Programación interactiva

José A. Alonso y María J. Hidalgo

Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Caracteres

- **Representación de caracteres en Scheme:**

```
Carácter Representación en Scheme
a        #\a
A        #\A
3        #\3
\        #\\
```

- **Evaluación de caracteres**

```
> #\a
#\a
> #\A
#\A
> #\3
#\3
```

Caracteres

- **Reconocimiento de caracteres:**

```
(char? #\a)  =>  #t
(char? 'a)   =>  #f
(char? #\3)  =>  #t
(char? 3)    =>  #f
```

- **Caracteres y códigos ASCII:**

```
(char->integer #\0)  =>  48
(char->integer #\1)  =>  49
(char->integer #\A)  =>  65
(char->integer #\B)  =>  66
(char->integer #\a)  =>  97
(char->integer #\b)  =>  98

(integer->char 98)  =>  #\b
```

Caracteres

- **Caracteres especiales:**

```
(char->integer #\space)      =>  32
(integer->char 32)           =>  #\space
(char->integer #\newline)    =>  10
(integer->char 10)           =>  #\nl
(char->integer #\return)     =>  13
(integer->char 13)           =>  #\cr
(char->integer #\backspace)  =>  8
(integer->char 8)            =>  #\bs
(char->integer #\tab)        =>  9
(integer->char 9)            =>  #\ht
```

Caracteres: Comparación

- **Criterio de ordenación: Por código ASCII.**

- **Procedimientos de comparación:**

```
char<?
char<=?
char>?
char>=?
char=?
```

- **Ejemplos:**

```
(char<? #\A #\C)          =>  #t
(char<? #\a #\C)          =>  #f
(char>? #\a #\C)          =>  #t
(char=? #\a #\A)          =>  #f
(char=? #\newline #\nl)   =>  #t
```

Caracteres: Comparación

- **Procedimientos de comparación insensibles:**

```
char-ci<?
char-ci<=?
char-ci>?
char-ci>=?
char-ci=?
```

- **Ejemplos:**

```
(char-ci=? #\a #\A)  =>  #t
(char-ci<? #\a #\C)  =>  #t
```

Caracteres: Mayúsculas y minúsculas

- **Predicados sobre mayúsculas y minúsculas:**

```
(char-lower-case? #\a)  =>  #t
(char-lower-case? #\A)  =>  #f
(char-upper-case? #\a)  =>  #f
(char-upper-case? #\A)  =>  #t
```

- **Transformación de mayúsculas a minúsculas y viceversa:**

```
(char-upcase #\a)    =>  #\A
(char-downcase #\A)  =>  #\a
```

Cadenas

- **Definición de cadena**

```
"Hola, como estás"
"1 2 3 45 6"
```

- **Reconocimiento de cadenas:**

```
(string? "a bc")  =>  #t
(string? 'a-bc)   =>  #f
```

- **Longitud de una cadena:**

```
(string-length "a bc")  =>  4
```

Cadenas

- **Procedimientos de comparación de cadenas:**

```
Sensible    Insensible
string<?    string-ci<?
string<=?   string-ci<=?
string>?    string-ci>?
string>=?   string-ci>=?
string=?    string-ci=?
```

- **Ejemplos de comparación de cadenas:**

```
(string<? "Nota aprobado" "Nota notable")  =>  #t
```

Cadenas

- **Procedimientos relativos a cadenas:**

```
(string-ref "a bc d" 0)                 =>  #\a
(string-ref "a bc d" 2)                 =>  #\b
(make-string 3 #\a)                     =>  "aaa"
(string #\a #\  #\b #\tab #\c)          =>  "a b    c"
(substring "a bc d e" 2 6)              =>  "bc d"
(string-append "a b" "d" " f")          =>  "a bd f"
(string-null? "")                       =>  #t
(string-null? "a b")                    =>  #f
(string->list "a bc")                   =>  (#\a #\space #\b #\c)
(list->string '(#\a #\space #\b #\c))   =>  "a bc"
```

Procedimiento substring?:

```
;;; (subcadena? "a b" "a bc d")    =>  #t
;;; (subcadena? "a bcd" "a bc d")  =>  #f

(define subcadena?
  (lambda (s1 s2)
    (let ((n1 (string-length s1))
          (n2 (string-length s2)))
      (letrec
          ((subcadena?-aux
            (lambda (k)
              (cond ((> (+ k n1) n2) #f)
                    ((string=? s1 (subcadena s2 k (+ k n1)))
                     #t)
                    (else (subcadena?-aux (+ k 1)))))))
        (subcadena?-aux 0)))))
```

Procedimiento cadena-inversa

```
;;; (cadena-inversa "a bc de")  =>  "ed cb a"
(define cadena-inversa
  (lambda (c)
    (if (string-null? c)
        c
        (string-append (cadena-inversa (substring c 1 (string-length c)))
                       (make-string 1 (string-ref c 0))))))

(define cadena-inversa
  (lambda (c)
    (list->string (reverse (string->list c)))))
```

Procedimientos de entrada y salida:

- **Procedimientos de escritura:**

```
> (begin
    (write "a bc d")
    (newline)
    (display "a bc d")
    (newline))
"a bc d"
a bc d
#<unspecified>
```

Procedimientos de entrada y salida:

- **Procedimiento de lectura:**

```
> (begin
    (display "Escribe un numero: ")
    (define n (read))
    (display "El cuadrado de ")
    (display n)
    (display " es ")
    (display (* n n))
    (newline))
Escribe un numero: 3
El cuadrado de 3 es 9
#<unspecified>
```

El juego de la adivinanza

- **Sesión**

```
> (adivina)
Escribe un numero: 50
Es bajo
Escribe un numero: 75
Es alto
Escribe un numero: 62
Es bajo
Escribe un numero: 68
Es alto
Escribe un numero: 64
Es bajo
Escribe un numero: 66
Acertado
#<unspecified>
```

El juego de la adivinanza

- **Procedimiento**

```
(require 'random)
(define adivina
  (lambda ()
    (letrec
        ((numero (random 100))
         (adivina-aux
          (lambda ()
            (display "Escribe un numero: ")
            (let ((n (read)))
              (cond ((= n numero)
                     (display "Acertado")
                     (newline))
                    (else (display "Es ")
                          (if (< n numero)
                              (display "bajo")
                              (display "alto"))
                          (newline)
                          (adivina-aux)))))))
      (adivina-aux))))
```

Raiz cuadrada interactiva

- **Sesión**

```
> (raiz-cuadrada-interactiva)

Bienvenido al calculador de raices cuadradas

Escribe un número o q si quieres dejarlo: 9
Escribe un número para empezar el cálculo: 1
Escribe el error máximo aceptable: 0.01
La raiz cuadrada de 9 es 3.00009155413138

Escribe un número o q si quieres dejarlo: 25
Escribe un número para empezar el cálculo: 2
Escribe el error máximo aceptable: 0.0001
La raiz cuadrada de 25 es 5.00000000001678

Escribe un número o q si quieres dejarlo: q

Gracias, vuelve otra vez
```

Raiz cuadrada interactiva

```
(define raiz-cuadrada-interactiva
  (lambda ()
    (letrec
        ((interaccion
          (lambda ()
            (let ((n (llamada
                      "Escribe un número o q si quieres dejarlo: ")))
              (if (eq? n 'q)
                  (escribe "Gracias, vuelve otra vez")
                  (let ((d (llamada
                            "Escribe un número para empezar el cálculo: "))
                        (e (llamada
                            "Escribe el error máximo aceptable: ")))
                    (display "La raiz cuadrada de ") (display n)
                    (display " es ") (display (raiz-cuadrada n d e))
                    (newline) (newline)
                    (interaccion))))))))
      (escribe "Bienvenido al calculador de raices cuadradas")
      (interaccion))))
```

Raiz cuadrada interactiva

```
(define escribe
  (lambda (cadena)
    (newline)
    (display cadena)
    (newline)
    (newline)))

(define llamada
  (lambda (cadena)
    (display cadena)
    (read)))
```

Raiz cuadrada interactiva

```
(define raiz-cuadrada
  (lambda (n d e)
    (letrec
      ((aceptable?
        (lambda (y x)
          (< (abs (- (* y y) x))
             e)))
       (mejora
        (lambda (y x)
          (/ (+ y (/ x y)) 2)))
       (raiz-cuadrada-iter
        (lambda (y x)
          (if (aceptable? y x)
              y
              (raiz-cuadrada-iter (mejora y x) x)))))
      (raiz-cuadrada-iter d n))))
```