

## Ansi Common Lisp

### Chapter 2 exercise Answers

1. a) 

```
> ( + ( - 5 1 ) ( + 3 7 ) )
```

14  
b) 

```
(list 1 (+ 2 3))
```

( 1 5)  
c) 

```
> (if (listp 1 ) ( + 2 3 ))
```

5 *because (listp 1 ) returns nil.*  
d) 

```
> (list (and (listp 3 ) t) ( + 1 2 ))
```

(nil 3)
2. i. 

```
(cons 'a (cons 'b (cons 'c nil)))
```

  
ii. 

```
(cons 'a (list 'b 'c))
```

  
iii. 

```
(list 'a 'b 'c)
```
3. 

```
(defun fourth-element (lst)
  (cdr (cdr (cdr (cdr lst)))))
```
4. 

```
(defun fourth-element (lst)
  (car (cdr (cdr (cdr lst)))))
```
5. a) Returns true if any element of a list, is an empty list (nil).  
b) Returns the index of the first occurrence of a number in a list of numbers.
6. a) 

```
> (car (x (cdr '(a (b c) d))))
```

B  
x <-- car  
cdr returns the tail ( ( b c ) d)  
car returns (B)  
the second car returns B.  
  
b) 

```
> (x 13 (/ 1 0))
```

13  
x <-- or  
The expression gets short-circuited, and (/ 1 0) does not get evaluated.  
c) 

```
> (x #'list 1 nil)
```

(1)  
x <-- apply
7. 

```
(defun is-any-element-a-list (x)
  (if (null x)
      nil
      (if (listp (car x))
          t
          (is-any-element-a-list (cdr x)))))
```

8. (a) Recursive version

```
(defun print-dots(n)
  (if (null n)
      0
      ( if (= n 0)
          'done
          ( progn
              (format t ".")
              (print-dots (- n 1)))))))
```

Iterative Version

```
(defun print-dots (n )
  ( if (null n)
      nil
      ( do ( ( i 0 ( + i 1 )))
            ( ( > i n ) 'done )
            ( format t "." )))
```

(b) Recursive Version

```
(defun count-elements (lst x)
  ( if (not (listp lst))
      nil
      ( if (null lst)
          0
          ( if (eql x (car lst))
              ( + 1 (count-elements (cdr lst) x))
              ( count-elements (cdr lst) x))))))
```

Iterative Version

```
(defun count-elements-iter (lst x)
  ( if ( not (listp lst))
      nil
      (if (null lst)
          0
          ( let ( ( occur 0 ) )
              (dolist (e lst)
                ( if (eql x e)
                    (setf occur ( + occur 1))))
              occur ))))
```

9. a) Correct Version

*The problem is that (setf ...) is not called to remove nil and then resetting.*

*The function is a summation of the list lst to the one that removed '()*

```
(defun summit (lst)
  (setf lst (remove nil lst))
  (apply #' + lst))
```

b) *There is no base case to terminate the recursion*

```
(defun summit (lst)
  (if (null lst)
      0
      (if (null (car lst))
          (summit (cdr lst))
          (+ (car lst) (summit (cdr lst)))))))
```