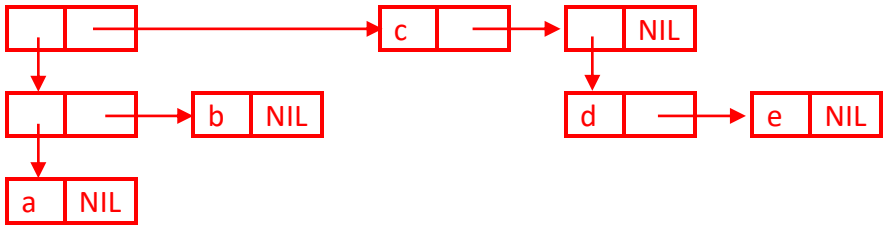
SOLUTIONS - EXERCISES ON SCHEME

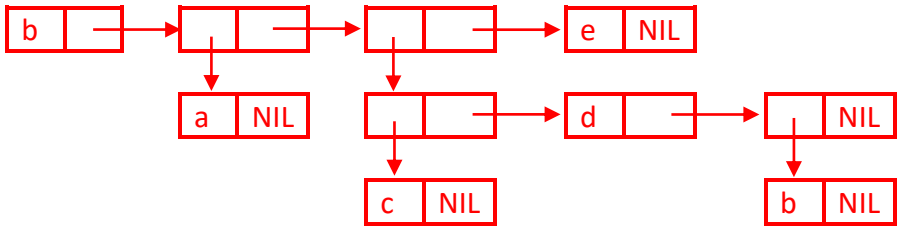
1. Compute the values of the following expressions.

Expression	Value
(car ‘(6 3 2))	6
(car (6 3 2))	error (6 3 2) has no value
(let ((L ‘(6 3 2))) (car L))	6
(cdr ‘((6 3) 2 (7 6) 3))	(2 (7 6) 3)
(car ‘((6 3) 2 (7 6) 3))	(6 3)
(cons 6 ‘(3 2 8))	(6 3 2 8)
(cons ‘(5 6) ‘(3 2 8))	((5 6) 3 2 8)

2. Draw box diagrams (i.e. graphical representation) for the following Scheme lists:
   a. (((a) b) c (d e))



   b. (b(a ( )) ((c) d (b)) e)



3. Write a function that returns the nth element of a list.
   examples: (nth 2 ‘(a b (c d))) returns b
   (nth 3 ‘(a b (c d))) returns (c d)

```
(define  ( nth  n  L )
         ( cond ( ( null?  L )  0 )
                ( ( =  n  1 )  ( car  L ) )
                ( else  ( nth  ( -  n  1 ) ( cdr  L ) ) ) ) )
```

4. Write a function that count the number of occurrences of atoms in a list of atoms.
   example (countatom '(a b c d e)) is 5

```
(define ( countatom  L ) ( if    ((null?  L )  0 )
                                  ( + 1 ( countatom ( cdr L ) ) ) ) )
```

5. Write a function that counts the number of occurrences of atoms at the "top" level in an arbitrary list.
   example (counttopatom '(a (b c) d a)) returns 3

```
( define  ( counttopatom  L )
            ( cond    ( ( null?   L ) 0 )
                      ( ( atom?  ( car   L ) )   ( + 1   ( counttopatom   ( cdr L ) ) ) )
                      ( else      ( counttopatom   ( cdr   L ) ) ) ) )
```

6. Write a Scheme function with two parameters, an atom and a list, that returns the list with all occurrences, no matter how deep, of the given atom deleted. The returned list cannot contain anything in place of the deleted atom. For example:
   (removeatom a '(a (b c (a) a) (a b)))
   returns ((b c ( )) (b))

```
(define (remove_atom x L)
  (cond  ( (null? L ) '( ) )
      ( (atom? (car L) ) (if (eq? (car L) x) (cons (remove_atom x (cdr L) ) null )
                             (cons (car L) (remove_atom x (cdr L) ) ) ) )
      ( else (append (remove_atom x (car L) ) (remove_atom x (cdr L) ) ) ) ) )
```

7. The Scheme function reverse described in the notes reverses only the "top level" of a list: if L = ((2 3) 4 (5 6)) then (reverse 'L) = ((5 6) 4 (2 3)). Write a Scheme function deep-reverse that also reverses all sublists (deep-reverse L) - ((6 5) 4 (3 2)).

```
( define  ( deep-reverse L )
   ( cond ( ( null?  L ) '(  ) )
          ( ( pair?  ( car L ) )  ( append   ( deep-reverse ( cdr L ) )
             ( list ( deep-reverse ( car L ) ) )   ) )
          ( else ( append  ( deep-reverse  ( cdr  L ) )  ( list  ( car  L ) ) ) )   ))
```

8. Write a Scheme function with three parameters, two atoms and a list, that returns the list with all occurrences, no matter how deep, of the first atom replaced by the second atom. For example:
   (replaceatom 'a 'x '(a (b c (a) a) (a b)))
   returns (x (b c (x) x) (x b)))

```
(define ( replaceatom   x  y  L )
    (cond     ( ( null?   L)   '(  ) )
              ( ( list?  ( car  L ) )  ( cons  ( replaceatom x y ( car  L ) )   ( replaceatom x y
               (cdr   L)  )  )  )
              ( ( eq? ( car L )  x )  ( cons  y   ( replaceatom   x y  (cdr   L)  )   )    )
              ( else ( cons ( car  L )   ( replaceatom x y   (cdr   L)  )   )    )   }   }
```

9. What will the following return?

```
( map  fact     '( 2 3 4 ) )
```

```
( 2 6 24 )
```

10. What will the following return?

```
( map  ( lambda       ( x )    ( -       0       x ) )          '( -1 2 3 -4 5 ) )
```

```
( 1  -2  -3  4  -5  )
```

11. Define the following function:

```
( define        ( is-negative?         x )      ( <      x        0 ) )
```

What will the following return?

```
( map  is-negative?          '( -1 2 3 -4 5 ) )
```

```
( #T #F #F #T #F )
```

12. Write a function remove-if that will remove elements of a list which meet the conditions expressed in a predicate f (i.e. remove the elements for which the function f is true). Examples:

```
( remove-if     even? '( 7 8 12 13 15 ) ) will return ( 7 13 15 )
```

```
( define        ( remove-if    f        L )
                        ( cond (  ( null?       L )  '( ) )
                                  ( ( f  ( car    L ) )  ( remove-if  f  ( cdr  L )  )  )
                                  ( else  ( cons  ( car  L )
                                                  ( remove-if  f  ( cdr  L ) )  ) ) )  )
```

13. What will the following return?

```
( remove-if    ( lambda       ( x )  ( <       x        0 ) )    '( -1  2 3  -4 5 ) )
```

```
( 2 3 5 )
```

14. Write a tail recursive version of remove-if.

```
(define  ( remove-if-aux  f  L  list-so-far )
                ( cond      (  ( null?  L )  list-so-far )
                           ( ( f  ( car  L ) )  ( remove-if-aux  f  ( cdr  L )  list-so-far ) )
                           (  else  ( remove-if-aux  f  ( cdr  L )
                                                     ( append  list-so-far
                                                               ( list  ( car  L ) ) ) ) ) ) ) )
```

15. Define a function add-n with an integer n as a parameter which returns a function which will add n to its parameter.
    Examples:
        ((add-n 5) 6) will return 11

```
( define ( add-n n )
    ( lambda ( y ) (+ n y ) ) )
```

16. Use the function you have just defined to define a function add-6 which will add 6 to its parameter.

```
( define add-6
    ( add-n 6 ) )
```

17. What would be displayed by (display (add-6 8))

    14

18. Using remBuilder define a function remAtom which will remove the top atoms from a list.

```
( define remAtom ( remBuilder ( lambda ( a ) ( not ( list? a ) ) ) ) )
```

Check these against notes:

19. Define a function add-n with an integer n as a parameter which returns a function which will add n to its parameter.
    Examples:
        ((add-n 5) 6) will return 11

```
( define ( add-n n )
    ( lambda ( y ) (+ n y ) ) )
```

20. Use the function you have just defined to define a function add-6 which will add 6 to its parameter.

```
( define add-6
    ( add-n 6 ) )
```

21. What would be displayed by (display (add-6 8))

    14

22. Using remBuilder define a function remAtom which will remove the top atoms from a list.

```
( define remAtom ( remBuilder ( lambda ( a ) ( not ( list? a ) ) ) ) )
```