## Определение

**Лисп** (*LISt Processing language* — «язык обработки списков») — семейство языков программирования, программы и данные в которых представляются системами линейных списков символов.

Высокоуровневый мультипарадигмальный язык. Функциональный, но с ранних версий проявляет черты императивности, позволяет реализовать объектно-ориентированное программирование.

Является языком для т.н. лисп-машин — специальных вычислительных машин, архитектура которых была подстроена под Лисп.

## История

Лисп — второй по старшинству до сих пор используемых высокоуровневых ЯП (первый — Фортран). Его создатель — Джон Маккарти (MIT, позже Стэнфорд), автор первого транслятора — Стив Рассел.

Забавный факт: Формализм, использованный Маккарти, известен как “Лисп на Лиспе” — это написание транслятора языка в машинный код на нем самом — и используется достаточно часто.

В 1960—1980-е гг шла разработка первых диалектов Лиспа компаниями и университетами:

* MacLisp для MAC,
* Interlispдля BBN, Scientific Data Systems и Xerox
* Scheme в MIT

В итоге к середине 1980-х параллельно существовали и развивались более десятка крупных диалектов Лиспа, общее же число несовместимых между собой реализаций было существенно больше. Усугубиться этой проблеме не дала стандартизация языка Common Lisp.

Появление Common Lisp затормозило создание новых диалектов языка. Но в стремлении дать Лиспу возможность взаимодействовать с другими ЯП (Python, Java, C++) было создано и несколько диалектов Лиспа, ориентированных на исполнение в динамических средах других языков. Эти диалекты получают возможность напрямую работать с библиотеками соответствующей языковой среды и взаимодействовать с программами на других языках, исполняемыми в той же среде.

В 1990 и 1995 были проведены дополнительные стандартизации Common Lisp.

## Применение

Создавался для работы по искусственному интеллекту (как представление математической нотации для символьных вычислений) и до сих пор остаётся одним из основных инструментальных средств в этой области.   
Это наиболее частая область его применения, но популярным его назвать сложно. Его любят за гибкость, эффективность и скорость. Проблемы составляют низкая читаемость кода и сложность в целом.

Применяется он и как средство обычного промышленного программирования, от встроенных скриптов до веб-приложений массового использования, но популярным не считается.  
Имеет 26 диалектов, наиболее известные из которых на сегодня: Common Lisp, Clojure, Scheme и Racket (все, кроме первого, иногда называются отдельными ЯП. В целом, это так — между собой они несовместимы и развивались параллельно и независимо).

На этих диалектах, например, написаны такие известные приложения и системы как:

* текстовый редактор семейства GNU Emacs (диалект Emacs Lisp)
* приложение и браузерное расширение для редактирования английских текстов Grammarly (диалект Common Lisp)
* Apache Storm — распределенный инструмент обработки больших объёмов данных (диалект Clojure и Java)
* Сервер и пакет программ для Boeing и Airbus (диалект Common Lisp)
* ПО Лондонского метро (диалект Common Lisp)
* Продукты компании Atlassian (менеджмент командой работы) (диалект Clojure)
* Французская информационная администрация (диалект Clojure)

## Сравнение: общие соображения

Здесь — очевидные характеристики.

Фундаментальное отличие языков — парадигма:

* Haskell —образцовый чисто функциональный язык,
* Лисп — исходно функциональный, но поддерживает практически все парадигмы и очень адаптивен. Следовательно, расширяемый.

Основа архитектур и Лиспа, и Haskell — лямбда-исчисление. По подходу к характеру функций как языковых объектов разные диалекты Лиспа различаются. Наиболее близки к Haskell в этом контексте диалекты группы Scheme, где функцию можно:

* передавать в функцию в качестве аргумента
* возвращать как результат работы функции
* присваивать переменным

Common Lisp накладывает существенные ограничения на подобные операции.

По типизации все, в целом, ясно. После Python было сложно привыкать к неспособности С делить целые числа с дробным результатом, после С — к неспособности Haskell складывать Bool c Int. В Lisp на слабом уровне присутствует автоматическое приведение типов в выражениях и отсутствует необходимость объявлять типы везде.

Оба языка поддерживают два типа запуска программ: с помощью компилятора и интерпретатора. Для Haskell перечислены основные компилятор и интерпретатор — на самом деле, их больше. Glasgow Haskell Compiler написан на С и Haskell, HUGS — на С.

В частности, Yale Haskell — одна из сред разработки Haskell, поддерживающая компиляцию и интерпретацию — реализован на Лиспе. В VSCode реализован подход компилятора.

## Сравнение: императивный Лисп

B синтаксис Лиспа исходно заложена возможность описания алгоритмов путём перечисления последовательности требуемых действий. Это т.н. «неявный PROGN» в структуре лямбда-вызовов Лиспа: в том месте, где должна размещаться команда, составляющая основу лямбда-выражения, может быть записано не одна, а несколько команд, и результатом лямбда-вызова станет результат последней из них. С соответствующими, возможно, побочными эффектами.

Конструкция progn принимает в качестве аргументов несколько форм и последовательно, слева направо, оценивает их значения. Результаты всех вычислений, кроме последнего, отбрасываются, а значение последней вычисленной формы становится значением всей конструкции progn.

Помимо этого progn можно задать в явном виде — тогда исполнятся все переданные в лямбда-функцию аргументы, но в качестве результата вернется значение одного с заданным номером.

В Лиспе есть возможность задавать переменные и осуществлять присваивания. Эти переменные, конечно, можно использовать в дальнейших последовательных операциях.

Типов циклов в Лисп принципиально больше, чем стандартные for и while, но по сути они аналогичны. Рекурсия, заменяющая циклы для Haskell, в Лиспе, разумеется, тоже поддерживается.

### Примеры:

* цикл do: аргументы, условие, тело цикла
* различие циклов loop и loop for, объявление переменной, присваивание, четкая инструкция по действиям при выполнении определенного условия

Таким образом, Лисп поддерживает последовательное исполнение операций.

## Сравнение: макропрограммирование в Лисп

Макропрограммирования позволяет программе верхнего уровня сформировать исполняемый код, который будет впоследствии исполнен, т.е. выступить в качестве макрогенератора. В Лиспе его можно реализовать через:

* непосредственный вызов интерпретатора специальной функцией — громоздко
* использование макросов.

Лисп известен благодаря своей развитой системе макросов.

Макросы задаются ключевым словом defmacro и имеют аналогичный функциям синтаксис, но на этом сходства макросов и функций фактически заканчиваются.

* Каждый вызов макроса раскрывает его код при трансляции так, будто этот код написан вместо вызова макроса. Это позволяет компилятору читать функцию “сложить” в примере как валидную и обрабатывать ее соответственно.
* Ошибки в макросах выдаются еще на этапе трансляции, а не при выполнении программы.
* Аргументы макросов по умолчанию не вычисляются.

Дополнительный важный момент здесь — наличие макросов чтения, непосредственно преобразующих **язык** программы. Они позволяют включать в Лисп новые конструкции и дополнять синтаксис, потому используются в реализации на Лиспе исходно не поддерживаемых в нём методов и средств программирования. Макросы чтения — одно из главных средств расширения языка.

## Сравнение: объектно-ориентированный Лисп

ООП в Лиспе исходно не предусматривалось и было добавлено только когда стало достаточно популярным. Особенности Лиспа упростили такое внедрение:

* Свойства (слоты) у символов ⇒ изначальная поддержка инкапсуляции.
* Функциональные свойства Лиспа (поддержка функционалов, присваивание функций переменным и сохранение их в свойствах символов) ⇒ связь кода (методов) с данными (объектами).
* Динамический характер языка в сочетании с вышеперечисленными особенностями ⇒ полиморфное поведение кода и данных в программе.

Единственный компонент ООП-системы, которого нет в базовом Лиспе — наследование, но оно было реализовано без особых затруднений. В частности, такое расширение обусловили рассмотренные ранее макросы.

Таким образом, Лисп содержит в себе все элементы, на которых базируется технология ООП, и реализация её поддержки в языке сводится к созданию соответствующих синтаксических элементов. Благодаря развитому механизму макросов, они могут быть добавлены средствами самого языка, без необходимости расширения базового синтаксиса и модификации трансляторов.

## Сравнение: user experience

Хаскель признается объективно более сложным в изучении, чем Лисп. Тому несколько причин:

1. Строгость языка. Будучи чистым функциональным языком, Haskell требует от разработчика знаний особенностей парадигмы, которые сложно воспринимаются привычными к императивной парадигме новичками. Обладающий императивными чертами Лисп воспринимается менее инородным.
2. Для Haskell сложны процессы настройки среды программирования, установки пакетов, запуска дебаггера. В случае Лиспа основная сложность возникает на этапе выбора версии — остальное решается с меньшими по сравнению с Haskell усилиями.

Вместе с тем, Haskell признается более “красивым”, чем Лисп, в т.ч. ввиду изобилия скобок в последнем. Цитата из Википедии: “Внешне программа на Лиспе выглядит как гигантское нагромождение скобок.”

Как бы сложен ни казался Haskell, именно его стабильно рекомендуют учить первым среди функциональных языков: распространено мнение, что Haskell наилучшим образом демонстрирует принципы и качества функционального программирования и учит новичков “правильному стилю и образу мышления”. Также у Haskell сильное и сплоченное сообщество, размерами сильно превосходящее сообщество Лиспа.

Отзывы относительно процесса настройки IDE и установки и использования дебаггера примерно идентичны — это сложно и документация не покрывает все возникающие у пользователей проблемы. В т.ч. в этом контексте чувствуется разница между сообществами обоих языков: для Haskell, как правило, источников больше и они релевантны.