Scheme (programming language)

From Wikipedia, the free encyclopedia

Scheme is a dialect of the day family of programming languages. Scheme was created during the 1970s by Gerald J. Sussman and Guy L. Steele Jr.[1] and later refined by a large community of programmers and researchers through the Internet and at meetings and workshops. Scheme is used in education, research, and industry, being popular in functional programming and in the teaching of computer science. Scheme is used in the Guile extension language for GNU software, and in various other software applications.

Scheme conforms to the lexical variant of the Algol family of programming languages, and it is the second-oldest dialect still in active use after Lisp 1.5. Scheme is known for its small, elegant, and orthogonal core language, and for its powerful macro system. Scheme was the first Lisp to make lexical scoping and closures mainstream, and it was the first dialect to require implementations to support proper tail recursion, a property that makes it practical to use recursion for looping in place of iteration. Scheme also was the first Lisp to treat continuations as first-class objects, and it was the first dialect to require implementations to support continuations with indefinite extent. Scheme is defined by a formal specification, and there are many implementations that conform to the specification. Scheme also has a standard application programming interface (API) for extending its core language, and there are many libraries that implement various functions and data structures. Scheme is used in a variety of applications, including in the Guile extension language for GNU software, and in various other software applications. Scheme programs are radically simple and include examples of Scheme code symbolically manipulating Scheme code, and Scheme has been used in artificial intelligence research and in the teaching of computer science.

Scheme is used in a variety of applications, including in the Guile extension language for GNU software, and in various other software applications. Scheme is used in artificial intelligence research and in the teaching of computer science. Scheme is used

Exponentials	`exp, log`
Complex numbers	`make-rectangular, make-polar, real-part, imag-part, magnitude, angle, complex?`
Input-output	`number->string, string->number`
Type predicates	`integer?, rational?, real?, complex?, number?`

Implementations of - and / that take more than two arguments are defined but left optional at R5RS.

Scheme Requests for Implementation [edit]

Main article: Scheme Requests for Implementation

Because of Scheme's minimalism, many common procedures and syntactic forms are not defined by the standard. In order to keep the core language small but facilitate standardization of extensions, the Scheme community has a "Scheme Request for Implementation" (SRFI) process by which extension libraries are defined through careful discussion of extension proposals. This promotes code portability. Many of the SRFIs are supported by all or most Scheme implementations.

SRFIs with fairly wide support in different implementations include:[32]

- 0: feature-based conditional expansion construct
- 1: list library
- 4: homogeneous numeric vector datatypes
- 6: basic string ports
- 8: receive, binding to multiple values
- 9: defining record types
- 13: string library
- 14: character-set library
- 16: syntax for procedures of variable arity
- 17: generalized set!
- 18: Multithreading support
- 19: time data types and procedures
- 25: multi-dimensional array primitives
- 26: notation for specializing parameters without currying
- 27: sources of random bits
- 28: basic format strings
- 29: localization
- 30: nested multi-line comments
- 31: a special form for recursive evaluation
- 37: args-fold: a program argument processor
- 39: parameter objects
- 41: streams
- 42: eager comprehensions
- 43: vector library
- 45: primitives for expressing iterative lazy algorithms
- 60: integers as bits
- 61: a more general cond clause
- 66: octet vectors
- 67: compare procedures

Implementations [edit]

Main category: Scheme (programming language) implementations

The elegant, minimalist design has made Scheme a popular target for language designers, hobbyists, and educators, and because of its small size, that of a typical interpreter, it is also a popular choice for embedded systems and scripting. This has resulted in scores of implementations,[33] most of which differ from each other so much that porting programs from one implementation to another is quite difficult, and the small size of the standard language means that writing a useful program of any great complexity in standard, portable Scheme is almost impossible.[13] The R6RS standard specifies a much broader language, in an attempt to broaden its appeal to programmers.

Almost all implementations provide a traditional Lisp-style read–eval–print loop for development and debugging. Many also compile Scheme programs to executable binary. Support for embedding Scheme code in programs written in other languages is also common, as the relative simplicity of Scheme implementations makes it a popular choice for adding scripting capabilities to larger systems developed in languages such as C. The Gambit, Chicken, and Bigloo Scheme interpreters compile Scheme to C, which makes embedding particularly easy. In addition, Bigloo's compiler can be configured to generate JVM bytecode, and it also features an experimental bytecode generator for .NET.

Some implementations support additional features. For example, Kawa and JScheme provide integration with Java classes, and the Scheme to C compilers often make it easy to use external libraries written in C, up to allowing the embedding of actual C code in the Scheme source. Another example is Pvts, which offers a set of visual tools for supporting the learning of Scheme.

Usage [edit]

Scheme is widely used by a number[34] of schools; in particular, a number of introductory Computer Science courses use Scheme in conjunction with the textbook *Structure and Interpretation of Computer Programs* (SICP).[35] For the past 12 years, PLT has run the ProgramByDesign (formerly TeachScheme!) project, which has exposed close to 600 high school teachers and thousands of high school students to rudimentary Scheme programming. MIT's old introductory programming class 6.001 was taught in Scheme,[36] Although 6.001 has been replaced by more modern courses, SICP continues to be taught at MIT.[37] Likewise, the introductory class at UC Berkeley, CS 61A, was until 2011 taught entirely in Scheme, save minor diversions into Logo to demonstrate dynamic scope. Today, like MIT, Berkeley has replaced the syllabus with a more modern version that is primarily taught in Python 3, but the current syllabus is still based on the old curriculum, and parts of the class are still taught in Scheme.[38]

The textbook *How to Design Programs* by Matthias Felleisen, currently at Northeastern University, is used by some institutes of higher education for their introductory computer science courses. Both Northeastern University and Worcester Polytechnic Institute use Scheme exclusively for their introductory courses Fundamentals of Computer Science (CS2500) and Introduction to Program Design (CS1101), respectively.[39][40] Rose-Hulman uses Scheme in its more advanced Programming Language Concepts course.[41] Brandeis University's core course, Structure and Interpretations of Computer Programs (COSI121b), is also taught exclusively in Scheme by theoretical computer scientist Harry Mairson.[42] Indiana University's introductory class, C211, is taught entirely in Scheme. A self-paced version of the course, CS 61AS, continues to use Scheme.[43] The introductory computer science courses at Yale and Grinnell College are also taught in Scheme.[44] Programming Design Paradigms,[45] a mandatory course for the Computer science Graduate Students at Northeastern University, also extensively uses Scheme. The former introductory Computer Science course at the University of Minnesota - Twin Cities, CSCI 1901, also used Scheme as its primary language, followed by a course that introduced students to the Java programming language;[46] however, following the example of MIT, the department replaced 1901 with the Python-based CSCI 1133,[47] while functional programming is covered in detail in the third-semester course CSCI 2041.[48] In the software industry, Tata Consultancy Services, Asia's largest software consultancy firm, uses Scheme in their month-long training program for fresh college graduates.[*citation needed*]

Scheme is/was also used for the following:

- The Document Style Semantics and Specification Language (DSSSL), which provides a method of specifying SGML stylesheets, uses a Scheme subset.[49]
- The well-known open source raster graphics editor GIMP uses TinyScheme as a scripting language.[50]
- Guile has been adopted by GNU project as its official scripting language, and that implementation of Scheme is embedded in such applications as GNU LilyPond and GnuCash as a scripting language for extensions. Likewise, Guile used to be the scripting language for the desktop environment GNOME,[51] and GNOME still has a project that provides Guile bindings to its library stack.[52] There is a project to incorporate Guile into GNU Emacs, GNU's flagship program, replacing the current Emacs Lisp interpreter.[*citation needed*]
- Elk Scheme is used by Synopsys as a scripting language for its technology CAD (TCAD) tools.[53]
- Shiro Kawai, senior programmer on the movie *Final Fantasy: The Spirits Within*, used Scheme as a scripting language for managing the real-time rendering engine.[54]
- Google App Inventor for Android uses Scheme, where Kawa is used to compile the Scheme code down to byte-codes for the Java Virtual Machine running on Android devices.[55]

See also [edit]

- *Essentials of Programming Languages*, textbook using Scheme as foundation

Computer programming portal

References [edit]

1. ^ Common LISP: The Language, 2nd Ed., Guy L. Steele Jr. Digital Press; 1981. ISBN 978-1-55558-041-4. "Common Lisp is a new dialect of Lisp, a successor to MacLisp, influenced strongly by ZetaLisp and to some extent by Scheme and InterLisp."
2. ^ a b b 1178-1990 (Reaff 2008) IEEE Standard for the Scheme Programming Language. IEEE part number STDPD14209, unanimously reaffirmed at a meeting of the IEEE-SA Standards Board Standards Review Committee (RevCom), March 26, 2008 (item 6.3 on minutes), reaffirmation minutes accessed October 2009. NOTE: this document is only available for purchase from IEEE and is not available online at the time of writing (2009).
3. ^ a b c d e f g h i j k l m n o p q Richard Kelsey; William Clinger; Jonathan Rees; et al. (August 1998). "Revised5 Report on the Algorithmic Language Scheme". *Higher-Order and Symbolic Computation*. **11** (1): 7–105. doi:10.1023/A:1010051815785. S2CID 14069423. Retrieved 2012-08-09.
4. ^ a b b "R7RS final available" (PDF). 2013-07-06.
5. ^ a b c d 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.
6. ^ a b c d Sussman, Gerald Jay; Steele, Guy L. (1 December 1998). "The First Report on Scheme Revisited". *Higher-Order and Symbolic Computation*. **11** (4): 399–404. doi:10.1023/A:1010079421970. S2CID 7704398.
7. ^ "R6RS Implementations". r6rs.org. Retrieved 2017-11-24.
8. ^ Abdulaziz Ghuloum (2007-10-27). "R6RS Libraries and syntax-case system (psyntax)". Ikarus Scheme. Retrieved 2009-10-20.
9. ^ Keep, Andrew W.; Dybvig, R. Kent (November 2014). "A run-time representation of scheme record types". *Journal of Functional Programming*. **24** (6): 675–716. doi:10.1017/S0956796814000203. S2CID 40001845.
10. ^ "Revised^6 Report on the Algorithmic Language Scheme, Appendix E: language changes". Scheme Steering Committee. 2007-09-26. Retrieved 2009-10-20.
11. ^ "R6RS Electorate". Scheme Steering Committee. 2007. Retrieved 2012-08-09.
12. ^ Marc Feeley (compilation) (2007-10-26). "Implementors' intentions concerning R6RS". Scheme Steering Committee, r6rs-discuss mailing list. Retrieved 2012-08-09.
13. ^ a b Will Clinger, Marc Feeley, Chris Hanson, Jonathan Rees and Olin Shivers (2009-08-20). "Position Statement *(draft)*". Scheme Steering Committee. Retrieved 2012-08-09
14. ^ "R7RS 9th draft available" (PDF). 2013-04-15.
15. ^ Will Clinger (2013-05-10). "extension of voting period". Scheme Language Steering Committee, scheme-reports mailing list. Archived from the original on 2013-07-21. Retrieved 2013-07-07.
16. ^ The Scheme 48 implementation is so-named because the interpreter was written by Richard Kelsey and Jonathan Rees in 48 hours (August 6th – 7th, 1986. See Richard Kelsey; Jonathan Rees; Mike Sperber (2008-01-10). "The Incomplete Scheme 48 Reference Manual for release 1.8". Jonathan Rees, s48.org. Retrieved 2012-08-09.
17. ^ a b b Gerald Jay Sussman & Guy Lewis Steele Jr. (December 1975). "Scheme: An Interpreter for Extended Lambda Calculus" (PDF). *AI Memos*. MIT AI Lab. AIM-349. hdl:1721.1/5794. Retrieved 23 December 2021.
18. ^ Joel Moses (June 1970), *The Function of FUNCTION in LISP, or Why the FUNARG Problem Should Be Called the Environment Problem*, hdl:1721.1/5854, AI Memo 199, "A useful metaphor for the difference between FUNCTION and QUOTE in LISP is to think of QUOTE as a porous or an open covering of the function since free variables escape to the current environment. FUNCTION acts as a closed or nonporous covering (hence the term "closure" used by Landin). Thus we talk of "open" Lambda expressions (functions in LISP are usually Lambda expressions) and "closed" Lambda expressions. [...] My interest in the environment problem began while Landin, who had a deep understanding of the problem, visited MIT during 1966-67. I then realized the correspondence between the FUNARG lists which are the results of the evaluation of "closed" Lambda expressions in LISP and ISWIM's Lambda Closures."
19. ^ van Tonder, André (1 January 2004). "A Lambda Calculus for Quantum Computation". *SIAM Journal on Computing*. **33** (5): 1109–1135. arXiv:quant-ph/0307150. doi:10.1137/S0097539703432165. S2CID 613571.
20. ^ Niehren, J.; Schwinghammer, J.; Smolka, G. (November 2006). "A concurrent lambda calculus with futures" (PDF). *Theoretical Computer Science*. **364** (3): 338–356. doi:10.1016/j.tcs.2006.08.016.
21. ^ Gerald Jay Sussman & Guy Lewis Steele Jr. (March 1976). "Lambda: The Ultimate Imperative". *AI Memos*. MIT AI Lab. AIM-353. Archived from the original (postscript or PDF) on 2016-05-10. Retrieved 2012-08-09.
22. ^ Gabriel, Richard P.; Pitman, Kent (1988). "Technical Issues of Separation in Function Cells and Value Cells". *Lisp and Symbolic Computation*. Vol. 1, no. 1 (published June 1988). pp. 81–101. doi:10.1007/BF01806178. Retrieved 2012-08-09.
23. ^ a b b Philip L. Bewig (2008-01-24). "SRFI 41: Streams". The SRFI Editors, schemers.org. Retrieved 2012-08-09.
24. ^ William Clinger and Jonathan Rees, ed. (1991). "Revised4 Report on the Algorithmic Language Scheme". *ACM Lisp Pointers*. **4** (3): 1–55. Retrieved 2012-08-09.
25. ^ Flatt, Matthew (2016). "Binding as sets of scopes". *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. pp. 705–717. doi:10.1145/2837614.2837620. ISBN 978-1-4503-3549-2. S2CID 15401805.
26. ^ Jonathan Rees, The Scheme of Things The June 1992 Meeting Archived 2011-07-16 at the Wayback Machine (postscript), in Lisp Pointers, V(4), October–December 1992. Retrieved 2012-08-09
27. ^ Taylor Campbell (2005-07-21). "SRFI 62: S-expression comments". The SRFI Editors, schemers.org. Retrieved 2012-08-09.
28. ^ William D Clinger (1999-07-01). "SRFI 6: Basic String Ports". The SRFI Editors, schemers.org. Retrieved 2012-08-09.
29. ^ Scott G. Miller (2002-06-25). "SRFI 28: Basic Format Strings". The SRFI Editors, schemers.org. Retrieved 2012-08-09.
30. ^ J.W. Backus; F.L. Bauer; J.Green; C. Katz; J. McCarthy P. Naur; et al. (January–April 1960). "Revised Report on the Algorithmic Language Algol 60". *Numerische Mathematik, Communications of the ACM, and Journal of the British Computer Society*. Retrieved 2012-08-09.
31. ^ Jonathan Rees; William Clinger, eds. (December 1986). "Revised(3) Report on the Algorithmic Language Scheme (Dedicated to the Memory of ALGOL 60)". *ACM SIGPLAN Notices*. **21** (12): 37–79. CiteSeerX 10.1.1.29.2012. doi:10.1145/15042.15043. hdl:1721.1/6424. S2CID 43884422. Retrieved 2012-08-09.
32. ^ "Scheme Systems Supporting SRFIs". The SRFI Editors, schemers.org. 2009-08-30. Retrieved 2012-08-09.
33. ^ 75 known implementations of Scheme are listed by "scheme-faq-standards". Community Scheme Wiki. 2009-06-25. Retrieved 2009-10-20.
34. ^ Ed Martin (2009-07-20). "List of Scheme-using schools". Schemers Inc. Retrieved 2009-10-20.
35. ^ "List of SICP-using schools". MIT Press. 1999-01-26. Retrieved 2009-10-20.
36. ^ Eric Grimson (Spring 2005). "6.001 Structure and Interpretation of Computer Programs". MIT Open Courseware. Retrieved 2009-10-20.
37. ^ Alex Vandiver; Nelson Elhage; et al. (January 2009). "6.184 - Zombies drink caffeinated 6.001". MIT CSAIL. Retrieved 2009-10-20.
38. ^ John DeNero (Fall 2019). "Computer Science 61A, Berkeley". Department of Electrical Engineering and Computer Sciences, Berkeley. Retrieved 2019-12-17.
39. ^ CS 2500: Fundamentals of Computer Science I, Northeastern University
40. ^ CS 1101: Introduction to Program Design (A05): course software, Worcester Polytechnic Institute
41. ^ "CSSE 304: Programming Language Concepts". Rose-Hulman Institute of Technology.
42. ^ "Spring 2021 CS121b Syllabus" (PDF). Brandeis University.
43. ^ "Home". *berkeley-cs61as.github.io*.
44. ^ Dana Angluin (Fall 2009). "Introduction to Computer Science (CPSC 201)". The Zoo, Yale University Computer Science Department. Retrieved 2009-10-20.
45. ^ "Programming Design Paradigms CSG107 Course Readings". Northeastern University College of Computer and Information Science. Fall 2009. Retrieved 2012-08-09.
46. ^ Structure of Computer Programming I Archived 2010-06-19 at the Wayback Machine, Computer Science Department, University of Minnesota, Spring 2010 (accessed 2010-01-30).
47. ^ CSci Required Class Course Descriptions and Other Information Archived 2019-10-25 at the Wayback Machine, Computer Science Department, University of Minnesota (accessed 2019-10-25)
48. ^ CSCI 2041 – New Course CSE Curriculum Committee, University of Minnesota (accessed 2019-10-25)
49. ^ Robin Cover (2002-02-25). "DSSSL - Document Style Semantics and Specification Language. ISO/IEC 10179:1996". Cover Pages. Retrieved 2012-08-09.
50. ^ *"The major scripting language for the GIMP that has been attached to it today is Scheme."* From Dov Grobgeld (2002). "The GIMP Basic Scheme Tutorial". The GIMP Team. Retrieved 2012-08-09.
51. ^ Todd Graham Lewis; David Zoll; Julian Missig (2002). "GNOME FAQ from Internet Archive". The Gnome Team, gnome.org. Archived from the original on 2000-05-22. Retrieved 2012-08-09.
52. ^ "guile-gnome". Free Software Foundation. Retrieved 2012-08-09.
53. ^ Laurence Breward (2006-11-09). "Synopsys MAP-inSM Program Update: EDA Interoperability Developers' Forum" (PDF). Synopsis Inc. Retrieved 2012-08-09.
54. ^ Kawai, Shiro (October 2002). "Gluing Things Together - Scheme in the Real-time CG Content Production". *Proceedings of the First International Lisp Conference, San Francisco*: 342–348. Retrieved 2012-08-09.
55. ^ Bill Magnuson; Hal Abelson & Mark Friedman (2009-08-11). "Under the Hood of App Inventor for Android". Google Inc, Official Google Research blog. Retrieved 2012-08-09.

Further reading [edit]

- An Introduction to Scheme and its Implementation (a mirror)
- Christopher T. Haynes (1999-06-22). "The Scheme Programming Language Standardization Experience".
- Guy L. Steele Jr., Richard P. Gabriel. "The Evolution of Lisp" (PDF).
- Gerald Jay Sussman & Guy Lewis Steele Jr. (December 1975). *Scheme: An Interpreter for Extended Lambda Calculus*. Vol. AI Memo 349. MIT Artificial Intelligence Lab. CiteSeerX 10.1.1.128.80 – via Wikisource.

External links [edit]

- Scheme at Curlie
- schemers.org provides links to a number of Scheme resources including the specifications
- Scheme Programming at Wikibooks
- Write Yourself a Scheme in 48 Hours at Wikibooks
- Media related to Scheme (programming language) at Wikimedia Commons
- Bookmarklet that add Interactive Scheme REPL to any website

v · t · e **Lisp programming language** [show]

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

Categories: Scheme (programming language) | Academic programming languages | Articles with example Scheme (programming language) code | Dynamically typed programming languages | Extensible syntax programming languages | Lisp programming language family | Multi-paradigm programming languages | Programming languages created in 1975

This page was last edited on 27 August 2022, at 03:21 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License 3.0; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

Privacy policy About Wikipedia Disclaimers Contact Wikipedia Mobile view Developers Statistics Cookie statement

Powered by MediaWiki