ALTELE RACKET

```
(define (greater-sum L)
  (filter (λ (S)            ; extrag din lista L acele liste pentru care
           (>=
            (foldr + 0 S)  ; suma e mai mare
            (foldr * 1 S)  ; decat produsul
            ))
          L))

(define (greater-sum L)
  (let ((check? (λ (l)
                  (>= (foldr + 0 l) (foldr * 1 l)
                  ))))
    (filter (λ (p) (check? p)) L)
    ))

(define (count-occ L1 L2)
  (let ([count (λ (x)
                 (length (filter (λ (p) (equal? x p)) L2))
                 )]) ; functie ce determina numarul de aparitii ale unui numar in lista L2
    (map (λ (y) (cons y (count y))) L1) ; la final construiesc rezultatul adaugand la fiecare
    ))

(define (zip L1 L2)
  (map cons L1 L2))

(define (unzip L)
  (cons
   (foldr (λ(x acc) (cons (car x) acc)) '() L)
   (list (foldr (λ(x acc) (cons (cdr x) acc)) '() L))))


(define (divisors L1 L2)
  (let ((find-divs (λ(x) ;consider o functie anonima
```

```
            ;ce imi extrage toate elementele din L2 ce sunt divizori
            ;ai lui "x".
            (filter (λ(num) (= (modulo x num) 0)) L2)
            )))
(map (λ(y) (cons y (list (find-divs y)))) L1) ;adaug la fiecare element
;corespunzator din L1 lista formata cu divizorii sai din L2.
))
```