Racket (programming language)

From Wikipedia, the free encyclopedia

Racket is a general-purpose, multi-paradigm programming language and a multi-platform distribution that includes the Racket language, compiler, large standard library, IDE, development tools, and a set of additional languages including Typed Racket (a sister language of Racket with a static type-checker), Swindle, FrTime, Lazy Racket, R5RS & R6RS Scheme, Scribble, Datalog, Racklog, Algol 60 and several teaching languages.

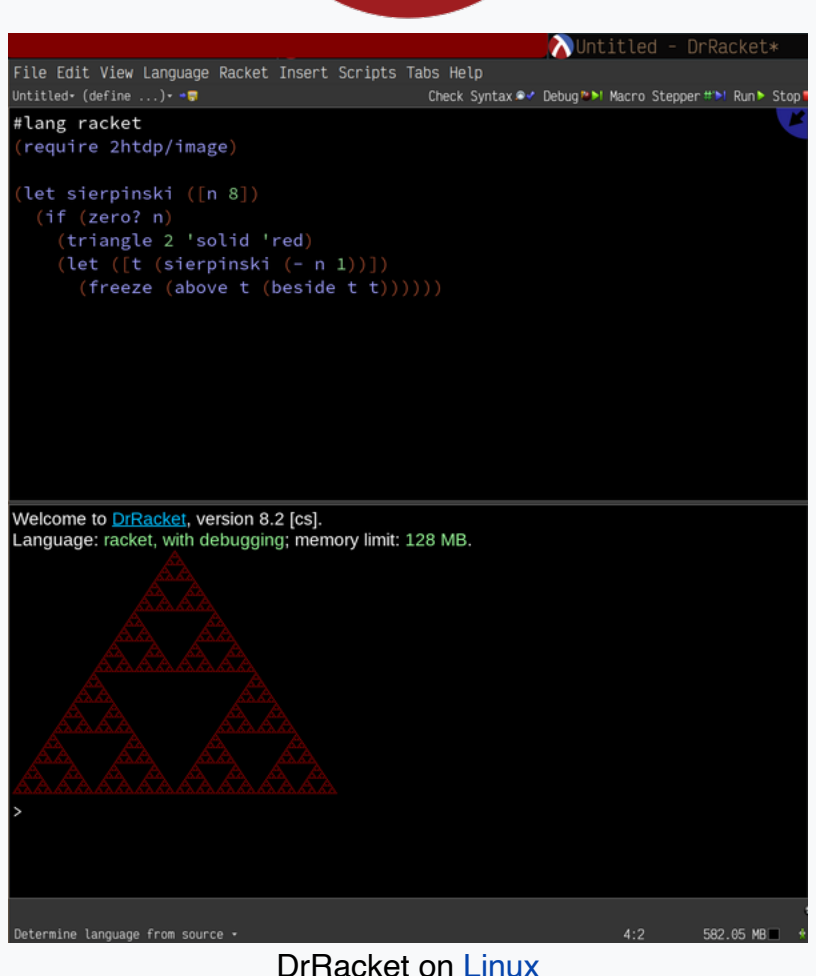
The Racket language is a modern dialect of Lisp and a descendant of Scheme. It is designed as a platform for programming language design and implementation.[9] In addition to the core Racket language, *Racket* is also used to refer to the family of programming languages[10] and set of tools supporting development on and with Racket.[11] Racket is also used for scripting, computer science education, and research.

The Racket platform provides an implementation of the Racket language (including a runtime system,[12] libraries, and compiler supporting several compilation modes: machine code, machine-independent, interpreted, and JIT) along with the DrRacket integrated development environment (IDE) written in Racket.[13] Racket is used by the ProgramByDesign outreach program, which aims to turn computer science into "an indispensable part of the liberal arts curriculum".[14][15]

The core Racket language is known for its extensive macro system which enables creating embedded and domain-specific languages, language constructs such as classes or modules, and separate dialects of Racket with different semantics.[16][17][18][19]

The platform distribution is free and open-source software distributed under the Apache 2.0 and MIT licenses.[20] Extensions and packages written by the community may be uploaded to Racket's package catalog.

Racket



DrRacket on Linux

Paradigm	Multi-paradigm: functional, imperative, logic, meta, modular, object-oriented, reflective
Family	Lisp
Designed by	PLT Inc.
Developer	PLT Inc.
First appeared	1995; 27 years ago
Stable release	8.6[1] / 10 August 2022; 22 days ago
Typing discipline	Dynamic, static, strong
Platform	x86, PowerPC, SPARC, MIPS, ARM
OS	Cross-platform
License	MIT or Apache 2.0[2]
Filename extensions	.rkt[3]
Website	racket-lang.org
Dialects	
Typed Racket, FrTime, Lazy Racket, Scribble	
Influenced by	
Eiffel,[4] Scheme	
Influenced	
Clojure,[5] Rust,[6][7] Scheme[8]	

Contents [hide]

History [edit]

Development [edit]

Matthias Felleisen founded PLT Inc. in the mid 1990s, first as a research group, soon after as a project dedicated to producing pedagogic materials for novice programmers (lectures, exercises/projects, software). In January 1995, the group decided to develop a pedagogic programming environment based on Scheme. Matthew Flatt cobbled together MrEd, the original virtual machine for Racket, from libscheme,[21] wxWidgets, and a few other free systems.[22] In the years that followed, a team including Flatt, Robby Findler, Shriram Krishnamurthi, Cormac Flanagan, and many others produced DrScheme, a programming environment for novice Scheme programmers and a research environment for soft typing.[13] The main development language that DrScheme supported was named PLT Scheme.

In parallel, the team began conducting workshops for high school teachers, training them in program design and functional programming. Field tests with these teachers and their students provided essential clues for directing the development.

Over the following years, PLT added teaching languages, an algebraic stepper,[23] a transparent read–eval–print loop, a constructor-based printer, and many other innovations to DrScheme, producing an application-quality pedagogic program development environment. By 2001, the core team (Felleisen, Findler, Flatt, Krishnamurthi) had also written and published their first textbook, *How to Design Programs*, based on their teaching philosophy.

The Racket Manifesto[9] details the principles driving the development of Racket, presents the evaluation framework behind the design process, and details opportunities for future improvements.

Version history [edit]

The first generation of PLT Scheme revisions introduced features for programming in the large with both modules and classes. Version 42 introduced units – a first-class module system – to complement classes for large scale development.[24] The class system gained features (e.g. Java-style interfaces) and also lost several features (e.g. multiple inheritance) throughout these versions.[16] The language evolved throughout a number of successive versions, and gaining milestone popularity in Version 53, leading to extensive work and the following Version 100, which would be equivalent to a "1.0" release in current popular version systems.

The next major revision was named Version 200, which introduced a new default module system that cooperates with macros.[24] In particular, the module system ensures that run-time and compile-time computation are separated to support a "tower of languages".[25] Unlike units, these modules are not first-class objects.

Version 300 introduced Unicode support, foreign library support, and refinements to the class system.[24] Later on, the 300 series improved the performance of the language runtime with an addition of a JIT compiler and a switch to a default generational garbage collection.

By the next major release, the project had switched to a more conventional sequence-based version numbering. Version 4.0 introduced the `#lang` shorthand to specify the language that a module is written in. Further, the revision introduced immutable pairs and lists, support for fine-grained parallelism, and a statically-typed dialect.[26]

On 7 June 2010, PLT Scheme was renamed Racket.[27] The renaming coincided with the release of Version 5.0. Subsequently, the graphical user interface (GUI) backend was rewritten in Racket from C++ in Version 5.1 using native UI toolkits on all platforms.[22] Version 5.2 included a background syntax checking tool, a new plotting library, a database library, and a new extended REPL.[28] Version 5.3 included a new submodule feature for optionally loaded modules,[29] new optimization tools, a JSON library, and other features.[30] Version 5.3.1 introduced major improvements to DrRacket: the background syntax checker was turned on by default and a new documentation preview tool was added.[31]

In version 6.0, Racket released its second-generation package management system. As part of this development, the principal DrRacket and Racket repository was reorganized and split into a large set of small packages, making it possible to install a *minimal racket* and to install only those packages needed.[32]

Version 7 of Racket was released with a new macro expander written in Racket as part the preparations for supporting moving to the Chez Scheme runtime system and supporting multiple runtime systems.[33] [34] On 19 November 2019, Racket 7.5 was released. The license of Racket 7.5 was less restrictive. They use now either the Apache 2.0 license or the MIT license.[35][36]

On 2021 February 13, Racket 8.0 was released. Racket 8.0 marks the first release where Racket with the Chez Scheme runtime system, known as Racket CS, is the default implementation. Racket CS is faster, easier to maintain and develop, backward-compatible with existing Racket programs, and has better parallel garbage collection.[37]

Features [edit]

Main article: Racket features

Racket's core language includes macros, modules, lexical closures, tail calls, delimited continuations,[38] parameters (fluid variables), software contracts,[39] green threads and OS threads,[40][41][42] and more. The language also comes with primitives, such as eventspaces and custodians, which control resource management and enables the language to act like an operating system for loading and managing other programs.[12] Further extensions to the language are created with the powerful macro system, which together with the module system and custom parsers can control all aspects of a language.[43] Most language constructs in Racket are implemented as macros in the base language. These include a mixin class system,[16] a component (or module) system as expressive as opaque ascription in the ML module system,[17] and pattern matching.

Further, the language features the first contract system for a higher-order programming language.[44] Racket's contract system is inspired by the Design by Contract work for Eiffel and extends it to work for higher-order values such as first-class functions, objects, reference cells, and so on. For example, an object that is checked by a contract can be ensured to make contract checks when its methods are eventually invoked.

Racket includes both bytecode and JIT (JIT) compilers. The bytecode compiler produces an internal bytecode format run by the Racket virtual machine, and the JIT compiler translates bytecode to machine code at runtime.

Since 2004, the language has also shipped with PLaneT, a package manager that is integrated into the module system so that third-party libraries can be transparently imported and used. Also, PLaneT has a built-in versioning policy to prevent dependency hell.[45]

At the end of 2014, much of Racket's code was moved into a new packaging system separate from the main code base. This new packaging system is serviced by a client program named *raco*. The new package system provides fewer features than PLaneT; a blog post by Jay McCarthy on the Racket blog explains the rationale for the change and how to duplicate the older system.[46]

Integrated language extensibility and macros [edit]

See also: Racket language extensions

The features that most clearly distinguish Racket from other languages in the Lisp family are its integrated language extensibility features that support building new domain-specific and general-purpose languages. Racket's extensibility features are built into the module system to allow context-sensitive and module-level control over syntax.[18] For example, the `#%app` syntactic form can be overridden to change the semantics of function application. Similarly, the `#%module-begin` form allows arbitrary static analysis of the entire module.[18] Since any module can be used as a language, via the `#lang` notation, this effectively means that virtually any aspect of the language can be programmed and controlled.

The module-level extensibility features are combined with a Scheme-like hygienic macro system, which provides more features than Lisp's s-expression manipulation system,[47][48] Scheme 84's hygienic extend-syntax macros, or R5RS's syntax-rules. Indeed, it is fair to say that the macro system is a carefully tuned application programming interface (API) for compiler extensions. Using this compiler API, programmers can add features and entire domain-specific languages in a manner that makes them completely indistinguishable from built-in language constructs.

The *macro* system in Racket has been used to construct entire language dialects. This includes Typed Racket, which is a gradually typed dialect of Racket that eases the migration from untyped to typed code,[49] Lazy Racket—a dialect with lazy evaluation,[50] and Hackett, which combines Haskell and Racket.[51] The pedagogical programming language Pyret was originally implemented in Racket.[52][53]

Other dialects include FrTime (functional reactive programming), Scribble (documentation language),[54] Slideshow (presentation language),[55] and several languages for education.[56][57]

Racket's core distribution provides libraries to aid the development of programming languages.[18] Such languages are not restricted to s-expression based syntax. In addition to conventional readtable-based syntax extensions, the directive `#lang` enables the invocation of arbitrary parsers, which can be implemented using the parser tools library.[58] See Racket logic programming for an example of such a language.

Programming environment [edit]

The language platform provides a self-hosted IDE[13] named DrRacket, a continuation-based web server,[59] a graphical user interface,[22] and other tools. As a viable scripting tool with libraries like common scripting languages, it can be used for scripting the Unix shell. It can parse command line arguments and execute external tools.

DrRacket IDE [edit]

DrRacket (formerly DrScheme) is widely used among introductory computer science courses that teach Scheme or Racket and is lauded for its simplicity and appeal to beginner programmers. The IDE was originally built for use with the TeachScheme! project (now ProgramByDesign), an outreach effort by Northeastern University and a number of affiliated universities for attracting high school students to computer science courses at the college level.

The editor provides highlighting for syntax and run-time errors, parenthesis matching, a debugger and an algebraic stepper. Its student-friendly features include support for multiple "language levels" (Beginning Student, Intermediate Student and so on). It also has integrated library support, and sophisticated analysis tools for advanced programmers. Further, module-oriented programming is supported with the module browser, a contour view, integrated testing and coverage measurements, and refactoring support. It provides integrated, context-sensitive access to an extensive hyper-linked help system named "Help Desk".

DrRacket is available for Windows, macOS, Unix, and Linux with the X Window System and programs behave similarly on all these platforms.

Code examples [edit]

Here is a trivial hello world program:

```
#lang racket
"Hello, World!"
```

Running this program produces the output:

```
"Hello, World!"
```

Here is a slightly less trivial program:

```
#lang racket
(require 2htdp/image)

(let sierpinski ([n 8])
  (if (zero? n)
    (triangle 2 'solid 'red)
    (let ([t (sierpinski (- n 1))])
      (freeze (above t (beside t t))))))
```

The result of this program, as shown in DrRacket

This program, taken from the Racket website, draws a Sierpinski triangle, nested to depth 8.

Using the `#lang` directive, a source file can be written in different dialects of Racket. Here is an example of the factorial program in Typed Racket, a statically typed dialect of Racket:

```
#lang typed/racket

(: fact (Integer -> Integer))
(define (fact n)
  (if (zero? n) 1 (* n (fact (- n 1)))))
```

Applications and practical use [edit]

Apart from having a basis in programming language theory, Racket was designed as a general-purpose language for production systems. Thus, the Racket distribution features an extensive library that covers systems and network programming, web development,[59] a uniform interface to the underlying operating system, a dynamic foreign function interface,[60] several flavours of regular expressions, lexer/parser generators,[58] logic programming, and a complete GUI framework.

Racket has several features useful for a commercial language, among them an ability to compile standalone executables under Windows, macOS, and Unix, a profiler and debugger included in the integrated development environment (IDE), and a unit testing framework.

Racket has been used for commercial projects and web applications. A notable example is the Hacker News website, which runs on Arc, which is developed in Racket. Naughty Dog has used it as a scripting language in several of their video games.[61]

Racket is used to teach students algebra through game design in the Bootstrap program.[62]

References [edit]

1. ^ "Racket v8.6". 11 August 2022. Retrieved 11 August 2022.
2. ^ Tobin-Hochstadt, Sam; Gerard, Sage; Dueck, Joel; Flatt, Matthew; Software Freedom Conservancy; Chestek, Pamela (2019-11-15). "Completing Racket's relicensing effort". Retrieved 2019-12-27.
3. ^ "DrRacket Files". Retrieved 21 July 2019. "The standard file extension for a Racket program file is ".rkt". The extensions ".ss", ".scm", and ".sch" are also historically popular."
4. ^ Strickland, T.S.; Fellesisen, Matthias (2010). "DLS 2010: Contracts for First-Class Classes" (PDF).
5. ^ Bonnaire-Sergeant, Ambrose (2012). *A Practical Optional Type System for Clojure* (Thesis). The University of Western Australia.
6. ^ "Planet2 questions".
7. ^ "Rust Bibliography".
8. ^ Sperber, Michael; Dybvig, R. Kent; Flatt, Matthew; Van Straaten, Anton; et al. (August 2007). "Revised⁶ Report on the Algorithmic Language Scheme (R6RS)". Scheme Steering Committee. Retrieved 2011-09-13.
9. ^ a b Felleisen, M.; Findler, R.B.; Flatt, M.; Krishnamurthi, S.; Barzilay, E.; McCarthy, J.; Tobin-Hochstadt, S. (2015). "The Racket Manifesto" (PDF). *Proceedings of the First Summit on Advances in Programming Languages*: 113–128.
10. ^ "Dialects of Racket and Scheme". Retrieved 2011-08-15.
11. ^ "Welcome to Racket". Retrieved 2019-05-15.
12. ^ a b Flatt; Findler; Krishnamurthi; Felleisen (1999). *Programming Languages as Operating Systems (or, Revenge of the Son of the Lisp Machine)*. International Conference on Functional Programming.
13. ^ a b c Findler; Clements; Flanagan; Flatt; Krishnamurthi; Steckler; Felleisen (2001). "DrScheme: A Programming Environment for Scheme" (PDF). *Journal of Functional Programming*.
14. ^ Felleisen; Findler; Flatt; Krishnamurthi (2004). "The TeachScheme! Project: Computing and Programming for Every Student". *Journal of Computer Science Education*.
15. ^ "Overview". Program by Design. Retrieved 2011-08-17.
16. ^ a b c Flatt, M.; Findler, R. B.; Felleisen, M. (2006). "Scheme with Classes, Mixins, and Traits" (PDF). *Asian Symposium on Programming Languages and Systems*.
17. ^ a b Flatt, M.; Felleisen, M. (1998). "Units: Cool Modules for Hot Languages". *Programming Language Design and Implementation*.
18. ^ a b c d Tobin-Hochstadt, S.; St-Amour, V.; Culpepper, R.; Flatt, M.; Felleisen, M. (2011). "Languages as Libraries" (PDF). *Programming Language Design and Implementation*.
19. ^ Felleisen, Matthias; Findler, Robert Bruce; Flatt, Matthew; Krishnamurthi, Shriram; Barzilay, Eli; McCarthy, Jay; Tobin-Hochstadt, Sam (2018). "A Programmable Programming Language". *Communications of the ACM*. **61** (3): 62–71. doi:10.1145/3127323. S2CID 3887010.
20. ^ "Racket: Software License". Retrieved 2015-10-20.
21. ^ Benson, Brent W. Jr. (26–28 October 1994). "libscheme: Scheme as a C Library". Written at Santa Fe, NM. *Proceedings of the USENIX Symposium on Very High Level Languages*. Berkeley, CA: USENIX Association. pp. 7–19. ISBN 978-1880446652. Retrieved 7 July 2013.
22. ^ a b c "Rebuilding Racket's Graphics Layer". 2010-12-08. Retrieved 2017-12-11.
23. ^ Clements, J.; Flatt, M.; Felleisen, M. (2001). "Modeling an Algebraic Stepper" (PDF). *European Symposium on Programming Languages*.
24. ^ a b c "Racket Core Release Notes". Archived from the original on 2013-07-05. Retrieved 2012-04-15.
25. ^ Flatt, M. (2002). "Composable and Compilable Macros". *International Conference on Functional Programming*.
26. ^ "PLT Scheme version 4.0". 2008-06-12. Archived from the original on 2013-02-02. Retrieved 2012-08-07.
27. ^ "From PLT Scheme to Racket". Racket-lang.org. Retrieved 2011-08-17.
28. ^ "Racket 5.2". PLT, Inc. 2011-11-09. Retrieved 2012-06-16.
29. ^ "Submodules". 2012-06-03. Retrieved 2012-08-07.
30. ^ "Racket 5.3". PLT, Inc. 2012-08-07. Retrieved 2012-08-07.
31. ^ "Racket 5.3.1". PLT, Inc. 2012-11-07. Retrieved 2012-11-07.
32. ^ "Racket 6.0". PLT, Inc. 2014-02-26. Retrieved 2016-02-23.
33. ^ "Racket-on-Chez Status: January 2018". 2018-01-05. Archived from the original on 2018-06-28. Retrieved 2018-04-13.
34. ^ "building Racket on Chez Scheme (Experience Report)" (PDF). 2019-08-01. Retrieved 2019-07-25.
35. ^ "Racket 7.5 release". *Packt Hub*. Retrieved 2019-11-28.
36. ^ "Racket v7.5". *Racket | Blog*. Retrieved 2019-11-28.
37. ^ "Racket v8.0".
38. ^ Flatt, M.; Yu, G.; Findler, R. B.; Felleisen, M. (2007). "Adding Delimited and Composable Control to a Production Programming Environment" (PDF). *International Conference on Functional Programming*.
39. ^ "Contracts".
40. ^ "Threads".
41. ^ "Futures".
42. ^ "Places".
43. ^ Flatt, Matthew (2012). "Creating Languages in Racket". *Communications of the ACM*. Retrieved 2012-04-08.
44. ^ Findler, R. B.; Felleisen, M. (2002). "Contracts for Higher-Order Functions" (PDF). *International Conference on Functional Programming*.
45. ^ Matthews, J. (2006). "Component Deployment with PLaneT: You Want it Where?". *Scheme and Functional Programming Workshop*.
46. ^ "The Racket package system and Planet".
47. ^ Flatt, Matthew (2002). "Composable and Compilable Macros, You Want it When?" (PDF). *International Conference on Functional Programming*.
48. ^ Flatt, Culpepper, Darais, Findler, Macros that Work Together; Compile-Time Bindings, Partial Expansion, and Definition Contexts
49. ^ Tobin-Hochstadt, S.; Felleisen, M. (2008). "The Design and Implementation of Typed Scheme". *Principles of Programming Languages*.
50. ^ Barzilay, E.; Clements, J. (2005). "Laziness Without All the Hard Work: Combining Lazy and Strict Languages for Teaching". *Functional and Declarative Programming in Education*.
51. ^ "The Hackett Programming Language". *Alexis King's Blog*. Retrieved 16 June 2019.
52. ^ The Pyret Crew (24 May 2011). "The Pyret Code; or A Rationale for the Pyret Programming Language". *Pyret*. Retrieved 16 June 2019.
53. ^ "Programming and Programming Languages". *Index of /*. 20 September 2017. Retrieved 16 June 2019.
54. ^ Flatt, M.; Barzilay, E.; Findler, R. B. (2009). "Scribble: Closing the Book on Ad Hoc Documentation Tools". *International Conference on Functional Programming*.
55. ^ Findler, R. B.; Flatt, M. (2004). "Slideshow: Functional Presentations". *International Conference on Functional Programming*.
56. ^ Felleisen, M.; Findler, R. B.; Flatt, M.; Krishnamurthi, S. (2009). "A Functional I/O System (or Fun for Freshman Kids)" (PDF). *International Conference on Functional Programming*.
57. ^ Felleisen, M.; Findler, R. B.; Flatt, M.; Krishnamurthi, S. (2004). "The Structure and Interpretation of the Computer Science Curriculum" (PDF). *Journal of Functional Programming*. **14** (4): 365–378. doi:10.1017/S0956796804005076.
58. ^ a b "Parser Tools: lex and yacc-style Parsing". Retrieved 2011-08-16.
59. ^ a b Krishnamurthi, Hopkins; McCarthy; Graunke; Pettyjohn; Felleisen (2007). "Implementation and Use of the PLT Scheme Web Server" (PDF). *Journal of Higher-Order and Symbolic Programming*. **20** (4): 431–460. doi:10.1007/s10990-007-9008-y. S2CID 17731194.
60. ^ Barzilay, E.; Orlovsky, D. (2004). "Foreign Interface for PLT Scheme" (PDF). *Scheme and Functional Programming*.
61. ^ "Functional mzScheme DSLs in Game Development". Retrieved 2012-05-08.
62. ^ "Bootstrap". *bootstrapworld.org*. Retrieved 2015-08-11.

Further reading [edit]

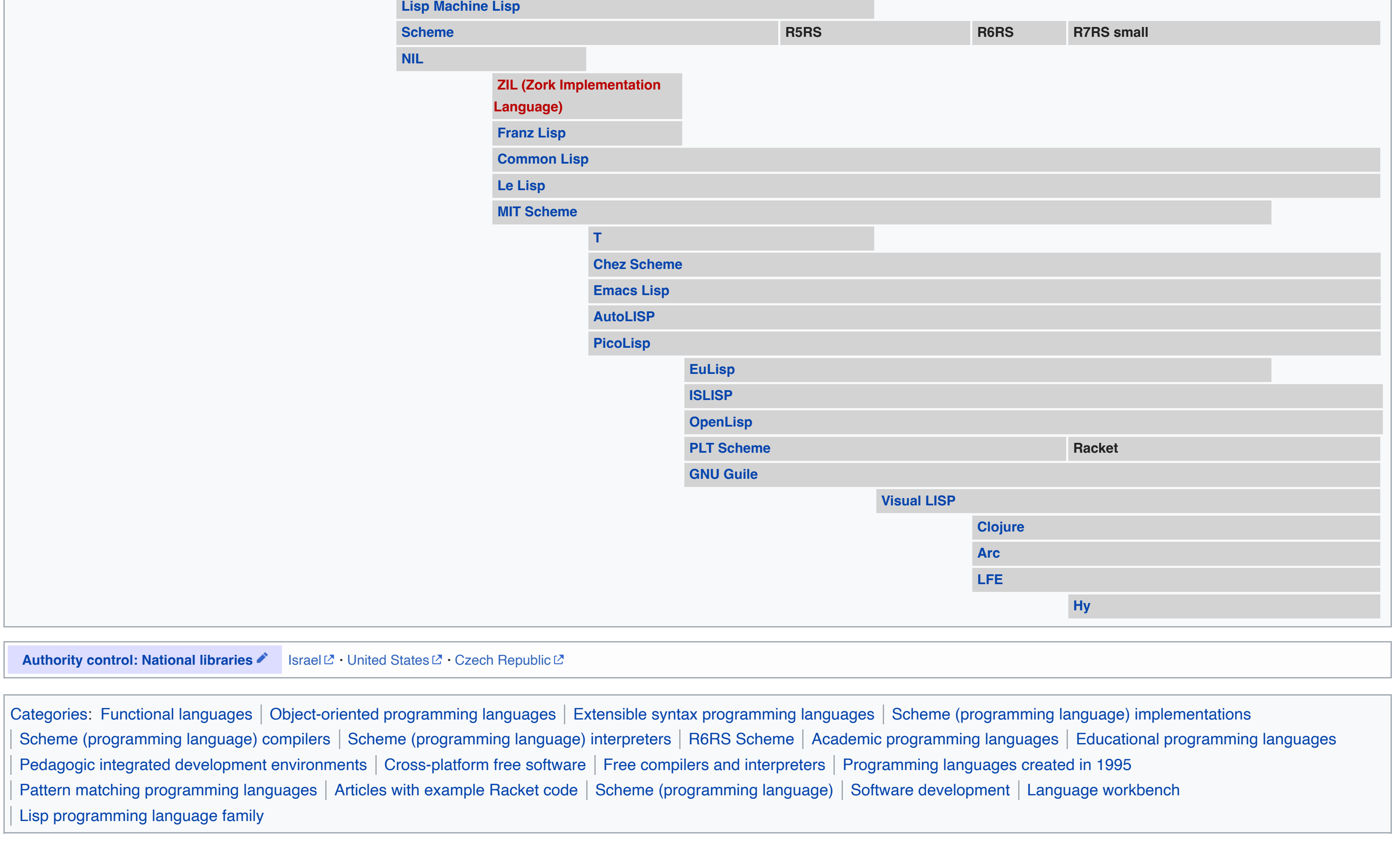
- Felleisen *et al.*, 2013. *Realm of Racket*. No Starch Press.
- Felleisen *et al.*, 2003. *How to Design Programs*. MIT Press.

External links [edit]

- Official website

V·T·E **Lisp programming language** [show]

Timeline of Lisp dialects



Authority control: National libraries: Israel · United States · Czech Republic

Categories: Functional languages | Object-oriented programming languages | Extensible syntax programming languages | Scheme (programming language) implementations | Scheme (programming language) compilers | Scheme (programming language) interpreters | R6RS Scheme | Academic programming languages | Educational programming languages | Pedagogic integrated development environments | Cross-platform free software | Free compilers and interpreters | Programming languages created in 1995 | Pattern matching programming languages | Articles with example Racket code | Scheme (programming language) | Software development | Language workbench | Lisp programming language family