УДК 004.04
ББК 32.372
Ф82

Дэниел П. Фридман, Маттиас Феллейзен

Ф82 The Little Schemer: чудесное функциональное программирование. – М.: ДМК Пресс, 2024. – 230 с.: ил.

ISBN 978-5-93700-234-1

The Little Schemer – не просто введение в функциональное программирование, это захватывающее исследование самих принципов вычислительного мышления. Книга проводит читателя через тонкости рекурсивного программирования, представляя материал в удобном для восприятия табличном формате («вопрос–ответ»).

Это незаменимое руководство для тех, кто стремится не только уметь понимать код, но и мыслить в нем и перестраивать его, выходя за рамки обыденного программирования и погружаясь в более абстрактный мир вычислений.

@MIT Press 2023. Права на русское издание получены через агентство Александра Корженевского.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-0-26256-099-2
ISBN 978-5-93700-234-1

© Massachusetts Institute of Technology., 1996
© Оформление, перевод, издание, ДМК Пресс, 2024

Оглавление

Предисловие

> Это предисловие появлялось во втором и третьем изданиях книги «The Little LlSPer» (предшествовавшей «The Little Schemer»). Мы цитируем его здесь с разрешения автора.

В 1967 году я посещал вводные занятия по фотографии. Большинство слушателей (включая меня) записались на этот курс в надежде научиться художественной фотографии, и делать такие же восхитительные снимки, как известный художник Эдвард Уэстон (Edward Weston). В первый день преподаватель терпеливо огласил длинный список технологических навыков, которым он собирался обучить нас в течение семестра. Ключевым из них была «Зонная система» Анселя Адамса (Ansel Adams) – методика определения оптимальной экспозиции в фотографии, параметров лабораторной обработки полученного снимка (черноты на конечном отпечатке) и их зависимости от освещенности сцены. В дополнение к этому мы должны были научиться пользоваться экспонометрами для измерения уровня освещенности и определять время экспозиции и обработки в проявителе для контроля уровня черного и контраста изображения. Для этого, в свою очередь, мы должны были научиться заряжать пленку, смешивать химикаты и печатать снимки. А чтобы постоянно получать стабильные результаты – изучить правила обращения со светочувствительными материалами. На первом лабораторном занятии мы выяснили, что проявитель скользкий на ощупь, а фиксаж ужасно пахнет.

А где же творчество в композиции? Чтобы начать творить, необходимо сначала освоить инструменты для съёмки. Невозможно создать отличную фотографию без навыков, позволяющих это сделать. В инженерном деле, как и в других областях творчества, мы должны научиться анализировать, чтобы начать творить. Нельзя построить красивый и надежный мост, не имея знаний о свойствах сталей и грунтов и не владея математическим аппаратом для вычисления свойств конструкций. Точно так же нельзя построить красивую компьютерную систему, порождаемую написанными процедурами, не понимая процесса «печати снимков».

Некоторые фотографы предпочитают использовать черно-белые пластины 8×10, другие – фотопленку шириной 35 мм. И те, и другие имеют свои преимущества и недостатки. Как и в фотоделе, в про-

граммировании требуется выбрать инструменты. Многие выбирают язык Lisp за его свободный стиль и гибкость. Lisp изначально задумывался как теоретическое средство для развития теории рекурсии и символьной алгебры. В дальнейшем он превратился в уникальный, мощный и гибкий инструмент разработки программного обеспечения, позволяющий быстро создавать прототипы программных систем. Как и другие языки, Lisp предлагает обширную библиотеку готовых процедур, созданных сообществом пользователей. В Lisp процедуры являются обычными данными первого класса, которые можно передавать в виде аргументов, возвращать и хранить в структурах данных. Такая гибкость очень ценна, но самое главное – она обеспечивает механизмы формализации, именования и сохранения идиом – общих шаблонов использования, составляющих основу инженерного проектирования. Кроме того, программы на Lisp способны манипулировать представлениями программ на Lisp. Это его свойство способствовало развитию обширного комплекса инструментов синтеза и анализа программ, таких как источники перекрестных ссылок.

Книга «Little LISPer» – это уникальный подход к развитию навыков, лежащих в основе творческого программирования на Lisp. В ней просто и остроумно изложены теория и практика, необходимые для овладения навыками построения рекурсивных процессов и манипулирования рекурсивными структурами данных. Для изучающих программирование на Lisp книга «Little LISPer» может оказать такую же услугу, как упражнения Ханона (Hanon) для пальцев или фортепианные этюды Черни (Czerny) для обучающихся игре на фортепиано.

Джеральд Дж. Сассман (Gerald J. Sussman)
Кембридж, Массачусетс

Вступление

> В честь двадцатой годовщины создания Scheme мы в третий раз переработали содержимое книги «The Little LISPer», дали ей более точное название «The Little Schemer» и написали продолжение: «The Seasoned Schemer».

Программы принимают и генерируют данные. Разработка программы требует глубокого понимания данных; хорошая программа отражает форму данных, с которыми она работает. Большинство коллекций данных, а значит, и большинство программ, являются рекурсивными. Рекурсия – это способ определения объекта или решения задачи в терминах самого себя.

Цель этой книги – научить читателя мыслить рекурсивно. Наша первая задача – решить, какой язык использовать для передачи этой идеи. Есть три очевидных варианта: естественный язык, например русский, формальная математика или язык программирования. Естественные языки неоднозначны, неточны и иногда излишне многословны. Они отлично подходят для рассуждений на общие темы, но не подходят для краткого описания такой точной концепции, как рекурсия. Язык математики, в противоположность естественному языку, способен выражать мощные формальные идеи всего несколькими символами. Но, к сожалению, язык математики часто трудно понять без специальной подготовки. Сочетание технологии и математики дает нам третий, почти идеальный вариант: язык программирования. Мы считаем, что языки программирования лучше всего подходят для объяснения сути рекурсии. Они, как и язык математики, способны придавать формальное значение набору символов. Но, в отличие от языка математики, языки программирования позволяют экспериментировать – вы сможете ввести программы из этой книги в свой компьютер и понаблюдать за их поведением, изменить их и посмотреть, какой эффект оказывают эти изменения.

Пожалуй, лучшим языком программирования для изучения рекурсии является Scheme. Scheme поддерживает символьные вычисления – программисту не нужно думать о связи между символами своего языка и их представлением в компьютере. Рекурсия является естественным вычислительным механизмом в Scheme, а основной задачей программиста является создание (потенциально) рекурсивных определений. Реализации Scheme преимуществен-

но интерактивны – программист может писать программы и сразу же наблюдать за их поведением. И что особенно важно для наших уроков в конце этой книги, существует прямое соответствие между структурами программ на Scheme и данными, которыми эти программы манипулируют.

Язык Scheme можно описать достаточно формально, но для его понимания не требуется особой математической подготовки. Фактически книга «The Little Schemer» основана на конспектах лекций двухнедельного краткого введения в Scheme: для студентов без опыта программирования и с нелюбовью ко всему математическому. Многие из этих студентов готовились к карьере на государственных должностях. Мы считаем, что *написание рекурсивных программ на языке Scheme – это, по сути, простое распознавание закономерностей*. Поскольку нас интересует только рекурсивное программирование, мы ограничиваемся рассмотрением причинно-следственного устройства всего лишь нескольких функций языка Scheme: `car`, `cdr`, `cons`, `eq?`, `null?`, `zero?`, `addl`, `subl`, `number?`, `and`, `or`, `quote`, `lambda`, `define` и `cond`. По сути наш язык – это *идеализированный* язык Scheme.

Книги «The Little Schemer» и «The Seasoned Schemer» не являются введением в практику программирования, но овладение концепциями, представленными в этих книгах, дает начало к пониманию природы вычислений.

Что нужно знать, чтобы читать эту книгу

Читатель должен уметь читать, распознавать цифры и считать.

Благодарности

Мы благодарны многим людям за их помощь, которую они оказывали в процессе работы над вторым и третьим изданиями этой книги. Мы благодарим Брюса Дуба (Bruce Duba), Кента Дибвига (Kent Dybvig), Криса Хейнса (Chris Haynes), Юджина Колбекера (Eugene Kohibecker), Ричарда Салтера (Richard Salter), Джорджа Спрингера (George Springer), Митча Уанда (Mitch Wand) и Дэвида С. Уайза (David S. Wise) за многочисленные пояснения, повлиявшие на наше мышление при создании нашей книги. Гассан Аббас (Ghassan Abbas), Чарльз Бейкер (Charles Baker), Дэвид Бойер (David Boyer), Майк Данн (Mike Dunn), Терри Фалькенберг (Terry Falkenberg), Роб Фридман (Rob Friedman), Джон Гейтли (John Gateley), Майер Голдберг (Mayer Goldberg), Икбал Хан (Iqbal Khan), Джулия Лоуолл (Julia Lawall), Джон Мендельсон (Jon Mendelsohn), Джон Ниенарт (John Nienart), Джеффри Д. Перотти (Jeffrey D. Perotti), Эд Робертсон (Ed Robertson),

Энн Шпунтофф (Anne Shpuntoff), Эрих Смайт (Erich Smythe), Гай Стил (Guy Steele), Тодд Стайн (Todd Stein) и Ларри Вайссельберг (Larry Weisselberg) дали множество ценных замечаний к рукописи книги. Мы выражаем особую благодарность Бобу Филману (Bob Filman) за его бескомпромиссную критику на протяжении нескольких этапов. Наконец, мы выражаем признательность Нэнси Гарретт (Nancy Garrett), Пег Флетчер (Peg Fletcher) и Бобу Филману (Bob Filman) за вклад в дизайн и оформление в формате TEX.

В работе над четвертым и последним изданием нам очень помогла невероятно умная программа Дорай Ситарам (Dorai Sitaram) для набора текста на языке Scheme– SLATEX. Программа Chez Scheme Кента Дибвига (Kent Dybvig) сделала программирование на Scheme очень приятным занятием. Мы благодарны за критику и предложения Шеласвау Бушуэллу (Shelaswau Bushnell), Ричарду Коббу (Richard Cobbe), Дэвиду Комбсу (David Combs), Питеру Дрейку (Peter Drake), Кенту Дибвигу (Kent Dybvig), Робу Фридману (Rob Friedman), Стиву Ганцу (Steve Ganz), Крису Хейнсу (Chris Haynes), Эрику Хилсдейлу (Erik Hilsdale), Юджину Кольбекеру (Eugene Kohlbecker), Шрираму Кришнамурти (Shriram Krishnamurthi), Джулии Лоуолл (Julia Lawall), Сюзанне Мензел (Suzanne Menzel), Коллину МакКарди (Collin McCurdy), Джону Ниенарту (John Nienart), Джону Росси (Ion Rossie), Джонатану Собелу (Jonathan Sobel), Джорджу Спрингеру (George Springer), Гаю Стилу (Guy Steele), Джону Дэвиду Стоуну (John David Stone), Викраму Субраманиаму (Vikram Subramaniam), Митчу Ванду (Mitch Wand) и Мелиссе Вингард-Филлипс (Melissa Wingard-Phillips).

Примечание для читателей

Не торопитесь, читайте эту книгу вдумчиво; по всему ее тексту разбросаны ценные советы. Не стремитесь прочитать книгу в три приема. Читайте систематически. Если вы не до конца поняли одну главу, то следующую вы поймете еще меньше. Вопросы расположены в порядке возрастания сложности: вам будет трудно ответить на последующие вопросы, не зная ответов на предшествующие.

Книга организована как диалог, который мы с вами могли бы вести, обсуждая примеры программ на Scheme. Если есть возможность, поэкспериментируйте со встречающимися в процессе чтения примерами.У Scheme есть множество доступных реализаций. Несмотря на некоторые синтаксические отличия между ними (в основном это касается написания отдельных имен и пространств

имен функций), сам язык однороден. Для работы с Scheme вам потребуется определить функции `atom?`, `sub1` и `add1`:

```
(define atom?
  (lambda (x)
    (and (not (pair? x)) (not (null? x)))))
```

Чтобы узнать, правильно ли в вашей реализации Scheme определена функция `atom?`, попробуйте выполнить выражение `(atom? (quote ()))` – оно должно дать результат `#f`. На самом деле книгу можно также использовать для знакомства с современными версиями языка Lisp, такими как Common Lisp. При этом вам также придется добавить функцию `atom?`:

```
(defun atom? (x)
  (not (listp x)))
```

Вам также может понадобиться вносить изменения в программы. Обычно их совсем немного. Подсказки о том, как экспериментировать с программами, приведенные в книге, вы найдете в сносках. Сноски, начинающиеся с «S:», относятся к Scheme, а начинающиеся с «L:» – к Common Lisp.

В главе 4 мы разработаем базовую арифметику на основе трех операторов: *add1*, *sub1* и *zero?*. Поскольку в Scheme нет встроенных операторов *add1* и *sub1*, вы должны определить их, используя встроенные примитивы сложения и вычитания. А чтобы избежать цикличности, наши базовые арифметические операции сложения и вычитания должны быть записаны с использованием других символов: **+** и **–** соответственно.

В этой книге мы не даем никаких формальных определений, считая, что вы сможете сформулировать свои определения и тем самым запомнить и понять их лучше, чем если бы мы написали эти определения для вас. Внимательно читайте встречающиеся *Законы* и *Заповеди* и продолжайте чтение, только убедившись, что поняли их. Ключом к изучению Scheme является «распознавание закономерностей». *Заповеди* указывают на закономерности, которые вы уже видели. В начале книги некоторые понятия сужены для простоты, но потом они дополняются и уточняются. Также имейте в виду, что язык Scheme намного шире, чем описано в книге, и включает намного больше возможностей, чем можно было бы доходчиво описать во вводной книге. В этой книге вы получите знания, которые помогут вам прочитать и понять более продвинутые и исчерпывающие книги о Scheme.

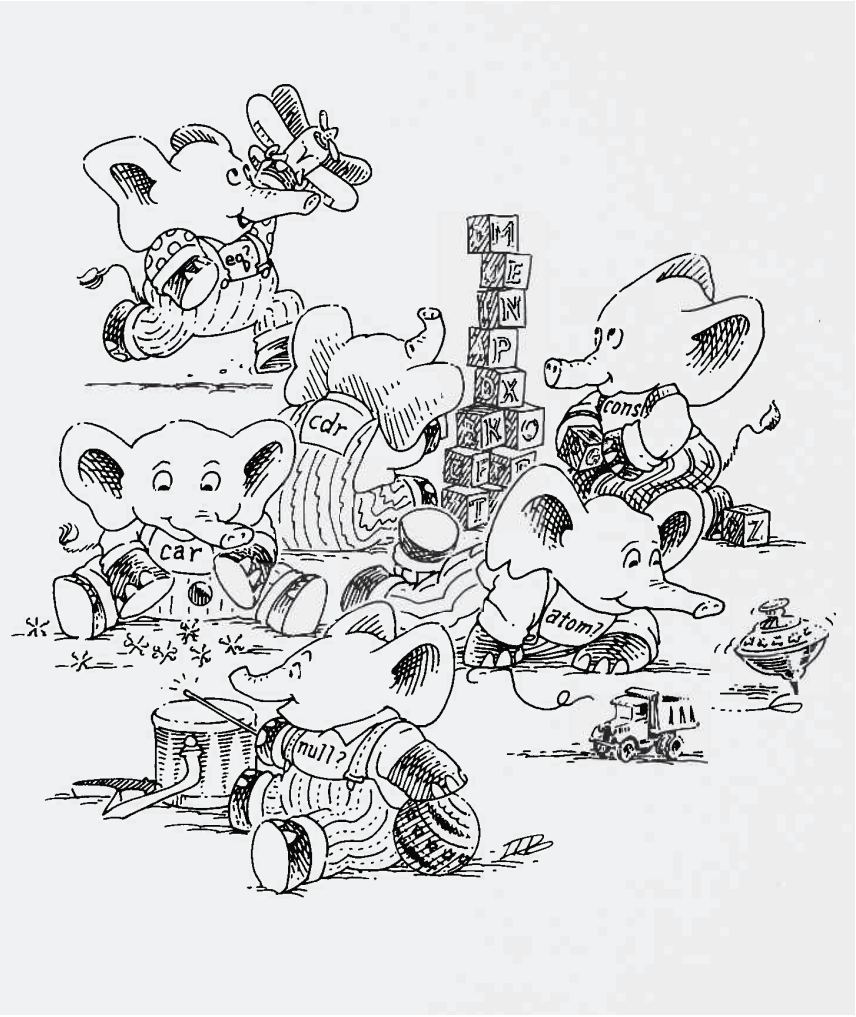
Мы используем ряд типографских соглашений для обозначения символов различных классов. Переменные и имена примитивных операций выделены *курсивом*. Простые данные, включая числа и значения истинности/ложности, набраны шрифтом без засечек. Ключевые слова, например **define**, **lambda**, **cond**, **else**, **and**, **or** и **quote**, выделены **жирным**. Когда вы будете экспериментировать с программами, то просто игнорируйте различия в оформлении, но учитывайте замечания в сносках. Согласно тем же типографским соглашениям, программы в сносках выделены моноширинным шрифтом. Оформление шрифтов можно смело игнорировать до главы 10, где мы будем рассматривать программы как данные.

Наконец, словарь Вебстера определяет термин «пунктуация» как действие по расстановке знаков препинания; в частности, действие, практика или система использования стандартных знаков в письменной и печатной формах для разделения предложений или их частей, чтобы сделать смысл более ясным. Мы восприняли это определение буквально и отказались от некоторых привычных способов использования знаков препинания, чтобы сделать смысл более ясным. В частности, мы отказались от использования знаков препинания в левой колонке во всех случаях, когда элемент, предшествующий знаку препинания, является термином языка программирования.

Во многих наших примерах для демонстрации используются названия продуктов питания. Тому есть две причины. Во-первых, еду проще вообразить, в отличие от абстрактных символов. (Эту книгу не стоит читать сидящим на диете.) Мы надеемся, что выбор продуктов поможет вам понять примеры и идеи, которые мы используем. Во-вторых, мы хотели немного развлечь вас. Мы знаем, насколько раздражающим может быть предмет, и небольшое развлечение поможет вам сохранить ясность мысли.

Теперь вы готовы приступить к работе. Удачи! Мы надеемся, что вам понравятся задания, которые ждут вас на следующих страницах.

1. ИГРУШКИ



Верно ли, что это является атомом? атом[1] [1] L, S: `(quote атом)` или `'атом`. Значения символов «L:» и «S:» описаны во вступлении.	Да, потому что атом – это строка символов, начинающаяся с буквы а.
Верно ли, что это является атомом? индейка	Да, потому что индейка – это строка символов, начинающаяся с буквы.
Верно ли, что это является атомом? 1942	Да, потому что 1492 – это строка цифр.
Верно ли, что это является атомом? u	Да, потому что u – это строка из одного символа, который является буквой.
Верно ли, что это является атомом? *abc$	Да, потому что *abc$ – это строка символов, начинающаяся с буквы или специального символа, отличного от открывающей «(» или закрывающей «)» круглой скобки.
Верно ли, что это является списком? (атом)[1] [1] L, S: `(quote атом)` или `'атом`.	Да, потому что (атом) – это атом, заключенный в круглые скобки.
Верно ли, что это является списком? (атом индейка или)	Да, потому что это – коллекция атомов, заключенная в круглые скобки.

Верно ли, что это является списком? (атом индейка) или	Нет, потому что на самом деле это два S-выражения, не заключенные в круглые скобки. Первое – список с двумя атомами, а второе – атом.
Верно ли, что это является списком? ((атом индейка) или)	Да, потому что теперь эти два S-выражения заключены в круглые скобки.
Верно ли, что это является S-выражением? xyz	Да, потому что все атомы являются S-выражениями.
Верно ли, что это является S-выражением? (x y z)	Да, потому что это список.
Верно ли, что это является S-выражением? ((x y) z)	Да, потому что все списки являются S-выражениями.
Верно ли, что это является списком? (как ваши дела на данный момент)	Да, потому что это коллекция S-выражений, заключенная в круглые скобки.
Сколько S-выражений в следующем списке? (как ваши дела на данный момент) Перечислите их.	Шесть: как, ваши, дела, на, данный, момент.
Верно ли, что это является списком? (((как) ваши) ((дела) (на данный)) момент)	Да, потому что это коллекция S-выражений, заключенная в круглые скобки.

Сколько S-выражений в следующем списке? (((как) ваши) ((дела) (на данный)) момент) Перечислите их.	Три: ((как) ваши), ((дела) (на данный)), момент.
Верно ли, что это список? ()	Да, потому что это содержит ноль S-выражений, заключенных в круглые скобки. Это особое S-выражение называется пустым списком, или null.
Верно ли, что это атом? ()	Нет, потому что `()` – это список.
Верно ли, что это список? (() () () ())	Да, потому что это коллекция S-выражений, заключенная в круглые скобки.
Что выдаст выражение *car l*, где *l* – это аргумент (a b c)?	a, потому что a – это первый атом в данном списке.
Что выдаст выражение *car l*, где *l* имеет значение ((a b c) x y z)?	(a b c), потому что (a b c) – это первое S-выражение в этом непустом списке.
Что выдаст выражение *car l*, где *l* имеет значение хот-дог?	Нет ответа. Нельзя взять *car* от атома.
Что выдаст выражение *car l*, где *l* имеет значение ()?	Нет ответа[1]. Нельзя взять *car* от пустого списка. [1] `L: nil.`

Правило Car'a

Примитив *car* определен только для непустых списков.

Что выдаст выражение *car l*, где *l* имеет значение (((хот-дог)) (маринованные огурцы) (и) соус)?	((хот-дог)) читается как «список, содержащий другой список с единственным атомом хот-дог». ((хот-дог)) – первое S-выражение в *l*.
Что выдаст выражение (*car l*), где *l* имеет значение (((хот-дог)) (маринованные огурцы) (и) соус)?	((хот-дог)), потому что (*car l*) – это просто другой способ взять «*car* от списка *l*».
Что выдаст выражение (*car* (*car l*)), где *l* имеет значение (((хот-дог)) (маринованные огурцы))?	((хот-дог)).
Что выдаст выражение *cdr l*, где *l* имеет значение (a b c)? Примечание: «cdr» произносится как «coulder» («кьюда»).	(b c), потому что (b c) – список *l* без (*car l*).
Что выдаст выражение *cdr l*, где *l* имеет значение ((a b c) x y z)?	(x y z).
Что выдаст выражение *cdr l*, где *l* имеет значение (гамбургер)?	().
Что выдаст выражение (*cdr l*), где *l* имеет значение ((x) t r)?	(t r) потому что (*cdr l*) – это всего лишь другой способ взять «*cdr* от списка *l*».

Что выдаст выражение (*cdr a*), где *a* имеет значение хот-дог?	Нет ответа. Нельзя взять *cdr* от атома.
Что выдаст выражение (*cdr l*), где *l* имеет значение ()?	Нет ответа[1]. Нельзя взять *cdr* от пустого списка. [1] L: `nil`.

Правило Cdr'а

Примитив *cdr* определен только для непустых списков. Применение *cdr* к непустому списку всегда дает другой список.

Что выдаст выражение (*car* (*cdr l*)), где *l* имеет значение ((b) (x y) ((c)))?	(x y), потому что (*cdr l*) выдаст ((x y) ((c))), соответственно, (*car* (*cdr l*)) выдаст (x y).
Что выдаст выражение (*cdr* (*cdr l*)), где *l* имеет значение ((b) (x y) ((c)))?	(((c))), потому что (*cdr l*) выдаст ((x y) ((c))), соответственно, (*cdr* (*cdr l*)) выдаст (((c))).
Что выдаст выражение (*cdr* (*car l*)), где *l* имеет значение (a (b (c)) d)?	Нет ответа, потому что (*car l*) – это атом, а *cdr* не принимает атом в аргументе; см. «Закон cdr» выше.
Что *car* принимает в виде аргумента?	Любой непустой список.
Что *cdr* принимает в виде аргумента?	Любой непустой список.

Что выдаст *cons* при применении к атому *a* и списку *l*, где *a* имеет значение арахис, а *l* – (масло и желе)? Это выражение можно записать как (*cons a l*). Читается как: «*cons* (вставить) атом *a* в список *l*».	(арахис масло и желе), потому что *cons* добавляет атом в начало заданного списка.
Что выдаст выражение *cons s l*, где *s* имеет значение (банан и), а *l* – (арахис масло и желе)?	((банан и) арахис масло и желе), потому что *cons* добавляет любое S-выражение в начало заданного списка.
Что выдаст выражение (*cons s l*), где *s* имеет значение ((помогите) мне), а *l* – (это очень ((сложная) тема для изучения))?	(((помогите) мне) это очень ((сложная) тема для изучения)).
Что *cons* принимает в виде аргументов?	*cons* принимает два аргумента: первый – любое S-выражение; второй – любой список.
Что выдаст выражение (*cons s l*), где *s* имеет значение (a b (c)), а *l* – ()?	((a b (c))), потому что () – это список.
Что выдаст выражение (*cons s l*), где *s* имеет значение a, а *l* – ()?	(a).
Что выдаст выражение (*cons s l*), где *s* имеет значение ((a b c)), а *l* – b?	Нет ответа[1], потому что второй аргумент *l* должен быть списком. [1] На практике (*cons* α β) работает со всеми значениями α и β, при этом (*car* (*cons* α β)) = α (*cdr* (*cons* α β)) = β.

Что выдаст выражение (*cons s l*), где *s* имеет значение a, а *l* – b?	Нет ответа. Почему?

Правило Cons'а

Примитив *cons* принимает два аргумента. Второй аргумент должен быть списком. Результат – всегда список.

Что выдаст выражение (*cons s* (*car l*)), где *s* имеет значение a, а *l* – ((b) c d)?	(a b) Почему?
Что выдаст выражение (*cons s* (*cdr l*)), где *s* имеет значение a, а *l* – ((b) c d)?	(a c в) Почему?
Можно ли сказать, что список *l* пустой, если *l* имеет значение ()?	Да, потому что этот список содержит нуль S-выражений. Этот вопрос можно записать так: (*null& l*).
Что выдаст выражение (*null?*[1] (**quote** ())? [1] L: `null`	Истинное значение, потому что (**quote** ())[1] – это форма записи пустого списка. [1] L: а также `()` и `'()`. S: а также `'()`.
Что выдаст выражение (*null? l*), где *l* имеет значение (a b c)?	Ложное значение, потому что *l* – непустой список.

Что выдаст выражение (*null? a*), где *a* имеет значение спагетти?	Нет ответа[1], потому что *a* – это атом, а *null?* не принимает атомы.

[1] На практике `(null? a)` выдает ложное значение для всего, что не является пустым списком.

Правило Null?

Примитив *null?* определен только для списков.

Верно ли, что *s* – это атом, если *s* имеет значение Гарри?	Да, потому что Гарри – это строка символов, начинающаяся с буквы.
Что выдаст выражение (*atom?*[1] *s*), где *s* имеет значение Гарри?	Истинное значение, потому что (*atom? s*) – это просто другой способ записать вопрос «Верно ли, что *s* – это атом?».
Что выдаст выражение (*atom? s*), где *s* имеет значение (Гарри набрал кучу яблок)?	Ложное значение, потому что *s* – это список.
Сколько аргументов принимает *atom?* и что это за аргументы?	*atom?* принимает один аргумент – любое S-выражение.
Что выдаст выражение (*atom?* (*car l*)), где *l* имеет значение (Гарри набрал кучу яблок)?	Истинное значение, потому что (*car l*) выдаст значение Гарри, а Гарри – это атом.
Что выдаст выражение (*atom?* (*cdr l*)), где *l* имеет значение (Гарри набрал кучу яблок)?	Ложное значение.

[1]
```
L: (defun  atom? (x)
     (not  listp x)))
S: (define  atom?
     (lambda (x)
    (and (not (pair? x)) (not (null? x)))))
```

Что выдаст выражение (*atom?* (*cdr l*)), где *l* имеет значение (Гарри)?	Ложное значение, потому что *l* результатом будет () – пустой список, а не атом.
Что выдаст выражение (*atom?* (*car* (*cdr l*))), где *l* имеет значение (овсянка со сладкой черешней)?	Истинное значение, потому что (*cdr l*) выдаст результат (со сладкой черешней), а (*car* (*cdr l*)) – со, т. е. атом.
Что выдаст выражение (*atom?* (*car* (*cdr l*))), где *l* имеет значение (овсянка (со сладкой) черешней)?	Ложное значение, потому что (*cdr l*) выдаст результат ((со сладкой) черешней), а (*car* (*cdr l*)) – (со сладкой), т. е. список.
Верно ли, что *a1* и *a2* – это один и тот же атом, если *a1* имеет значение Гарри и *a2* имеет значение Гарри	Да, потому что (*eq? a1 a2*) отвечает на вопрос «*a1* и *a2* – это один и тот же нечисловой атом?».
Что выдаст выражение (*eq?*[1] *a1 a2*), где *a1* имеет значение Гарри и *a2* имеет значение Гарри [1] L: eq.	Истинное значение, потому что (*eq? a1 a2*) – это просто другой способ задать вопрос «*a1* и *a2* – это один и тот же нечисловой атом?».
Что выдаст выражение (*eq? a1 a2*), где *a1* имеет значение маргарин и *a2* имеет значение масло	Ложное значение, потому что *a1* и *a2* – разные атомы.
Сколько аргументов принимает *eq?* и что это за аргументы?	*eq?* принимает два аргумента. Оба должны быть нечисловыми атомами.

Что выдаст выражение (*eq? l1 l2*), где *l1* имеет значение () и *l2* имеет значение (земляника)	Нет ответа[1], потому что () и (земляника) – это списки. [1] На практике eq? может принимать списки в аргументах. Два списка считаются одинаковыми, если являются одним и тем же списком.
Что выдаст выражение (*eq? n1 l2*), где *n1* имеет значение 6 и *n2* имеет значение 7?	Нет ответа[1], потому что 6 и 7 – это числа. [1] На практике некоторые числа могут быть аргументами eq?.

Правило Eq?

Примитив *eq?* принимает два аргумента, каждый из которых должен быть нечисловым атомом.

Что выдаст выражение (*eq?* (*car l*) *a*), где *l* имеет значение Мэри заказала маленькую отбивную из ягненка и *a* имеет значение Мэри	Истинное значение, потому что (*car l*) выдает атом Мэри и аргумент *a* тоже является атомом Мэри.
Что выдаст выражение (*eq?* (*cdr l*) *a*), где *l* имеет значение кислое молоко и *a* имеет значение молоко	Нет ответа. См. «Правило Eq?» и «Правило Cdr'a».
Что выдаст выражение (*eq?* (*car l*) (*car* (*cdr l*))), где *l* имеет значение (бобы бобы нам нужны желейные бобы)	Истинное значение, потому что сравниваются первый и второй атомы в списке.

=> А теперь сделайте себе сэндвич
с арахисовым маслом и джемом. <=