ANSI Common Lisp Practice

ismdeep

December 29, 2019

Contents

1 chapter-01 **1**
1.1 sum 1
1.2 addn 1

2 chapter-02 **1**
2.1 Form 1
2.2 Evaluation 1
2.3 Data 1
2.4 List Operations 1
2.5 Truth 1
2.6 Functions 1
2.7 Recursion 1
2.8 Reading Lisp 1
2.9 Input and Output 1
2.10 Variables 1
2.11 Assignment 2
2.12 Functional Programming 2
2.13 Iteration 2
2.14 Functions as Objects 2

1 chapter-01

1.1 sum

```
; (dotimes (i n s) () ...)
; i => [0, 1, ... , n]
; return value is s
; ... is operations

(defun sum (n)
   (let ((s 0))
      (dotimes (i n s)
         (incf s i))))

(format t "~D~%" (sum 10))
```

1.2 addn

```
; lambda ?
; I don't know how to use it yet. -_-

(defun addn (n)
   #'(lambda (x)
      (+ x n)))

(format t "~A~%" (addn 10))
```

2 chapter-02

2.1 Form

```
(format t "~A~%" (+ 1 2))
(format t "~A~%" (+ 1 2 3 4 5))
(format t "~A~%" (/ (- 7 1) (- 4 2)))
```

2.2 Evaluation

```
(format t "~A~%" (quote (+ 3 5)))
(format t "~A~%" '(+ 3 4))
```

2.3 Data

```
(format t "~A~%" 'Hello)
(format t "~A~%" '(my 3 "Sons"))
(format t "~A~%" (list 'my (+ 2 1) "Sons"))
(format t "~A~%" ())
(format t "~A~%" nil)
```

2.4 List Operations

```
(format t "~A~%" (cons 1 '(2 3 4)))
(format t "~A~%" (car '(1 2 3 4)))
(format t "~A~%" (cdr '(1 2 3 4)))
(format t "~A~%" (car (cdr (cdr '(1 2 3 4)))))
(format t "~A~%" (third '(1 2 3 4)))
```

2.5 Truth

```
(format t "~A~%" (listp '(1 2 3 4)))
(format t "~A~%" (null nil))
(format t "~A~%" (not nil))
(format t "~A~%" (if (listp '(a b c))
   (+ 1 2)
   (+ 5 6)))
```

2.6 Functions

```
(defun our-third (x)
   (car (cdr (cdr x))))

(format t "~A~%" (our-third '(a b c d)))
```

2.7 Recursion

```
(defun is-member (obj lst)
   (if (null lst)
      nil
      (if (eql (car lst) obj)
         T
         (is-member obj (cdr lst)))))

(format t "~A~%" (is-member 1 '(2 3 4 1 7 8)))
```

2.8 Reading Lisp

```
(defun our-member (obj lst) (if (null lst) nil
    (if
(eql (car lst) obj) lst (our-member obj (cdr
    lst)))))
```

2.9 Input and Output

```
(format t "~A plus ~A equals ~A. ~%" 2 3 (+ 2 3))

(defun askem (string)
   (format t "~A~%" string)
   (read))

(let ((age (askem "How old are you?")))
   (format t "I'm ~A year old.~%" age))
```

2.10 Variables

```
; create local variable through let
(let ((x 1) (y 2))
   (format t "~A~%" (+ x y)))

; create local variable throught let in a
    function
(defun ask-number ()
   (format t "Please enter a number.~%")
   (let ((val (read)))
      (if (numberp val)
         val
         (ask-number))))

; call function ask-number
(format t "~A~%" (ask-number))

; create a global variable
(defparameter *global-var* 100)

; create a global constant
(defconstant LIMIT 100)


(format t "~A~%" *global-var*)

; test a symbol is a global variable
(format t "~A~%" (boundp '*global-var*))

(format t "~A~%" LIMIT)
```

2.11 Assignment

```
(declaim (sb-ext:muffle-conditions cl:warning))

(setf *glob* 98)

(format t "~A~%" *glob*)

(format t "~A~%" (let ((n 10))
   (setf n 2)
   n))

(setf x (quote (a b c)))
(setf (car x) 'x)
(format t "~A~%" x)

(setf a 1
     b 2
     c 3)

(format t "~A~%" b)
```

2.12 Functional Programming

```
(defparameter lst '(c a r a t))
(format t "~A~%" (remove 'a lst))
(format t "~A~%" lst)
(setf lst (remove 'a lst))
(format t "~A~%" lst)
```

2.13 Iteration

```
; iteration version
(defun show-squares-iteration (start end)
   (do ((i start (+ i 1)))
      ((> i end) 'done)
      (format t "~A ~A~%" i (* i i))))

; recursion version
(defun show-squares-recursion (start end)
   (if (> start end)
      'done
      (progn
         (format t "~A ~A~%" start (* start
            start))
         (show-squares-recursion (+ start 1)
            end))))



(show-squares-iteration 1 10)
(show-squares-recursion 1 10)




; our-length iteration version
(defun our-length-iteration (lst)
   (let ((len 0))
      (dolist (obj lst)
         (setf len (+ len 1)))
      len))

; our-length recursion version
(defun our-length-recursion (lst)
   (if (null lst)
      0
      (+ (our-length-recursion (cdr lst)) 1)))

(defparameter *lst* (quote (1 2 3 4 5)))
(format t "~A~%" (our-length-iteration *lst*))
(format t "~A~%" (our-length-recursion *lst*))
```

2.14 Functions as Objects

```
(format t "~A~%" (function +))
(format t "~A~%" #'+)
(format t "~A~%" (apply #'+ '(1 2 3)))
(format t "~A~%" (apply #'+ 1 2 '(3 4 5)))
(format t "~A~%" (funcall #'+ 1 2 3 4 5))
(format t "~A~%" (lambda (x y) (+ x y)))
(format t "~A~%" ((lambda (x) (* x x)) 10))
(format t "~A~%" (funcall #'(lambda (x) (* x x))
   10))
```