

Steven Wells Programming Samples

The material below contains sample functions that I had written in the AutoLISP language, which is an AutoCAD-specific dialect of common LISP. All material in this sample fits within 80 columns (characters) per line of text (or code). LISP treats all characters after a semicolon as a comment. If space permits, I start comments at column 54 in a line, but this is only my personal standard. Many casual LISP programmers include few or no comments within their code. I also use an unusual format in that I place all available closing parentheses aligned within a single line, rather than each on a separate line. This makes the code more compact and easier to debug because of the shorter distances required for visual alignment of code elements.

LISP stands for LISt Processing, or the jocular Lost In Stupid Parentheses. It finds use in artificial intelligence and is especially suitable for the freeform input environment of CAD design and editing. LISP is extremely powerful for handling lists, and incredibly terse; that is, a lot of functionality can be crammed into very little code. The following function demonstrates an example of LISP minimalism.

I designed this function to use a calculated pen weight to select from a list of the standard AutoCAD pens used for plotting onto large architectural sheets. I write my routines so as to be applicable for as general use as possible, so the list can contain integers and/or reals supplied in any order, not just ascending or descending.

Functional design: When the specified number is:

A member of the list, it is returned.

Less than the smallest member of the list, the smallest member is returned.

Greater than the largest member of the list, the largest member is returned.

Not a member of the list but between list values, the larger of the two closest surrounding values is returned.

This version of my function uses shortened variable names, contains only built-in functions, and is trimmed of white space that only helps human readability. The entire function contains only 75 characters:

```
(defun i(m n)(apply 'min(mapcar '(lambda(x)(if(<= m x)x(apply 'max n))))n))
```

```
;-----  
My actual production version containing 94 characters of code follows:
```

```
=====
```

```
; #inlist.lsp                Written by Steven Wells        Update: 2005 June 8
```

```
; For a given number, select the smallest corresponding number within a list.
```

```
; This function takes two arguments: a number 'n' as a potential selection and  
; a list, in any order, of selectable numbers. The function returns n if it is  
; a member of the list. Otherwise it returns the smallest member of the list  
; that is greater than n. If n is greater than any member of the list, the  
; function returns the list's largest member.
```

```
; Required defined functions: [none]
```

```
; Usage examples: (#inlist 3.25 nums!) [or] (#inlist 1 '(2 8 6 0 4))  
;                [or] (#inlist 1.5 '(-1.0 -0.5 0.0 0.5 1.0 1.5 2.0 3.0 5.0))
```

```
=====
```

```
(defun #inlist (m n!)  
  (apply 'min (mapcar '(lambda (x) (if (<= m x) x (apply 'max n!))) n!))  
)
```

I wrote the following program (and its auxiliary functions) to enable a coworker to align and resize oversized pseudorandomly oriented text within large maps that contained hundreds of small, irregularly shaped parcels. Typically, each parcel contained five or six lines of text, each on different layers, to denote APN (Assessor Parcel Number), owner, description, size, assessment date, and the like. The text was oriented at various angles to accommodate the parcel shape, and non-uniformly sized and oriented because persons who originally created the maps were sloppy in their work. My coworker had intended to reposition and resize each text line individually to insure legibility and a professional appearance, and estimated that it would take her 15 days of grueling effort. She used my program to complete the task in only two days.

```
; RespaceText.lsp          Written by Steven Wells      Update: 2004 January 12
;                          SCVWD - Santa Clara Valley Water District
```

```
; Resize and respace text entities that have common justification.
```

```
=====
; This command function uniformly resizes the text height of all selected text
; entities and respaces them to a new user specified line spacing starting with
; the text below the topmost line of text. All the selected text must have the
; same justification. If justifications vary, or is either ALIGN or FIT
; justified, none of the entities are modified, and a message refers the user to
; the JUSTIFYTEXT command. Resetting justification is beyond the scope of this
; function, and easily accomplished as suggested or with the SCALETEXT command.

; The function checks for the existence of either of a pair of global variables
; used to specify the new text height and line-to-line spacing. If a variable
; does not exist, the user is prompted for the appropriate size, which must be a
; positive number. The user is prompted to select text objects. Non-TEXT objects
; are ignored. If the user presses Enter without selecting any text, the
; function offers the option to change the height, spacing, or both. A continue
; option, the default, continues object selection.

; The selected identical-justification text entities are sorted for processing
; in order based on each entity's alignment point y-coordinate. The topmost
; (largest y-coordinate) entity is resized to the desired height, and its
; rotation angle and alignment point are used as references for the remaining
; text entities, if any. Each in turn is resized, reoriented, and repositioned
; by its alignment point to a point below the topmost line of text. The new
; alignment points are angled 90 degrees down from the top line orientation,
; spaced by their order multiple of the specified line-to-line spacing.

; As this is a command function, the MULTIPLE command can aid in heavy reuse.
; At the command line, enter MULTIPLE and RESPACETEXT in upper, lower, or mixed
; case. Once the height and spacing values are set, the user can repeatedly
; specify a leftward crossing selection that includes the desired text objects,
; and press Enter. In the event of an accidental null selection, pressing Enter
; recovers via the Continue option.

; Required defined functions: fore aftr error var get mod
; Required functions included here: getd esort sstlist takout

; Global variables: #ht #sp

; Usage examples: RespaceText [or] Multiple RespaceText
=====
```

```

(defun C:RESPACETEXT (/ ss kw ob ent! code n mbr pt ang n en)
  (fore) ; Preparation handler
  (if (not #ht) (setq #ht (getd "New text height"))) ; Set height if not set
  (if (not #sp) (setq #sp (getd "Line-to-line spacing"))); Set spacing if not set
  (princ (strcat "\nSelection filtered for TEXT objects only. "
    "Press Enter to change settings." ; Message
  )
  ) ; Select text objects
  (while (not (setq ss (ssget '((0 . "TEXT"))))) ; until found
    (initget "Height Spacing Both Continue") ; Prompt for keyword
    (setq kw (getkword "\n[Height/Spacing/Both/Continue] <Continue>: "))
    (cond
      ((= kw "Height") (setq #ht (getd "New text height"))) ; Set height
      ((= kw "Spacing") (setq #sp (getd "Line-to-line spacing"))); Set spacing
      ((= kw "Both") (setq #ht (getd "New text height")); Set height &
        (setq #sp (getd "Line-to-line spacing")); Set spacing
      )
    )
    (T nil) ; Continue
  )
  )
  (setq n (sslength ss) ; Text object count
    ob (ssname ss 0) ; of one object
    ss (ssget "P" (list (cons 72 (get 72 (entget ob))) ; Find if all
      (cons 73 (get 73 (entget ob))) ; have same
    )
  )
  ) ; justification
  (if (> n (sslength ss)) ; Not all same justification
    (progn
      (princ (strcat "\nNon-uniformly justified. " ; Message with suggestion
        "Use JUSTIFYTEXT to set uniform justification."
      )
      )
      (exit) ; Exit
    )
  )
  (setq code (get 72 (entget ob))) ; Horizontal justification
  (if (or (= code 3) (= code 5)) ; Aligned or fit text
    (progn ; disallowed
      (princ (strcat "\nAligned or fit text not allowed. " ; Message
        "Use JUSTIFYTEXT to change justification."
      )
      )
      (exit) ; Exit
    )
  )
  (setq ent! (sslist ss)) ; Objects as entity list
  (if (equal (get 11 (entget (car ent!))) '(0.0 0.0 0.0)); Left baseline text?
    (setq code 10) ; use insert point, else
    (setq code 11) ; use alignment point
  )
  (setq ent! (esort ent! code nil nil) ; Sort by y-coordinate
    n mbr (length ent!) ; Number of text entities
    en (car ent!) ; Top text's entity name
    pt (get code (entget en)) ; Top alignment point
    ang (get 50 (entget en)) ; Top text rotation angle
    n 1 ; Index start at 1
  )
  (mod en 40 #ht) ; Resize top line of text
  (if (> n mbr 1) ; Respace if more than 1
    (progn
      (repeat (1- n mbr) ; For each after 1st
        (setq en (nth n ent!)) ; Next line's entity
        (mod en 40 #ht) ; Resize line of text
        (mod en 50 ang) ; Reorient text
        (mod en code) ; Reposition text
        (polar pt (- ang (/ pi 2)) (* n #sp)) ; angled below top text
      )
      (setq n (1+ n)) ; n spacings down
    )
    (setq n (1+ n)) ; Increment counter
  )
  ) ; Message for n objects

```

```

    (princ (strcat "\n" (itoa nمبر) " text objects scaled and respaced.))
  )
  (princ "\n 1 text object scaled.") ; Message for 1 object
)
(aftr) ; Exit from error handler
)
;=====
; getd - Get a positive distance with a prompt.

; This function takes a text string as the argument and returns a user
; specified distance. Null, zero, or negative entries are disallowed.
; The prompt is formatted as a new line, with a trailing colon and space.

(defun getd (tx)
  (initget (+ 1 2 4)) ; No null, zero, negative
  (getdist (strcat "\n" tx ": ")) ; Get the value
)
;-----
; esort - Sort a list of entities based on an orthogonal direction.

; This function takes a list of entities, an entity point code, an orthogonal
; orientation, and an ascending order direction, and returns a spatially sorted
; list. A non-nil orientation argument specifies horizontal, and nil specifies
; vertical. A non-nil direction argument specifies ascending, and nil specifies
; descending.
; The returned order is such that each progressing entity has a primary entity
; point further along the specified direction. An auxiliary list of coordinates
; is created to map the positions of the entities along the specified direction.

(defun esort (ls! cd hz dr / l! val len n obj srt)
  (if hz ; For horizontal direction
    (foreach item ls! ; Build a list of X coords
      (setq l! (append l! (list (car (get cd (entget item)))))))
    ) ; For vertical direction
    (foreach item ls! ; Build a list of Y coords
      (setq l! (append l! (list (cadr (get cd (entget item)))))))
    )
  (repeat (length l!) ; Repeat for list length
    (setq val (if dr (apply 'min l!) (apply 'max l!)); Obtain min/max value
      len (length l!) ; Current length of list
      n (- len (length (member val l!))) ; Nth position in list
      obj (nth n ls!) ; Next entity to use
      l! (takout val l!) ; Remove coordinate &
      ls! (takout obj ls!) ; entity from lists
      srt (append srt (list obj)) ; New sorted list
    )
  ) ); end defun repeat setq
;-----
; ssslist - Return a list of entity names in a selection set.

; This function takes a selection set as the argument and returns a list of the
; entity names of the entities in the selection set. The function returns nil if
; the argument set is nil or zero length.

(defun ssslist (ss / n l!)
  (if ss ; If selection set exists
    (progn
      (setq n -1) ; Initialize counter
      (repeat (sslength ss) ; Repeat for each entity
        (setq n (1+ n) ; Increment counter
          l! (append l! (list (ssname ss n))) ; Entity name onto list
        )
      )
    )
  ) ) )
;-----

```

```

; takout - Take a member of a list out of the list.

; This function takes an item as a potential member of a list and the list as
; arguments. If the item is indeed a member of the list, the function returns
; the list with the first instance of that item removed. If the item is not a
; member of the list, the list is returned unmodified.

(defun takout (item l! / retn! sub)
  (if (setq sub (member item l!)) ; Is item a member of list?
      (progn
        (repeat (- (length l!) (length sub)) ; For items ahead of item
          (setq retn! (append retn! (list (car l!))) ; Append to a list
                l! (cdr l!)) ; Trim off first item
          ) ); end repeat setq
        (setq retn! (append retn! (cdr sub))) ; Take item from sublist
      ) ; end progn ; append to pre-item list
      l! ; Not member? Return source
  ) ); end defun if
;=====
(princ) ; Load quietly

```

```

; RW-2003.mnl          Written by Steven Wells      Update: 2004 January 12
;                    SCVWD - Santa Clara Valley Water District

; This loads AutoLISP files used by the RW-2003 menu file.

;[This is an excerpt of the RW-2003 menu file. It contains functions used by the
; RespaceText.lsp program.]
;=====
; Control Functions - Handle setvars, AutoLISP errors, Cancels, programmed exits
;-----
; error - General error handler.

; This function is a general error handler. It is invoked automatically for
; error conditions, programmed exits, or user issued Esc cancellations.
; It takes one parameter, a text string specifying the error. If the exit was
; pre-programmed or the user issued a cancel, the function exits quietly,
; otherwise the error string is displayed in an alert box. The aftr function ends
; the Undo group, restores the previous error handler and restores any saved set
; variables. The ended Undo group is undone.

; Required defined functions: aftr

(defun-q error (s / ce)
  (command) (command) (command)          ; Cancel command 3 times
  (if (or (= s "quit / exit abort") (= s "Function cancelled")); If programmed
    (princ)                               ; exit or Esc, exit quietly
    (alert s)                             ; Display the error message
  )
  (aftr)                                  ; End UNDO group, restore
  (setq ce (getvar "CMDECHO"))            ; error handler & setvars
  (setvar "CMDECHO" 0)                   ; Save command echo
  (command ".UNDO" 1)                    ; Quiet for undo command
  (setvar "CMDECHO" ce)                  ; UNDO the group
  (princ)                                ; Restore command echo
  )
  )
;-----
; var - Save and configure set variables.

; This function sets one or more set variables in a list, saving the previous
; values in a list for later restoration.

; Required defined functions: [none]

; Usage examples: (var '(("CMDECHO" . 0)))
;                [or] (var '(("GRIPS" . 1) ("OSMODE" . 7)))

(defun var (vr / vname new old)
  (foreach item vr                          ; For each item in the list
    (setq vname (car item) old (getvar vname) new (cdr item); Get variable name
      #var! (cons (cons vname old) #var!)    ; & value, place old value
    )
    (setvar vname new)                      ; onto list
  )
  )
;-----

```

```

; fore - Establish an error handler.

; This function restores the graphics screen, sets cmdecho into the saved list
; of set variables, and sets the error handler. It saves the previous handler
; to the global variable #err.

; Required defined functions: error

; Usage example: (fore)

(defun fore ()
  (graphscr) ; Set to graphics screen
  (var '("CMDECHO" . 0)) ; Set command echo off
  (setq #err *error* *error* error) ; Establish error handler
  (command ".UNDO" "GROUP") ; Begin UNDO group
)
;-----
; aftr - Restores from the error handler, retaining existing error control.

; This function ends the UNDO group, and restores the saved set variables.
; However, it does NOT restore the previous error handler.

; Required defined functions: [none]

; Usage example: (aftr)

(defun aftr ()
  (command ".UNDO" "END") ; End UNDO group
  (foreach item #var! (setvar (car item) (cdr item))); Restore setvars from list
  (setq #var! nil) ; Clear setvar list
  (princ) ; Exit quietly
)
;=====
; Entity Functions
;-----
; get - Get an association pair data element.

; This function takes an integer data code and an entity data list, and returns
; the corresponding data element of the association pair.

; Required defined functions: [none]

; Usage example: (setq layr (get 8 edata))

(defun get (n lst) (cdr (assoc n lst))) ; Return entity code's data
;-----
; mod - Modify the association pair data for an entity.

; This function takes an entity name, an association pair group code, and the
; new entity data value as arguments, and returns nil. The new value replaces
; the association pair data in the entity, and updates the entity.

; Required defined functions: [none]

; Usage example: (mod ename 8 layr)

(defun mod (ename code item / edata)
  (setq edata (entget ename)) ; Get the entity data
  (entmod (subst (cons code item) (assoc code edata) edata)); Update the entity
  ) ; with substituted data
;=====
(princ) ; Load quietly

```

