## **Задание 1**

Чем принципиально отличаются функции cons, list, append?

1. cons – помещает первый аргумент в начало второго;
2. list – создает список, состоящий из аргументов;
3. append – создает список, состоящий из элементов аргументов.

Пусть (setf lst1 '(a b))

(setf lst2 '(c d)).

Каковы результаты вычисления следующих выражений?

1. (cons lstl lst2) -> ((A B) C D)
2. (list lst1 lst2) -> ((A B) (C D))
3. (append lst1 lst2) -> (A B C D)

## **Задание 2**

Каковы результаты вычисления следующих выражений, и почему?

1. (reverse ()) -> NIL
2. (last ()) -> NIL
3. (reverse '(a)) -> (A)
4. (last '(a)) -> (A)
5. (reverse '((a b c))) -> ((A B C))
6. (last '((a b c))) -> ((A B C))

## **Задание 3**

Написать, по крайней мере, два варианта функции, которая возвращает последний элемент своего списка-аргумента.

**Решение**

(defun last\_v1 (x)

(if (listp x)

(if (<= (length x) 1)

(car x)

(last\_1 (cdr x)))

NIL))

(defun last\_v2 (x)

(cond ((null (listp x)) NIL)

((null (cdr x)) (car x))

(T (last\_v2 (cdr x)))))

(defun last\_v3 (x)

(if (or (null (listp x)) (null x))

NIL

(nth (- (length x) 1) x)))

(defun last\_v4 (x)

(if (listp x)

(car (reverse x))

NIL))

## **Задание 4**

Написать, по крайней мере, два варианта функции, которая возвращает свой список-аргумент без последнего элемента.

**Решение**

(defun except\_last\_v1 (x)

(if (null (and (listp x) (cdr x)))

NIL

(cons (car x) (except\_last\_v1 (cdr x)))))

(defun except\_last\_v2 (x)

(if (listp x)

(reverse (cdr (reverse x)))))

(defun except\_last\_v3 (x)

(if (listp x)

(butlast x)

NIL))

## **Задание 5**

Написать простой вариант игры в кости, в котором бросаются две правильные кости. Если сумма выпавших очков равна 7 или 11 – выигрыш, если выпало (1,1) или (6,6) – игрок имеет право снова бросить кости, во всех остальных случаях ход переходит ко второму игроку, но запоминается сумма выпавших очков. Если второй игрок не выигрывает абсолютно, то выигрывает тот игрок, у которого больше очков. Результат игры и значения выпавших костей выводить на экран с помощью функции print.

**Решение**

(setf \*random-state\* (make-random-state t))

(defvar Player\_1 (cons "Player\_1" 0))

(defvar Player\_2 (cons "Player\_2" 0))

(defun throw\_bones ()

(cons (+ 1 (random 6 \*random-state\*)) (+ 1 (random 6 \*random-state\*)))

)

(defun get\_bones\_sum(bones)

(+ (car bones) (cdr bones))

)

(defun print\_bones\_throw (player bones)

(format t "~a throws ~a" (car player) bones)

(terpri)

)

(defun print\_player\_win (player)

(format t "~S Wins" (car player))

(terpri)

)

(defvar win\_sum (cons 7 11))

(defvar reroll\_sum (list (cons 1 1) (cons 6 6)))

(defun my\_compare (first second)

(and (= (car first) (car second)) (= (cdr first) (cdr second)))

)

(defun check\_is\_reroll (bones)

(or (my\_compare bones (car reroll\_sum)) (my\_compare bones (cadr reroll\_sum)))

)

(defun player\_turn (player)

(let\* ((bones (throw\_bones)) (bones\_sum (get\_bones\_sum bones)))

(print\_bones\_throw player bones)

(if (or (= bones\_sum (car win\_sum)) (= bones\_sum (cdr win\_sum)))

(or (print\_player\_win player) -1)

;else

(if (check\_is\_reroll bones)

(or (format t "Reroll")

(terpri)

(player\_turn player)

)

;else

bones\_sum

)

)

)

)

(defun play()

(let\* ((first\_sum (player\_turn Player\_1)))

(if (= first\_sum -1)

(list 1)

;else

(let\* ((second\_sum (player\_turn Player\_2)))

(if (= second\_sum -1)

(list 2)

;else

(if (> first\_sum second\_sum)

(print\_player\_win Player\_1)

(print\_player\_win Player\_2)

)

)

)

)

)

)

(play)