9. Write a function that removes all occurrences of an atom from any level of a list.

;clean-nil(nil) = nil, if list is empty
;clean-nil(nil :: T) = clean-nil(T), if first element is nil
;clean-nil(H :: T) = H :: clean-nil(T), if first element is not null

```
(defun clean-nil (l)
  (cond
    ((null l) nil)
    ((null (car l)) (clean-nil (cdr l)))
    (t (cons (car l) (clean-nil (cdr l))))))
```

;l – a possibly nested list (can contain atoms and sublists)
;a – the atom that must be removed from all levels of the list
;The function returns a list with the same structure as l, but without any occurrences of
;atom a.
;remove-atom(nil, a) = nil, if list is empty
;remove-atom(l, a) =
;   nil   , if l = a
;   l   , otherwise
;remove-atom(l, a) =
;   clean-nil(mapcar(λx. remove-atom(x, a), l))

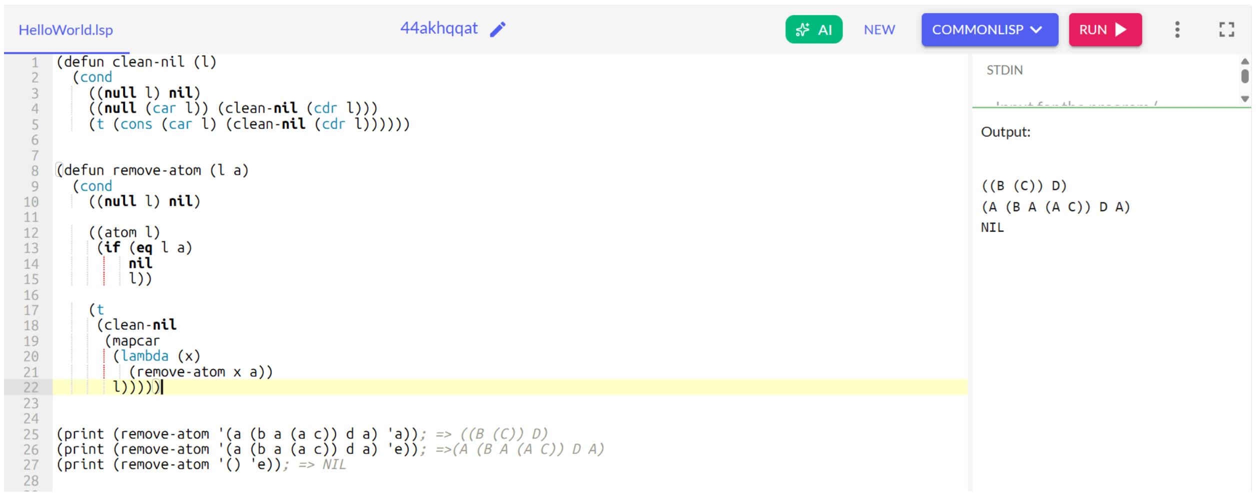
```
(defun remove-atom (l a)
  (cond
    ((null l) nil)

    ((atom l)
     (if (eq l a)
         nil
         l))

    (t
     (clean-nil
      (mapcar
       (lambda (x)
         (remove-atom x a))
```

l)))))

(print (remove-atom '(a (b a (a c)) d a) 'a)); => ((B (C)) D)
(print (remove-atom '(a (b a (a c)) d a) 'e)); =>(A (B A (A C)) D A)
(print (remove-atom '() 'e)); => NIL



```lisp
(defun clean-nil (l)
  (cond
    ((null l) nil)
    ((null (car l)) (clean-nil (cdr l)))
    (t (cons (car l) (clean-nil (cdr l))))))


(defun remove-atom (l a)
  (cond
    ((null l) nil)

    ((atom l)
     (if (eq l a)
         nil
         l))

    (t
     (clean-nil
      (mapcar
       (lambda (x)
         (remove-atom x a))
       l)))))


(print (remove-atom '(a (b a (a c)) d a) 'a)); => ((B (C)) D)
(print (remove-atom '(a (b a (a c)) d a) 'e)); =>(A (B A (A C)) D A)
(print (remove-atom '() 'e)); => NIL
```

```
((B (C)) D)
(A (B A (A C)) D A)
NIL
```