SCHEME: ИСТОРИЯ, ПРИМЕНЕНИЕ, ЗАПУСК, HELLO WORLD

Выполнил Гребенкин Егор Дмитриевич
Группа 5030102/20202

Что вы получите после этой презентации

- Понимание, что такое Scheme и чем он отличается от "просто Lisp"
- Картина "где Scheme применим"(реалистично, без мифов)
- Два способа запуска примеров: Docker и локально/онлайн (ограничения)
- Умение читать базовый синтаксис: S-выражения, вызов функций, строка/символ
- "Здравствуй, мир!"+ маленький бонус: REPL и файл

Что такое Scheme (одним абзацем)

Scheme — минималистичный диалект Lisp, созданный как язык, в котором:

- маленькое "ядро"даёт большую выразительность
- процедуры (функции) — значения "первого класса"
- лексическая область видимости и замыкания — ключевая идея
- метапрограммирование и DSL удобно выражать через макросы

Что такое Scheme (продолжение)

Scheme часто выбирают как "язык для понимания идей": интерпретация, компиляция, рекурсия, абстракции.

Короткая история: Предыстория — Lisp

- Lisp (1958) закрепил идеи: списки, символы, lambda, рекурсия, код-как-данные

Короткая история: Рождение Scheme

- 1970-е: Scheme проектировался так, чтобы просто и чисто демонстрировать:
  - лексическое связывание
  - замыкания
  - модель вычисления (окружения/фреймы, если вы читали SICP)

В Scheme есть серия "Reporttандартов:

- R5RS — классический широко цитируемый базис
- R7RS (small) — минимальный современный стандарт (возврат к минимализму)
- SRFI — "запросы на реализацию": каталог расширений (часть поддерживается в Guile)

Почему стандартизация важна

- "Scheme"— это не один бинарник, а семейство реализаций и стандартов
- Примеры могут отличаться деталями (модули, исключения, потоки) в разных реализациях

Где Scheme применяют: Образование

- курсы по программированию, абстракциям, интерпретаторам/компиляторам
- исторически важен курс/книга SICP (идеологически близко Scheme-миру)

Где Scheme применяют: DSL и метапрограммирование

Scheme хорош как "платформа для языков потому что:

- код представлен как списки
- макросы позволяют расширять синтаксис (в рамках выбранной реализации/стандарта)

Где Scheme применяют: Встраиваемость и исследования

Встраиваемость и расширяемость:

- GNU Guile часто рассматривают как "Scheme-движок который можно встраивать и расширять приложения

Исследования:

- модели вычислений, оптимизации, представления программ, экспериментальные расширения

Где Scheme НЕ применяют

- корпоративный "стандартный"backend язык — редко
- фронтенд — не про Scheme (хотя идеи влияют на JS)

Итого: Scheme — отличный учебный и исследовательский язык, а также сильная база для DSL

Базовая модель кода: S-выражения

Всё записывается как выражения в скобках:

```
(+ 2 3)
```

Это читается как: "вызвать + с аргументами 2 и 3

Почему префиксная запись — это удобно

- синтаксис единообразный для любых функций
- легко писать преобразователи и DSL
- "оператор"ничем не отличается от "обычной функции"

Важное правило истинности

В Scheme ложь — только #f
Любое другое значение (включая 0, пустую строку, пустой список — зависит от диалекта) трактуется как "истина"в if

Как запускать Scheme-код: DrRacket (Racket Desktop)

Для курса демонстрации запускаются в DrRacket (десктопная IDE для Racket/Scheme).

DrRacket: шаги запуска файла

1) DrRacket
2) File -> Open... -> hello-world.scm
3) Language -> Choose Language...
   - "" Scheme R5RS/R7RS ( )
4) Run

Ожидаемый вывод:

, !

Почему DrRacket — хороший вариант

- удобный редактор + запуск одной кнопкой
- есть интерактивная консоль (REPL) прямо внизу
- можно явно выбрать диалект/язык (R5RS/R7RS/...)

Программа "Здравствуй, мир!"

Файл: 01/src/hello-world.scm

```
;; hello-world.scm      Scheme (Racket/DrRacket)

(display ", !")
(newline)
```

Разбор построчно

- ;; ... — комментарий до конца строки
- (display "...") — печатает строку в стандартный вывод (без автоматического перевода строки)
- (newline) — печатает перевод строки

Почему именно display, а не write

Условно:

- display — "для человека"(строки без кавычек, меньше служебного)
- write — "для машинного чтения"(может печатать строки с кавычками и экранированием)

Пример сравнения (в REPL)

```
;;  REPL ():
;; DrRacket / Racket
(display "hi") ; : hi
(write "hi") ; : "hi"
```

Мини-вариант с функцией

```
(define (hello)
  (display ", !")
  (newline))

(hello)
```

Здесь define объявляет процедуру hello без аргументов

REPL: быстрый интерактивный режим

REPL (Read-Eval-Print Loop) позволяет:

- быстро пробовать выражения
- отлаживать маленькие куски
- изучать стандартные функции

Пример REPL

```
Welcome to Racket
> (+ 2 3)
5
> (display "ok") (newline)
ok
```

Практическая привычка: в REPL проверяем идею, потом переносим в файл

Частые ошибки новичка

Скобки: (f x) — вызов, f — значение
Если вы написали лишние скобки, вы пытаетесь "вызвать не-функцию"

- Кавычка ’: ’alpha — символ alpha, а (alpha) — вызов функции alpha
- if ожидает выражения: в ветках должен быть код/значение, а не "оператор"

SCHEME: План следующих 5 презентаций

Презентация 02 — Данные и переменные

- Типы данных: числа, булевы, символы, строки, пары/списки, векторы, процедуры
- Пользовательские типы: records/struct (в Racket — struct), "объекты"через замыкания
- Операции и сравнения (eq?, eqv?, equal?, числовые сравнения)

Презентация 02 — Данные и переменные (продолжение)

- Преобразования (string<->number, symbol<->string)
- Переменные, define, let/let*/letrec, лексическая область видимости
- Владение/передача владения: как концепция отсутствует (GC); обсуждаем "общие ссылки"+ мутабельность

Презентация 03 — Функции

- lambda, define процедур, variadic/rest аргументы
- "по значению/по ссылке": что реально происходит
- Возврат значений (одно значение как базовая модель)
- Рекурсия и хвостовая рекурсия (TCO)
- Замыкания и функциональные паттерны

Презентация 04 — Управление потоком

- if, cond, case, and/or
- Итерация: рекурсия, do, именованный let, map/for-each/fold
- Блоки: begin, локальные определения

Презентация 05 — Ошибки, отладка, модули, I/O

- error, обработка ошибок (в Racket — with-handlers)
- Assert-проверки и отладочный вывод (display/write)
- Модули: provide/require (в Racket), импорт/экспорт
- Ввод-вывод: порты, файлы

Презентация 06 — Продвинутые темы

- FP: композиция, HOF, дисциплина неизменяемости
- OO: class в Racket + альтернативы (message passing)
- "ZZZZZZ-ориентированное": DSL через макросы
- Макросы syntax-rules, гигиена
- VM/GC/память (обзорно)
- Конкурентность: треды (реализационно-зависимо)

Литература и материалы

- Общий список: references.md
- Рекомендуемое для старта:
  - SICP (идеология абстракций, окружения, интерпретаторы)
  - The Scheme Programming Language (Dybvig)
  - Документация Racket / DrRacket (модули, макросы, потоки, классы)