Metody Programowania lista 2

Szymon Kopa

20 lutego 2018

1 Zadanie 2

```
#lang racket

(define (compose f g)
  (lambda (a) (f (g a))))
(
(define (square x) (* x x))

(define (inc x) (+ x 1))

(define (identity x) x)
```

2 Zadanie 3

```
#lang racket

(define (repeated f n)
  (cond
    [(= n 0) identity]
    [(= n 1) f]
    [else (compose (identity f) (repeated f (- n 1 )))]))
```

3 Zadanie 5

```
#lang racket

(define (accumulate combiner null-value term a next b)
  (define ( accumulate-iter a acc)
    (if [> a b ]
        null-value
        (accumulate-iter (next a) (combiner acc (term a))))))

;REKURENCJA

(define (accumulate-recursive combiner null-value term a next b)
  (if [> a b]
      null-value
      (combiner (term a) (accumulate-recursive term (next a) next b))))
```

4 Zadanie 6

```
#lang racket

(define (cont-frac num den k)
```

```
  (define (counter acc)
    (cond
      [(= acc k) 0]
      [else (/ (num acc) (+(den acc) counter )(+ acc 1)])))
  (counter 0))

;iter
(define (cont-frac num den k)
  (define (counter acc wynik)
    (cond
      [(= acc 0) wynik]
      [else (counter (\ (num acc) (+(den acc) wynik))(- acc 1))])))
```

5 Ćwiczenie 11

```
  #lang racket
  (define (compose f g)
    (lambda (x) (f (g x))))

  (define (repeated f n)
    (if (= 1 n)
        f
        (compose f (repeated f (- n 1)))))


  (define (fixed-point f first-guess)
    (define (close-enough? v1 v2)
      (let ((tolerance 0.00001))
        (< (abs (- v1 v2)) tolerance)))
    (define (try guess)
      (let ((next (f guess)))
        (if (close-enough? guess next)
            next
            (try next))))
    (try first-guess))

  (define (average a b)
    (/ (+ a b) 2))

  (define (average-damp f)
    (lambda (x) (average x (f x))))

  (define (fixed-point-of-transform g transform guess)
    (fixed-point (transform g) guess))

  (define (damp-number n)
    (floor (/ (log n) (log 2))))

  (define (nth-root-of x n)
    (fixed-point-of-transform
      (lambda (y) (/ x (expt y (- n 1))))
```

```
    (repeated average-damp (damp-number n))
    1.0))

(define (test x n)
  (<= ( -( abs (- ( nth-root-of x n) (expt x (/ 1 n))))) 0.00001))

(nth-root-of 16 4) (expt 16 (/ 1 4)) (test 16 4)
(nth-root-of 512 9) (expt 512 (/ 1 9)) (test 512 9)
(nth-root-of 31 5) (expt 31 (/ 1 5)) (test 31 5)
(nth-root-of 118 7) (expt 118 (/ 1 7)) (test 118 7)
(nth-root-of 216 6) (expt 216 (/ 1 6)) (test 216 6)
(nth-root-of 27 3) (expt 27 (/ 1 3)) (test 27 3)
(nth-root-of 81 2) (expt 81 (/ 1 2)) (test 81 2)
(nth-root-of 18181818 17) (expt 18181818 (/ 1 17)) (test 18181818 17)
```