

Упражнения 3 и 4

Списъци: създаване и извличане на елементи. Дефиниране и използване на функции за обработка на списъци.

3.1. Конструирайте списъците по-долу като използвате функцията **cons**, необходимите символи и константи:

```
a) (1 (a b) 2)
> (cons 1 (cons (cons 'a (cons 'b '())) (cons 2 '())))

б) ((a) 1 2 (e r))
> (cons (cons 'a '()) (cons 1 (cons 2 (cons (cons 'e (cons 'r '())) '()))))

в) (1 2 (3 4) ((d f)))
> (cons 1 (cons 2 (cons (cons 3 (cons 4 '()))
                        (cons (cons (cons 'd (cons 'f '())) '()) '()))))

г) (((m)) 1)
д) ((1 (2)) x y)
е) (a (b (c) 1) ((2)))
ж) (1 2 (b (c) 1) (3))
```

3.2. Конструирайте списъците от зад. 1 като използвате функцията **list**.

```
a) > (list 1 (list 'a 'b) 2)
б) > (list (list 'a) 1 2 (list 'e 'r))
в) > (list 1 2 (list 3 4) (list (list 'd 'f)))
```

3.3. Напишете израз, който конструира списъка ((a) 2 ((1 2) b 1)). Можете да използвате само cons (или list), празен списък, константи и символи.

```
> (cons (cons 'a '())
        (cons 2 (cons (cons (cons 1 (cons 2 '())) (cons 'b (cons 1 '())) '()))))

> (list (list 'a) 2 (list (list 1 2) 'b 1))
```

3.4. Нека символът f е свързан със стойността (1 a (b (h))(d)), т.е. оценен е изразът (define f '(1 a (b (h))(d))). Като използвате car и cdr, напишете израз, който от стойността на f извлича символа:

```
- h
> (car (car (cdr (car (cdr (cdr f))))))
или
> (caadr (caddr f))

- b

- d
```

3.5. Напишете израз, който извлича 3 от израза ((a) 2 ((1 3) b 1)).

```
> (define x '((a) 2 ((1 3) b 1)))
> (cadar (caddr x))
```

3.6. Дефинирайте функция, преброяваща колко елемента на даден списък са цели числа.

```
(define (intnum x)
  (cond ((null? x) 0)
        ((integer? (car x)) (+ 1 (intnum (cdr x))))
        (else (intnum (cdr x)) )
  ))
```

II реш.

```
(define (intnum1 x)
  (if (null? x) 0
      (+ (intnum1 (cdr x)) (if (integer? (car x)) 1 0))
  ))
```

3.7. Дефинирайте функция, която преброява елементите на списък x, които са списъци от n елемента.

```
(define (br x n)
  (cond ((null? x) 0)
        ((and (list? (car x)) (= (length (car x)) n)) (+ (br (cdr x) n) 1))
        (else (br (cdr x) n))
  ))
```

3.8. Дефинирайте функция, която намира сумата (произведението) на елементите на списък, които са цели числа (които са числа по-малки от 5 или по-големи от 20).

3.9. Дефинирайте функция, която намира сумата на всички цели числа в списък, вкл. съдържащите се и в елементите-списъци (Напр. за (a 2 (b 3 c) 8.12 ((5 3.4))) резултатът е 10).

```
(define (sumell L)
  (cond ((null? L) 0)
        ((integer? (car L)) (+ (car L) (sumell (cdr L))))
        ((list? (car L)) (+ (sumell (car L)) (sumell (cdr L))))
        (else (sumell (cdr L)))
  ))
```

3.10. Напишете функция, която пресмята сумата от четните числа в (числов) списък.

3.11. Напишете функция, която пресмята $\prod_{k=1}^n (a_k - k)$ за даден числов списък $(a_1 a_2 \dots a_n)$.

```
(define (prod L)
  (define (f i L)
    (if (null? L) 1 (* (- (car L) i) (f (+ i 1) (cdr L)))))
  (if (not (null? L)) (f 1 L))
)
```

II решение (итеративно)

```
(define (prodl L)
  (define (f i L p)
    (if (null? L) p (f (+ i 1) (cdr L) (* p (- (car L) i)))))
  (if (not (null? L)) (f 1 L 1))
)
```

3.12. Дефинирайте функция, която за даден числов списък $(a_1 a_2 a_3 a_4 \dots)$ намира сумата: $a_1^2 \cdot a_2 + a_3^2 \cdot a_4 + \dots$

```
(define (suma L)
  (cond ((null? L) 0)
        ((null? (cdr L)) (expt (car L) 2))
        (else (+ (* (expt (car L) 2) (cadr L)) (suma (cddr L)))))
  ))
```

3.13. Дефинирайте функция, която проверява дали в един списък има 2 съседни равни елемента.

```
(define (ne2eq x)
  (cond ((< (length x) 2) #f)
        ((equal? (car x) (cadr x)) #t)
        (else (ne2eq (cdr x))))
  ))
```

II реш.

```
(define (twoeq x)
  (and (> (length x) 1)
       (or (equal? (car x) (cadr x)) (twoeq (cdr x)))))
```

3.14. Дефинирайте функция, която проверява дали един списък от числа е монотонно намаляващ.

```
(define (namal? x)
  (or (<= (length x) 1)
      (and (>= (car x) (cadr x)) (namal? (cdr x))))
)
```

3.15. Дефинирайте функция, която конструира списък от n еднакви елемента x .

```
(define (dupln x n)
  (cond ((= n 0) '())
        ((> n 0) (cons x (dupln x (- n 1))))))
```

3.16. Дефинирайте функция, която конструира списък от числата от n до 1.

3.17. Дефинирайте функция, която конструира списък от числата от n до 1, всяко записано по два пъти един след друг: напр. (5 5 4 4 3 3 2 2 1 1).

```
(define (make2 n)
  (cond ((= n 0) '())
        ((> n 0) (cons n (cons n (make2 (- n 1))))))
  )
```

3.18. Дефинирайте функция, която конструира списък от числата от 1 до n

```
(define (make3 n)
  (define (pom i)
    (cond ((> i n) '())
          ((<= i n) (cons i (pom (+ i 1))))))
  (pom 1)
  )
```

3.19. Напишете функция, която по зададено число n конструира списък на числата от 1 до n , записани по следния начин: (2 1 4 3 ...).

3.20. Дефинирайте функция, която от дадени три списъка ($a_1 a_2 a_3 \dots$), ($b_1 b_2 b_3 \dots$) и ($c_1 c_2 c_3 \dots$) конструира четвърти по следния начин: (($a_1 b_1 c_1$) ($a_2 b_2 c_2$) ($a_3 b_3 c_3$)...). (При изчерпване на единия списък елементите, останали в другите се игнорират.)

```
(define (makel x y z)
  (if (or (null? x) (null? y) (null? z)) '()
      (cons (list (car x) (car y) (car z)) (makel (cdr x) (cdr y) (cdr z)))))
```

3.21. Дефинирайте функция, която от дадени два списъка ($a_1 a_2 a_3 \dots$) и ($b_1 b_2 b_3 \dots$) конструира трети по следния начин: ($a_1 b_2 a_3 b_4 \dots$).

```
(define (slivanel x y)
  (cond
    ((null? x) '())
    ((< (length y) 2) (cons (car x) '()))
    ((= (length x) 1) (cons (car x) (cons (cadr y))))
    ((cons (car x)
           (cons (cadr y) (slivanel (cddr x) (cddr y))))))
  )
```

II начин

```
(define (sliv x y)
  (cond ((null? x) '())
        ((null? y) (cons (car x) '()))
        ((cons (car x) (sliv (cdr x) (cdr y)))))
  )
```

3.22. Дефинирайте функция, която дублира един след друг елементите на списък, които са символи: (1 a c 3 (a d))-->(1 a a c c (a d)).

```
(define (duplsym x)
  (cond ((null? x) '())
        ((symbol? (car x)) (cons (car x) (cons (car x) (duplsym (cdr x)))))
        ((cons (car x) (duplsym (cdr x)))))
  )
```

3.23. Дефинирайте функция, която вмъква символа s след всяка 0 в списък.

```
(define (ins x)
  (cond ((null? x) '())
        ((equal? (car x) 0) (cons (car x) (cons 's (ins (cdr x)))))
        (else (cons (car x) (ins (cdr x)))))
  )
```

3.24. Напишете функция, която конструира нов списък от даден чрез замяна на всяко срещане на x в списъка с y .

```
(define (subst x y L)
  (cond ((null? L) '())
        ((equal? x (car L)) (cons y (subst x y (cdr L))))
        (else (cons (car L) (subst x y (cdr L))))
  ))
```

Решение, ако се търси и заменя и в елементите-списъци. Пр.: ако търсим 'а и заменяме с 1 в (а 3 4 (а d а) ((а)) d а (1 а)), ще получим (1 3 4 (1 d 1) ((1)) d 1 (1 1)).

```
(define (subst1 x y L)
  (cond ((null? L) '())
        ((equal? x (car L)) (cons y (subst1 x y (cdr L))))
        ((list? (car L)) (cons (subst1 x y (car L)) (subst1 x y (cdr L))))
        (else (cons (car L) (subst1 x y (cdr L))))
  ))
```

3.25. Дефинирайте функция, която увеличава с 2 всеки 3-ти елемент на числов списък.

```
(define (uvel2 L)
  (if (< (length L) 3) L
      (cons (car L) (cons (cadr L) (cons (+ (caddr L) 3) (uvel2 (cdddr L))))))
  ))
```

3.26. Дефинирайте функция, която изтрива първото срещане на 4 в даден списък.

```
(define (del4 x)
  (cond ((null? x) '())
        ((equal? 4 (car x)) (cdr x))
        (else (cons (car x) (del4 (cdr x))))
  ))
```

3.27. Дефинирайте функция, която изтрива всяко срещане на x в списък (x се подава като параметър).

```
(define (del x L)
  (cond ((null? L) '())
        ((equal? (car L) x) (del x (cdr L)))
        (else (cons (car L) (del x (cdr L))))
  ))
```

3.28. Дефинирайте функция, която от списъка ($a_1 a_2 a_3 \dots$), конструира списъка ($a_1^2 a_2 a_3^2 a_4 \dots$).

```
(define (formlist L)
  (cond ((null? L) '())
        ((null? (cdr L)) (cons (* (car L) (car L)) '()))
        (else (cons (* (car L) (car L)) (cons (cadr L) (formlist (cddr L))))))
  ))
```

3.29. Дефинирайте функция, която конструира нов списък състоящ се от елементите на даден, записани в обратен ред.

```
(define (rev L)
  (define (pom L R)
    (if (null? L) R (pom (cdr L) (cons (car L) R))))
  (pom L '()))

(define (invert L)
  (if (null? L) '() (append (invert (cdr L)) (cons (car L) '()))))
  )
```

3.30. Дефинирайте функция, която намира максималния елемент на числов списък.

```
(define (maxel L)
  (if (not (null? L))
      (if (null? (cdr L)) (car L)
          (max (car L) (maxel (cdr L)))))
  ))
```

3.31. Дефинирайте функция, която намира n-ия елемент на списък.

3.32. Дефинирайте функция, която намира последния елемент на списък.

```
(define (last L)
  (cond ((= (length L) 1) (car L))
        ((not (null? L)) (last (cdr L))))
)
```

Напишете решение и с използването на reverse или ф-ята за намиране на n-и елемент.

Бележки:

1. Символът empty има стойност празен списък и навсякъде, където е използван '(), можете да го заместите с empty. (На лекциите използвате символа nil за празен списък, но във версията от упражненията не е дефиниран! Ако искате можете да си го дефинирате и после използвате.)

2. Някои вградени предикатни функции (върщат логическа стойност (#t или #f)):

- с един аргумент: integer?, real?, rational?, number?, symbol?, string?, list?, null?, odd?, even?, zero?;

- с два аргумента: equal? (за сравняване и на нечислови стойности като зависи от типа), eq? и eqv? за сравняване на обекти за еквивалентност.

> (equal? '(1 2) '(1 2))	> (equal? 5 5.0)
#t	#f
> (eq? '(1 2) '(1 2))	> (equal? 3/2 1.5)
#f	#f
> (eqv? '(1 2) '(1 2))	> (equal? 5.0 5.0)
#f	#t
	> (equal? 3/2 3/2)
> (equal? "abc" "abc")	#t
#t	> (equal? 5 5)
> (eq? "abc" "abc")	#t
#f	
> (eqv? "abc" "abc")	> (eqv? 3/2 3/2)
#f	#t
	> (eq? 3/2 3/2)
> (define a '(1 2))	#f
> (define b a)	
> (eq? a b)	
#t	
> (eqv? a b)	
#t	

3. Някои вградени функции за обработка на списъци: length, member, append

```
> (length '())
0
> (length '(1 2 () a (5 6)))
5
> (member 'a '(1 s d a h (3 4) a k))
(a h (3 4) a k)
> (member 1 '(a b c))
#f
> (append '(1 2 3) '(4 5))
(1 2 3 4 5)
> (append '(a b c) 'd)
(a b c . d)
> (append '(a b c) '(d))
(a b c d)
> (reverse '(1 2 3 4))
(4 3 2 1)
```