Функции от по-висок ред
част 3: lambda

доц. Атанас Семерджиев

1

Специална форма (lambda)

Специалната форма (lambda) има следния общ вид:

```
(lambda (<параметри>) <тяло>)
```

Примери:

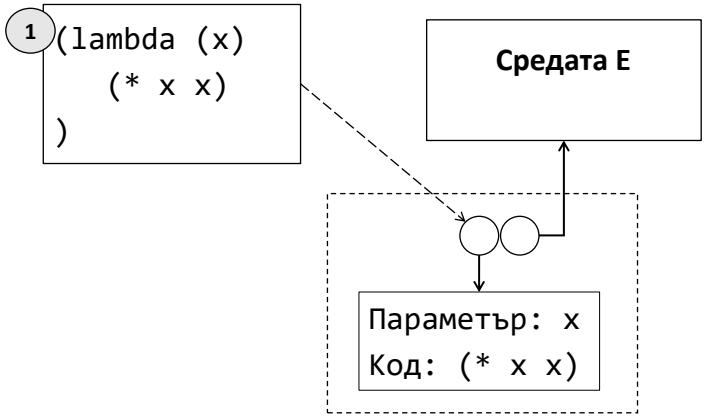
```
(lambda (x) (* x x))
(lambda (x y) (- x y))
```

2

2

Специална форма (lambda)

Нека предположим, че работим в средата E.



3

3

Специална форма (lambda)

Следните две дефиниции са функционално еквивалентни:

```
(define (square x) (* x x))
```

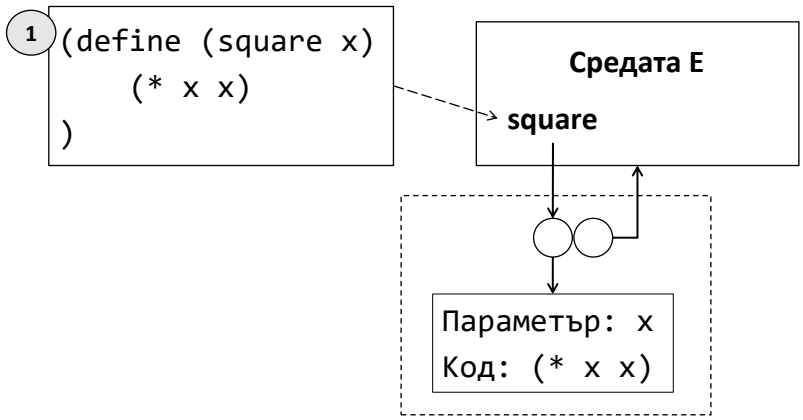
```
(define square (lambda (x) (* x x)))
```

4

4

Дефиниране на процедури

- Нека предположим, че работим в средата E.



5

5

Пример за употреба (1)

Специалната форма (lambda) може да се използва навсякъде, където очакваме процедура. Например:

```
(define (f operation a b)
   (operation a b) )
```

```
(f + 3 4) → 7
(f (lambda (x y) (* x y)) 3 4) → 12
```

6

6

Пример за употреба (2)

Ето как бихме могли да използваме accumulate, за да пресметнем сумата от всички цели числа в интервала [1, 10]:

```
(accumulate +
            0
            (lambda (x) x)
            1
            (lambda (x) (+ x 1))
            10)
```

7

7

Пример за употреба (3)

Специалната форма (lambda) може да се използва като оператор в комбинация. Например:

(`square` 100) → 10000

(`(lambda (x) (* x x))` 100) → 10000

(`(lambda (x y) (+ x y))` 10 20) → 30

8

8

Пример за употреба (4)

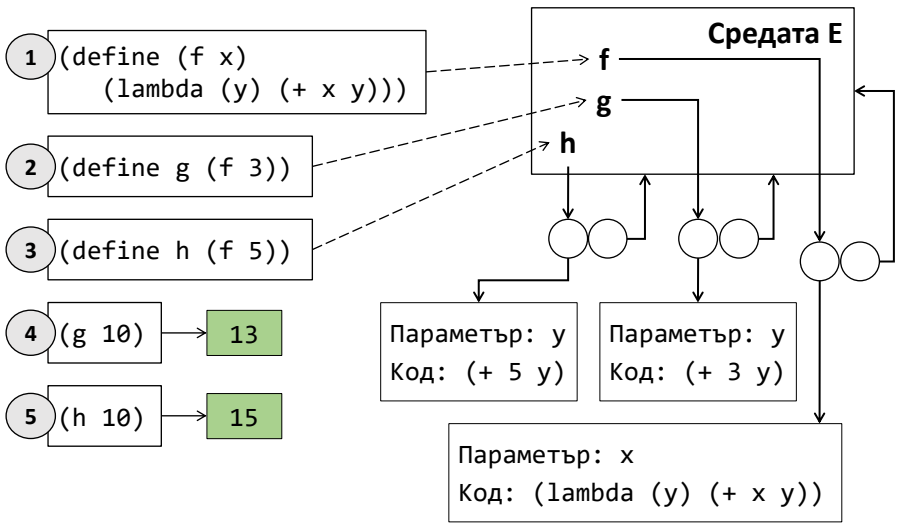
Функциите могат да конструират и връщат нови функции:

```
(define (f x)
   (lambda (y) (+ x y))
)
(define g (f 3))
(define h (f 5))
(g 10) → 13
(h 10) → 15
```

9

9

Процедурите като връщана стойност



10

10

Намиране на производна

Пресмятане на производна в точка

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

$$\frac{f(x+dx)-f(x)}{dx}$$

Намиране на производната като функция

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

11

11

Композиция

```
(define (compose f g)
  (lambda (x)
    (f (g x))
  )
)
```

```
(define sq2 (compose square square))

(sq2 2) → 16; (2 * 2) * (2 * 2)
```

12

12

N-кратно прилагане на функция

```
(define (repeated f N)

 (cond ((= N 0) (lambda (x) x))

       ((= N 1) f)

       (else  (compose f
                       (repeated f (- N 1)))
       )
))
```

13

13

```
; Итеративна версия
(define (repeated f n)
  (define (loop result i)
    (if (= i 0) result
        (loop (compose result f) (- i 1))
    )
  )

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

14

14

N-кратно прилагане на функция

```
(define (repeated f N)

 (cond ((= N 0) (lambda (x) x))

       ((= N 1) f)

       (else  (lambda (x)
                (f ((repeated f (- N 1)) x) )
              )
       )
))
```

Аргументът на f

15

15

Намиране на N-та производна

```
(define dx 0.0000001)

(define (simple-derive f)
  (derive f dx)
)

(repeated simple-derive 3) →

(lambda (f)
  (simple-derive
     (simple-derive
        (simple-derive f)) ))
```

16

16

Намиране на N-та производна

```
(define dx 0.0000001)

(define (simple-derive f)
  (derive f dx)
)

(define (deriveN f n)
  ( (repeated simple-derive n) f)
)
```

17

17

Намиране на N-та производна

```
(define (deriveN f n dx)

  (
    (repeated (lambda (f) (derive f dx))
              n)
    f
  )

)
```

18

18

19