Функции от по-висок ред – част 2

Трифон Трифонов

Функционално програмиране, 2024/25 г.

23 октомври 2024 г.

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)` $\longrightarrow$?

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)` $\longrightarrow$ 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)` $\longrightarrow$ 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)` $\longrightarrow$ 81
- `(define (twice f) (lambda (x) (f (f x))))`
- `(twice square 3)` $\longrightarrow$?

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)` $\longrightarrow$ 81
- `(define (twice f) (lambda (x) (f (f x))))`
- `(twice square 3)` $\longrightarrow$ 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)` $\longrightarrow$ 81
- `(define (twice f) (lambda (x) (f (f x))))`
- `(twice square 3)` $\longrightarrow$ Грешка!
- `(twice square)` $\longrightarrow$?

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)` ⟶ ?

2

4

16

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

Оценка на lambda

{E}

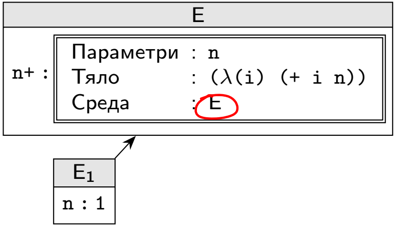
```
(define (n+ n)
  (lambda (i) (+ i n)))
```

E			
n+ :	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))
```
↓

{E_1}
```
(lambda (i) (+ i n))
```

Оценка на lambda

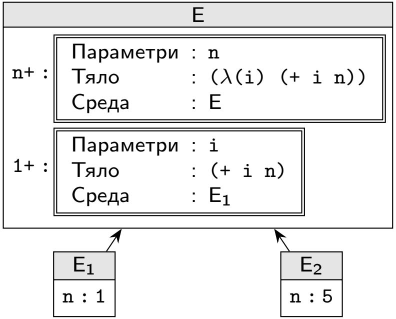
{E} `(define (n+ n)`
`(lambda (i) (+ i n)))`

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

↓

{E_1} `(lambda (i) (+ i n))`

{E} `(define 5+ (n+ 5))`

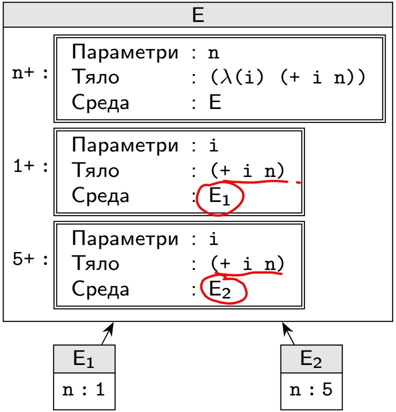


Оценка на lambda

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

  {E}       (define 1+ (n+ 1))
                        ↓
{E1}      (lambda (i) (+ i n))

  {E}       (define 5+ (n+ 5))
                        ↓
{E2}      (lambda (i) (+ i n))
```



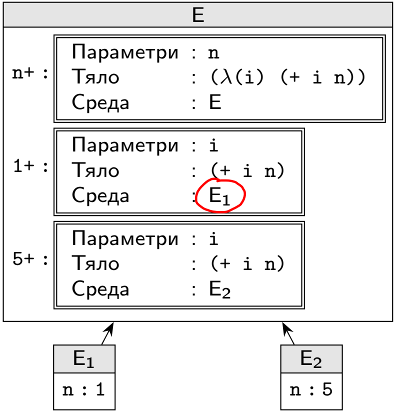
Оценка на lambda

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

  {E}        (define 1+ (n+ 1))
                          ↓
{E₁}       (lambda (i) (+ i n))

  {E}        (define 5+ (n+ 5))
                          ↓
{E₂}       (lambda (i) (+ i n))

 {E}       (1+ 7)
```



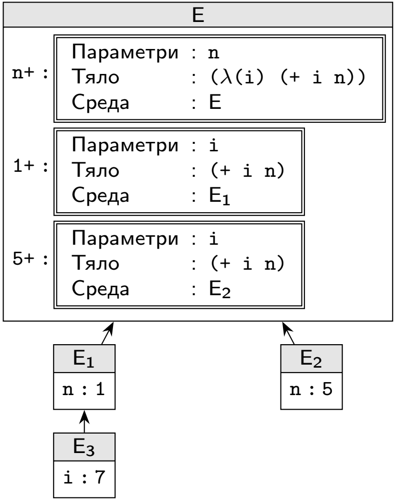
Оценка на lambda

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

{E}      (define 1+ (n+ 1))
                     ↓
{E1}     (lambda (i) (+ i n))

{E}      (define 5+ (n+ 5))
                     ↓
{E2}     (lambda (i) (+ i n))

{E}      (1+ 7)
           ↓
{E3}     (+ i n)
```



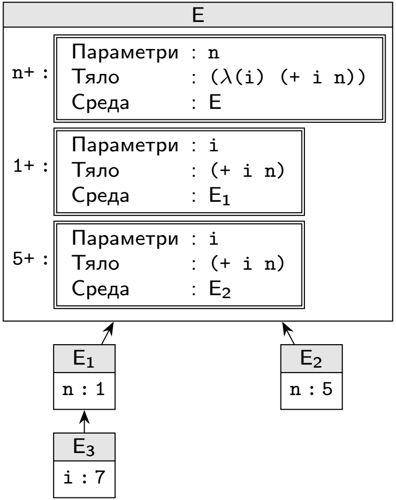
Оценка на lambda

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

 {E}      (define 1+ (n+ 1))
                        ↓
{E₁}     (lambda (i) (+ i n))

 {E}      (define 5+ (n+ 5))
                        ↓
{E₂}     (lambda (i) (+ i n))

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



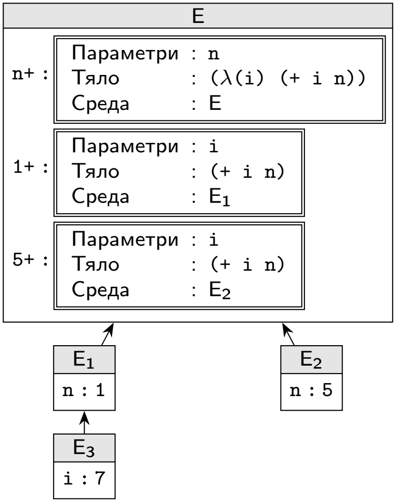
Оценка на lambda

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

{E}     (define 1+ (n+ 1))
                     ↓
{E1}    (lambda (i) (+ i n))

{E}     (define 5+ (n+ 5))
                     ↓
{E2}    (lambda (i) (+ i n))

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



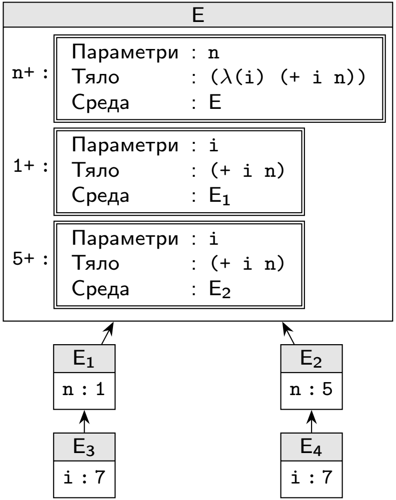
Оценка на lambda

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

{E}     (define 1+ (n+ 1))
                    ↓
{E1}    (lambda (i) (+ i n))

{E}     (define 5+ (n+ 5))
                    ↓
{E2}    (lambda (i) (+ i n))

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



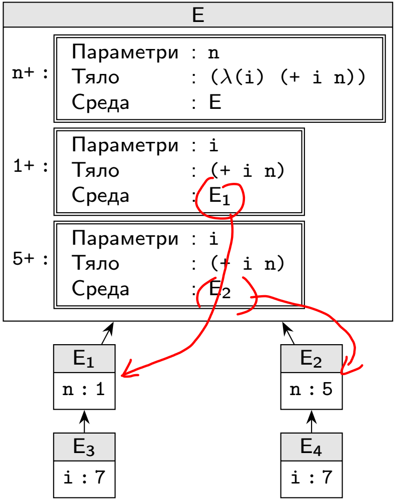
Оценка на lambda

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

{E}      (define 1+ (n+ 1))
                 ↓
{E1}    (lambda (i) (+ i n))

{E}      (define 5+ (n+ 5))
                 ↓
{E2}    (lambda (i) (+ i n))

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



Намиране на производна

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

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) $\longrightarrow$ 10.009999999999764

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)` $\longrightarrow$ `10.009999999999764`
- `((derive square 0.0000001) 5)` $\longrightarrow$ `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)` ⟶ `10.009999999999764`
- `((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)` $\longrightarrow$ `10.009999999999764`
- `((derive square 0.0000001) 5)` $\longrightarrow$ `10.000000116860974`
- `((derive (derive (lambda (x) (* x x x)) 0.001) 0.001) 3)` $\longrightarrow$ `18.006000004788802`

Повторено прилагане

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

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

Повторено прилагане

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

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

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.

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

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.

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

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.

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

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.

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

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: $f^n = \underbrace{f \circ f \circ \ldots \circ f}_{n} \circ id$

Повторено прилагане

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

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

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: $f^n = \underbrace{f \circ f \circ \ldots \circ f}_{n} \circ id$

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

Повторено прилагане

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

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

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: $f^n = \underbrace{f \circ f \circ \ldots \circ f}_{n} \circ id$

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

Повторено прилагане

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

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

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: $f^n = \underbrace{f \circ f \circ \ldots \circ f}_{n} \circ id$

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

Повторено прилагане

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

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

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: $f^n = \underbrace{f \circ f \circ \ldots \circ f}_{n} \circ id$

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

Повторено прилагане

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

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

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: $f^n = \underbrace{f \circ f \circ \ldots \circ f}_{n} \circ id$

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

Повторено прилагане

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

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

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: $f^n = \underbrace{f \circ f \circ \ldots \circ f}_{n} \circ id$

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

Повторено прилагане

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

$$f^n(x) = \underbrace{f(f(f(\ldots(f}_{n}(x))\ldots)))$$

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: $f^n = \underbrace{f \circ f \circ \ldots \circ f}_{n} \circ id$

```
(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: $f^{(n)} = f^{\overbrace{''''\ldots''}^{n}}$

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: $f^{(n)} = f^{\overbrace{''''\ldots''}^{n}}$

```
(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: $f^{(n)} = f^{\overbrace{''''\ldots''}^{n}}$

```
(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: $f^{(n)} = f^{\overbrace{''''\ldots''}^{n}}$

```
(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: $f^{(n)} = f^{\overbrace{''''\ldots''}^{n}}$

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

Решение №3: $^{(n)} = \underbrace{'\circ'\circ\ldots\circ'}_{n}$

```
(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: $f^{(n)} = f^{\overbrace{''''\ldots''}^{n}}$

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

Решение №3: $^{(n)} = \underbrace{'\circ'\circ\ldots\circ'}_{n}$

```
(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: $f^{(n)} = f^{\overbrace{''''\ldots''}^{n}}$

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

Решение №3: $^{(n)} = \underbrace{'\circ'\circ\ldots\circ'}_{n}$

```
(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: $f^{(n)} = f^{\overbrace{''''\ldots''}^{n}}$

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

Решение №3: $^{(n)} = \underbrace{{}'\circ{}'\circ\ldots\circ{}'}_{n}$

```
(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: $f^{(n)} = f^{\overbrace{''''\ldots''}^{n}}$

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

Решение №3: $^{(n)} = \underbrace{'\circ'\circ\ldots\circ'}_{n}$

```
(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: $f^{(n)} = f^{\overbrace{''''\ldots''}^{n}}$

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

Решение №3: $^{(n)} = \underbrace{'\circ'\circ\ldots\circ'}_{n}$

```
(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: $f^{(n)} = f^{\overbrace{''''\ldots''}^{n}}$

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

Решение №3: $^{(n)} = \underbrace{'\circ'\circ\ldots\circ'}_{n}$

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

All you need is λ

Специалната форма `lambda` е достатъчна за реализацията на (почти) всички конструкции в Scheme!

All you need is λ — `let`

Специалната форма `lambda` е достатъчна за реализацията на (почти) всички конструкции в Scheme!

Симулация на `let`:

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

$$\Longleftrightarrow$$

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

All you need is λ — `let`

Специалната форма `lambda` е достатъчна за реализацията на (почти) всички конструкции в Scheme!

Симулация на `let`:

(**let** ((<символ> <израз>)) <тяло>)

$\Longleftrightarrow$

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

(**let** ((<символ$_1$> <израз$_1$>)
(<символ$_2$> <израз$_2$>)
...
(<символ$_n$> <израз$_n$>))
<тяло>)

$\Longleftrightarrow$

((**lambda** (<символ$_1$> ... <символ$_n$>) <тяло>)
<израз$_1$> ... <израз$_n$>)

All you need is λ — 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 λ — 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 λ — 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 λ — 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)))` $\longrightarrow$ `8`
- `(lambda-if #f (lambda () +) (lambda () "abc"))` $\longrightarrow$ `"abc"`
- `(define (not b) (lambda (x y) (b y x)))`

All you need is λ — числа

Симулация на естествени числа (*нумерали на Чърч*)

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

All you need is λ — 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 λ — 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 λ — 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 λ — 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 λ — наредени двойки

Можем да симулираме `cons`, `car` и `cdr` чрез `lambda`!

All you need is λ — наредени двойки

Можем да симулираме `cons`, `car` и `cdr` чрез `lambda`!

Вариант №1:

```
(define (lcons x y) (lambda (p) (if p x y)))
(define (lcar z) (z #t))
(define (lcdr z) (z #f))
```

All you need is λ — наредени двойки

Можем да симулираме `cons`, `car` и `cdr` чрез `lambda`!

Вариант №1:

```
(define (lcons x y) (lambda (p) (if p x y)))
(define (lcar z) (z #t))
(define (lcdr z) (z #f))
```

Вариант №2:

```
(define (lcons x y) (lambda (p) (p x y)))
(define (lcar z) (z (lambda (x y) x)))
(define (lcdr z) (z (lambda (x y) y)))
```