Week 01 - Lecture 3 Slides

Dr. Yeganeh Bahoo

Created: 2023-09-10 Sun 21:24

Table of Contents

Lecture 3: crash course on Lisp (cont.)

Learning objectives:

By the end of this lecture you should be able to:

- Write lisp programs that involve commonly used forms

The CASE special form

See documentation at hyperSpec.

Write a function that takes in an final score n, $0 \leq n \leq 100$, and returns the respective grade number:

```
(defun convert-to-letter-grade (numeric-grade)
  (case (floor numeric-grade 10)
    (10 "A")
    (9 "A")
    (8 "B")
    (7 "C")
    (6 "D")
    (otherwise "F")))
```

Another example.

```
(defun mypair (x y msg)
  (case msg
    (sum (+ x y))
    (diff (abs (- x y)))
    (dist (sqrt (+ (* x x) (* y y))))
    (print (pprint x) (pprint y))))
```

```
CL-USER> (mypair 3 4 'sum)
7
CL-USER> (mypair 4 3 'diff)
1
```

Special forms (cont.): defining global variables

We will introduce the notion of global variables through a game.

The GUESS-MY-NUMBER game.

You think of a number, say 21, then you ask lisp to guess it.

```
CL-USER> (guess-my-number)  ; You ask lisp
50                          ; lisp guesses a number
CL-USER> (smaller)          ; You tell lisp the number is smaller
25                          ; Lisp guesses another number
CL-USER> (smaller)          ; again you tell lisp the number is smaller
12
CL-USER> (bigger)           ; The guessed number is too small
21                          ; Lisp guesses the right number
```

To create this game, we will write these three functions using the following strategy:

1. Define the upper and lower bounds for the player's number
2. Guess a number between these two numbers
3. If the player says the number is smaller, lower the upper bound.
4. If the player says the number is bigger, raise the lower bound.

This search strategy is called binary search.

The Guess-My-Number Game

We will define and use global variables to store the lower and upper bound numbers

```
CL-USER> (defvar *lowerb* 1)    ; Notice the "earmuffs" in the global variable name
*LOWERB*                        ; This is just Lisp programming etiquette; makes the variable name
CL-USER> (defvar *upperb* 100)  ; really standout for a human reader
*UPPERB*
CL-USER> *lowerb*
1
CL-USER> *upperb*
100
```

To define GUESS-MY-NUMBER, we will use the ASH function to half a number by shiftting its bits one position to the right. This one bit shift operation is executed much faster than a division by two using the '/' operator.

```
CL-USER> (ash 5 -1)
2
CL-USER> (ash 6 -1)
3
CL-USER> (ash 6 1)
12
```

Since we need to change the values of `*lowerb*` and `*upperb*`, let's learn how to do that in lisp.

Modifying the value of a variable

- In programming languages, modification to global variables is a dangerous operation, as it may create side effects in other locations of the code.

- That's why we convention the use of `*earmuffs*` when naming global variables and parameters.
- This issue does not happen with lexical variables, because LET creates its own scope that shields previous values from modification.

```
CL-USER> (let ((x 2))
           (print (+ x x))
           (let ((x 4))
             (print (+ x x))
             (setf x 10))
           (print (+ x x)))
4
8
4
```

We use the SETF special form to modify the value bound to a variable.

```
CL-USER> (setf *upperb* 10)
10
CL-USER> *upperb*
10
```

In this course (and in the textbook), we will sometimes use `:=` as a synonym for SETF.

Let's now define functions GUESS-MY-NUMBER, SMALLER, and BIGGER

The Guess-My-Number Game (cont.)

```
 (defun guess-my-number ()
   "Returns the integer mean of *upperb* and *lowerb*"
   (ash (+ *upperb* *lowerb*) -1))

(defun smaller ()
  (setf *upperb* (1- (guess-my-number)))  ; Note: 1- is the name of a function
  (guess-my-number))

(defun bigger ()
  (setf *lowerb* (1+ (guess-my-number)))   ; Note: 1+ is the name of a function
  (guess-my-number))
```

Let's run some tests. Suppose the number is 40. Let's see if lisp can guess it.

```
CL-USER> (guess-my-number)
25
CL-USER> (bigger)
37
CL-USER> (bigger)
43
CL-USER> (smaller)
40
```

Since the values of `*lowerb*` and `*upperb*` are changing as we play the game, let's define a function that resets the game parameters and enables us to start over.

For further study and practice

The Guess-My-Number Game (cont.)

Starting over the game:

```
(defun start-over ()
  (setf *lowerb* 1)
  (setf *upperb* 100)
  (guess-my-number))
```

```
CL-USER> (start-over)
50
CL-USER> (bigger)
75
CL-USER> (bigger)
88
CL-USER> (smaller)
81
CL-USER> (start-over)
50
```

For further study and practice (cont.)

Let's define a bank account application considering the following requirements:

- A user may either withdraw or deposit an amount to their account

- Withdrawals:
    - if there is enough balance in the account, the WITHDRAW function should return the current balance after the amount is subtracted from the balance
    - If there isn't enough balance, then WITHDRAW should output the message "Insufficient funds" and return the balance unchanged.

```
;; bank-account.lisp
(defvar *balance* 100)

(defun withdraw (amount)
  (if (>= *balance* amount)
    (setf *balance* (- *balance* amount))
    (print "Insufficient funds"))
  *balance*)
```

```
CL-USER> (load "bank-account.lisp")
WITHDRAW
CL-USER> *balance*
100
CL-USER> (withdraw 5)
95
CL-USER> (withdraw 100)
"Insuficient funds"
100
```

Exercise

Change the WITHDRAW function so that there is a limit of 10000 dollars per withdrawal and negative amounts should not be processed. For both cases, WITHDRAW should output an appropriate warning message and return the current balance, without changing it.

```
;; bank-account.lisp
(defvar *balance* 100)

(defun withdraw (amount)
  (if (>= *balance* amount)
    (setf *balance* (- *balance* amount))
    (print "Insufficient funds"))
  *balance*)
```

Solution

```
(defvar *balance* 100)

(defun withdraw (amount)
  (if (>= amount 10000)
      (print "Exceeds maximum withdrawal amount"))
  (if (< amount 0)
      (print "Negative amount"))
  (if (and (< amount 10000)
           (> amount 0)
           (>= *balance* amount))
      (setf *balance* (- *balance* amount))
      (print "Insufficient funds"))
  *balance*)
```