# 第三章作业

2019年9月8日 0:47

```
#lang racket
(define (make-accumulator x)
  (define (add amount)
    (begin (set! x (+ x amount))
    x
    )
    )
  add
  )

(define A (make-accumulator 5))
(A 10)
(A 10)

(define (make-monitored f)
  (define (helper f times)
    (define (run)
      (begin (set! times (+ times 1))
             (lambda (t) (f t))))
    (define (how-many-calls?)
      times)
    (define (reset)
      (set! times 0))
    (define (dispatch m)
      (cond ((eq? m 'how-many-calls?) (how-many-calls?))
            ((eq? m 'reset) (reset))
            (else ((run) m))))
    dispatch
    )

  (helper f 0)
  )
(define s (make-monitored sqrt))
(s 100)
(s 'how-many-calls?)
(s 25)
(s 'how-many-calls?)
(s 'reset)
(s 'how-many-calls?)
(s 'how-many-calls?)
```

## 3.5

```
#lang racket

(define (random-in-range low high)
  (let ((diff (- high low)))
            (+ low (* (random) diff))))

(define (mente-carlo trials experiment)
(define (iter trials-remaining trials-passed)
  (cond ((= trials-remaining 0)
         (/ trials-passed trials))
        ((experiment)
         (iter (- trials-remaining 1) (+ trials-passed 1)))
        (else
         (iter (- trials-remaining 1) trials-passed))))
  (iter trials 0)
  )

(define (square x) (* x x))

(define (meto-cario x1 x2 y1 y2)
  (let ((x (random-in-range x1 x2))
        (y (random-in-range y1 y2)))
        (< (+ (square (- x  1)) (square (- y 1))) 1)))

(define (estimate-pi trails x1 x2 y1 y2)
  (* (mente-carlo trails (lambda () (meto-cario x1 x2 y1 y2))) (- x2 x1) (- y2 y1)))

(define (rand command)
  (case command
    ('generate (random))
    ('reset
     (lambda (new)
       (random-seed new)))
    (else
     (error "Bad command -- " command))))
```

3.8

```
#lang racket
(define (f n)
  (if (= n 0)
      (begin (set! f (lambda (t) 0)) 0)
      (begin (set! f (lambda (t) 0)) 1)
      )
  )
      (+ (f 0) (f 1))

#lang planet neil/sicp
(define (loop? t)
  (define (iter x visited)
    (cond
      ((null? x) #f)
      ((memq (car x) visited) #t)
      (else (iter (cdr x) (cons (car x) visited)))
      )
    )
  (iter t '())
  )


(define (last-pair x)
(if (null? (cdr x)) x (last-pair (cdr x))))


(define (make-cycle x)
  (set-cdr! (last-pair x) x)
  x)

(loop? (list 1 2 3))

 (define loop (make-cycle (list 1 2 3)))

(loop? loop)
(loop? (list 1))
(define loop-list (list 1 2 3))

(set-cdr! (last-pair loop-list) loop-list)

(loop? loop-list)

(define (loop2? t)
  (define (iter sp fp)
    (let ((slow-pointer (cdr sp)))
      (if (or (null? slow-pointer) (null? (cdr fp))) #f
          (let ((fast-pointer (cdr (cdr fp))))
            (if (eq? slow-pointer fast-pointer) #t
            (iter slow-pointer fast-pointer)))
            )))
    (iter t t)

  )

(loop2? (list 1 2 3))

 (define loop2 (make-cycle (list 1 2 3)))

(loop2? loop2)
(loop2? (list 1))
(define loop-list2 (list 1 2 3))

(set-cdr! (last-pair loop-list2) loop-list2)

(loop2? loop-list2)
```

## 3.38

A) 45,  50, 35, 40

B) 55, 110, 80, 90, 30, 60