Aluno: Michel Bolzon Souza Dos Reis
Ra:83558

Lista de naturais

Questão 2:

```
#lang racket

(require rackunit)
(require rackunit/text-ui)

;; Número -> Número
;; A função recebe dois números e soma eles.

(define verifica-tests-soma
  (test-suite
   "verifica-test1"
   (check-equal? (soma 0 1) 1)
   (check-equal? (soma 1 1) 2)
   (check-equal? (soma 9999 5000000) 5009999)))

(define (soma n m)
  (cond
    [(zero? m) n]
    [else (soma (add1 n) (sub1 m))]))

;; Número -> Número
;; A função recebe dois números e subtrai eles.

(define verifica-tests-sub
  (test-suite
   "verifica-test1"
   (check-equal? (sub 0 1) -1)
   (check-equal? (sub 1 1) 0)
   (check-equal? (sub 100 50) 50)
   (check-equal? (sub 110 30) 80)))

(define (sub n m)
  (cond
    [(zero? m) n]
    [else (sub (sub1 n) (sub1 m))]))

;; Número -> Número
;; A função recebe dois números e multiplica eles.
```

```
(define verifica-tests-mul
  (test-suite
   "verifica-test1"
   (check-equal? (mul 0 1) 0)
   (check-equal? (mul 1 1) 1)
   (check-equal? (mul 5 4) 20)
   (check-equal? (mul 4 5) 20)
   (check-equal? (mul 6 3) 18)
   (check-equal? (mul 25 4) 100)))

(define (mul n m)
  (cond
    [(zero? m) 0]
    [(zero? n) 0]
    [(equal? m 1) n]
    [else (soma (mul n (sub1 m)) n)]))


;; Teste ... -> void
;; Executa um conjunto de testes
(define (executa-tests . testes)
  (run-tests (test-suite "Todos os Testes" testes))
  (void))

;; Chama a função para executar os testes
(executa-tests verifica-tests-soma verifica-tests-sub verifica-tests-mul)
```

Lista de Combinação de Modelos

Questão 3:

```
#lang racket

(require rackunit)
(require rackunit/text-ui)

;; Lista Número -> Lista
;; A função recebe um número x e uma lista, e retorna a lista sem os x primeiros elementos.

(define verifica-tests
  (test-suite
   "verifica-test1"
   (check-equal? (drop empty 5) empty)
   (check-equal? (drop (list 2 3 4) 5) empty)
   (check-equal? (drop (list 0 2 3 4) 1) (list 2 3 4))
```

```
   (check-equal? (drop (list 2 3 4 5 6) 2) (list 4 5 6))
   (check-equal? (drop (list 1 2 3 4 5 6) 0) (list 1 2 3 4 5 6))
   (check-equal? (drop (list 0 2 3 5 4) 5) empty)))

(define (drop lst x)
  (cond
    [(empty? lst) empty]
    [(zero? x) rest lst]
    [else (drop (rest lst) (sub1 x))]))


;; Teste ... -> void
;; Executa um conjunto de testes
(define (executa-tests . testes)
  (run-tests (test-suite "Todos os Testes" testes))
  (void))

;; Chama a função para executar os testes
(executa-tests verifica-tests)
```

Questão 5:

```
#lang racket

(require rackunit)
(require rackunit/text-ui)

;; Lista Número Número -> Lista
;; A função recebe uma lista e dois números, e retorna a lista com o primeiro número na posição que o segundo número informa.

(define verifica-tests
  (test-suite
   "verifica-test1"
   (check-equal? (insert-at empty 5 2) (list 5))
   (check-equal? (insert-at (list 2 3 4) 5 5) (list 2 3 4 5))
   (check-equal? (insert-at (list 2 3 4) 5 0) (list 5 2 3 4))
   (check-equal? (insert-at (list 2 3 4) 5 1) (list 2 5 3 4))
   (check-equal? (insert-at (list 2 3 4 5 6) 5 3) (list 2 3 4 5 5 6))
   (check-equal? (insert-at (list 1 2 3 4 5 6) 5 -1) (list 5 1 2 3 4 5 6))
   (check-equal? (insert-at (list 1 0 2 3 5 4) 5 6) (list 1 0 2 3 5 4 5))
   (check-equal? (insert-at (list 1 0 2 3 5 4) 5 3) (list 1 0 2 5 3 5 4))))

(define (insert-at lst x p)
  (cond
    [(empty? lst) (cons x empty)]
```

```
    [(< p 0) (cons x lst)]
    [(zero? p) (cons x lst)]
    [else (cons (first lst) (insert-at (rest lst) x (sub1 p)))]))


;; Teste ... -> void
;; Executa um conjunto de testes
(define (executa-tests . testes)
  (run-tests (test-suite "Todos os Testes" testes))
  (void))

;; Chama a função para executar os testes
(executa-tests verifica-tests)
```

Questão 8:

```
#lang racket

(require rackunit)
(require rackunit/text-ui)

;; Lista Lista -> Lista
;; A função recebe duas listas, e retorna uma lista com os elementos da segunda lista após
os elementos da primeira.

(define verifica-tests
  (test-suite
   "verifica-test1"
   (check-equal? (append empty empty) empty)
   (check-equal? (append (list 2 3 4) empty) (list 2 3 4))
   (check-equal? (append empty (list 5 0)) (list 5 0))
   (check-equal? (append (list 2 3 4) (list 5 1)) (list 2 3 4 5 1))
   (check-equal? (append (list 2 3 3 4 5 6 7 4) (list 5 1 1 1 1 1 1 1 1 1)) (list 2 3 3 4 5 6 7 4 5 1
1 1 1 1 1 1 1 1))
   ))

(define (append lsta lstb)
  (cond
    [(and (empty? lsta) (empty? lstb)) empty]
    [(empty? lsta) lstb]
    [(empty? lstb) lsta]
    [(empty? (rest lsta)) (cons (first lsta) lstb)]
    [else (cons (first lsta) (append (rest lsta) lstb))]))


;; Teste ... -> void
```

```
;; Executa um conjunto de testes
(define (executa-tests . testes)
  (run-tests (test-suite "Todos os Testes" testes))
  (void))

;; Chama a função para executar os testes
(executa-tests verifica-tests)
```