instituto tecnológico de veracruz

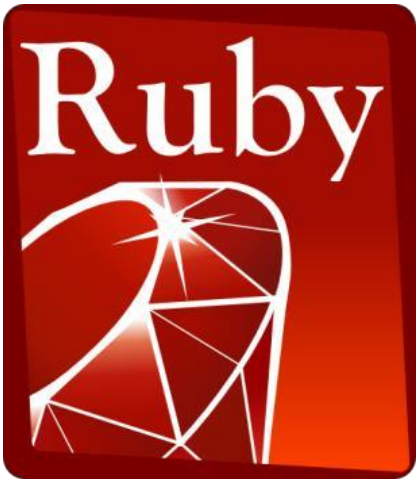
Documentación Ruby (LyA I)

Catedrático: Martha

Por: Ricardo David Rojas Flores
Jazmín Arlette Carrasco Cruz
Ernesto Álvarez Calderón

INDICE

INTRODUCCIÓN



Es un lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por el programador japonés Yukihiro "Matz" Matsumoto, quien comenzó a trabajar en Ruby en 1993, y lo presentó públicamente en 1995.

El creador del lenguaje ha dicho que Ruby está diseñado para la productividad y la diversión del desarrollador, siguiendo los principios de una buena interfaz de usuario. Sostiene que el diseño de sistemas necesita enfatizar las necesidades humanas más que las de la máquina:[2]

A menudo la gente, especialmente los ingenieros en computación, se centran en las máquinas. Ellos piensan, "Haciendo esto, la máquina funcionará más rápido. Haciendo esto, la máquina funcionará de manera más eficiente. Haciendo esto..." Están centrados en las máquinas, pero en realidad necesitamos centrarnos en las personas, en cómo hacen programas o cómo manejan las aplicaciones en los ordenadores. Nosotros somos los jefes. Ellos son los esclavos.

Ruby sigue el *principio de la menor sorpresa*, lo que significa que el lenguaje debe comportarse de tal manera que minimice la confusión de los usuarios experimentados. Matz ha dicho que su principal objetivo era hacer un lenguaje que le divirtiera a él mismo, minimizando el trabajo de programación y la posible confusión. Él ha dicho que no ha aplicado el principio de menor sorpresa al diseño de Ruby,[2] sin embargo, la frase se ha asociado al lenguaje de programación Ruby. La frase en sí misma ha sido fuente de controversia, ya que los no iniciados pueden tomarla como que las características de Ruby intentan ser similares a las características de otros lenguajes conocidos. En mayo de 2005 en una discusión en el grupo de noticias comp.lang.ruby, Matz trató de distanciar Ruby de la mencionada filosofía, explicando que cualquier elección de diseño será sorprendente para alguien, y que él usa un estándar personal de evaluación de la sorpresa. Si ese estándar personal se mantiene consistente habrá pocas sorpresas para aquellos familiarizados con el estándar.

Matz lo definió de esta manera en una entrevista:

Todo el mundo tiene un pasado personal. Alguien puede venir de Python, otro de Perl, y pueden verse sorprendidos por distintos aspectos del lenguaje. Entonces podrían decir 'Estoy sorprendido por esta característica del lenguaje, así que Ruby viola el principio de la menor sorpresa.' Espera, espera. El principio de la menor sorpresa no es sólo para ti. El principio de la menor sorpresa significa el principio de 'mi' menor sorpresa. Y significa el principio de la menor sorpresa después de que aprendes bien Ruby. Por ejemplo, fui programador de C++ antes de empezar a diseñar Ruby. Programé exclusivamente en C++ durante dos o tres años. Y después de dos años de programar en C++, todavía me sorprendía.

Es indudable que que Ruby es uno de tantos lenguajes contemporáneos que resulta simples, elegantes y funcionales. Resultado de una amalgama de varios lenguajes que inspiraron al creador: Perl, Lisp y Python, este último conocido por su rapidez de ejecución, Ruby nos presenta un conjunto de funciones que el creador creyó las mejores y las implementó en un nuevo lenguaje.

Ruby se ha desarrollado estos últimos 6 años, tan significativo ha sido su avance, que incluso es utilizado para servidores gracias al framework basado en Ruby llamado **Ruby on Rails**. Es indudable declarar que Ruby posee un potencial a ser explotado en un futuro próximo. En cuanto a la selección de dicho lenguaje para nuestra asignatura de Lenguajes y Autómatas II, surge como necesidad de explorar nuevos paradigmas en el ámbito de la programación y empaparnos de tantos lenguajes como sea posible—desarrollando en el camino nuevas habilidades que nos harán estudiantes y futuros ingenieros más competentes.

ANTECEDENTES

Ruby fue creado en Japón por Yukihiro Matsumoto mientras trabajaba como programador con lenguajes como Perl y PHP.
Según el creador de Ruby, el nombre del lenguaje lo decidió en honor a un colega suyo a partir de la piedra correspondiente a su mes de nacimiento. También se dice que es un juego de palabras relacionado con el lenguaje Perl, ya que en un principio su idea era solo crear una versión mejorada del lenguaje.
En un principio, su intención fue crear Perl avanzado, para mejorar algunos detalles de dicho lenguaje, pero en lugar de mejorarlo, se vio tentado en crear uno propio a partir de sus lenguajes favoritos: Perl, Smalltalk, Eiffel y Lisp.

La primera versión que se le presentó al público fue la 0.95 (en 1995), más adelante en 1996 se presentó Ruby 1.0 y se les fue ofrecido al público. En el 2000 IBM se interesó en el lenguaje y publico un artículo. En 2004 Rails fue liberado, a partir de eso, Ruby en el 2006 fue considerado entre los primeros 10 más populares.

Ruby-Talk, la lista de correo más importante sobre el lenguaje Ruby ha crecido hasta lograr un promedio de 200 mensajes por día.

El índice TIOBE, que mide el crecimiento de los lenguajes de programación, ubica a Ruby en la posición #13 del ranking mundial. Refiriéndose a su crecimiento, predicen, “Todo indica que Ruby llegará a estar entre los 10 primeros en menos de 6 meses”. Gran parte de su crecimiento se atribuye a la popularidad alcanzada por aplicaciones desarrolladas con Ruby, en particular el framework de desarrollo web Ruby on Rails.

Ruby es totalmente libre. No sólo gratis, sino también libre para usarlo, copiarlo, modificarlo y distribuirlo.

CARACTERÍSTICAS

- Orientado a objetos.
- Cuatro niveles de ámbito de variable: global, clase, instancia y local.
- Manejo de excepciones.
- iteradores y clausuras o closures (pasando bloques de código).
- expresiones regulares nativas similares a las de Perl a nivel del lenguaje.
- Posibilidad de redefinir los operadores (sobrecarga de operadores).

- recolección de basura automática.
- Altamente portable.
- Hilos de ejecución simultáneos en todas las plataformas usando hilos verdes, o no gestionados por el sistema operativo.
- Carga dinámica de DLL/bibliotecas compartidas en la mayoría de las plataformas.
- Introspección, reflexión y metaprogramación.
- Amplia librería estándar.
- Soporta inyección de dependencias.
- Soporta alteración de objetos en tiempo de ejecución.
- continuaciones y generadores.

Viendo todo como un objeto

Inicialmente, Matz buscó en otros lenguajes para encontrar la sintaxis ideal. Recordando su búsqueda, dijo, "quería un lenguaje que fuera más poderoso que Perl, y más orientado a objetos que Python".

En Ruby, todo es un objeto. Se le puede asignar propiedades y acciones a toda información y código. La programación orientada a objetos llama a las propiedades *variables de instancia* y las acciones son conocidas como *métodos*. La orientación a objetos pura de Ruby se suele demostrar con un simple código que aplica una acción a un número.

```
5.times { print "Nos *encanta* Ruby -- ¡es fuera de serie!" }
```

En muchos lenguajes, los números y otros tipos primitivos no son objetos. Ruby sigue la influencia del lenguaje Smalltalk pudiendo poner métodos y variables de instancia a todos sus tipos de datos. Esto facilita el uso de Ruby, porque las reglas que se aplican a los objetos son aplicables a todo Ruby.

La flexibilidad de Ruby

Ruby es considerado un lenguaje flexible, ya que permite a sus usuarios alterarlo libremente. Las partes esenciales de Ruby pueden ser quitadas o redefinidas a placer. Se puede agregar funcionalidad a partes ya existentes. Ruby intenta no restringir al desarrollador.

Por ejemplo, la suma se realiza con el operador suma (+). Pero si prefieres usar la palabra *sumar*, puedes agregar un método llamado sumar a la clase Numeric que viene incorporada.

```
class Numeric
  def sumar(x)
    self.+(x)
  end
end
y = 5.sumar 6
# ahora y vale 11
```

Los operadores de Ruby son simples conveniencias sintácticas para los métodos. Los puedes redefinir como y cuando quieras.

Los Bloques, una funcionalidad realmente expresiva

Los bloques de Ruby son también vistos como una fuente de gran flexibilidad. El desarrollador puede anexar una cláusula a cualquier método, describiendo cómo debe actuar. La cláusula es llamada *bloque* y se ha convertido en una de las más famosas funcionalidades para los recién llegados a Ruby que vienen de otros lenguajes imperativos como PHP o Visual Basic.

Los bloques están inspirados por los lenguajes funcionales. Matz dijo, "en las cláusulas de Ruby, quise respetar la cultura de Lisp".

```
motores_de_busqueda =
  %w[Google Yahoo MSN].map do |motor|
    "http://www." + motor.downcase + ".com"
  end
```

En este código, el bloque está descrito entre la construcción do ... end. El método map aplica el bloque a la lista de palabras provista. Muchos otros métodos en Ruby dejan abierta la posibilidad al desarrollador para que escriba su propio bloque describiendo los detalles de qué debe hacer ese método.

La apariencia visual de Ruby

A pesar de que Ruby utiliza la puntuación muy limitadamente y se prefieren las palabras clave en inglés, se utiliza algo de puntuación para decorar el código. Ruby no necesita declaraciones de variables. Se utilizan convenciones simples para nombrar y determinar el alcance de las mismas.

- var puede ser una variable local.
- @var es una variable de instancia.
- $var es una variable global.

Estos detalles mejoran la legibilidad permitiendo que el desarrollador identifique fácilmente los roles de las variables. También se hace innecesario el uso del molesto self. como prefijo de todos los miembros de instancia.

Más allá de lo básico

Ruby tiene un conjunto de otras funcionalidades entre las que se encuentran las siguientes:

- Manejo de excepciones, como Java y Python, para facilitar el manejo de errores.
- Un verdadero mark-and-sweep garbage collector para todos los objetos de Ruby. No es necesario mantener contadores de referencias en bibliotecas externas. Como dice Matz, "Esto es mejor para tu salud".
- Escribir extensiones en C para Ruby es más fácil que hacer lo mismo para Perl o Python, con una API muy elegante para utilizar Ruby desde C. Esto incluye llamadas para embeber Ruby en otros programas, y así usarlo como lenguaje de scripting. También está disponible una interfaz SWIG.
- Puede cargar bibliotecas de extensión dinámicamente si lo permite el sistema operativo.
- Tiene manejo de hilos (threading) independiente del sistema operativo. De esta forma, tienes soporte multi-hilo en todas las plataformas en las que corre Ruby, sin importar si el sistema operativo lo soporta o no, ¡incluso en MS-DOS!

- Ruby es fácilmente portable: se desarrolla mayoritariamente en GNU/Linux, pero corre en varios tipos de UNIX, Mac OS X, Windows, DOS, BeOS, OS/2, etc.

SEMÁNTICA

Ruby es orientado a objetos: todos los tipos de datos son un objeto, incluidas las clases y tipos que otros lenguajes definen como primitivas, (como enteros, booleanos, etcétera). Este lenguaje soporta herencia con enlace dinámico, mixins y métodos singleton (pertenecientes y definidos por una sola instancia más que definidos por la clase).
A pesar que Ruby no soporta herencia múltiple, las clases pueden importar módulos como mixins.

Ruby ha sido descrito como un lenguaje de programación multiparadigma:
permite programación procedural, con orientada a objetos o funcionalmente. Además de soporte para hilos de ejecución gestionados por el intérprete.

Este lenguaje tiene tipado dinámico y soporta polimorfismo de tipos (permite tratar a subclases utilizando la interfaz de la clase padre). No requiere polimorfismo de funciones al no ser fuertemente tipado.

SINTAXIS

La sintaxis de Ruby es similar a la de Perl y Python. La definición de clases y métodos está definida por palabras claves. Sin embargo, en Perl, las variables no llevan prefijos. Cuando se usa, un prefijo indica el ámbito de las variables. La mayor diferencia con C y Perl es que las palabras clave son usadas para definir bloques de código sin llaves. Los saltos de línea son significativos y son interpretados como el final de una sentencia; el punto y coma tiene el mismo uso. De forma diferente que Python, la indentación no es significativa

¿POR QUÉ USAR RUBY?

- Ruby es un lenguaje de scripts, moderno y orientado a objetos, que combina una importante flexibilidad con alta productividad
- Su alcance parece ilimitado y hoy se encuentra presente en aplicaciones que van desde el desarrollo web hasta la simulación de ambientes complejos.
- Promueve las mejores prácticas de programación sin perder usabilidad.
- Mediante su uso se pueden complementar las características de la lógica imperativa con la lógica funcional.
- Permite utilizar la más simple expresión para un programa o algoritmo; esto sumado a las actuales prácticas ágiles permite desarrollar en forma amigable.

TOKENS

Los Tokens, también llamados como unidades léxicas, seleccionados para nuestro analizador de Ruby fueron los siguientes:

Token	Tipo
__ENCODING__	PALABRA RESERVADA
__LINE__	
__FILE__	
__BEGIN__	
END	
alias	
and	
begin	
break	
catch	
case	
class	
defined	
def	
do	
dir	
else	
elseif	
end	
ensure	
false	
for	
if	
Initialize	

in
load
module
next
new
nil
not
or
raise
require
redo
rescue
retry
return
self
super
then
true
throw
undef
unless
until
when
while
yield

Token	Tipo
public	Visibilidad
protected	
private	
[]	Corchetes
{}	Llaves
()	Paréntesis

Token	**Tipo**
[0-9]	Número

and	Operador lógico
or	
&&	
|	
||	
!	
not	
<<	Operadores de bits
>>	
&	
|	
^	
~	
=	Asignación
**	Operadores aritméticos
*	
+	
-	
/	
<=>	Operador relacional
===	
>=	
<=	
==	
!=	
.eql?	
equal?	
<	

>	
...	Operadores de rango
..	

Token	Tipo
[A-Z]+	Constante
#	Comentario
"=begin"	
"=end"	
($[a-zA-Z]*)	Variable global
(@@[a-zA-Z]*)	Variable de clase
(@[a-zA-Z]*)	Variable instanciable
(_[a-z][a-zA-Z_]*)	Variable local
([a-z][a-zA-Z_]*)	
	Espacio en blanco
“([\w\s\d]*)”	Literal
([0-9])+\.([0-9])+	Decimal
,	Coma
""	Comilla
^[a-zAZ]	Error
S	
Ç	
C	

EXPRESIONES REGULARES

Las reglas gramaticales que utilizamos fueron las siguientes:

- **Identificador de Clase**: *[A-Z][a-z_]+[A-Z][a-z_]+*
- **Palabra Reservada:**
  "(__ENCODING__|__LINE__|__FILE__|BEGIN|END|alias|and|begin|break|catch|case|class|defined|def|do|dir|else|elsif|end|ensure|false|for|if|initialize|in|load|module|next|new|nil|not|

or|raise|require|redo|rescue|retry|return|self|super|then|true|throw|undef|unless|until|whe n|while|yield)"

- **Operador Aritmético:** *"(**|\\+|-|*|\\/)"*
- **Operador Relacional:** *"(<=>|===|>=|<=|==|!=|\\.eql\\?|equal\\?|>|<)"*
- **Operador de Rango:** *"(\\.\\.\\.|\\.\\.)"*
- **Comentario:** *"(#|=begin\\b|=end\\b)"*
- **Visibilidad:** *"(public|protected|private)"*
- **Variable Global:** *"(\\$[a-zA-Z_]*)"*
- **Variable de Clase:** *"(@@[a-zA-Z_]*)"*
- **Variable Instanciada:** *"(@[a-zA-Z_]*)"*
- **Variable Local:** *"(_[a-z][a-zA-Z_]*)"*
- **Variable Local:** *"([a-z][a-zA-Z_]*)"*
- **Número:** *"([0-9]+)"*
- **Corchetes:** *"(\\[|\\])"*
- **Llaves:** *"(\\{|\\})"*
- **Operador Lógico***: "(and|or|&&|\\||\\||!|not)"*
- **Operador de Bits:** *"(<<|>>|&|\\||\\^|~)"*
- **Paréntesis:** *"(\\(|\\))"*
- **Espacio en blanco:** *"(\\s)+"*
- **Operador de asignación**: *"(=)"*
- **Coma:** *"(,)"*
- **Comilla:** *"(")"*
- **Constante:** *[A-Z]+[0-9]**
- **Error:** *"(\\D|\\S|Ç|ç)"*

EJEMPLOS

```
puts "Hello World"
print 'Enter your name: '
name= gets
puts "Hello #{name}"
http://naupacto.com/rubydoc/guiausuario/c90.htm

class Ave
 def acicala
   puts "Estoy limpiando mis plumas."
 end
 def vuela
   puts "Estoy volando."
 end
end
class Pinguino<Ave
```

```
 def vuela
   puts "Lo siento, prefiero nadar."
 end
end
class Aguila<Ave
end
puts "Pinguino"
p = Pinguino.new
p.acicala
p.vuela
puts "Aguila"
a = Aguila.new
a.acicala
a.vuela
```

De:https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Ruby/Ejemplos_del_lenguaje

CONCLUSIONES

A lo largo del desarrollo de este proyecto nos percatamos de la complejidad del crear desde cero un compilador. Hay mucho trabajo de por medio que en un principio desconocíamos, y hasta cierto punto, nos hace apreciar más los esfuerzos colectivos que nos han significado el nacimiento de los diversos lenguajes de programación.

En términos generales fue una grata experiencia, aunque bastante difícil, pero como todo lo bueno en la vida, requirió un gran esfuerzo. Esperamos poder continuar este proyecto en Lenguajes y Autómatas II.

Reglas Gramaticales

VARIABLE:<VARIABLEINSTANCIADA><ASIGNACION>(<NUMERO>|<CADENA>|<CARACTER>)
<VARIABLEIGLOBAL><ASIGNACION>(<NUMERO>|<CADENA>|<CARACTER>)
<VARIABLELOCAL><ASIGNACION>(<NUMERO>|<CADENA>|<CARACTER>)
OPERACION NUMERICA:<NUMERO><OPERADORARITMETICO><NUMERO>
OPERACION RELACIONAL:<NUMERO><OPERADORRELACIONAL><NUMERO>
DEFINICION DE METODO:
<DEF><IDENTIFICADOR><PARENTIZQ>(<NUMERO>|<CADENA>|<CARACTER>)<PARENTDER><END>
LLAMADA A METODO:
<PUNTO><IDENTIFICADOR><PARENTIZQ>(<NUMERO>|<CADENA>|<CARACTER>)<PARENTDER><END>
CONDICION:
<IF>(<NUMERO>|<VARIABLE>)<OPERADORRELACIONAL>(<NUMERO>|<VARIABLE>)…<ELSEIF><ELSE>…<END>

CICLO WHILE:
<WHILE>(<NUMERO>|<VARIABLE>)<OPERADORRELACIONAL>(<NUMERO>|<VARIABLE>)|<CONDICION><DO>
...<END>
CICLO FOR:<VARIABLE><NUMERO><RANGO><NUMERO><CONDICION><END>

REFERENCIAS:

JavaCC. (2017). The Java Parser Generator. 22 de agosto del 2017, de JavaCC Sitio web: https://javacc.org/

University at Buffalo. (2017). Ruby BNF. 22 de agosto del 2017, de University at Buffalo Sitio web: https://www.cse.buffalo.edu/~regan/cse305/RubyBNF.pdf

tlazaro. (2017). NetBeans. 22 de agosto del 2017, de NetBeans Sitio web: http://plugins.netbeans.org/plugin/20277/javacc

Matz, hablando en la lista de correo Ruby-Talk, 12 de mayo del 2000.

Matz, en An Interview with the Creator of Ruby, 29 de noviembre del 2001.

Matz, en Blocks and Closures in Ruby, 22 de diciembre del 2003.