```
;;-------------------------
;; Solución práctica 4
;; Fecha: 03/03/2014
;-------------------------

;-------------------------
; Ejercicio 1
;-------------------------

; Apartado a)

(define (g f)
    (lambda (x) (f (f (f x)))))
(define (suma-1 x)
    (+ x 1))

; a.1)

((g suma-1) 2) ;=> 5

; a.2)

((g (g suma-1)) 0) ;=> 9

; Explicación:
; La llamada a (g suma-1) devuelve una función que suma 3 a
; su entrada. Podemos llamar a esa función "suma-3". Por eso,
; (g (g suma-1)) sería equivalente a (g suma-3). Y la definición
; de g muestra se construye una función que llama 3 veces a la
; la función que se pasa por parámetro. O sea, que la expresión
; (g (g suma-1)) devolverá una función que suma 9 al número que
; se le pase como parámetro.

; Apartado b)

; b.1)
; Puede haber muchas soluciones. Lo importante es que f es una
; función que devuelve una función de un parámetro, que a su vez
; invocamos con el argumento 5

(define f (lambda () (lambda (x) (+ x 3))))
((f) 5) ; => 8

; b.2)
; También puede haber muchas soluciones. La llamada a f debe
; devolver una función de un parámetro que invocamos con el
; argumento 3. Una forma sencilla es que devuelva la propia
; "g" que se pasa como parámetro.

(define (g x)
  (+ x 10))
```

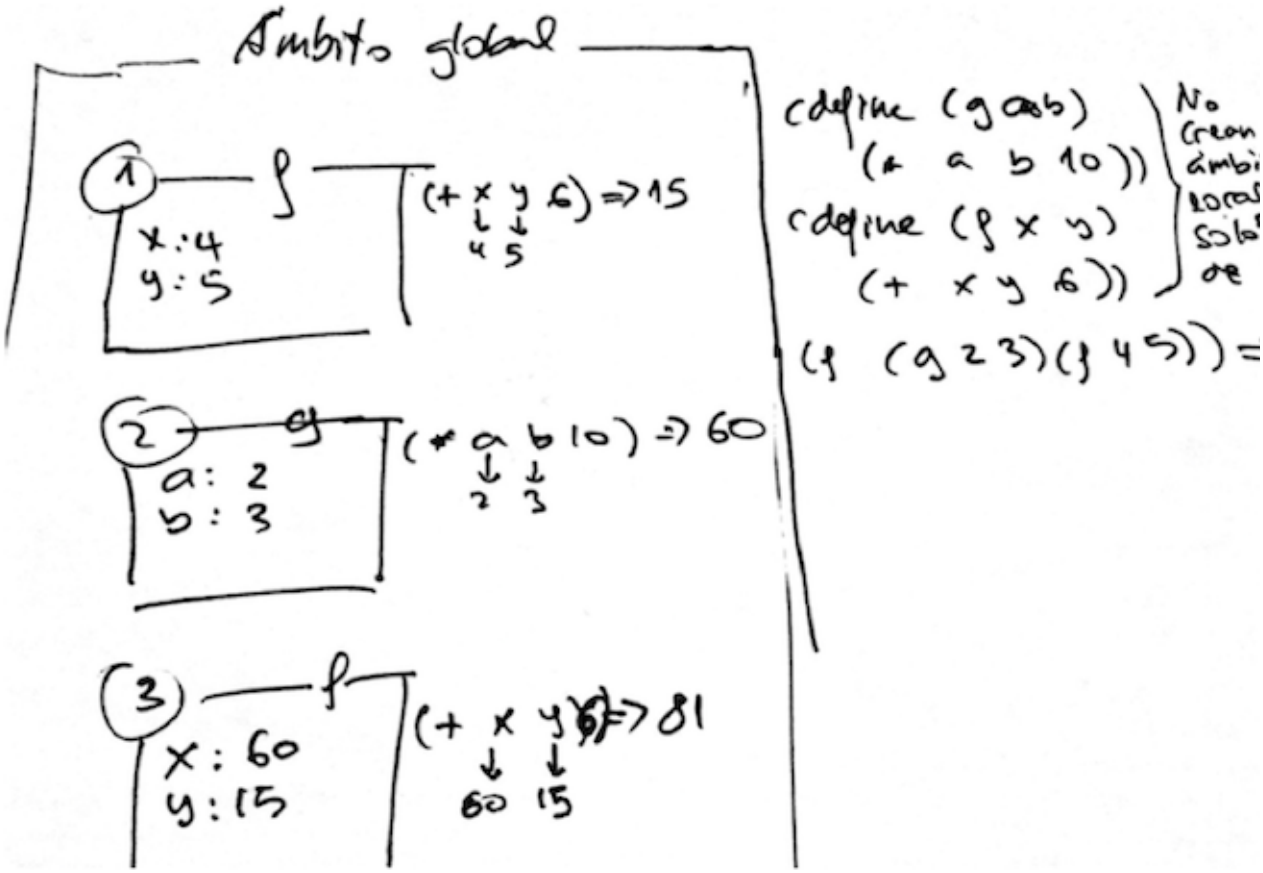
```
(define (f g)
  g)

((f g) 3)  ; => 13

;-------------------------
; Ejercicio 2
;-------------------------

(define (cumple-predicados lista-funcs num)
  (if (null? lista-funcs)
      '()
      (cons ((car lista-funcs) num)
            (cumple-predicados (cdr lista-funcs) num))))

; Pruebas
"Ejercicio 2"
(cumple-predicados (list even? (lambda (x) (> x 5)) (lambda (x) (< x 10))) 6) ;
=> (#t #t #t)


;-------------------------
; Ejercicio 3
;-------------------------

;; a)

(define (fold-with-last f lista)
  (if (null? (cdr lista))
      (car lista)
      (f (car lista) (fold-with-last f (cdr lista)))))

; Pruebas
"Ejercicio 4"

(define (suma x y) (+ x y))
(fold-with-last suma '(1 2 3 4 5)) ; => 15
(define (prueba x y) (* x y 2))
(fold-with-last prueba '(1 2 3 4 5)) ; => 1920

;; b)

;;
;; Funciones auxiliares de prácticas anteriores
;;

(define (intersectan? a1 a2 b1 b2)
  (and (<= b1 a2) (<= a1 b2)))

(define (intersectan-intervalos? a b)
  (intersectan? (car a) (cdr a) (car b) (cdr b)))
```

```
(define (intersectan-intervalos a b)
  (cond ((or (equal? a 'vacio)
             (equal? b 'vacio))  'vacio)
        ((not (intersectan-intervalos? a b)) 'vacio)
        (else (cons (max (car a) (car b))
                    (min (cdr a) (cdr b))))))

;;
;; Solución con la función fold-with-last
;;

(define (mayor x y)
  (if (> x y) x y))

(define (maximo lista)
  (fold-with-last mayor lista))

(define (interseccion-lista-intervalos lista-intervalos)
  (fold-with-last intersectan-intervalos lista-intervalos))

; Pruebas

(maximo '(1 2 20 3 42 -10)) ; => 42
(interseccion-lista-intervalos '((12 . 30) (-8 . 20) (13 . 35)))   ; => (13 . 20)

;--------------------------
; Ejercicio 4
;--------------------------

;;
;; Funciones de orden superior vistas en teoría
;;

(define (filter pred lista)
  (cond
    ((null? lista) '())
    ((pred (car lista)) (cons (car lista)
                              (filter pred (cdr lista))))
    (else (filter pred (cdr lista)))))

(define (fold func base lista)
  (if (null? lista)
      base
      (func (car lista) (fold func base (cdr lista)))))

;; Función "mayor-que" con map

(define (mayor-que n lista)
  (map (lambda (x) (> n x)) lista))

;; Función divisores con filter
```

```
(define (divisores n)
  (filter (lambda (x)
            (divisor? x n)) (lista-hasta n)))

(define (lista-hasta x)
   (if (= x 0)
      '()
      (cons x (lista-hasta (- x 1)))))

(define (divisor? x y)
      (= 0 (remainder y x)))

;; Función "suma-pares-impares" con fold

(define (suma-pares-impares lista)
  (fold suma-pareja-par-impar (cons 0 0) lista))

(define (suma-izquierda num pareja)
  (cons (+ num (car pareja))
        (cdr pareja)))

(define (suma-derecha num pareja)
  (cons (car pareja)
        (+ num (cdr pareja))))

(define (suma-pareja-par-impar num pareja)
  (if (even? num)
      (suma-izquierda num pareja)
      (suma-derecha num pareja)))


; Pruebas

"Ejercicio 4"

(mayor-que 10 '(1 23 4 -1 12 11 10)) ; => (#f #t #f #f #t #t #f)
(divisores 12) ;=> (12 6 4 3 2 1)
(suma-pares-impares '(1 2 3 4 5 6))  ; => (12 . 9)
(suma-pares-impares '(1 3 5 7))      ; => (0 . 16)


;-------------------------
; Ejercicio 5
;-------------------------

;; a
(define (divide-pares-impares lista)
  (if (null? lista) (cons '() '())
      (let ((pareja (divide-pares-impares (cdr lista))))
        (if (even? (car lista))
            (cons (cons (car lista) (car pareja))
                  (cdr pareja))
```

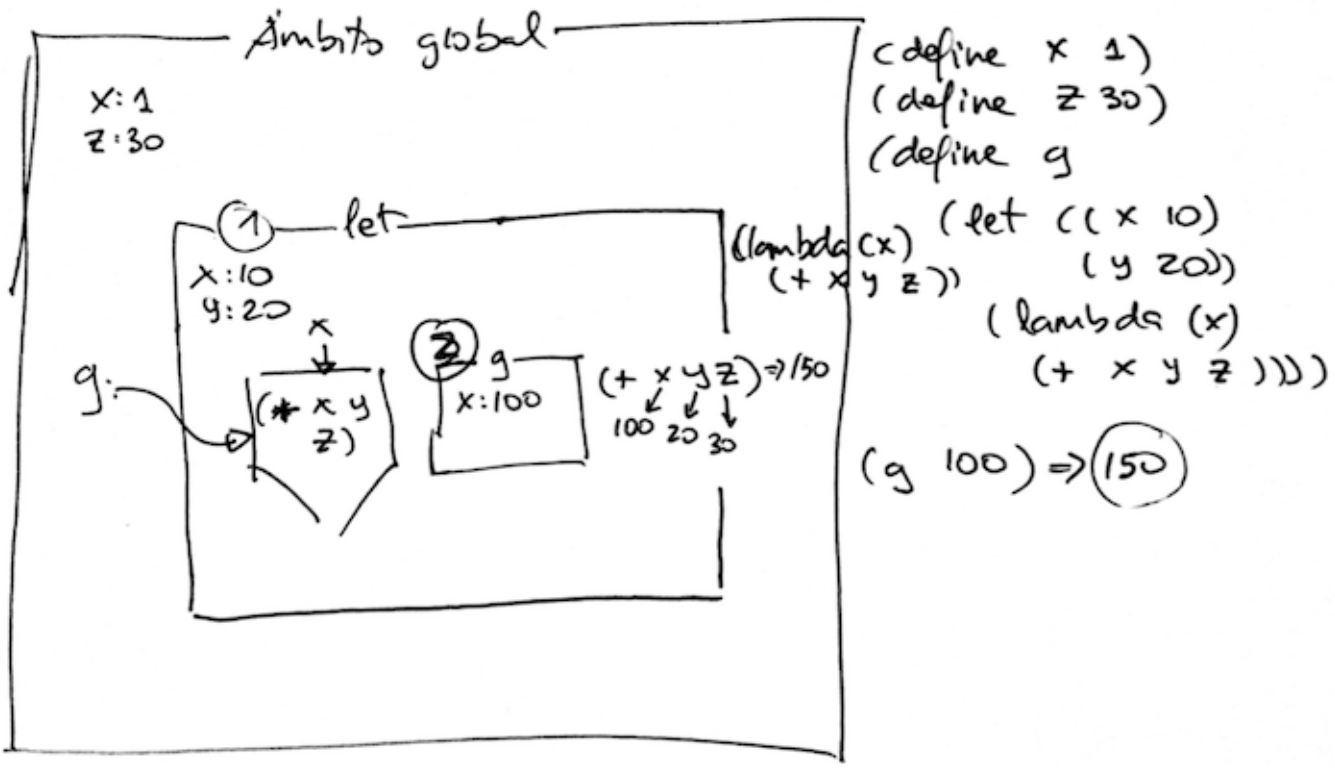
```
            (cons (car pareja)
                  (cons (car lista) (cdr pareja)))))))

;; Pruebas
"Ejercicio 5"
(divide-pares-impares '(1 3 5 2 4 9 8 11)) ; ((2 4 8) 1 3 5 9 11)

;; b
(define (divide-pares-impares lista)
  (cons (filter even? lista) (filter odd? lista)))

;; Pruebas
(divide-pares-impares '(1 3 5 2 4 9 8 11)) ; ((2 4 8) 1 3 5 9 11)


;--------------------------
; Ejercicio 6
;--------------------------

;; a

"Ejercicio 6 a"

(define (g a b)
   (* a b 10))
(define (f x y)
   (+ x y 6))
(f (g 2 3) (f 4 5))

;Resultado de la última expresión: 81
;¿Cuántos ámbitos se crean?: 3
;¿Se crea alguna clausura?: no
;
;
; Explicación:
;
;Todas las expresiones se evalúan en el ámbito global.
;
; 1. Los dos primeros "defines" crean las funciones "g" y "f".
;    Esas definiciones no crean ningún ámbito local.
; 2. La siguiente expresión se evalúa de dentro a fuera.
; 3. Primero evaluamos la expresión "(f 4 5)". Se invoca a la
;    función "f" con los argumentos 4 y 5. Esta invocación
;    crea el ámbito 1, en el que las variables "x" e "y" (parámetros
;    de f) toman los valores 4 y 5.
; 4. En este ámbito local se evalúa el cuerpo de "f", la expresión
;    "(+ x y 6)". La variable "x" vale 4 en el ámbito, la
;    "y" vale 5 y la expresión devuelve 15.
; 5. Después evaluamos en el ámbito global el otro
;    argumento de "f", la expresión "(g 2 3)". Se invoca a la
;    función "g" con los argumentos 2 y 3. Esta invocación
;    crea el ámbito 2, en el que las variables "a" y "b"
```

```
;    (parámetros de g) toman los valores 2 y 3.
; 6. En este ámbito local se evalúa el cuerpo de "g",
;    la expresión "(* a b 10)". La variable a vale 2 en
;    el ámbito, la "b" vale 3 y la expresión devuelve 60.
; 7. Por último, cuando las dos expresiones se han evaluado
;    y han devuelto 15 y 60, se invoca en el ámbito global a "f"
;    con estos dos valores. Se crea el ámbito local 3, en
;    el que las variables "x" e "y" toman el valor de 60
;    y 15.
; 8. En este ámbito local se evalúa el cuerpo de "f",
;    la expresión "(+ x y 6)". Esta expresión devuelve 81.
;
;Dibujo de los ámbitos creados:
;
```




```
;; b

"Ejercicio 6 b"

(define x 1)
(define z 30)
(define g
   (let ((x 10)
         (y 20))
```

```
      (lambda (x)
              (+ x y z))))
(g 100)

;
;Resultado de la última expresión: 150
;¿Cuántos ámbitos locales se crean?: 2
;¿Se crea alguna clausura?: si
;
;
; Explicación:
;
;Todas las expresiones se evalúan en el ámbito global.
;
; 1. Los dos primeros "defines" crean las variables "x"
;    y "z" y les dan los valores 1 y 30.
; 2. El tercer "define" realiza primero una evaluación
;    del "let": se crea el ámbito local 1 en el que
;    se definen las variables "x" e "y" y con los valores
;    10 y 20.
; 3. En el ámbito del let se evalúa la expresión
;    "(lambda (x) (+ x y z)" que crea una función
;    de parámetro "x" y cuerpo "(+ x y z)". Esta
;    función es el resultado que devuelve la evaluación
;    del "let". Es una clausura.
; 4. La función devuelta por el "let" se asigna a la
;    variable g en el ámbito global.
; 5. Se evalúa la expresión "(g 100)". Se invoca
;    la función ligada a la variable g (la clausura).
;    Se crea un ámbito local DENTRO DEL ÁMBITO EN EL QUE
;    ESTÁ ENCERRADA LA CLAUSURA en el que la variable
;    "x" (parámetro de g) toma el valor de 100.
; 6. En este nuevo ámbito 2 se evalúa el cuerpo de la
;    clausura, la expresión "(+ x y z)". La variable
;    x tiene el valor 100 en el ámbito 2, la variable
;    y no está definida en el ámbito 2 y se comprueba
;    el valor en el ámbito padre (ámbito 1). Vale 20.
;    Por último, la variable z toma su valor del ámbito
;    global, donde se ha definido con 30. La suma
;    devuelve 150.


;Dibujo de los ámbitos creados:
```




```
;-------------------------
; Ejercicio 6
;-------------------------

(define x 10)
(define (g z)
    (let ((x 20)
          (y 5))
       (lambda (y) (+ x y z))))
(define a (g x))
(a x)

;
;Resultado de la última expresión: 40
;¿Cuántos ámbitos locales se crean?: 2
;¿Se crea alguna clausura?: si
;


; Explicación:
;
; 1. Se define la variable "x" con el valor 10
; 2. Se crea la función "g" con parámetro "z" y un "let"
;    en su cuerpo. No se crea ningún ámbito local, porque
;    sólo estamos definiendo una función, no invocándola.
; 3. Al evaluar "(define a (g x))" se invoca a la función
```

```
;    g pasándole 10 (el valor de "x"). Se crea el ámbito
;    local 1 en el que se evalúa la función. La variable
;    z (el parámetro de "g") toma el valor 10.
; 4. En este ámbito se evalúa la expresión "(let ...)".
;    Se crea un nuevo ámbito local 2 en el que "x" vale 20
;    e "y" vale 5. En este ámbito 2 se evalúa el cuerpo
;    del "let", la forma especial "lambda" que crea una
;    clausura. La clausura es el objeto devuelto por el
;    let y se asigna a la variable "a" en el ámbito global.
; 5. Se evalúa la expresión "(a x)" en el ámbito global.
;    La variable "x" toma el valor 10 y se invoca a la función
;    a, la clausura dentro del ámbito 2.
; 6. La invocación a la clausura crea un nuevo ámbito local 3
;    dentro del 2, con la variable "y" (el parámetro de la
;    clausura) valiendo 10.
; 7. En este ámbito 3 se evalúa el cuerpo de la clausura,
;    la expresión "(+ x y z)". La variable "x" toma el valor
;    20 del ámbito 2, la variable "y" toma el valor 10 del
;    propio ámbito 3 y la variable "z" toma el valor 10 del
;    ámbito 1. La suma devuelve 40.
;

;Dibujo de los ámbitos creados:
```