

# Основни понятия в Scheme

Трифон Трифонов

Функционално програмиране, 2017/18 г.

12–19 октомври 2017 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<sup>5</sup>RS

# 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

- Литерали
  - Булеви константи (`#f`, `#t`)
  - Числови константи (`15`, `2/3`, `-1.532`)
  - Знакови константи (`#\a`, `#\newline`)
  - Низови константи (`"Scheme"`, `"hi "`)
- Символи (`f`, `square`, `+`, `find-min`)
- Комбинации

$(\langle \text{израз}_1 \rangle \langle \text{израз}_2 \rangle \dots \langle \text{израз}_n \rangle)$

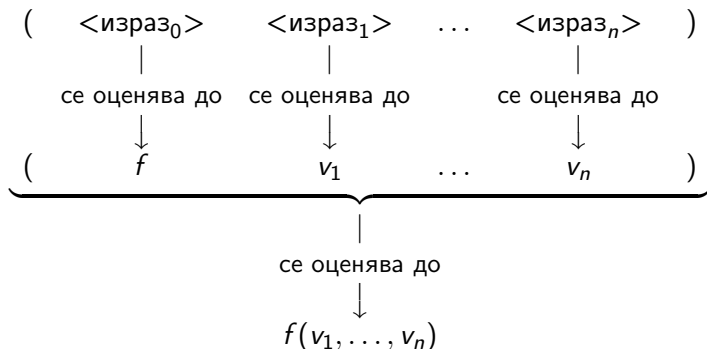
# Оценки на литерали и символи

На всеки израз се дава оценка.

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

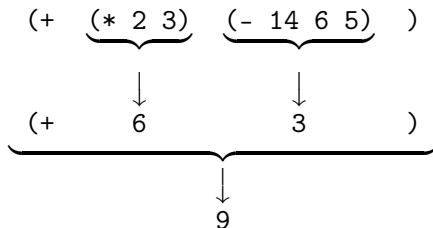
# Основно правило за оценяване

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

## Оценяване на комбинации с дефинирани функции

(f 2 4)  
↓  
(+ (square (1+ 2)) (square 4) 5)  
↓  
(+ (square (+ 2 1)) (square 4) 5)  
↓  
(+ (square 3) (square 4) 5)  
↓  
(+ (\* 3 3) (\* 4 4) 5)  
↓  
(+ 9 16 5)  
↓  
30

# 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> <израз1> <израз2>)`
- Оценява се `<условие>`
  - Ако оценката е `#t` — връща се оценката на `<израз1>`
  - Ако оценката е `#f` — връща се оценката на `<израз2>`
- `if` е специална форма!



# 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>)])`
- `(cond (<условие1> <израз1>)`  
`(<условие2> <израз2>)`  
`...`  
`(<условиеn> <изразn>)`  
`(else <изразn+1>))`
- Оценява се <условие<sub>1</sub>>, при #t се връща <израз<sub>1</sub>>, а при #f:
- Оценява се <условие<sub>2</sub>>, при #t се връща <израз<sub>2</sub>>, а при #f:
- ...
- Оценява се <условие<sub>n</sub>>, при #t се връща <израз<sub>n</sub>>, а при #f:
- Връща се <израз<sub>n+1</sub>>

## 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>))]`
- `(case <тест> ((<случай1,1> ... <случай1,k1>) <израз1>)  
 ((<случай2,1> ... <случай2,k2>) <израз2>)  
 ...  
 ((<случайn,1> ... <случайn,kn>) <изразn>)  
 (else <изразn+1>))`
- Оценява се <тест>
- при някое от <случай<sub>1,1</sub>>... <случай<sub>1,k<sub>1</sub></sub>> → <израз<sub>1</sub>>, иначе:
- при някое от <случай<sub>2,1</sub>>... <случай<sub>2,k<sub>2</sub></sub>> → <израз<sub>2</sub>>, иначе:
- ...
- при някое от <случай<sub>n,1</sub>>... <случай<sub>n,k<sub>n</sub></sub>> → <израз<sub>n</sub>>, иначе:
- Връща се <израз<sub>n+1</sub>>

## Пример с формата case

```
(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>)**
  - Връща отрицанието на <булев-израз>
- **(and {<булев-израз>})**
- **(and <булев-израз<sub>1</sub>> <булев-израз<sub>2</sub>> ... <булев-израз<sub>n</sub>>)**
  - Оценява последователно всички <булев-израз<sub>i</sub>>
  - Ако всички се оценяват до #t, връща #t
  - Ако <булев-израз<sub>i</sub>> се оценява до #f, връща #f без да оценява следващите <булев-израз<sub>i+1</sub>> ... <булев-израз<sub>n</sub>>
- **(or {<булев-израз>})**
- **(or <булев-израз<sub>1</sub>> <булев-израз<sub>2</sub>> ... <булев-израз<sub>n</sub>>)**
  - Оценява последователно всички <булев-израз<sub>i</sub>>
  - Ако всички се оценяват до #f, връща #f
  - Ако <булев-израз<sub>i</sub>> се оценява до #t, връща #t без да оценява следващите <булев-израз<sub>i+1</sub>> ... <булев-израз<sub>n</sub>>
- **and и or са специални форми!**

# Примери с логически операции

`(not x)`       $\iff$       `(if x #f #t)`

`(and x y)`     $\iff$       `(if x y #f)`

`(or x y)`      $\iff$       `(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))))
```