

Ruby

a focal comparison report

Catherine Austria | COSC450 | Fall 2016

# A Brief History of the Language [8]

# Ruby was created in 1993 by Yukihiro Matsumoto who discussed with his colleague the possibility of a truly object-oriented scripting-language. The first public version of Ruby (version 0.95) was released in December of 1995. In 2005 Ruby took popularity due to Ruby on Rails, which implemented a Model-View-Controller and paved the way for rapid web development.

# Language Data Types [7]

# Ruby supports the following data types an example of each are given as well:

## Booleans

In Ruby Boolean data can either be true or false.

Ex. bool = [true, false]

isTrue = bool[rand(2)]

if isTrue

puts “True”

else

puts “False”

## Symbols

Symbols are used to represent other objects and using them instead of strings may save some resources. (see source) A colon (:) before an identifier is used to generate them or some objects have a “to\_sym” method that convert those objects to symbols.

Ex. :name

:name.class

## Numbers

Ruby supports

1. Integers (ex. -2)
2. Floating Point (ex. 0.3455)
3. Rational Numbers (ex. 2.6.to\_r; where to\_r is a method used to convert to a rational number)
4. nil Value (ex. nil; this special feature denotes the absence of a value)

## Strings

In Ruby Strings is a sequence of Unicode characters that are used to represent the text data in a program. Strings are enclosed using double or single quotes. (ex. “Ruby” , ‘Ruby’)

## Arrays

This represent ordered collections of objects in Ruby. An example is a collection of integer numbers as seen below:

Ex. numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]

## Hashes

This represent collection of key-value pairs in Ruby. An example is the shorthand letters representing US states to their textual counterparts as seen below:

Ex. states = { :md => “Germany”, :pa => “Pensylvania” , :wa => “Washington” }

# Flow Control

Below are a list of the different flow control mechanisms of the language:

## Comparison Operators

There are basically used for conditional logic. The types are seen below listed as from left to right, equal to, less than, greater than, greater than or equal to, less than or equal to and not equal to:

Ex. ==, <, >, >=, <=, !=

## Spaceship Operator

This special operator gives three different possible output depending on whether the left side is greater than, less than, or equal to the right side. An example is provided with the operator below:

Operator: <=>

Ex.

> 1 <=> 1000

=> -1 #result is less than

> 1 <=> 1

=> 0 #result is equal

> 1 <=> -1000

=> 1 #result is greater than

## Logical Operators

These allows comparison of statements in Ruby. These are mainly used inside if conditional statements and loop conditions. It is good to note that Ruby will only evaluate an expression only till a point where the expression is definitively true or false and will disregard the rest.

* AND (&&) – where both sides must be true to for the code within to execute or process.
* OR (||) – where only one side, either left or right of the operator, is required to be true for the code within to execute or process.
* NOT (!) – reverses the statement from true to false and false to true.

# Subroutines

A subroutine starts with def and ends with a corresponding end. Subroutines pass back values with the return keyword.

Ex. def greeting

greetWith = "Hello World!"

return greetWith

end

someString = greeting

puts someString #prints “Hello World!”

# User Defined Types

In Ruby you can create a class for holding data like the example below:

Ex. class Cat < Struct.new(:name, :furLength, :breed)

end

However, a user cannot declare the class of a variable or method argument, like in C. To remedy this, a user can instead check for the type at run time.

Ex. animal.is\_a?(Cat) # This will go through the subclasses of Cat as well

animal.class == Cat

# Program File Organization [4]

A ruby gem usually has the following major components :

* lib – is the directory which contains the code for the gem.
* test or spec – is the directory which contains tests, depending on the framework used and needed by the gem.
* Rakefile – is used by the rake program to generate and automate tests and perform other tasks.
* bin – is an optional directory which contains an executable file that can be loaded to a users’s PATH when the gem is installed.
* README – is the documentation for the installed gem. Ruby often generates this automatically upon gem installation.
* gemspec – contains the test information, gem version and other gem related information like author contact details and etc.

# Generics

# Ruby doesn’t have generics. Afterall, there's no explicit typing anywhere, so the concept of `generic' makes no sense. In Ruby, everything is an object! That what it makes it a true object-oriented language.

# Concurrency [3]

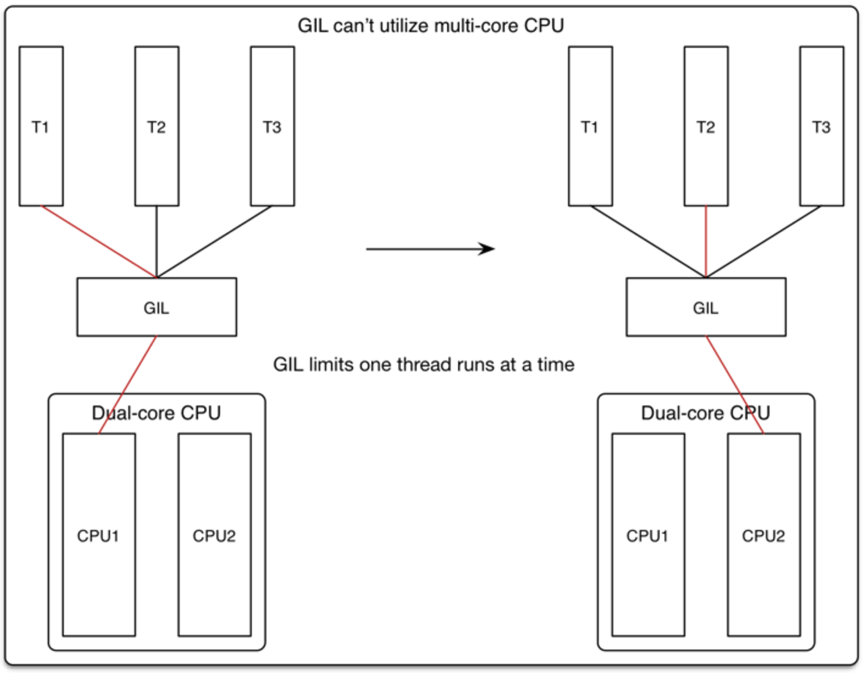


Fig. 1: GIL in Ruby

## Threads

Ruby uses a Global Interpreter Lock (GIL) which acts as a locking mechanism to protect data integrity. This allows for data to only be manipulated by one thread at a time. This makes it so that data isn’t corruptible by multiple threads running at the same time but also in a way doesn’t make the Ruby run threads truly concurrently. Although this may seem disadvantageous, GIL makes single-threaded programs faster. It is also worth noting that there are several Ruby implementations that allow for multiple threads, this comes at the cost of having to deal with the problem of synchronization and thread-handling to prevent deadlock.

## Multiprocessing and Forking

This solution adheres to the fact that since the language by default isn’t capable of true concurrency, starting up new processes is instead used as the common solution. This however brings problems of using too much memory due to the large amount of processes running, each using their own memory. As an added solution to this, forking may be applied to provide a middle ground solution for unix environments. Forking entails creating a copy of the main process but instead of multiprocessing, these processes share the same physical memory. This is good to note because despite sharing memory, forking processes allows each individual process capable of modifying its own memory without affecting other processes. If a forked process begins to misbehave, the developer can simply opt to end it and start a new forked process again from the main process.

## Actors

This concept is similar to threads in that it can run concurrently with other actors. Unlike threads though, actors are not pre-emptive, meaning it doesn’t share the same memory context. The message-passing between actors are handled a bit similarly to a mailbox in which a routine is called (“receive”) that checks for new messages. The routine takes a filter and if none matches the filter, it sleeps until it receives a match. After the match is received the actor is then rescheduled for execution. Message-passing is handled in the following way for actors:

1. Send a finite number of messages to other actors.
2. Spawn a finite number of new actors.
3. Change its own internal behavior, taking effect when the next incoming message is handled.

## Fibers

Ever since Ruby 1.9 a new lightweight type of thread was introduced which is called “fibers”. It’s main difference is that fibers are simple threads that are scheduled not by the virtual environment but by the developer. Advantages of this over threads is that it uses less memory, performs faster and can be paused and resumed from within or outside of the fiber. GIL limits the ability to run more than one fiber at a time, however fibers still help with concurrency by the fact that the developer is given the right to schedule the fiber to control the flow of traffic in and out of the fiber. This is true even to the extent of having the ability to implement the fiber to auto-schedule itself. Blocking IO operations may present a problem for fibers since the fiber becomes blocked and thusly blocks other fibers, but a shorthand workaround to this is to avoid designing the fiber to block IO operations.

# Comparison of Ruby vs. C [8]

## Similarities of both languages

* Operators are the same except for Ruby it does not have increment (++) and decrement(--)
* Procedural programming can still be done; however it is still object-oriented in the background.
* Constants can be used. For C, it is const and for Ruby, the naming convention of all capital letters is used.
* Strings are in double-quotes and are mutable.
* Same kind of cmd line debugger is used.
* The similar syntax of \_\_FILE\_\_ and \_\_LINE\_\_.

## Differences in Ruby from C

* Ruby doesn’t need to compile code.
* Objects are strongly typed.
* No variable declarations. Assigned values can be assigned to directly to new variables.
* There is no casting, typedefs, sizeof, semicolons, enums and parenthesis for if and while expressions.
* Constants replace #define
* No header files.
* All variables are in a heap implementation and the garbage collector is tasked to free them instead of manually having to do it yourself like in C.
* Arguments to methods / functions are passed by value, where values are always object references.
* When tested for truth, only false and nil evaluate to a false value. Everything else is true.
* No char type, just 1 letter strings and Strings don’t end in a null byte.
* Array literals go in brackets instead of braces and arrays just automatically get bigger when adding elements to it. Adding two arrays thusly automatically returns a new bigger array containing the elements of both arrays.

# Summary of Defining Features [8]

## Syntactically Simple

* Since it is an interpreted language, dynamic, no parenthesis’ and no need for variable declarations, the code using the language can easily be typed fast.

## Truly Object-Oriented

* As explained previously, in Ruby everything is an object [7].

## Having Iterators and Closures

* Iterators are easy and instead of using for loops, Ruby uses each. An array of elements can even be directly assigned from the loop expression using map [6].

Ex. varOfSomething = someArray.map do |s|

## Exception Handling

* Ruby gives the developer a vast amount of ways to handle exceptions [1].

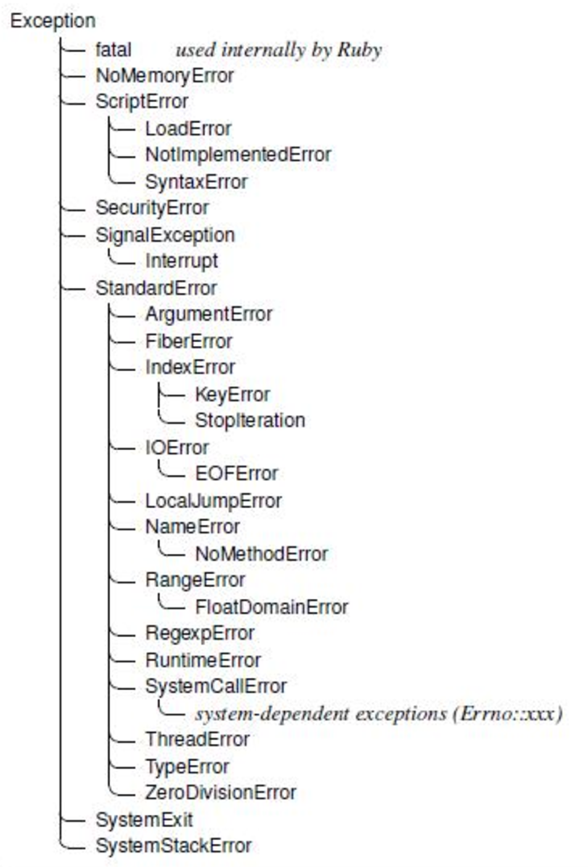


Fig. 2: Exception Hierarchy in Ruby [1]

## Garbage Collection

* The garbage collection automatically handles the freeing of data from the heap [2].

## Portability

* Ruby is easily best known for Ruby on Rails. Rails allows for speedy web development [2].

# Resources

[1] E. Aligam, "Ruby Exceptions", Rubylearning.com, 2016. [Online]. Available: http://rubylearning.com/satishtalim/ruby\_exceptions.html. [Accessed: 07- Dec- 2016].

[2] "To Ruby From C and C++", Ruby-lang.org, 2016. [Online]. Available: https://www.ruby-lang.org/en/documentation/ruby-from-other-languages/to-ruby-from-c-and-cpp/. [Accessed: 07- Dec- 2016].

[3] "Ruby Concurrency In Theory - Wendi's Blog", Blog.ifyouseewendy.com, 2016. [Online]. Available: http://blog.ifyouseewendy.com/blog/2016/02/16/ruby-concurrency-in-theory/. [Accessed: 07- Dec- 2016].

[4] G. Sawyer, "What is a gem? - RubyGems Guides", Guides.rubygems.org, 2016. [Online]. Available: http://guides.rubygems.org/what-is-a-gem/. [Accessed: 07- Dec- 2016].

[5] S. Litt, "Ruby Basic Tutorial", Troubleshooters.com, 2016. [Online]. Available: http://www.troubleshooters.com/codecorn/ruby/basictutorial.htm#\_Subroutines. [Accessed: 07- Dec- 2016].

[6] E. Trautman, "Erik Trautman's world of startups, creativity and code", Eriktrautman.com, 2016. [Online]. Available: http://www.eriktrautman.com/posts/ruby-explained-conditionals-and-flow-control. [Accessed: 07- Dec- 2016].

[7] J. Bodnar, "Ruby data types", Zetcode.com, 2016. [Online]. Available: http://zetcode.com/lang/rubytutorial/datatypes/. [Accessed: 07- Dec- 2016].

[8] J. Herrick, "The History of Ruby", SitePoint, 2016. [Online]. Available: https://www.sitepoint.com/history-ruby/. [Accessed: 07- Dec- 2016].