FUNCTII RKT

1. `list-length`: Returnează numărul de elemente dintr-o listă de liste.

```
(define (list-length lst)
  (if (null? lst)
      0
      (+ (length (car lst)) (list-length (cdr lst)))))
```

2. `list-flatten`: Aplatizează o listă de liste într-o singură listă.

```
(define (list-flatten lst)
  (cond
    [(null? lst) '()]
    [(list? (car lst)) (append (list-flatten (car lst)) (list-flatten (cdr lst)))]
    [else (cons (car lst) (list-flatten (cdr lst)))]))
```

3. `list-reverse`: Inversează ordinea elementelor într-o listă de liste.

```
(define (list-reverse lst)
  (cond
    [(null? lst) '()]
    [(list? (car lst)) (append (list-reverse (cdr lst)) (list (list-reverse (car lst))))]
    [else (cons (car lst) (list-reverse (cdr lst)))]))
```

4. `list-append`: Concatenează două liste de liste.

```
(define (list-append lst1 lst2)
  (append lst1 lst2))
```

5. `list-member`: Verifică dacă un element se găsește într-o listă de liste.

```
(define (list-member item lst)
  (cond
    [(null? lst) #f]
    [(list? (car lst)) (or (list-member item (car lst)) (list-member item (cdr lst)))]
    [(equal? item (car lst)) #t]
    [else (list-member item (cdr lst))]))
```

6. `list-filter`: Filtrează elementele dintr-o listă de liste pe baza unui predicat.

```
(define (list-filter pred lst)
  (cond
    [(null? lst) '()]
    [(list? (car lst)) (cons (list-filter pred (car lst)) (list-filter pred (cdr lst)))]
    [(pred (car lst)) (cons (car lst) (list-filter pred (cdr lst)))]
    [else (list-filter pred (cdr lst))]))
```

7. `list-map`: Aplică o funcție pe fiecare element dintr-o listă de liste și returnează rezultatele.

```
(define (list-map func lst)
  (cond
    [(null? lst) '()]
    [(list? (car lst)) (cons (list-map func (car lst)) (list-map func (cdr lst)))]
    [else (cons (func (car lst)) (list-map func (cdr lst)))]))
```

8. `list-reduce`: Reducere stânga pe o listă de liste utilizând o funcție de agregare.

```
(define (list-reduce func lst)
  (if (null? lst)
      (error "Cannot reduce an empty list")
      (let loop ([result (car lst)] [rest (cdr lst)])
        (if (null? rest)
            result
            (loop (func result (car rest)) (cdr rest))))))
```

9. `list-count`: Numără de câte ori un element apare într-o listă de liste.

```
(define (list-count item lst)
  (cond
    [(null? lst) 0]
    [(list? (car lst)) (+ (list-count item (car lst)) (list-count item (cdr lst)))]
    [(equal? item (car lst)) (+ 1 (list-count item (cdr lst)))]
    [else (list-count item (cdr lst))]))
```

10. `list-replace`: Înlocuiește toate aparițiile unui element într-o listă de liste cu un alt element.

```
(define (list-replace old new lst)
```

```
  (cond
    [(null? lst) '()]
    [(list? (car lst)) (cons (list-replace old new (car lst)) (list-replace old new (cdr lst)))]
    [(equal? old (car lst)) (cons new (list-replace old new (cdr lst)))]
    [else (cons (car lst) (list-replace old new (cdr lst)))]))
```

11. `list-depth`: Calculează adâncimea maximă a unei liste de liste.

```
(define (list-depth lst)
  (cond
    [(null? lst) 0]
    [(list? (car lst)) (+ 1 (list-depth (car lst)))]
    [else (list-depth (cdr lst))]))
```

12. `list-sublist`: Extrage o sublistă dintr-o listă de liste bazată pe un interval dat.

```
(define (list-sublist start end lst)
  (cond
    [(null? lst) '()]
    [(and (>= start 0) (< end (length (car lst))))
     (cons (sublist (car lst) start end) (list-sublist start end (cdr lst)))]
    [else (list-sublist start end (cdr lst))]))
```

13. `list-rotate`: Rotire ciclică la stânga a elementelor dintr-o listă de liste.

```
(define (list-rotate lst)
  (cond
    [(null? lst) '()]
    [(list? (car lst)) (cons (list-rotate (cdr lst)) (list-rotate (car lst)))]
    [else (cons (car lst) (list-rotate (cdr lst)))]))
```

14. `list-interleave`: Interlevează două liste de liste.

```
(define (list-interleave lst1 lst2)
  (if (or (null? lst1) (null? lst2))
      '()
      (append (list (car lst1) (car lst2)) (list-interleave (cdr lst1) (cdr lst2))))))
```