EVALUARE

Aplicativa	Lenga
Aplicativa (Xx. Ly (x+y) 1 (Lz. (z+2) 3))	(xx. ly (x+y) 1 (1/2. (2+2) 3))
(xx. xy (x+1) 1 5)	1 + (x2. (2+2) 3)
6	1 + 5
	6
-> se evalue să întăi parametrii fundiei -> "vreau resultatul!" (eager sudu ation)	-> intai re aplică fundia -> re infarrie evaluarea param. pâna când chiar est revoie de euzullat -> "mai aylept" (lazy evaluation)
Racket	Haskell
transferred param: poin volcare	trous find param: prin nume

Exemple

Aplicativa	Lenza
(Fix d) = (xf. (f (Fix d)) d)	(Fix d) = X9. (f (Fix d) d)
= d $(Fix d) => ev. paremet sul}= d (\lambda f.(f (Fix d)) d)$	= d (Fix d) = \(\lambda \times \text{(Fix d)}\)
= d (d (<u>Fix d))</u> = d (d ()/(f (Fix d)) d)	= & poram. nu mai e
= ch (d (d (Fix d)))	evalual, pt in mu
	mai e mevoir de el

Cum polem "traduce" evaluour lunga in Racket?

1 N CHI DE RI	PROMISIUNI
(define sum (lambda(x y)	(define sum (lambda(x y)
(lambda () (+ x y))))	(delay (+ x y))))
(sum 12) 2) le inhance aux april ? 200 fundit	(sum 1 2); va afiṣa # <promise></promise>
Pentru a forța evaluarea	Pentru a forța evaluarea
((sum 12))	(force (sum 1 2))

FLUXURI

= oliveta infinite / ji ruri -> x lazează pe evaluare levuă

= (elem. curent, generator)

primul elem.

from siene inch. fd.

(define ones-stream
(cons 1 (lambda () ones-stream)))

Jennator
(inclidere fd)

(define ones-stream
(cons 1 (delay ones-stream)))

Lum.wunt (promi num)

(e re intampla? -> generatorul produce reloxile atunci cond est nevoie de ele

Lioke	Flexuri
cons	stream-cons
car	stream-first
cdr	stream-rest
'()	empty-stream
null?	stream-empty?
map / filter	stream-map / stream-filter

```
(define (stream-take s n)

(cond ((zero? n) '())

((stream-empty? s)' ())

(else (cons (stream-first s)

(stream-take (stream-rest s) (- n 1))))))

(define (make-naturals k)

(stream-cons k (make-naturals (add1 k))))

(define naturals-stream (make-naturals 0))

(stream-take naturals-stream 4)
```

stream-take (0, gen0) 3 = (cons (stream-first (0, gen0))

52 1, map
$$(f, 5)$$
 $f(x) = x^4 \lambda$
2) 52 1, $f(x_0)$, $f(x_1)$, $f(x_2)$... $x_0 = 1$
521, $f(1)$, $f(x_1)$,... $x_1 = 1$
521, λ , $f(x_1)$,... $x_1 = 1$
521, λ , $f(\lambda)$,...

) 5 = x0, x1, x2, ... /map(f,5)= f(x0), f(x1), f(x2)....

Exercitü

035: Cánd gluviam sheam-wi su gândim cum am face pt. liste /

 $1 - \frac{1}{3} = \frac{1}{5} - \frac{1}{7} \dots$

1) explicit =>] m: 1 3 5 7 3 la fierare pas: m+2 - sign: 1-1 1 -1 -1 - sign

hint: mamed let with 2 params (m, sign)

2 implicit

2) paten plea de la Johnsmel no. impare 2) sheam-filler Sheamel 1-1 1-1... 2) regrat

hint: sheam - zip -with

(2) ×, (f, x), (f, 2), (f, 2)...

sham- 2'p -with

5, 2 x0, x1, x2... = f1, f2, f1, f2.... 5, 2 y0, y1, y2... = y0, map(?, 52)

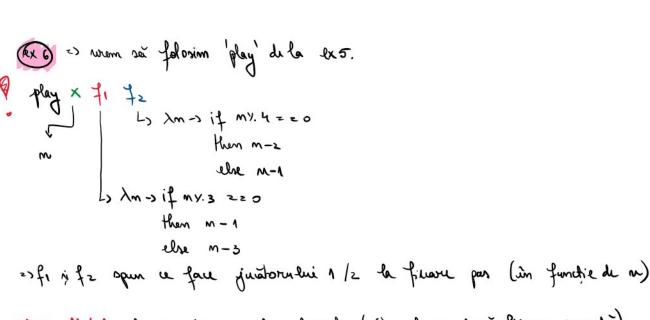
»mu pulem folosi

522 Xo, (dream-zip-with f 5, 52")

apply fifz, ... Xo, xo,

abby (x0) (x1) (x2) (x3)

=> 52 = sheam-map list 52



52: 2,1,2,1,2...

Ne dorim ca din 53 sã exhagem numai poudile in com m; ∠0

≥> struem-felor ≥> 54

1) Ne dorin ca din su na exhagem numai prima pouche (mi, juston), dar din avanta pouche, il exhagem pe juvator (con este 1 san 2)

$$5_1: M_1, M_2, \dots, M_{2_1}$$
 $5_2: z_1, M_2, \dots, M_{2_{n-1}}$
 $5_2: z_1, M_2, \dots, M_{2_{n-1}}$
 $5_2: x_1, M_2, \dots, M_{2_{n-1}}$
 $5_1: X_1, X_2, \dots, X_{n-1}$
 $5_1: X_1, X_1, \dots, X_{n-1}$



