Lab 5

Problem 1

- Union:

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

- Test Cases:

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

- **Intersection:**

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

- **Test Cases:**

```lisp
(print
    (my_intersection '(1 2 3 4 5) '(4 nil 5 6 7 8))
)
; 4 5

(print
    (my_intersection nil '(4 5 6))
)
; nil
```

- Difference:

```
(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)
    )
)
```

- Test Cases:

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

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

- Equal:

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

- Test Cases:

```lisp
(print
    (my_equal '(3 2 1) '(1 2 3))
)
; T

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

Problem 2
- DeMorgan:

```
(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)))
; )
```

- Test Cases:

```
(print (DeMorgan '(and a (not b)) ))
; (NAND (NAND A (NAND B B)) TRUE)

(print (DeMorgan '(or a b c) ))
; (NAND (NAND A A) (NAND B B) (NAND C C))

(print (DeMorgan '(and a (or c d) (not e)) ))
; (NAND (NAND A (NAND (NAND C C) (NAND D D)) (NAND E E)) TRUE)
```

Problem 3

- Count Atom:

```
(defun count_atom (elem nums)
    (if (atom nums)
        (if (eq elem nums)
            1
            0
        )
        (apply '+
            (mapcar
                (lambda (lst) (count_atom elem lst))
                nums
            )
        )
    )
)
```

- Test Cases:

```
(print
    (count_atom nil '(2 b () a (4 nil nil c v a (a) 3) a))
)
; 3
```