvar "LUNITS" |LUNITS|)
  (setvar "LUPREC" |LUPREC|)
  (setvar "FILLETRAD" |FILLETRAD|)
  (setvar "LTSCALE" |LTSCALE|)
  (setvar "AUTOSNAP" |AUTOSNAP|)
  (setvar "TEXTSTYLE" |TEXTSTYLE|)
  (command "dimstyle" "r" |DIMSTYLE|)
  (prompt
    (strcat "\nCurrent layer is now " (getvar "CLAYER") ".")
  )
  (prompt "\nRunning object snaps are now back to normal.")
  (prompt
    "\nPolar and object snap tracking are turned back on."
  )
  (princ)
)
```

A Better Way

- Use a ***LIST*** to store the name of each variable to be changed

- Use the (mapcar) function to iterate through each variable in the list and store the setting of each variable in a new ***LIST***

The (mapcar) Function

(mapcar function list)

- Takes a function and a list (or more than one list) as arguments
- Iterates through the list performing the function on each atom in the list
- The result of each application of the function to the atoms is returned in a ***list***
- I don’t really know what iterate means, but it makes sense

Diagramming the (mapcar) function – Part 1

- Example:

(setq FIRSTNAMES (list “Jimi” “Eric” “Duane”))

Does this To this Then this Then this

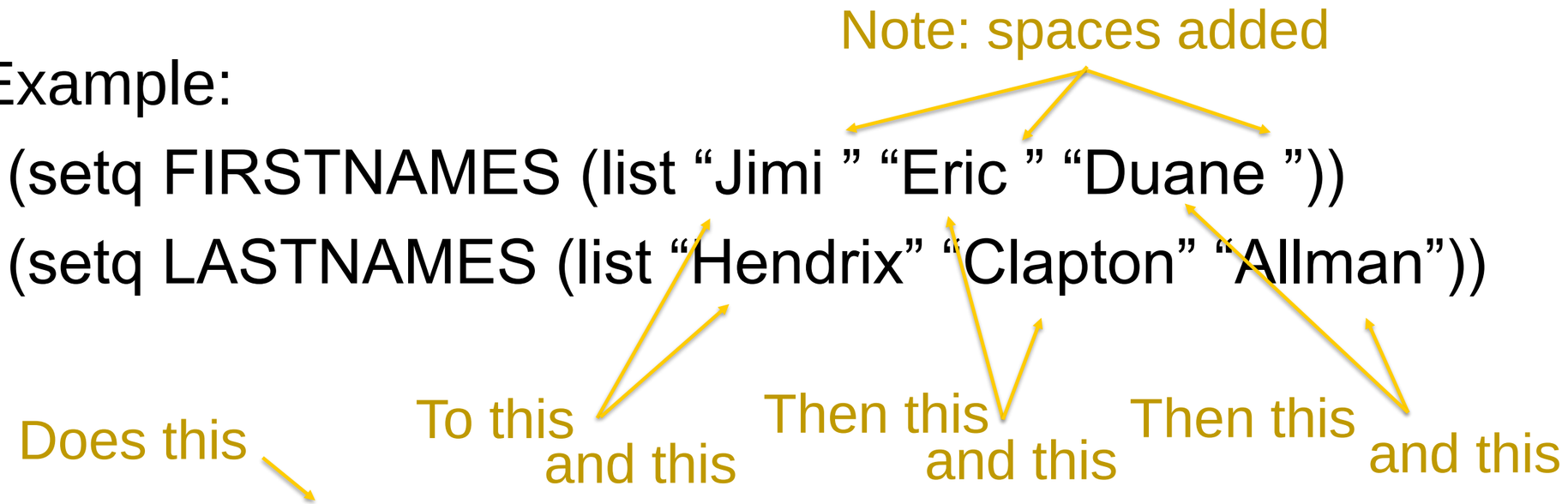
(mapcar ‘prompt FIRSTNAMES)

Returns:

Jimi Eric Duane (nil nil nil)

Diagramming the (mapcar) function – Part 2

- Example:




```
(setq FIRSTNAMES (list “Jimi ” “Eric ” “Duane ”))
(setq LASTNAMES (list “Hendrix” “Clapton” “Allman”))

(mapcar ‘strcat FIRSTNAMES LASTNAMES)
```

Returns:

(“Jimi Hendrix” “Eric Clapton” “Duane Allman”)

Applying (mapcar) to Our Program

3

1

2

```
(defun C:LISTSTORE ( / VARLIST VARVALS PT1 PT2 PT3)
  (setq VARLIST (list "CMDECHO"     "CLAYER"      "OSMODE"      "PICKBOX"
                      "AUNITS"      "AUPREC"      "LUNITS"      "LUPREC"
                      "FILLETRAD"   "LTSCALE"     "POLARANG"    "AUTOSNAP"
                      "TEXTSTYLE"
                    )
  )
  (setq VARVALS (mapcar 'getvar VARLIST))

  (setvar "CLAYER" "HIDDEN")
  (prompt
    (strcat "\nCurrent layer was changed to "
            (getvar "CLAYER")
            "."
    )
  )
  (setvar "OSMODE" 0)
  (prompt "\nRunning object snaps were turned off.")
  (setvar "AUTOSNAP" 0)
  (prompt "\nPolar and object snap tracking were turned off.")
  (setq PT1 (getpoint "\nPick a first point: "))
  (setq PT2 (getpoint "\nPick a second point: "))
  (setq PT3 (getpoint "\nPick a third point: "))
  (command "line" PT1 PT2 PT3 "Close")
  (mapcar 'setvar VARLIST VARVALS)
  (prompt
    (strcat "\nCurrent layer is now " (getvar "CLAYER") ".")
  )
  (prompt "\nRunning object snaps are now back to normal.")
  (prompt
    "\nPolar and object snap tracking are turned back on."
  )
  (princ)
)
```

Exercise No. 2

- If not already open, open the Test Settings.dwg file
- If not already open, open the Visual LISP Editor
- Start a new program file
- Type the above lines of code shown in Figure 3 (in the interest of saving time just put “CLAYER”, “OSMODE”, and “AUTOSNAP” in the variables list to be saved)
- Save the file as LISTSTORE.lsp
- Press the “Load active edit window” button
- Switch to AutoCAD
- Set any layer other than “Hidden” as the current layer
- Turn polar tracking and object snap tracking on
- Type LISTSTORE into the Command: prompt
- As it is asking you to “Pick the first point”, note that the hidden layer is current and polar tracking and object snap tracking is now off
- Finish following the prompts of the program
- Note that the previous layer is now current again, and polar tracking and object snap tracking are now on again.

```
(defun C:LISTSTORE (/ VARLIST VARVALS PT1 PT2 PT3)
  (setq VARLIST (list "CMDECHO"     "CLAYER"      "OSMODE"      "PICKBOX"
                      "AUNITS"      "AUPREC"      "LUNITS"      "LUPREC"
                      "FILLETRAD"   "LTSCALE"     "POLARANG"    "AUTOSNAP"
                      "TEXTSTYLE"
                    )
  )
  (setq VARVALS (mapcar 'getvar VARLIST))

  (setvar "CLAYER" "HIDDEN")
  (prompt
    (strcat "\nCurrent layer was changed to "
            (getvar "CLAYER")
            "."
    )
  )
  (setvar "OSMODE" 0)
  (prompt "\nRunning object snaps were turned off.")
  (setvar "AUTOSNAP" 0)
  (prompt "\nPolar and object snap tracking were turned off.")
  (setq PT1 (getpoint "\nPick a first point: "))
  (setq PT2 (getpoint "\nPick a second point: "))
  (setq PT3 (getpoint "\nPick a third point: "))
  (command "line" PT1 PT2 PT3 "Close")
  (mapcar 'setvar VARLIST VARVALS)
  (prompt
    (strcat "\nCurrent layer is now " (getvar "CLAYER") ".")
  )
  (prompt "\nRunning object snaps are now back to normal.")
  (prompt
    "\nPolar and object snap tracking are turned back on."
  )
  (princ)
)
```

I am a Packers Fan

Learning Objective:
Create Your Own Error Routines

Before We Do Our Own Thing

When you need to get data of a certain type from a user, use the appropriate function:

- (getint) Returns an integer. Returns nil if user hits <enter>.
- (getreal) Returns a real. Returns nil if user hits <enter>.
- (getdist) Returns a real. Returns nil if user hits <enter>.
- (getstring) Returns a string. Returns an empty string if user hits <enter>.
- (getpoint) Returns a point list. Returns nil if user hits <enter>.
- (entsel) Returns a list of an ename and a point list. (Beyond the scope of this course)
- (ssget) Returns a pickset. (Beyond the scope of this course)

Notice that (getint), (getreal), (getdist), and (getpoint) can all return nil if the user hits <enter>

Their Built-in Error Traps

- Play around with (getint), (getreal), and (getpoint)
- Note that you can only enter valid data types

```
Command: (setq X (getint "\nEnter an integer: "))
Enter an integer: 1.2
Requires an integer value.
Enter an integer:
```

- But! The <enter> key is a problem
- Hitting the <enter> key returns nil – crashes programs that are expecting X to be holding an integer

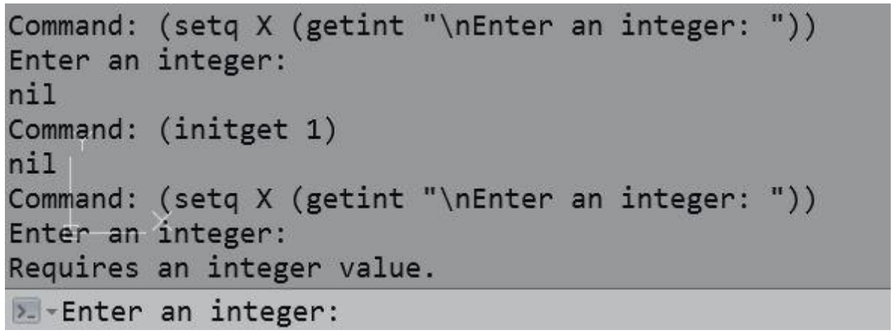
Trapping the <enter> key

- Calling the (initget) function, and passing it a bit code argument of 1, disallows hitting <enter>
  - (initget 1) disallows null (the <enter> key) input
  - (initget 2) disallows 0 as input
  - (initget 4) disallows negative values to be input
  - (initget 7) disallows all of the above

  (There are other bit codes that can be used, but they are beyond the scope of this course)

Applying the (initget) function

- Examine the following code:

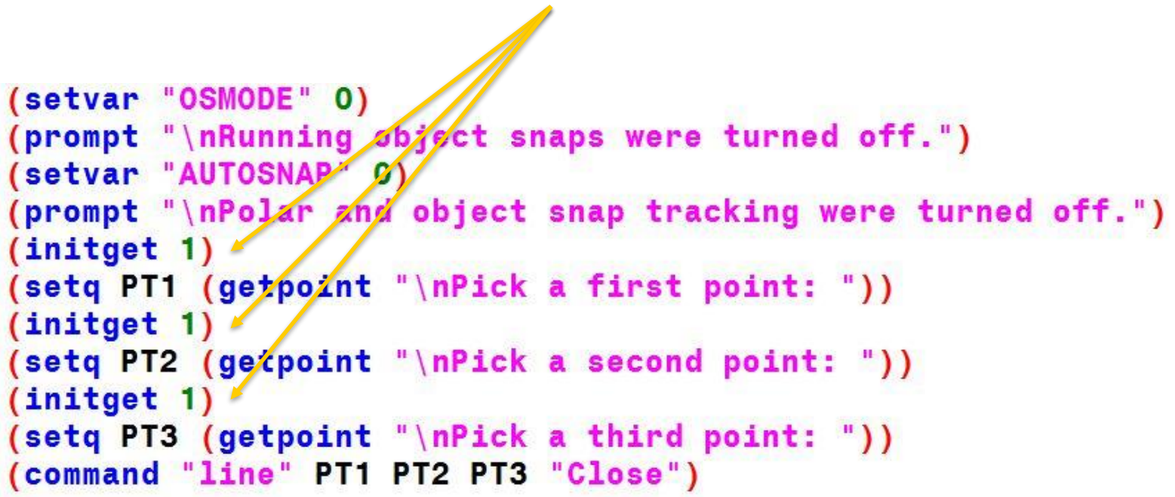


- The (initget) function:
  - should almost always be used before calling the (getxxx) functions
  - applies to the next (getxxx) function only
  - does not work with the (getstring) function, as the (getstring) function does not return nil when <enter> is typed.

Exercise No. 3

- Activate the Visual LISP editor
- If not already open, open the LISTSTORE.lsp file
- Add (initget 1) on the line previous to each of the (getpoint) expressions as shown in Figure 6.
- Save the file
- Press the “Load active edit window” button
- Switch to AutoCAD
- Run the command – trying hitting the <enter> key when prompted to pick a point.

Add these lines



Green Bay Packers Super Bowl Trophies – in the Packers Hall of Fame… which I am in.

AutoLISP’s *error* Function

Examine the follow sequence:

```
(setq X 1)
1

(prompt X)
; error: bad argument type: stringp 1

(defun NEWERROR (msg) (prompt "\nSomething went wrong..."))
NEWERROR

(setq OLDERROR *error*)
#<SUBR @00000000337fe8e0 *ERROR*>

(setq *error* NEWERROR)
#<SUBR @000000003b1c9390 NEWERROR>

(prompt X)
Something went wrong...

(setq *error* OLDERROR)
#<SUBR @00000000337fe8e0 *ERROR*>

(prompt X)
; error: bad argument type: stringp 1
```

Making an *error* Function that Works for Us

- The previous slide shows that the *error* routine is customizeable – we need to make it useful!
- As programmers, we try to account for all possible errors – and create code to handle them
- We just can’t handle everything – when things go South, we have our FAVSET routine to get things back to normal
- Instead, we will incorporate the variables restoring code into our *error* routine

Our Own *error* Function – Continued…

An overview of how we are going to do that:

1. Define a NEWERROR function that will reset the variables (per our previously written code), and let the user know that the variables have been reset, and set the error routine back to the built in error routine.
2. Within the main program the built-in error routine will be stored, and the error routine will be set to the newly defined version.
3. At the end of the program, just before completion, the old error routine is restored.

Exercise No. 4

1. Activate the Visual LISP editor
2. If not already open, open the LISTSTORE.lsp file
3. Add the code to define the NEWERROR routine as shown in Figure 7
4. Add the two lines after the (defun c:LISTSTORE… line to redefine and store the error routine
5. Add the line near the end of the program, on the line just before the (princ) to reset the error routine
6. Save the file
7. Press the “Load active edit window” button
8. Switch to AutoCAD
9. To be on the safe side, run the FAVSET command to set everything back to “normal”.
10. Run the LISTSTORE command
11. Before picking the first point, note the layer has changed, and that polar tracking, object snap tracking, and running object snaps have been turned off.
12. Hit the <esc> key when prompted to pick a point.
(This will crash the program, and would normally leave our variables all messed up – but they have been set back to “normal”!)

Step 3

```
(defun NEWERROR (MSG)
  (mapcar 'setvar VARLIST VARVALS)
  (prompt "\nVariables reset to previous values...")
  (setq *ERROR* OLDERROR)
  (princ)
)

(defun C:LISTSTORE (/ VARLIST VARVALS PT1 PT2 PT3)
  (setq OLDERROR *ERROR*)
  (setq *ERROR* NEWERROR)
  (setq VARLIST (list "CMDECHO"     "CLAYER"     "OSMODE"     "PICKBOX"
                      "AUNITS"      "AUPREC"     "LUNITS"     "LUPREC"
                      "FILLETRAD"   "LTSCALE"    "POLARANG"   "AUTOSNAP"
                      "TEXTSTYLE"
                    )
  )
  (setq VARVALS (mapcar 'getvar VARLIST))

  (setvar "CLAYER" "HIDDEN")
  (prompt
    (strcat "\nCurrent layer was changed to "
            (getvar "CLAYER")
            "."
    )
  )
  (setvar "OSMODE" 0)
  (prompt "\nRunning object snaps were turned off.")
  (setvar "AUTOSNAP" 0)
  (prompt "\nPolar and object snap tracking were turned off.")
  (initget 1)
  (setq PT1 (getpoint "\nPick a first point: "))
  (initget 1)
  (setq PT2 (getpoint "\nPick a second point: "))
  (initget 1)
  (setq PT3 (getpoint "\nPick a third point: "))
  (command "line" PT1 PT2 PT3 "Close")

  (mapcar 'setvar VARLIST VARVALS)
  (prompt
    (strcat "\nCurrent layer is now " (getvar "CLAYER") ".")
  )
  (prompt "\nRunning object snaps are now back to normal.")
  (prompt
    "\nPolar and object snap tracking are turned back on."
  )
  (setq *ERROR* OLDERROR)
  (princ)
)
```

Step 4

Step 5

First Teaching Job – (Sorry this has nothing to do with the Packers Hall of Fame…)

VOLUME 2—NO. 11 Friday, December 24, 1971

Published Every Third Saturday

Ashwaubenon Press

Newsstand price .15

Front Row, L to R, Becky Jewell, Cheryl Cooney, Bonnie Retza, Crosby, Mari Paulus, Sandra Smith, Mark Bell, Dale Craig Black, Gary Ruge, Brian King; second row, L to R, Lisa Chelebowski, Kyle Kreuter, Mark Menne, Randy Brylski; third row, L to R, Janice Rentmeester, Tracy Dorn, Nancy Wierzba and Roxanne Greatens.

Valley View Sixth Graders

Learning Objective:
Understand More of the Subtle Nuances of How AutoLISP Works

It All Boils Down to Logic

“Programming” is nothing more than simplifying a task or tasks by automating or reducing the steps needed to produce the desired outcome by intelligently using or reusing program supplied or user supplied data.

Data – We Cannot Program Without It!

AutoLISP is a very logical language. We use functions to obtain results. We have to pass data to functions. The data needs to be passed in a logical order. We get data from users, and that data must be obtained logically. Functions then return data in a very logical format.

Data Types – There Really Are Not Too Many!

- Integers – whole numbers, no decimal point, negative numbers preceded with a - sign
- Reals – includes a decimal point, decimal place must be “covered” (0.25), negative numbers preceded with –
- Strings – text values, always enclosed in quotation marks
- Lists – multiple (or individual) data types enclosed in parenthesis
- Enames – AutoCAD “objects”
- Picksets – a group of AutoCAD “objects”

Data Related Error Messages

- Again, functions typically require arguments of specific data types, and they return values of specific data types.
- Watch the instructor demo what is meant by various returned error messages, including:

```
Command: (setq X "Great Class!")
"Great Class!"
Command: (repeat X (+ 1 1))
; error: bad argument type: fixnump: "Great Class!"
Command: (+ 1 X)
; error: bad argument type: numberp: "Great Class!"
Command: (setq Y 2)
2
Command: (strcat X Y)
; error: bad argument type: stringp 2
Command: (getpoint X "\nPick a point: ")
; error: bad argument type: point: "\nPick a point: "
```

How did I do?

- Your class feedback is critical. Fill out a **class survey** now.
- Remember! That green check is a noth too low!
- Use the AU mobile app or fill out a class survey online.
- Give feedback after each session.
- AU speakers will get feedback in real-time.
- **Your feedback results in better classes and a better AU experience.**