(1) Escrever as funções:

(a) (apaga L X)

Dada uma lista L e um elemento X, retorna L sem X.

Se L não contem X, retorna uma cópia exata de L

> apaga ‘(a b c d a) ‘a)

(B C D)

Solução:

```
(define (remove_value  obj lista)
  (if (null? lista)
     '()
  (if (equal? obj (car lista))
      (remove_value  obj (cdr lista))
      (cons (car lista) (remove_value  obj (cdr lista)))))
)

(define (apaga lista a)
  (remove_value a lista)
)
```

4 Encontrar o n-ésimo elemento de uma lista.

```
> (nth 3 '(1 3 5 7 9))
        7
> (nth 0 '((1 3) 5 7 9 6))
        (1 3)
```

Solução:

```
(define (nth n lista)
  (list-ref lista n)
)
```

5 Contar o número de vezes que um certo elemento aparece em uma dada lista.

```
> (count 4 '(1 2 3 4 5 6 4))
        2
> (count '(1) '((1) 2 (1) (1 2) (1)))
        3
> (count 2 '(1 3 6 7 9 5 0))
        0
```

Solução:

```
(define (count n lista)
  (if (not(null? lista))
      (if (equal? (car lista) n)
        (+ 1 (count n (cdr lista)))
         (+ 0 (count n (cdr lista)))
      )
      0
  ))
```

Obs. Usa-se o equal, pois ele permite comparar se elementos de 2 listas são iguais

6 Remover de um lista todas as ocorrências de um dado valor.

Solução:

```
(define (remove_value  obj lista)
  (if (null? lista)
     '()
  (if (equal? obj (car lista))
      (remove_value  obj (cdr lista))
      (cons (car lista) (remove_value  obj (cdr lista)))))
)
```

7Remover o k-ésimo elemento de uma lista.

> (remove_kth 3 '(1 3 5 7 9))
(1 3 5 9)
> (remove_kth 0 '(1 3 5 7 9 6))
(3 5 7 9 6)
> (remove_kth 10 '(1 3 5 7 9 6))
(1 3 5 7 9 6)

Solução:

```
(define (remove_kth n lista)
  (if(and (> n 0) (< n (size lista)))
    (append (list (car lista)) (remove_kth (- n 1) (cdr lista)))
    (if (= n 0)
      (cdr lista)
      lista
    )
  )
)



(define (size lista)
  (if (null? (cdr lista))
    1
    (+ 1 (size (cdr lista)))
  )
)
```

8 Reverter uma lista.

> (reverse '(1 3 5 7 9))
(9 7 5 3 1)
> (reverse '((1 . 4) (3) () (1 2 3)))
((1 2 3) null (3) (1 . 4))

Solução:

```
(define (reverse lista)
  (if (null? (cdr lista))
    lista
    (append (reverse (cdr lista)) (list (car lista)))
  )
)
```

9 Dividir uma lista em duas partes, de modo que a primeira parte receba os N primeiros elementos e a segunda metade receba os elementos restantes.

> (split 3 '(1 3 5 7 9))
((1 3 5) (7 9))
> (split 7 '(1 3 5 7 9))
((1 3 5 7 9) null)
> (split 0 '(1 3 5 7 9))
(null (1 3 5 7 9))

Solução:

```
(define (split n lista)

    (if (and (< n (size lista)) (> n 0))

      (list (reverse (suffixx (- (size lista) n) (reverse lista)))
(suffixx n lista))
      (if (= n 0)
        (list '() lista)
        (if (>= n (size lista))
          (list lista '())
        )
      )
    )

)
(define (suffixx k lista)

    (if (> k 0)
      (suffixx (- k 1) (cdr lista))
      lista
    )

)
```

```
(define (reverse lista)
  (if (null? (cdr lista))
    lista
    (append (reverse (cdr lista)) (list (car lista)))
  )
)
(define (size lista)
  (if (null? (cdr lista))
    1
    (+ 1 (size (cdr lista)))
  )
)
```

10 Encontrar o menor elemento de uma lista formada apenas por números.

> (min '(3 5 1 7 9))
1
> (min null)
null

Solução:

```
(define (min lista)
  (if (null? lista)
    '()
     (if (null? (cdr lista))
      (car lista)
      (minValue (car lista) (min (cdr lista)) )
     )
  )

)

(define (minValue a b)
  (if (= a b)
    b
    (if (< a b)
      a
      b
    )

  )
)
```

11 Determinar a união de duas listas representando conjuntos (listas sem repetições de elementos). O resultado da união será um conjunto, logo também não terá elementos repetidos.

> (union '(1 2 5) '(7 2 9 11))
(1 2 5 7 9 11)
> (union '(1 2 3 4 5 6) '(4 5 6 7 8))
(1 2 3 4 5 6 7 8)

Solução:

```
(define (union listaA listaB)
  (if (null? listaA)
    listaB
    (union (cdr listaA) (addu (car listaA) listaB) )
  )
)
(define (addu a lista)
  (if (not( member? a lista ))
    (append lista (list a))
    lista
  )
)


(define (member? n lista)
  (if (not (null? lista))
    (if (equal? n (car lista))
      #t
      (member? n (cdr lista))
    )
    #f
  )
)
```

14 Ordenar uma lista formada apenas por números.

> (sort '(1 5 3 9 7))
(1 3 5 7 9)
> (sort '(5 1 7 3 7 9))
(1 3 5 7 7 9)

Solução:

```
(define (sort lista)
    (sort-ex (length lista) lista)
)

(define (sort-ex N lista)
    (cond ((= N 1) (sort-aux lista))
          (else (sort-ex (- N 1) (sort-aux lista)))))

(define (sort-aux lista)
  (if (null? (cdr lista))
    lista
    (swap lista)
  )
)

(define (swap lista)
    (if (< (car lista) (cadr lista))
        (cons (car lista) (sort-aux (cdr lista)))
        (cons (cadr lista) (sort-aux (cons (car lista) (cddr lista))))
    )
)
```

16 Dados um dinteiro k e uma seqüência (lista), encontrar o prefixo formado pelos k primeiros elementos da sequência.

```
> (prefix 3 '(1 2 3 4 5 6))
        (1 2 3)
> (prefix 4 '(() (1 2 3) (4 . 2) (3 . null) 6 7 9))
        (null (1 2 3) (4 . 2) (3))
> (prefix 0 '(1 2 3 4 5 6))
        null
```

Solução:

```
(define (suffixx k lista)

    (if (> k 0)
      (suffixx (- k 1) (cdr lista))
      lista
    )

)



(define (size lista)
  (if (null? (cdr lista))
    1
    (+ 1 (size (cdr lista)))
  )
)

(define (prefix n lista)
  (if (= n 0)
    '()
    (reverse (suffixx (- (size lista) n) (reverse lista)))
  )
)
```

17 Dados um inteiro k e uma seqüência (lista), encontrar o sufixo formado pelos k últimos elementos da seqüência.

Solução:

```
(define (suffixx k lista)

    (if (> k 0)
      (suffixx (- k 1) (cdr lista))
      lista
    )

)

(define (suffix k lista)
  (if (equal? k 0)
     '()
     (suffixx (- (size lista) k) lista)
  )

)

(define (size lista)
  (if (null? (cdr lista))
    1
    (+ 1 (size (cdr lista)))
  )
)
```

23 Dados dois números inteiros N1, N2, gerar uma lista contendo todos os inteiros do intervalo [N1,...,N2] em ordem crescente.

```
> (range 4 7)
        (4 5 6 7)
> (range 7 4)
        null
> (range 5 5)
        (5)
```

Solução:

```
(define (range n1 n2)
  (if (<= n1 n2)
    (append (list n1) (range (+ 1 n1) n2))
    '()
  )
)
```