Racket CheatSheet

Laborator4

let

Colorată - zona de vizibilitate pentru id1
Valoare de retur - exprn

```
(let ((id1 val1)
      (id2 val2)
      ...
      (idn valn))
   expr1
   expr2
   ...
   exprn)

(define a 10)

(let ((a 1) (b (+ a 1)))
   (cons a b))                              (1 . 11)
```

let*

Colorată - zona de vizibilitate pentru id1
Valoare de retur - exprn

```
(let* ((id1 val1)
       (id2 val2)
       ...
       (idn valn))
   expr1
   expr2
   ...
   exprn)

(define a 10)

(let* ((a 1) (b (+ a 1)))
   (cons a b))                              (1 . 2)
```

named let

nume - apare în corp ca un apel recursiv al funcției cu parametrii id1 .. idn și corpul corp

```
(let nume ((id1 val1)
           (id2 val2)
           ...
           (idn valn))
   corp)

(let loop ((n 5)
           (fact 1))
   (if (zero? n)
       fact
       (loop (sub1 n) (* n fact))))          120
```

letrec

Colorată - zona de vizibilitate pentru id2
Valoare de retur - exprn

```
(letrec ((id1 val1)
         (id2 val2)
         ...
         (idn valn))
   expr1
   expr2
   ...
   exprn)

;; cand evaluez b, b trebuie sa fi fost definit
(letrec ((a b) (b 1))
   (cons a b))                                eroare

;; corpul unei inchideri functionale
;; nu se evalueaza la momentul definirii
(letrec
     ((even-length?
       (lambda (L)
         (if (null? L)
             #t
             (odd-length? (cdr L)))))
      (odd-length?
       (lambda (L)
         (if (null? L)
             #f
             (even-length? (cdr L))))))
   (even-length? '(1 2 3 4 5 6)))               #t
```

let-values

Ca let, pentru expresii care întorc valori multiple

```
;; val-expr este o expresie care intoarce n valori
(let-values ( ((id1 id2 .. idn) val-expr)
              ... )
   corp)

(let-values (((x y) (quotient/remainder 10 3)))
   (cons x y))                                (3 . 1)
```

Alte funcții

sort remove assoc andmap findf splitf-at

```
(sort '(5 2 1 6 4) >)                      (6 5 4 2 1)
(remove 2 '(1 2 3 4 3 2 1))              (1 3 4 3 2 1)
(assoc 3 '((1 2) (3 4) (3 6) (4 5)))          (3 4)
(andmap positive? '(1 2 3))                     #t
(andmap number? '(1 b 3))                       #f
(findf (lambda (x) (> x 4)) '(1 3 5 6 4))        5
(findf (lambda (x) (> x 6)) '(1 3 5 6 4))       #f
(splitf-at '(1 3 4 5 6) odd?)                (1 3)
                                           (4 5 6)
(splitf-at '(1 3 4 5 6) even?)                   ()
                                       (1 3 4 5 6)
```

Folositi cu incredere!

http://docs.racket-lang.org/