AI Assignment: Practice with the Basics, Evaluators, and Property Lists

What's it all about?

The first bit of this challenge asks you to do some modest Lisp interactions. Beyond that, this challenge is intended to provide you with an opportunity to write and demo five simple Lisp function definitions. If you carefully consider the example Lisp code that has been provided, you should have no trouble piecing together the required definitions. As always, you will be asked to post your work to your AI Work Site in a reasonable way.

Why do it?

Upon successful completion of this programming challenge it is anticipated that you will be better able to:

1. Perform some simple manipulations on lists.
2. Write some simple function definitions in Lisp.
3. Demonstrate the ability to properly scope variables in Lisp.
4. Perform simple numeric computations in Lisp.
5. Make effective use of Lisp evaluators.
6. Represent knowledge effectively using property lists.

Task 0: Establish a Solution Document for this Assignment

Quickly read through the remaining tasks so that you will be in a good position to create a structurally sound stub of a solution document. Then, create it. Simply reserve spots in your solution document to place the work that you do for Tasks 1 through 6.

Task 1: Simple Things

Simply do the following:

1. PRELIMINARY TASK: Start a Lisp process.

2. DISTANCE-BETWEEN-TWO-POINTS: Bind symbols x1, y1, x2, and y2 to real numbers. Interpret the (x1,y1) as one point in 2-space and (x2,y2) as another point in 2-space. Write a sequence of one or more forms to compute the distance between these two points.

3. REFERENCING X: Without using any variables in any way, write compositions of `CAR` and `CDR` (in either "long hand" or "short hand"), write a form to reference just X in each of the following lists:

   (a) (A B X C D)
   (b) (A B (X) C D)
   (c) ((A (B X C D)))

4. A FEW META-FORMS: Please **bind the symbol `colors` to a list of seven colors.** Then, type each of the following forms and observe the results:

   (a) (quote colors)
   (b) 'colors
   (c) colors
   (d) (describe 'colors)
   (e) (describe colors)
   (f) (type-of 'colors)
   (g) (type-of colors)
   (h) (typep colors 'cons)
   (i) (typep colors 'list)
   (j) (typep colors 'symbol)
   (k) (typep 'colors 'cons)
   (l) (typep 'colors 'symbol)

5. B5 DECK: Five times, establish a binding, and then verify it. In doing so you are going to build the four suits and a complete 20-card deck for a card game called "B5". Specifically:

   (a) Establish the binding: `CLUBS` $\longrightarrow$ `( (10 C) (J C) (Q C) (K C) (A C) )`
   (b) Ask Lisp to evaluate the following form, by simply typing it at the prompt: `CLUBS`
   (c) Establish the binding: `DIAMONDS` $\longrightarrow$ `( (10 D) (J D) (Q D) (K D) (A D) )`
   (d) Ask Lisp to evaluate the following form, by simply typing it at the prompt: `DIAMONDS`
   (e) Establish the binding: `HEARTS` $\longrightarrow$ `( (10 H) (J H) (Q H) (K H) (A H) )`
   (f) Ask Lisp to evaluate the following form, by simply typing it at the prompt: `HEARTS`
   (g) Establish the binding: `SPADES` $\longrightarrow$ `( (10 S) (J S) (Q S) (K S) (A S) )`
   (h) Ask Lisp to evaluate the following form, by simply typing it at the prompt: `DIAMONDS`
   (i) Bind `DECK` to a list of all 20 cards by making good use of the `APPEND` function and the four suit bindings that have been established.
   (j) Ask Lisp to evaluate the following form, by simply typing it at the prompt: `DECK`
   (k) Bind `PARTITIONED-DECK` to the list of lists obtained by replacing `APPEND` with `LIST` in the form that you wrote for the previous item.

(l) Ask Lisp to evaluate the following form, by simply typing it at the prompt: `PARTITIONED-DECK`

6. Copy the session to your solution document.

Task 2: Distance Between Two Points

Simply do the following:

1. In a file called `perimeter_of_triangle.l` define a function to compute the distance between two points which is consistent with the following specifications:
   - The name of the function will be `distance`.
   - It will receive **four** parameters, called `x1` and `y1` (which collectively constitute point 1) and `x2` and `y2` (which collectively constitute point 2) – in that order, from left to right.
   - The function returns the distance between `(x1,y1)` and `(x2,y2)`.
2. Create a nice demo of the function, by loading the code into a Lisp process, and then applying the `distance` function a number of times.
3. Copy the Lisp code relevant to this task to your solution document. Copy the demo to your solution document.

Task 3: Perimeter of Triangle

Simply do the following:

1. In the file called `perimeter_of_triangle.l` define a function, which makes good use of the `distance` function, to compute the perimeter of a triangle and which is consistent with the following specifications:
   - The name of the function will be `perimeter`.
   - It will receive six parameters called `x1` and `y1` (point 1), and `x2` and `y2` (point 2) and `x3` and `y3` (point 3) – in that order, from left to right.
   - The function returns the perimeter of the triangle defined by the three points.
2. Create a nice demo of the function, by loading the code into a Lisp process, and then applying the `distance` function a number of times.

3. Copy the Lisp code relevant to this task to your solution document. Copy the demo to your solution document.

Task 4: LR Infix Calculator

Simply do the following:

1. In a file called `two_calculators.l` define a function to simulate a calculator that is consistent with the following specifications:
   - The name of the function will be `calculator-LR`.
   - The function takes no parameters and returns no values.
   - The function repeatedly reads and evaluates a parenthesis-free infix expression, from left to right, of the form `operand operator operand operator operand`, and prints the result.
   - The calculator will evaluate the leftmost operator prior to evaluating the rightmost operator, which is what is meant by "from left to right". (Thus, for example, `1 + 2 * 3` would be interpreted as `( ( 1 + 2 ) * 3 )`.)
   - The function makes good use of the `apply` function.
   - The calculator need only handle the four basic arithmetic operators and numeric operands.
2. Demo your program by loading the file into a Lisp process, running the `calculator-LR` function, and evaluating several representative expressions, including `12.3 + 4.5 * 9.6` and `12.3 / 4.5 - 9.6`.
3. Copy the Lisp code relevant to this task to your solution document. Copy the demo to your solution document.

Task 5: RL Infix Calculator

Simply do the following:

1. In the file called `two_calculators.l` define a function to simulate a calculator that is consistent with the following specifications:
   - The name of the function will be `calculator-RL`.
   - The function takes no parameters and returns no values.
   - The function repeatedly reads and evaluates a parenthesis-free infix expression, from right to left, of the form `operand operator operand operator operand`, and prints the result.

- The calculator will evaluate the rightmost operator prior to evaluating the leftmost operator, which is what is meant by "from right to left". (Thus, for example, `1 + 2 * 3` would be interpreted as `( 1 + ( 2 * 3 ))`.)
- The function makes good use of the `funcall` function.
- The calculator need only handle the four basic arithmetic operators and numeric operands.

2. Demo your program by loading the file into a Lisp process, running the `calculator-LR` function, and evaluating several representative expressions, including `12.3 + 4.5 * 9.6` and `12.3 / 4.5 - 9.6`.
3. Copy the Lisp code relevant to this task to your solution document. Copy the demo to your solution document.

Task 6: Representing Quarto Pieces using Property Lists

Please study the Quarto example in the lesson on property lists. After you are comfortable with the representation, please proceed.

What to do ...

1. Enter code into a file called `Quarto.l` that:
   (a) Represents each of the 16 Quarto pieces in terms the appropriately named symbol on which a property list containing four properties is placed.
   (b) Binds the symbol `*pieces*` to a list of the 16 symbols which stand for the 16 quarto pieces.
   (c) Defines a parameterless function called `display-random-piece` which displays a randomly selected Quarto piece in terms of a list containing five items, its symbolic name, its size, its color, its style, and its shape - in that order.
   (d) Is consistent with the given demo.
2. Demo your program by issuing just the forms that I did in the sample demo.
3. Copy the Lisp code relevant to this task to your solution document. Copy the demo to your solution document.

Demo

```
bash-3.2$ clisp
...

[1]> ( load "Quarto.l" )
;; Loading file Quarto.l ...
;; Loaded file Quarto.l
T
[2]> *pieces*
(BRHS BRHC BRSS BRSC BBHS BBHC BBSS BBSC SRHS SRHC SRSS SRSC SBHS SBHC SBSS SBSC)
[3]> ( symbol-plist 'brsc )
(SIZE BIG COLOR RED STYLE SOLID SHAPE CIRCLE)
[4]> ( symbol-plist 'sbhs )
(SIZE SMALL COLOR BLUE STYLE HOLLOW SHAPE SQUARE)
[5]> ( display-random-piece )
(SRHC SMALL RED HOLLOW CIRCLE)
NIL
[6]> ( display-random-piece )
(BRSS BIG RED SOLID SQUARE)
NIL
[7]> ( display-random-piece )
(SBSS SMALL BLUE SOLID SQUARE)
NIL
[8]> ( display-random-piece )
(BRSS BIG RED SOLID SQUARE)
NIL
[9]> ( display-random-piece )
(SBHC SMALL BLUE HOLLOW CIRCLE)
NIL
[10]> ( display-random-piece )
(BRSS BIG RED SOLID SQUARE)
NIL
[11]> (bye)
Bye.
bash-3.2$
```

Task 7: Web work site

Make "an entry" for this programming challenge on your web work site.

Due Date

Friday, September 16, 2022