**Question 1:** Write a Scheme function named pair-up that constructs an association list from a list of keys and a corresponding list of values.

**File name:**  q1\_haque\_faaiz\_ul.scm

**Code:**

(define pair-up(lambda (list1 list2)

(cond

[(eqv? list2 '()) list1]

[else (cons (cons (car list1) (car list2) ) (pair-up (cdr list1) (cdr list2) ) ) ] ) ) )

**Explanation:**

Since the question states we can assume all inputs are valid with the two lists having the same size, we do not have to check these conditions when implementing our function. The function works by taking two lists as input parameters and recursively outputting each element in the first list with the same index value of the corresponding element in the second list. The way this is implemented is using the cond condition, where we have two conditions the first one is our base case for the recursive solution, in which we return a list when one list is empty. In this case if list1 is empty, list2 is also empty since they have the same sizes, so therefore we can check any of the two lists and return any of them as well. This is done using the eqv? operator. If the lists are not empty, we return the cons of the first element in both lists, which results in (a . b) where a is the first element in the first list and b is the first element in the second list. And this is also put in another list. So there is improper lists inside of one proper list. We then recursively call the rest of each list using the cdr function.

**Sample Executions:**

1)(pair-up ’(a b c) (list 1 2 3))

* ((a . 1) (b . 2) (c . 3))

2)(pair-up '() '() )

* ()

3) (pair-up '(1 2 3) '(a b c))

* ((1.a) (2.b) (3.c))

4) (pair-up '(x) '(x))

* ((x . x))

5) (pair-up '(a x 3 1 z 4 0 c) '(3 2 x x 2 0 1 b))

* ((a . 3) (x . 2) (3 . x) (1 . x) (z . 2) (4 . 0) (0 . 1) (c . b))

6) Although the input condition is the lists must have the same size, we can try some inputs with different sizes and comment on the results

(pair-up '(a b c d) (list 1 2 3))

* ((a . 1) (b . 2) (c . 3) d)

Since there is 4 elements in the first list and 3 in the second, we can see that the last element in the resulting list just returned ‘d’ as a single atom.

**Question 2:** Write a Scheme function deep-remove with two parameters, an atom and a list, that returns a list identical to the parameter list except with all occurrences, no matter how deep, of the given atom deleted. The returned list cannot contain anything in place of the deleted atoms.

**File name:**  q2\_haque\_faaiz\_ul.scm

**Code:**

(define (deep-remove var1 list1)

(cond [(null? list1) '()]

[(list? (car list1)) (cons (deep-remove var1 (car list1)) (deep-remove var1 (cdr list1)))]

[(eqv? var1 (car list1)) (deep-remove var1 (cdr list1))]

[else (cons (car list1) (deep-remove var1 (cdr list1)))]))

**Explanation:**

Since we must delete the occurrence of a single token in a list, wherever present, we have a deep remove of the implementation. We use the same recursion logic like in the first code. Our base case will be when list1 is null we shall return an empty list as well. This will occur when we are finished iterating through the list using cdr. We have a cond statement to represent each case. The second case checks if there is another list, i.e a list inside a list, if this is the case we recursively call the function twice on the first element of the list and the rest of the list and cons the result in order to retain the list structure of the original list. If the current value we are visiting is the token we want to remove we simply recursively call the function on the rest of the list and exclude that element, however if it is not equal we first cons that element then move on to the rest of the list. This process is repeated using recursion until the base case is satisfied.

**Sample Executions:**

1)(deep-remove ’a ’(x a (b a (g a g) c (a))))

* (x (b (g g) c ()))

2) (deep-remove ‘() '(a b c '(d e)))

* (a b c '(d e)))

3) (deep-remove 'x '(x x y '(d e x '(x y) x ) ) )

* (y ‘(d e ‘(y)))

4) (deep-remove 'x '())

* ()

5) (deep-remove 1 (list 1 2 3 4 '(2 1 3) 1 '(1 2 '(1 2 (1) ) 1 )))

* (2 3 4 (2 3) (2 '(2 ())))

6) (deep-remove 'a '(a a '(a a a) a '(a '(a '( a a) a) ) ) )

* ('() '('('())))

Comments: Works for both atoms and list of integers, and deep-removes in each case no matter how deep the list is.