УПРАВЛЕНИЕ ПОТОКОМ В SCHEME: УСЛОВИЯ, ЦИКЛЫ, БЛОКИ

Выполнил Гребенкин Егор Дмитриевич
Группа 5030102/20202

Условные конструкции: if

if — минимальная развилка:

```
(if <> <-true> <-false>)
```

Замечание: в Scheme "ложь— это только #f. Всё остальное — истина

cond

cond удобен для "лесенки"условий:

```
(cond
  ((< x 0) 'negative)
  ((= x 0) 'zero)
  (else 'positive))
```

case

case — ветвление по значению (обычно по символам):

```
(case day
  ((sat sun) 'weekend)
  ((mon tue wed thu fri) 'workday)
  (else 'unknown))
```

Логические связки и короткое замыкание

and и or:

- вычисляют выражения слева направо
- останавливаются как только результат уже ясен (short-circuit)

```
(and (>= x lo) (<= x hi))
```

Блоки и последовательность: begin

begin позволяет выполнить несколько выражений по порядку и вернуть результат последнего

```
(begin
  (display "hi")
  (newline)
  'done)
```

Циклы / итерация

В Scheme "циклы"часто выражают:

- хвостовой рекурсией (особенно через именованный let)
- специальной формой do
- функциями над коллекциями (map, for-each, fold и т.п.)

Именованный let как цикл

```
(let loop ((i 0) (acc 0))
  (if (> i n)
      acc
      (loop (+ i 1) (+ acc i))))
```

do

do — "классический"цикл:

```
(do ((i n (- i 1))
     (acc 1 (* acc i)))
    ((= i 0) acc))
```

Итераторы

В Scheme обычно нет "итераторов"в стиле Python/Java как обязательной концепции ядра, но есть:

- протоколы/библиотеки в реализациях
- и, главное, высокоуровневые функции (map/fold) как идиоматичная замена

Демонстрационные программы

Условия, case, and, блоки begin
Файл: 04/src/control-demo.scm

Вывод control-demo.scm

```
sign -5: neg
sign 0: non-neg
classify -1: negative
classify 0: zero
classify 7: positive
weekday-type 'sat: weekend
weekday-type 'mon: workday
between? 5 1 10: #t
between? 0 1 10: #f
```

Вывод (продолжение)

```
line1
line2
demo-begin returns: done
```

Итерация: named let, do, map, for-each, fold

Файл: 04/src/loops-demo.scm

Вывод loops-demo.scm

```
sum-0..10: 55
fact-do 5: 120
map (lambda (x) (* x x)) xs: (1 4 9 16)
for-each prints: 1 2 3 4
foldl + 0 xs: 10
```

Как запустить в DrRacket (Racket Desktop)

1) DrRacket
2) File -> Open... -> control-demo.scm loops-demo.scm 04/src
3) Language -> Choose Language... (Scheme-)
4) Run

Альтернатива: запуск через Racket из терминала (опционально)

```
racket control-demo.scm
racket loops-demo.scm
```

Литература и материалы

- См. references.md
- Scheme reports: if/cond/case, порядок вычисления, хвостовые вызовы
- SICP: рекурсия как главный инструмент управления потоком

Условие — это выражение, а не "булевый тип"

В Scheme условие в if/cond/and/or — это выражение, результат которого проверяется на "ложность"
Ключевое правило:

- ложь только #f
- всё остальное считается "истиной"(включая 0, пустые строки, списки и т.д.)

Стиль работы с условиями

Это меняет стиль:

- "нет значения"часто кодируют как #f
- и используют and/or для короткого управления потоком

and / or — это управляющие формы

Важная деталь: and и or не обязаны возвращать #t/#f
Примеры:

```
(and #t 10) ; => 10
(or #f 42) ; => 42
```

Использование and / or

Это позволяет писать компактные конструкции:

- (and x (f x)) — вызываем f только если x не #f

Демо: 04/src/short-circuit-demo.scm

cond с => (полезный паттерн)

Во многих Scheme-диалектах cond поддерживает форму:

```
((test) => receiver)
```

Если test вернул значение v (не #f), то вычисляется (receiver v)

Когда использовать cond =>

Идеально для "распарсить/проверить и сразу обработать"

Блоки: begin vs let

begin
Используйте begin, когда нужно:

- выполнить несколько выражений последовательно
- вернуть результат последнего
- не заводить новых переменных

let

Используйте let, когда нужно:

- завести локальные имена
- не загрязнять внешнюю область видимости

```
(let ((x 10) (y 20))
  (+ x y))
```

На практике "блок"в Scheme часто равен "letлок

Итерация как выбор стиля

В Scheme есть несколько равноценных способов повторения:

- хвостовая рекурсия (идеологически "канонично")
- do (похоже на классический цикл)
- "функциональные циклы"(map, for-each, fold)

Практическая рекомендация

- если вы строите новый список → используйте map/свёртки
- если делаете сайд-эффект (печать) → for-each или хвостовую рекурсию
- если нужно много состояния цикла → именованный let или do

Демонстрационная программа №3: short-circuit

Файл: 04/src/short-circuit-demo.scm

Вывод short-circuit-demo.scm

```
(and #t 10): 10
(and #f 10): #f
(or #f 42): 42
(or 0 42): 0
maybe-name length: 5
maybe-none length: #f
```

Вывод (продолжение)

```
string->maybe-number "10": 11
string->maybe-number "nope": #f
```

do подробнее: форма

```
(do ((var1 init1 step1)
     (var2 init2 step2)
     ...)
    (test result1 result2 ...)
  body1
  body2
  ...)
```

do подробнее: порядок выполнения

- сначала вычисляются init*
- потом на каждой итерации выполняется тело
- затем применяются step*
- когда test становится истинным, цикл заканчивается и возвращаются result* (обычно одно значение)

Когда использовать do

Практика: do удобен, когда вы хотите "цикл как выражение"с явным состоянием

Выбор между cond и case

cond:

- когда условия "логические"(диапазоны, проверки, предикаты)
- когда хочется писать "лесенкой"

Выбор между cond и case (продолжение)

case:

- когда есть фиксированный набор значений (символы-теги)
- когда удобно мыслить "таблицей соответствий"

Идиома для тегов

Идиома: значения-теги обычно кодируют как символы (’ok, ’error, ’empty), а не строки