

# Функции от по-висок ред

Трифон Трифонов

Функционално програмиране, спец. Информатика, 2015/16 г.

28 октомври 2015 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? + 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`
- `(fixed-point? exp 1) → #f`
- `(fixed-point? + 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? + 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? + 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? + 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? + 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? + 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? + 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? + 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? + 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? + 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

### 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?

## 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  $\int_a^b f(x) \approx \Delta x [f(a) + f(a + \Delta x) + f(a + 2\Delta x) + \dots + f(b)]$
- 3  $x + e^x + e^{e^x} + e^{e^{e^x}} + \dots$  докато поредното събираемо е  $\leq 10^{1000}$

## 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 събираемо е  $\leq 10^{1000}$

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



## 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 събираемо е  $\leq 10^{1000}$

```
(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))))
```

## 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 събираемо е  $\leq 10^{1000}$

```
(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 събираемо е  $\leq 10^{1000}$

```
(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 (exp 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 събираемо е  $\leq 10^{1000}$

```
(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 (exp 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 събираемо е  $\leq 10^{1000}$

```
(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 (exp 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+))
```

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

## 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+))
```

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

```
(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+))
```

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

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

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

## 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 (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:

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

## 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 (product a b term next)
  (if (> a b) 1 (* (term a) (product (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 (product a b term next)
  (if (> a b) 1 (* (term a) (product (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 (product a b term next)
  (if (> a b) 1 (* (term a) (product (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

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

където  $\oplus$  е бинарна операция,

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

## Обобщена функция за натрупване

Да се напише функция, която пресмята

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

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

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

където  $\oplus$  е бинарна операция,

$a \perp$  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))))

(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+))

```

## 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

$$P_n(x) = \left( \left( \left( \dots ((x + 2)x + 3)x + \dots \right) x + (n - 1) \right) x + n \right) x + (n + 1)$$



# Правило на Хорнер

$$P_n(x) = \left( \left( \left( \dots ((x + 2)x + 3)x + \dots \right) x + (n - 1) \right) x + n \right) x + (n + 1)$$

Можем ли да сметнем с accumulate?

# Правило на Хорнер

$$P_n(x) = \left( \left( \left( \dots \left( (x+2)x + 3 \right) x + \dots \right) x + (n-1) \right) x + n \right) x + (n+1)$$

Можем ли да сметнем с `accumulate`?

Идея: Да използваме операцията  $a \oplus b := ax + b$ .

# Правило на Хорнер

$$P_n(x) = \left( \left( \left( \dots \left( (x + 2)x + 3 \right) x + \dots \right) x + (n - 1) \right) x + n \right) x + (n + 1)$$

Можем ли да сметнем с accumulate?

**Идея:** Да използваме операцията  $a \oplus b := ax + b$ .

Коя е “нулевата стойност”  $\perp$ ?

## Правило на Хорнер

$$P_n(x) = \left( \left( \left( \dots \left( (x + 2)x + 3 \right) x + \dots \right) x + (n - 1) \right) x + n \right) x + (n + 1)$$

Можем ли да сметнем с accumulate?

Идея: Да използваме операцията  $a \oplus b := ax + b$ .

Коя е “нулевата стойност”  $\perp$ ?

Решение №2:

```
(define (p n x)
  (define (op a b) (+ (* a x) b))
  (accumulate op 0 1 (1+ n) id 1+))
```

# Правило на Хорнер

$$P_n(x) = \left( \left( \left( \dots \left( (x + 2)x + 3 \right) x + \dots \right) x + (n - 1) \right) x + n \right) x + (n + 1)$$

Можем ли да сметнем с accumulate?

Идея: Да използваме операцията  $a \oplus b := ax + b$ .

Коя е “нулевата стойност”  $\perp$ ?

Решение №2:

```
(define (p n x)
  (define (op a b) (+ (* a x) b))
  (accumulate op 0 1 (1+ n) id 1+))
```

Не смята правилно!

# Правило на Хорнер

Всъщност пресметнахме:

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

# 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  $a \oplus b := a + bx$ .

# 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  $a \oplus b := a + bx$ .

**Решение №3:**

```
(define (p n x)
  (define (op a b) (+ a (* b 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  $a \oplus b := a + bx$ .

**Решение №3:**

```
(define (p n x)
  (define (op a b) (+ a (* b x)))
  (accumulate op 0 1 (1+ n) id 1+))
```

**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

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

## Ляво и дясно натрупване

Всъщност пресметнахме:

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

ВМЕСТО

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

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

## Обобщена функция за натрупване

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

$$(\dots ((\perp \oplus \mathit{term}(a)) \oplus \mathit{term}(\mathit{next}(a))) \oplus \dots) \oplus \mathit{term}(b))$$

## Обобщена функция за натрупване

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

$$(\dots((\perp \oplus \mathit{term}(a)) \oplus \mathit{term}(\mathit{next}(a))) \oplus \dots) \oplus \mathit{term}(b))$$

```
(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 натрупване

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

$$(\dots((\perp \oplus \mathit{term}(a)) \oplus \mathit{term}(\mathit{next}(a))) \oplus \dots) \oplus \mathit{term}(b))$$

```
(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-i` и  $a \oplus b := ax + b$ .

## 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-i` и  $a \oplus b := ax + b$ .

### Решение №4:

```

(define (p n x)
  (define (op a b) (+ (* a x) b))
  (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 им даваме имена?

## Анонимни функции

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

- `(lambda ({<параметър>}) <тяло>)`

## Анонимни функции

Можем ли да ги конструираме параметрите на функциите от по-висок ред “на място”, без да им 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 им 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 им 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 им 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))`  $\rightarrow$  `#<procedure>`

## Анонимни функции

Можем ли да ги конструираме параметрите на функциите от по-висок ред “на място”, без да им 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))`  $\rightarrow$  `#<procedure>`
  - `((lambda (x) (+ x 3)) 5)`  $\rightarrow$  `8`

## Анонимни функции

Можем ли да ги конструираме параметрите на функциите от по-висок ред “на място”, без да им 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))`  $\rightarrow$  `#<procedure>`
  - `((lambda (x) (+ x 3)) 5)`  $\rightarrow$  `8`
  - `(define (<име> <параметри>) <тяло>)`  
 $\longleftrightarrow$   
`(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 (1+ n) (lambda (a b) (+ (* a x) b))
                (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 (1+ n) (lambda (a b) (+ (* a x) b))
    (lambda (i) (+ i 1))))
```

Как можем да реализираме с accumulate:

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

## Функции, които връщат функции

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

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

## Функции, които връщат функции

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

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

## Функции, които връщат функции

Да напишем 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 напишем 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 напишем 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 напишем 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 напишем 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 напишем 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 напишем 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 напишем 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 напишем 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))))`

# 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`

# 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`



# 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) → ?`

# 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) → 26`

## Оценка на lambda

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

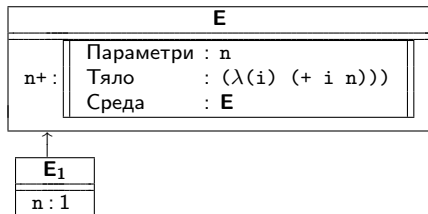
E							
n+:	<table border="1"> <tr> <td>Параметри</td> <td>: n</td> </tr> <tr> <td>Тяло</td> <td>: (λ(i) (+ i n))</td> </tr> <tr> <td>Среда</td> <td>: E</td> </tr> </table>	Параметри	: n	Тяло	: (λ(i) (+ i n))	Среда	: E
Параметри	: n						
Тяло	: (λ(i) (+ i n))						
Среда	: E						

## 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))
```



## Оценка на lambda

```

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

```

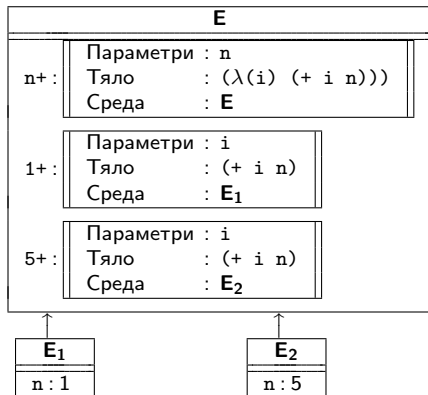


## Оценка на lambda

```

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

```



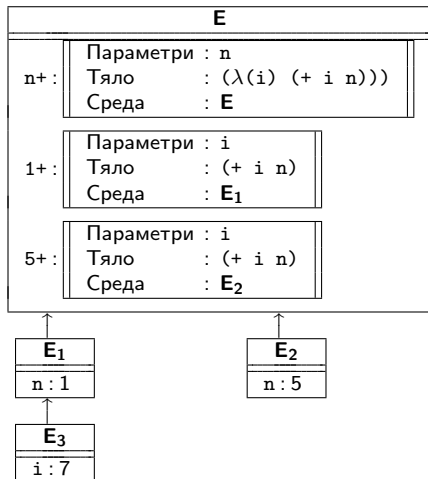


## Оценка на lambda

```

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

```

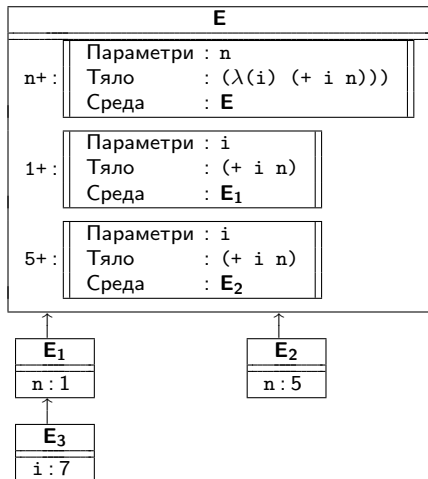


## Оценка на lambda

```

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

```

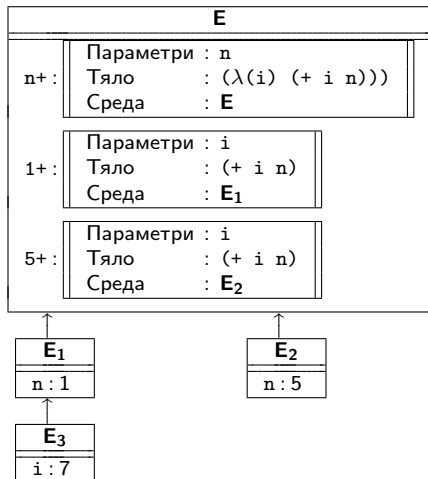


## Оценка на lambda

```

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

```

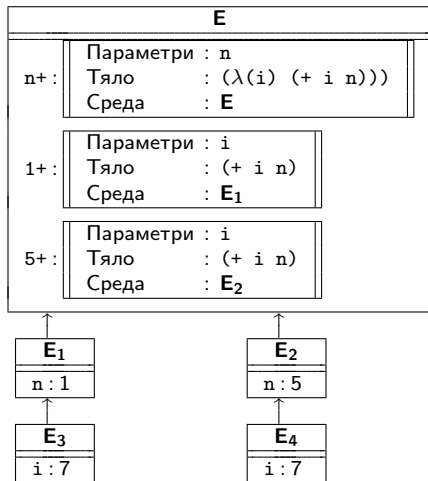


## Оценка на lambda

```

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

```

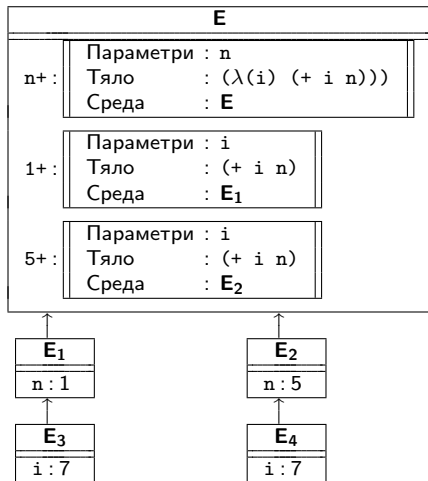


## Оценка на lambda

```

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

```



## Намиране на производна

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

# Намиране на производна

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

## Намиране на производна

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

```
(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) → 10.0099999999999764

# 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) → 10.0099999999999764`
- `((derive square 0.0000001) 5) → 10.000000116860974`

## Намиране на производна

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

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

- `(define 2* (derive square 0.01))`
- `(2* 5) → 10.0099999999999764`
- `((derive square 0.0000001) 5) → 10.000000116860974`
- `((derive (derive (lambda (x) (* x x x)) 0.001) 0.001) 3)`  
→ ?

# 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) → 10.0099999999999764`
- `((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 функция.

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

## Повторено прилагане

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

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

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

## Повторено прилагане

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

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

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

```
(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 функция.

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

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

```
(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 функция.

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

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

```
(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 функция.

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

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

```
(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:**  $f^n = \underbrace{f \circ f \circ \dots \circ f}_{n} \circ id$

## Повторено прилагане

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

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

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

```
(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 функция.

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

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

```
(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 функция.

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

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

```
(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 функция.

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

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

```
(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 функция.

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

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

```
(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 функция.

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

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

```
(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 функция.

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

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

```
(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 функция.

## $n$ -та производна

Да се намери  $n$ -та производна на дадена функция.

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

## $n$ -та производна

Да се намери  $n$ -та производна на дадена функция.

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

```
(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 функция.

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

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

**Решение №2:**  $f^{(n)} = f \overset{n}{\underbrace{'' \dots '}}$

## $n$ -та производна

Да се намери  $n$ -та производна на дадена функция.

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

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

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

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

## $n$ -та производна

Да се намери  $n$ -та производна на дадена функция.

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

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

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

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



## $n$ -та производна

Да се намери  $n$ -та производна на дадена функция.

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

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

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

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