David Guillermo Morales Sáez

1. Escribir una función que dadas dos listas de entrada, devuelva una tercera con la posición de cada elemento de la primera lista en la segunda, si está presente, y ***nil*** si no lo está (no usar la función ***position***). Programar dos versiones, una iterativa y otra recursiva, indicando ejemplos de ejecución.

```
(defun busca_pos_it (lista1 lista2 lista3)
        (let (
                (tam1 (length lista1))
                (tam2 (length lista2))
        )
        (dotimes (pos1 tam1)
                (dotimes (pos2 tam2)
                        (if (= (nth pos1 lista1) (nth pos2 lista2))
                                (setq lista3 (append lista3 (list pos2)))
                        )
                )
                (if (/= (length lista3) (+ 1 pos1))
                        (setq lista3 (append lista3 '(nil)))
                )
        )
        )
        lista3
)


(defun busca_pos_rec_aux (A B C n)
        (if (= (length B) 0)
                nil
                (if (= (first A) (first B))
                        n
                        (busca_pos_rec_aux A (nthcdr 1 B) C (+ n 1))
                )
         )
)

(defun bloq_func(lista1 lista2 lista3)
        (setq lista3 (append lista3 (cons (busca_pos_rec_aux lista1 lista2 lista3 0 ) nil)))
        (busca_pos_rec (nthcdr 1 lista1) lista2 lista3 )
)

(defun busca_pos_rec (lista1 lista2 lista3)
  (if (= (length lista1) 0) lista3
        (bloq_func lista1 lista2 lista3)
  )
)
```

2. Realizar una traza y estimar el tiempo de ejecución que consumen las funciones definidas en la pregunta anterior.

```
Timing the evaluation of (BUSCA_POS_IT LISTA1 LISTA2 LISTA3)
0 BUSCA_POS_IT > ...
  >> LISTA1 : (2 3 1 5)
  >> LISTA2 : (1 2 3 4)
  >> LISTA3 : NIL
0 BUSCA_POS_IT < ...
  << VALUE-0 : (1 2 0 NIL)

User time    =        0.000
System time  =        0.000
Elapsed time =        0.001
```

```
Allocation   = 3768 bytes
6 Page faults

Timing the evaluation of (BUSCA_POS_REC LISTA1 LISTA2 LISTA3)
0 BUSCA_POS_REC > ...
  >> LISTA1 : (2 3 1 5)
  >> LISTA2 : (1 2 3 4)
  >> LISTA3 : NIL
  1 BUSCA_POS_REC > ...
    >> LISTA1 : (3 1 5)
    >> LISTA2 : (1 2 3 4)
    >> LISTA3 : (1)
    2 BUSCA_POS_REC > ...
      >> LISTA1 : (1 5)
      >> LISTA2 : (1 2 3 4)
      >> LISTA3 : (1 2)
      3 BUSCA_POS_REC > ...
        >> LISTA1 : (5)
        >> LISTA2 : (1 2 3 4)
        >> LISTA3 : (1 2 0)
        4 BUSCA_POS_REC > ...
          >> LISTA1 : NIL
          >> LISTA2 : (1 2 3 4)
          >> LISTA3 : (1 2 0 NIL)
        4 BUSCA_POS_REC < ...
          << VALUE-0 : (1 2 0 NIL)
      3 BUSCA_POS_REC < ...
        << VALUE-0 : (1 2 0 NIL)
    2 BUSCA_POS_REC < ...
      << VALUE-0 : (1 2 0 NIL)
  1 BUSCA_POS_REC < ...
    << VALUE-0 : (1 2 0 NIL)
0 BUSCA_POS_REC < ...
  << VALUE-0 : (1 2 0 NIL)

User time    =        0.002
System time  =        0.000
Elapsed time =        0.002
Allocation   = 24060 bytes
0 Page faults
```

3. Escribir una función recursiva que devuelva una lista con elementos replicados. A continuación se incluye un ejemplo:

```
>(replicar '(a b c) 3) (A A A B B B C C C)
```

```
(defun replica_letra (letra n)
  (if (> n 1)
          (setq out2 (append (list letra) (replica_letra letra (- n 1))))
          (setq out2 (list letra))
  )
  out2
)

(defun replicar (lista n)
  (setq out nil)
  (if (> (length lista) 0)
        (setq out1 (replicar (nthcdr 1 lista) n))
        (setq out1 '())
  )
  (if (> (length lista) 0)
        (setq out (append (replica_letra (nth 0 lista) n) out1))
        NIL
```

```
  )
  out
)
```

4. Programar, mediante recursividad terminal y no terminal, una función para cada caso que dados dos valores positivos ***x*** e ***y*** devuelva una lista con la serie resultante de restar sucesivamente el segundo del primero, hasta que la resta se haga negativa. Por ejemplo, para 7 y 3 debe devolver la lista (7 4 1).

```
(defun prof_no_terminal(a b c)
        (if (> a b)
          (setq c (append (list a) (prof_no_terminal (- a b) b c)))
          (setq c (list a))
        )
        c
)

(defun prof_terminal(a b c)
        (setq c (append c (list a)))
        (if (> a b)
                (setq c (prof_terminal (- a b) b c))
                NIL
        )
        c
)
```

5. Pon un ejemplo de uso de los argumentos extendidos diferente al presentado en clase.

```
(defun foo (a b &optional c d)
          (list a b c d)
)

CL-USER 10 : 2 > (foo 1 2)
(1 2 NIL NIL)

CL-USER 11 : 2 > (foo 1 2 3)
(1 2 3 NIL)

CL-USER 12 : 2 > (foo 1 2 3 4)
(1 2 3 4)
```