Práctica 4: Funciones como datos de primera clase

Nota: Para trabajar con la función de orden superior `filter` puedes copiarla de los apuntes de teoría o activar el lenguaje “Muy Grande” en el que sí que está definida.

Ejercicio 1

Intenta hacer todos los siguientes apartados **sin usar el intérprete de Scheme**, usando sólo lápiz y papel. Usa el intérprete sólo para comprobar que la solución que has escrito es correcta.

a.1) Dada la siguiente definición:

```
(define (f p g h x)
    (if (p x)
        (g x)
        (h x)))
```

Escribe un ejemplo de código en Scheme en que se realice una invocación correcta a `f`.

a.2) Escribe una función `(make-saludo principio final)` que funcione como muestra el siguiente ejemplo:

```
(define f (make-saludo "Hola " ", me alegro de verte"))
(define g (make-saludo "¿Cómo estás " "?"))
(f "Pepe") ⇒ "Hola Pepe, me alegro de verte"
(g "Pepe") ⇒ "¿Cómo estás Pepe?"
```

a.3) Explica qué hace `g` y escribe una expresión en Scheme que llame a `g` y que devuelva 5.

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

b) Para cada una de las siguientes expresiones, rellena los huecos de forma que la última expresión sea correcta y escribe el resultado que devuelve Scheme:

b.1)

```
(define f _____)
(f) ⇒ 5
```

b.2)

```
(define g ______)
((g 10) 5) ⇒  ?
```

c) Cada una de las expresiones siguientes contiene una llamada a una función de orden superior. Indica qué valor devolverá:

c.1)

```
(define (doble pal)
   (string-append pal pal))

(map (lambda (x)
        (doble x)) '("hola" "adios" "hasta luego"))
```

c.2)

```
(filter (lambda (x)
           (> (car x) (cdr x))) '((3 . 12) (12 . 4) (10 . 3) (1 . 10)))
```

c.3)

```
(apply append '((1 2 3) (4 5) (6 7 8 9)))
```

Ejercicio 2

Define la función `(cumple-predicados lista-preds n)` sin utilizar funciones de orden superior. La función recibe una lista de predicados y un número y devuelve una lista con los resultados de aplicar cada predicado al número.

Ejemplo:

```
(cumple-predicados (list even? mayor-5? menor-10?) 6)
⇒ (#t #t #t)
```

Ejercicio 3

a) Implementa la función `(mayor-que num lista)` usando `map`

```
(mayor-que 10 '(1 23 4 -1 12 11 10)) ⇒ (#f #t #f #f #t #t #f)
```

b) En el último apartado del tema 3.1 se define la función `(divisores x)`. Escribe una nueva versión de la misma usando la función de orden superior `filter`.

```
(divisores 12) ⇒ (12 6 4 3 2 1)
```

c) Escribe la función recursiva de orden superior `(fold-with-last f lista)` que reciba una función de dos argumentos y una lista. La función de dos argumentos debe *plegar* la lista, al igual que `fold`, pero utilizará el último elemento de la lista como caso base. La lista tendrá como mínimo un elemento.

Ejemplo:

```
(define (suma x y) (+ x y))
(define (prueba x y) (* x y 2))

(fold-with-last suma '(1 2 3 4 5)) ⇒ 15
(fold-with-last prueba '(1 2 3 4 5)) ⇒ 1920
```

d) Implementa las siguientes funciones de ejercicios anteriores usando `fold-with-last`:

- `maximo`
- `interseccion-lista-intervalos`

Ejemplos:

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

Ejercicio 4

Escribe la función `(construye-sumadores lista)` que recibe una lista de números y devuelve una lista de procedimientos que implementan sumadores-k (funciones que reciben un número y devuelven el mismo número al que se le ha sumado otro número `k`). Cada procedimiento debe sumar el número `k` correspondiente de la lista original:

```
(define lista (construye-sumadores '(10 100 45 90)))
lista ⇒ (#<procedure> #<procedure> #<procedure> #<procedure>)
((caddr lista) 10) ⇒ 55
((list-ref lista 3) 10) ⇒ 100
```

Ejercicio 5

a) Supongamos el siguiente código en Scheme

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

1) Indica el resultado de la última expresión
2) Dibuja y explica el diagrama de ámbitos

3) ¿Se ha creado alguna clausura? Explícalo.

b) Supongamos el siguiente código en Scheme

```
(define suma (lambda (x) (+ x k)))
(define (make-sumador k)
    suma)
(define f (make-sumador 10))
(f 2)
```

1) Indica el resultado de la última expresión
2) Dibuja y explica el diagrama de ámbitos
3) ¿Se ha creado alguna clausura? Explícalo.

c) Supongamos el siguiente código en Scheme

```
(define x 1)
(define z 30)
(define g
   (let ((x 10)
         (y 20))
      (lambda (x)
              (+ x y z))))
(g 100)
```

1) Indica el resultado de la última expresión
2) Dibuja y explica el diagrama de ámbitos
3) ¿Se ha creado alguna clausura? Explícalo.
