

# Функции от по-висок ред

Трифон Трифонов

Функционално програмиране, 2017/18 г.

26 октомври – 2 ноември 2017 г.

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- `(define (fixed-point? f x) (= (f x) x))`

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- `(define (fixed-point? f x) (= (f x) x))`
- `(fixed-point? sin 0) → ?`

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- `(define (fixed-point? f x) (= (f x) x))`
- `(fixed-point? sin 0) → #t`

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- `(define (fixed-point? f x) (= (f x) x))`
- `(fixed-point? sin 0) → #t`
- `(fixed-point? exp 1) → ?`

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- `(define (fixed-point? f x) (= (f x) x))`
- `(fixed-point? sin 0) → #t`
- `(fixed-point? exp 1) → #f`

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- `(define (fixed-point? f x) (= (f x) x))`
- `(fixed-point? sin 0) → #t`
- `(fixed-point? exp 1) → #f`
- `(fixed-point? expt 0) → ?`

*x y*

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- `(define (fixed-point? f x) (= (f x) x))`
- `(fixed-point? sin 0) → #t`
- `(fixed-point? exp 1) → #f`
- `(fixed-point? expt 0) → Грешка!`

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- `(define (fixed-point? f x) (= (f x) x))`
- `(fixed-point? sin 0) → #t`
- `(fixed-point? exp 1) → #f`
- `(fixed-point? expt 0) → Грешка!`
- `(define (branch p? f g x) (if (p? x) (f x) (g x)))`

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- (`define` (fixed-point? f x) (= (f x) x))
- (fixed-point? sin 0) → #t
- (fixed-point? exp 1) → #f
- (fixed-point? expt 0) → Грешка!
- (`define` (branch p? f g x) (if (p? x) (f x) (g x)))
- (branch odd? exp fact 4) → ?

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- (`define` (fixed-point? f x) (= (f x) x))
- (fixed-point? sin 0) → #t
- (fixed-point? exp 1) → #f
- (fixed-point? expt 0) → Грешка!
- (`define` (branch p? f g x) (if (p? x) (f x) (g x)))
- (branch odd? exp fact 4) → 24

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- (`define` (fixed-point? f x) (= (f x) x))
- (fixed-point? sin 0) → #t
- (fixed-point? exp 1) → #f
- (fixed-point? expt 0) → Грешка!
- (`define` (branch p? f g x) (if (p? x) (f x) (g x)))
- (branch odd? exp fact 4) → 24
- (`define` (id x) x)

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- (`define` (fixed-point? f x) (= (f x) x))
- (fixed-point? sin 0) → #t
- (fixed-point? exp 1) → #f
- (fixed-point? expt 0) → Грешка!
- (`define` (branch p? f g x) (if (p? x) (f x) (g x)))
- (branch odd? exp fact 4) → 24
- (`define` (id x) x)
- (branch number? log id "1") → ?

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- (`define (fixed-point? f x) (= (f x) x))`
- (`(fixed-point? sin 0)` → #t
- (`(fixed-point? exp 1)` → #f
- (`(fixed-point? expt 0)` → Грешка!
- (`define (branch p? f g x) (if (p? x) (f x) (g x)))`
- (`(branch odd? exp fact 4)` → 24
- (`define (id x) x)`
- (`(branch number? log id "1")` → "1"

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

**Примери:**

- (`define` (fixed-point? f x) (= (f x) x))
- (fixed-point? sin 0) → #t
- (fixed-point? exp 1) → #f
- (fixed-point? expt 0) → Грешка!
- (`define` (branch p? f g x) (if (p? x) (f x) (g x)))
- (branch odd? exp fact 4) → 24
- (`define` (id x) x)
- (branch number? log id "1") → "1"
- (branch string? number? procedure? symbol?) → ?

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

Примери:

- (`define (fixed-point? f x) (= (f x) x))`
- (`(fixed-point? sin 0)` → #t
- (`(fixed-point? exp 1)` → #f
- (`(fixed-point? expt 0)` → Грешка!
- (`define (branch p? f g x) (if (p? x) (f x) (g x)))`
- (`(branch odd? exp fact 4)` → 24
- (`define (id x) x)`
- (`(branch number? log id "1")` → "1"
- (`(branch string? number? procedure? symbol?)` → #t

# Подаване на функции като параметри

В Scheme функциите са “първокласни” стойности.

Примери:

- (`define (fixed-point? f x) (= (f x) x))`
- (`(fixed-point? sin 0)` → #t
- (`(fixed-point? exp 1)` → #f
- (`(fixed-point? expt 0)` → Грешка!
- (`define (branch p? f g x) ((if (p? x) f g) x))`
- (`(branch odd? exp fact 4)` → 24
- (`define (id x) x)`
- (`(branch number? log id "1")` → "1"
- (`(branch string? number? procedure? symbol?)` → #t

# Функции от по-висок ред

## Дефиниция

Функция, която приема функция за параметър се нарича *функция от по-висок ред*.

# Функции от по-висок ред

## Дефиниция

Функция, която приема функция за параметър се нарича *функция от по-висок ред*.

- `fixed-point?` и `branch` са функции от по-висок ред

# Функции от по-висок ред

$$\frac{\partial}{\partial r} : (\mathbb{R} \rightarrow \mathbb{R}) \rightarrow (\mathbb{R} \rightarrow \mathbb{R}) \quad \int dx : \mathbb{R} \rightarrow \mathbb{R} \rightarrow (\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R}$$

## Дефиниция

Функция, която приема функция за параметър се нарича *функция от по-висок ред*.

- fixed-point? и branch са функции от по-висок ред
- Примери за математически функции от по-висок ред?

$$\int dx : (\mathbb{R} \rightarrow \mathbb{R}) \rightarrow (\mathbb{R} \rightarrow (\mathbb{R} \rightarrow \mathbb{R})) \int x^2 dx = \frac{x^3}{3} + C$$

$$h_a n^b : \mathbb{N} \rightarrow \mathbb{R}$$

# Функции от по-висок ред

## Дефиниция

Функция, която приема функция за параметър се нарича *функция от по-висок ред*.

- fixed-point? и branch са функции от по-висок ред
- Примери за математически функции от по-висок ред?
- Всички функции в  $\lambda$ -смятането са от по-висок ред!

# Задачи за сумиране

**Задача:** Да се пресметнат следните суми:

- ①  $k^2 + (k+1)^2 + \dots + 100^2$  за  $k \leq 100$
- ②  $\int_a^b f(x) dx \approx \Delta x [f(a) + f(a + \Delta x) + f(a + 2\Delta x) + \dots + f(b)]$
- ③  $x + e^x + e^{e^x} + e^{e^{e^x}} + \dots$  докато поредното събираме  $e \leq 10^{1000}$



## Задачи за сумиране

**Задача:** Да се пресметнат следните суми:

①  $k^2 + (k+1)^2 + \dots + 100^2$  за  $k \leq 100$

②  $\int_a^b f(x) \approx \Delta x [f(a) + f(a + \Delta x) + f(a + 2\Delta x) + \dots + f(b)]$

③  $x + e^x + e^{e^x} + e^{e^{e^x}} + \dots$  докато поредното събираме  $e \leq 10^{1000}$

```
(define (sum1 k)
  (if (> k 100) 0 (+ (* k k) (sum1 (+ k 1)))))
```

# Задачи за сумиране

**Задача:** Да се пресметнат следните суми:

$$\textcircled{1} \quad k^2 + (k+1)^2 + \dots + 100^2 \text{ за } k \leq 100$$

$$\textcircled{2} \quad \int_a^b f(x) \approx \Delta x [f(a) + f(a + \Delta x) + f(a + 2\Delta x) + \dots + f(b)]$$

$$\textcircled{3} \quad x + e^x + e^{e^x} + e^{e^{e^x}} + \dots \text{ докато поредното събираме } e \leq 10^{1000}$$

```
(define (sum1 k)
  (if (> k 100) 0 (+ (* k k) (sum1 (+ k 1)))))
```

```
(define (sum2 a b f dx)
  (if (> a b) 0 (+ (* dx (f a)) (sum2 (+ a dx) b f dx))))
```



$$\int_a^b f(y) dy \approx f(a) \Delta x + \int_{a+\Delta x}^b f(x) dx$$

# Задачи за сумиране

**Задача:** Да се пресметнат следните суми:

$$\textcircled{1} \quad k^2 + (k+1)^2 + \dots + 100^2 \text{ за } k \leq 100$$

$$\textcircled{2} \quad \int_a^b f(x) \approx \Delta x [f(a) + f(a + \Delta x) + f(a + 2\Delta x) + \dots + f(b)]$$

$$\textcircled{3} \quad x + e^x + e^{e^x} + e^{e^{e^x}} + \dots \text{ докато поредното събираме } e \leq 10^{1000}$$

(define (sum1 k) *y := ex*   *x + y + ey + ...*)

```
(if (> k 100) 0 (+ (* k k) (sum1 (+ k 1))))
```

(define (sum2 a b f dx)

```
(if (> a b) 0 (+ (* dx (f a)) (sum2 (+ a dx) b f dx))))
```

(define (sum3 x)

```
(if (> x (expt 10 1000)) 0 (+ x (sum3 (exp x)))))
```

# Задачи за сумиране

**Задача:** Да се пресметнат следните суми:

- ①  $k^2 + (k+1)^2 + \dots + 100^2$  за  $k \leq 100$

- ②  $\int_a^b f(x) \approx \Delta x [f(a) + f(a + \Delta x) + f(a + 2\Delta x) + \dots + f(b)]$

- ③  $x + e^x + e^{e^x} + e^{e^{e^x}} + \dots$  докато поредното събираме  $e \leq 10^{1000}$

```
(define (sum1 k)
  (if (> k 100) 0 (+ (* k k) (sum1 (+ k 1)))))

(define (sum2 a b f dx)
  (if (> a b) 0 (+ (* dx (f a)) (sum2 (+ a dx) b f dx)))))

(define (sum3 x)
  (if (> x (expt 10 1000)) 0 (+ x (sum3 (exp x)))))
```

# Задачи за сумиране

**Задача:** Да се пресметнат следните суми:

- ①  $k^2 + (k+1)^2 + \dots + 100^2$  за  $k \leq 100$

- ②  $\int_a^b f(x) \approx \Delta x [f(a) + f(a + \Delta x) + f(a + 2\Delta x) + \dots + f(b)]$

- ③  $x + e^x + e^{e^x} + e^{e^{e^x}} + \dots$  докато поредното събираме  $e \leq 10^{1000}$

```
(define (sum1 k)
  (if (> k 100) 0 (+ (* k k) (sum1 (+ k 1)))))

(define (sum2 a b f dx)
  (if (> a b) 0 (+ (* dx (f a)) (sum2 (+ a dx) b f dx)))))

(define (sum3 x)
  (if (> x (expt 10 1000)) 0 (+ x (sum3 (exp x)))))
```

# Задачи за сумиране

**Задача:** Да се пресметнат следните суми:

- ①  $k^2 + (k+1)^2 + \dots + 100^2$  за  $k \leq 100$

- ②  $\int_a^b f(x) \approx \Delta x [f(a) + f(a + \Delta x) + f(a + 2\Delta x) + \dots + f(b)]$

- ③  $x + e^x + e^{e^x} + e^{e^{e^x}} + \dots$  докато поредното събираме  $e \leq 10^{1000}$

```
(define (sum1 k)
  (if (> k 100) 0 (+ (* k k) (sum1 (+ k 1)))))

(define (sum2 a b f dx)
  (if (> a b) 0 (+ (* dx (f a)) (sum2 (+ a dx) b f dx)))))

(define (sum3 x)
  (if (> x (expt 10 1000)) 0 (+ x (sum3 (exp x)))))
```

# Обобщена функция за сумиране

Да се напише функция от по-висок ред `sum`, която пресмята сумата:

$$\sum_{\substack{i=a \\ i \leftarrow \text{next}(i)}}^b \text{term}(i).$$

# Обобщена функция за сумиране

Да се напише функция от по-висок ред sum, която пресмята сумата:

$$\sum_{\substack{i=a \\ i \leftarrow \text{next}(i)}}^b \text{term}(i).$$

```
(define (sum a b term next)
  (if (> a b) 0 (+ (term a) (sum (next a) b term next))))
```

## Приложения на sum

Решение на задачите за суми чрез sum:

$$\sum_{i=k}^{100} i^2$$

## Приложения на sum

Решение на задачите за суми чрез sum:

$$\sum_{i=k}^{100} i^2$$

```
(define (square x) (* x x))
(define (1+ x) (+ x 1))
(define (sum1 k) (sum k 100 square 1+))
```

# Приложения на sum

Решение на задачите за суми чрез sum:

$$\sum_{i=k}^{100} i^2$$

```
(define (square x) (* x x))
(define (1+ x) (+ x 1))
(define (sum1 k) (sum k 100 square 1+))
```

$$\sum_{\substack{i=a \\ i \rightarrow i + \Delta x}}^b \Delta x f(i)$$

(Sum a b term next)  
 ↓      ↓      i → Δx f(i)      Δx  
 a      b

## Приложения на sum

Решение на задачите за суми чрез sum:

$$\sum_{i=k}^{100} i^2$$

```
(define (square x) (* x x))
(define (1+ x) (+ x 1))
(define (sum1 k) (sum k 100 square 1+))
```

$$\sum_{\substack{i=a \\ i \rightarrow i + \Delta x}}^b \Delta x f(i)$$

```
(define (sum2 a b f dx)
  (define (term x) (* dx (f x)))
  (define (next x) (+ x dx))
  (sum a b term next))
```

# Приложения на sum

Решение на задачите за суми чрез sum:

$$\sum_{i=k}^{100} i^2$$

```
(define (square x) (* x x))
(define (1+ x) (+ x 1))
(define (sum1 k) (sum k 100 square 1+))
```

$$\Delta x \sum_{\substack{i=a \\ i \rightarrow i + \Delta x}}^b f(i)$$

```
(define (sum2 a b f dx)
  (define (next x) (+ x dx))
  (* dx (sum a b f next)))
```

# Приложения на sum

Решение на задачите за суми чрез sum:

$$\sum_{i=k}^{100} i^2$$

```
(define (square x) (* x x))
(define (1+ x) (+ x 1))
(define (sum1 k) (sum k 100 square 1+))
```

$$\Delta x \sum_{\substack{i=a \\ i \rightarrow i+\Delta x}}^b f(i)$$

```
(define (sum2 a b f dx)
  (define (next x) (+ x dx))
  (* dx (sum a b f next)))
```

$$\sum_{\substack{i=x \\ i \rightarrow e^i}}^{10^{1000}} i$$

(sum  $a$   $b$  term next)

# Приложения на sum

Решение на задачите за суми чрез sum:

$$\sum_{i=k}^{100} i^2$$

```
(define (square x) (* x x))
(define (1+ x) (+ x 1))
(define (sum1 k) (sum k 100 square 1+))
```

$$\Delta x \sum_{\substack{i=a \\ i \rightarrow i+\Delta x}}^b f(i)$$

```
(define (sum2 a b f dx)
  (define (next x) (+ x dx))
  (* dx (sum a b f next)))
```

$$\sum_{\substack{i=x \\ i \rightarrow e^i}}^{10^{1000}} i$$

```
(define (sum3 x)
  (sum x (expt 10 1000) id exp))
```

# Обобщена функция за произведение

Да се напише функция от по-висок ред `product`, която пресмята:

$$\prod_{\substack{i=a \\ i \leftarrow next(i)}}^b term(i).$$

# Обобщена функция за произведение

Да се напише функция от по-висок ред product, която пресмята:

$$\prod_{\substack{i=a \\ i \leftarrow \text{next}(i)}}^b \text{term}(i).$$

```
(define (prod a b term next)
  (if (> a b) 1 (* (term a) (prod (next a) b term next))))
```

# Обобщена функция за произведение

Да се напише функция от по-висок ред product, която пресмята:

$$\prod_{\substack{i=a \\ i \leftarrow \text{next}(i)}}^b \text{term}(i).$$

```
(define (prod a b term next)
  (if (> a b) 1 (* (term a) (prod (next a) b term next)))))

(define (sum a b term next)
  (if (> a b) 0 (+ (term a) (sum (next a) b term next))))
```

# Обобщена функция за произведение

Да се напише функция от по-висок ред product, която пресмята:

$$\prod_{\substack{i=a \\ i \leftarrow \text{next}(i)}}^b \text{term}(i).$$

```
(define (prod a b term next)
  (if (> a b) [1] ([*] (term a) (prod (next a) b term next)))))

(define (sum a b term next)
  (if (> a b) [0] ([+] (term a) (sum (next a) b term next))))
```

# Обобщена функция за натрупване

Да се напише функция, която пресмята

$$\underbrace{\text{term}(a)}_{u} \oplus \left( \text{term}(\text{next}(a)) \oplus \left( \dots \oplus (\underbrace{\text{term}(b)}_{\text{term}(i)} \oplus \perp) \dots \right) \right),$$

където  $\oplus$  е бинарна операция,  
а  $\perp$  е нейната “нулева стойност”, т.e.  $x \oplus \perp = x$ .

Чθ V  
 ↙ ↘  
 ↘ ↗  
 term(i)

# Обобщена функция за натрупване

Да се напише функция, която пресмята

$$\text{term}(a) \oplus \left( \text{term}(\text{next}(a)) \oplus \left( \dots \oplus (\text{term}(b) \oplus \perp) \dots \right) \right),$$

където  $\oplus$  е бинарна операция,

а  $\perp$  е нейната "нулева стойност", т.е.  $x \oplus \perp = x$ .

```
(define (accumulate op nv a b term next)
  (if (> a b) nv
    (op (term a) (accumulate op nv (next a) b term next))))
```

# Обобщена функция за натрупване

Да се напише функция, която пресмята

$$\text{term}(a) \oplus \left( \text{term}(\text{next}(a)) \oplus \left( \dots \oplus (\text{term}(b) \oplus \perp) \dots \right) \right),$$

където  $\oplus$  е бинарна операция,

а  $\perp$  е нейната "нулева стойност", т.e.  $x \oplus \perp = x$ .

```
(define (accumulate op nv a b term next)
  (if (> a b) nv
      (op (term a) (accumulate op nv (next a) b term next)))))

(define (sum a b term next) (accumulate + 0 a b term next))
(define (product a b term next) (accumulate * 1 a b term next))
```

## Задача: пресмятане на полином

Да се пресметне стойността на полинома

$$P_n(x) = x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1)$$

## Задача: пресмятане на полином

Да се пресметне стойността на полинома

$$\begin{aligned}P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\&= \sum_{i=0}^n (n+1-i)x^i\end{aligned}$$

## Задача: пресмятане на полином

Да се пресметне стойността на полинома

$$\begin{aligned}P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\&= \sum_{i=0}^n (n+1-i)x^i\end{aligned}$$

Решение №1:

```
(define (p n x)
```

```
(accumulate + ? ? ? ? ?))
```

## Задача: пресмятане на полином

Да се пресметне стойността на полинома

$$\begin{aligned}P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\&= \sum_{i=0}^n (n+1-i)x^i\end{aligned}$$

Решение №1:

```
(define (p n x)
```

```
(accumulate + 0 ? ? ? ?))
```

## Задача: пресмятане на полином

Да се пресметне стойността на полинома

$$\begin{aligned}P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\&= \sum_{i=0}^n (n+1-i)x^i\end{aligned}$$

Решение №1:

```
(define (p n x)
```

```
  (accumulate + 0 0 ? ? ?))
```

## Задача: пресмятане на полином

Да се пресметне стойността на полинома

$$\begin{aligned}P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\&= \sum_{i=0}^n (n+1-i)x^i\end{aligned}$$

Решение №1:

```
(define (p n x)
```

```
(accumulate + 0 0 n ? ?))
```

## Задача: пресмятане на полином

Да се пресметне стойността на полинома

$$\begin{aligned}P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\&= \sum_{i=0}^n (n+1-i)x^i\end{aligned}$$

Решение №1:

```
(define (p n x)
  (define (term i) (* (- (1+ n) i) (expt x i)))
  (accumulate + 0 0 n term ?))
```

## Задача: пресмятане на полином

Да се пресметне стойността на полинома

$$\begin{aligned}P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\&= \sum_{i=0}^n (n+1-i)x^i\end{aligned}$$

Решение №1:

```
(define (p n x)
  (define (term i) (* (- (1+ n) i) (expt x i)))
  (accumulate + 0 0 n term 1+))
```

## Задача: пресмятане на полином

Да се пресметне стойността на полинома

$$\begin{aligned}P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\&= \sum_{i=0}^n (n+1-i)x^i\end{aligned}$$

Решение №1:

```
(define (p n x)
  (define (term i) (* (- (1+ n) i) (expt x i)))
  (accumulate + 0 0 n term 1+))
```

Можем ли да решим задачата без да извикваме expt на всяка стъпка?

# Правило на Хорнер

$$\begin{aligned}P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\&= \left( \left( \left( \dots \left( (x+2)x + 3 \right) x + \dots \right) x + (n-1) \right) x + n \right) x + (n+1)\end{aligned}$$

# Правило на Хорнер

$$\begin{aligned}P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\&= \left( \left( \left( \dots ((x+2)x+3)x+\dots \right)x+(n-1) \right)x+n \right)x+(n+1)\end{aligned}$$

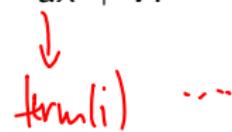
Можем ли да сметнем с accumulate?

# Правило на Хорнер

$$\begin{aligned}
 P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\
 &= \left( \left( \left( \dots ((x+2)x+3)x+\dots \right)x+(n-1) \right)x+n \right)x+(n+1)
 \end{aligned}$$

Можем ли да сметнем с accumulate?

Идея: Да използваме операцията  $u \oplus v := ux + v$ .


  
 term(i) ...


  
 1.x + 2.x + 3.x + ...

# Правило на Хорнер

$$\begin{aligned}
 P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\
 &= \left( \left( \left( \dots ((x+2)x+3)x+\dots \right)x+(n-1) \right)x+n \right)x+(n+1)
 \end{aligned}$$

Можем ли да сметнем с accumulate?

Идея: Да използваме операцията  $u \oplus v := ux + v$ .

Коя е "нулевата стойност"  $\perp$ ?

$$\perp \oplus 1$$

$$\perp \oplus v = v$$

# Правило на Хорнер

$$\begin{aligned}
 P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\
 &= \left( \left( \left( \dots ((x+2)x+3)x+\dots \right)x+(n-1) \right)x+n \right)x+(n+1)
 \end{aligned}$$

Можем ли да сметнем с accumulate?

**Идея:** Да използваме операцията  $u \oplus v := ux + v$ .

Коя е “нулевата стойност”  $\perp$ ?

**Решение №2:**

```
(define (p n x)
  (define (op u v) (+ (* u x) v))
  (accumulate op 0 1 (1+ n) id 1+))
```

# Правило на Хорнер

$$\begin{aligned}
 P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\
 &= \left( \left( \left( \dots ((x+2)x+3)x+\dots \right)x+(n-1) \right)x+n \right)x+(n+1)
 \end{aligned}$$

Можем ли да сметнем с accumulate?

**Идея:** Да използваме операцията  $u \oplus v := ux + v$ .

Коя е “нулевата стойност”  $\perp$ ?

**Решение №2:**

```
(define (p n x)
  (define (op u v) (+ (* u x) v))
  (accumulate op 0 1 (1+ n) id 1+))
```

Не смята правилно!

# Правило на Хорнер

Всъщност пресметнахме:

$$Q_n(x) = x + 2x + 3x + \dots + nx + (n+1)x = \frac{(n+1)(n+2)}{2}x.$$

$$\textcircled{4} := u \cdot x + v \quad 1 \cdot x + (2 \cdot x + (-\dots \dots))$$

$fun := id$

$u := 1$

# Правило на Хорнер

Всъщност пресметнахме:

$$Q_n(x) = x + 2x + 3x + \dots + nx + (n+1)x = \frac{(n+1)(n+2)}{2}x.$$

**Идея:** Да използваме операцията  $u \oplus v := u + vx$ .

$$1 + x(2 + x(\dots + x(n + x(1 + x(0)))))$$

↓  
term(i) (---)

# Правило на Хорнер

Всъщност пресметнахме:

$$Q_n(x) = x + 2x + 3x + \dots + nx + (n+1)x = \frac{(n+1)(n+2)}{2}x.$$

**Идея:** Да използваме операцията  $u \oplus v := u + vx$ .

**Решение №3:**

```
(define (p n x)
  (define (op u v) (+ u (* v x)))
  (accumulate op 0 1 (1+ n) id 1+))
```

# Правило на Хорнер

Всъщност пресметнахме:

$$Q_n(x) = x + 2x + 3x + \dots + nx + (n+1)x = \frac{(n+1)(n+2)}{2}x.$$

**Идея:** Да използваме операцията  $u \oplus v := u + vx$ .

**Решение №3:**

```
(define (p n x)
  (define (op u v) (+ u (* v x)))
  (accumulate op 0 1 (1+ n) id 1+))
```

1 + (2 + (3 + ...)) \* x

Пак не смята правилно!!!

## Ляво и дясно натрупване

Всъщност пресметнахме:

$$\begin{aligned} R_n(x) &= 1 + x \left( 2 + x \left( \dots + x \left( (n-1) + x(n+x(n+1)) \right) \dots \right) \right) \\ &= (n+1)x^n + nx^{n-1} + (n-1)x^{n-2} + \dots + 3x^2 + 2x + 1 \end{aligned}$$

вместо

$$\begin{aligned} P_n(x) &= \left( \left( \left( \dots ((x+2)x+3)x+\dots \right)x+(n-1) \right)x+n \right)x+(n+1) \\ &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1). \end{aligned}$$

## Ляво и дясно натрупване

Всъщност пресметнахме:

$$(5/3)/2 \neq 5/(3/2)$$

$$\begin{aligned} R_n(x) &= 1 + x \left( 2 + x \left( \dots + x \left( (n-1) + x(n+x(n+1)) \right) \dots \right) \right) \\ &= (n+1)x^n + nx^{n-1} + (n-1)x^{n-2} + \dots + 3x^2 + 2x + 1 \end{aligned}$$

вместо

$$\begin{aligned} P_n(x) &= \left( \left( \left( \dots ((x+2)x+3)x + \dots \right) x + (n-1) \right) x + n \right) x + (n+1) \\ &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1). \end{aligned}$$

За неассоциативни операции  $\oplus$  има значение в какъв ред са скобите!

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C$$

# Обобщена функция за ляво натрупване

Да се напише функция, която пресмята ляво натрупване: ✓

$$\left( \dots \left( (\perp \oplus \text{term}(a)) \oplus \text{term}(\text{next}(a)) \right) \oplus \dots \right) \oplus \text{term}(b)$$

$$( \perp \oplus \text{term}(u), \dots \dots \oplus \dots \oplus \text{term}(u) )$$

u⊕✓

## Обобщена функция за ляво натрупване

Да се напише функция, която пресмята **ляво натрупване**:

$$\left( \dots \left( (\perp \oplus term(a)) \oplus term(next(a)) \right) \oplus \dots \right) \oplus term(b)$$

```
(define (accumulate-i op nv a b term next)
  (if (> a b) nv
      (accumulate-i op (op nv (term a)) (next a) b term next)))
```

# Обобщена функция за ляво натрупване

Да се напише функция, която пресмята **ляво натрупване**:

$$\left( \dots \left( (\perp \oplus term(a)) \oplus term(next(a)) \right) \oplus \dots \right) \oplus term(b)$$

```
(define (accumulate-i op nv a b term next)
  (if (> a b) nv
      (accumulate-i op (op nv (term a)) (next a) b term next)))
```

- accumulate — дясно натрупване, рекурсивен процес
- accumulate-i — ляво натрупване, итеративен процес

# Правило на Хорнер

$$\begin{aligned}
 P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\
 &= \left( \left( \left( \dots ((x+2)x+3)x + \dots \right) x + (n-1) \right) x + n \right) x + (n+1)
 \end{aligned}$$

Идея: използваме `accumulate`-и и  $u \oplus v := ux + v$ .

$$\left( \left( \left( \dots \left( \underbrace{(0x+1)x+2}_{u} \right) \dots \right) x + n \right) x + n+1 \right)$$

# Правило на Хорнер

$$\begin{aligned}
 P_n(x) &= x^n + 2x^{n-1} + \dots + (n-2)x^3 + (n-1)x^2 + nx + (n+1) \\
 &= \left( \left( \left( \dots ((x+2)x+3)x + \dots \right) x + (n-1) \right) x + n \right) x + (n+1)
 \end{aligned}$$

Идея: използваме accumulate-и и  $u \oplus v := ux + v$ .

## Решение №4:

```
(define (p n x)
  (define (op u v) (+ (* u x) v))
  (accumulate-i op 0 1 (1+ n) id 1+))
```

## Анонимни функции

Можем ли да ги конструираме параметрите на функциите от по-висок ред “на място”, без да даваме имена?

## Анонимни функции

Можем ли да ги конструираме параметрите на функциите от по-висок ред "на място", без да даваме имена?

- `(lambda ( {<параметър>} ) <тяло>)`

## Анонимни функции

Можем ли да ги конструираме параметрите на функциите от по-висок ред "на място", без да даваме имена?

- `(lambda ( {<параметър>} ) <тяло>)`
- Оценява се до функционален обект със съответните параметри и тяло

## Анонимни функции

Можем ли да ги конструираме параметрите на функциите от по-висок ред "на място", без да даваме имена?

- `(lambda ( {<параметър>} ) <тяло>)`
- Оценява се до функционален обект със съответните параметри и тяло
- Анонимната функция пази указател към средата, в която е оценена

## Анонимни функции

Можем ли да ги конструираме параметрите на функциите от по-висок ред "на място", без да даваме имена?

- **(lambda ( {<параметър>} ) <тяло>)**
- Оценява се до функционален обект със съответните параметри и тяло
- Анонимната функция пази указател към средата, в която е оценена
- Примери:

## Анонимни функции

Можем ли да ги конструираме параметрите на функциите от по-висок ред "на място", без да даваме имена?

- **(lambda ( {<параметър>} ) <тяло>)**
- Оценява се до функционален обект със съответните параметри и тяло
- Анонимната функция пази указател към средата, в която е оценена
- Примери:
  - `(lambda (x) (+ x 3))` → #<procedure>

## Анонимни функции

Можем ли да ги конструираме параметрите на функциите от по-висок ред "на място", без да даваме имена?

- **(lambda ( {<параметър>} ) <тяло>)**
- Оценява се до функционален обект със съответните параметри и тяло
- Анонимната функция пази указател към средата, в която е оценена
- Примери:
  - `(lambda (x) (+ x 3))` → #<procedure>
  - `((lambda (x) (+ x 3)) 5)` → 8

## Анонимни функции

Можем ли да ги конструираме параметрите на функциите от по-висок ред "на място", без да даваме имена?

- **(lambda ( {<параметър>} ) <тяло>)**
- Оценява се до функционален обект със съответните параметри и тяло
- Анонимната функция пази указател към средата, в която е оценена
- Примери:
  - `(lambda (x) (+ x 3))` → #<procedure>
  - `((lambda (x) (+ x 3)) 5)` → 8
  - `(define (<име> <параметри>) <тяло>)`  
↔  
`(define <име> (lambda (<параметри>) <тяло>))`

## Примери

```
(define (integral a b f dx)
  (* dx (accumulate + 0 a b f (lambda (x) (+ x dx)))))
```

## Примери

```
(define (integral a b f dx)
  (* dx (accumulate + 0 a b f (lambda (x) (+ x dx)))))

(define (p n x)
  (accumulate-i + 0 1 (+ n 1) (lambda (u v) (+ (* u x) v))
                (lambda (i) (+ i 1))))
```

# Примери

```
(define (integral a b f dx)
  (* dx (accumulate + 0 a b f (lambda (x) (+ x dx)))))

(define (p n x)
  (accumulate-i + 0 1 (+ n 1) (lambda (u v) (+ (* u x) v))
                (lambda (i) (+ i 1))))
```

**Задача:** Как можем да реализираме с accumulate:

- $n!$

$$1 \cdot 2 \cdot 3 \cdots h$$

$$\prod_{i=1}^n ?$$

## Примери

```
(define (integral a b f dx)
  (* dx (accumulate + 0 a b f (lambda (x) (+ x dx)))))

(define (p n x)
  (accumulate-i + 0 1 (+ n 1) (lambda (u v) (+ (* u x) v))
                (lambda (i) (+ i 1))))
```

**Задача:** Как можем да реализираме с accumulate:

- $n!$
- $x^n$

X V, ... X

## Примери

$$\underline{u} \oplus \underline{v} = \underline{1} + \underline{uv}$$

```
(define (integral a b f dx)
  (* dx (accumulate + 0 a b f (lambda (x) (+ x dx)))))

(define (p n x)
  (accumulate-i + 0 1 (+ n 1) (lambda (u v) (+ (* u x) v))
                (lambda (i) (+ i 1))))
```

**Задача:** Как можем да реализираме с accumulate:

- $n!$

- $x^n$

$$\begin{aligned}
 & \bullet \sum_{i=0}^n \frac{x^i}{i!} \approx e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!} = \checkmark \\
 & = 1 + \left( \frac{x}{1} \right) \left( 1 + \frac{x}{2} \right) \left( 1 + \frac{x}{3} \right) \left( 1 + \dots + \frac{x}{n} \right) (1+0) \cdots
 \end{aligned}$$

## Примери

$(\lambda x. (+ x)) \leftrightarrow f$

```
(define (integral a b f dx)
  (* dx (accumulate + 0 a b f (lambda (x) (+ x dx)))))

(define (p n x)
  (accumulate-i + 0 1 (+ n 1) (lambda (u v) (+ (* u x) v))
                (lambda (i) (+ i 1))))
```

**Задача:** Как можем да реализираме с accumulate:

- $n!$
- $x^n$
- $\sum_{i=0}^n \frac{x^i}{i!}$
- $\exists x \in [a; b] p(x)$   $\hookrightarrow p(1) \vee p(1+1) \vee \dots \vee p(16)$

## Функции, които връщат функции

Да разгледаме функция, която прилага дадена функция два пъти над аргумент.

- (define (twice f x) (f (f x)))

## Функции, които връщат функции

Да разгледаме функция, която прилага дадена функция два пъти над аргумент.

- (`define (twice f x) (f (f x)))`)
- (`(twice square 3)` → ?

## Функции, които връщат функции

Да разгледаме функция, която прилага дадена функция два пъти над аргумент.

- `(define (twice f x) (f (f x)))`
- `(twice square 3) —→ 81`

## Функции, които връщат функции

Да разгледаме функция, която прилага дадена функция два пъти над аргумент.

- `(define (twice f x) (f (f x)))` 
- `(twice square 3)` → 81
- `(define (twice f) (lambda (x) (f (f x))))`

## Функции, които връщат функции

Да разгледаме функция, която прилага дадена функция два пъти над аргумент.

- `(define (twice f x) (f (f x)))`
- `(twice square 3) → 81`
- `(define (twice f) (lambda (x) (f (f x))))`
- `(twice square 3) → ?`

## Функции, които връщат функции

Да разгледаме функция, която прилага дадена функция два пъти над аргумент.

- `(define (twice f x) (f (f x)))`
- `(twice square 3)` → 81
- `(define (twice f) (lambda (x) (f (f x))))`
- `(twice square 3)` → Грешка!

## Функции, които връщат функции

Да разгледаме функция, която прилага дадена функция два пъти над аргумент.

- `(define (twice f x) (f (f x)))`
- `(twice square 3)` → 81
- `(define (twice f) (lambda (x) (f (f x))))`
- `(twice square 3)` → Грешка!
- `(twice square)` → ?

## Функции, които връщат функции

Да разгледаме функция, която прилага дадена функция два пъти над аргумент.

- `(define (twice f x) (f (f x)))`
- `(twice square 3)` → 81
- `(define (twice f) (lambda (x) (f (f x))))`
- `(twice square 3)` → Грешка!
- `(twice square)` → #<procedure>

## Функции, които връщат функции

Да разгледаме функция, която прилага дадена функция два пъти над аргумент.

- `(define (twice f x) (f (f x)))`
- `(twice square 3)` → 81
- `(define (twice f) (lambda (x) (f (f x))))`
- `(twice square 3)` → Грешка!
- `(twice square)` → #<procedure>
- `((twice square) 3)` → 81

## Функции, които връщат функции

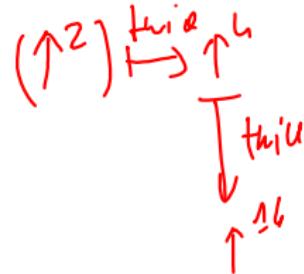
Да разгледаме функция, която прилага дадена функция два пъти над аргумент.

- (`define (twice f x) (f (f x)))`)
- (`(twice square 3)`) → 81
- (`define (twice f) (lambda (x) (f (f x))))`)
- (`(twice square 3)`) → Грешка!
- (`(twice square)`) → #<procedure>
- (`((twice square) 3)`) → 81
- (`((twice (twice square)) 2)`) → ?

# Функции, които връщат функции

Да разгледаме функция, която прилага дадена функция два пъти над аргумент.

- (define (twice f x) (f (f x)))
- (twice square 3) → 81
- (define (twice f) (lambda (x) (f (f x))))
- (twice square 3) → Грешка!
- (twice square) → #<procedure>
- ((twice square) 3) → 81
- ((twice (twice square)) 2) → 65536



## Примери

- (define (n+ n) (lambda (i) (+ i n)))

## Примери

- (`define` (n+ n) (`lambda` (i) (+ i n)))
- (`define` 1+ (n+ 1))

## Примери

- (`define` (n+ n) (`lambda` (i) (+ i n)))
- (`define` 1+ (n+ 1))
- (1+ 7) → 8

## Примери

- (`define` (n+ n) (`lambda` (i) (+ i n)))
- (`define` 1+ (n+ 1))
- (1+ 7) → 8
- (`define` 5+ (n+ 5))

## Примери

- (`define` (n+ n) (`lambda` (i) (+ i n)))
- (`define` 1+ (n+ 1))
- (1+ 7) → 8
- (`define` 5+ (n+ 5))
- (5+ 7) → 12

# Примери

- (`define` (n+ n) (`lambda` (i) (+ i n)))
- (`define` 1+ (n+ 1))
- (1+ 7) → 8
- (`define` 5+ (n+ 5))
- (5+ 7) → 12
- (`define` (compose f g) (`lambda` (x) (f (g x))))

f o g

## Примери

- (`define` (n+ n) (`lambda` (i) (+ i n)))
- (`define` 1+ (n+ 1))
- (1+ 7) → 8
- (`define` 5+ (n+ 5))
- (5+ 7) → 12
- (`define` (compose f g) (`lambda` (x) (f (g x))))
- ((compose square 1+) 3) → ?

# Примери

- (`define` (n+ n) (`lambda` (i) (+ i n)))
- (`define` 1+ (n+ 1))
- (1+ 7) → 8
- (`define` 5+ (n+ 5))
- (5+ 7) → 12
- (`define` (compose f g) (`lambda` (x) (f (g x))))
- ((compose square 1+) 3) → 16

$$\lambda_x. (1+x)^2$$

# Примери

- (`define` (n+ n) (`lambda` (i) (+ i n)))
- (`define` 1+ (n+ 1))
- (1+ 7) → 8
- (`define` 5+ (n+ 5))
- (5+ 7) → 12
- (`define` (compose f g) (`lambda` (x) (f (g x))))
- ((compose square 1+) 3) → 16
- ((compose 1+ square) 3) → ?

# Примери

- (`define` (n+ n) (`lambda` (i) (+ i n)))
- (`define` 1+ (n+ 1))
- (1+ 7) → 8
- (`define` 5+ (n+ 5))
- (5+ 7) → 12
- (`define` (compose f g) (`lambda` (x) (f (g x))))
- ((compose square 1+) 3) → 16
- ((compose 1+ square) 3) → 10

$$f \circ (g \circ h) = (f \circ g) \circ h$$

$\downarrow \quad \quad \quad \downarrow$

$$\lambda x \ f(g(h(x)))$$

$$\lambda x \ x^2 + 1$$

# Примери

- (`define` (n+ n) (`lambda` (i) (+ i n)))
- (`define` 1+ (n+ 1))
- (1+ 7) → 8
- (`define` 5+ (n+ 5))
- (5+ 7) → 12
- (`define` (compose f g) (`lambda` (x) (f (g x))))
- ((compose square 1+) 3) → 16
- ((compose 1+ square) 3) → 10
- ((compose 1+ (compose square (n+ 2))) 3) → ?

$$\lambda x \quad (x+2)^2 + 1$$

# Примери

- `(define (n+ n) (lambda (i) (+ i n)))`
- `(define 1+ (n+ 1))`
- $(1+ 7) \rightarrow 8$
- `(define 5+ (n+ 5))`
- $(5+ 7) \rightarrow 12$
- `(define (compose f g) (lambda (x) (f (g x))))`
- $((compose square 1+) 3) \rightarrow 16$
- $((compose 1+ square) 3) \rightarrow 10$
- $((compose 1+ (compose square (n+ 2))) 3) \rightarrow 26$

# Оценка на lambda

{E} `(define (n+ n)  
 (lambda (i) (+ i n)))`

E	
n+ :	Параметри : n Тяло : $(\lambda(i) (+ i n))$ Среда : E

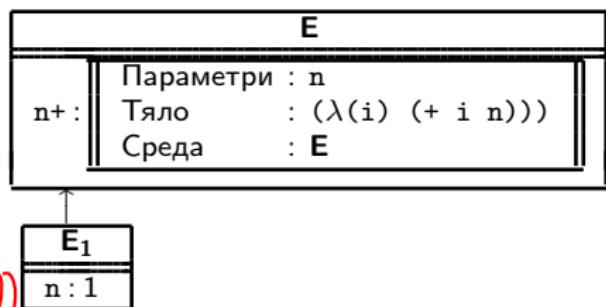
# Оценка на lambda

{E} `(define (n+ n)  
 (lambda (i) (+ i n)))`

{E} `(define 1+ (n+ 1))`



$E_1 \quad (\lambda(i) (+ i_n))$



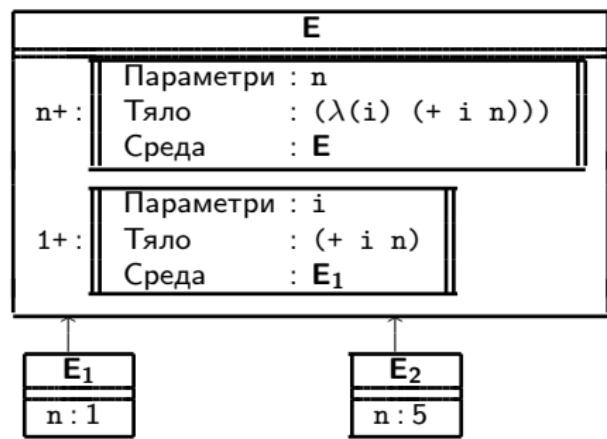
# Оценка на lambda

```
{E} (define (n+ n)
      (lambda (i) (+ i n)))
{E} (define 1+ (n+ 1))
```



# Оценка на lambda

```
{E} (define (n+ n)
      (lambda (i) (+ i n)))
{E} (define 1+ (n+ 1))
{E} (define 5+ (n+ 5))
```



# Оценка на lambda

```
{E} (define (n+ n)
      (lambda (i) (+ i n)))
{E} (define 1+ (n+ 1))
{E} (define 5+ (n+ 5))
```



# Оценка на lambda

```
{E} (define (n+ n)
{E}   (lambda (i) (+ i n)))
{E} (define 1+ (n+ 1))
{E} (define 5+ (n+ 5))
{E} (1+ 7)
```

E		
	Параметри : n	
n+ :	Тяло : ( $\lambda(i) (+ i n)$ )	
	Среда : E	
	Параметри : i	
1+ :	Тяло : (+ i n)	
	Среда : E <sub>1</sub>	
	Параметри : i	
5+ :	Тяло : (+ i n)	
	Среда : E <sub>2</sub>	

E <sub>1</sub>	E <sub>2</sub>
n : 1	n : 5

E<sub>1</sub>

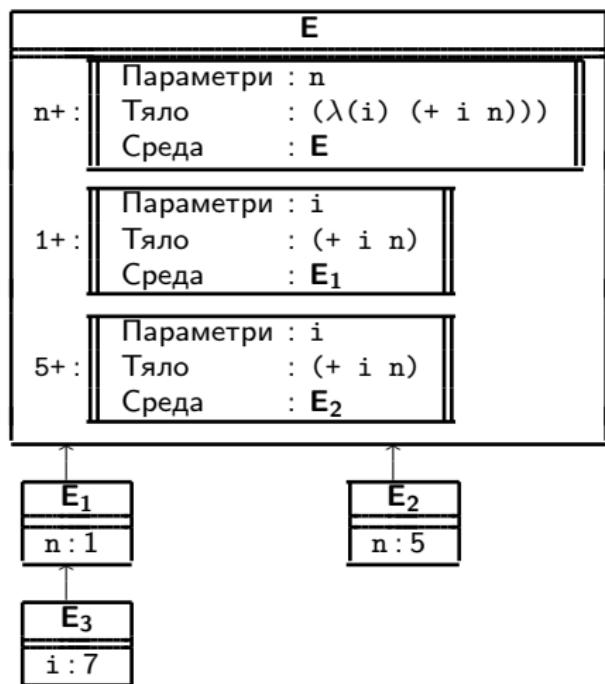
E <sub>2</sub>
n : 5

# Оценка на lambda

```

{E}   (define (n+ n)
{E}     (lambda (i) (+ i n)))
{E}   (define 1+ (n+ 1))
{E}   (define 5+ (n+ 5))
{E}           (1+ 7)
{E}           ↓
{E_3}          (+ i n)

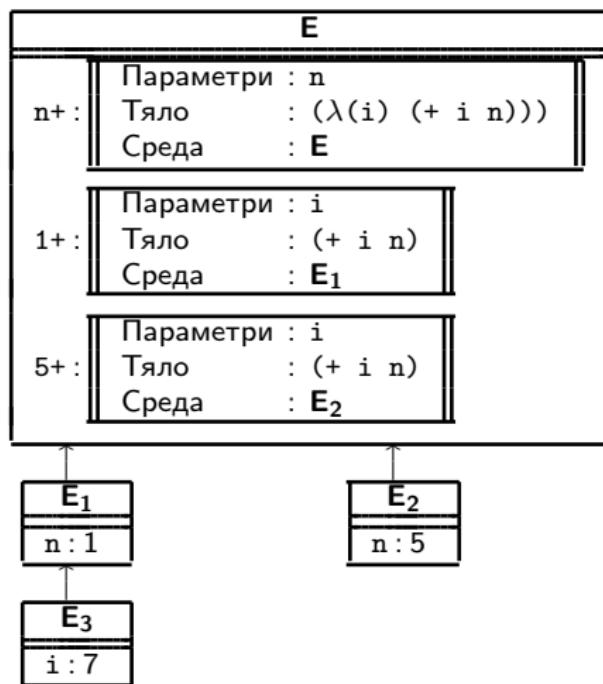
```



# Оценка на lambda

```

{E}   (define (n+ n)
{E}     (lambda (i) (+ i n)))
{E}   (define 1+ (n+ 1))
{E}   (define 5+ (n+ 5))
{E}           (1+ 7)
{E}           ↓
{E3}          (+ i n)
{E3}          ↓
{E3}          8
  
```



# Оценка на lambda

```

{E}   (define (n+ n)
{E}     (lambda (i) (+ i n)))
{E}   (define 1+ (n+ 1))
{E}   (define 5+ (n+ 5))
{E}           (1+ 7)
{E}           ↓
{E3}         (+ i n)
{E}           ↓
{E}           8
{E}           (5+ 7)
  
```

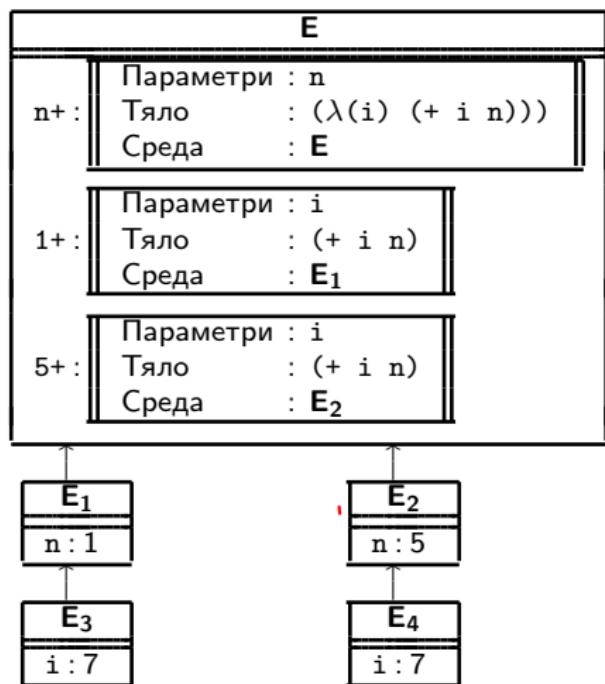


# Оценка на lambda

```

{E}   (define (n+ n)
{E}     (lambda (i) (+ i n)))
{E}   (define 1+ (n+ 1))
{E}   (define 5+ (n+ 5))
{E}           (1+ 7)
{E}           ↓
{E3}         (+ i n)
{E3}         ↓
{E3}         8
{E}           (5+ 7)
{E}           ↓
{E4}         (+ i n)

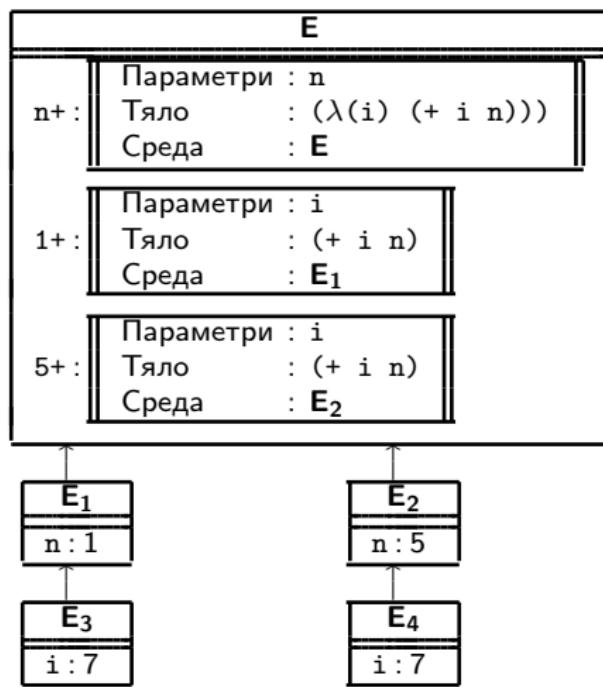
```



# Оценка на lambda

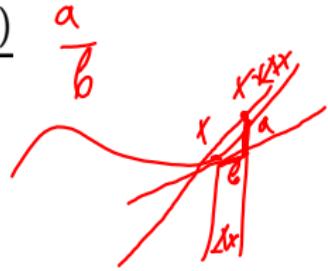
```

{E}   (define (n+ n)
{E}     (lambda (i) (+ i n)))
{E}   (define 1+ (n+ 1))
{E}   (define 5+ (n+ 5))
{E}           (1+ 7)
{E}           ↓
{E3}         (+ i n)
{E3}         ↓
{E3}         8
{E}           (5+ 7)
{E}           ↓
{E4}         (+ i n)
{E4}         ↓
{E4}         12
  
```



## Намиране на производна

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$



# Намиране на производна

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \text{ за малки } \Delta x$$

# Намиране на производна

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \text{ за малки } \Delta x$$

```
(define (derive f dx)
  (lambda (x) (/ (- (f (+ x dx)) (f x)) dx)))
```

# Намиране на производна

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \text{ за малки } \Delta x$$

```
(define (derive f dx)
  (lambda (x) (/ (- (f (+ x dx)) (f x)) dx)))
```

- (define 2\* (derive square 0.01))

# Намиране на производна

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \text{ за малки } \Delta x$$

```
(define (derive f dx)
  (lambda (x) (/ (- (f (+ x dx)) (f x)) dx)))
```

- (define 2\* (derive square 0.01))
- (2\* 5) → 10.009999999999764

# Намиране на производна

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \text{ за малки } \Delta x$$

```
(define (derive f dx)
  (lambda (x) (/ (- (f (+ x dx)) (f x)) dx)))
```

- (`define` 2\* (`derive square` 0.01))
- (2\* 5) → 10.009999999999764
- ((`derive square` 0.0000001) 5) → 10.000000116860974

$$(x^2)'(5)$$

# Намиране на производна

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \text{ за малки } \Delta x$$

```
(define (derive f dx)
  (lambda (x) (/ (- (f (+ x dx)) (f x)) dx)))
```

- (`define` 2\* (`derive square` 0.01))
- (2\* 5) → 10.009999999999764
- ((`derive square` 0.0000001) 5) → 10.000000116860974
- ((`derive` (`derive` (`lambda` (x) (\* x x x)) 0.001) 0.001) 3) → ?

$$(x^3)' = (3x^2)^1 = 6x$$

# Намиране на производна

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \text{ за малки } \Delta x$$

```
(define (derive f dx)
  (lambda (x) (/ (- (f (+ x dx)) (f x)) dx)))
```

- (define 2\* (derive square 0.01))
- (2\* 5) → 10.009999999999764
- ((derive square 0.0000001) 5) → 10.000000116860974
- ((derive (derive (lambda (x) (\* x x x)) 0.001) 0.001) 3) → 18.006000004788802

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

**Решение №1:**  $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

**Решение №1:**  $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

```
(define (repeated f n)
  (lambda (x) (if (= n 0) x (f ((repeated f (- n 1)) x)))))
```

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

**Решение №1:**  $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

```
(define (repeated f n)
  (lambda (x) (if (= n 0) x (f ((repeated f (- n 1)) x)))))
```

**Решение №2:**  $f^0 = id, f^n = f \circ f^{n-1}$

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

**Решение №1:**  $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

```
(define (repeated f n)
  (lambda (x) (if (= n 0) x (f ((repeated f (- n 1)) x)))))
```

**Решение №2:**  $f^0 = id, f^n = f \circ f^{n-1}$

```
(define (repeated f n)
  (if (= n 0) id (compose f (repeated f (- n 1)))))
```

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

**Решение №1:**  $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

```
(define (repeated f n)
  (lambda (x) (if (= n 0) x (f ((repeated f (- n 1)) x)))))
```

**Решение №2:**  $f^0 = id, f^n = f \circ f^{n-1}$

```
(define (repeated f n)
  (if (= n 0) id (compose f (repeated f (- n 1)))))
```

**Решение №3:**  $f^n = \underbrace{f \circ f \circ \dots \circ f}_{n} \circ id$

$f^1 \circ f^2 \circ f^3 \circ f^4 \circ \dots \circ f^n$

$x \cdot x \cdot \dots \cdot x \cdot 1$

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

**Решение №1:**  $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

```
(define (repeated f n)
  (lambda (x) (if (= n 0) x (f ((repeated f (- n 1)) x)))))
```

**Решение №2:**  $f^0 = id, f^n = f \circ f^{n-1}$

```
(define (repeated f n)
  (if (= n 0) id (compose f (repeated f (- n 1)))))
```

**Решение №3:**  $f^n = \underbrace{f \circ f \circ \dots \circ f}_{n} \circ id$

```
(define (repeated f n)
  (accumulate ? ? ? ? ? ?))
```

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

**Решение №1:**  $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

```
(define (repeated f n)
  (lambda (x) (if (= n 0) x (f ((repeated f (- n 1)) x)))))
```

**Решение №2:**  $f^0 = id, f^n = f \circ f^{n-1}$

```
(define (repeated f n)
  (if (= n 0) id (compose f (repeated f (- n 1)))))
```

**Решение №3:**  $f^n = \underbrace{f \circ f \circ \dots \circ f}_{n} \circ id$

```
(define (repeated f n)
  (accumulate compose ? ? ? ? ?))
```

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

**Решение №1:**  $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

```
(define (repeated f n)
  (lambda (x) (if (= n 0) x (f ((repeated f (- n 1)) x)))))
```

**Решение №2:**  $f^0 = id, f^n = f \circ f^{n-1}$

```
(define (repeated f n)
  (if (= n 0) id (compose f (repeated f (- n 1)))))
```

**Решение №3:**  $f^n = \underbrace{f \circ f \circ \dots \circ f}_{n} \circ id$

```
(define (repeated f n)
  (accumulate compose id ? ? ? ?))
```

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

**Решение №1:**  $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

```
(define (repeated f n)
  (lambda (x) (if (= n 0) x (f ((repeated f (- n 1)) x)))))
```

**Решение №2:**  $f^0 = id, f^n = f \circ f^{n-1}$

```
(define (repeated f n)
  (if (= n 0) id (compose f (repeated f (- n 1)))))
```

**Решение №3:**  $f^n = \underbrace{f \circ f \circ \dots \circ f}_{n} \circ id$

```
(define (repeated f n)
  (accumulate compose id 1 ? ? ?))
```

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

**Решение №1:**  $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

```
(define (repeated f n)
  (lambda (x) (if (= n 0) x (f ((repeated f (- n 1)) x)))))
```

**Решение №2:**  $f^0 = id, f^n = f \circ f^{n-1}$

```
(define (repeated f n)
  (if (= n 0) id (compose f (repeated f (- n 1)))))
```

**Решение №3:**  $f^n = \underbrace{f \circ f \circ \dots \circ f}_{n} \circ id$

```
(define (repeated f n)
  (accumulate compose id 1 n ? ?))
```

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

**Решение №1:**  $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

```
(define (repeated f n)
  (lambda (x) (if (= n 0) x (f ((repeated f (- n 1)) x)))))
```

**Решение №2:**  $f^0 = id, f^n = f \circ f^{n-1}$

```
(define (repeated f n)
  (if (= n 0) id (compose f (repeated f (- n 1)))))
```

**Решение №3:**  $f^n = \underbrace{f \circ f \circ \dots \circ f}_{n} \circ id$

```
(define (repeated f n)
  (accumulate compose id 1 n (lambda (i) f) ?))
```

## Повторено прилагане

Да се намери  $n$ -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

**Решение №1:**  $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

```
(define (repeated f n)
  (lambda (x) (if (= n 0) x (f ((repeated f (- n 1)) x)))))
```

**Решение №2:**  $f^0 = id, f^n = f \circ f^{n-1}$

```
(define (repeated f n)
  (if (= n 0) id (compose f (repeated f (- n 1)))))
```

**Решение №3:**  $f^n = \underbrace{f \circ f \circ \dots \circ f}_{n} \circ id$

```
(define (repeated f n)
  (accumulate compose id 1 n (lambda (i) f) 1+))
```

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f$ ,  $f^{(n)} = (f^{(n-1)})'$

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

**Решение №2:**  $f^{(n)} = f \overbrace{''''...''}^n | \rightarrow (n)$

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

**Решение №2:**  $f^{(n)} = f \overbrace{''''...''}^n$

```
(define (derive-n f n dx)
  (repeated ? n))
```

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

**Решение №2:**  $f^{(n)} = f \overbrace{''''...''}^n$

```
(define (derive-n f n dx)
  (repeated (lambda (f) (derive f dx)) n))
```

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

**Решение №2:**  $f^{(n)} = f \overbrace{''''...''}^n$

```
(define (derive-n f n dx)
  ((repeated (lambda (f) (derive f dx)) n) f))
```

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

**Решение №2:**  $f^{(n)} = f \overbrace{''''...''}^n$

```
(define (derive-n f n dx)
  ((repeated (lambda (f) (derive f dx)) n) f))
```

**Решение №3:**  $f^{(n)} = \underbrace{f \circ f \circ \dots \circ f}_n$

```
(define (derive-n f n dx)
  ((accumulate ? ? ? ? ?) f))
```

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

**Решение №2:**  $f^{(n)} = f \overbrace{''''...''}^n$

```
(define (derive-n f n dx)
  ((repeated (lambda (f) (derive f dx)) n) f))
```

**Решение №3:**  $f^{(n)} = \underbrace{f \circ f \circ \dots \circ f}_n$

```
(define (derive-n f n dx)
  ((accumulate compose ? ? ? ? ?) f))
```

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

**Решение №2:**  $f^{(n)} = f \overbrace{''''...''}^n$

```
(define (derive-n f n dx)
  ((repeated (lambda (f) (derive f dx)) n) f))
```

**Решение №3:**  $f^{(n)} = \underbrace{f \circ f \circ \dots \circ f}_n$

```
(define (derive-n f n dx)
  ((accumulate compose id ? ? ? ?) f))
```

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

**Решение №2:**  $f^{(n)} = f \overbrace{''''...''}^n$

```
(define (derive-n f n dx)
  ((repeated (lambda (f) (derive f dx)) n) f))
```

**Решение №3:**  $f^{(n)} = \underbrace{f \circ f \circ \dots \circ f}_n$

```
(define (derive-n f n dx)
  ((accumulate compose id 1 ? ? ?) f))
```

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

**Решение №2:**  $f^{(n)} = f \overbrace{''''...''}^n$

```
(define (derive-n f n dx)
  ((repeated (lambda (f) (derive f dx)) n) f))
```

**Решение №3:**  $f^{(n)} = \underbrace{f \circ f \circ \dots \circ f}_n$

```
(define (derive-n f n dx)
  ((accumulate compose id 1 n ? ?) f))
```

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

**Решение №2:**  $f^{(n)} = f \overbrace{''''...''}^n$

```
(define (derive-n f n dx)
  ((repeated (lambda (f) (derive f dx)) n) f))
```

**Решение №3:**  $f^{(n)} = \underbrace{f \circ f \circ \dots \circ f}_n$

```
(define (derive-n f n dx)
  ((accumulate compose id 1 n
    (lambda (i) (lambda (f) (derive f dx)))) ?) f))
```

## *n*-та производна

Да се намери *n*-та производна на дадена едноместна функция.

**Решение №1:**  $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

**Решение №2:**  $f^{(n)} = f \overbrace{''''...''}^n$

```
(define (derive-n f n dx)
  ((repeated (lambda (f) (derive f dx)) n) f))
```

**Решение №3:**  $f^{(n)} = \underbrace{f \circ f \circ \dots \circ f}_n$

```
(define (derive-n f n dx)
  ((accumulate compose id 1 n
    (lambda (i) (lambda (f) (derive f dx))) 1+) f))
```

# All you need is $\lambda$

Специалната форма lambda е достатъчна за реализацията на всички останали конструкции в Scheme!

# All you need is $\lambda$ – let

Специалната форма lambda е достатъчна за реализацията на всички останали конструкции в Scheme!

(**let** ((**<символ>** **<израз>**)) **<тяло>**)

Симулация на let:

$\iff$

((**lambda** (**<символ>**) **<тяло>**) **<израз>**)

# All you need is $\lambda$ – let

Специалната форма lambda е достатъчна за реализацията на всички останали конструкции в Scheme!

$$(\text{let } ((<\text{символ}> <\text{израз}>)) <\text{тяло}>)$$

Симулация на let:  $\iff$

$$((\text{lambda } (<\text{символ}>) <\text{тяло}>) <\text{израз}>)$$

$$(\text{let } ((<\text{символ}_1> <\text{израз}_1>)$$

$$(<\text{символ}_2> <\text{израз}_2>)$$

$$\dots$$

$$(<\text{символ}_n> <\text{израз}_n>)$$

$$<\text{тяло}>)$$

$$\iff$$

$$((\text{lambda } (<\text{символ}_1> \dots <\text{символ}_n>) <\text{тяло}>)$$

$$<\text{израз}_1> \dots <\text{израз}_n>)$$

# All you need is $\lambda$ — булева логика

Симулация на булеви стойности и **if**:

```
(define #t (lambda (x y) x))  
(define #f (lambda (x y) y))  
(define (lambda-if b x y) (b x y))
```

# All you need is $\lambda$ — булева логика

Симулация на булеви стойности и **if**:

```
(define #t (lambda (x y) x))  
(define #f (lambda (x y) y))  
(define (lambda-if b x y) (b x y))
```

# All you need is $\lambda$ — булева логика

Симулация на булеви стойности и **if**:

```
(define #t (lambda (x y) x))  
(define #f (lambda (x y) y))  
(define (lambda-if b x y) ((b x y)))
```

# All you need is $\lambda$ — булева логика

Симулация на булеви стойности и **if**:

```
(define #t (lambda (x y) x))  
(define #f (lambda (x y) y))  
(define (lambda-if b x y) ((b x y)))
```

Примери:

- (lambda-if #t (lambda () (+ 3 5)) (lambda () (/ 4 0)))  $\rightarrow$  8
- (lambda-if #f (lambda () +) (lambda () "abc"))  $\rightarrow$  "abc"
- (define (not b) (lambda (x y) (b y x)))

# All you need is $\lambda$ — числа

Симулация на естествени числа (нумерали на Чърч)

Идея: представяне на числото  $n$  като  $\lambda f, x f^n(x)$

# All you need is $\lambda$ — числа

Симулация на естествени числа (нумерали на Чърч)

Идея: представяне на числото  $n$  като  $\lambda f, x f^n(x)$

- (define c3 (lambda (f x) (f (f (f x)))))

# All you need is $\lambda$ — числа

Симулация на естествени числа (нумерали на Чърч)

Идея: представяне на числото  $n$  като  $\lambda f, x f^n(x)$

- `(define c3 (lambda (f x) (f (f (f x)))))`
- `(define c5 (lambda (f x) (f (f (f (f (f x)))))))`

# All you need is $\lambda$ — числа

Симулация на естествени числа (*нумерали на Чърч*)

Идея: представяне на числото  $n$  като  $\lambda f, x f^n(x)$

- `(define c3 (lambda (f x) (f (f (f x)))))`
- `(define c5 (lambda (f x) (f (f (f (f (f x)))))))`
- `(define c1+ (lambda (a) (lambda (f x) (f (a f x)))))`

# All you need is $\lambda$ — числа

Симулация на естествени числа (*нумерали на Чърч*)

Идея: представяне на числото  $n$  като  $\lambda f, x f^n(x)$

- `(define c3 (lambda (f x) (f (f (f x)))))`
- `(define c5 (lambda (f x) (f (f (f (f (f x)))))))`
- `(define c1+ (lambda (a) (lambda (f x) (f (a f x)))))`
- `(define c+ (lambda (a b) (lambda (f x) (a f (b f x)))))`

# All you need is $\lambda$ — рекурсия

Рекурсивна дефиниция:

```
(define f (gamma f))
```

# All you need is $\lambda$ — рекурсия

Рекурсивна дефиниция:

```
(define f (gamma f))
```

Пример:

```
(define (fact n)
  (if (= n 0) 1 (* n (fact (- n 1)))))
```

# All you need is $\lambda$ — рекурсия

Рекурсивна дефиниция:

```
(define f (gamma f))
```

Пример:

```
(define fact
  (lambda (n)
    (if (= n 0) 1 (* n (fact (- n 1))))))
```

# All you need is $\lambda$ — рекурсия

Рекурсивна дефиниция:

```
(define f (gamma f))
```

Пример:

```
(define (fact n) ((gamma fact) n))
(define (gamma f)
  (lambda (n)
    (if (= n 0) 1 (* n (f (- n 1))))))
```

# All you need is $\lambda$ — рекурсия

Рекурсивна дефиниция:

```
(define f (gamma f))
```

Пример:

```
(define (fact n) ((gamma fact) n))
(define gamma
  (lambda (f)
    (lambda (n)
      (if (= n 0) 1 (* n (f (- n 1)))))))
```

# All you need is $\lambda$ — рекурсия

Рекурсивна дефиниция:

```
(define f (gamma f))
```

Пример:

```
(define (fact n) ((gamma fact) n))
(define gamma
  (lambda (f)
    (lambda (n)
      (if (= n 0) 1 (* n (f (- n 1)))))))
```

fact е най-малка неподвижна точка на оператора gamma.

# All you need is $\lambda$ — рекурсия

Рекурсивна дефиниция:

```
(define f (gamma f))
```

Пример:

```
(define (fact n) ((gamma fact) n))
(define gamma
  (lambda (f)
    (lambda (n)
      (if (= n 0) 1 (* n (f (- n 1)))))))
```

fact е най-малка неподвижна точка на оператора gamma.

Търсим fact такова, че  $(\text{fact } n) = ((\text{gamma } \text{fact}) \ n) =$   
 $((\text{gamma } (\text{gamma } \text{fact})) \ n) = ((\text{gamma } (\text{gamma } (\text{gamma } \text{fact}))) \ n) = \dots$

# All you need is $\lambda$ — намиране на неподвижна точка

Идея №1: (`define fact (((repeated gamma ?) 'empty))`)

# All you need is $\lambda$ — намиране на неподвижна точка

**Идея №1:** (`define fact (((repeated gamma ?) 'empty))`)

**Проблем №1:** Не знаем колко пъти да повторим `gamma`...

# All you need is $\lambda$ — намиране на неподвижна точка

**Идея №1:** (`define fact (((repeated gamma ?) 'empty))`)

**Проблем №1:** Не знаем колко пъти да повторим `gamma`...

**Идея №2:** Да повтаряме `gamma` безкрайно!

```
(define (gamma-inf) (lambda (n) ((gamma (gamma-inf)) n)))  
(define fact (gamma-inf))
```

# All you need is $\lambda$ — намиране на неподвижна точка

**Идея №1:** (`define fact (((repeated gamma ?) 'empty))`)

**Проблем №1:** Не знаем колко пъти да повторим `gamma`...

**Идея №2:** Да повтаряме `gamma` безкрайно!

```
(define (gamma-inf) (lambda (n) ((gamma (gamma-inf)) n)))
(define fact (gamma-inf))
```

**Проблем №2:** `gamma-inf` се дефинира рекурсивно...

# All you need is $\lambda$ — намиране на неподвижна точка

**Идея №1:** (`define fact (((repeated gamma ?) 'empty))`)

**Проблем №1:** Не знаем колко пъти да повторим `gamma`...

**Идея №2:** Да повтаряме `gamma` безкрайно!

```
(define (gamma-inf) (lambda (n) ((gamma (gamma-inf)) n)))
(define fact (gamma-inf))
```

**Проблем №2:** `gamma-inf` се дефинира рекурсивно...

**Идея №3:** Да подменим рекурсивното извикване с параметър:

```
(define (gamma-inf me) (lambda (n) ((gamma (me me)) n)))
```

# All you need is $\lambda$ — намиране на неподвижна точка

**Идея №1:** (`define fact (((repeated gamma ?) 'empty))`)

**Проблем №1:** Не знаем колко пъти да повторим `gamma`...

**Идея №2:** Да повтаряме `gamma` безкрайно!

```
(define (gamma-inf) (lambda (n) ((gamma (gamma-inf)) n)))
(define fact (gamma-inf))
```

**Проблем №2:** `gamma-inf` се дефинира рекурсивно...

**Идея №3:** Да подменим рекурсивното извикване с параметър:

```
(define (gamma-inf me) (lambda (n) ((gamma (me me)) n)))
```

**Идея №4:** Да подадем `gamma-inf` като параметър на себе си!

# All you need is $\lambda$ — намиране на неподвижна точка

**Идея №1:** (`define fact (((repeated gamma ?) 'empty))`)

**Проблем №1:** Не знаем колко пъти да повторим `gamma`...

**Идея №2:** Да повтаряме `gamma` безкрайно!

```
(define (gamma-inf) (lambda (n) ((gamma (gamma-inf)) n)))
(define fact (gamma-inf))
```

**Проблем №2:** `gamma-inf` се дефинира рекурсивно...

**Идея №3:** Да подменим рекурсивното извикване с параметър:

```
(define (gamma-inf me) (lambda (n) ((gamma (me me)) n)))
```

**Идея №4:** Да подадем `gamma-inf` като параметър на себе си!

```
(define fact (gamma-inf gamma-inf))
```

# All you need is $\lambda$ — комбинатор Y

Да напишем функция, която намира неподвижната точка на произволен оператор *gamma*:

```
(define (Y gamma)
  (define (gamma-inf me) (lambda (n) ((gamma (me me)) n)))
  (gamma-inf gamma-inf))
```

# All you need is $\lambda$ — комбинатор Y

Да напишем функция, която намира неподвижната точка на произволен оператор `gamma`:

```
(define (Y gamma)
  (define (gamma-inf me) (lambda (n) ((gamma (me me)) n)))
  (gamma-inf gamma-inf))
```

А сега само с  $\lambda$ :

```
(define Y
  (lambda (gamma)
    ((lambda (gamma-inf) (gamma-inf gamma-inf))
     (lambda (me) (lambda (n) ((gamma (me me)) n))))))
```

# All you need is $\lambda$ — комбинатор Y

Да напишем функция, която намира неподвижната точка на произволен оператор `gamma`:

```
(define (Y gamma)
  (define (gamma-inf me) (lambda (n) ((gamma (me me)) n)))
  (gamma-inf gamma-inf))
```

А сега само с  $\lambda$ :

```
(define Y
  (lambda (gamma)
    ((lambda (me) (lambda (n) ((gamma (me me)) n)))
     (lambda (me) (lambda (n) ((gamma (me me)) n))))))
```

# All you need is $\lambda$ — комбинатор Y

Да напишем функция, която намира неподвижната точка на произволен оператор *gamma*:

```
(define (Y gamma)
  (define (gamma-inf me) (lambda (n) ((gamma (me me)) n)))
  (gamma-inf gamma-inf))
```

А сега само с  $\lambda$ :

```
(define Y
  (lambda (gamma)
    ((lambda (me) (lambda (n) ((gamma (me me)) n)))
     (lambda (me) (lambda (n) ((gamma (me me)) n))))))
```

Y се нарича комбинатор за намиране на най-малка неподвижна точка (fixpoint combinator).