another function working with trees

---

brief

investigation into writing a function which checks if an item is present in a tree (in a nested list)

version 1

a typical structure of a function which searches through a tree. This works ok according to our aims.

```
(defun in-tree? (item tree)
  (cond ((null tree)
         nil
         )
        ((eq item (first tree))
         t
         )
        ((listp (first tree))
         (or (in-tree? item (first tree))
             (in-tree? item (rest tree)))
         )
        (t
         (in-tree? item (rest tree))
         )))


cg-user(12): (in-tree? 'x '(a (b c x) (d e f) g))
t
cg-user(13): (in-tree? 'y '(a (b c x) (d e f) g))
nil
cg-user(14): (in-tree? 'x nil)
nil
cg-user(15): (in-tree? nil '(a (b c x) (d e f) g))
nil
cg-user(16): (in-tree? 'spud 'spud)
Error: Attempt to take the car of spud which is not listp.
```

version 2

a function which allows trees to be single symbols – this works mostly ok except that it always finds **nil** in trees (even when **nil** is not a leaf node)

```
(defun in-tree? (item tree)
  (if (atom tree)
      (eq item tree)
    (or (in-tree? item (first tree))
        (in-tree? item (rest tree)))
    ))


cg-user(17): (in-tree? 'x '(a (b c x) (d e f) g))
t
cg-user(18): (in-tree? 'y '(a (b c x) (d e f) g))
nil
cg-user(19): (in-tree? 'x nil)
nil
cg-user(20): (in-tree? nil '(a (b c x) (d e f) g))
t
cg-user(21): (in-tree? 'spud 'spud)
t
```

version 3

a version which uses and & or in place of the **if**. This does not quite work.

Your task… understand why this does not work properly & try to correct it. What are the problems? which of the three versions do you prefer?

```
(defun in-tree? (item tree)
  (and tree
       (or (eq item tree)
           (in-tree? item (first tree))
           (in-tree? item (rest tree)))
    ))


cg-user(24): (in-tree? 'x nil)
nil
cg-user(25): (in-tree? 'x '(x a b c))
t
cg-user(26): (in-tree? 'x '(((x a) b c)))
t
cg-user(27): (in-tree? 'x '(a x b c))
Error: Attempt to take the car of a which is not listp.
```

a task

write a function to sum all of the numbers in a tree – you can assume the tree will only contain numbers, eg…

```
cg-user(31): (sumt nil)
0
cg-user(32): (sumt '((((1 2) 3)(4 5)(((6))))7 (8 9)))
45
```