# Metody Programowania lista 3

Szymon Kopa

7 marca 2018

# 1 Zadanie 1

```
  #lang racket
(define (make-rat n d)
  (let ((g (gcd n d)))
  (cons (/ n g) (cons (/ d g) null))))

(define(numer x)
  (car x))

(define (denom x)
  (car (cdr x)))

(define (add-rat x y)
  (make-rat (+ (* (numer x) (denom y))
               (* (numer y) (denom x)))
            (* (denom x) (denom y))))

(define (sub-rat x y)
  (make-rat (- (* (numer x) (denom y))
               (* (numer y) (denom x)))
            (* (denom x) (denom y))))

(define (mul-rat x y)
  (make-rat (* (numer x) (numer y))
            (* (denom y) (denom x))))

(define (div-rat x y)
  (make-rat (* (numer x) (denom y))
            (* (numer y) (denom x))))


(define (list? x)
  (or (null? x)
      (and (pair? x)
           (list? (cdr x)))))


(define (length x)
  (and (list? x)
       (if (null? x)
       0
       (+ 1 (length (cdr x))))))

(define (rat? x)
  (and (list? x)
       (= 2 (length x))
       (not (= (denom x) 0))
       (= 1 (gcd (numer x) (denom x)))))
```

## 2 Zadanie 2

```
  (define (make-point x y)
  (cons x y))

(define (point-x x)
  (car x))

(define (point-y x)
  {cdr x})

(define (point? x)
  (and (pair? x)
       (number? (point-x x))
       (number? (point-y x))))

( define ( display-point p )
   ( display "(")
   ( display ( point-x p ) )
   ( display ", ")
   ( display ( point-y p ) )
   ( display ")") )


(define (make-vect a b)
  (cons a b))

(define (vect-begin x)
  (car x))

(define (vect-end x)
  (cdr x))

(define (vector? x)
  (and (pair? x)
       (point? (vect-begin x))
       (point? (vect-end x))))

( define ( display-vect v )
   ( display "[")
   ( display-point ( vect-begin v ) )
   ( display ", ")
   ( display-point ( vect-end v ) )
   ( display "]") )

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

(define (vect-length x)
  (sqrt (+ (square (- (point-x (vect-end x))
                      (point-x (vect-begin x))))
           (square (- (point-y (vect-end x))
                      (point-y (vect-begin x)))))))
```

```
(define (vect-scale v k)
  (make-vect (vect-begin v)
             (make-point (* k (point-x (vect-end v)))
                         (* k (point-y (vect-end v))))))

(define (vect-translate v p)
  (make-vect p
             (make-point (+ (point-x p) (- (point-x (vect-end v))
                                           (point-x (vect-begin v))))
                         (+ (point-y p) (- (point-y (vect-end v))
                                           (point-y (vect-begin v)))))))
```

---