Racket CheatSheet

Laborator1

Sintaxa Racket

(nume_functie arg1 arg2 ...)

```
(max 2 3)                3
(+ 2 3)                  5
```

Tipuri de bază

Valori booleene: `#t #f` (sau `true false`)
Numere: `1 2 3 3.14 ...`
Simboli (literali): `'a 'b 'abc 'non-alnum?!`

Operatori aritmetici

+ - * / modulo quotient add1 sub1

```
(+ 1 2)              3
(- 7 2)              5
(* 2 11)             22
(/ 5 2)              2.5
(quotient 5 2)       2
(modulo 5 2)         1
(add1 4)             5
(sub1 4)             3
```

Operatori relationali

< <= > >= = eq? equal? zero?

```
(< 3 2)                    #f
(>= 3 2)                   #t
(= 1 1)                    #t          (numere)
(= '(1 2) '(1 2))          eroare
(equal? '(1 2) '(1 2))     #t          (valori)
(eq? '(1 2 3) '(1 2 3))    #f          (referinte)

(define x '(1 2 3))
(eq? x x)                  #t

(zero? 0)                  #t (true)
(zero? 1)                  #f (false)
```

Operatori logici

not and or

```
(not true)                 #f
(not false)                #t
(or true false)            #t
(and #f #t)                #f
```

Constructori liste

null '() cons list

```
'()                                ()
null                               ()

(cons 1 null)                      (1)
(cons 'a '(b c))                   (a b c)

(list 1)                           (1)
(list 1 2 3 4)                     (1 2 3 4)
```

Operatori pe liste

car cdr first rest null? length member reverse append

```
(car '(1 2 3 4))                   1
(first '(1 2 3 4))                 1
(cdr '(1 2 3 4))                   (2 3 4)
(rest '(1 2 3 4))                  (2 3 4)
(cadr '(1 2 3 4 5))                2
(cdar '(1 2 3 4 5))                eroare
(cddr '(1 2 3 4 5))                (3 4 5)
(caddr '(1 2 3 4 5))               3

(null? null)                       #t (true)
(null? '(1 2))                     #f (false)

(length '(1 2 3 4))                4
(length '(1 (2 3) 4))              3

(member 'a '(b c a d a e))         '(a d a e)
(member 'f '(b c a d a e))         #f

(reverse '(1 (2 3) 4))             (4 (2 3) 1)

(list? '())                        #t
(list? 2)                          #f

(append '(1 2 3) '(4) '(5))        (1 2 3 4 5)
(append 1 '(2 3 4))                eroare
```

take și drop

```
(take '(1 2 3 4) 2)                '(1 2)
(drop  '(1 2 3 4 5) 2)             '(3 4 5)

(take-right '(1 2 3 4) 2)          '(3 4)
(drop-right  '(1 2 3 4 5) 2)       '(1 2 3)
```

Funcții anonime (lambda) și funcții cu nume

(lambda (arg1 arg2 ...) rezultat) (define nume val)

```
(lambda (x) x)                          functia identitate
((lambda (x) x) 2)                 2    aplicare functie
(define idt (lambda (x) x))             legare la un nume
(define (idt x) x)                      sintaxa alternativa
(idt 3)                            3
```

Sintaxa if

```
(if test exp1 exp2)

(if (< a 0)
    a
    (if (> a 10) (* a a) 0))
```

Sintaxa cond

```
(cond  (test1 exp1) (test2 exp2) ... (else exp))

(cond
   ((< a 0) a)
   ((> a 10) (* a a))
   (else 0))
```

AȘA DA / AȘA NU

```
DA: (cons x L)          NU: (append (list x) L)
                        NU: (append (cons x '()) L)
DA: (if c vt vf)        NU: (if (equal? c #t) vt vf)
DA: (null? L)           NU: (= (length L) 0)
DA: (zero? x)           NU: (equal? x 0)
DA: test                NU: (if test #t #f)
DA: (or ceva1 ceva2)    NU: (if ceva1 #t ceva2)
DA: (and ceva1 ceva2)   NU: (if ceva1 ceva2 #f)
```

Programare cu funcții recursive

1. După ce variabilă(e) fac recursivitatea? (ce variabilă(e) se schimbă de la un apel la altul?)
2. Care sunt condițiile de oprire în funcție de aceste variabile?(cazurile "de bază")
3. Ce se întâmplă când problema nu este încă elementară? (Obligatoriu aici cel puțin un apel recursiv)

Folositi cu incredere!

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