```lisp
;1
(defun rotar-izda (lista)
    (append (cdr lista) (list (first lista)))
)

;2
(defun primer-num (lista)
    (cond ((null lista) (print "lista vacia"))
        ((numberp (first lista)) (first lista))
        (t (primer-num (cdr lista)))
    )
)

;3
(defun cambiar (el1 el2 lista)
    (cond ((null lista) nil)
        ((equal el1 (first lista)) (cons el2 (cambiar el1 el2 (cdr lista))))
        t (cons (first lista) (cambiar el1 el2 (cdr lista))))
    )
)

;4
(defun lista-mayores (elem lista &optional resul)
    (cond ((null lista) resul)
        ((> (first lista) elem ) (lista-mayores elem (cdr lista)
            (append resul (list (first lista)))))
        (t (lista-mayores elem (cdr lista) resul))
    )
)
;Otra sol:
(defun lista-mayores (elem lista)
    (cond ((null lista) nil)
        ((> (car lista) elem) (cons (car lista) (lista-mayores elem (cdr lista))))
        (t (lista-mayores elem (cdr lista)))
    )
)
;Otra sol:
(setf elem 4)
(setf lista '(7 2 3 6 1))
(mapcar #'(lambda (x) (if (> x elem) x)) lista)

;5
(defun insertar (elem lista)
    (cond ((null lista) (list elem))
        ((> elem  (car lista)) (cons (car lista) (insertar elem (cdr lista))))
        (t (cons elem lista))
    )
)
```