Racket CheatSheet

Laborator 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

((if true + -) (+ 1 2) 3)       6   if-ul se evaluează
                                    la o funcție

(define (comp f g)                  funcția de
  (lambda (x)                       compunere (o)
    (f (g x))))                     a altor 2 funcții

((comp car cdr) '(1 2 3))       2   car o cdr

((comp (lambda (x) (+ x 1)) 11 inc o dublare
       (lambda (x) (* x 2)))
 5)
```

Funcții *curried*/ *uncurried*

```
(define add-uncurried               parametri luați
  (lambda (x y)                     simultan
    (+ x y)))

(add-uncurried 1 2)             3

(define add-curried                 parametri luați
  (lambda (x)                       succesiv
    (lambda (y)
      (+ x y))))

((add-curried 1) 2)             3

(define inc (add-curried 1))        aplicație parțială
```

Perspective asupra funcțiilor binare *curried*:

- Iau parametrii „pe rând”, fiind aplicabile parțial.
- Iau un parametru și întorc o altă funcție de un parametru.

Funcționala `filter`

(filter funcție listă)

Păstrează dintr-o listă elementele pentru care funcția NU întoarce `false`. (`filter f (list` e_1 ... e_n`)`) → (`list` e_{i_1} ... e_{i_m}), cu (`f` e_{i_k}) ↛ `false`.

```
(filter even? '(1 2 3))                 '(2)
```

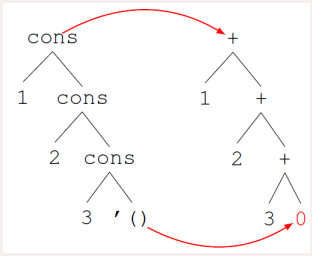
Funcționalele `foldl` și `foldr`

(fold* funcție acumulator listă)
funcție → (lambda (element acumulator') acumulator")

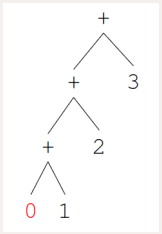
Îmbină toate elementele unei liste pentru a construi o valoare finală, pornind de la un acumulator inițial. Într-un pas, funcția dată ca parametru combină elementul curent din listă cu acumulatorul, întorcând un nou acumulator. Acumulatorul final este întors ca rezultat al funcționalelor `fold*`. Acesta poate fi chiar o listă.

- `foldr` (*right*) poate fi înțeleasă cel mai ușor prin faptul că funcția dată ca parametru se substituie lui `cons`, iar acumulatorul inițial, listei vide de la finalul listei. Prin urmare, elementele listei sunt prelucrate de la dreapta la stânga: (`foldr f acc (list` e_1 ... e_n`)`) → (`f` e_1 (`f` ... (`f` e_n `acc`)...))
- `foldl` (*left*) prelucrează elementele de la stânga la dreapta: (`foldl f acc (list` e_1 ... e_n`)`) → (`f` e_n (`f` ... (`f` e_1 `acc`)...))

Se pot furniza mai multe liste, caz în care comportamentul este similar cu cel al lui `map` pe mai multe liste.




```
(foldr + 0 '(1 2 3))  6
```




```
(foldl + 0 '(1 2 3))  6
(foldr cons '() '(1 2 3))    '(1 2 3)   identitate!
(foldl cons '() '(1 2 3))    '(3 2 1)   inversare
(foldl (lambda (x y acc)              21   2 liste
         (+ x y acc))
       0 '(1 2 3) '(4 5 6))
```

Funcționala `map`

(map funcție listă)
(map funcție lista1 lista2 ...)

Transformă independent elementele de pe poziții diferite ale uneia sau mai multor liste. Întoarce o listă cu același număr de elemente ca lista/ listele date ca parametru.

- Pentru o singură listă, aplică funcția, pe rând asupra fiecărui element: (`map f (list` e_1 ... e_n`)`) → (`list (f` e_1`)` ... (`f` e_n`)`)
- Pentru mai multe liste de aceeași lungime, funcția este aplicată la un moment dat asupra tuturor elementelor de pe aceeași poziție: (`map f (list` e_{11} ... e_{1n}`)` ... (`list` e_{m1} ... e_{mn}`)`) → (`list (f` e_{11} ... e_{m1}`)` ... (`f` e_{1n} ... e_{mn}`)`)

Există și funcționalele `andmap` și `ormap`. Prima se asigură că, în urma aplicării lui `map`, toate rezultatele sunt diferite de `false`, iar a doua, că cel puțin un rezultat este diferit de `false`.

```
(map (lambda (x) (* x 10)) '(1 2 3))    '(10 20 30)
(map * '(1 2 3) '(10 20 30))            '(10 40 90)
(map list '(1 2 3))                   '((1) (2) (3))
(map list '(1 2) '(3 4))              '((1 3) (2 4))

(define (mult-by q)                       ; Curried
  (lambda (x)
    (* x q)))
(map (mult-by 5) '(1 2 3))                '(5 10 15)
```

Funcționala `apply`

(apply funcție listă_arg)
(apply funcție arg_1 ... arg_n listă_arg)

Aplică o funcție asupra parametrilor dați de elementele unei liste. Opțional, primii parametri ai funcției îi pot fi furnizați individual lui `apply`, înaintea listei cu restul parametrilor. (`apply f` x_1 ... x_m (`list` e_1 ... e_n`)`) → (`f` x_1 ... x_m e_1 ... e_n)

```
(apply + '(1 2 3))                  6          suma
(apply + 1 '(2 3))                  6          la fel
(apply list '(1 2 3))               '(1 2 3)
(apply list '(1 2 3) '(5 6 7))      '((1 2 3) 5 6 7)
```