Foundations on Programming Language - Lecture Notes

Version 8.15

Juan Pablo Sandoval

March 4, 2025

Contents

This text book have a set of exercises and test to learn scheme and programming languages.

1 Part I: Scheme Basics

1.1 Data Types, Functions and Lists

Primitive Data Types.

```
3       ; Number
1.02    ; Decimal Number
2/3     ; Fraction
#t      ; True
#f      ; False
```

Strings and Symbols.

```
"hola" ; Strings
'hola  ; Symbols
```

Predefined Functions.

```
(+ 1 2)     ; + is actually a function that recieve N arguments
(* 1 2 3)   ; * is also a function
(sqrt 4)    ; root square
(and (> 3 2) (equal? 1 1))   ; composing function calls
```

Console Printing.

```
(printf "Hello, World!\n")
```

Global Identifiers.

```
(define MAX 100)
MAX
(define score 99)
(+ score 1)
```

Local Identifiers.

```
(let ([precio 100]    ; price before taxes
      [impuesto 0.19])  ; tax rate
  (+ precio (* precio impuesto)))
```

Conditionals and Functions.

```
(define (grade score)
  (if (> score 90)
      "A"
      (if (> score 80)
          "B"
          (if (> score 70)
              "C"
              "D"))))
(grade 92) ; return "A"

(define (max a b) (if (< a b) b a))
(max 2 3) ; return 3
```

Recursion

```
(define (factorial n)
  (if (= n 0)
      1
      (* n (factorial (- n 1)))))
(factorial 3) ; return 6
```

1.2 Lists and Pairs

Pares.

```
(cons 1 2)
(car (cons 1 2)) ; extract the first element
(cdr (cons 1 2)) ; extract the second element
```

Lists.

```
empty            ; empty list
(cons 1 (cons 2 (cons 3 empty)))   ; a list with elements 1 2 3
(list 1 2 3)                       ; function list build a list with the arguments passed
(append (list 1 2 3) (list 4 5 6)) ; join two lists
```

Recursive Functions.

```
(define (sum lst)
  (if (empty? lst)
      0
      (+ (car lst) (sum (cdr lst)))))

(sum (list 1 1 1))  ; return 3
```

```
(define (exist? elem lst)
  (if (empty? lst)
      #f
      (if (equal? elem (car lst))
          #t
          (exist? elem (cdr lst)))))
(exist? 2 (list 1 2 3))  ; return true #t
```

1.3 Exercises

Invertir el Orden. Esta función debe devoler una lista que tenga todos los elementos de la lista recibida como argumento pero en orden inverso.

```
;; reverse: (List) --> (List)
(define (reverse lst) ...)
```

Considere los siguientes test para ayudarlo a realizar su solución.

```
(test (reverse empty) empty)
(test (reverse (list 1)) (list 1))
(test (reverse (list 1 2)) (list 2 1))
(test (reverse (list 1 2 3 4 5 6 7 8 9 10)) (list 10 9 8 7 6 5 4 3 2 1))
```

Insertar al Final. Esta función debe devolver una lista que incluye el elemento a insertar al final de la lista

```
;; insert: (Any,List) --> (List)
(define (insert elem lst) ...)
```

Considere los siguientes test para ayudarlo a realizar su solución.

```
(test (insert 1 empty) (list 1))
(test (insert 2 (list 1)) (list 1 2))
(test (insert 3 (list 1 2)) (list 1 2 3))
(test (insert 4 (list 1 2 3)) (list 1 2 3 4))
```

Eliminar Elemento. Esta función devuelve una lista sin el elemento a eliminar. Solo debe eliminar el primer elemento que encuentre.

```
;; delete: (Any,List) --> (List)
(define (delete elem lst) ...)
```

```
(test (delete 1 empty) empty)
(test (delete 1 (list 1)) empty)
(test (delete 2 (list 1 2)) (list 1))
(test (delete 3 (list 1 2 3)) (list 1 2))
(test (delete 5 (list 1 2 3 4 5 6 7 8 9 10)) (list 1 2 3 4 6 7 8 9 10))
(test (delete 10 (list 1 2 3 4 5 6 7 8 9 10)) (list 1 2 3 4 5 6 7 8 9))
```