Основни понятия в Scheme

Трифон Трифонов

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

14 октомври 2015 г.

Що за език е Scheme?

- Създаден през 1975 г. от Guy L. Steele и Gerald Jay Sussman
- Диалект на LISP, създаден с учебна цел
- “Structure and Interpretation of Computer Programs”, Abelson & Sussman, MIT Press, 1985.
- Минималистичен синтаксис
- Най-разпространен стандарт: R^5RS

Програмиране на Scheme

- Среда за програмиране: DrRacket
- Има компилатори и интерпретатори
  - Ние ще ползваме интерпретатор
- REPL = Read-Eval-Print-Loop
- Програма = списък от дефиниции
- Изпълнение = оценка на израз

Синтаксис в Scheme

- Атоми
  - Булеви константи (`#f`, `#t`)
  - Числа (`15`, `2/3`, `-1.532`)
  - Знаци (`#\a`, `#\newline`)
  - Низове (`"Scheme"`, `"hi "`)
  - Символи (`f`, `square`, `+`, `find-min`)
- Комбинации

  (<израз$_1$> <израз$_2$> ... <израз$_n$>)

Оценки на атоми

На всеки израз се дава оценка.

- Оценката на булевите константи, знаците, числата и низовете са самите те

Оценки на атоми

На всеки израз се дава оценка.

- Оценката на булевите константи, знаците, числата и низовете са самите те
  - $5 \longrightarrow 5$

Оценки на атоми

На всеки израз се дава оценка.

- Оценката на булевите константи, знаците, числата и низовете са самите те
  - `5` $\longrightarrow$ `5`
  - `#t` $\longrightarrow$ `#t`

Оценки на атоми

На всеки израз се дава оценка.

- Оценката на булевите константи, знаците, числата и низовете са самите те
  - `5` $\longrightarrow$ `5`
  - `#t` $\longrightarrow$ `#t`
  - `#\a` $\longrightarrow$ `#\a`

Оценки на атоми

На всеки израз се дава оценка.

- Оценката на булевите константи, знаците, числата и низовете са самите те
  - `5` $\longrightarrow$ `5`
  - `#t` $\longrightarrow$ `#t`
  - `#\a` $\longrightarrow$ `#\a`
  - `"scheme"` $\longrightarrow$ `"scheme"`

Оценки на атоми

На всеки израз се дава оценка.

- Оценката на булевите константи, знаците, числата и низовете са самите те
  - `5` $\longrightarrow$ `5`
  - `#t` $\longrightarrow$ `#t`
  - `#\a` $\longrightarrow$ `#\a`
  - `"scheme"` $\longrightarrow$ `"scheme"`
- Оценката на символ е стойността, свързана с него

Оценки на атоми

На всеки израз се дава оценка.

- Оценката на булевите константи, знаците, числата и низовете са самите те
  - `5` $\longrightarrow$ `5`
  - `#t` $\longrightarrow$ `#t`
  - `#\a` $\longrightarrow$ `#\a`
  - `"scheme"` $\longrightarrow$ `"scheme"`
- Оценката на символ е стойността, свързана с него
  - `+` $\longrightarrow$ `#<procedure:+>`

Оценки на атоми

На всеки израз се дава оценка.

- Оценката на булевите константи, знаците, числата и низовете са самите те
  - `5` $\longrightarrow$ `5`
  - `#t` $\longrightarrow$ `#t`
  - `#\a` $\longrightarrow$ `#\a`
  - `"scheme"` $\longrightarrow$ `"scheme"`
- Оценката на символ е стойността, свързана с него
  - `+` $\longrightarrow$ `#<procedure:+>`
  - `a` $\longrightarrow$ Грешка!

Оценки на атоми

На всеки израз се дава оценка.

- Оценката на булевите константи, знаците, числата и низовете са самите те
  - `5` $\longrightarrow$ `5`
  - `#t` $\longrightarrow$ `#t`
  - `#\a` $\longrightarrow$ `#\a`
  - `"scheme"` $\longrightarrow$ `"scheme"`
- Оценката на символ е стойността, свързана с него
  - `+` $\longrightarrow$ `#<procedure:+>`
  - `a` $\longrightarrow$ Грешка!
  - `(define a 5)`

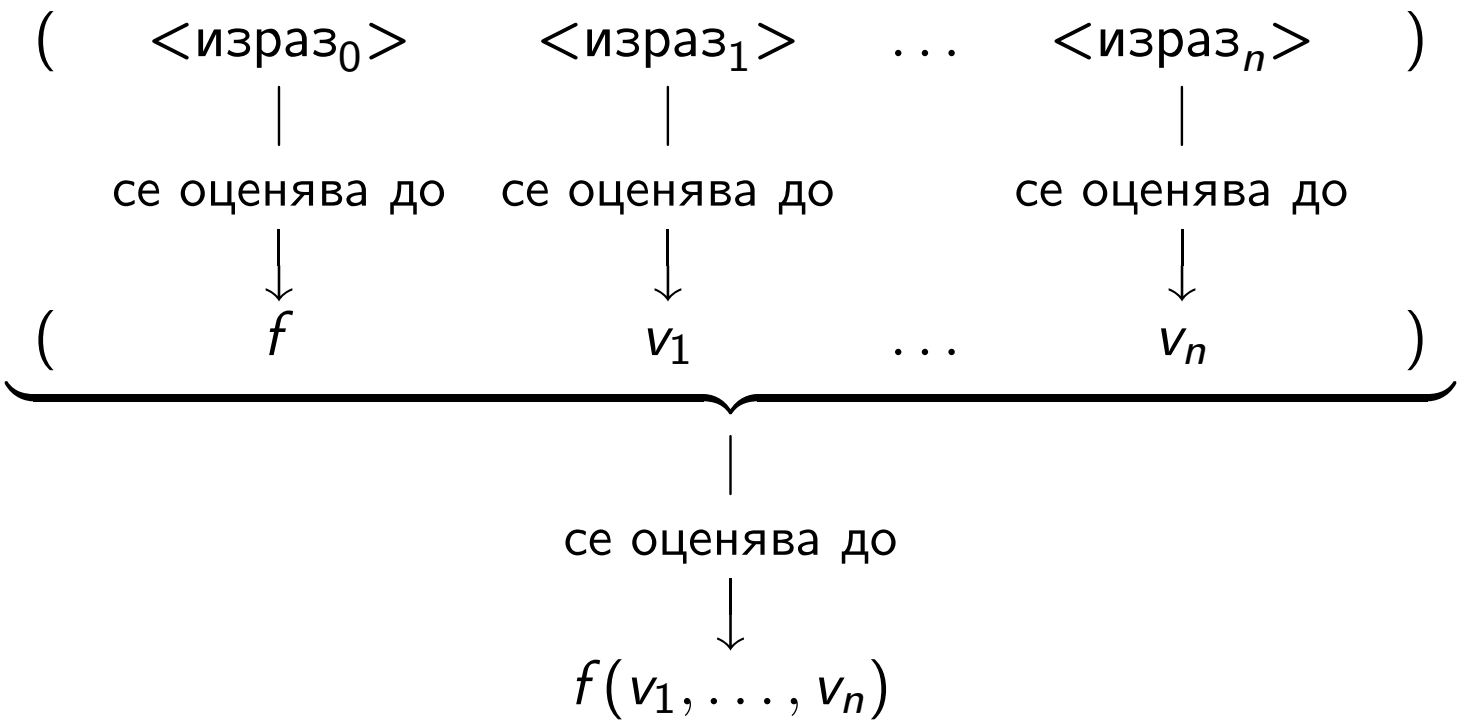
Оценки на атоми

На всеки израз се дава оценка.

- Оценката на булевите константи, знаците, числата и низовете са самите те
  - `5` $\longrightarrow$ `5`
  - `#t` $\longrightarrow$ `#t`
  - `#\a` $\longrightarrow$ `#\a`
  - `"scheme"` $\longrightarrow$ `"scheme"`
- Оценката на символ е стойността, свързана с него
  - `+` $\longrightarrow$ `#<procedure:+>`
  - `a` $\longrightarrow$ Грешка!
  - `(define a 5)`
  - `a` $\longrightarrow$ `5`

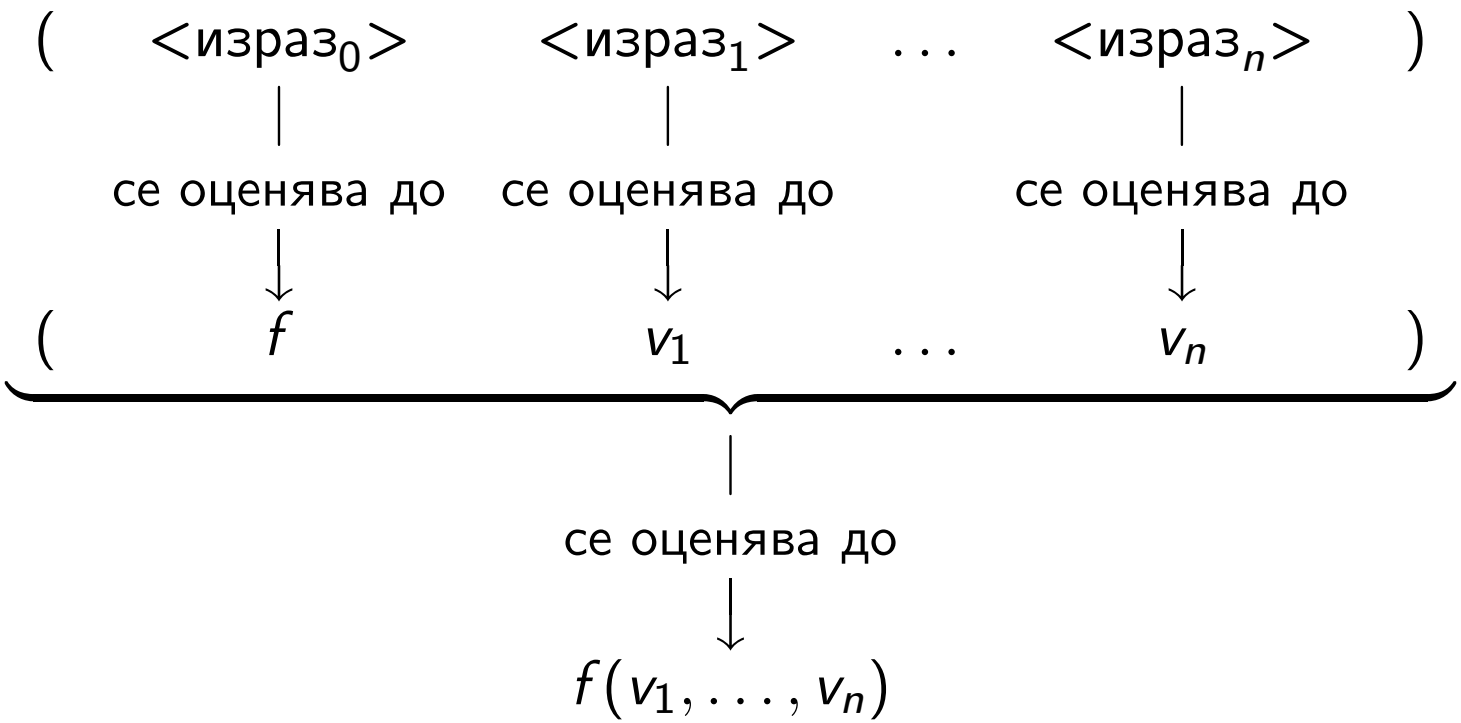
Основно правило за оценяване

Оценка на комбинация **(основно правило за оценяване)**



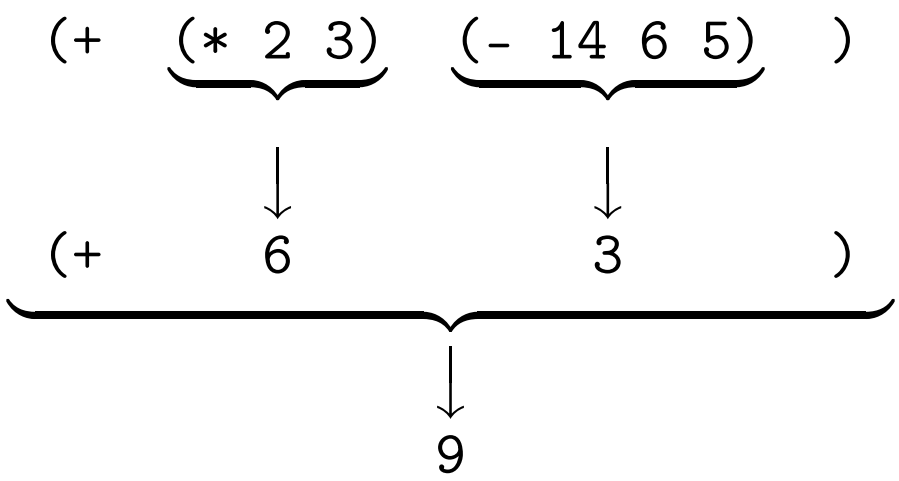
Основно правило за оценяване

Оценка на комбинация **(основно правило за оценяване)**



Ако f не е функция — грешка!

Пример за оценяване на комбинации



(1 2 3) $\longrightarrow$ Грешка!

Дефиниция на символи

- `(define` <символ> <израз>`)`

Дефиниция на символи

- `(define` <символ> <израз>`)`
- Оценява <израз> и свързва <символ> с оценката му.

Дефиниция на символи

- `(define` <символ> <израз>`)`
- Оценява <израз> и свързва <символ> с оценката му.
- Примери:

Дефиниция на символи

- `(define` <символ> <израз>`)`
- Оценява <израз> и свързва <символ> с оценката му.
- Примери:
  - `(define s "Scheme is cool")`

Дефиниция на символи

- `(define` <символ> <израз>`)`
- Оценява <израз> и свързва <символ> с оценката му.
- Примери:
  - `(define s "Scheme is cool")`
  - `s` $\longrightarrow$ `"Scheme is cool"`

Дефиниция на символи

- `(define` <символ> <израз>`)`
- Оценява <израз> и свързва <символ> с оценката му.
- Примери:
  - `(define s "Scheme is cool")`
  - `s` $\longrightarrow$ `"Scheme is cool"`
  - `(define x 2.5)`

Дефиниция на символи

- `(define` <символ> <израз>`)`
- Оценява <израз> и свързва <символ> с оценката му.
- Примери:
  - `(define s "Scheme is cool")`
  - `s` $\longrightarrow$ `"Scheme is cool"`
  - `(define x 2.5)`
  - `x` $\longrightarrow$ `2.5`

Дефиниция на символи

- `(define` <символ> <израз>`)`
- Оценява <израз> и свързва <символ> с оценката му.
- Примери:
  - `(define s "Scheme is cool")`
  - `s` $\longrightarrow$ `"Scheme is cool"`
  - `(define x 2.5)`
  - `x` $\longrightarrow$ `2.5`
  - `(+ x 3.2)` $\longrightarrow$ `5.7`

Дефиниция на символи

- `(define` <символ> <израз>`)`
- Оценява <израз> и свързва <символ> с оценката му.
- Примери:
  - `(define s "Scheme is cool")`
  - `s` $\longrightarrow$ `"Scheme is cool"`
  - `(define x 2.5)`
  - `x` $\longrightarrow$ `2.5`
  - `(+ x 3.2)` $\longrightarrow$ `5.7`
  - `(define y (+ x 3.2))`

Дефиниция на символи

- `(define` <символ> <израз>`)`
- Оценява <израз> и свързва <символ> с оценката му.
- Примери:
  - `(define s "Scheme is cool")`
  - `s` $\longrightarrow$ `"Scheme is cool"`
  - `(define x 2.5)`
  - `x` $\longrightarrow$ `2.5`
  - `(+ x 3.2)` $\longrightarrow$ `5.7`
  - `(define y (+ x 3.2))`
  - `(> y 3)` $\longrightarrow$ `#t`

Дефиниция на символи

- `(define` <символ> <израз>`)`
- Оценява <израз> и свързва <символ> с оценката му.
- Примери:
  - `(define s "Scheme is cool")`
  - `s` $\longrightarrow$ `"Scheme is cool"`
  - `(define x 2.5)`
  - `x` $\longrightarrow$ `2.5`
  - `(+ x 3.2)` $\longrightarrow$ `5.7`
  - `(define y (+ x 3.2))`
  - `(> y 3)` $\longrightarrow$ `#t`
  - `(define z (+ y z))` $\longrightarrow$ Грешка!

Специални форми

- По основното правило ли се оценява `(define x 2.5)`?

Специални форми

- По основното правило ли се оценява `(define x 2.5)`?
- Не!

Специални форми

- По основното правило ли се оценява `(define x 2.5)`?
- Не!
- В синтаксиса на Scheme има конструкции, които са изключение от стандартното правило

Специални форми

- По основното правило ли се оценява `(define x 2.5)`?
- Не!
- В синтаксиса на Scheme има конструкции, които са изключение от стандартното правило
- Такива конструкции се наричат **специални форми**

Специални форми

- По основното правило ли се оценява `(define x 2.5)`?
- Не!
- В синтаксиса на Scheme има конструкции, които са изключение от стандартното правило
- Такива конструкции се наричат **специални форми**
- `define` е пример за специална форма

Цитиране

- `(quote` <израз>`)`

Цитиране

- `(quote` <израз>`)`
- Алтернативен запис: `'`<израз>

Цитиране

- `(quote` <израз>`)`
- Алтернативен запис: `'`<израз>
- Оценката на `(quote` <израз>`)` или `'`<израз> е самият <израз>

Цитиране

- `(quote` <израз>`)`
- Алтернативен запис: `'`<израз>
- Оценката на `(quote` <израз>`)` или `'`<израз> е самият <израз>
- **Примери:**

Цитиране

- `(quote` <израз>`)`
- Алтернативен запис: `'`<израз>
- Оценката на `(quote` <израз>`)` или `'`<израз> е самият <израз>
- **Примери:**
  - `'2` $\longrightarrow$ `2`

Цитиране

- `(quote` <израз>`)`
- Алтернативен запис: `'`<израз>
- Оценката на `(quote` <израз>`)` или `'`<израз> е самият <израз>
- **Примери:**
  - `'2` $\longrightarrow$ `2`
  - `'+` $\longrightarrow$ `+`

Цитиране

- `(quote` <израз>`)`
- Алтернативен запис: `'`<израз>
- Оценката на `(quote` <израз>`)` или `'`<израз> е самият <израз>
- **Примери:**
  - `'2` $\longrightarrow$ `2`
  - `'+` $\longrightarrow$ `+`
  - `'(+ 2 3)` $\longrightarrow$ `(+ 2 3)`

Цитиране

- `(quote` <израз>`)`
- Алтернативен запис: `'`<израз>
- Оценката на `(quote` <израз>`)` или `'`<израз> е самият <израз>
- **Примери:**
  - `'2` $\longrightarrow$ `2`
  - `'+` $\longrightarrow$ `+`
  - `'(+ 2 3)` $\longrightarrow$ `(+ 2 3)`
  - `(quote quote)` $\longrightarrow$ `quote`

Цитиране

- `(quote` <израз>`)`
- Алтернативен запис: `'`<израз>
- Оценката на `(quote` <израз>`)` или `'`<израз> е самият <израз>
- **Примери:**
  - `'2` $\longrightarrow$ `2`
  - `'+` $\longrightarrow$ `+`
  - `'(+ 2 3)` $\longrightarrow$ `(+ 2 3)`
  - `(quote quote)` $\longrightarrow$ `quote`
  - `('+ 2 3)` $\longrightarrow$ Грешка!

Цитиране

- `(quote` <израз>`)`
- Алтернативен запис: `'`<израз>
- Оценката на `(quote` <израз>`)` или `'`<израз> е самият <израз>
- **Примери:**
  - `'2` $\longrightarrow$ `2`
  - `'+` $\longrightarrow$ `+`
  - `'(+ 2 3)` $\longrightarrow$ `(+ 2 3)`
  - `(quote quote)` $\longrightarrow$ `quote`
  - `('+ 2 3)` $\longrightarrow$ Грешка!
  - `(/ 2 0)` $\longrightarrow$ Грешка!

Цитиране

- `(quote` <израз>`)`
- Алтернативен запис: `'`<израз>
- Оценката на `(quote` <израз>`)` или `'`<израз> е самият <израз>
- **Примери:**
  - `'2` $\longrightarrow$ `2`
  - `'+` $\longrightarrow$ `+`
  - `'(+ 2 3)` $\longrightarrow$ `(+ 2 3)`
  - `(quote quote)` $\longrightarrow$ `quote`
  - `('+ 2 3)` $\longrightarrow$ Грешка!
  - `(/ 2 0)` $\longrightarrow$ Грешка!
  - `'(/ 2 0)` $\longrightarrow$ `(/ 2 0)`

Цитиране

- `(quote` <израз>`)`
- Алтернативен запис: `'`<израз>
- Оценката на `(quote` <израз>`)` или `'`<израз> е самият <израз>
- **Примери:**
  - `'2` $\longrightarrow$ `2`
  - `'+` $\longrightarrow$ `+`
  - `'(+ 2 3)` $\longrightarrow$ `(+ 2 3)`
  - `(quote quote)` $\longrightarrow$ `quote`
  - `('+ 2 3)` $\longrightarrow$ Грешка!
  - `(/ 2 0)` $\longrightarrow$ Грешка!
  - `'(/ 2 0)` $\longrightarrow$ `(/ 2 0)`
  - `'(+ 1 '(* 3 4))` $\longrightarrow$ `(+ 1 (quote (* 3 4)))`

Дефиниция на функции

- `(define (`<функция> {<параметър>}`)` <тяло>`)`

Дефиниция на функции

- `(define (`<функция> {<параметър>}`)` <тяло>`)`
- <функция> и <параметър> са символи

Дефиниция на функции

- `(define (`<функция> {<параметър>}`)` <тяло>`)`
- <функция> и <параметър> са символи
- <тяло> е <израз>

Дефиниция на функции

- `(define (`<функция> {<параметър>}`)` <тяло>`)`
- <функция> и <параметър> са символи
- <тяло> е <израз>
- Символът <функция> се свързва с поредица от инструкции, които пресмятат <тяло> при подадени стойности на <параметър>

Примери за дефиниция на функции

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

Примери за дефиниция на функции

- `(define (square x) (* x x))`
- `(square 5)` $\longrightarrow$ `25`

Примери за дефиниция на функции

- `(define (square x) (* x x))`
- `(square 5)` $\longrightarrow$ `25`
- `(define (1+ k) (+ k 1))`

Примери за дефиниция на функции

- `(define (square x) (* x x))`
- `(square 5)` $\longrightarrow$ `25`
- `(define (1+ k) (+ k 1))`
- `(square (1+ (square 3)))` $\longrightarrow$ `100`

Примери за дефиниция на функции

- `(define (square x) (* x x))`
- `(square 5)` $\longrightarrow$ `25`
- `(define (1+ k) (+ k 1))`
- `(square (1+ (square 3)))` $\longrightarrow$ `100`
- `(define (f x y) (+ (square (1+ x)) (square y) 5))`

Примери за дефиниция на функции

- `(define (square x) (* x x))`
- `(square 5)` $\longrightarrow$ `25`
- `(define (1+ k) (+ k 1))`
- `(square (1+ (square 3)))` $\longrightarrow$ `100`
- `(define (f x y) (+ (square (1+ x)) (square y) 5))`
- `(f 2 4)` $\longrightarrow$ `30`

Примери за дефиниция на функции

- `(define (square x) (* x x))`
- `(square 5)` $\longrightarrow$ `25`
- `(define (1+ k) (+ k 1))`
- `(square (1+ (square 3)))` $\longrightarrow$ `100`
- `(define (f x y) (+ (square (1+ x)) (square y) 5))`
- `(f 2 4)` $\longrightarrow$ `30`
- `(define (g x) (- (g (+ x 1)) 1))`

Примери за дефиниция на функции

- `(define (square x) (* x x))`
- `(square 5)` $\longrightarrow$ `25`
- `(define (1+ k) (+ k 1))`
- `(square (1+ (square 3)))` $\longrightarrow$ `100`
- `(define (f x y) (+ (square (1+ x)) (square y) 5))`
- `(f 2 4)` $\longrightarrow$ `30`
- `(define (g x) (- (g (+ x 1)) 1))`
- `(g 0)` $\longrightarrow$ `...`

Примери за дефиниция на функции

- `(define (square x) (* x x))`
- `(square 5)` $\longrightarrow$ `25`
- `(define (1+ k) (+ k 1))`
- `(square (1+ (square 3)))` $\longrightarrow$ `100`
- `(define (f x y) (+ (square (1+ x)) (square y) 5))`
- `(f 2 4)` $\longrightarrow$ `30`
- `(define (g x) (- (g (+ x 1)) 1))`
- `(g 0)` $\longrightarrow$ `...`
- `(define (h) (+ 2 3))`

Примери за дефиниция на функции

- `(define (square x) (* x x))`
- `(square 5)` $\longrightarrow$ `25`
- `(define (1+ k) (+ k 1))`
- `(square (1+ (square 3)))` $\longrightarrow$ `100`
- `(define (f x y) (+ (square (1+ x)) (square y) 5))`
- `(f 2 4)` $\longrightarrow$ `30`
- `(define (g x) (- (g (+ x 1)) 1))`
- `(g 0)` $\longrightarrow$ `...`
- `(define (h) (+ 2 3))`
- `h` $\longrightarrow$ `#<procedure:h>`

Примери за дефиниция на функции

- `(define (square x) (* x x))`
- `(square 5)` $\longrightarrow$ `25`
- `(define (1+ k) (+ k 1))`
- `(square (1+ (square 3)))` $\longrightarrow$ `100`
- `(define (f x y) (+ (square (1+ x)) (square y) 5))`
- `(f 2 4)` $\longrightarrow$ `30`
- `(define (g x) (- (g (+ x 1)) 1))`
- `(g 0)` $\longrightarrow$ `...`
- `(define (h) (+ 2 3))`
- `h` $\longrightarrow$ `#<procedure:h>`
- `(h)` $\longrightarrow$ `5`

Оценяване на комбинации с дефинирани функции

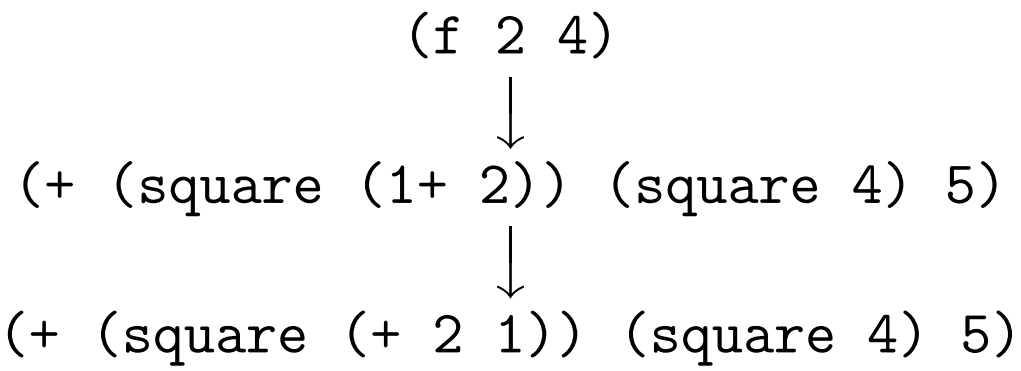
```
(f 2 4)
```

Оценяване на комбинации с дефинирани функции

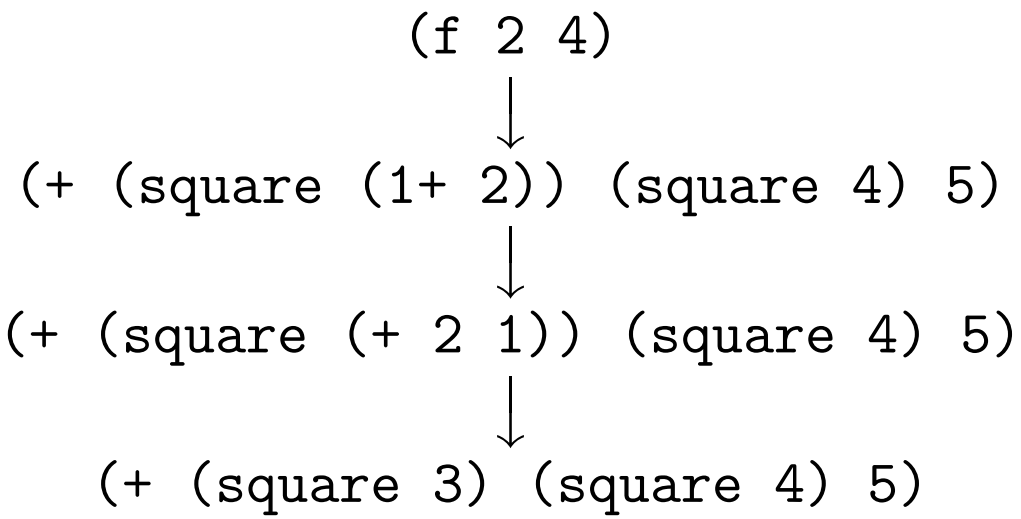
```
(f 2 4)
```

```
(+ (square (1+ 2)) (square 4) 5)
```

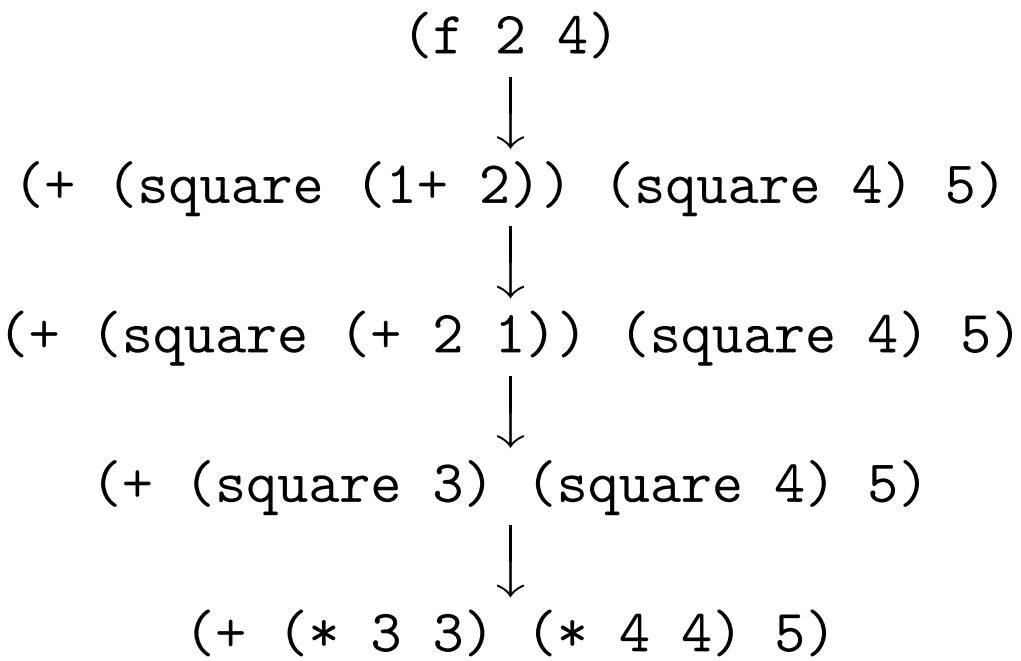
Оценяване на комбинации с дефинирани функции



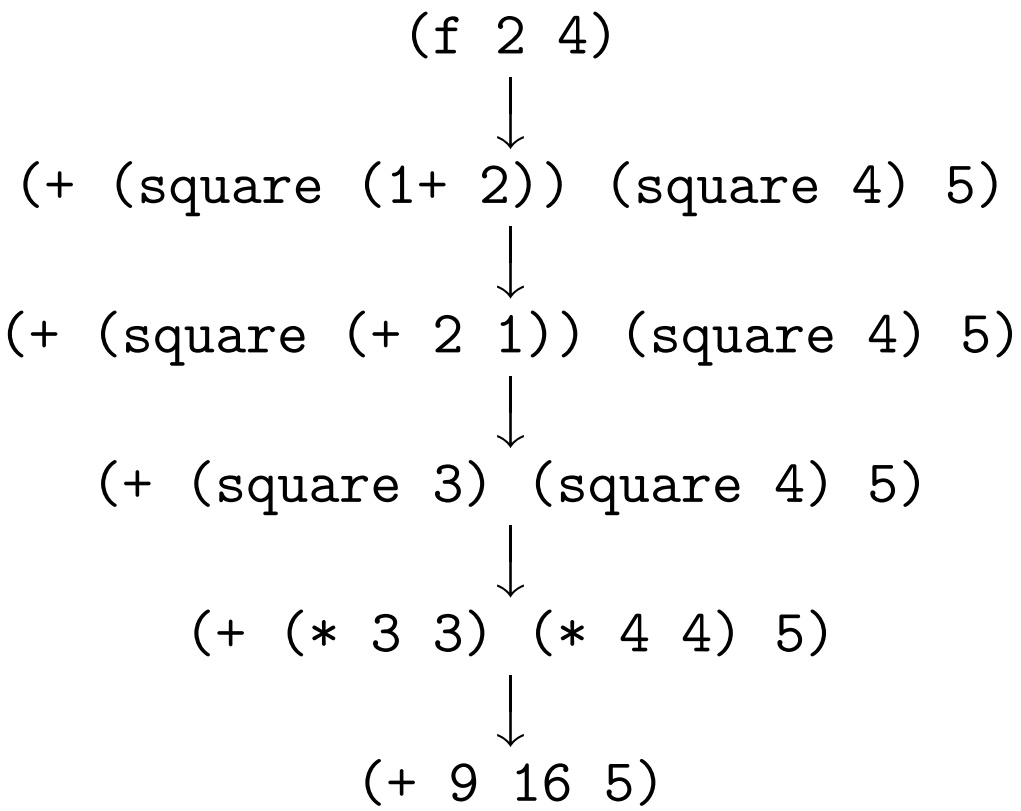
Оценяване на комбинации с дефинирани функции



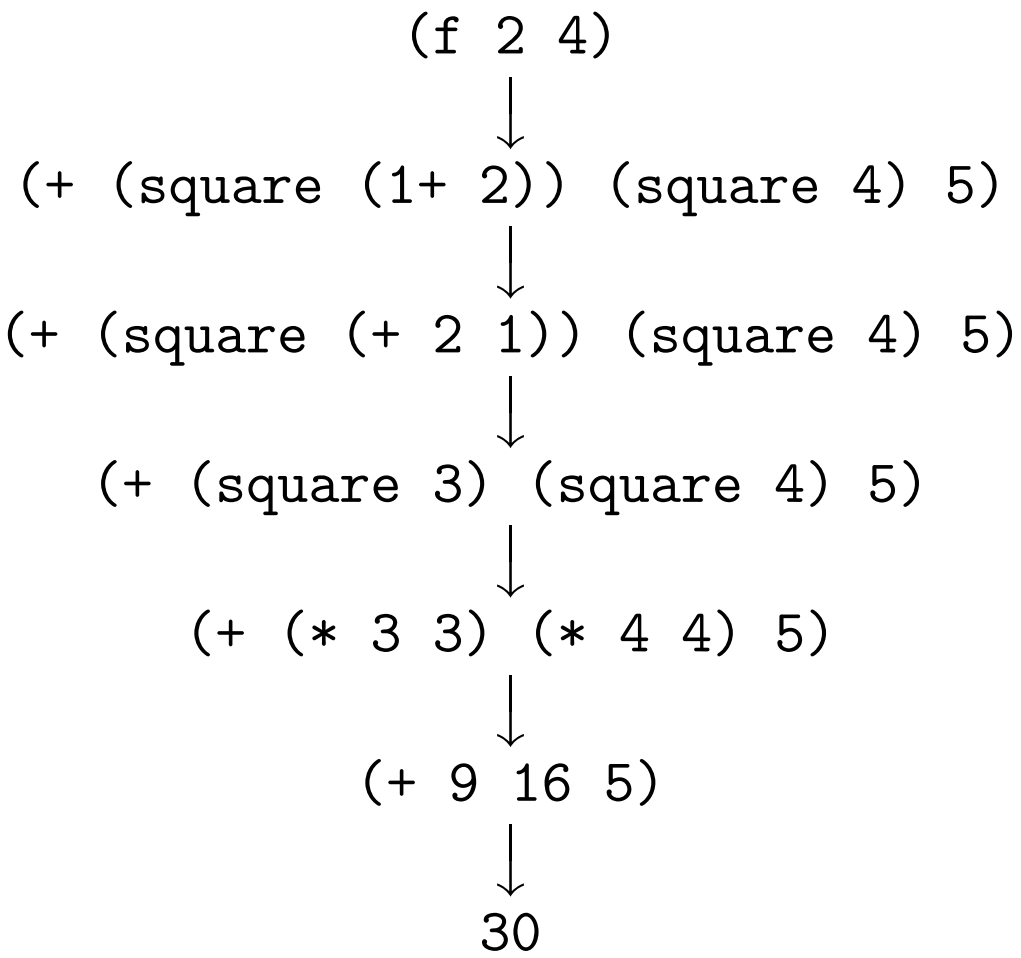
Оценяване на комбинации с дефинирани функции



Оценяване на комбинации с дефинирани функции



Оценяване на комбинации с дефинирани функции



Стандартни числови функции

Аритметични операции

`+`, `-`, `*`, `/`

Други числови функции

`remainder`, `quotient`, `max`, `min`, `gcd`, `lcm`

Функции за закръгляне

`floor`, `ceiling`, `round`

Функции над дробни числа

`exp`, `log`, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `expt`, `sqrt`

Стандартни предикати

Предикати за сравнение на числа
<, >, =, <=, >=

Числови предикати
`zero?`, `negative?`, `positive?`, `odd?`, `even?`

Предикати за проверка на тип
`boolean?`, `number?`, `char?`, `string?`, `symbol?`, `procedure?`

Условна форма `if`

- `(if` <условие> <израз$_1$> <израз$_2$>`)`

Условна форма `if`

- `(if` <условие> <израз$_1$> <израз$_2$>`)`
- Оценява се `<условие>`

Условна форма `if`

- `(if` <условие> <израз$_1$> <израз$_2$>`)`
- Оценява се <условие>
  - Ако оценката е `#t` — връща се оценката на <израз$_1$>

Условна форма `if`

- `(if` <условие> <израз$_1$> <израз$_2$>`)`
- Оценява се <условие>
  - Ако оценката е `#t` — връща се оценката на <израз$_1$>
  - Ако оценката е `#f` — връща се оценката на <израз$_2$>

Условна форма `if`

- `(if` <условие> <израз$_1$> <израз$_2$>`)`
- Оценява се <условие>
  - Ако оценката е `#t` — връща се оценката на <израз$_1$>
  - Ако оценката е `#f` — връща се оценката на <израз$_2$>
- `if` е специална форма!

Примери с условната форма `if`

- `(if (< 3 5) (+ 7 3) (- 4 2))` $\longrightarrow$ `10`

Примери с условната форма `if`

- `(if (< 3 5) (+ 7 3) (- 4 2))` $\longrightarrow$ `10`
- `(define (abs x) (if (< x 0) (- x) x))`

Примери с условната форма `if`

- `(if (< 3 5) (+ 7 3) (- 4 2))` $\longrightarrow$ `10`
- `(define (abs x) (if (< x 0) (- x) x))`
- `(abs -5)` $\longrightarrow$ `5`, `(abs (+ 1 2))` $\longrightarrow$ `3`

Примери с условната форма `if`

- `(if (< 3 5) (+ 7 3) (- 4 2))` $\longrightarrow$ `10`
- `(define (abs x) (if (< x 0) (- x) x))`
- `(abs -5)` $\longrightarrow$ `5`, `(abs (+ 1 2))` $\longrightarrow$ `3`
- `(define (f x) (if (< x 5) (+ x 2) "Error"))`

Примери с условната форма if

- `(if (< 3 5) (+ 7 3) (- 4 2))` $\longrightarrow$ `10`
- `(define (abs x) (if (< x 0) (- x) x))`
- `(abs -5)` $\longrightarrow$ `5`, `(abs (+ 1 2))` $\longrightarrow$ `3`
- `(define (f x) (if (< x 5) (+ x 2) "Error"))`
- `(f 3)` $\longrightarrow$ `5`, `(f 5)` $\longrightarrow$ `"Error"`

Примери с условната форма if

- `(if (< 3 5) (+ 7 3) (- 4 2))` $\longrightarrow$ `10`
- `(define (abs x) (if (< x 0) (- x) x))`
- `(abs -5)` $\longrightarrow$ `5`, `(abs (+ 1 2))` $\longrightarrow$ `3`
- `(define (f x) (if (< x 5) (+ x 2) "Error"))`
- `(f 3)` $\longrightarrow$ `5`, `(f 5)` $\longrightarrow$ `"Error"`
- `(define (g x y) (if (< x y) (+ x y) (* x y)))`

Примери с условната форма if

- `(if (< 3 5) (+ 7 3) (- 4 2))` $\longrightarrow$ `10`
- `(define (abs x) (if (< x 0) (- x) x))`
- `(abs -5)` $\longrightarrow$ `5`, `(abs (+ 1 2))` $\longrightarrow$ `3`
- `(define (f x) (if (< x 5) (+ x 2) "Error"))`
- `(f 3)` $\longrightarrow$ `5`, `(f 5)` $\longrightarrow$ `"Error"`
- `(define (g x y) (if (< x y) (+ x y) (* x y)))`
- `(define (g x y) ((if (< x y) + *) x y))`

Примери с условната форма if

- `(if (< 3 5) (+ 7 3) (- 4 2))` $\longrightarrow$ `10`
- `(define (abs x) (if (< x 0) (- x) x))`
- `(abs -5)` $\longrightarrow$ `5`, `(abs (+ 1 2))` $\longrightarrow$ `3`
- `(define (f x) (if (< x 5) (+ x 2) "Error"))`
- `(f 3)` $\longrightarrow$ `5`, `(f 5)` $\longrightarrow$ `"Error"`
- `(define (g x y) (if (< x y) (+ x y) (* x y)))`
- `(define (g x y) ((if (< x y) + *) x y))`
- `(g 2 3)` $\longrightarrow$ `5`, `(g 3 2)` $\longrightarrow$ `6`

Форма за многозначен избор cond

- `(cond {(<условие> <израз>)} [(else <израз>)])`

Форма за многозначен избор cond

- `(cond {(`<условие> <израз>`)} [(else` <израз>`)])`
- `(cond (`<условие$_1$> <израз$_1$>`)`
  `(`<условие$_2$> <израз$_2$>`)`
  ...
  `(`<условие$_n$> <израз$_n$>`)`
  `(else` <израз$_{n+1}$>`))`

Форма за многозначен избор cond

- `(cond {(`<условие> <израз>`)} [(else` <израз>`)])`
- `(cond (`<условие$_1$> <израз$_1$>`)`
  `(`<условие$_2$> <израз$_2$>`)`
  ...
  `(`<условие$_n$> <израз$_n$>`)`
  `(else` <израз$_{n+1}$>`))`
- Оценява се <условие$_1$>, при `#t` се връща <израз$_1$>, а при `#f`:

Форма за многозначен избор cond

- `(cond {(`<условие> <израз>`)} [(else` <израз>`)])`
- `(cond (`<условие$_1$> <израз$_1$>`)`
  `(`<условие$_2$> <израз$_2$>`)`
  ...
  `(`<условие$_n$> <израз$_n$>`)`
  `(else` <израз$_{n+1}$>`))`
- Оценява се <условие$_1$>, при `#t` се връща <израз$_1$>, а при `#f`:
- Оценява се <условие$_2$>, при `#t` се връща <израз$_2$>, а при `#f`:

Форма за многозначен избор cond

- `(cond {(`<условие> <израз>`)} [(else` <израз>`)])`
- `(cond (`<условие$_1$> <израз$_1$>`)`
  `(`<условие$_2$> <израз$_2$>`)`
  ...
  `(`<условие$_n$> <израз$_n$>`)`
  `(else` <израз$_{n+1}$>`))`
- Оценява се <условие$_1$>, при `#t` се връща <израз$_1$>, а при `#f`:
- Оценява се <условие$_2$>, при `#t` се връща <израз$_2$>, а при `#f`:
- ...

Форма за многозначен избор cond

- `(cond {(`<условие> <израз>`)} [(else` <израз>`)])`
- `(cond (`<условие$_1$> <израз$_1$>`)`
  `(`<условие$_2$> <израз$_2$>`)`
  ...
  `(`<условие$_n$> <израз$_n$>`)`
  `(else` <израз$_{n+1}$>`))`
- Оценява се <условие$_1$>, при `#t` се връща <израз$_1$>, а при `#f`:
- Оценява се <условие$_2$>, при `#t` се връща <израз$_2$>, а при `#f`:
- ...
- Оценява се <условие$_n$>, при `#t` се връща <израз$_n$>, а при `#f`:

Форма за многозначен избор cond

- `(cond {(`<условие> <израз>`)} [(else` <израз>`)])`
- `(cond (`<условие$_1$> <израз$_1$>`)`
  `(`<условие$_2$> <израз$_2$>`)`
  ...
  `(`<условие$_n$> <израз$_n$>`)`
  `(else` <израз$_{n+1}$>`))`
- Оценява се <условие$_1$>, при `#t` се връща <израз$_1$>, а при `#f`:
- Оценява се <условие$_2$>, при `#t` се връща <израз$_2$>, а при `#f`:
- ...
- Оценява се <условие$_n$>, при `#t` се връща <израз$_n$>, а при `#f`:
- Връща се <израз$_{n+1}$>

Пример с формата cond

```
(define (grade x)
  (cond ((>= x 5.5) "Отличен")
        ((>= x 4.5) "Много добър")
        ((>= x 3.5) "Добър")
        ((>= x 3)   "Среден")
        (else       "Слаб")))
```

Форма за разглеждане на случаи case

- `(case` <тест> `{(({`<случай>`}) `<израз>`)} [(else` <израз>`)])`

Форма за разглеждане на случаи case

- (case <тест> {(({<случай>}) <израз>)} [(else <израз>)])
- (case <тест> ((<случай$_{1,1}$> ... <случай$_{1,k_1}$>) <израз$_1$>)
  ((<случай$_{2,1}$> ... <случай$_{2,k_2}$>) <израз$_2$>)
  ...
  ((<случай$_{n,1}$> ... <случай$_{n,k_n}$>) <израз$_n$>)
  (else <израз$_{n+1}$>))

Форма за разглеждане на случаи case

- (case <тест> {(({<случай>}) <израз>)} [(else <израз>)])
- (case <тест> ((<случай$_{1,1}$> ... <случай$_{1,k_1}$>) <израз$_1$>)
  ((<случай$_{2,1}$> ... <случай$_{2,k_2}$>) <израз$_2$>)
  ...
  ((<случай$_{n,1}$> ... <случай$_{n,k_n}$>) <израз$_n$>)
  (else <израз$_{n+1}$>))
- Оценява се <тест>

Форма за разглеждане на случаи case

- (case <тест> {(({<случай>}) <израз>)} [(else <израз>)])
- (case <тест> ((<случай$_{1,1}$> ... <случай$_{1,k_1}$>) <израз$_1$>)
  ((<случай$_{2,1}$> ... <случай$_{2,k_2}$>) <израз$_2$>)
  ...
  ((<случай$_{n,1}$> ... <случай$_{n,k_n}$>) <израз$_n$>)
  (else <израз$_{n+1}$>))
- Оценява се <тест>
- при някое от <случай$_{1,1}$>...<случай$_{1,k_1}$> $\rightarrow$ <израз$_1$>, иначе:

Форма за разглеждане на случаи case

- (case <тест> {(({<случай>}) <израз>)} [(else <израз>)])
- (case <тест> ((<случай$_{1,1}$> ... <случай$_{1,k_1}$>) <израз$_1$>)
  ((<случай$_{2,1}$> ... <случай$_{2,k_2}$>) <израз$_2$>)
  ...
  ((<случай$_{n,1}$> ... <случай$_{n,k_n}$>) <израз$_n$>)
  (else <израз$_{n+1}$>))
- Оценява се <тест>
- при някое от <случай$_{1,1}$>...<случай$_{1,k_1}$> $\rightarrow$ <израз$_1$>, иначе:
- при някое от <случай$_{2,1}$>...<случай$_{2,k_2}$> $\rightarrow$ <израз$_2$>, иначе:

Форма за разглеждане на случаи case

- (case <тест> {(({<случай>}) <израз>)} [(else <израз>)])
- (case <тест> ((<случай$_{1,1}$> ... <случай$_{1,k_1}$>) <израз$_1$>)
  ((<случай$_{2,1}$> ... <случай$_{2,k_2}$>) <израз$_2$>)
  ...
  ((<случай$_{n,1}$> ... <случай$_{n,k_n}$>) <израз$_n$>)
  (else <израз$_{n+1}$>))
- Оценява се <тест>
- при някое от <случай$_{1,1}$>...<случай$_{1,k_1}$> $\rightarrow$ <израз$_1$>, иначе:
- при някое от <случай$_{2,1}$>...<случай$_{2,k_2}$> $\rightarrow$ <израз$_2$>, иначе:
- ...

Форма за разглеждане на случаи case

- (case <тест> {(({<случай>}) <израз>)} [(else <израз>)])
- (case <тест> ((<случай$_{1,1}$> ... <случай$_{1,k_1}$>) <израз$_1$>)
  ((<случай$_{2,1}$> ... <случай$_{2,k_2}$>) <израз$_2$>)
  ...
  ((<случай$_{n,1}$> ... <случай$_{n,k_n}$>) <израз$_n$>)
  (else <израз$_{n+1}$>))
- Оценява се <тест>
- при някое от <случай$_{1,1}$>...<случай$_{1,k_1}$> $\rightarrow$ <израз$_1$>, иначе:
- при някое от <случай$_{2,1}$>...<случай$_{2,k_2}$> $\rightarrow$ <израз$_2$>, иначе:
- ...
- при някое от <случай$_{n,1}$>...<случай$_{n,k_n}$> $\rightarrow$ <израз$_n$>, иначе:

Форма за разглеждане на случаи case

- `(case` <тест> `{(({`<случай>`}) `<израз>`)} [(else` <израз>`)])`
- `(case` <тест> `((`<случай$_{1,1}$> ... <случай$_{1,k_1}$>`)` <израз$_1$>`)`
  `((`<случай$_{2,1}$> ... <случай$_{2,k_2}$>`)` <израз$_2$>`)`
  ...
  `((`<случай$_{n,1}$> ... <случай$_{n,k_n}$>`)` <израз$_n$>`)`
  `(else` <израз$_{n+1}$>`))`
- Оценява се <тест>
- при някое от <случай$_{1,1}$>...<случай$_{1,k_1}$> $\rightarrow$ <израз$_1$>, иначе:
- при някое от <случай$_{2,1}$>...<случай$_{2,k_2}$> $\rightarrow$ <израз$_2$>, иначе:
- ...
- при някое от <случай$_{n,1}$>...<случай$_{n,k_n}$> $\rightarrow$ <израз$_n$>, иначе:
- Връща се <израз$_{n+1}$>

Пример с формата cond

```
(define (days-in-month m y)
  (case m
    ((1 3 5 7 8 10 12) 31)
    ((4 6 9 11) 30)
    (else (if (leap? y) 29 28))))
```

Логически операции

- `(not` <булев-израз>`)`

Логически операции

- `(not` <булев-израз>`)`
  - Връща отрицанието на <булев-израз>

Логически операции

- `(not` <булев-израз>`)`
  - Връща отрицанието на <булев-израз>
- `(and` {<булев-израз>}`)`

Логически операции

- (not <булев-израз>)
  - Връща отрицанието на <булев-израз>
- (and {<булев-израз>})
- (and <булев-израз$_1$> <булев-израз$_2$> ... <булев-израз$_n$>)

Логически операции

- `(not` <булев-израз>`)`
  - Връща отрицанието на <булев-израз>
- `(and` {<булев-израз>}`)`
- `(and` <булев-израз$_1$> <булев-израз$_2$> . . . <булев-израз$_n$>`)`
  - Оценява последователно всички <булев-израз$_i$>

Логически операции

- `(not` <булев-израз>`)`
  - Връща отрицанието на <булев-израз>
- `(and` {<булев-израз>}`)`
- `(and` <булев-израз$_1$> <булев-израз$_2$> . . . <булев-израз$_n$>`)`
  - Оценява последователно всички <булев-израз$_i$>
  - Ако всички се оценяват до `#t`, връща `#t`

Логически операции

- `(not` <булев-израз>`)`
  - Връща отрицанието на <булев-израз>
- `(and` {<булев-израз>}`)`
- `(and` <булев-израз$_1$> <булев-израз$_2$> . . . <булев-израз$_n$>`)`
  - Оценява последователно всички <булев-израз$_i$>
  - Ако всички се оценяват до `#t`, връща `#t`
  - Ако <булев-израз$_i$> се оценява до `#f`, връща `#f` без да оценява следващите <булев-израз$_{i+1}$> . . . <булев-израз$_n$>

Логически операции

- `(not` <булев-израз>`)`
  - Връща отрицанието на <булев-израз>
- `(and` {<булев-израз>}`)`
- `(and` <булев-израз$_1$> <булев-израз$_2$> . . . <булев-израз$_n$>`)`
  - Оценява последователно всички <булев-израз$_i$>
  - Ако всички се оценяват до `#t`, връща `#t`
  - Ако <булев-израз$_i$> се оценява до `#f`, връща `#f` без да оценява следващите <булев-израз$_{i+1}$> . . . <булев-израз$_n$>
- `(or` {<булев-израз>}`)`

Логически операции

- `(not` <булев-израз>`)`
  - Връща отрицанието на <булев-израз>
- `(and` {<булев-израз>}`)`
- `(and` <булев-израз$_1$> <булев-израз$_2$> . . . <булев-израз$_n$>`)`
  - Оценява последователно всички <булев-израз$_i$>
  - Ако всички се оценяват до `#t`, връща `#t`
  - Ако <булев-израз$_i$> се оценява до `#f`, връща `#f` без да оценява следващите <булев-израз$_{i+1}$> . . . <булев-израз$_n$>
- `(or` {<булев-израз>}`)`
- `(or` <булев-израз$_1$> <булев-израз$_2$> . . . <булев-израз$_n$>`)`

Логически операции

- (not <булев-израз>)
  - Връща отрицанието на <булев-израз>
- (and {<булев-израз>})
- (and <булев-израз$_1$> <булев-израз$_2$> ... <булев-израз$_n$>)
  - Оценява последователно всички <булев-израз$_i$>
  - Ако всички се оценяват до #t, връща #t
  - Ако <булев-израз$_i$> се оценява до #f, връща #f без да оценява следващите <булев-израз$_{i+1}$> ... <булев-израз$_n$>
- (or {<булев-израз>})
- (or <булев-израз$_1$> <булев-израз$_2$> ... <булев-израз$_n$>)
  - Оценява последователно всички <булев-израз$_i$>

Логически операции

- `(not` <булев-израз>`)`
  - Връща отрицанието на <булев-израз>
- `(and` {<булев-израз>}`)`
- `(and` <булев-израз$_1$> <булев-израз$_2$> ... <булев-израз$_n$>`)`
  - Оценява последователно всички <булев-израз$_i$>
  - Ако всички се оценяват до `#t`, връща `#t`
  - Ако <булев-израз$_i$> се оценява до `#f`, връща `#f` без да оценява следващите <булев-израз$_{i+1}$> ... <булев-израз$_n$>
- `(or` {<булев-израз>}`)`
- `(or` <булев-израз$_1$> <булев-израз$_2$> ... <булев-израз$_n$>`)`
  - Оценява последователно всички <булев-израз$_i$>
  - Ако всички се оценяват до `#f`, връща `#f`

Логически операции

- `(not` <булев-израз>`)`
  - Връща отрицанието на <булев-израз>
- `(and` {<булев-израз>}`)`
- `(and` <булев-израз$_1$> <булев-израз$_2$> ... <булев-израз$_n$>`)`
  - Оценява последователно всички <булев-израз$_i$>
  - Ако всички се оценяват до `#t`, връща `#t`
  - Ако <булев-израз$_i$> се оценява до `#f`, връща `#f` без да оценява следващите <булев-израз$_{i+1}$> ... <булев-израз$_n$>
- `(or` {<булев-израз>}`)`
- `(or` <булев-израз$_1$> <булев-израз$_2$> ... <булев-израз$_n$>`)`
  - Оценява последователно всички <булев-израз$_i$>
  - Ако всички се оценяват до `#f`, връща `#f`
  - Ако <булев-израз$_i$> се оценява до `#t`, връща `#t` без да оценява следващите <булев-израз$_{i+1}$> ... <булев-израз$_n$>

Логически операции

- `(not` <булев-израз>`)`
  - Връща отрицанието на <булев-израз>
- `(and` {<булев-израз>}`)`
- `(and` <булев-израз$_1$> <булев-израз$_2$> ... <булев-израз$_n$>`)`
  - Оценява последователно всички <булев-израз$_i$>
  - Ако всички се оценяват до `#t`, връща `#t`
  - Ако <булев-израз$_i$> се оценява до `#f`, връща `#f` без да оценява следващите <булев-израз$_{i+1}$> ... <булев-израз$_n$>
- `(or` {<булев-израз>}`)`
- `(or` <булев-израз$_1$> <булев-израз$_2$> ... <булев-израз$_n$>`)`
  - Оценява последователно всички <булев-израз$_i$>
  - Ако всички се оценяват до `#f`, връща `#f`
  - Ако <булев-израз$_i$> се оценява до `#t`, връща `#t` без да оценява следващите <булев-израз$_{i+1}$> ... <булев-израз$_n$>
- `and` и `or` са специални форми!

Примери с логически операции

```
(not x)    ⟷  (if x #f #t)
(and x y)  ⟷  (if x y #f)
(or x y)   ⟷  (if x #t y)
```

```
(define (divisible? a b)
  (= (remainder a b) 0))
```

```
(define (leap? y)
  (and (divisible? y 4)
       (or (not (divisible? y 100))
           (divisible? y 400))))
```