Funciones Sin Nombre

No es necesario que una función tenga asociado un nombre para ser evaluada.

```
> ((lambda (x) (* x x)) 5)
25

> ((lambda (radio) (* 2 pi radio)) 5)
31.41592

> ((lambda (f) (lambda (x) (f (f x))))
   (lambda (x) (* x x)))
#<procedure>

> (((lambda (f) (lambda (x) (f (f x))))
    (lambda (x) (* x x))) 3)
81

> ((lambda (a b) ((if (> b a) * +) a b)) 4 5)
20
```

Funciones de Orden Superior

Algunas funciones de 1 argumento:

```
(define (suma1 x) (+ x 1))
(define (cuadrado x) (* x x))
(define (cubo x) (* x x x))
(define (id x) x)
```

Ejemplos de funciones de orden superior:

```
(define dos-veces
  (lambda (f) (lambda (x) (f (f x)))))

(define componer
  (lambda (f g) (lambda (x) (f (g x)))))
```

Funciones de Orden Superior

Evaluaciones:

```
> ((dos-veces cuadrado) 2)
16

> ((dos-veces cdr) '(a b c d e))
(c d e)

> ((componer cuadrado suma1) 4)
25

> ((componer cubo car) '(10 20 30))
1000

> (((dos-veces dos-veces) cuadrado) 2)
65536

> ((dos-veces (componer car cdr)) '(10 (20 30)))
30
```

Funciones de Orden Superior

Usando la función "dos-veces" para definir nuevas funciones:

```
(define dv-cuadrado (dos-veces cuadrado))
(define cuatro-veces (dos-veces dos-veces))
```

Evaluaciones:

```
> (dv-cuadrado 5)
625

> ((cuatro-veces cuadrado) 2)
65536

> ((cuatro-veces cdr) '(a b c d e f))
(e f)
```

Funciones de Orden Superior

Ejemplo lista de funciones:

```
(define lista-fun (list cubo suma1 cuadrado))
```

Evaluaciones:

```
> lista-fun
(#<procedure:cubo> #<procedure:suma1> #<procedure:cuadrado>)

> (car lista-fun)
#<procedure:cubo>

> ((car lista-fun) 10)
1000
```

Funciones de Orden Superior

Generalización de la función "componer"

Ejercicio: Dada una lista de funciones, obtener una función de 1 argumento equivalente a la composición de todas las funciones de dicha lista.

```
(define componer-funciones
  (lambda (LF)
    (lambda (x)
      (if (null? LF)
          x
          ((car LF) ((componer-funciones (cdr LF)) x))
       )
     )
   )
 )
```

Funciones de Orden Superior

Evaluaciones:

```
> ((componer-funciones lista-fun) 2)
125

> ((componer-funciones (list cuadrado car cdr))
   '(10 20 30))
400
```

Evaluar y explicar el resultado:

```
> (((componer-funciones (list dos-veces dos-veces dos-veces))
    cuadrado) 2)
```

Funciones de Orden Superior

Otra alternativa para definir "componer-funciones", usando las funciones identidad y componer:

```
(define componer-funciones
  (lambda (LF)
    (if (null? LF)
        id
        (componer (car LF)
                  (componer-funciones (cdr LF)))
    )
  )
)
```

Funciones de Orden Superior

Para introducir la siguiente función de orden superior, planteamos algunas funciones simples para operar con listas:

```
(define (suma-lista lista)
  (if (null? lista)
      0
      (+ (car lista)
         (suma-lista (cdr lista)))
   )
 )

(define (producto-lista lista)
  (if (null? lista)
      1
      (* (car lista)
         (producto-lista (cdr lista)))
   )
 )
```

Funciones de Orden Superior

Función ACUMULAR: Generalización de las funciones anteriores.

```
(define (acumular func base)
  (lambda (lista)
    (if (null? lista)
        base
        (func (car lista)
              ((acumular func base) (cdr lista)))
    )
  )
)
```

Esta función se conoce con otros nombres tales como:
foldr (fold-right), accumulate, reduce, aggregate, compress, inject

Funciones de Orden Superior

Evaluaciones:

```
> ((acumular + 0) '(1 2 3 4 5))
15

> ((acumular * 1) '(1 2 3 4 5))
120

> ((acumular cons '()) '(a b c d))
(a b c d)
```

Definición de las funciones vistas usando ACUMULAR:

```
(define suma-lista (acumular + 0))
(define producto-lista (acumular * 1))

(define componer-funciones (acumular componer id))
```

Funciones de Orden Superior

Función map

Función disponible en Scheme que evalúa una función con cada uno de los elementos de una lista, retornando una lista con los resultados.

```
> (map cubo '(1 2 3 4 5))
(1 8 27 64 125)

> (map (componer cuadrado suma1) '(1 2 3 4 5))
(4 9 16 25 36)
```

Posible definición de la función anterior:

```
(define (map func lista)
  (if (null? lista)
      '()
      (cons (func (car lista))
            (map func (cdr lista)))
   )
 )
```

Funciones de Orden Superior

Definición de MAPEAR como función de 1 argumento (una función) que retorna una función de 1 argumento (una lista):

```
(define (mapear func)
  (lambda (lista)
    (if (null? lista)
        '()
        (cons (func (car lista))
              ((mapear func) (cdr lista)))
    )
   )
 )
```

```
> (map sqrt '(4 9 16 25))
(2 3 4 5)

> ((mapear sqrt) '(4 9 16 25))
(2 3 4 5)
```

Funciones de Orden Superior

Función FILTRAR:

```
(define (filtrar func lista)
  (if (null? lista)
      '()
      (if (func (car lista))
          (cons (car lista) (filtrar func (cdr lista)))
          (filtrar func (cdr lista))
       )
   )
 )
```

Evaluaciones:

```
> (filtrar odd? '(4 7 9 2 5 11 14))
(7 9 5 11)

> (filtrar even? '(4 7 9 2 5 11 14))
(4 2 14)
```

Funciones de Orden Superior

Ejemplo con registros de empleados:

```
(define registros
  '(((juan perez)      programador  20000)
    ((andres garcia)   sereno       15000)
    ((antonio rios)    programador  22000)
    ((ariel gonzalez)  gerente      35000)))

(define (nombre reg)    (caar reg))
(define (apellido reg)  (cadar reg))
(define (cargo reg)     (cadr reg))
(define (salario reg)   (caddr reg))

(define (es-programador? reg)
  (eq? (cargo reg) 'programador))

(define (programadores lista-regs)
  (filtrar es-programador? lista-regs))
```

Funciones de Orden Superior

Ejemplo con registros de empleados (cont.):

```
(define (mayor-salario-de-programador lista-regs)
  (maximo (map salario (programadores lista-regs))))

(define (maximo lista)
  ((acumular max (car lista)) (cdr lista)))
```

Evaluaciones:

```
> (map apellido registros)
(perez garcia rios gonzalez)

> (car (programadores registros))
((juan perez) programador 20000)

> (mayor-salario-de-programador registros)
22000
```