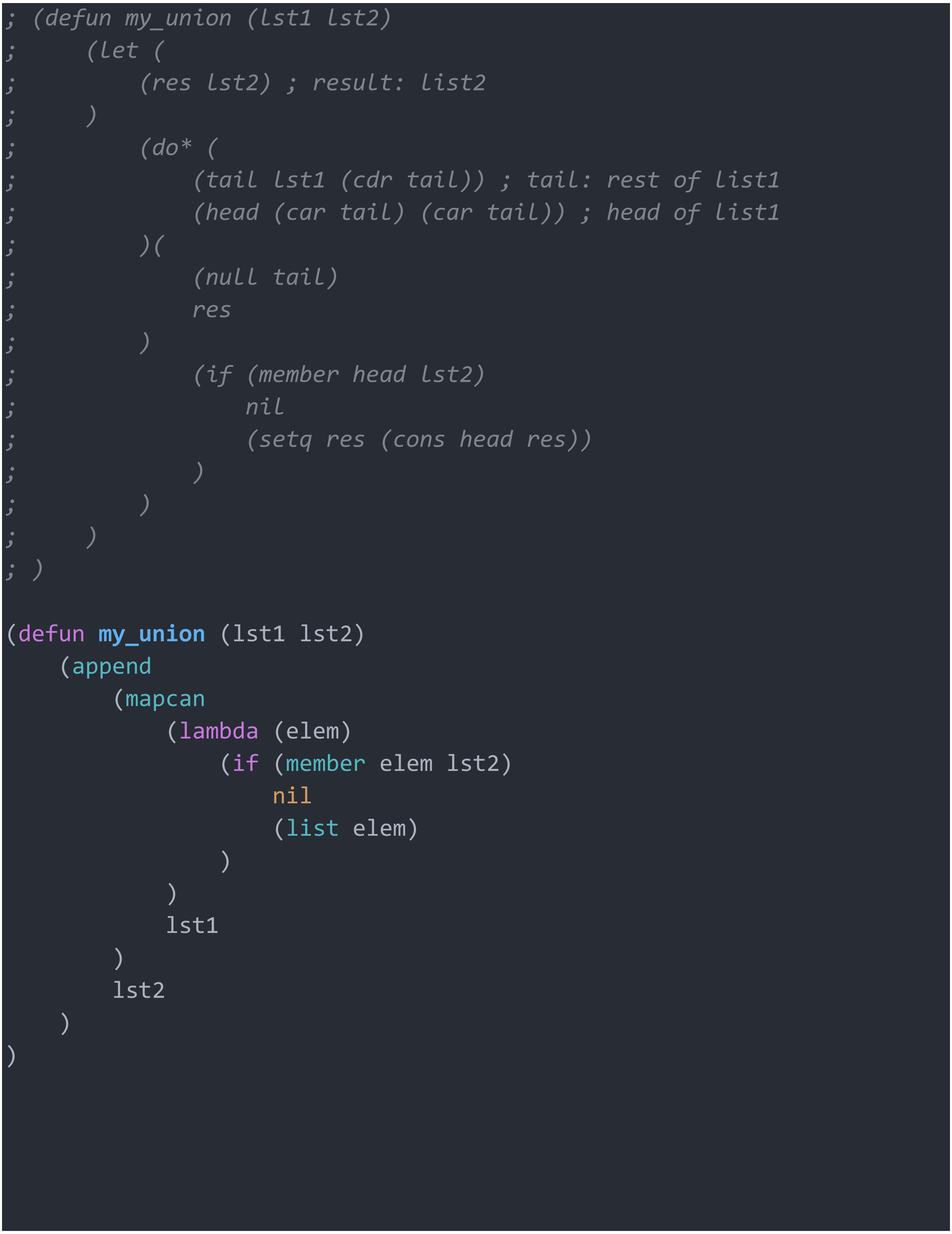
LAB 5

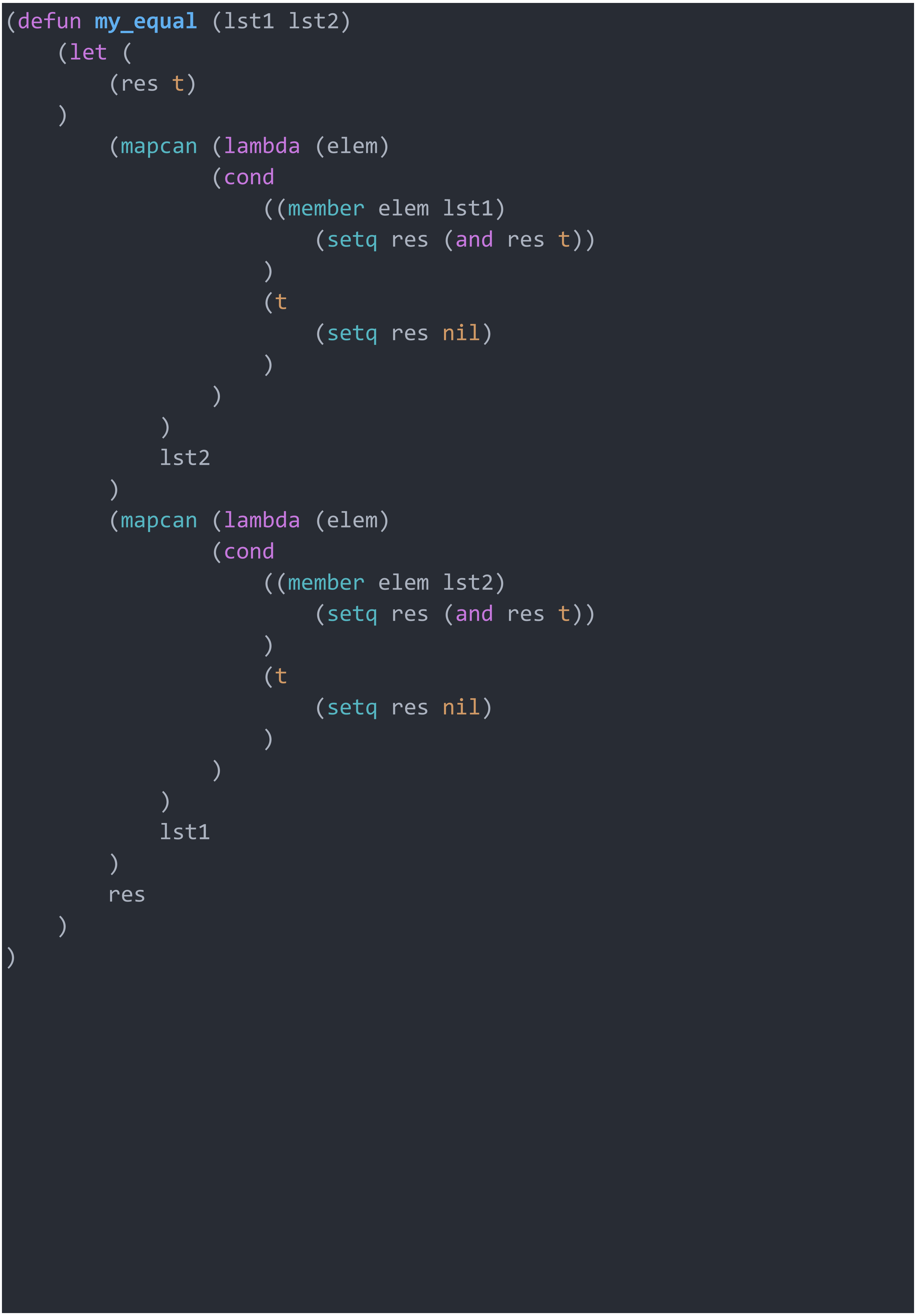
Problem 1



```
; (defun my_union (lst1 lst2)
;     (let (
;         (res lst2) ; result: list2
;     )
;         (do* (
;             (tail lst1 (cdr tail)) ; tail: rest of list1
;             (head (car tail) (car tail)) ; head of list1
;         )(
;             (null tail)
;             res
;         )
;             (if (member head lst2)
;                 nil
;                 (setq res (cons head res))
;             )
;         )
;     )
; )

(defun my_union (lst1 lst2)
    (append
        (mapcan
            (lambda (elem)
                (if (member elem lst2)
                    nil
                    (list elem)
                )
            )
            lst1
        )
        lst2
    )
)
```

```lisp
(defun my_intersection (lst1 lst2)
    (mapcan
        (lambda (elem)
            (if (member elem lst2)
                (list elem)
                nil
            )
        )
        lst1
    )
)

(defun my_difference (lst1 lst2)
    (defun diff (lst1 lst2)
        (mapcan
            (lambda (elem)
                (if (member elem lst1)
                    nil
                    (list elem)
                )
            )
            lst2
        )
    )

    (append
        (diff lst2 lst1)
        (diff lst1 lst2)
    )
)
```

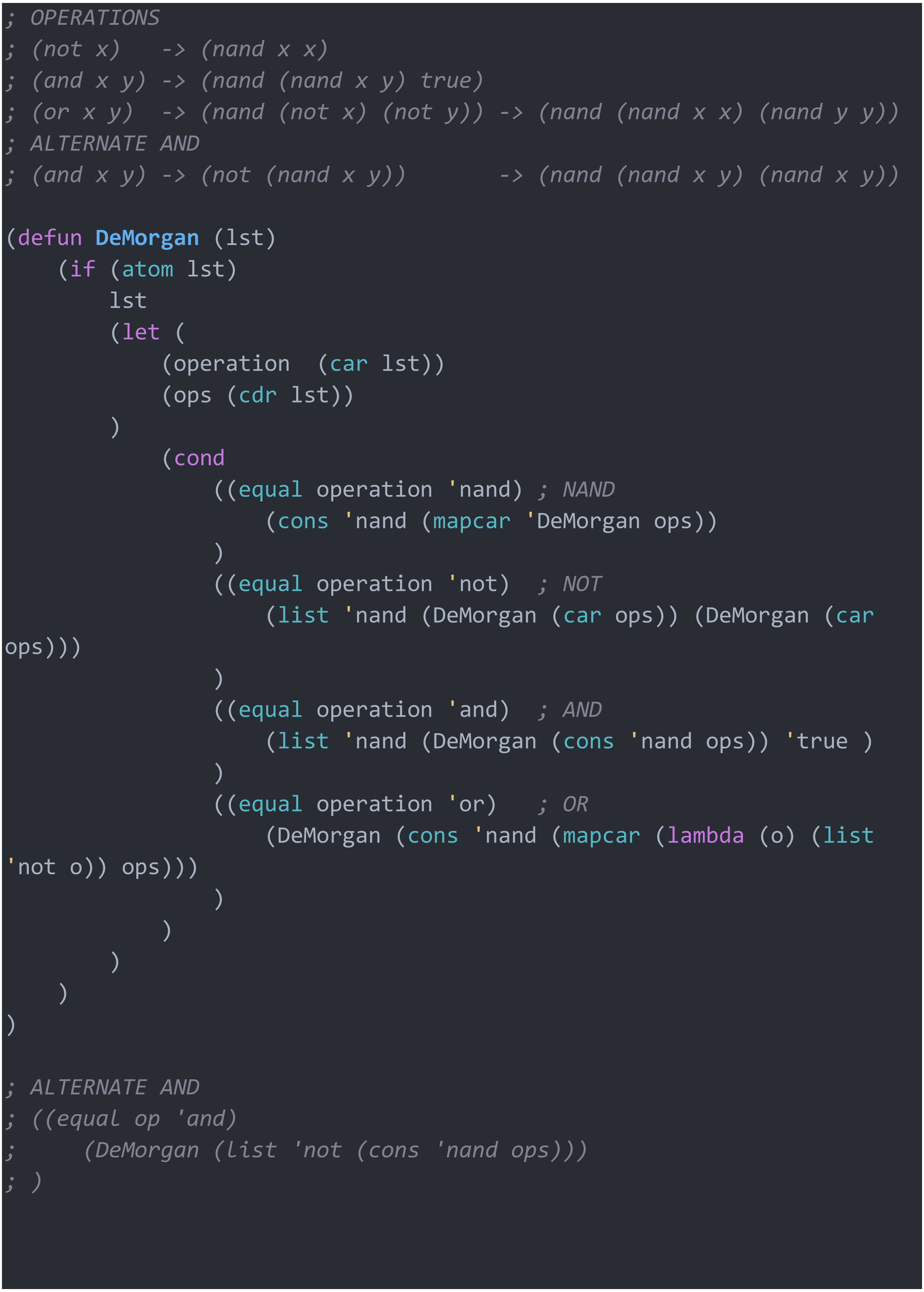
```
; (defun my_equal (lst1 lst2)
;     (if (= (length lst1) (length lst2))
;         (do* (
;             (tail lst1 (cdr tail))
;             (head (car tail) (car tail))
;             (res t)
;         )(
;             (or (null res) (null tail))
;             res
;         )
;             (if (member head lst2)
;                 nil
;                 (setq res nil)
;             )
;         )
;     )
; )
```



```
(defun my_equal (lst1 lst2)
    (let (
        (res t)
    )
        (mapcan (lambda (elem)
                (cond
                    ((member elem lst1)
                        (setq res (and res t))
                    )
                    (t
                        (setq res nil)
                    )
                )
            )
            lst2
        )
        (mapcan (lambda (elem)
                (cond
                    ((member elem lst2)
                        (setq res (and res t))
                    )
                    (t
                        (setq res nil)
                    )
                )
            )
            lst1
        )
        res
    )
)
```

```
(print
    (my_union '(1 2 3 4 5) '(4 5 6 7 8))
)

(print
    (my_intersection '(1 2 3 4 5) '(4 5 6 7 8))
)

(print
    (my_intersection '(1 2 3) '(4 5 6))
)

(print
    (my_difference '(1 2 3 4 5) '(4 5 6 7 8))
)

(print
    (my_difference '(1 2 3) '(4 5 6))
)

(print
    (my_equal '(1 a b 3 4 5) '(1 a b 3 4 5))
)

(print
    (my_equal '(3 2 1) '(1 2 3))
)

(print
    (my_equal '(1 2 3 4 5) '(4 5 6 7 8))
)
```

Problem 2



```
; OPERATIONS
; (not x)   -> (nand x x)
; (and x y) -> (nand (nand x y) true)
; (or x y)  -> (nand (not x) (not y)) -> (nand (nand x x) (nand y y))
; ALTERNATE AND
; (and x y) -> (not (nand x y))       -> (nand (nand x y) (nand x y))

(defun DeMorgan (lst)
    (if (atom lst)
        lst
        (let (
            (operation  (car lst))
            (ops (cdr lst))
        )
            (cond
                ((equal operation 'nand) ; NAND
                    (cons 'nand (mapcar 'DeMorgan ops))
                )
                ((equal operation 'not)  ; NOT
                    (list 'nand (DeMorgan (car ops)) (DeMorgan (car ops)))
                )
                ((equal operation 'and)  ; AND
                    (list 'nand (DeMorgan (cons 'nand ops)) 'true )
                )
                ((equal operation 'or)   ; OR
                    (DeMorgan (cons 'nand (mapcar (lambda (o) (list 'not o)) ops)))
                )
            )
        )
    )
)

; ALTERNATE AND
; ((equal op 'and)
;     (DeMorgan (list 'not (cons 'nand ops)))
; )
```

```
(print (DeMorgan '(and a (not b)) ))
(print (DeMorgan '(or a b c) ))
(print (DeMorgan '(and a (or c d) (not e)) ))
```

Problem 3

```
(defun count_atom (elem nums)
    (cond
        ((listp nums)
            (apply '+
                (mapcar
                    (lambda (lst) (count_atom elem lst))
                    nums
                )
            )
        )
        ((equal elem nums)
            1
        )
        (t
            0
        )
    )
)

(print
    (count_atom 3 '(2 2 3 (4 2 4 (3) 3) 4))
)
```