## Ruby (programming language )

**Ruby** is a [dynamic](http://en.wikipedia.org/wiki/Dynamic_programming_language), [reflective](http://en.wikipedia.org/wiki/Reflection_(computer_science)), general-purpose object-oriented programming language that combines syntax inspired by [Perl](http://en.wikipedia.org/wiki/Perl) with [Smalltalk](http://en.wikipedia.org/wiki/Smalltalk)-like features. It was also influenced by [Eiffel](http://en.wikipedia.org/wiki/Eiffel_(programming_language)" \o "Eiffel (programming language))and [Lisp](http://en.wikipedia.org/wiki/Lisp_(programming_language)). Ruby was first designed and developed in the mid-1990s by [Yukihiro "Matz" Matsumoto](http://en.wikipedia.org/wiki/Yukihiro_Matsumoto) in [Japan](http://en.wikipedia.org/wiki/Japan).

Ruby supports multiple [programming paradigms](http://en.wikipedia.org/wiki/Programming_paradigm), including [functional](http://en.wikipedia.org/wiki/Functional_programming), [object oriented](http://en.wikipedia.org/wiki/Object-oriented_programming), [imperative](http://en.wikipedia.org/wiki/Imperative_programming) and [reflective](http://en.wikipedia.org/wiki/Reflective_programming). It also has a [dynamic type](http://en.wikipedia.org/wiki/Dynamic_type) system and automatic [memory management](http://en.wikipedia.org/wiki/Memory_management); it is therefore similar in varying respects to [Smalltalk](http://en.wikipedia.org/wiki/Smalltalk), [Python](http://en.wikipedia.org/wiki/Python_(programming_language)), [Perl](http://en.wikipedia.org/wiki/Perl), [Lisp](http://en.wikipedia.org/wiki/Lisp_(programming_language)), [Dylan](http://en.wikipedia.org/wiki/Dylan_(programming_language)), [Pike](http://en.wikipedia.org/wiki/Pike_(programming_language)), and [CLU](http://en.wikipedia.org/wiki/CLU_(programming_language)).

The standard 1.8.7 [implementation](http://en.wikipedia.org/wiki/Ruby_MRI) is written in [C](http://en.wikipedia.org/wiki/C_(programming_language)), as a single-pass [interpreted language](http://en.wikipedia.org/wiki/Interpreted_language). The specifications of Ruby language are currently being developed by the Open Standards Promotion Center of the Information-Technology Promotion Agency (a [Japanese government](http://en.wikipedia.org/wiki/Government_of_Japan) agency) for submission to the Japanese Industrial Standards Committee and then to the[International Organization for Standardization](http://en.wikipedia.org/wiki/International_Organization_for_Standardization). The [final draft](http://www.ipa.go.jp/osc/english/ruby/index.html) of specifications is yet to be accepted as an Industrial Standard. As of 2010, there are a number of complete or upcoming alternative implementations of Ruby, including [YARV](http://en.wikipedia.org/wiki/YARV), [JRuby](http://en.wikipedia.org/wiki/JRuby" \o "JRuby), [Rubinius](http://en.wikipedia.org/wiki/Rubinius" \o "Rubinius), [IronRuby](http://en.wikipedia.org/wiki/IronRuby" \o "IronRuby), [MacRuby](http://en.wikipedia.org/wiki/MacRuby" \o "MacRuby), and [HotRuby](http://en.wikipedia.org/wiki/HotRuby" \o "HotRuby). Each takes a different approach, with IronRuby, JRuby and MacRuby providing[just-in-time compilation](http://en.wikipedia.org/wiki/Just-in-time_compilation) and MacRuby also providing [ahead-of-time compilation](http://en.wikipedia.org/wiki/Ahead-of-time_compilation). The official 1.9 branch uses [YARV](http://en.wikipedia.org/wiki/YARV), as will 2.0 (development), and will eventually supersede the slower[Ruby MRI](http://en.wikipedia.org/wiki/Ruby_MRI).

## History

Ruby was conceived on February 24, 1993 by [Yukihiro Matsumoto](http://en.wikipedia.org/wiki/Yukihiro_Matsumoto) who wished to create a new language that balanced [functional programming](http://en.wikipedia.org/wiki/Functional_programming) with [imperative programming](http://en.wikipedia.org/wiki/Imperative_programming).[[8]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-7)Matsumoto has said, "I wanted a [scripting language](http://en.wikipedia.org/wiki/Scripting_language) that was more powerful than [Perl](http://en.wikipedia.org/wiki/Perl), and more [object-oriented](http://en.wikipedia.org/wiki/Object-oriented) than [Python](http://en.wikipedia.org/wiki/Python_(programming_language)). That's why I decided to design my own language."[[9]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-linuxdevcenter-8)

At a Google Tech Talk in 2008 Matsumoto further stated, "I hope to see Ruby help every programmer in the world to be productive, and to enjoy programming, and to be happy. That is the primary purpose of Ruby language."[[10]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-googletechtalk-9)

# Choice of the name "Ruby"

The name "Ruby" was decided on during an online chat session between Matsumoto and [Keiju Ishitsuka](http://en.wikipedia.org/w/index.php?title=Keiju_Ishitsuka&action=edit&redlink=1" \o "Keiju Ishitsuka (page does not exist)) on February 24, 1993, before any code had been written for the language.[[11]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-rubyconf-history-of-ruby-10)Initially two names were proposed: "Coral" and "Ruby", with the latter being chosen by Matsumoto in a later email to Ishitsuka.[[12]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-11) Matsumoto has later remarked that a factor in choosing the name "Ruby" was because it was the [birthstone](http://en.wikipedia.org/wiki/Birthstone) of one of his colleagues.[*[citation needed](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed" \o "Wikipedia:Citation needed)*]

# First publication

The first public release of Ruby 0.95 was announced on Japanese domestic [newsgroups](http://en.wikipedia.org/wiki/Newsgroup) on December 21, 1995.[[13]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-12)[[14]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-13) Subsequently three more versions of Ruby were released in two days.[[11]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-rubyconf-history-of-ruby-10) The release coincided with the launch of the Japanese-language *ruby-list* mailing list, which was the first mailing list for the new language.

Already present at this stage of development were many of the features familiar in later releases of Ruby, including [object-oriented](http://en.wikipedia.org/wiki/Object_oriented_programming) design, [classes](http://en.wikipedia.org/wiki/Class_(computer_science)) with inheritance, [mixins](http://en.wikipedia.org/wiki/Mixin" \o "Mixin), [iterators](http://en.wikipedia.org/wiki/Iterator" \o "Iterator),[closures](http://en.wikipedia.org/wiki/Closure_(computer_science)), [exception handling](http://en.wikipedia.org/wiki/Exception_handling), and [garbage collection](http://en.wikipedia.org/wiki/Garbage_collection_(computer_science)).[[15]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-14)

# Ruby 1.0

Ruby reached version 1.0 on December 25, 1996.[[11]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-rubyconf-history-of-ruby-10)

Following the release of Ruby 1.3 in 1999 the first English language mailing list ruby-talk began,[[9]](http://en.wikipedia.org/wiki/Ruby_(programming_language)" \l "cite_note-linuxdevcenter-8) which signalled a growing interest in the language outside of Japan. In September 2000, the first English language book [Programming Ruby](http://en.wikipedia.org/wiki/Programming_Ruby) was printed, which was later freely released to the public further widening the adoption of Ruby amongst English speakers.

# Ruby on Rails

Around 2005, interest in the Ruby language surged in tandem with [Ruby on Rails](http://en.wikipedia.org/wiki/Ruby_on_Rails), a popular [web application framework](http://en.wikipedia.org/wiki/Web_application_framework) written in Ruby. Rails is frequently credited with making Ruby "famous".[[16]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-15)

Ruby 1.9

The latest stable version of the reference implementation is 1.9.3 and is dual-licensed under the Ruby License and a BSD License.

Ruby 1.9 introduces many significant changes over the 1.8 series. Some examples are:

* [Block local](http://en.wikipedia.org/wiki/Local_variable) variables (variables that are local to the [block](http://en.wikipedia.org/wiki/Block_(programming)) in which they are declared)
* An additional [lambda](http://en.wikipedia.org/wiki/Anonymous_function) syntax (fun = ->(a,b) { puts a + b })
* Per-string [character encodings](http://en.wikipedia.org/wiki/Character_encoding) are supported
* New socket API ([IPv6](http://en.wikipedia.org/wiki/IPv6) support)
* require\_relative import security

Ruby 1.9 will be followed by Ruby 2.0. [[17]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-16)

## Philosophy

Matsumoto has said that Ruby is designed for programmer productivity and fun, following the principles of good [user interface](http://en.wikipedia.org/wiki/User_interface) design.[[18]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-17) He stresses that systems design needs to emphasize human, rather than computer, needs:[[19]](http://en.wikipedia.org/wiki/Ruby_(programming_language)" \l "cite_note-artima-18)

|  |  |  |
| --- | --- | --- |
| **“** | Often people, especially computer engineers, focus on the machines. They think, "By doing this, the machine will run faster. By doing this, the machine will run more effectively. By doing this, the machine will something something something." They are focusing on machines. But in fact we need to focus on humans, on how humans care about doing programming or operating the application of the machines. We are the masters. They are the slaves. | **”** |

Ruby is said to follow the [principle of least astonishment](http://en.wikipedia.org/wiki/Principle_of_least_astonishment) (POLA), meaning that the language should behave in such a way as to minimize confusion for experienced users. Matsumoto has said his primary design goal was to make a language which he himself enjoyed using, by minimizing programmer work and possible confusion. He has said that he had not applied the principle of least surprise to the design of Ruby,[[19]](http://en.wikipedia.org/wiki/Ruby_(programming_language)" \l "cite_note-artima-18) but nevertheless the phrase has come to be closely associated with the Ruby programming language. The phrase has itself been a source of surprise, as novice users may take it to mean that Ruby's behaviors try to closely match behaviors familiar from other languages. In a May 2005, discussion on the newsgroup comp.lang.ruby, Matsumoto attempted to distance Ruby from POLA, explaining that because any design choice will be surprising to someone, he uses a personal standard in evaluating surprise. If that personal standard remains consistent, there would be few surprises for those familiar with the standard.[[20]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-19)

Matsumoto defined it this way in an interview:[[19]](http://en.wikipedia.org/wiki/Ruby_(programming_language)" \l "cite_note-artima-18)

|  |  |  |
| --- | --- | --- |
| **“** | Everyone has an individual background. Someone may come from [Python](http://en.wikipedia.org/wiki/Python_(programming_language)), someone else may come from [Perl](http://en.wikipedia.org/wiki/Perl), and they may be surprised by different aspects of the language. Then they come up to me and say, 'I was surprised by this feature of the language, so Ruby violates the principle of least surprise.' Wait. Wait. The principle of least surprise is not for you only. The principle of least surprise means principle of least*my* surprise [[*sic*](http://en.wikipedia.org/wiki/Sic)]. And it means the principle of least surprise after you learn Ruby very well. For example, I was a [C++](http://en.wikipedia.org/wiki/C%2B%2B) programmer before I started designing Ruby. I programmed in [C++](http://en.wikipedia.org/wiki/C%2B%2B) exclusively for two or three years. And after two years of C++ programming, it still surprises me. | **”** |

## Features

* Thoroughly [object-oriented](http://en.wikipedia.org/wiki/Object-oriented) with [inheritance](http://en.wikipedia.org/wiki/Inheritance_(computer_science)), [mixins](http://en.wikipedia.org/wiki/Mixin) and [metaclasses](http://en.wikipedia.org/wiki/Metaclass) [[21]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-stewart-20)
* [Dynamic typing](http://en.wikipedia.org/wiki/Dynamic_typing) and [Duck typing](http://en.wikipedia.org/wiki/Duck_typing)
* Everything is an [expression](http://en.wikipedia.org/wiki/Expression_(programming)) (even [statements](http://en.wikipedia.org/wiki/Statement_(programming))) and everything is executed [imperatively](http://en.wikipedia.org/wiki/Imperative_programming) (even [declarations](http://en.wikipedia.org/wiki/Declaration_(computer_science)))
* Succinct and flexible syntax [[22]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-venners-productivity-21) that minimizes [syntactic noise](http://en.wikipedia.org/w/index.php?title=Syntactic_noise&action=edit&redlink=1) and serves as a foundation for [domain-specific languages](http://en.wikipedia.org/wiki/Domain-specific_languages) [[23]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-fowler-dsl-22)
* Dynamic [reflection](http://en.wikipedia.org/wiki/Reflection_(computer_science)) and [alteration](http://en.wikipedia.org/wiki/Dynamic_programming_language#Object_runtime_alteration) of objects to facilitate [metaprogramming](http://en.wikipedia.org/wiki/Metaprogramming" \o "Metaprogramming)[[24]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-codeproject-dynamic-23)
* [Lexical closures](http://en.wikipedia.org/wiki/Closure_(computer_science)), [iterators](http://en.wikipedia.org/wiki/Iterator" \o "Iterator) and [generators](http://en.wikipedia.org/wiki/Generator_(computer_science)), with a unique [block syntax](http://en.wikipedia.org/wiki/Ruby_(programming_language)#Blocks_and_iterators) [[25]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-venners-closures-24)
* Literal notation for [arrays](http://en.wikipedia.org/wiki/Dynamic_array), [hashes](http://en.wikipedia.org/wiki/Associative_array), [regular expressions](http://en.wikipedia.org/wiki/Regular_expressions) and [symbols](http://en.wikipedia.org/wiki/Symbol_(Lisp))
* Embedding code in strings ([interpolation](http://en.wikipedia.org/wiki/Variable_interpolation))
* [Default arguments](http://en.wikipedia.org/wiki/Default_arguments)
* Four levels of variable scope ([global](http://en.wikipedia.org/wiki/Global_variable), [class](http://en.wikipedia.org/wiki/Class_variable), [instance](http://en.wikipedia.org/wiki/Instance_variable), and [local](http://en.wikipedia.org/wiki/Local_variable)) denoted by [sigils](http://en.wikipedia.org/wiki/Sigil_(computer_programming)) or the lack thereof
* [Garbage collection](http://en.wikipedia.org/wiki/Garbage_collection_(computer_science))
* [First-class continuations](http://en.wikipedia.org/wiki/First-class_continuations)
* Strict boolean [coercion](http://en.wikipedia.org/wiki/Implicit_type_conversion) rules (everything is true except false and [nil](http://en.wikipedia.org/wiki/Nil))
* [Exception handling](http://en.wikipedia.org/wiki/Exception_handling)
* [Operator overloading](http://en.wikipedia.org/wiki/Operator_overloading)
* Built-in support for [rational numbers](http://en.wikipedia.org/wiki/Rational_numbers), [complex numbers](http://en.wikipedia.org/wiki/Complex_numbers) and [arbitrary-precision arithmetic](http://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic)
* Custom dispatch behavior (through method\_missing and const\_missing)
* Native [threads](http://en.wikipedia.org/wiki/Thread_(computer_science)) and cooperative [fibers](http://en.wikipedia.org/wiki/Fiber_(computer_science))
* Initial support for [Unicode](http://en.wikipedia.org/wiki/Unicode) and multiple [character encodings](http://en.wikipedia.org/wiki/Character_encodings) (still buggy as of version 1.9)[[26]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-rubyunicodesupport-25)
* Native [plug-in](http://en.wikipedia.org/wiki/Plug-in_(computing)) API in [C](http://en.wikipedia.org/wiki/C_(programming_language))
* [Interactive Ruby Shell](http://en.wikipedia.org/wiki/Interactive_Ruby_Shell) (a [REPL](http://en.wikipedia.org/wiki/REPL))
* Centralized package management through [RubyGems](http://en.wikipedia.org/wiki/RubyGems" \o "RubyGems)
* Implemented on all major platforms
* Large standard library

## Semantics

Ruby is [object-oriented](http://en.wikipedia.org/wiki/Object-oriented): every value is an object, including classes and instances of types that many other languages designate as primitives (such as [integers](http://en.wikipedia.org/wiki/Integer_(computer_science)), booleans, and "[nil](http://en.wikipedia.org/wiki/Null_pointer)"). Variables always hold references to objects. Every [function](http://en.wikipedia.org/wiki/Function_(programming)) is a [method](http://en.wikipedia.org/wiki/Method_(programming)) and methods are always called on an object. Methods defined at the top level scope become members of the Object class. Since this class is an ancestor of every other class, such methods can be called on any object. They are also visible in all scopes, effectively serving as "global" procedures. Ruby supports [inheritance](http://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)) with [dynamic dispatch](http://en.wikipedia.org/wiki/Dynamic_dispatch), [mixins](http://en.wikipedia.org/wiki/Mixin" \o "Mixin) and singleton methods (belonging to, and defined for, a single [instance](http://en.wikipedia.org/wiki/Instance_(computer_science)" \o "Instance (computer science))rather than being defined on the class). Though Ruby does not support [multiple inheritance](http://en.wikipedia.org/wiki/Multiple_inheritance), classes can import [modules](http://en.wikipedia.org/wiki/Module_(programming)) as mixins.

Ruby has been described as a [multi-paradigm programming language](http://en.wikipedia.org/wiki/Multi-paradigm_programming_language): it allows procedural programming (defining functions/variables outside classes makes them part of the root, 'self' Object), with object orientation (everything is an object) or [functional programming](http://en.wikipedia.org/wiki/Functional_programming) (it has [anonymous functions](http://en.wikipedia.org/wiki/Anonymous_function), [closures](http://en.wikipedia.org/wiki/Closure_(computer_science)), and [continuations](http://en.wikipedia.org/wiki/Continuation); statements all have values, and functions return the last evaluation). It has support for [introspection](http://en.wikipedia.org/wiki/Introspection_(computer_science)), [reflection](http://en.wikipedia.org/wiki/Reflection_(computer_science)) and [metaprogramming](http://en.wikipedia.org/wiki/Metaprogramming" \o "Metaprogramming), as well as support for interpreter-based[[27]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-26) [threads](http://en.wikipedia.org/wiki/Thread_(computer_science)). Ruby features [dynamic typing](http://en.wikipedia.org/wiki/Type_system), and supports [parametric polymorphism](http://en.wikipedia.org/wiki/Parametric_polymorphism).

According to the Ruby [FAQ](http://en.wikipedia.org/wiki/FAQ),[[28]](http://en.wikipedia.org/wiki/Ruby_(programming_language)" \l "cite_note-27) "If you like [Perl](http://en.wikipedia.org/wiki/Perl), you will like Ruby and be right at home with its syntax. If you like [Smalltalk](http://en.wikipedia.org/wiki/Smalltalk), you will like Ruby and be right at home with its semantics. If you like [Python](http://en.wikipedia.org/wiki/Python_(programming_language)), you may or may not be put off by the huge difference in design philosophy between Python and Ruby/Perl."

## Syntax

The syntax of Ruby is broadly similar to [Perl](http://en.wikipedia.org/wiki/Perl) and [Python](http://en.wikipedia.org/wiki/Python_(programming_language)). Class and method definitions are signaled by keywords. In contrast to Perl, variables are not obligatorily prefixed with a [sigil](http://en.wikipedia.org/wiki/Sigil_(computer_programming)). When used, the sigil changes the semantics of scope of the variable. One difference from C and [Perl](http://en.wikipedia.org/wiki/Perl) is that keywords are typically used to define logical code blocks, without braces (i.e., pair of { and }). For practical purposes there is no distinction between [expressions](http://en.wikipedia.org/wiki/Expression_(programming)) and[statements](http://en.wikipedia.org/wiki/Statement_(programming)).[[29]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-28) Line breaks are significant and taken as the end of a statement; a semicolon may be equivalently used. Unlike Python, indentation is not significant.

One of the differences of Ruby compared to [Python](http://en.wikipedia.org/wiki/Python_(programming_language)) and [Perl](http://en.wikipedia.org/wiki/Perl) is that Ruby keeps all of its instance variables completely private to the class and only exposes them through accessor methods (attr\_writer, attr\_reader, etc.). Unlike the "getter" and "setter" methods of other languages like [C++](http://en.wikipedia.org/wiki/C%2B%2B) or [Java](http://en.wikipedia.org/wiki/Java_(programming_language)), accessor methods in Ruby are created with a single line of code via [metaprogramming](http://en.wikipedia.org/wiki/Metaprogramming" \o "Metaprogramming). As invocation of these methods does not require the use of parentheses, it is trivial to change an instance variable into a full function, without modifying a single line of code or having to do any refactoring achieving similar functionality to [C#](http://en.wikipedia.org/wiki/C_Sharp_(programming_language)) and [VB.NET](http://en.wikipedia.org/wiki/VB.NET) property members.

Python's property descriptors are similar, but come with a tradeoff in the development process. If one begins in Python by using a publicly exposed instance variable, and later changes the implementation to use a private instance variable exposed through a property descriptor, code internal to the class may need to be adjusted to use the private variable rather than the public property. Ruby’s design forces all instance variables to be private, but also provides a simple way to declare set and get methods. This is in keeping with the idea that in Ruby, one never directly accesses the internal members of a class from outside of it; rather, one passes a message to the class and receives a response.

## Deviations from behavior elsewhere

# Some features which differ notably from languages such as [C](http://en.wikipedia.org/wiki/C_(programming_language)) or [Perl](http://en.wikipedia.org/wiki/Perl):

* The language [syntax](http://en.wikipedia.org/wiki/Syntax_(programming_languages)) is sensitive to the capitalization of identifiers, in most cases treating capitalized variables as constants.
* The [sigils](http://en.wikipedia.org/wiki/Sigil_(computer_programming)) $ and @ do not indicate variable [data type](http://en.wikipedia.org/wiki/Data_type) as in Perl, but rather function as [scope resolution operators](http://en.wikipedia.org/wiki/Scope_resolution_operator).
* To denote a floating point without a decimal component, one must follow with a zero digit (99.0) or an explicit conversion (99.to\_f). It is insufficient to append a dot (99.) since numbers are susceptible to method syntax.
* [Boolean](http://en.wikipedia.org/wiki/Boolean_datatype) evaluation of non-boolean data is strict: 0, "" and [] are all evaluated to *true*. In [C](http://en.wikipedia.org/wiki/C_(programming_language)), the expression 0 ? 1 : 0 evaluates to 0 (i.e. false). In Ruby, however, it yields 1, as all numbers evaluate to true; only nil andfalse evaluate to *false*. A [corollary](http://en.wikipedia.org/wiki/Corollary) to this rule is that Ruby methods by convention — for example, [regular-expression](http://en.wikipedia.org/wiki/Regular_expression) searches — return numbers, strings, lists, or other non-false values on success, but nil on failure. This convention is also used in Smalltalk, where only the special objects true and false can be used in a boolean expression.
* Versions prior to 1.9 use plain integers to represent single characters, much like [C](http://en.wikipedia.org/wiki/C_(programming_language)). This may cause surprises when slicing strings: "abc"[0] yields 97 (the [ASCII](http://en.wikipedia.org/wiki/ASCII) code of the first character in the string); to obtain "a" use"abc"[0,1] (a substring of length 1) or "abc"[0].chr.
* The notation statement until expression, like Perl but unlike other languages' equivalent statements (e.g. do { statement } while (!(expression)); in [C](http://en.wikipedia.org/wiki/C_(programming_language))/[C++](http://en.wikipedia.org/wiki/C%2B%2B)/...), actually never runs the statement if the expression is already true. This is because statement until expression is actually [syntactic sugar](http://en.wikipedia.org/wiki/Syntactic_sugar) over until expression; statement; end, the equivalent of which in [C](http://en.wikipedia.org/wiki/C_(programming_language))/[C++](http://en.wikipedia.org/wiki/C%2B%2B) is while (!(expression)) { statement; }, just as statement if expression is equivalent to if (expression) { statement; }. However, the notation begin statement end until expression in Ruby will in fact run the statement once even if the expression is already true, acting similar to the "do-while" of other languages. (Matz has expressed a desire to remove the special behavior of begin statement end until expression,[[30]](http://en.wikipedia.org/wiki/Ruby_(programming_language)" \l "cite_note-29) but it still exists as of ruby 1.9.)
* Because constants are references to objects, changing what a constant refers to generates a warning, but modifying the object itself does not. For example, Greeting << " world!" if Greeting == "Hello"does not generate an error or warning. This is similar to final variables in Java or a const pointer to a non-const object in [C++](http://en.wikipedia.org/wiki/C%2B%2B), but Ruby provides the functionality to "freeze" an object, unlike [Java](http://en.wikipedia.org/wiki/Java_(programming_language)).

# Some features which differ notably from other languages:

* The usual operators for conditional expressions, *and* and *or*, do not follow the normal rules of precedence: *and* does not bind tighter than *or*. Ruby also has expression operators || and && which work as expected.

A list of so-called [gotchas](http://en.wikipedia.org/wiki/Gotcha_(programming)" \o "Gotcha (programming)) may be found in Hal Fulton's book *The Ruby Way*, 2nd ed ([ISBN 0-672-32884-4](http://en.wikipedia.org/wiki/Special:BookSources/0672328844)), Section 1.5. A similar list in the 1st edition pertained to an older version of Ruby (version 1.6), some problems of which have been fixed in the meantime. retry, for example, now works with while, until, and for, as well as iterators.

## Interaction

The Ruby official distribution also includes "irb", an interactive command-line interpreter which can be used to test code quickly.

## Examples

The following examples can be run in a Ruby shell such as [Interactive Ruby Shell](http://en.wikipedia.org/wiki/Interactive_Ruby_Shell) or saved in a file and run from the command line by typing ruby <filename>.

# Strings

There are a variety of methods for defining strings in Ruby.

The following assignments are equivalent and support [Variable interpolation](http://en.wikipedia.org/wiki/Variable_interpolation):

a = "\nThis is a double-quoted string\n"

a = **%**Q**{**\nThis is a double**-**quoted **string**\n**}**

a = **%{**\nThis is a double**-**quoted **string**\n**}**

a = **%/**\nThis is a double**-**quoted **string**\n**/**

a = **<<-**BLOCK

This is a double**-**quoted **string**

BLOCK

The following assignments are equivalent and produce [raw strings](http://en.wikipedia.org/wiki/Raw_string):

a = 'This is a single-quoted string'

a = **%**q**{**This is a single**-**quoted **string}**

# Collections

Constructing and using an [array](http://en.wikipedia.org/wiki/Array_data_type):

a = **[**1, 'hi', 3.14, 1, 2, **[**4, 5**]]**

**puts** a**[**2**]** *# 3.14*

**puts** a.**[](**2**)** *# 3.14*

**puts** a.reverse *# [[4, 5], 2, 1, 3.14, 'hi', 1]*

**puts** a.flatten.uniq *# [1, 'hi', 3.14, 2, 4, 5]*

Constructing and using an [associative array](http://en.wikipedia.org/wiki/Associative_array) (called hashes in Ruby):

hash = **{** **:water** **=>** 'wet', **:fire** **=>** 'hot' **}**

**puts** hash**[:fire]** *# Prints: hot*

hash.each\_pair **do** **|**key, value**|** *# Or: hash.each do |key, value|*

**puts** "#{key} is #{value}"

**end**

*# Prints: water is wet*

*# fire is hot*

hash.delete **:water** *# Deletes :water => 'wet'*

hash.delete\_if **{|**key,value**|** value=='hot'**}** *# Deletes :fire => 'hot'*

# Blocks and iterators

* + The two syntaxes for creating a code block:

When a code block is created it is always attached to a method as an optional block argument.

* + Parameter-passing a block to be a [closure](http://en.wikipedia.org/wiki/Closure_(computer_science)):
  + Creating an [anonymous function](http://en.wikipedia.org/wiki/Anonymous_function):
  + Returning [closures](http://en.wikipedia.org/wiki/Closure_(computer_science)) from a method:
  + Yielding the flow of program control to a block which was provided at calling time:
  + Iterating over enumerations and arrays using blocks:
  + A method such as inject() can accept both a parameter and a block. Inject iterates over each member of a list, performing some function on it while retaining an aggregate. This is analogous to the [foldl](http://en.wikipedia.org/wiki/Foldl" \o "Foldl) function in [functional programming languages](http://en.wikipedia.org/wiki/Functional_programming_languages).

On the first pass, the block receives 10 (the argument to inject) as sum, and 1 (the first element of the array) as element; this returns 11. 11 then becomes sum on the next pass, which is added to 3 to get 14. 14 is then added to 5, to finally return 19.

Blocks work with many built-in methods:

Using an enumeration and a block to square the numbers 1 to 10 (using a range):

# Classes

The following code defines a class named Person. In addition to 'initialize', the usual constructor to create new objects, it has two methods: one to override the <=> comparison operator (so Array#sort can sort by age) and the other to override the to\_s method (so Kernel#puts can format its output). Here, "attr\_reader" is an example of metaprogramming in Ruby: "attr\_accessor" defines getter and setter methods of instance variables, "attr\_reader" only getter methods. Also, the last evaluated statement in a method is its return value, allowing the omission of an explicit 'return'.

# Open classes

In Ruby, classes are never closed: you can always add methods to an existing class. This applies to the classes you write as well as the standard, built-in classes. All you have to do is open up a class definition for an existing class, and the new contents you specify will be added to whatever's there.

Adding methods to previously defined classes is often called [monkey-patching](http://en.wikipedia.org/wiki/Monkey_patch). This practice, however, can lead to possible collisions of behavior and subsequent unexpected results, and is a concern for code scalability if performed recklessly.

# Exceptions

An exception is raised with a raise call:

**raise**

An optional message can be added to the exception:

**raise** "This is a message"

You can also specify which type of exception you want to raise:

**raise** **ArgumentError**, "Illegal arguments!"

Alternatively, you can pass an exception instance to the raise method:

**raise** **ArgumentError**.new**(**"Illegal arguments!"**)**

This last construct is useful when you need to raise a custom exception class featuring a constructor which takes more than one argument:

**class** ParseError **<** **Exception**

**def** initialize input, line, pos

**super** "Could not parse '#{input}' at line #{line}, position #{pos}"

**end**

**end**

**raise** ParseError.new**(**"Foo", 3, 9**)**

Exceptions are handled by the rescue clause. Such a clause can catch exceptions which inherit from StandardError. Also supported for use with exceptions are else and ensure

It is a common mistake to attempt to catch all exceptions with a simple rescue clause. To catch all exceptions one must write:

**begin**

*# Do something*

**rescue** **Exception**

*# don't write just rescue -- that only catches StandardError, a subclass of Exception*

*# Handle exception*

**end**

Or catch particular exceptions:

**begin**

*# ...*

**rescue** **RuntimeError**

*# handling*

**end**

It is also possible to specify that the exception object be made available to the handler clause:

**begin**

*# ...*

**rescue** **RuntimeError** **=>** e

*# handling, possibly involving e, such as "print e.to\_s"*

**end**

Alternatively, the most recent exception is stored in the magic global $!.

You can also catch several exceptions:

**begin**

*# ...*

**rescue** **RuntimeError**, **Timeout::Error** **=>** e

*# handling, possibly involving e*

**end**

# Metaprogramming

Ruby code can programmatically modify, at [runtime](http://en.wikipedia.org/wiki/Run_time_(program_lifecycle_phase)), aspects of its own structure that would be fixed in more rigid languages, such as class and method definitions. This sort of [metaprogramming](http://en.wikipedia.org/wiki/Metaprogramming" \o "Metaprogramming) can be used to write more concise code and effectively extend the language.

For example, the following Ruby code generates new methods for the built-in String class, based on a list of colors. The methods wrap the contents of the string with an HTML tag styled with the respective color.

COLORS = **{** **:black** **=>** "000",

**:red** **=>** "f00",

**:green** **=>** "0f0",

**:yellow** **=>** "ff0",

**:blue** **=>** "00f",

**:magenta** **=>** "f0f",

**:cyan** **=>** "0ff",

**:white** **=>** "fff" **}**

**class** **String**

COLORS.each **do** **|**color,code**|**

define\_method "in\_#{color}" **do**

"<span style=\"color: ##{code}\">#{self}</span>"

**end**

**end**

**end**

The generated methods could then be used like so:

"Hello, World!".in\_blue

**=>** "<span style=\"color: #00f\">Hello, World!</span>"

To implement the equivalent in many other languages, the programmer would have to write each method (in\_black, in\_red, in\_green, etc.) by hand.

Some other possible uses for Ruby metaprogramming include:

* intercepting and modifying method calls
* implementing new inheritance models
* dynamically generating classes from parameters
* automatic object serialization
* interactive help and debugging

## Implementations

The newest version of Ruby, the recently released version 1.9, has a single working implementation written in C that utilizes a Ruby-specific virtual machine.

Ruby version 1.8 has three main implementations: the official Ruby [interpreter](http://en.wikipedia.org/wiki/Interpreter_(computer_software)) often referred to as the [Matz's Ruby Interpreter](http://en.wikipedia.org/wiki/Ruby_MRI" \o "Ruby MRI) or MRI, which is the most widely used; [JRuby](http://en.wikipedia.org/wiki/JRuby" \o "JRuby), a [Java](http://en.wikipedia.org/wiki/Java_(programming_language))-based implementation that runs on the [Java Virtual Machine](http://en.wikipedia.org/wiki/Java_Virtual_Machine); and [Rubinius](http://en.wikipedia.org/wiki/Rubinius" \o "Rubinius), a reimplementation of Ruby focusing on writing as much of the core in Ruby as possible.

There are other less-known or upcoming implementations such as Cardinal (an implementation for the [Parrot virtual machine](http://en.wikipedia.org/wiki/Parrot_virtual_machine)), [IronRuby](http://en.wikipedia.org/wiki/IronRuby" \o "IronRuby) ([alpha](http://en.wikipedia.org/wiki/Development_stage#Alpha) version available since July 24, 2008),[[31]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-30) [MacRuby](http://en.wikipedia.org/wiki/MacRuby" \o "MacRuby), [MagLev](http://en.wikipedia.org/w/index.php?title=MagLev_(Ruby_interpreter)&action=edit&redlink=1" \o "MagLev (Ruby interpreter) (page does not exist)), [Ruby.NET](http://rubydotnetcompiler.googlecode.com/), [XRuby](http://xruby.com/)and [HotRuby](http://en.wikipedia.org/wiki/HotRuby" \o "HotRuby) (runs Ruby source code on a web browser and Flash).

The maturity of Ruby implementations tends to be measured by their ability to run the [Ruby on Rails](http://en.wikipedia.org/wiki/Ruby_on_Rails) (Rails) framework, because it is a complex framework to implement, and it uses many Ruby-specific features. The point when a particular implementation achieves this goal is called *The Rails singularity*. The [reference implementation (MRI)](http://en.wikipedia.org/wiki/Ruby_MRI), [JRuby](http://en.wikipedia.org/wiki/JRuby" \o "JRuby), and [Rubinius](http://en.wikipedia.org/wiki/Rubinius" \o "Rubinius) are all able to run Rails unmodified in a production environment. [IronRuby](http://en.wikipedia.org/wiki/IronRuby" \o "IronRuby)[[33]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-32)[[34]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-33) is starting to be able to run Rails test cases, but is still far from being production-ready.

Ruby is available on many [operating systems](http://en.wikipedia.org/wiki/Operating_system), such as [Linux](http://en.wikipedia.org/wiki/Linux), [Mac OS X](http://en.wikipedia.org/wiki/Mac_OS_X), [Microsoft Windows](http://en.wikipedia.org/wiki/Microsoft_Windows), [Windows Phone 7](http://en.wikipedia.org/wiki/Windows_Phone_7),[[35]](http://en.wikipedia.org/wiki/Ruby_(programming_language)#cite_note-34) [Windows CE](http://en.wikipedia.org/wiki/Windows_CE) and most flavors of [Unix](http://en.wikipedia.org/wiki/Unix).

Ruby 1.9 has recently been ported onto [Symbian OS](http://en.wikipedia.org/wiki/Symbian_OS" \o "Symbian OS) 9.x.

## Repositories and libraries

The [Ruby Application Archive](http://en.wikipedia.org/wiki/Ruby_Application_Archive) (RAA), as well as [RubyForge](http://en.wikipedia.org/wiki/RubyForge" \o "RubyForge), serve as repositories for a wide range of Ruby applications and libraries, containing more than seven thousand items. Although the number of applications available does not match the volume of material available in the [Perl](http://en.wikipedia.org/wiki/Perl) or [Python](http://en.wikipedia.org/wiki/Python_(programming_language)) community, there are a wide range of tools and utilities which serve to foster further development in the language.

[RubyGems](http://en.wikipedia.org/wiki/RubyGems) has become the standard package manager for Ruby libraries. It is very similar in purpose to Perl's [CPAN](http://en.wikipedia.org/wiki/CPAN), although its usage is more like [apt-get](http://en.wikipedia.org/wiki/Apt-get).

Recently, many new and existing libraries have found a home on [GitHub](http://en.wikipedia.org/wiki/GitHub" \o "GitHub), which is focused on [Git](http://en.wikipedia.org/wiki/Git_(software)" \o "Git (software)).