
Awesome Lisp Languages

A list of lisp-flavored programming languages implemented on top of existing programming languages.

Why should I care as a lisp programmer?

If you already love s-expressions then lisp-flavored languages will make it nicer when you need to build on existing platforms. In case the target language does not support advanced features like macros and REPL-driven development, these can often be easily added by using the s-expressions layer.

The second point is about helping to spread lisp and its powerful ideas more. The example of Clojure and its relative popularity shows that being hosted on existing mainstream language and leveraging ecosystems of existing libraries is a key to broader adoption. It also lowers the barrier for people to try lisp and learn about the ideas behind it. Traditionally to learn lisp one needs to learn and get used to very unfamiliar syntax while at the same time being exposed to a completely new environment and ecosystem. Taking the environment out of the equation can make the experience of trying out lisp more approachable.

Why should I care as a programmer in other language?

Learning about Lisp will make you a better programmer. You can pick any language below based on the language you are familiar with to get you started with the lisp syntax more easily. It is also worth to read a post to get intuition for lisp syntax.

In general when one learns any new programming language it opens new horizons and improves programming insight. Modern programming languages are converging and sometimes are being very similar to each other. The similarities can be missed because they are hidden behind a specific syntax.

If we translate the languages to a common syntax the similarities are more apparent and the different concepts stand out more. That way we can focus on the new innovative concepts and ideas to broaden our horizons.

Classification

- **Type-A:** Simple syntax mapping

These languages usually just provide s-expressions (parentheses) syntax and are translated to the

target language without extra features/semantics. Also sometimes being called transpilers.

- **Type-B:** Syntax and additional semantics

In addition to translating the syntax some additional features/semantics that are not present in the target language are added. Usually if a language does not fit in other category, it can be considered being a Type-B.

- **Type-C:** Clojure-like

Distinctive syntax that besides parentheses also uses brackets and curly braces. Distinctive features are persistent data structures, namespaces and vars, protocols.

- **Type-L:** Common Lisp

Implementing ANSI Common Lisp standard or being inspired by it.

- **Type-S:** Scheme

Implementing some of RxRS standards or being inspired by Scheme.

Languages

Listed primarily by the language which can be used for interoperability / FFI.

Language section does not necessarily mean the language of the implementation. For example [Ferret](#) compiles into [C++](#) but the compiler is written in [Clojure](#). Or [Carp](#) interops with [C](#) but it is mostly written in [Haskell](#). In case of [SBCL](#) it contains only small amounts of [C](#), but it is implemented almost entirely in [Common Lisp](#).

- Multi Lang
- Common Lisp
- Scheme
- C/C++
- C#
- Erlang
- Fortran
- Go
- Java
- JavaScript
- Julia
- Lua
- Objective-C
- OCaml
- PHP

-
- Python
 - R
 - Rust
 - Shell
 - VHDL
 - WASM

Multi Lang

- Bigloo [Type-S] compiles into native binaries, interop with C, JVM, .NET
- Lux [Type-B] functional, statically-typed Lisp that will run on several platforms
- Mal is an educational lisp with implementations in dozens of languages. It is a great resource for learning about lisp implementation.
- Ribbit [Type-S] small and portable Scheme implementation (R4RS, 4 KB footprint), AOT and incremental compilers, targets C, JavaScript, Python and Scheme
- STELLA - strongly typed, object-oriented, compiles down to Common Lisp, C++, or Java
- Shen [Type-B] implementations in many programming languages, builtin pattern-matching and logic programming, optional static typing and lazy evaluation
- Wax [Type-A] tiny programming language, strongly statically typed, manual memory management, transpiles to C, C++, Java, TypeScript, Python, C#, Swift, Lua and WebAssembly
- Zick Standard Lisp minimal lisp with 42 implementations

Common Lisp

- SBCL [Type-L] high performance native code compiler, native threading support, type inference engine
- CLISP [Type-L] uses bytecode compiler, easily portable
- Clozure CL [Type-L] fast compilation speed, native threads, precise generational compacting garbage collector, convenient foreign-function interface
- Clasp [Type-L] compiled using LLVM, seamless integration with existing libraries
- ECL [Type-L] embeddable and portable, can build standalone executables
- Coalton [Type-L] efficient, statically typed functional programming language that supercharges Common Lisp
- See list of additional implementations.

Scheme

- Chez Scheme [Type-S] compiles to native binaries, among the fastest available Scheme implementations, R6RS
- Chicken Scheme [Type-S] produces portable and efficient C, supports R5RS and R7RS (work in progress)
- Guile [Type-S] embeddable, useful for extending programs with scripting
- Racket [Type-S] large standard library, powerful macro system, includes DrRacket IDE
- Cyclone [Type-S] Scheme-to-C compiler, R7RS, native threading support, generates fast native binaries
- Microscheme [Type-S] Scheme subset for microcontrollers (like Arduino boards)
- Loko Scheme [Type-S] runs on bare hardware
- See list of additional implementations and benchmarks.

C/C++

- C-Mera [Type-A] also includes extensions to generate code to run on CUDA, GLSL
- Cakelisp [Type-A] performance-oriented, good for game development, compiles down to C/C++, macros and compile-time code modification
- Carp [Type-B] statically typed, no GC (Rust-like borrow checking)
- Dale [Type-B] Lisp-flavoured C with additional features, no GC, LLVM backend
- Extempore [Type-S] designed for live coding and music performances, temporal scheduling based on audio card sample rate
- FemtoLisp [Type-S] scheme-like lisp, powers the compiler of the Julia language
- Ferret [Type-C] aimed towards embedded systems
- Janet [Type-B] embeddable, large standard library, GC
- Jank [Type-C] LLVM-hosted, Clojure-compatible, type-analysis, JIT
- jo_clojure [Type-C] Fast Embeddable Clojure in C/C++, including persistent datastructures and STM
- Lcc [Type-A] Lisp-like syntax for writing C
- Liz [Type-A] written as EDN, compiles to Zig, customizable memory allocators, native binaries for many architectures
- Maru [Type-B] minimal self-hosting lisp, multimethods, user-defined types and structures, GC
- PicoLisp [Type-B] compiled to bytecode and interpreted, C and Java interop, built-in database and GUI
- Owl Lisp [Type-S] dialect of the Scheme, code can be interpreted or compiled into C files
- Toccata [Type-C] Clojure-inspired, gradually typed, no nil values, reference counting, compiles into native binaries

C

- Clojure CLR [Type-C] great for game development with arcadia and unity
- RainLisp [Type-B] inspired by Scheme, interpreted, can be used as DSL integrating with .NET

Erlang

- Clojerl [Type-C]
- Lisp Flavored Erlang [Type-A]

Fortran

- fscheme [Type-S] small scheme interpreter written in Fortran 95
- Schemetran [Type-A] Expressing Fortran computations in Scheme, compiles to readable Fortran code

Go

- Joker [Type-C] interpreter, linter, great for scripting, Go interop is very limited
- Slick [Type-L] Lisp/Scheme-style s-expression surface syntax for the Go programming language
- Zygo [Type-B] embedable, call into native Go using reflection, optional infix syntax
- ZYLISP [Type-A] simple Lisp that compiles to Go (source or bytecode)

Java

- ABCL [Type-L] CL interpreter and compiler, embedable using Java scripting API (JSR-223)
- Armed Bear Clojure [Type-C+L] Common Lisp embedded in Clojure via ABCL
- Clojure [Type-C]
- Kawa [Type-S] scheme implementation (R7RS)
- PicoLisp [Type-B] compiled to bytecode and interpreted, C and Java interop, built-in database and GUI
- Venice [Type-C] Clojure-inspired, sandboxed, Java interop, 800+ builtin functions

JavaScript

- BiwaScheme [Type-S] compact Scheme written in JavaScript, integrates well with web browsers and Node

-
- ClojureScript [Type-C]
 - eslisp [Type-A] S-expression syntax for ECMAScript/JavaScript, Lisp-like macros
 - JACL [Type-L] extended subset of Common Lisp, async reader and REPL development workflow
 - JSLisp (source) [Type-L] Lisp-2, similar to Common Lisp, includes GUI library and IDE
 - LIPS [Type-S] similar to BiwaScheme, has better notation to call JS functions
 - Lumen [Type-A] self-hosted Lisp for Lua and JavaScript, uses arrays as first-class datastructures
 - Parenscript [Type-L] Common Lisp to JavaScript translator, native JS types, native calling convention
 - RacketScript [Type-S] Racket to JavaScript compiler, interop with both Racket and JS ecosystem
 - Valtan [Type-L] Common Lisp to JavaScript compiler
 - Whalesong [Type-S] Racket to JavaScript compiler
 - Wisp [Type-C] Clojure-like, has protocols, no persistent data structures

Julia

- LispSyntax.jl [Type-A] Clojure-like lisp syntax to Julia translator with convenience macros, uses Julia's compiler and JIT

Lua

- Fennel [Type-A] full Lua compatibility, embedable, compiled code with no runtime dependency
- Lumen [Type-A] self-hosted Lisp for Lua and JavaScript, uses arrays as first-class datastructures
- Urn [?] focus on minimalism, should work with LuaJIT, influenced by Common Lisp and Clojure

Objective-C

- DreamLisp [Type-B] Clojure-inspired, originally based on MAL, added modules, lazy collections
- nu [?] interpreted

OCaml

- Reason-Lisp [Type-A] very incomplete

PHP

- Phel [Type-C] Phel is a functional programming language that compiles to PHP.

Python

- Hy [Type-A] compiles to Python AST, use Python ML libraries, runs on PyPy
- Hissp [Type-A] compiles to a functional subset of Python, macro metaprogramming with Python ecosystem
- Pixie [Type-B] Clojure inspired, written in RPython, custom GC and JIT
- Basilisp [Type-C] Clojure-compatible, targeting Python3.6+

R

- Ilr [Type-C] Clojure inspired, in R compiles and interops with R

Rust

- BLisp [Type-B] statically typed scripting language, type inference, algebraic data types, generics
- GameLisp [Type-B] scripting language for Rust game development, interpreted, pattern-matching, coroutines, macros
- Ketos [Type-B] scripting and extension language for Rust programs, compiled to bytecode
- Rustly [Type-C] transpiler, only small subset of Clojure supported
- Steel [Type-S] embedded scheme interpreter in Rust, inspired by Racket

Shell

- Gherkin [Type-B] (dormant) implemented in Bash, shell interop
- Fleck [Type-A] Clojure-like, based on Mal, packaged as single-file Bash script

VHDL

- Vhdl Lisp - alternative s-expression based notation to describe programmable integrated circuits (FPGAs)

WASM

- Arboreta WASM [?] Common Lisp tooling for WebAssembly
- clj-wasm [Type-A] Clojure-flavored WASM's text format
- Hoot [Type-S] ahead-of-time compiler for R7RS-small Scheme, aiming to support all of Guuile
- Liz [Type-A] general purpose programming language, supports WASM compilation target

-
- Schism [Type-S] self-hosting compiler from a subset of R6RS Scheme to WebAssembly
 - WebAssembly Scheme [Type-S] partial implementation of R7RS scheme, written using WebAssembly Text format

Misc

- Bel - self-hosted lisp dialect, see also markdown formatted mirror
 - Bel Clojure - implementation in Clojure, includes continuations, Java numbers and strings, read blog post
 - Language::Bel - implementation of Bel in Perl 5, includes extensive test suite
 - Chime - implementation of Bel written in Haskell
 - Babybel - Ruby implementation of Bel
 - Bel-sml - implementation written in Standard ML
- uLisp - Lisp for microcontrollers, fits into 2 Kbytes of RAM
- CLJSL - subset of Clojure compiled to GLSL for GPU programming
- A list of more Clojure-like languages.
- Additional “write C in Lisp” projects (most of them not ready for a prime time).
- Build your own lisp - a book describing building a Lisp dialect
- See also list of languages implemented in Lisp.
- Map of Common Lisp implementations
- Benchmarks of Scheme implementations

Contribute

Anything incorrect? Is there an interested project that is missing? Open an issue or PR to request adding a project to the list.