Week 02 - Lecture 1 Slides

Dr. Yeganeh Bahoo

Created: 2023-09-11 Mon 17:05

Table of Contents

Lecture 1: crash course on Lisp (cont.)

Learning objectives:

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

- describe and write Lisp expressions that combine the following programming elements to create abstractions
  - Simple recursion
  - conditionals (multi branching)
  - looping using DOTIMES

Exercise

A quick review of what we have learned so far:

1. What does REPL mean and what is it?
2. If you type the forms below on the REPL, what would you get?

```
CL-USER (setf x 3)
x

CL-USER x

???
CL-USER> '(+ 2 x)

???
CL-USER> (+ 2 x)

???
CL-USER> (eval '(+ 2 x))

???
CL-USER> 'x

???
```

Solution

```
CL-USER (setf x 3)
x

CL-USER x

3
CL-USER> '(+ 2 x)

(+ 2 x)
CL-USER> (+ 2 x)

5
CL-USER> (eval '(+ 2 x))

5
```

```
CL-USER> 'x

X
```

Exercise

Review (cont.)

If you type the forms below on the REPL, what would you get?

```
CL-USER> (if nil 'hello 'world)

???
CL-USER> (if (< 2 3) "2 smaller" "3 smaller")

???
CL-USER> (when (> 2 3) (print "no way"))

???
CL-USER> (unless (< 4 2) 10)

???
```

Solution

```
CL-USER> (if nil 'hello 'world)

WORLD
CL-USER> (if (< 2 3) "2 smaller" "3 smaller")

"2 smaller"
CL-USER> (when (> 2 3) (print "no way"))

NIL
CL-USER> (unless (< 4 2) 10)

10
```

Exercise

Provide the missing expressions in the definition of function FACT that computes the factorial of a number:

$$x! = \begin{cases} 1 & \text{if } x <= 1 \\ x(x-1)! & \text{if } x > 1 \end{cases}$$

```
(defun fact (x)
  (if ...   ...              ; 1
      (* x  (... (- x 1)))))  ; 2
```

Solution

```
(defun fact (x)
  (if (<= x 1) 1
      (* x (fact (- x 1)))))
```

```
RTL-USER> (trace fact)
(FACT)
RTL-USER> (fact 4)
  0: (FACT 4)
    1: (FACT 3)
      2: (FACT 2)
        3: (FACT 1)
        3: FACT returned 1
      2: FACT returned 2
    1: FACT returned 6
  0: FACT returned 24
24
```

The COND special form

No problems in the code below, but it's not pretty.

```
(defun whereis (city)
  (if (eq city 'toronto) 'canada
      (if (eq city '东风航天城) 'china ; DongFeng
          (if (eq city 'Звёздный-городо́к) 'russia ; Zvyozdny gorodok
              (if (or (eq city 'תל־אביב-יפו) (eq city 'تل أبيب - يافا)) 'israel  ; Tel Aviv-Yafo
                  'unknown)))))
```

The COND special form makes the code a lot more readable.

```
(defun whereis (city)
  (cond ((eq city 'toronto) 'canada)
        ((eq city '东风航天城) 'china) ; Dongfeng
        ((eq city 'Звёздный-городо́к) 'russia) ; Zvyozdny gorodok
```

```
        ((or (eq city 'תל־אביב-יפו) (eq city 'يافا - تل أبيب)) 'israel)
        (t 'unknown)))

CL-USER> (whereis 'Звёздный-городо́к)
RUSSIA
CL-USER> (whereis 'xanadu)
UNKNOWN
```

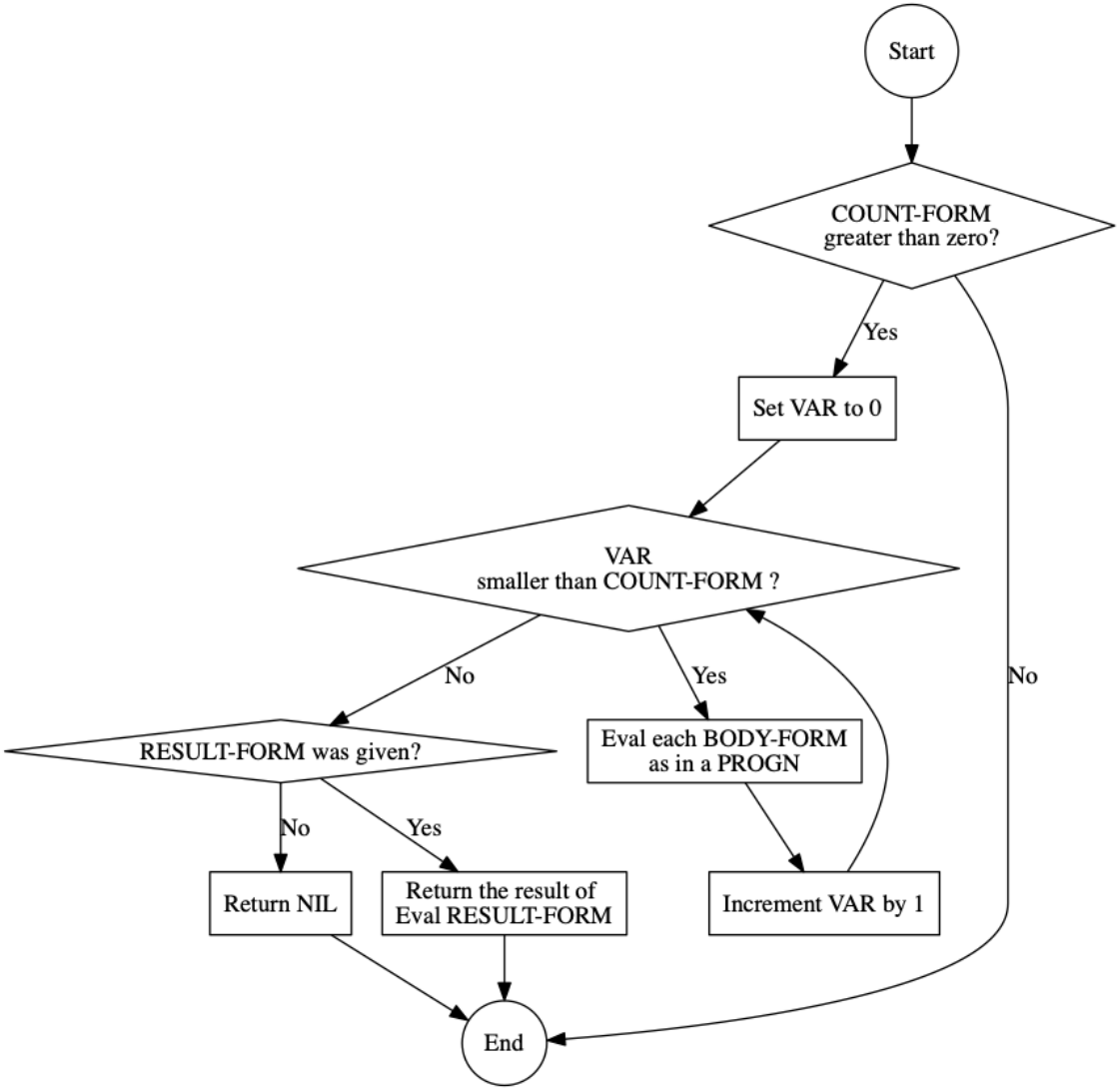
BONUS HINT:

- use EQ to compare symbols;
- use EQUAL to compare everything else.

Looping: DOTIMES

Syntax & Semantics:

```
(DOTIMES (var count-form [result-form])     =>  returns value of the optional RESULT-FORM;
  body-form*)                                   otherwise returns NIL
```



Example:

```
CL-USER> (let ((temp 1))
           (dotimes (temp1 3 temp)
             (setf temp (+ temp temp1))
             (format t "Temp: ~a Temp1: ~a~%" temp temp1)))
 Temp: 1 Temp1: 0
 Temp: 2 Temp1: 1
 Temp: 4 Temp1: 2
 4
CL-USER>
```

RETURN: breaking out of a loop

You can use the RETURN statement to break out of a loop

```
CL-USER> (dotimes (i 6 "done")  (print i))   ; No use of RETURN. DOTIMES returns the value of the RESULT-FORM
0
1
2
```

```
3
4
5
"done"
CL-USER> (dotimes (i 6) (if (= i 3) (return) (print i)))   ; RETURN without an argument returns NIL
0
1
2
NIL
CL-USER> (dotimes (i 6) (if (= i 3) (return "exited") (print i)))  ; RETURN with an argument returns its argument
0
1
2
"exited"
```

DOTIMES: example

- Let's use DOTIMES to iterate over the characters of a string
- Notice the example below uses some interesting functions
  - CHAR-CODE returns the ASCII code of a character (CODE-CHAR does the inverse)
  - CHAR>= and CHAR<= are relational operators for characters
  - the accessor `(AREF str i)` returns the *i*-th character in the string *str*.

    For example: `(AREF "fdsa" 1) => #\d`

```
(defun cpt-char (c)
  (if (and (char>= c #\a) (char<= c #\z))
      (code-char (- (char-code c) 32))
      c))

(defun capitalize (s)
  "Capitalizes the characters in string s"
  (dotimes (i (length s) s)
    (setf (aref s i) (cpt-char (aref s i)))))
```

```
RTL-USER> (cpt-char #\a)
 #\A
RTL-USER> (capitalize "cps305 Data Structures")
 "CPS305 DATA STRUCTURES"
```

Exercise

Given functions CPT-CHAR and VOWELP below, complete the blanks in function CPT-VOWELS that capitalizes the vowels of a given string.

```
(cpt-vowels "CPS305 Data Structures")  =>  "CPS305 DAtA StrUctUrEs"
```

```
(defun cpt-char (c)
  "Capitalizes the character in parameter c"
  (if (and (char>= c #\a) (char<= c #\z))
      (code-char (- (char-code c) 32))
      c))

(defun vowelp (c)
  "Returns true if c is a character representing a vowel"
   (or (char= c #\a)
       (char= c #\e)
       (char= c #\i)
       (char= c #\o)
       (char= c #\u)))

(defun cpt-vowels (s)
  (dotimes (i (length s) ...)
    (when ...
        (setf (aref s i) (... (aref s i))))))
```

Solution

```
(defun cpt-char (c)
  (if (and (char>= c #\a) (char<= c #\z))
      (code-char (- (char-code c) 32))
      c))

(defun vowelp (c)
  "Returns true if c is a character representing a vowel"
   (or (char= c #\a)
       (char= c #\e)
       (char= c #\i)
       (char= c #\o)
       (char= c #\u)))

  (defun cpt-vowels (s)
    (dotimes (i (length s) s)
      (when (vowelp (aref s i))
        (setf (aref s i) (cpt-char (aref s i))))))
```