Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

ALCANCES, LIMITACIONES Y USO A NIVEL NACIONAL E INTERNACIONAL DEL LENGUAJE DE PROGRAMACIÓN RUBY

Eduardo Alejandro Herrera Gutiérrez
Asesorado por el Ing. William Estuardo Escobar Argueta

Guatemala, septiembre de 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ALCANCES, LIMITACIONES Y USO A NIVEL NACIONAL E INTERNACIONAL DEL LENGUAJE DE PROGRAMACIÓN RUBY

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

EDUARDO ALEJANDRO HERRERA GUTIÉRREZ
ASESORADO POR EL ING. WILLIAM ESTUARDO ESCOBAR ARGUETA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, SEPTIEMBRE DE 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Raúl Eduardo Ticún Córdova
VOCAL V	Br. Henry Fernando Duarte García
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. Oscar Alejandro Paz Campos
EXAMINADOR	Ing. William Estuardo Escobar Argueta
EXAMINADOR	Ing. Sergio Arnaldo Méndez Aguilar
SECRETARIO	Ing. Pablo Christian de León Rodríguez (a. i.)

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

ALCANCES, LIMITACIONES Y USO A NIVEL NACIONAL E INTERNACIONAL DEL LENGUAJE DE PROGRAMACIÓN RUBY

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 18 de marzo de 2014.

Eduardo Alejandro Herrera Gutierrez

Guatemala, 6 de junio de 2016

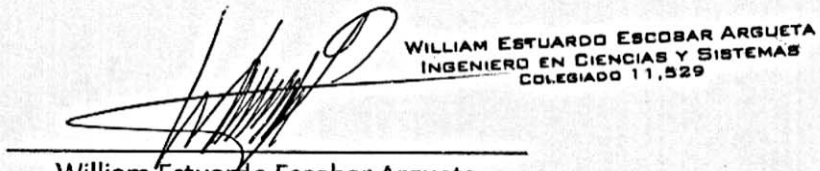
Ingeniero
Carlos Azurdia
Revisor de Trabajo de Graduación
Escuela de Ciencias y Sistemas
Facultad de Ingeniería

Respetable Ingeniero Azurdia

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **Eduardo Alejandro Herrera Gutierrez,** identificado con el número de carné **201020491**, titulado: **"ALCANCES, LIMITACIONES Y USO A NIVEL NACIONAL E INTERNACIONAL DEL LENGUAJE DE PROGRAMACIÓN RUBY"**, y a mi criterio el mismo cumple con los objetivos propuestos para su elaboración de acuerdo al protocolo presentado.

Sin otro particular, me suscribo de usted.

Atentamente,



William Estuardo Escobar Argueta
Ingeniero en Ciencias y Sistemas

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 13 de Julio de 2016

Ingeniero
Marlon Antonio Pérez Türk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **EDUARDO ALEJANDRO HERRERA GUTIERREZ** con carné **201020491**, titulado: **"ALCANCES, LIMITACIONES Y USO A NIVEL NACIONAL E INTERNACIONAL DEL LENGUAJE DE PROGRAMACIÓN RUBY"**, y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
INGENIERIA EN CIENCIAS Y SISTEMAS
COORDINACION TESIS

ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE INGENIERÍA EN
CIENCIAS Y SISTEMAS
TEL: 24767644

*El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación "**ALCANCES, LIMITACIONES Y USO A NIVEL NACIONAL E INTERNACIONAL DEL LENGUAJE DE PROGRAMACIÓN RUBY**", realizado por el estudiante EDUARDO ALEJANDRO HERRERA GUTIÉRREZ aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Antonio Pérez Türk
Director
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 08 de septiembre de 2016

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

Ref.DTG.D.403.2016

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al trabajo de graduación titulado: **ALCANCES, LIMITACIONES Y USO A NIVEL NACIONAL E INTERNACIONAL DEL LENGUAJE DE PROGRAMACIÓN RUBY,** presentado por el estudiante universitario: **Eduardo Alejandro Herrera Gutiérrez,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE.

Ing. Pedro Antonio Aguilar Polanco
Decano

DECANO
FACULTAD DE INGENIERIA

Guatemala, septiembre de 2016

/cc

ACTO QUE DEDICO A:

Dios	Por darme la vida y oportunidad de superarme y llegar hasta aquí.
Mis padres	Willy Orlando Herrera Monterroso y Nilda Ileana Gutiérrez Hernández, por su apoyo incondicional.
Mis hermanos	Walter, Rodrigo y José Alberto Herrera Gutiérrez, por su ayuda y aliento.
Mis primas y sobrinos	Claudia y Pamela Morales, María Argüello y los hijos de cada una de ellas.
Mi familia	Quienes estuvieron siempre pendientes de mí y ofrecieron palabras de aliento.
Mis amigos	Aquellos que me acompañaron durante esta parte de mi vida.

AGRADECIMIENTOS A:

Pueblo de Guatemala	Por permitirme continuar mis estudios en la Universidad de San Carlos de Guatemala.
Universidad de San Carlos de Guatemala	Por las diferentes experiencias y enseñanzas que obtuve durante toda la carrera.
Facultad de Ingeniería	Por todo el conocimiento que me transmitió a través de los diferentes catedráticos que tuve a lo largo de mi carrera.
Mis padres	Por su apoyo incondicional durante toda la carrera. Definitivamente este éxito se lo debo a ellos.
Mis hermanos	Por su ayuda brindada durante la carrera, el saber que cuento con ellos me da tranquilidad.
Mis amigos de la Facultad	Por su amistad, paciencia, ayuda y tiempo que me brindaron. Su compañía durante esta aventura fue inigualable.
Mis amigos	A todas esas personas que fuera de la facultad que siempre me mostraron su apoyo.
Mi familia	Por estar siempre pendientes de mis avances.

Mi asesor Ing. William Escobar, por su tiempo, consejos y acertadas correcciones brindadas durante el desarrollo del presente trabajo de graduación.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

GLOSARIO

API	Por sus siglas en inglés, Application Program Interface, es un conjunto de rutinas, protocolos y herramientas utilizadas para la construcción de software.
Array	Contenedor de objetos que puede almacenar una cantidad definida de objetos. El acceso a cada uno de los objetos es de manera directa y basada en índices.
Bytecode	Es código objeto procesado por un programa, regularmente una máquina virtual.
C	Lenguaje de programación multipropósito diseñado por Dennis Ritchie, base de lenguajes modernos.
CSS	Por sus siglas en inglés, Cascading Style Sheets, es un lenguaje utilizado para dar formato a los elementos HTML.
FTP	Por sus siglas en inglés, File Transfer Protocol, es un protocolo estándar utilizado para la transferencia de archivos entre un cliente y un servidor.

HTML	Por sus siglas en inglés, HyperText Maked Language, es un lenguaje utilizado para la creación de páginas web.
HTTP	Por sus siglas en inglés, Hypertext Transfer Protocol, es un protocolo de transferencia utilizado ampliamente en la web.
IRC	Internet Relay Chat, protocolo de comunicación que facilita el envío de mensajes en tiempo real entre dos o más personas.
Java	Lenguaje de programación multiplataforma orientado a objetos, creado por la empresa Sun Microsystems.
JavaScript	Lenguaje de programación interpretado, ampliamente utilizado en el desarrollo de sitios web dinámicos.
JSON	Por sus siglas en inglés, JavaScript Object Notation, es un formato liviano de intercambio de información.
Framework	Conjunto de módulos y rutinas que proveen funcionalidades genéricas, para la creación de aplicaciones específicas.
Hardware	Son las partes físicas que conforman un computador.

Librería de software	También llamadas bibliotecas, es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación.
Python	Lenguaje de programación de alto nivel, multipropósito e interpretado.
Ruby on Rails	Framework open source utilizado ampliamente en la creación de sitios web
Rubyista	Persona que utiliza y pertenece a la comunidad del lenguaje de programación Ruby.
Scripts	Programas escritos para un entorno específico, utilizado para automatizar la ejecución de tareas.
Scripting	También llamado lenguaje de *scripts*, se refiere a los lenguajes de programación que permiten la elaboración de *scripts*.
Scrum	Metodología ágil para el desarrollo de proyectos, usualmente desarrollo de software.
Software	Conjunto de instrucciones que un computador procesa para realizar tareas específicas.
Tablas hash	Es una estructura de datos basada en funciones, recibiendo así una llave y retorna el valor asociado a la llave.

UTF-8	Sistema de codificación de caracteres.
XP	Conocido como Extreme Programming, es una metodología de desarrollo de software enfocada en la calidad y respuesta a los cambios de los requerimientos.

RESUMEN

En la actualidad, Ruby es conocido mundialmente como un lenguaje bastante agradable para el programador, además de ser utilizado por varias compañías a nivel internacional.

Ruby es un lenguaje de programación dinámico y de código abierto. Fue diseñado por el japonés Yukihiro Matsumoto, quien tomó ideas de sus lenguajes favoritos (Perl, Smalltalk, Eiffel, Ada y Lisp), para la creación del lenguaje. Desde su inicio, Ruby ha madurado y ganado popularidad, tanto dentro como fuera de Japón.

Entre las razones por las que Ruby ha ganado popularidad se encuentra su sintaxis. Este lenguaje fue diseñado teniendo en mente al programador como persona, no solo aspectos funcionales; motivo por el cual Ruby es un lenguaje con una sintaxis flexible y amplia. En Ruby se maneja la idea de que existen más de una manera de realizar una misma acción, por lo que el programador es libre de elegir la manera que él desee.

Al ser un lenguaje de código abierto, la comunidad de Ruby participa activamente en su desarrollo. Ruby cuenta con un conjunto de librerías robustas, documentación bastante completa y variedad de material para aprender el lenguaje. Además, con varios *frameworks* orientados a la creación de sitios web.

El *framework* más conocido de este lenguaje es Ruby on Rails (RoR). Este es un *framework* completo, que permite la creación de sitios web de una

manera sencilla y rápida. La filosofía de RoR es convención sobre configuración, esto se refiere a que el programador tiene que conocer las convenciones y enfocarse en las funcionalidades sin perder tiempo en configuraciones. Este *framework* es utilizado en varios sitios conocidos, como Twitch, Github, Shopify, entre otros. RoR es uno de los motivos por el que Ruby ganó tanta popularidad.

Ruby, además de ser utilizados en ambientes web, también se usa para la realización de *scripts*. Google Sketchup es una aplicación que permite utilizar Ruby como lenguaje para la realización de *scripts*; a través de estos el usuario puede ampliar las funcionalidades de Google Sketchup, y así satisfacer necesidades más específicas.

Ruby es un lenguaje agradable y sencillo de utilizar. Al ser interpretado tiene ciertas limitaciones por su rendimiento frente a lenguajes como C, pero sus ventajas como su sencillez y libertad se contraponen a esta limitación. Definitivamente, Ruby es un lenguaje interesante que vale la pena conocer.

OBJETIVOS

General

Estudiar de manera profunda el lenguaje de programación Ruby, buscando definir las ventajas, desventajas, alcances y limitaciones del mismo en proyectos de programación, así como el panorama nacional e internacional del lenguaje actualmente.

Específicos

1. Definir las ventajas y desventajas que ofrece el lenguaje Ruby al utilizarlo en proyectos de programación.

2. Conocer los alcances y limitaciones técnicas propias del lenguaje Ruby, definiendo en qué ambientes se obtiene el mayor beneficio de su utilización.

3. Conocer el panorama general nacional e internacional del lenguaje de programación actualmente.

INTRODUCCIÓN

En el mundo de la programación existe gran variedad de lenguajes disponibles para utilizar, cada uno con sus ventajas y desventajas. A medida que evolucionan, los lenguajes de programación suelen facilitar la tarea al programador. Este patrón se observa al comparar el lenguaje ensamblador con el lenguaje C, o más recientemente el lenguaje C con Java.

La diversificación de los lenguajes también se ha dado por los diferentes problemas que se buscan solucionar. Por ejemplo, el lenguaje R, el cual está orientado a problemas estadísticos; el lenguaje PHP, el cual se orienta a aplicaciones web, entre otros.

Todo lo anterior es prueba de la diversificación de lenguajes de programación. Esto no es malo, ya que permite encontrar lenguajes que se adapten a las necesidades y gustos, en lugar de buscar adaptarse al lenguaje.

Ruby es uno de los tantos lenguajes que existen hasta la fecha, el cual es el propósito en la presente investigación. Según palabras del creador de Ruby, el japonés Yukihiro Matsumoto, menciona que fue diseñado para la productividad y la diversión del desarrollador, es decir, se enfatiza en las necesidades del ser humano más que en las de la máquina.

Con el presente estudio se busca presentar el lenguaje Ruby, para que sea conocido y posteriormente considerado a la hora que el lector decida aventurarse en un proyecto de programación. Teniendo en cuenta las ventajas y desventajas que pueda conllevar la utilización del lenguaje.

1. INTRODUCCIÓN GENERAL A RUBY

Ruby es un lenguaje de programación orientado a objetos con el propósito de hacer la programación entretenida y rápida. Asimismo, dinámico y de código abierto con una gramática compleja, pero bastante expresiva. Fue inspirado en lenguajes como Perl, Lisp y Smalltalk. A pesar que un lenguaje de programación orientado a objetos, también es adecuado para el estilo de programación funcional.

1.1. Reseña histórica

Ruby fue concebido el 14 de febrero de 1993, por el japonés Yukihiro Matsumoto, el nombre fue seleccionado entre Coral y Ruby. La idea que buscaba Matsumoto era un lenguaje de *scripts* completamente orientado a objetos. Aunque conocía otros lenguajes como Perl y Python, estos no llenaban las necesidades de Matsumoto, ya que Perl no le parecía muy serio y Python no le era un lenguaje puramente orientado a objetos.

Matsumoto lanzó la primera versión pública (0.95) de Ruby en diciembre de 1995. Un año después se publicó Ruby 1.0, posteriormente Ruby 1.1 en agosto de 1997. La primera versión estable de Ruby fue la 1.2, y fue publicada en diciembre de 1998.

Hasta ese momento Ruby era conocido únicamente en Japón, pero en 1998, Matsumoto creó la primera página web en inglés de Ruby. Luego inició la primera lista de correos (*mailing list*) en inglés, conocida como “Ruby-Talk”. Estas dos acciones, iniciaron la expansión de Ruby fuera de Japón.

En 1999, Yukihiro Matsumoto conjuntamente con Keiju Ishitsuka escriben el primer libro del lenguaje de programación Ruby en japonés, titulado *The Object-oriented Scripting Language Ruby*, y proporcionó popularidad en Japón. Posteriormente se publica el primer libro de Ruby en inglés, *Programming Ruby*, ayudando así a la expansión del lenguaje fuera de Japón.

En el 2003 hubo grandes cambios en el lenguaje, ya que en ese año se presentaría la versión 1.8 del lenguaje. Esta trabajó varias características importantes al lenguaje, entre ellas se encuentra el soporte nativo a YAML.

En el 2005 aparece el *framework* web Ruby on Rails. La combinación de Ruby con la ideología de Rails, convención sobre configuración, conjuntamente con la estructura modelo vista controlador encajaron perfectamente para el desarrollo de aplicaciones web. Debido a la facilidad y rapidez que otorgaba tuvo bastante aceptación, y con ello se aumentó significativamente la popularidad del lenguaje.

Posteriormente, Ruby tuvo diferentes versiones. En el 2011 entra en escena la versión 1.9.3, la cual venía acompañada de varios cambios significativos; como las mejoras en temas de velocidad, nueva sintaxis para la definición de datos hash, soporte IPv6, mejoras a las expresiones regulares, entre otros.

En febrero de 2013 aparece la versión 2.0.0 de Ruby, esta igualmente trae consigo mejoras al lenguaje. Entre ellas el uso UTF-8 por defecto, mejoras en velocidad, optimizaciones al recolector de basura, entre otras.

La versión estable de Ruby 2.2.0 integra nuevas características. Entre las más notables están: el versionamiento semántico, mejoras en velocidad y optimizaciones en el uso de la memoria.

Actualmente, Ruby va por la versión estable 2.3.0 y una de las características más notables es la introducción de las cadenas de texto inmutables, *frozen string literal*. Para obtener información más detallada sobre los cambios realizados en las distintas versiones, ver el sitio oficial de Ruby www.ruby-lang.org.

Desde sus inicios hasta hoy, Ruby ha demostrado un gran crecimiento en funcionalidades y popularidad y se espera continúe creciendo como lo ha hecho a la fecha.

1.2. Instalación

Para iniciar a programar en Ruby, hay que instalar Ruby. Existen diferentes maneras de hacerlo, dependiendo del sistema operativo sobre el que se trabaje. A continuación, se enumeran las diferentes maneras de realizar la instalación según el sistema operativo.

- Linux/Unix: se utiliza el sistema de gestión de paquetes de la distribución o instaladores de terceros como rbenv y RVM.
- OS X: emplear instaladores de terceros como rbenv y RVM.
- Windows: usar el instalador RubyInstaller.

1.2.1. Sistema de gestores de paquetes

Este modo de instalación es para sistemas operativos basados en UNIX, los cuales cuentan con un repositorio de aplicaciones. El comando suele variar dependiendo del sistema operativo.

Se recomienda evitar este tipo de instalación, porque los gestores de paquetes no suelen tener la versión más reciente de Ruby en sus repositorios oficiales.

1.2.2. Instaladores de terceros

Si se desea tener la versión más reciente de Ruby, se puede instalar haciendo uso de un instalador de terceros. Algunos, incluso ayudan a instalar varias versiones en un solo sistema, así como cambiar entre versiones de Ruby.

Entre estos instaladores están:

- Ruby-build (OSX, Linux)
- Ruby-install (OSX, Linux)
- RubyInstaller (Windows)

1.2.3. Compilar el código fuente

Otra alternativa es instalarlo utilizando el código fuente. Solo hay que descargarlo de la página oficial de Ruby, desempaquetarlo y luego compilarlo. Si no se tiene experiencia en este tipo de tareas, posiblemente usar un instalador de terceros sea una mejor opción.

Para más información sobre la instalación de Ruby visitar la página oficial www.ruby-lang.org.

1.3. Funcionamiento técnico

Todo lenguaje de programación pasa por un proceso de compilación o interpretación para ser ejecutado. La compilación es un proceso de transformación que consiste en traducir las instrucciones escritas en un lenguaje X a un lenguaje entendible para el computador, lenguaje máquina, o como mínimo que facilite su ejecución. La interpretación consiste en analizar cada instrucción y ejecutarla conforme se encuentran; en la interpretación pura no se generan instrucciones en lenguaje máquina.

Existen varias implementaciones de Ruby, y dependiendo de cuál se utilice, el proceso para ejecutar los programas puede variar. En esta sección se describirá la implementación estándar conocida como Matz's Ruby Interpreter (MRI), también se explorarán otras implementaciones como JRuby y Rubinius.

1.3.1. Matz's Ruby Interpreter

La MRI es la implementación estándar de Ruby. Esta es interpretada, es decir, las instrucciones son analizadas una por una y ejecutadas conforme son encontradas.

En Ruby existen dos tipos diferentes de interpretación soportadas por el intérprete de Ruby. Uno de estos tipos es el modo interactivo, consiste en ingresar al intérprete una instrucción a la vez, y el intérprete la ejecuta, retornando el resultado de la instrucción. Este modo es utilizado especialmente

en el proceso de aprendizaje del lenguaje, aunque también puede ser utilizado para realizar pruebas o incluso como una calculadora.

El segundo modo es llamado modo *batch*. En este el intérprete recibe un archivo, el cual contiene las instrucciones del programa, y lo prepara para la ejecución, y posteriormente lo ejecuta. Es utilizado por todo proyecto en Ruby.

1.3.1.1. Interpretación modo *batch*

Una vez escrito el código del programa y se ejecuta, el intérprete de Ruby se encarga de preparar el código para la ejecución. Este proceso es totalmente transparente para el programador. La fase inicial del proceso es un análisis léxico, seguido de un análisis sintáctico al código, siguiendo la compilación y, por último, la ejecución.

Ruby inicia este proceso abriendo el archivo y posteriormente leyendo el código carácter por carácter. Conforme lee los caracteres, Ruby los agrupa formando así tokens. Por ejemplo, el intérprete de Ruby encuentra el número 15 en su código, Ruby procedería a agrupar el carácter 1 con el 5, y los identificaría como un token integer. Ruby realiza este proceso desde el inicio del programa hasta la última línea de código, y como resultado obtiene los tokens que conforman el programa.

Una vez Ruby convierte el código en *tokens*, procede a analizar sintácticamente. Esto consiste en agrupar los *tokens* en sentencias o frases con sentido para Ruby. Para que Ruby interprete la siguiente línea de código “x = 15”, primeramente, busca los *tokens*: “x” es un identificador, “=” signo de asignación y “15” un integer. Luego agrupa estos 3 *tokens* y forma con ellos una sentencia de asignación, lo cual ya tiene un significado para el intérprete. El

resultado de este proceso es un árbol de sintaxis abstracta (AST por sus siglas en inglés, Abstract Sintax Tree).

Hay que aclarar que el proceso de análisis sintáctico se realiza simultáneamente con el proceso de lectura y agrupación de los caracteres. Cuando el proceso de parseo necesita *tokens*, este se los solicita al proceso de lectura, quien se encarga de leer y retornarle el siguiente *token* al proceso de análisis sintáctico.

Versiones anteriores a Ruby 1.9 proceden a ejecutar el código inmediatamente después del proceso de lectura y análisis sintáctico. La ejecución se realizaba recorriendo el árbol de sintaxis abstracta. Pero desde la versión 1.9 se implementó un paso extra, el correspondiente a la compilación.

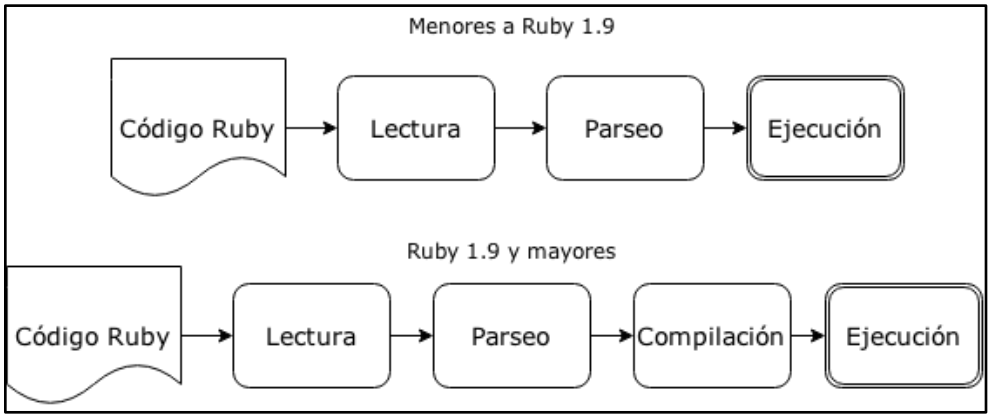
Desde Ruby 1.9, Koichi Sasada, conjuntamente con el equipo de Ruby introdujeron Yet Another Ruby Virtual (YARV). Esta máquina virtual trabaja con una idea similar a la Java Virtual Machine. YARV es la máquina virtual de Ruby, esta es la encargada de ejecutar el código generado por la fase de compilación.

La fase de compilación de Ruby consiste en tomar el árbol de sintaxis abstracta, recorrerlo y finalmente generar código *bytecode*. El *bytecode* es una serie de instrucciones de bajo nivel las cuales son entendidas por la máquina virtual YARV. Finalmente, el *bytecode* es el ejecutado por la máquina virtual YARV. Al igual que la máquina Java Virtual Machine, YARV interpreta el código *bytecode*. En la figura 1 se muestran ambos procesos de ejecución, en las versiones inferiores a la 1.9 y el correspondiente a la 1.9 y superiores.

1.3.2. Otras implementaciones de Ruby

La implementación estándar de Ruby está desarrollada en C. Otras del intérprete de Ruby han sido desarrolladas en otros lenguajes. Este es el caso de Rubinius y JRuby, los cuales se detallan a continuación.

Figura 1. **Proceso de ejecución**



Fuente: elaboración propia, empleando Microsoft Word.

1.3.2.1. JRuby

JRuby, al igual que la implementación estándar de Ruby, trabaja las mismas fases que el proceso de interpretación. Pero existen ciertos aspectos que diferencian a JRuby de la implementación estándar.

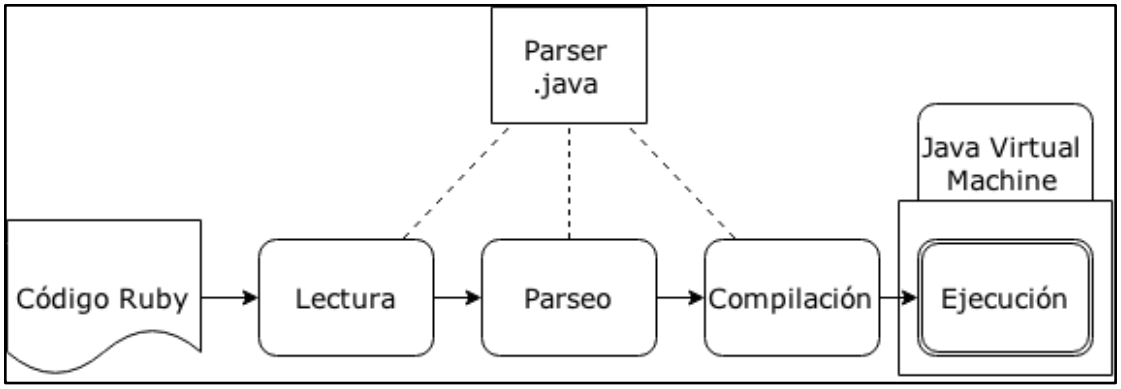
El analizador sintáctico de JRuby está escrito en Java, a diferencia de la implementación estándar el cual está escrito en C. En la implementación estándar, primeramente se obtienen los *tokens*, luego el árbol de sintaxis abstracta y, por último, genera *bytecode* entendible por la máquina virtual YARV. En JRuby se obtienen los *tokens*, seguido del árbol de sintaxis abstracta

y finalmente genera Java bytecode, es decir, código entendible para la máquina virtual de Java, Java Virtual Machine (JVM).

El Java *bytecode* generado no puede ser ejecutado en la máquina virtual YARV, por lo que el código necesariamente se ejecuta en la Java Virtual Machine. Ejecutar el código en la máquina virtual de Java trae consigo algunas ventajas:

- Permite compilar el código llegando al código máquina usando un compilador JIT.
- Lleva 20 años en desarrollo, por lo que tiene soluciones sofisticadas para diferentes problemas.

Figura 2. **Ejecución JRuby**



Fuente: elaboración propia, empleando Microsoft Word.

1.3.2.2. Rubinius

Rubinius trabaja similar a las dos implementaciones vistas anteriormente, teniendo su principal diferencia en la ejecución. Dentro de las características de

Rubinius está la compatibilidad nativa con extensiones C, uso de hilos nativos del sistema operativo y soporte para plataformas OSX y Linux; Windows aún no es soportado.

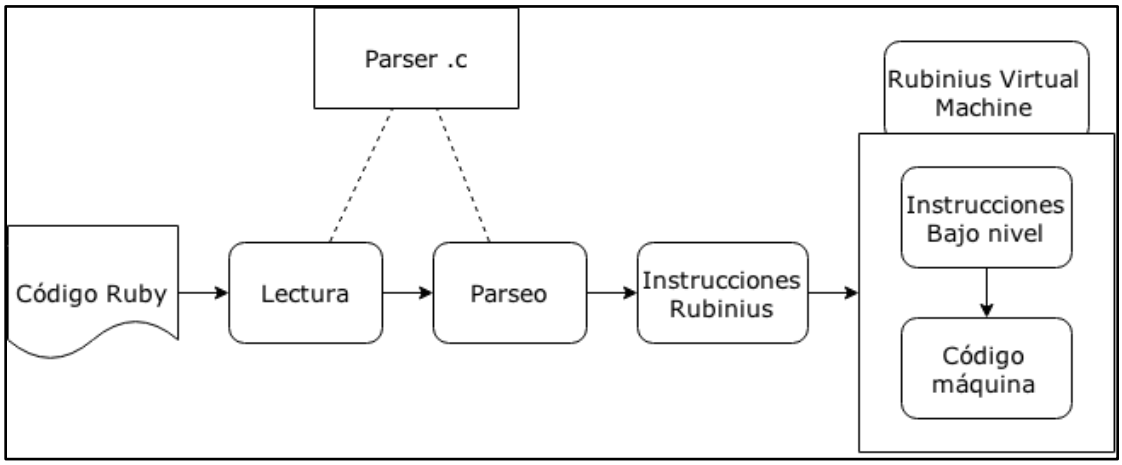
Rubinius se divide en dos partes.

- El kernel Rubinius: está escrita en Ruby. esta incluye la definición del lenguaje, incluyendo clases core como String y Array.
- Rubinius *virtual machine*: está escrita en C++. Se encarga de ejecutar las instrucciones *bytecode* del kernel de Rubinius y desarrollar tareas de bajo nivel, como el recolector de basura.

El proceso de ejecución de la implementación Rubinius, al igual que las implementaciones anteriores inicia con la obtención de los *tokens*, posteriormente la creación del árbol de sintaxis abstracta y finalmente la traducción a instrucciones entendibles por la máquina virtual de Rubinius.

Dentro de la máquina virtual de Rubinius es donde se encuentran las mayores diferencias con respecto a las implementaciones anteriores. Rubinius utiliza un *framework* de compilación llamado Low-Level Virtual Machine (LLVM), obteniendo así instrucciones de bajo nivel. LLVM puede compilar estas instrucciones para obtener instrucciones en lenguaje máquina, esto usando un compilador JIT (Just in Time). El proceso de Rubinius se ilustra en la figura 3.

Figura 3. **Ejecución Rubinius**



Fuente: elaboración propia, empleando Microsoft Word.

2. TIPOS, OPERADORES, CLASES Y CONTROLES DE FLUJO EN RUBY

En el capítulo anterior se describió la evolución de Ruby a lo largo de los años. También el proceso de ejecución de un programa escrito en Ruby y su instalación.

En este capítulo se explora el lenguaje Ruby como tal, sus tipos de datos y sintaxis. Se indagará en las diferentes características que hacen a este lenguaje único y preferido por muchos desarrolladores.

2.1. Tipos de datos

Como se ha mencionado anteriormente, Ruby es un lenguaje completamente orientado a objetos. Por lo que se necesita saber que todo en Ruby es un objeto, incluyendo los datos numéricos y cadenas de texto.

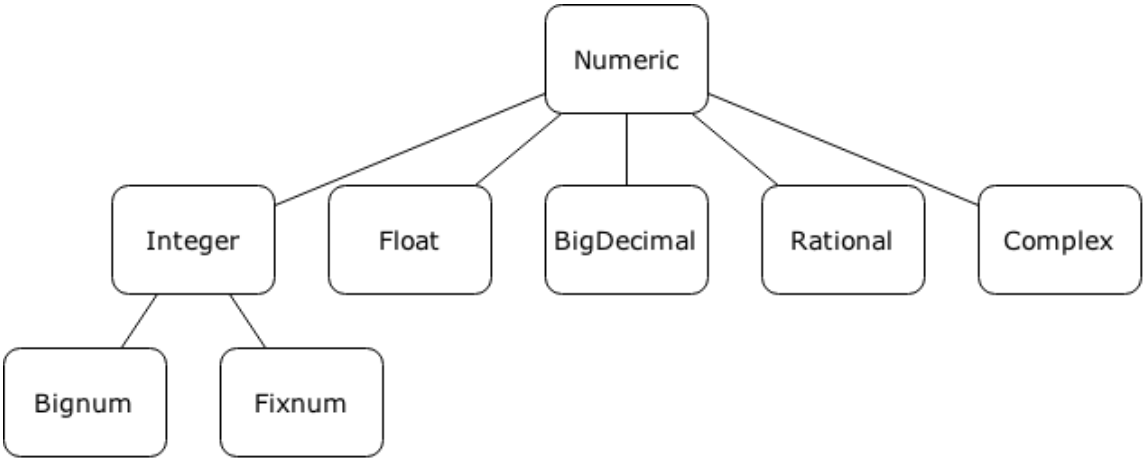
Los tipos de datos o estructuras básicas de Ruby son los datos numéricos, las cadenas de texto, los valores booleanos, los arrays y hashes; los cuales se describen a continuación.

2.1.1. Datos numéricos

Ruby incluye 5 clases para representar los valores numéricos. Estas clases se utilizan dependiendo del valor que se quiera utilizar, o nivel de exactitud necesaria.

En la figura 4 se muestran estas clases y cómo se ubican jerárquicamente.

Figura 4. **Jerarquía de clases para valores numéricos**



Fuente: FLANAGAN & MATSUMOTO. *The Ruby Programming Language.* p.42.

Como se evidencia en la figura 4, todo valor numérico en Ruby es una instacia de la clase Numeric. Pero dependiendo del tipo de valor numérico, se utiliza una de las siguientes clases:

- Integer: es la base para dos clases concretas:
    - Fixnum: contiene valores enteros que pueden ser representados en el tamaño word (generalmente 64 bits) menos 1 bit.
    - Bignum: contiene valores enteros fuera del rango de la Fixnum. Hay que notar que, si un número sale del rango de la clase Fixnum, este es convertido a la clase Integer correspondiente.
- Float: representa valores reales.

- BigDecimal: es más exacto en el manejo de números reales que la clase Float.

- Rational: representa números racionales a/b, donde tanto el numerador y denominador son números enteros.

- Complex: se utiliza para manipular números complejos a+bi, donde ‘a’ es la parte real, ‘b’ la parte imaginaria y ‘i’ la unidad imaginaria.

Hay que tener en cuenta que todos los valores numéricos son inmutables, es decir, no existen métodos que permitan cambiar el valor que contienen. Los tipos de datos numéricos más utilizados son, por lo general, valores enteros o reales, por lo que se describirán especialmente estos.

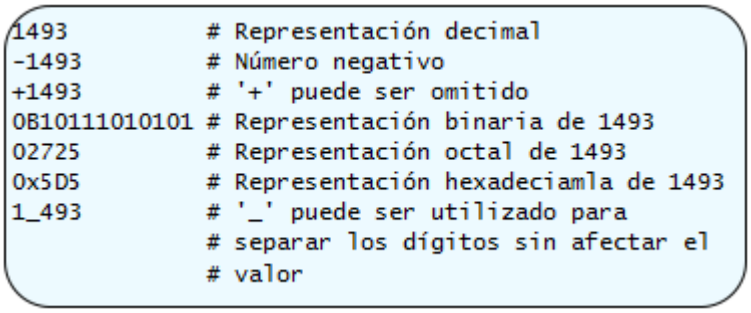
2.1.1.1. Número enteros

Son representados como una lista de dígitos y dependiendo del valor serán instancias de la clase Fixnum o Bignum. Ruby permite representar los valores numéricos en 4 diferentes bases: base 2 (binarios) se debe anteponer ‘0B’, base 8 (octal) se debe anteponer un ‘0’, base 10 (decimal) no necesita anteponer nada, como ‘2399’, base 16 (hexadecimal) se debe anteponer ‘0x’. En la figura 5 se muestran diferentes representaciones del valor 1493.

2.1.1.2. Números reales

La clase Float es la utilizada para representar valores reales. Ruby permite la utilización de diferentes notaciones para representar valores reales, estas se muestran en la figura 6.

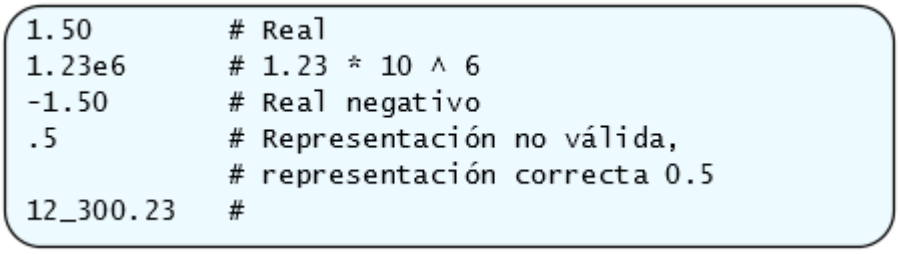
Figura 5. **Representaciones de números enteros**



```
1493          # Representación decimal
-1493         # Número negativo
+1493         # '+' puede ser omitido
0B10111010101 # Representación binaria de 1493
02725         # Representación octal de 1493
0x5D5         # Representación hexadeciamla de 1493
1_493         # '_' puede ser utilizado para
              # separar los dígitos sin afectar el
              # valor
```

Fuente: elaboración propia, empleando Microsoft Word.

Figura 6. **Representaciones de números reales**



```
1.50        # Real
1.23e6      # 1.23 * 10 ^ 6
-1.50       # Real negativo
.5          # Representación no válida,
            # representación correcta 0.5
12_300.23   #
```

Fuente: elaboración propia, empleando Microsoft Word.

2.1.1.3. Operaciones con valores numéricos

Todos los valores numéricos definen signos (operadores) para las operaciones permitidas en Ruby (suma, resta, multiplicación, división, entre otros). Los operadores se enumeran y describen en la tabla I.

Tabla I. **Operadores aritméticos**

Operador	Operación
+	Suma.
-	Resta.
*	Multiplicación.
/	División: la división de un entero entre 0 causa el error ZeroDivisionError. La división de un real entre 0 retorna el valor *infinity* (Infinito). Finalmente 0.0/0.0 retorna el valor especial NAN *Not a Number*.
%	Módulo: devuelve el residuo de la división de dos enteros. Válido solo para valores enteros.
**	Potencia

Fuente: elaboración propia, empleando Microsoft Word.

Los valores numéricos también pueden ser comparados a través de los operadores de comparación, suelen retornar valores booleanos (*true* y *false*). Los operadores principales son mostrados en la tabla II.

También existen otros operadores definidos únicamente para las clases Fixnum y Bignum. Estas realizan operaciones a nivel de bits. No suelen ser muy utilizados, pero es importante conocerlos, ver tabla III.

Tabla II. **Operadores de comparación**

Operador	Operación
<	Menor que.
>	Mayor que.
>=	Mayor o igual que.
<=	Menor o igual que.
==	Igual.
!=	Diferente o no igual.
<=>	Operador de comparación: si el valor de la izquierda es menor al valor de la derecha retorna -1, si el valor de la izquierda es mayor al valor de la derecha retorna 1, si son iguales retorna 0 y si no son comparables retorna nil.

Fuente: elaboración propia, empleando Microsoft Word.

Tabla III. **Operadores a nivel de bits**

Operador	Utilización
<<	0b1011 << 1 # => 0b10110
>>	0b10110 >> 2 # => 0b101
&	0b1010 & 0b1100 # => 0b1000
|	0b1010 | 0b1100 # => 0b1110
^	0b1010 ^ 0b1100 # => 0b110

Fuente: elaboración propia, empleando Microsoft Word.

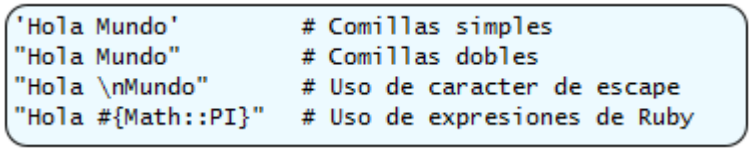
2.1.2. Cadenas de texto

El texto es manejado a través de la clase String, estos son objetos mutables, y define gran cantidad de operadores y métodos para insertar, borrar, buscar, reemplazar o extraer texto.

2.1.2.1. Representación de una cadena de texto

Existen 2 diferentes formas de representar cadenas en Ruby, ya sea usando comillas simples o dobles, ver figura 7. Estas formas se pueden usar indistintamente como se prefiera, aunque se aclara que existen ciertas diferencias.

Figura 7. **Representación de cadenas de texto**



Fuente: elaboración propia, empleando Microsoft Word.

Las comillas simples se utilizan para representar cadenas simples. Mientras que las dobles permiten utilizar caracteres de escape y expresiones de Ruby. Los caracteres de escape son representados por una diagonal inversa seguida de un carácter o secuencia de caracteres (\n). Las expresiones Ruby pueden ser insertadas a través de un signo numeral seguido de la expresión Ruby entre llaves (#{2+3}).

Ruby permite otras formas de representar cadenas de texto, pero las 2 mostradas anteriormente son las más utilizadas.

2.1.2.2. Operadores de cadenas de texto

La clase String define operadores bastantes útiles para manipular cadenas de texto. Estos operadores son presentados en la tabla IV.

Tabla IV. **Operadores de cadenas de texto**

Operador	Utilización
+	Devuelve una nueva cadena de texto nueva con las dos cadenas concatenadas.
<<	Concatena la cadena del lado derecho al final de la cadena del lado izquierdo.
*	Repite la cadena del lado izquierdo la cantidad que indique el valor numérico del lado derecho.

Fuente: elaboración propia, empleando Microsoft Word.

2.1.3. Array

Es una secuencia de valores que permite acceder a los valores a través de su posición o índice. En Ruby, al igual que C, el primer elemento de un array se encuentra en la posición 0. El tamaño de un array puede ser accedido a través de los métodos *size* y *length*, por lo que el índice mayor de cualquier array es size – 1. Los índices de un array también pueden ser negativos, estos índices se toman desde el final hacia el inicio del array.

Los arrays en Ruby son mutables y no tiene tipo, por lo que, los elementos del array no necesitan ser de la misma clase y pueden cambiar en cualquier momento.

2.1.3.1. Representación de un array

Los arrays en Ruby se representan como una lista de objetos separados por comas y encerrados entre corchetes. Esta es la forma común de representar un array, aunque al igual que las cadenas de texto, existen otras maneras de representarlos, como se observa en la figura 8.

Figura 8. **Representación de arrays**

```
[1, 2, 3, 4, 5]  # Representación de un array
[1 ..5]          # Representación anterior usando rangos
[1,[2, 3], 4,5]  # Distintos objetos
[1+2,3+4,5]      # Expresiones dentro del array
[]               # Array vacío
```

Fuente: elaboración propia, empleando Microsoft Word.

2.1.3.2. Operadores de arrays

Existen operadores útiles proporcionados por Ruby para el manejo de arrays. Estos operadores suelen recibir dos arrays y retornar un nuevo array resultado. Los operadores se detallan en la tabla V.

Tabla V. **Operadores de arrays**

Operador	Utilización
+	Retorna un nuevo array que primeramente contiene los datos del array de la izquierda y posterior a ellos contiene el array de la derecha.
|	Similar al operador anterior, solo que este asegura que cada elemento este una única vez.
&	Retorna un nuevo array con los elementos que se encuentren tanto en el array de la derecha como en el array de la iquierda.
<<	Ingresa el objeto de la derecha al final del array de la izquierda.
*	Retorna un nuevo array con el contenido del array de la izquierda repetido el número de la derecha.

Fuente: elaboración propia, empleando Microsoft Word.

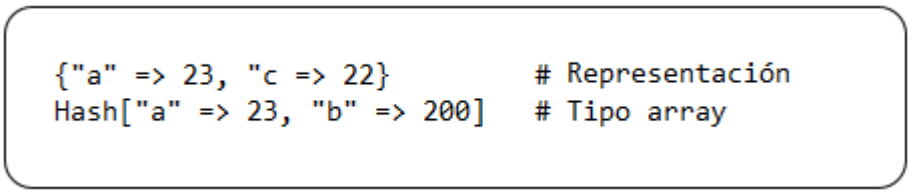
2.1.4. Tablas hash

Son estructuras que tienen un conjunto de objetos conocidos como llaves, y cada llave está asociada a un valor.

2.1.4.1. Representación de tablas hash

Son representadas como una lista entre llaves de parejas clave-valor separadas por comas. La llave y el valor son separados a través de una flecha formada por el signo igual y el signo mayor que (=>), ver figura 9.

Figura 9. **Representación de tablas hash**



Fuente: elaboración propia, empleando Microsoft Word.

2.1.5. Valores booleanos

Ruby, también cuenta con valores booleanos *true* y *false*. Y como se ha dicho anteriormente, todo en Ruby es un objeto, y los valores booleanos también son objetos. Cada uno se evalúa como un objeto especial, *true* como instancia singleton de la clase TrueClass, y *false* como instancia singleton de la clase FalseClass.

2.1.6. Valor *nil*

Es el valor equivalente del valor *null* en la mayoría de lenguajes. Como todo en Ruby *nil* corresponde a un objeto. Este es instancia singleton de la clase NilClass.

2.2. Estructuras de control de flujo

Ruby cuenta con estructura de control. Las estructuras de control alteran la secuencia de ejecución del código. Estas estructuras son las que permiten crear que los programas dinámicos que respondan a diferentes interacciones con el usuario. En esta sección se enumerarán las diferentes estructuras de control existentes en Ruby.

2.2.1. Condicionales

Los condicionales son maneras de ejecutar cierta parte del código solamente si se cumplen condiciones establecidas. En Ruby, toda condición es una expresión, y si la expresión al ser evaluada retorna los valores *false* o *nil* se considera que la condición no fue satisfecha, en caso contrario la condición se considera satisfecha.

2.2.1.1. Condicional if

El condicional if, simplemente ejecuta el código contenido si la condición es verdadera. La sentencia else se puede utilizar conjuntamente con el condicional if, para indicar código a ejecutarse en caso la condición fuese falsa.

También permite la utilización de la sentencia elif, permitiendo adherir a múltiples bloques de código con su respectiva condición. Al ejecutarse el código se evaluará la primera condición; si esta es verdadera se procederá a ejecutar el bloque de código correspondiente, en caso contrario, se procederá a evaluar la siguiente condición. Si no existe condición verdadera ejecutará el bloque de código correspondiente a la sentencia else si existiese.

La separación entre la condición y el bloque de código a ejecutar, se puede realizar a través del uso de la palabra reservada “then”, un punto y coma “;” o un salto de línea. Ruby exige utilizar como mínimo un separador. Se recomienda el uso de la palabra reservada “then”, debido a la expresividad y legibilidad que aporta al código. La sintáxis del condicional if se puede apreciar en la figura 10.

Al ser el condicional if una instrucción, esta devuelve un valor al ser ejecutada. El valor que devuelve será el valor de la última instrucción ejecutada dentro del condicional. En caso de no ejecutar ninguna instrucción, el condicional if retorna el valor nil.

2.2.1.2. Conodicional unless

Es el opuesto al condicional if. Su comportamiento consiste en ejecutar el bloque de código sí y solo sí la condición al ser evaluada devuelve falso o el valor nil. Al igual que el condicional if, unless puede ser acompañado por la sentencia else, y también puede ser utilizado como un modificador. La sintaxis se ilustra en la figura 11.

2.2.1.3. Condicional case

Este permite ejecutar distintos bloques de código, dependiendo de diferentes condiciones. Existen dos maneras diferentes de definir el condicional case. La primera consiste en realizar múltiples condicionales, equivalente a la sintaxis if/elif/else. La segunda, en la definición de una expresión y posibles valores que podría retornar dicha expresión. La sintaxis se ejemplifica en la figura 12.

Figura 10. **Sintaxis if**

```
#If simple
if <expresión> then
  <código>
end

#If-else
if <expresión> then
  <código>
else
  <código>
end

#Elif
if <expresión> then
  <código>
elif <expresión> then
  <código>
...
else
  <código>
End

#if como modificador
<sentencia> if <condición>
```

Fuente: elaboración propia, empleando Microsoft Word.

Al igual que la sentencia if y unless, el valor retornado por el condicional case corresponde a la última sentencia ejecutada, en caso contrario se retornará el valor nil.

Figura 11. **Sintaxis unless**

```
#unless simple
unless <expresión>
  <código>
end

#unless - else
unless <expresión>
  <código>
else
  <código>
End

#unless como modificador
<instruccion> unless <condición>
```

Fuente: elaboración propia, empleando Microsoft Word.

2.2.2. Loops

Son estructuras que permiten ejecutar un bloque de código cuantas veces se desee. El lenguaje de programación Ruby tiene distintas formas sintácticas que permiten realizar esta tarea.

2.2.2.1. Loop while

El loop while consiste simplemente en una expresión y un bloque de código. El bloque de código se ejecutará siempre y cuando la expresión no retorne el valor booleano falso o el valor nil. En este loop la condición es verificada antes de ejecutar el bloque de código. El ciclo while puede ser utilizado como modificador. La sintaxis se ejemplifica en la figura 13.

Figura 12. **Sintaxis case**

```
#Case simple
case
when <condición>
  <código>
..
else
  <código>
End

#Case minimizado
Case <expresión>
when <valor>
  <código>
..
else
  <código>
end
```

Fuente: elaboración propia, empleando Microsoft Word.

Figura 13. **Sintaxis while**

```
#while simple
while <condición>
  <código>
End

#while como modificador
<instrucción> while <codición>
```

Fuente: elaboración propia, empleando Microsoft Word.

2.2.2.2. Loop until

El ciclo until es bastante similar al ciclo while, con la diferencia que el bloque de código se ejecutará siempre y cuando la condición retorne el valor booleano false o el valor *nil*. Al igual que el ciclo while, el ciclo until puede ser utilizado como modificador, siempre cumpliendo con el comportamiento definido previamente. La sintaxis se muestra en la figura 14.

Figura 14. **Sintaxis until**

```
#until simple
until <expresión>
  <código>
end

#until como modificador
<instrucción> until <codición>
```

Fuente: elaboración propia, empleando Microsoft Word.

2.2.2.3. Loop for

El ciclo for varía notablemente en comparación de los dos ciclos anteriores, ya que en este debe contener una colección y una variable. El ciclo for ejecutará el bloque de código un número de veces igual al tamaño de la colección. Durante cada iteración la variable var contendrá el elemento de la colección correspondiente al número de la iteración, ver figura 15.

Figura 15. **Sintaxis for**

```
for <var> in <colección> do
  <código>
end
```

Fuente: elaboración propia, empleando Microsoft Word.

El ciclo for no es muy utilizado, ya que los programadores prefieren utilizar los métodos específicos de cada colección para iterar sobre ella.

2.2.2.4. Métodos de iteración

Dentro del lenguaje Ruby las colecciones (arrays, listas, hash), incluso las cadenas de texto y valores numéricos tienen métodos de iteración que permiten ejecutar un bloque de código múltiples veces dependiendo de la longitud o tamaño de la colección o cadena de texto.

En la figura 16 se muestra el uso de distintos métodos de iteración. Los que se observan a continuación no son los únicos existentes. Para conocer los diferentes métodos de iteración para una clase en específico se recomienda visitar la documentación oficial de la clase.

2.3. Clases

En Ruby todo es un objeto, y un objeto contiene sus características y métodos. En esta sección se estudiará la manera en que el programador puede definir sus propias clases.

Figura 16. **Métodos de iteración**

```
#Iteración sobre un array
array.each do |x|
  <código<
End

#Iteración sobre un hash
hash.each do |key, value|
  <código<
End

#Iteración sobre un string
str.each_char do |x|
  <código<
End

#Iteración sobre un entero
3.times do |i|
  <código<
end
```

Fuente: elaboración propia, empleando Microsoft Word.

Una clase está compuesta por sus características y comportamientos, dicho de manera más técnica: atributos y métodos. La sintaxis para definir una clase se presenta a continuación, posteriormente se mostrará cómo definir métodos y la manera de utilizar los atributos. Ver sintaxis en figura 17.

2.3.1. Variables

En Ruby las variables no necesitan ser declaradas, por lo que simplemente es necesario colocar un identificador para guardar y posteriormente utilizar su valor. Las variables de una clase corresponden a las

características de la clase. En Ruby las variables pueden ser pertenecientes a la instancia, o pertenecientes a la clase.

Figura 17. **Clase**

```
# Definición de un clase
class <identificador>
  <definición de la clase>
end

# Definición objeto
<var> = <identificador>.new
```

Fuente: elaboración propia, empleando Microsoft Word.

Las variables pertenecientes a la instancia se les antepone una arroba '@', y pueden ser utilizadas únicamente dentro de los métodos propios de la instancia. Las variables de instancia son propias de un objeto en específico, por lo que cada instancia de una clase cualquiera tiene sus propias variables de instancia.

Las variables pertenecientes a la clase se les antepone dos arrobas '@@', y pueden ser utilizadas dentro de cualquier método de la clase e instancia. Las variables de clase, como su nombre lo indica son propias de la clase, por lo que todos los objetos de una clase cualquiera comparten las variables de clase.

2.3.2. Métodos

Los métodos, a diferencia de las variables necesitan ser definidos. Los métodos de una clase corresponden al comportamiento de la clase. Al igual que las variables, los métodos pueden ser propios de la clase o de la instancia.

Los métodos se definen a través de la palabra reservada def seguido de un identificador, posteriormente los parámetros separados por coma (si tuviese), luego el bloque de código y, por último, la palabra reservada end para indicar el fin del método. Ver figura 18.

Figura 18. **Métodos**

```
# Definición de un método
def <identificador> <parámetros>
  <código>
end
```

Fuente: elaboración propia, empleando Microsoft Word.

2.3.2.1. Initialize

El método initialize es llamado siempre que se crea una nueva instancia. Cualquier parámetro que se pase al método new, se transfiere al método initialize. Como aclaración, este método es privado, por lo que es imposible acceder a él directamente.

Como el método initialize es llamado al generar instancia nueva, es perfecto para inicializar todas las variables de instancia que se utilizarán. El método initialize es el equivalente al constructor en otros lenguajes, su sintaxis se puede observar en la figura 19.

2.3.3. Encapsulación en Ruby

Como se mencionó anteriormente, Ruby es un lenguaje orientado a objetos puro. Debido a esto los atributos de la clase no son accesibles

directamente desde fuera de la clase, respetando así el principio de encapsulación de la programación orientada a objetos.

Figura 19. **Initialize**

```
class <identificador>
  def initialize <parámetros>
    <código>
  end
end

#   new   llama   al   método
initialize
```

Fuente: elaboración propia, empleando Microsoft Word.

Figura 20. **Encapsulación**

```
class Ejemplo
  def initialize(x)
    @x = x
  end
  def x            #getter
    @x
  end
  def x=(value)    #setter
    @x = value
  end
end
```

Fuente: elaboración propia, empleando Microsoft Word.

Como se muestra en la figura 20, el acceso a los atributos (lectura y escritura) se realiza por medio de métodos. El primero, definido simplemente como ‘x’ es el getter, y nos permite conocer el valor guardado por la variable.

Mientras que el segundo, definido como ‘x=’ es el setter, y permite asignarle valores desde fuera de la clase. La elegancia de Ruby se hace presente en esta pareja de métodos, principalmente en el setter, ya que con esta definición se habilita el operador ‘=’ para la asignación de nuevos valores a ese atributo.

Como la elaboración de los métodos getter y setter es una tarea bastante común y mecánica; Ruby dentro de su sintaxis implementa una forma abreviada de definirlos. Está la instrucción attr_accessor la cual crea el getter y setter de las variables que se les indique, y también la instrucción attr_reader, la cual crea solamente el getter. La utilización de estas instrucciones se muestra en la figura 21.

Figura 21. **Encapsulación, accesos**

```
class Ejemplo
  def initialize(x, y)
    @x, @y = x, y
  end
  attr_accessor :x
  attr_reader :y
end
```

Fuente: elaboración propia, empleando Microsoft Word.

2.4. Librerías estándares de Ruby

En Ruby todo es un objeto y en el core de Ruby se encuentran herramientas básicas para realizar cualquier tipo de aplicación. El core de Ruby incluye las clases básicas de las que se derivan todas las demás, hasta las clases que se utilizan de los tipos básicos como: String, Float, Numeric, entre otros.

Ruby también cuenta con un grupo amplio de librerías estándares que son utilizadas para una gran variedad de tareas, como: operaciones con archivos, interfaces para la utilización de diferentes protocolos, manejo de *logs benchamarks*, entre otras.

Algunas de las librerías estándar de Ruby son las siguientes:

- Benchmark: métodos para medir y reportar el tiempo utilizado para la ejecución de código.
- CGI: soporte para el protocolo Common Gateway Inferface (CGI).
- CSV: interfaz para leer y escribir archivos CSV.
- FileUtils: métodos para operaciones varias con archivos (copiar, mover, borrar, entre otros).
- Logger: para el manejo de logs.
- Net::HTTP: cliente HTTP.
- JSON: implementa Javascript Object Notation para Ruby.

Se puede encontrar la lista completa en la documentación oficial de Ruby, la cual se ubica en el sitio web ruby-doc.org.

3. DESARROLLANDO CON RUBY

El capítulo anterior se estudió la sintaxis y funcionamiento de Ruby, con énfasis en el core del lenguaje. Este capítulo se centrará en temas directamente relacionados con el lenguaje, pero que no forman parte del lenguaje estándar en sí. Es decir, librerías externas, *frameworks* y estándares definidos para este lenguaje.

3.1. Librerías no core de Ruby

Las librerías que forman parte del core de Ruby son bastante completas, pero muchas veces suelen necesitar realizar acciones específicas. Esto obliga a implementar estas operaciones o utilizar una librería externa que realice lo que se necesita.

La gran mayoría de librerías externas en Ruby son liberadas en forma de gemas. Las gemas de Ruby son un sistema de paquetes diseñados para facilitar la creación, distribución e instalación de librerías. Otras librerías son liberadas como archivos (.zip o .tar.gz).

3.1.1. Active Support

Es una colección de clases y extensiones de la biblioteca estándar que son útiles dentro del *framework* Ruby on Rails. Estas adiciones residen en este paquete para que puedan ser cargados según sea necesario en proyectos que utilizan Rails, así como en los que no. Esta gema fue liberada bajo la licencia MIT.

3.1.2. Facets

Es la principal colección de métodos de extensión para el lenguaje de programación Ruby. Estas extensiones se almacenan en archivos individuales, permitiendo un control altamente granular. Es decir, que se encuentra modularizada y se puede incluir únicamente las extensiones que se desean. Esta gema fue liberada bajo la licencia BSD 2-Clause.

3.1.3. Shoulda

Es una gema que permite, de manera sencilla, crear pruebas entendibles para aplicaciones Ruby. Dando herramientas para categorizar las pruebas de acuerdo a características específicas o escenarios. Esta gema fue liberada bajo la licencia MIT.

3.1.4. Sqlite3

Este módulo permite que los programas Ruby interactúen con el motor de base de datos SQLite3. Para utilizarlo se debe tener instalado el motor SQLite, es compatible con SQLite versión 3.6.16 o mayores. Gema liberada bajo la licencia BSD-3.

3.1.5. Bundler

Bundler, más que una librería es una herramienta que gestiona las dependencias de gemas de aplicaciones Ruby. Se basa en un archivo donde se colocan todas las dependencias, esta gema se encarga de la labor de descargar e instalar las diferentes dependencias necesarias según se especifica en el manifiesto.

Además de estas librerías, Ruby cuenta con miles de gemas que seguramente ayudarán en las diferentes tareas que se desean realizar.

3.2. Frameworks

"Un framework no es necesario: es solamente una herramienta disponible para ayudar a desarrollar mejor y rápido"[1].

"El propósito de un framework es mejorar la eficiencia de crear nuevo software. Los frameworks pueden mejorar la productividad del desarrollador y mejorar la calidad, confiabilidad y robustez del nuevo software"[2].

Es una abstracción en donde se proporcionan funcionalidades genéricas, para ser utilizadas en variedad de aplicaciones específicas. Un *framework* promueve la reutilización para facilitar el desarrollo de software.

Un *framework* contiene ciertas características que lo diferencian de las librerías:

- A diferencia de las librerías, los *frameworks* dictan el flujo del software. En las aplicaciones normales, el flujo del programa es dictado por el programador.
- Los *frameworks* tienen un comportamiento por defecto, este comportamiento debe ser útil.
- Un *framework* debe ser extensible, permitiendo al usuario añadir funcionalidades.

[1] SYMFONY. http://symfony.com/in-five-minutes. Consulta: 14 de noviembre de 2015.
[2] BAKER MIKE. http://info.cimetrix.com/blog/bid/22339/What-is-a-Software-Framework-And-why-should-you-like-em. Consulta: 14 de noviembre de 2015.

- En general, el código del *framework* no debe ser modificado. Los usuarios pueden adherir funcionalidades al *framework*, pero no modificarlo.

Existen varios *frameworks* para el lenguaje Ruby. La mayor parte de estos están enfocados a la web. A continuación se presentaran algunos *frameworks* de Ruby disponible.

3.2.1. Padrino

"Padrino fue creado para que sea divertido y fácil de codificar aplicaciones web avanzadas, mientras que se adhiere al espíritu que hace a Sinatra genial"[3].

Padrino es un *framework* de Ruby construido sobre la biblioteca web Sinatra. Sinatra es un DSL (Domain Specific Language) para la creación de aplicaciones web sencillas en Ruby.

Sinatra es software libre y de código abierto. Es un DSL (Domain Specific Language), escrito en Ruby orientado a la web. Sinatra fue diseñado y desarrollado por Blake Mizerany, no sigue el patrón modelo-vista-controlador y está enfocado en la creación rápida de aplicaciones web en Ruby sin mayor esfuerzo.

Padrino aprovecha la simplicidad y expresividad de Sinatra, y le adhiere funcionalidades que permitan el desarrollo de aplicaciones web más complejas. El *framework* padrino busca ampliar las funcionalidades de Sinatra, pero respetando y manteniendo la esencia de este DSL.

[3] PADRINORB. http://padrinorb.com/. Consulta: 14 de noviembre de 2015.

Entre las principales funcionalidades de Padrino están:

- Creación de aplicaciones Padrino, modelos y controladores.
- Soporte para pruebas, plantillad y librerías de bases de datos.
- Diseñado para montar múltiples aplicaciones.
- Soporte para el envío rápido y simple de correos electrónicos.
- Construcción de interfaz de administrador con autenticación.
- Provee un log unificado que puede interactuar con algún ORM o librería.
- Recarga automática del código del servidor durante el desarrollo.
- Cache para reducir el tiempo de carga del sitio efectivamente y con configuraciones mínimas.

El sitio web del *framework* provee bastante documentación para iniciarse en este *framework* y aún sigue en desarrollo. Este se encuentra bajo la licencia MIT.

3.2.2. Lotus

"Un framework web completo para Ruby. Lotus trae de vuelta la programación orientada a objetos al desarrollo web, aprovechando una API estable, DSL mínimo y objetos simples"[4].

Lotus es software de código abierto que busca la simplicidad, pocos DSLs (Domain Specific Language), convenciones mínimas, más objetos, y la separación entre las capas del patrón modelo-vista-controlador. El *framework* sugiere las mejores prácticas, pero da la suficiente libertad al desarrollador para que construya su propia arquitectura utilizando sus propios objetos.

[4] LOTUS. http://lotusrb.org/. Consulta: 14 de noviembre de 2015.

Está conformado por varios pequeños *frameworks* que pueden ser utilizados individualmente. Esto para enfatizar la separación de los componentes (controlador, vistas, entre otros).

Los *frameworks* que conforman Lotus son los siguientes:

- Lotus Utils: extensiones principales y clases de utilidad.
- Lotus Route: ligero y rápido direccionador HTTP.
- Lotus View: enfatiza la separación entre vista y plantilla. Define una vista como un objeto que encapsula la lógica de presentación de una página, mientras una plantilla define la semántica y elementos visuales de una página.
- Lotus Model: API publica para ejecutar queries y comandos en una base de datos. Mantiene la lógica del negocio separada de detalles como la persistencia.
- Lotus Controller: es una pequeña librería para *frameworks* web. Trabaja perfectamente con Lotus Router. Diseñada para ser rápida.
- Lotus Validations: conjunto ligero de validaciones para objetos Ruby.
- Lotus Helpers: ofrece un conjunto de utilidades para enriquecer las vistas web.

La página de Lotus provee la información básica acerca del *framework* e información a través de la comunidad, blog y listas de correos. Cada *framework* que conforma Lotus posee su propio repositorio. Lotus esta liberado bajo la licencia MIT.

3.2.3. Ruby on Rails

"Ruby on Rails es un framework web de código abierto que está optimizado para la felicidad del programador y la productividad sostenible. Permite escribir código hermoso favoreciendo la convención sobre la configuración"[5].

Rails es un *framework* de desarrollo de desarrollo de aplicaciones web escrito en Ruby. Busca hacer la programación web más fácil. Permite escribir menos código, realizando más que muchos otros lenguajes y *frameworks*.

La filosofía de Rails se basa en dos principios:

- DRY (*Don't Repeat Yourself*): sugiere que escribir el mismo código repetidas veces es una mala práctica.
- Convención sobre configuración: Rails hace algunas suposiciones sobre lo que se va hacer, evitando especificar cada pequeña cosa a través de archivos de configuración.

Ruby on Rails es el *framework* más popular de Ruby. Muy probablemente muchas personas conocieron Ruby debido a este *framework*. Algunos sitios que utilizan este *framework* son:

- Twitter: esta red social en su inicio se elaboró con el *framework* Ruby on Rails. Por temas de escalabilidad Twitter decidió migrar sus principales servicios a Scala.
- Shopify
- Slideshare

[5] RUBY ON RAILS. http://rubyonrails.org/. Consulta: 11 de mayo de 2015.

- Bloomberg
- Airbnb
- Soundcloud
- Heroku
- Github

Rails está liberado bajo la licencia MIT.

3.3. Metodologías de desarrollo

Ruby se utiliza mayormente para el desarrollo de sitios web, por lo que resulta natural que sea el ámbito donde más avances tiene. El principal atractivo que muestra Ruby a los programadores es la rapidez y facilidad para la creación de sitios, por medio de los distintos *frameworks* que existen para el lenguaje.

Los diferentes *frameworks*, en especial Ruby on Rails que promueve la arquitectura modelo-vista-controlador, ayudan a la aplicación de metodologías ágiles. Ruby on Rails dispone de herramientas que facilitan el desarrollo de prototipos y realización de pruebas, aspectos necesarios para la utilización de metodologías ágiles.

Principalmente, Ruby on Rails tiene ciertas prácticas que permiten la utilización de metodologías ágiles como XP y Scrum. Estas prácticas se enumeran a continuación:

- *Don't Repeat Yourself* (DRY): procurar la reutilización de código, y evitar todas las posibles copias de código.

- Prueba tu código: definir las pruebas a realizar y realizarlas es parte fundamental en la aplicación de diferentes metodologías. Por lo que se deben definir las pruebas que se utilizarán a lo largo del desarrollo: unitarias, de aceptación, integración, entre otros.
- Convención sobre configuración: esto es muy propio de Ruby on Rails, ya que al seguir convenciones ahorra tiempo que se utilizará en la configuración.
- Juego de la planeación: la rapidez y facilidad que ofrecen los *frameworks* Ruby permiten la realización de prototipos que guíen la elaboración del producto final.
- Desarrollo iterativo: es fundamental en las metodologías ágiles. Conjuntamente con la planeación hay que elegir el orden adecuado de desarrollo de los requerimientos del producto.
- Adaptabilidad: una característica importante es la adaptabilidad a futuros cambios, y esto depende principalmente del diseño, así como la manera en que se desarrolla el aplicativo.

3.4. Estándares y mejores prácticas de Ruby

A diferencia de otros lenguajes como Python, Ruby no tiene una estandarización o guía de estilo al programar. En Python existe la PEP 0008, la cual es una guía que busca que el código sea entendible para los lectores y sobre todo consistente.

En Ruby no existe una guía oficial de estilo, un programador de Ruby, usuario de Github (bbatsov), detectó esta deficiencia y decidió trabajar en una guía de estilo para Ruby; con su esfuerzo y la ayuda de la comunidad crearon una guía de estilo de Ruby.

Posteriormente a la guía de estilo de Ruby, aparece RuboCop. Este es un analizador de estilo y está basado en la guía de estilo de Ruby. Actualmente RuboCop cubre gran parte de las especificaciones definidas en la guía.

Algunos puntos importantes de esta guía se listan a continuación:

- Utilizar la codificación UTF-8.
- Usar dos espacios en blanco por tabulación.
- No usar punto y coma (;) para separar expresiones.
- Evitar definir los métodos en una única línea. Con excepción de los métodos vacíos
- Usar espacios alrededor de operadores (+, -, *, /, etc.). Después de comas (,), dos puntos (:) y punto y coma (;). Usar espacios alrededor de ‘{‘ y antes de ‘}’.
- Usar espacios alrededor del operador de asignación (=).
- No utilizar espacios después de ‘(‘, ‘[‘ o antes de ‘]’, ‘)’.
- No utilizar espacio después del signo de negación ‘!’.
- La tabulación del when debe ir a la misma profundidad del case.
- Evitar utilizar coma después del último parámetro cuando se llama a un método.
- Evitar el uso de continuación de línea ‘\’.
- Alinear los parámetros de la llamada a un método si estos ocupan más de una línea.
- Alinear los elementos de un array si utilizan múltiples líneas.
- Agregar guiones bajos en números grandes para aumentar la legibilidad (1_000_000).
- Limitar la longitud de las líneas a 80 caracteres.
- Terminar cada archivo con una nueva línea (línea en blanco).

- Al definir métodos utilizar paréntesis únicamente cuando existan parámetros, omitir si el método no acepta ningún parámetro.
- Evitar usar *for*, en su lugar utilizar iteradores.
- No utilizar *then* en *if* con múltiples líneas.
- Definir la condición en la misma línea donde se encuentra la palabra reservada *if*.
- Usar '*when* x *then* ..' para los case de una sola línea.
- Usar '*when* x: …' para los case con múltiples líneas .
- No utilizar los operadores *not*, *and* y *or*. Siempre utilizar los operadores '!', '&&' y '||' respectivamente.
- Evitar usar *unless* con *else*.
- Usar *loop* para la creación de ciclos infinitos.
- Utilizar operadores cortos de asignación cuando sea aplicable. (*=, +=, -=, etc.)
- Usar identificadores en inglés.
- Usar la notación snake_case para definir los nombres de símbolos, métodos y variables. Snake case consiste en que todas las letras que conforman el identificador deben ser minúsculas, separando las palabras con guiones bajos (_).
- Usar la notación Camel case para los identificadores clases y módulos.
- Utilizar la notación snake_case para los identificadores de archivos y directorios.
- Utilizar la notación SCREAMING_SNAKE_CASE para identificar constantes. La notación SCREAMING_SNAKE_CASE consiste en que todas las letras que conforman el identificador deben ser mayúsculas, separando las palabras con guiones bajos (_).
- Cuando se definen operadores binarios, definir el nombre del parámetro como *other*.

Estas son solo algunas especificaciones de la guía de estilo de Ruby. Si se desea acceder a la guía completa visitar el siguiente repositorio: https://github.com/bbatsov/ruby-style-guide.

Se hace la aclaración de que esta guía de estilo no es oficial, pero ha tenido el apoyo y aceptación de varios programadores de Ruby.

4. PANORAMA GENERAL DE RUBY

4.1. Comunidad de Ruby

Detrás de Ruby existe una gran comunidad dispuesta a trabajar por el crecimiento del lenguaje. La comunidad de Ruby está abierta a toda persona que desea formar parte de ella. La comunidad ha mostrado ser bastante atenta a programadores que quieran iniciarse en el lenguaje, motivo por el cual se puede encontrar variedad de información y tutoriales para iniciar en Ruby.

Existen muchos canales de comunicación para estar en contacto con la comunidad de Ruby. El sitio oficial de Ruby enumera seis diferentes maneras para establecer contacto con otros desarrolladores de Ruby, estos son:

- Grupos de usuarios
- Listas de correo
- Cana IRC de Ruby
- Core de Ruby
- Blogs
- Conferencias

4.1.1. Grupos de usuarios

“En la comunidad de desarrolladores, los grupos de usuarios forman redes de soporte para las personas interesadas en ciertos temas. Son un muy buen lugar para mejorar tus habilidades y establecer contacto con otros

desarrolladores. Los grupos de usuarios son informales y su estructura varía entre ellos. Cualquiera puede armar su propio grupo y establecer sus reglas[6]".

Los grupos de usuarios Ruby son perfectos para encontrar gente en el país que utiliza este lenguaje. Estos grupos suelen tener sus propias maneras de comunicarse con sus miembros: redes sociales, sitios web, listas de correos, reuniones mensuales, etc. Estos suelen limitarse geográficamente, ya que muchos realizan reuniones cada cierto tiempo.

Entre los países con grupos de usuarios de Ruby están:

- Ruby Argentina: esta comunidad es una de las más grandes en Latinoamérica. Tienen varios canales de comunicación, entre ellos esta Twitter, sitio web, listas de correos, entre otros. Organizaron el RubyConf Argentina 2014 y 2013.
- Ruby Uruguay: existe un grupo que organiza y realiza reuniones mensualmente, estas están abiertas a cualquier persona interesada en Ruby. En Uruguay al igual que Argentina, se realiza la RubyConf Uruguay.
- Ruby Brazil: al igual que en Uruguay y Argentina, se organiza el evento RubyConf Brazil.
- Ruby Minnesota: con nombre Ruby Users of Minnesota (RUM).
- Ruby Australia: separada en diferentes grupos independientes para la realización de reuniones.
- Ruby Boston: grupo Ruby de Boston. Cuenta con varios canales de comunicación: Twitter, listas de correos y reuniones mensuales.

[6] RUBY. https://www.ruby-lang.org/es/community/. Consulta: 15 de mayo de 2015.

- Ruby Mexico: en México existe una comunidad dedicada a Ruby on Rails. También existen comunidades dedicadas puramente al lenguaje Ruby, como Ruby GDL (Ruby Guadalajara).
- Ruby London: comunidad llamada London Ruby User Group (LRUG).
- Ruby San Diego: comunidad llamada San Diegos's Ruby Community (SD Ruby).
- Ruby Berlin: comunidad llamada Ruby User Group Berlin (RUG::B).
- Ruby Guatemala: en la red social Facebook se puede encontrar un grupo dedicado al lenguaje de programación Ruby.

En la figura 22 se muestra un mapa con reuniones Ruby organizadas alrededor del mundo.

Figura 22. **Reuniones Ruby en MeetUp**

Fuente: MeetUp. https://www.meetup.com. Consulta: abril de 2015.

4.1.2. Listas de correos

"La suscripción a las listas de correo es una buena manera de llevarle el pulso a la comunidad Ruby.[7]"

Las listas de correos son perfectas para estar informado acerca de las novedades, cambios y tendencias del lenguaje, también nuevas herramientas y estándares. Para inscribirse a una lista de correo basta con llenar un formulario bastante simple en la página oficial de Ruby.

En Ruby existen cuatro listas de correos principales, en todas se utiliza el lenguaje inglés, y estas son:

- Ruby-Talk: es la lista de correos más popular. Trata temas en general sobre Ruby.
- Ruby-Core: trata temas del núcleo y de implementación sobre Ruby, a veces es utilizada para evaluar parches.
- Ruby-Doc: aquí se discuten estándares y herramientas para la documentación de Ruby.
- Ruby-CVS: en esta lista se anuncian todos los commits al código en el repositorio Subversion de Ruby.

Las listas de correos, también son utilizadas como medio de comunicación en los diferentes grupos de Ruby, ya que resulta un medio idóneo para mantener informados a los miembros sobre actividades de la comunidad.

[7] RUBY. https://www.ruby-lang.org/es/community/. Consulta: 25 de mayo 2015.

4.1.3. Ruby en IRC

"El canal IRC The Ruby Language es un buen lugar para chatear con otros compañeros Rubyistas"[8].

IRC (Internet Relay Chat) es un protocolo de comunicación que facilita el envío de mensajes de texto en tiempo real entre dos o más personas. La gran ventaja de estos canales es que los usuarios no necesitan establecer comunicación directa entre ellos y los mensajes llegan a todos los que se encuentran activos en el canal.

El funcionamiento de estos canales inicia con los usuarios conectados a aplicaciones clientes, los clientes se conectan al servidor, el cual se encarga de gestionar los diferentes canales y conversaciones que pudieren existir.

Los canales IRC son utilizados por muchas comunidades. Ruby tiene un canal oficial identificado como #ruby-lang. También muchos grupos de usuarios Ruby tienen sus propios canales IRC.

4.1.4. El core de Ruby

"Es un momento fantástico para seguir el desarrollo de Ruby. Con la creciente atención que ha recibido Ruby en los años pasados, existe la gran necesidad de talentos para ayudar al mejoramiento y documentación de sus partes"[9].

[8] RUBY. https://www.ruby-lang.org/es/community/. Consulta: 25 de mayo de 2015.
[9] COMUNIDAD RUBY. www.rubygems.org. Consulta: 20 de abril d 2015.

Ruby pone a disposición su código a toda persona que quiera acceder a él, y así puedan estudiarlo, entenderlo mejor e incluso participar en el desarrollo y mejoramiento de Ruby.

Para acceder al código fuente de Ruby, basta con la realización de un *checkout* del repositorio oficial. La herramienta de versionado utilizada es Subversion. También existen repositorios espejos en GitHub, lo que permitiría utilizar Git para obtener el código fuente.

Ruby ofrece en su página oficial los pasos para colaborar efectivamente en el desarrollo y mejoramiento de Ruby, también algunas reglas para mantener el orden en el código y documentación del lenguaje.

4.1.5. Blogs

"Los blogs de Ruby se han disparado en los últimos años y si se busca lo suficiente, se pueden descubrir cientos de blogs que comparten trozos de código Ruby, que describen nuevas técnicas, o especulaciones sobre el futuro de Ruby"[10].

Existen gran variedad de blogs destinados a Ruby. Los más conocidos por su frecuencia y contenido son:

- O'Really Ruby
- Riding Rails
- Ruby Inside
- Matz' Blog: este pertenece al creador de Ruby y se encuentra escrito en japonés.

[10] RUBY. https://www.ruby-lang.org/es/community/. Consulta: 25 de mayo de 2015.

4.1.6. Conferencias sobre Ruby

"Los desarrolladores Ruby de todo el mundo se están involucrando cada vez en más conferencias, donde se juntan para compartir sus experiencias en sus desarrollos, discutir sobre el futuro de Ruby, y dar una bienvenida a los recién llegados a la comunidad Ruby"[11].

Se organizan varias conferencias acerca de Ruby alrededor del mundo. A continuación, se listan las principales conferencias de Ruby:

- RubyConf: conferencia anual, desde 2001 Ruby Central, Inc. ha organizado esta conferencia, la cual tiene carácter internacional, y suelen presentarse proyectos Ruby de la mano de sus creadores.
- RubyKaigi: conferencia anual, desde 2006 realizada en Japón.
- EuRuKo: conferencia anual, desde 2003 realizada en Europa, esta conferencia es organizada por un equipo de Rubyists alemanes.

También a nivel regional se realizan conferencias organizadas por las comunidades de la región. Ejemplo de ello son las conferencias RubyConf Brazil, RubyConf Uruguay y RubyConf Argetina.

http://rubyconferences.org/ es una página web con algunas de las conferencias que se realizaron o se realizarán.

4.2. Aplicaciones en Ruby

Ruby es un lenguaje de propósito general. Por lo que se puede utilizar en gran variedad de situaciones y aplicaciones.

[11] RUBY. https://www.ruby-lang.org/es/community/. Consulta: 25 de mayo de 2015.

4.2.1. Google SketchUp

Es un software utilizado para crear modelos 3D de lo que se quiera. Con su API Ruby se puede ampliar y personalizar el programa para satisfacer las necesidades del usuario.

Es una manera que permite a los programadores Ruby extender las capacidades de SketchUp para satisfacer sus necesidades. Las extensiones del programa se logran a través de *scripts* escrito en Ruby y puestos en la carpeta de plugins. Con esto se puede hacer que sketchUp haga todo tipo de cosas:

- Crear herramientas de dibujo personalizadas
- Adherir atributos a elementos de dibujo
- Automatizar tareas comunes
- Generar informes a partir de ciertos atributos
- Realizar animaciones
- Hacer juegos dentro de SketchUp

En Google SketchUp se hace diferencia entre *script* y extensión. Un *script* es un archivo de texto (.rb o .rbs) que contiene código Ruby. Una extensión es un *script* con la diferencia que realiza ciertas llamadas extra que le comunican a SketchUp que se trata de una extensión, lo que permite habilitarlo o deshabilitarlo en cualquier momento.

En esta herramienta Ruby es utilizado para desarrollar *scripts*, y que estos trabajen conjuntamente con Google SketchUp.

4.2.2. Sitios web

El *framework* Ruby on Rails ayudó enormemente a la expansión del lenguaje Ruby, ya que las diferentes novedades, velocidad y su filosofía “Convención sobre configuración” atrajeron a muchos desarrolladores web. Motivo por el cual Ruby es ampliamente utilizado en entornos web.

Algunos sitios desarrollados utilizando Ruby son:

- Twitch.tv: plataforma para gamers con el motivo de compartir videos: (*gameplays*, torneos, entre otros).
- Basecamp: es una aplicación desarrollada en Ruby on Rails que ofrece herramientas de gestión de proyectos.
- Lumosity: plataforma con juegos web y móviles orientados a desarrollar habilidades cognitivas.
- Imgur: provee servicio para alojar imágenes y posteriormente compartirlas en internet.
- Goodreads: sitio para los amantes de la lectura, ya que permite descubrir nuevos libros a partir de las recomendaciones de otros lectores.
- Braintree: provee una manera rápida de pagar a través de cualquier dispositivo.
- SoundCloud: es una plataforma donde cualquiera puede crear y compartir sonidos y música.
- Heroku: plataforma en la nube que provee soporte a diferentes lenguajes de programación.
- Hulu: es una plataforma que ofrece un servicio de video con diferentes shows, clips y películas.
- Github: plataforma que permite alojar repositorios utilizando la herramienta de versionado Git.

- Slideshare: plataforma que aloja presentaciones de diapositivas.

4.3. Uso de Ruby en universidades

Hay universidades alrededor del mundo que tienen cursos o material educativo acerca de Ruby. Existen otras que han desarrollado investigaciones y desarrollado software utilizando Ruby. Algunas de estas universidades se presentan a continuación.

- Shefield Hallam University (SHU) de Reino Unido presenta un interesante artículo acerca de una extensión para Ruby denominada HornetsEye. HornestsEye es una extensión que facilita el desarrollo de algoritmos de visión artificial en tiempo real dentro de Ruby, esta extensión también proporciona integración con bibliotecas importantes de entrada y salida.

- La Universidad de Washington ofrece 3 cursos dedicado a Ruby: “Ruby: The Core Language”, “Applications with Ruby on Rails” y Advanced Topics in Ruby on Rails”. El primer curso introduce al estudiante a los conceptos fundamentales de la programación en Ruby, excepciones, iteradores, parseo de texto, las RubyGems, entre otros. El segundo curso los estudiantes aprenden cómo diseñar, desarrollar, probar y desplegar aplicaciones con la plataforma Rails; los estudiantes trabajan en grupos pequeños grandes aplicaciones web definidas por el instructor. El último curso está enfocado a funcionalidades avanzadas de Ruby, y además experimentan con otros *frameworks* web de Ruby. Estos tres cursos son complementarios, es decir, para acceder al segundo curso previamente se debe aprobar el primer curso y así sucesivamente. Al final de esta serie de cursos el estudiante cuenta con un portafolio de aplicaciones

que demuestran sus habilidades en Ruby y Ruby on Rails. Actualmente esta serie de cursos se sigue impartiendo.

- La Universidad de Oxford ofrece el curso “Web Applications” en el cual está enfocado totalmente a las tecnologías Web. Se inicia con temas básicos de la web como HTML y CSS terminando con temas complejos como la seguridad y *datacenters*. Dentro de este curso, alrededor de la segunda y cuarta semana se estudia el lenguaje Ruby conjuntamente con el *framework* Ruby on Rails. Este curso no se encuentra enfocado solamente a Ruby, ya que se trata temas de la web en general. En la actualidad este curso se continúa impartiendo.

- La UCL (University College London) ofrece un curso dedicado al lenguaje Ruby. Este abarcaba los fundamentos del lenguaje, así como comparaciones con lenguajes como Perl y Java. También trataba temas como Ruby en la web incluyendo el *framework* Ruby on Rails. Terminando con manejo de bases de datos con Ruby y la creación de interfaces gráficas. Este curso fue impartido en el 2012, actualmente esta universidad no tiene en vigencia ningún curso orientado a Ruby.

- También la Universidad de Berkeley conjuntamente con la plataforma MOOC (Massive Online Open Course) Edx ofrecieron el curso Engineering Software as a Service usando Ruby. El curso inició en 2014, actualmente la plataforma edx no cuenta con curso de Ruby.

Pero si se desea aprender Ruby existe gran variedad de material disponible en la web, a través de distintas plataformas como:

- Coderbyte (http://www.coderbyte.com)

- GreeRuby (http://greenruby.org)
- RubyMonk (http://rubymonk.com/)
- Rails for Zombies (http://railsforzombies.org/)
- CodeAcademy (http://www.codecademy.com/tracks/ruby)
- CodeQuizzes (http://www.codequizzes.com)
- CodeWars (http://www.codewars.com/)
- Confreaks (http://confreaks.com/)
- Coderwave (http://coderwave.com/courses/)
- TryRuby (http://tryruby.org)

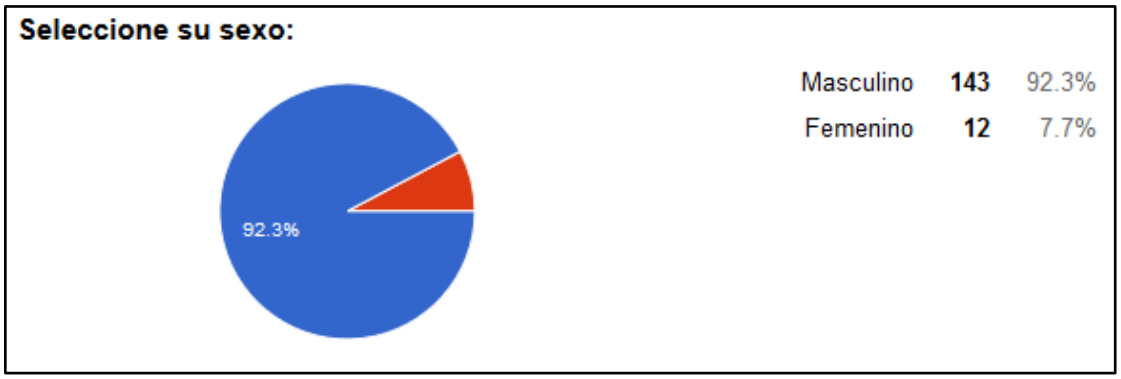
4.4. Encuesta de la Universidad de San Carlos de Guatemala

Buscando tener una idea de la situación de Ruby en Guatemala, se realizó una encuesta a los alumnos de la carrera de Ingeniería en Ciencias y Sistemas, de la Facultad de Ingeniería, de la Universidad de San Carlos de Guatemala. La encuesta se realizó por medio de la herramienta Google forms, y fue distribuida gracias a la colaboración de la Escuela de Ingeniería en Ciencias y Sistemas. Se procede a presentar los resultados obtenidos en esta encuesta.

Las personas que realizaron la encuesta son estudiantes o profesionales, cuya edad promedio es de 24 años, la gran mayoría son hombres, y una gran cantidad de encuestados se encuentra entre tercero y quinto año de la carrera. Esto se ilustra con las figuras 23 y 24.

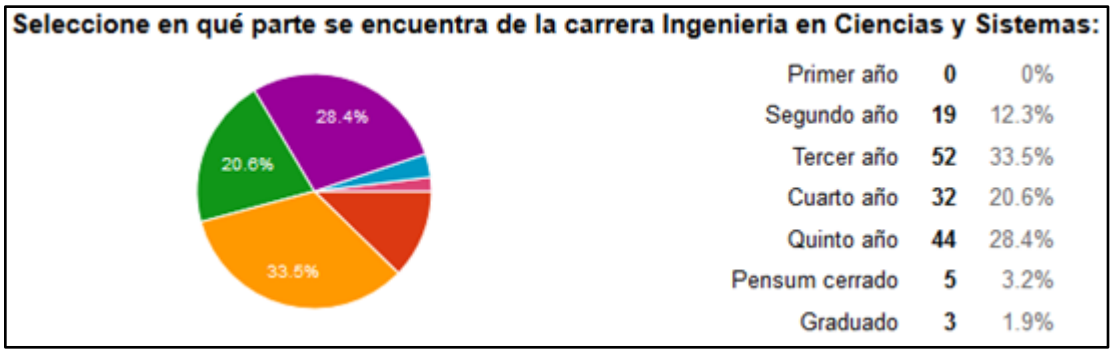
Posteriormente, los encuestados respondieron preguntas acerca su conocimiento y preferencias entre varios lenguajes de programación. Finalizando esta fase con una pregunta directa acerca de su conocimiento sobre el lenguaje de programación Ruby.

Figura 23. **Género de encuestados**



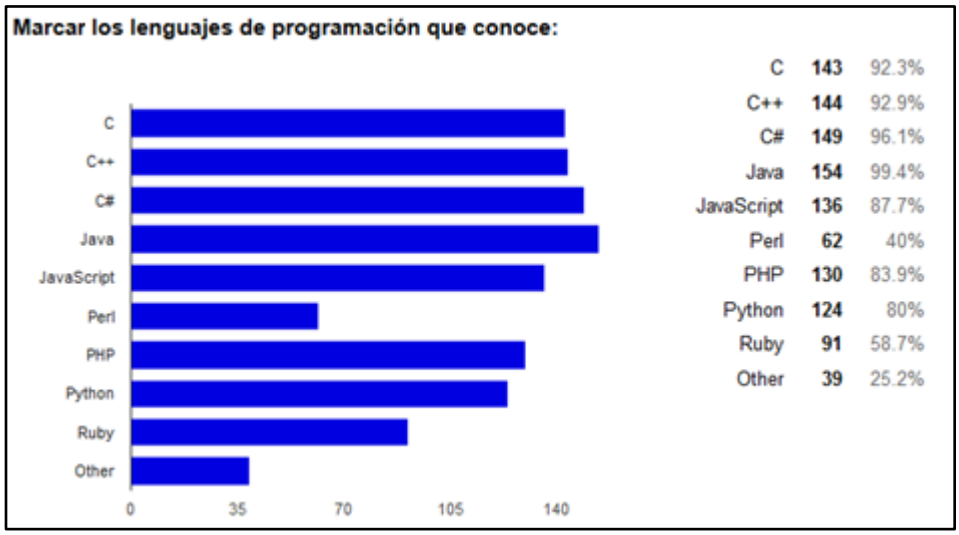
Fuente: elaboración propia, empleando Google forms.

Figura 24. **Nivel académico de los encuestados**



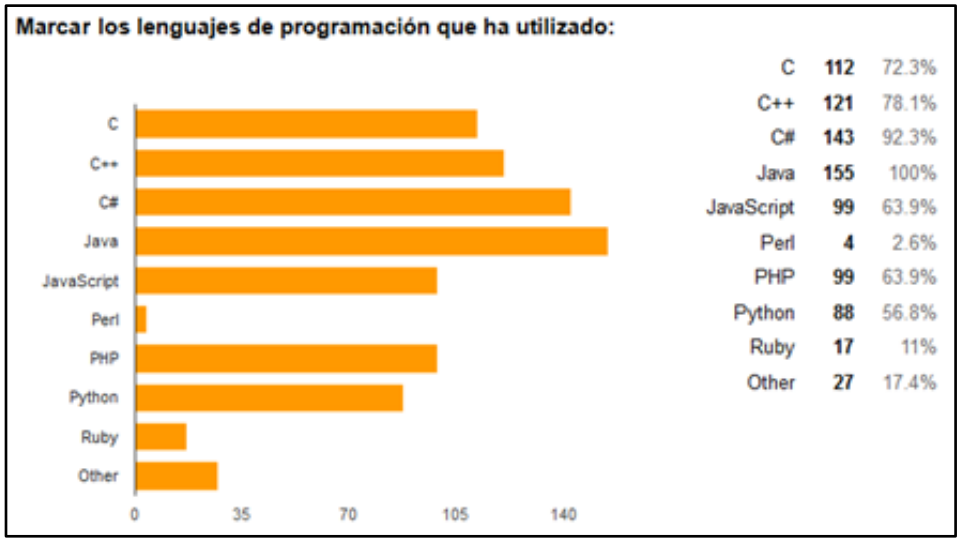
Fuente: elaboración propia, empleando Google forms.

Figura 25. **Lenguajes de programación que conoce**



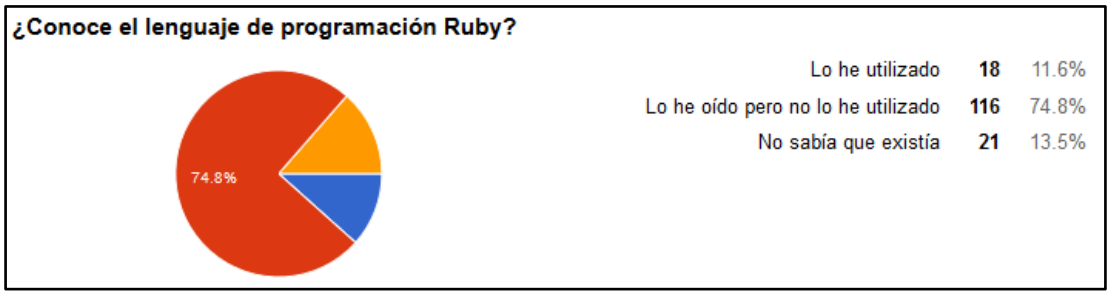
Fuente: elaboración propia, empleando Google forms.

Figura 26. **Lenguajes de programación que ha utilizado**



Fuente: elaboración propia, empleando Google forms.

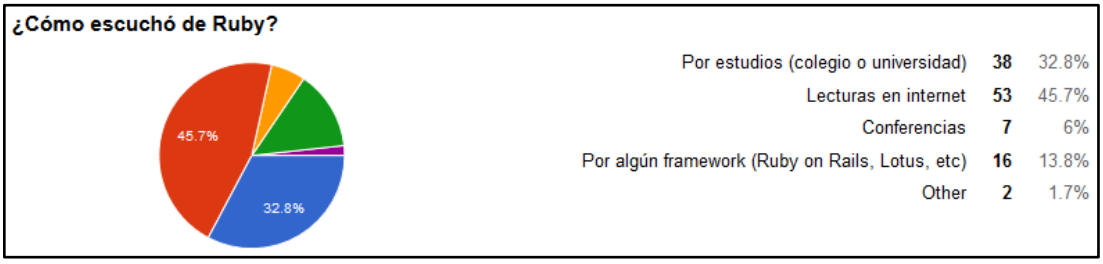
Figura 27. **¿Conoce Ruby?**



Fuente: elaboración propia, empleando Google forms.

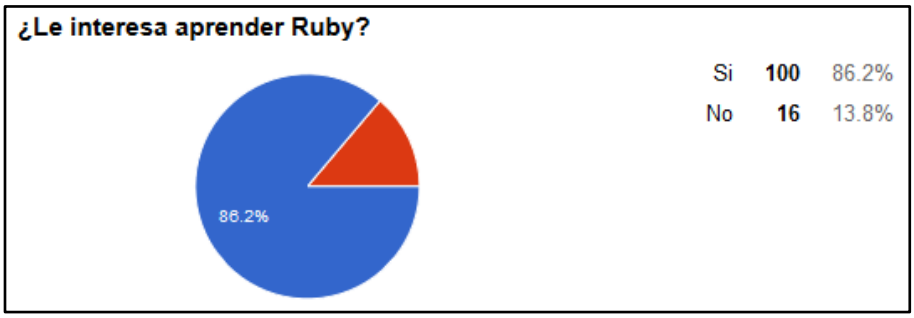
A la última pregunta de la fase anterior, la mayoría contestó que no había utilizado el lenguaje, pero sí lo había escuchado. A estas personas se les hicieron dos preguntas. La primera iba dirigida a cómo se enteró el entrevistado del lenguaje Ruby, y la segunda cuestionaba sobre la intención de aprender el lenguaje. Lecturas en internet y estudios fueron las respuestas predominantes en la primera pregunta de esta fase, mientras que la gran mayoría mostró interés en el aprendizaje de este lenguaje en la segunda pregunta.

Figura 28. **¿Cómo escuchó de Ruby?**



Fuente: elaboración propia, empleando Google forms.

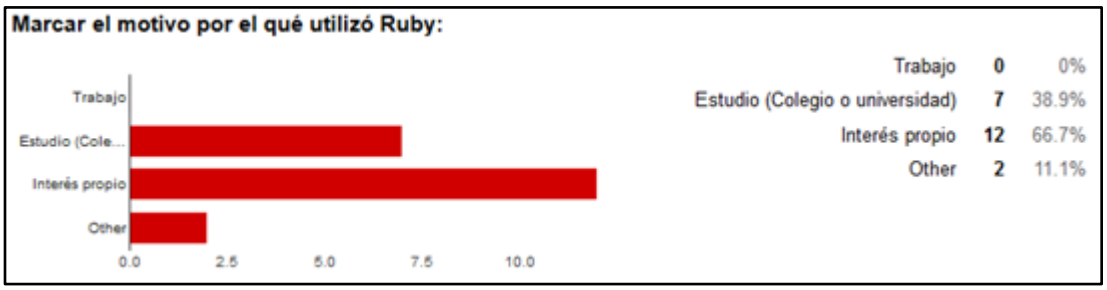
Figura 29. **¿Le interesa aprender Ruby?**



Fuente: elaboración propia, empleando Google forms.

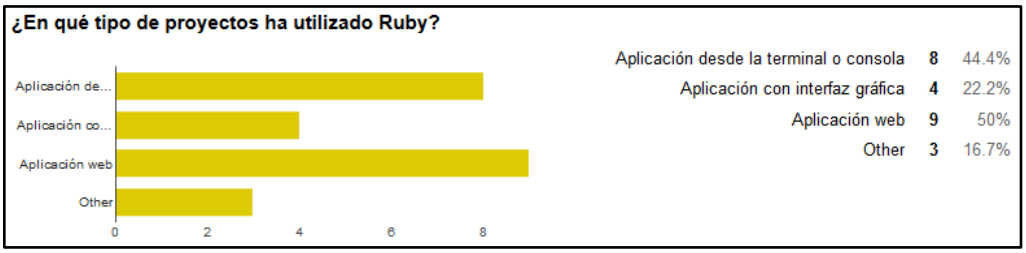
A las pocas personas que habían utilizado el lenguaje se les preguntó acerca de su experiencia con Ruby. Respondieron como se muestra en las figuras 30, 31, 32 y 33.

Figura 30. **Motivo por el que utilizó Ruby**



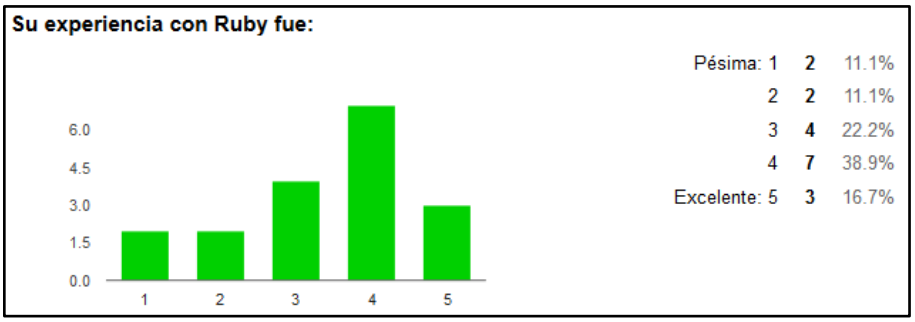
Fuente: elaboración propia, empleando Google forms.

Figura 31. **Tipo de proyecto en el que utilizó Ruby**



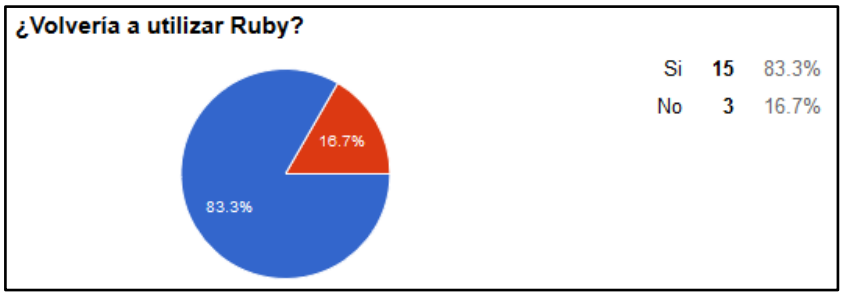
Fuente: elaboración propia, empleando Google forms.

Figura 32. **Evalúe su experiencia con Ruby**



Fuente: elaboración propia, empleando Google forms.

Figura 33. **¿Volvería a utilizar Ruby?**



Fuente: elaboración propia, empleando Google forms.

La encuesta evidenció poco conocimiento del lenguaje comparado a lenguajes más populares como Java y Javascript, esto se debe a que, a lo largo de la carrera, Java es el lenguaje más utilizado. También se observa que las personas que utilizaron el lenguaje la gran mayoría fue para aplicaciones de consola y web. Aunque también hay que considerar que la mayoría de personas que mencionaron haber escuchado sobre el lenguaje Ruby, mostraron interés en aprenderlo, pero esto depende de las necesidades y planes de cada persona.

4.5. Panorama nacional e internacional de Ruby

Otra manera de visualizar el panorama actual de Ruby, es por medio del volumen de búsquedas que se realizan diariamente sobre el lenguaje. Utilizando la herramienta Google Trends, se analiza el volumen de búsquedas y compara con otros lenguajes.

4.5.1. Internacional

Ruby es un lenguaje que se originó en Japón y se fue extendiendo al resto del mundo.

La figura 34 es una gráfica sobre las búsquedas realizadas con respecto al término Ruby como lenguaje de programación en el buscador de Google, sin restricción de territorio alguno.

Asimismo, se observa cierta estabilidad de 2004 hasta abril de 2005, luego inicia un incremento llegando a su punto máximo en junio de 2007. Posteriormente se ve un descenso significativo, teniendo 2 puntos altos

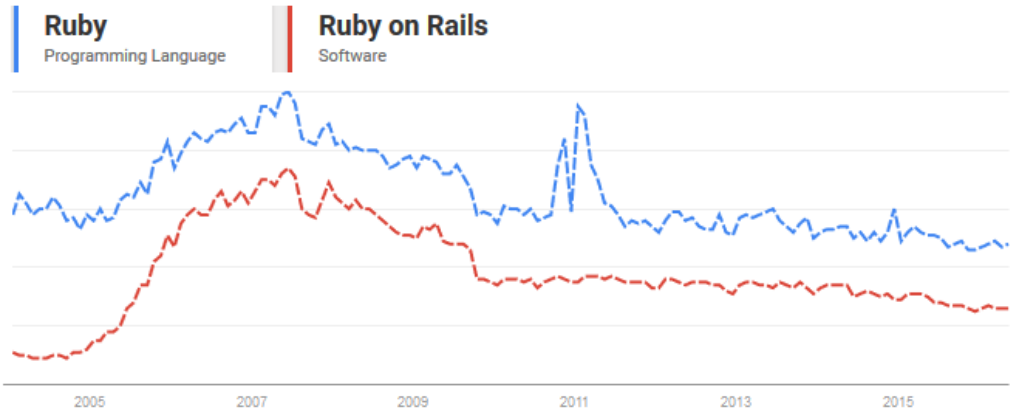
correspondientes a noviembre 2010 y febrero 2011. A partir de agosto de 2011 se puede observar cierta estabilización en el volumen de búsquedas.

Figura 34. **Ruby a lo largo del tiempo**

Fuente: Google Trends. Consulta: noviembre de 2015.

Actualmente, el volumen de búsquedas de Ruby ha disminuido si se compara con el 2007, pero se aprecia estabilidad en las mismas. Lo que significa que Ruby sigue siendo un lenguaje de interés

Figura 35. **Ruby on Rails a lo largo del tiempo**

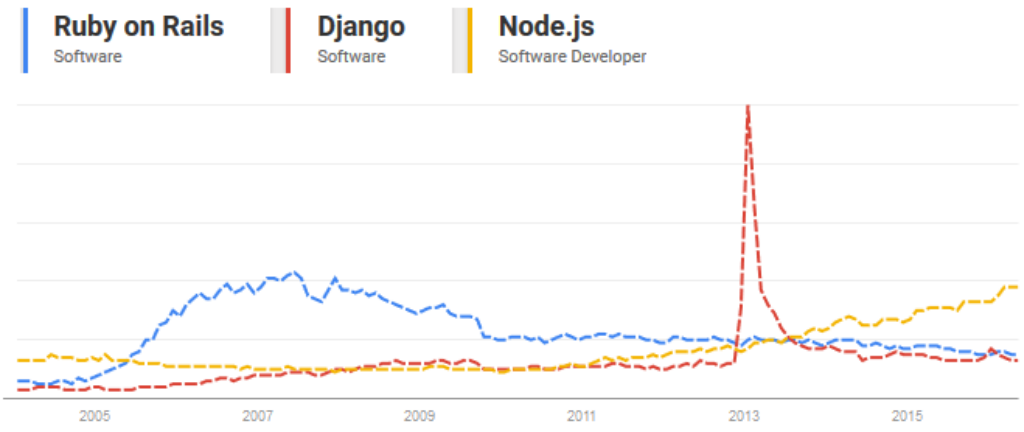


Fuente: Google Trends. Consulta: noviembre de 2015.

En la figura 35 se observan las búsquedas realizadas sobre Ruby (azul) y sobre Ruby on Rails en rojo, siempre utilizando el motor de búsqueda Google. Se puede apreciar como el *framework* Ruby on Rails afectó directamente al crecimiento del lenguaje, ya que mientras aumentaba el interés en el *framework*, esto se veía reflejado en el interés sobre el lenguaje. Se observa también que cuando el *framework* perdió popularidad, el lenguaje también se vio afectado.

Tomando en cuenta que uno de los ámbitos donde se utiliza comúnmente Ruby es el ambiente web, además el *framework* con mayor cantidad de usuarios para el desarrollo web en Ruby es Ruby on Rails, RoR. Se explica la similitud de las gráficas en sus volúmenes de búsquedas. Como explicó anteriormente, muchos usuarios se iniciaron en Ruby, porque deseaban utilizar el *framework* Ruby on Rails.

Figura 36. ***Frameworks* web**

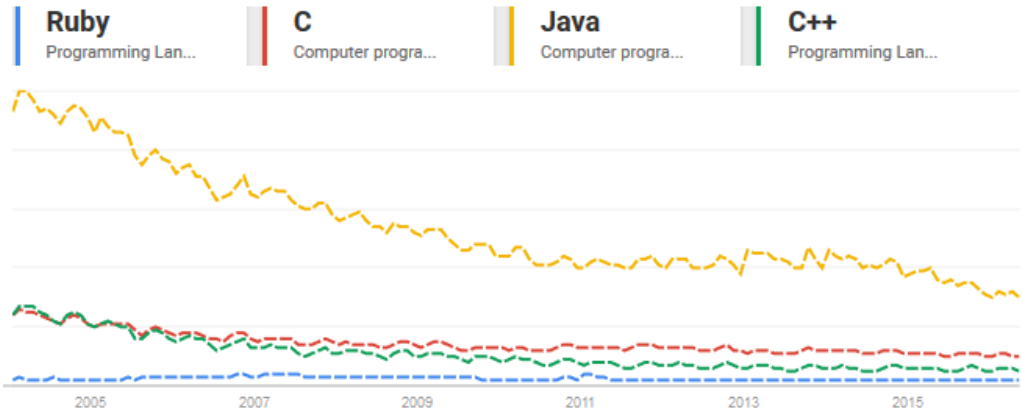


Fuente: Google Trends. Consulta: noviembre de 2015.

En la figura 38 se observa Ruby on Rails (azul), Node.js (amarillo) y Django (rojo). Correspondientes a los lenguajes Ruby, JavaScript y Python respectivamente. En la gráfica se logra apreciar un crecimiento lento para Django hasta 2013 donde inicia su descenso, un crecimiento constante para Node.js, y por último, un descenso del uso de Ruby on Rails. Tanto Django como Ruby on Rails muestran ligeros descensos, solamente Node.js muestra una pendiente positiva pronunciada.

Como se puede apreciar, el descenso en popularidad de Ruby on Rails se encuentra en proceso de estabilización. Como ya se mencionó anteriormente existen muchos sitios desarrollados en este *framework*, por lo que resulta necesario darles mantenimiento a los mismos. Motivo por el cual utilizar el *framework* Ruby on Rails resulta necesario, además debido a las facilidad y rapidez del *framework* sigue siendo utilizado para la realización de nuevos proyectos.

Figura 37. **Lenguajes de programación**



Fuente: Google Trends. Consulta: noviembre de 2015.

En la figura 37 se logra observar a Ruby (azul), Java (amarillo), C (rojo) y C++ (verde). Se muestra como Ruby, a pesar de sus incrementos de popularidad, nunca se igualó en el volumen de búsquedas a lenguajes conocidos y utilizados ampliamente en la industria, como Java, C y C++.

Figura 38. **Mapa Ruby**

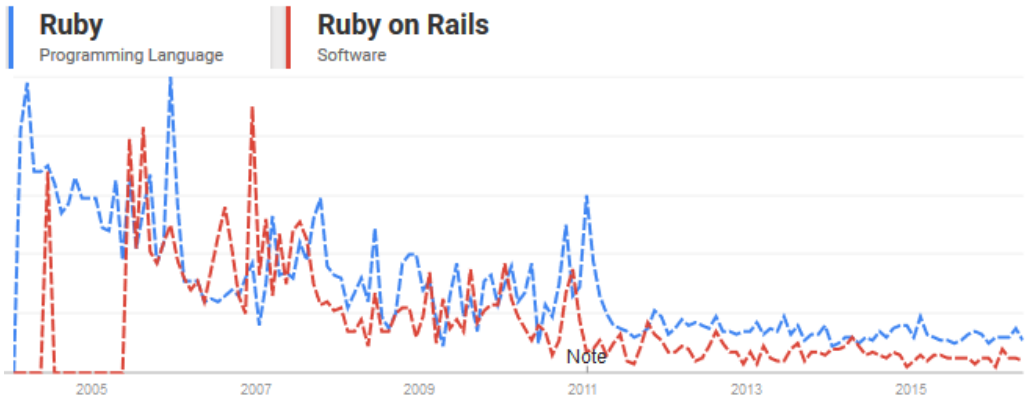
Fuente: Google Trends. Consulta: noviembre de 2015.

En el mapa de la figura 38 se pueden observar los países donde se realizan más búsquedas con respecto al lenguaje de programación Ruby. Como es lógico imaginar, Ruby tiene mayor popularidad en su país de origen, Japón, seguido por Nueva Zelanda, Rumania y posteriormente Estados Unidos.

4.5.2. Nacional

El volumen de búsquedas referentes a Ruby en Guatemala, resultan despreciables si se compara con países como Japón o Estados Unidos. Pero es interesante observar el comportamiento de los volúmenes de búsquedas en el territorio nacional.

Figura 39. **Ruby en Guatemala**



Fuente: Google Trends. Consulta: noviembre de 2015.

Desde el 2011 se observa cierta estabilización y similitud con la gráfica internacional. También al igual que en la gráfica internacional, existe similitud entre los volúmenes de búsqueda correspondientes a Ruby on Rails, y Ruby como lenguaje.

5. VENTAJAS, CUÁNDO UTILIZARLO Y QUÉ ESPERAR DE RUBY

5.1. Ventajas y desventajas

Como todo lenguaje Ruby ofrece ventajas y desventajas al programador. Conocer los aspectos positivos y negativos de un lenguaje, permite aprovechar al máximo todas las bondades que el lenguaje provee.

5.1.1. Ventajas

Ruby es un lenguaje de programación sencillo de aprender en comparación de lenguajes como Java o C++, ya que su expresividad resulta bastante natural para el programador al escribir o leer código.

Ruby permite muchas maneras de hacer una misma acción. Esto da libertad al programador a elegir cómo desea realizar cada acción, evita limitar al programador y permite que haga las cosas de la manera en la que se sienta más cómodo. Ruby proporciona varias maneras de realizar una misma tarea.

Hacer más con menos código, este es otro punto de su filosofía. Ruby busca que el programador gaste su tiempo en la resolución del problema, no en tareas triviales que conforman la solución, pero no son la solución en sí. Ejemplo de esto es la lectura y escritura de archivos, en Ruby esta tarea es bastante simple y resulta sencillo para el programador realizarla.

Ruby cuenta con una gran cantidad de librerías externas que impulsan y facilitan el trabajar en Ruby. Esto se logra apreciar más en proyectos web, ya que la variedad de librerías y herramientas para proyectos web es sencillamente sorprendente

Ruby cuenta con una comunidad relativamente grande, la cual es bastante amigable con los nuevos usuarios. Encontrar guías o tutoriales donde iniciarse en Ruby es bastante sencillo, además de ser de gran calidad y bastante interactivo. La comunidad es bastante activa y organiza variedad de eventos a lo largo de los años para todas las personas interesadas en el lenguaje.

Asimismo, es de código abierto, por lo que es posible conocer, estudiar y aportar al proyecto. El código de Ruby está a disposición de quien lo desee, como se comentó anteriormente, solo es necesario realizar un *checkout* de su repositorio oficial, utiliza el controlador de versiones Subversion, o de alguno de los espejos, utiliza Git.

Además, es multiplataforma, ya que solo se necesita tener instalada la versión de la máquina virtual de Ruby o de alguna de las implementaciones de Ruby como JRuby o Rubinius.

5.1.2. Desventajas

Aunque se mencionó la característica de que Ruby permite realizar una misma acción de diferentes maneras como una ventaja, también es posible mencionarla como desventaja, ya que la libertad que ofrece Ruby al programador permite que este escriba código no tan entendible y natural. Esto se complica a la hora de dar mantenimiento al código, ya que, si no se definieron estándares, probablemente resulte bastante difícil realizar un cambio.

Lento en procesamiento: Ruby al ser un lenguaje interpretado, su rendimiento se ve superado por lenguajes compilados. Además, la máquina virtual de Ruby no se puede comparar con la máquina virtual de Java, ya que esta última lleva años en desarrollo y ha solucionado y optimizado diferentes aspectos que aún le falta recorrer a la máquina de Ruby.

5.2. Alcances y limitaciones

Conocer los alcances y limitaciones de los lenguajes permite seleccionar el lenguaje correcto para cada proyecto.

5.2.1. Alcances

Ruby, al ser un lenguaje de propósito general tiene un amplio alcance, ya que puede ser utilizado tanto en aplicaciones de escritorio, web o *scripting*.

En aplicaciones de escritorio, Ruby es multiplataforma, por lo que puede ser utilizado en cualquier sistema operativo, siempre y cuando se cuente con la máquina virtual correspondiente. Además, permite realizar aplicaciones gráficas gracias a librerías como GTK+ o wxWidgets.

En el ambiente web, Ruby es bastante conocido, especialmente por el *framework* Ruby on Rails. En la elaboración de sitios web Ruby sobresale por su rapidez y sencillez en el desarrollo. En este ambiente Ruby también sobresale por su gran variedad de librerías, herramientas y *frameworks* disponibles. Con respecto al tamaño de las aplicaciones web, Ruby es perfecto para aplicaciones web pequeñas y aplicaciones grandes como Soundcloud, GitHub o Twitch.tv.

Ruby es un lenguaje de *scripting*, por lo que su presencia en el *scripting* es algo normal.

5.2.2. Limitaciones

Ruby al ser un lenguaje interpretado no tendrá el mismo desempeño que los lenguajes compilados. Además, al ser un lenguaje relativamente joven muestra algunas carencias frente a otras máquinas virtuales como la Java Virtual Machine.

Asimismo, ha sido mayormente explotado en entornos web, por lo que su uso en aplicaciones de escritorio puede verse limitado en funcionalidades, ya que como se mencionó, su área de confort son los ambientes web, lo que hace que Ruby no sea una opción viable en el desarrollo de una aplicación de escritorio.

5.3. Escenarios idóneos para utilizar Ruby

Con base en las ventajas, desventajas, alcances y limitaciones se pueden definir escenarios idóneos donde utilizar Ruby.

5.3.1. Ambiente web

Debido al *framework* Ruby on Rails, Ruby es un lenguaje con grandes fortalezas para ambientes web, ya que, por la gran cantidad de proyectos desarrollados durante el boom de este *framework*, se crearon una vasta variedad de librerías dedicadas a resolver problemas en este ambiente. Problemas como el ruteo, nombramiento de URL, autenticación, entre otros.

Por lo anterior, Ruby ofrece gran variedad de *frameworks* y herramientas para el desarrollo de aplicaciones web. Entre estos *frameworks* se encuentran Lotus y Padrino los cuales están enfocados en el desarrollo rápido y sencillo de aplicaciones web. Entre las herramientas se encuentra SASS, esta herramienta que actualmente resulta fundamental para gran cantidad de desarrolladores web, que no se limitan a desarrolladores Ruby, ya que los beneficios que ofrece en la escritura y posterior mantenimiento de archivos CSS es bastante beneficiosa.

Definitivamente el ambiente web es uno de los puntos donde Ruby goza de mayor fuerza.

5.3.2. *Scripting*

Debido a que Ruby es un lenguaje de *scripting* su beneficio para este tipo de ambientes resulta bastante lógico. Prueba de ello es Google SketchUp donde todos los *plugins* y extensiones son desarrolladas totalmente en Ruby, utilizando una API proporcionada por google para la integración con el software. Google SketchUp es una prueba fehaciente del éxito de Ruby como lenguaje de *scripting*.

Si se toma en cuenta la facilidad para aprender Ruby, resulta más convincente, ya que es un lenguaje idóneo para ser utilizado en ambientes de *scripting*.

5.4. Futuro de Ruby

Se ve amenazado por nuevos lenguajes emergentes como Go-lang de google, ya que ofrecen buen desempeño y una relativa facilidad en el uso del

lenguaje. Pero aún hay personas que prefieren Ruby por la variedad de herramientas web, su facilidad y rapidez en el desarrollo de aplicaciones.

A pesar de que la popularidad de Ruby ha disminuido desde 2011, este se muestra bastante estable en los últimos años. Además, la comunidad sigue mostrándose activa y el soporte propio del lenguaje Ruby aún continúa activo.

Personalmente, Ruby facilitó en su momento la elaboración de sitios web a través de sus diferentes *frameworks*. Motivo por el cual, actualmente existen muchos sitios grandes y pequeños que son potenciados por este lenguaje, por lo que resulta inevitable que Ruby siga siendo utilizado en la para proporcionar mantenimiento a estos sitios. También hay que considerar que Ruby sigue siendo una herramienta seleccionada para la elaboración de proyectos, ya que su sencillez y rapidez son características difíciles de ignorar. Por lo que a corto y mediano plazo Ruby seguirá siendo utilizado.

Solo el tiempo definirá si a largo plazo Ruby seguirá siendo una opción viable o cederá el lugar a nuevos lenguajes emergentes.

CONCLUSIONES

1. Ruby tuvo bastante aceptación en entornos web debido al *framework* Ruby on Rails, motivo por el cual cuenta con gran variedad de herramientas para facilitar el desarrollo de aplicaciones web.

2. Ruby, como lenguaje de propósito general, permite programar aplicaciones tanto de escritorio como web. Pero muestra mayores ventajas en el desarrollo de aplicaciones web.

3. El ambiente web como el *scripting* son los entornos que presentan mayores ventajas para la aplicación de Ruby.

4. Ruby, al ser un lenguaje interpretado, muestra carencias en tiempos de respuesta, en comparación con lenguajes compilados como lo son C, C++, entre otros; pero muestra ventajas en su sencillez en comparación con estos.

RECOMENDACIONES

1. Si no se ha tenido contacto con un lenguaje de programación totalmente orientado a objetos, es conveniente explorar Ruby, ya que este lenguaje es sencillo y es completamente orientado a objetos.

2. Explorar el *framework* Ruby on Rails, ya que permite realizar sitios web de manera rápida y sencilla, ya que los ambientes web son una de las mayores fortalezas de Ruby, debido a la gran cantidad de librerías y herramientas que ofrece.

BIBLIOGRAFÍA

1. BAKER, Mike. *What is a Software Framework and why should you like 'em?.* [en línea]. <http://info.cimetrix.com>. [Consulta: 14 de noviembre de 2015].

2. Comunidad Ruby. *Descargas: Ruby*. *Ruby.* [en línea]. <https://www.ruby-lang.org>. [Consulta: 15 de mayo de 2015].

3. ————**.** *Ruby doc*. [en línea]. <http://www.rubydoc.info>. [Consulta: mayo de 2015].

4. ————**.** *Ruby gems*. [en línea]. <www.rubygems.org>. [Consulta: mayo de 2015].

5. FLANAGAN, David; MATSUMOTO, Yukihiro. *The Ruby Programming Language.* Estados Unidos : O'Reilly Media Inc., 2008. 429 p.

6. GARCÍA GARCÍA, Iván. *Metodologías ágiles en el desarrrollo web.* [en línea]. <http://blog.intelligenia.com/2007/01/metodologas-giles-en-el-desarrollo-web.html>. [Consulta: septiembre de 2015].

7. HERRICK, Jesse*. The History of Ruby.* [en línea]. <http://www.sitepoint.com>. [Consulta: abril de 2015].

8. LOTUS**.** *Lotus*. [en línea]. <http://lotusrb.org/>. [Consulta: 14 de noviembre de 2015].

9. MATSUMOTO, Yukihiro. *Ruby in a nutshell.* Estados Unidos : O'Reilly Media Inc., 2001. 218 p.

10. PADRINORB. *Padrinorb.* [en línea]. <http://www.padrinorb.com/>. [Consulta: 11 de mayo de 2015].

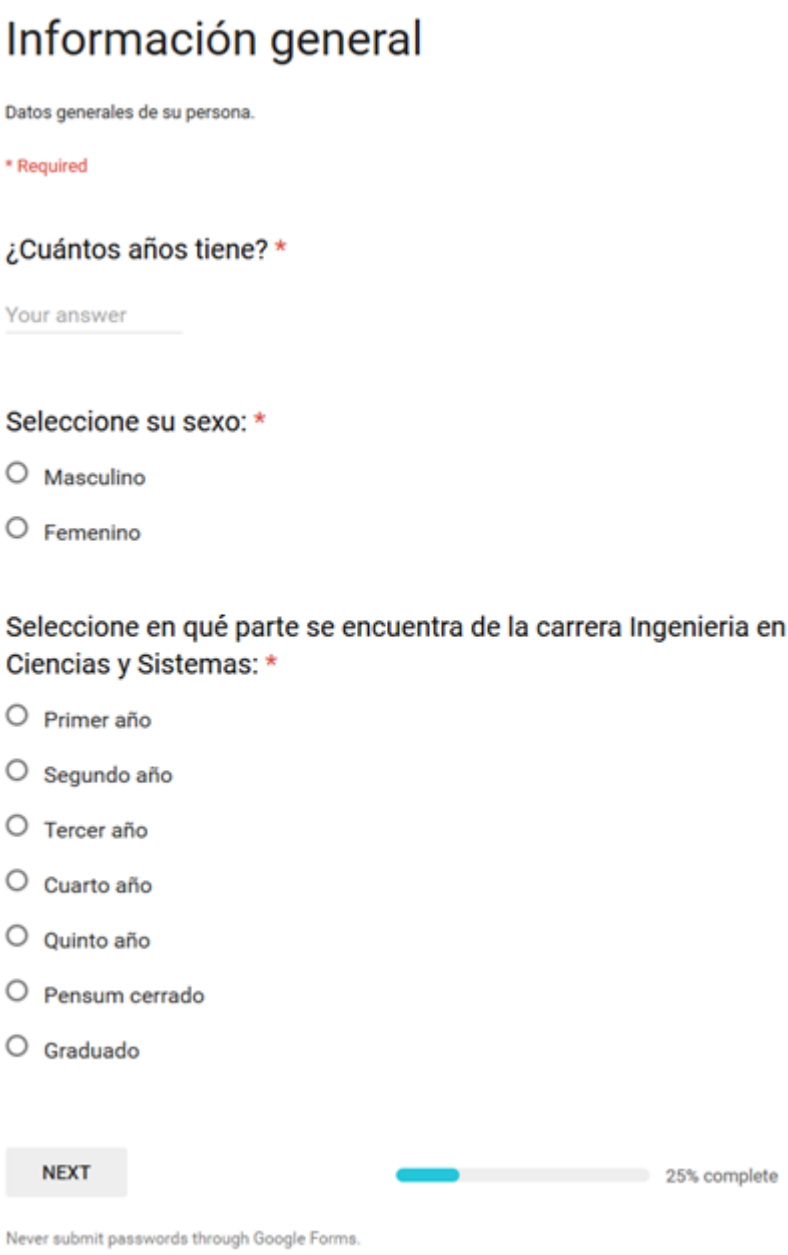
11. RUBY ON RAILS. *Ruby on Rails*. [en línea]. <http://rubyonrails.org/>. [Consulta: 11 de mayo de 2015].

12. SHAUGHNESSY, Patrick. *Ruby Under a Microscope.* Estados Unidos : No starch press, 2014. 362 p.

13. SYMFONY. *Symfony in 5 minutes*. [en línea]. <http://symfony.com>. [Consulta: 14 de noviembre de 2015].

APÉNDICE

Apéndice 1. **Encuesta realizada**



Información general

Datos generales de su persona.

* Required

¿Cuántos años tiene? *

Your answer

Seleccione su sexo: *

- Masculino
- Femenino

Seleccione en qué parte se encuentra de la carrera Ingenieria en Ciencias y Sistemas: *

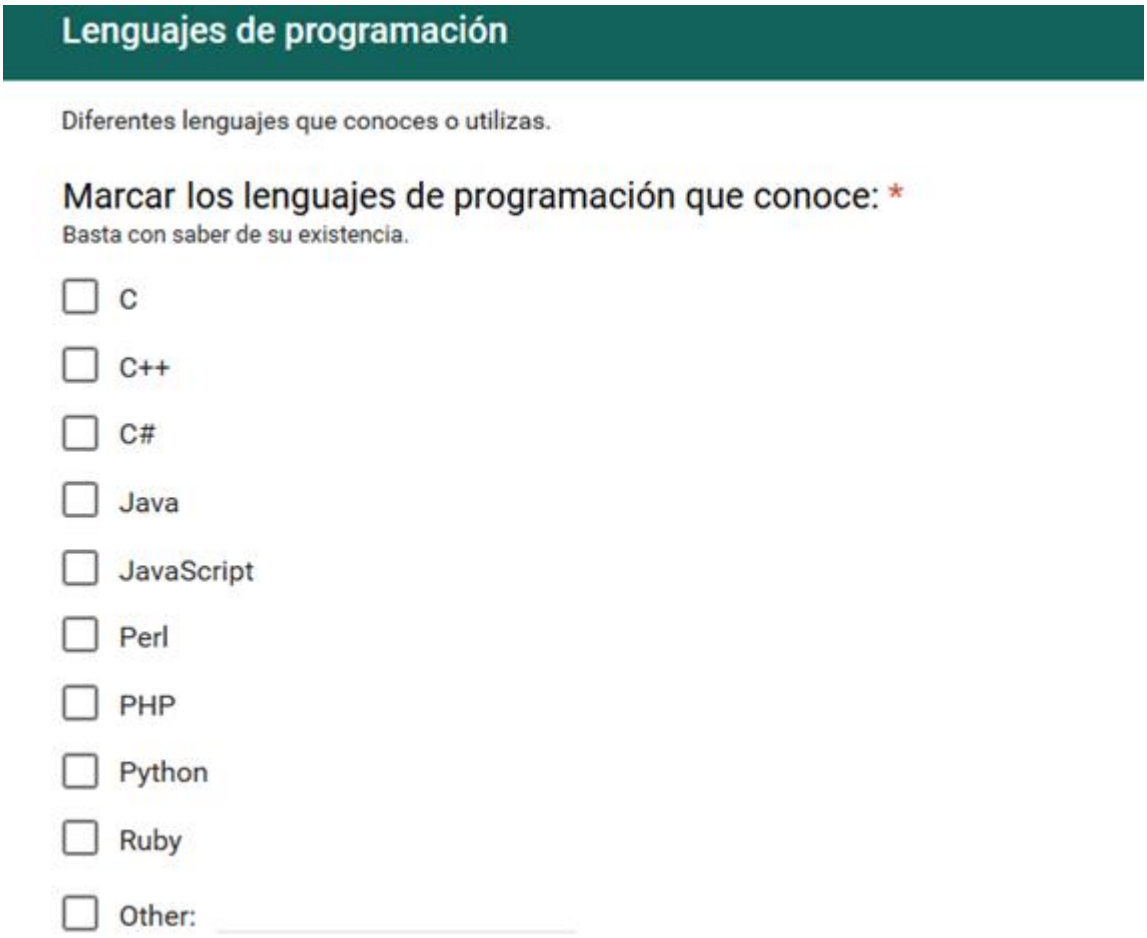
- Primer año
- Segundo año
- Tercer año
- Cuarto año
- Quinto año
- Pensum cerrado
- Graduado

NEXT

25% complete

Never submit passwords through Google Forms.

Continuación apéndice 1.



Lenguajes de programación

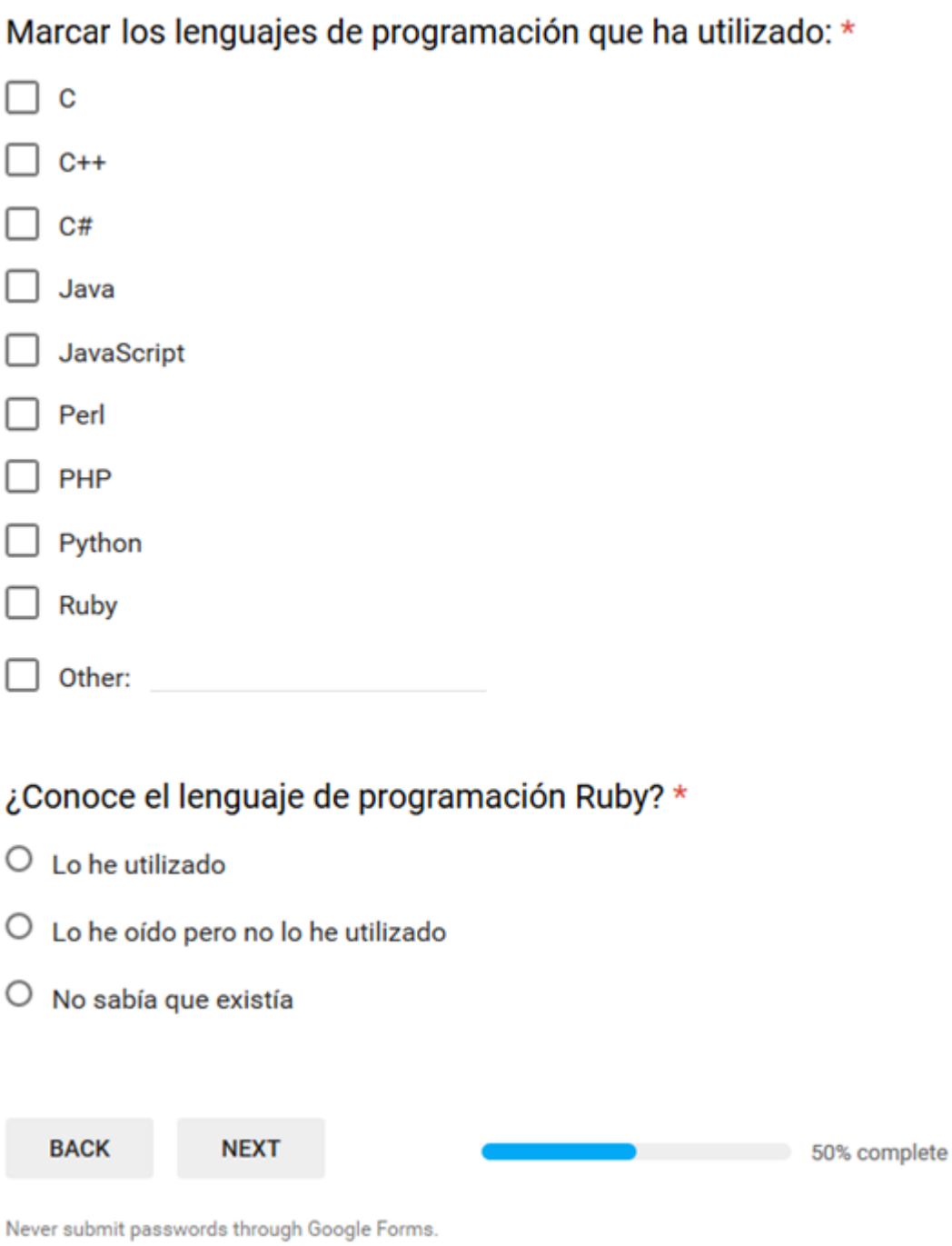
Diferentes lenguajes que conoces o utilizas.

Marcar los lenguajes de programación que conoce: *

Basta con saber de su existencia.

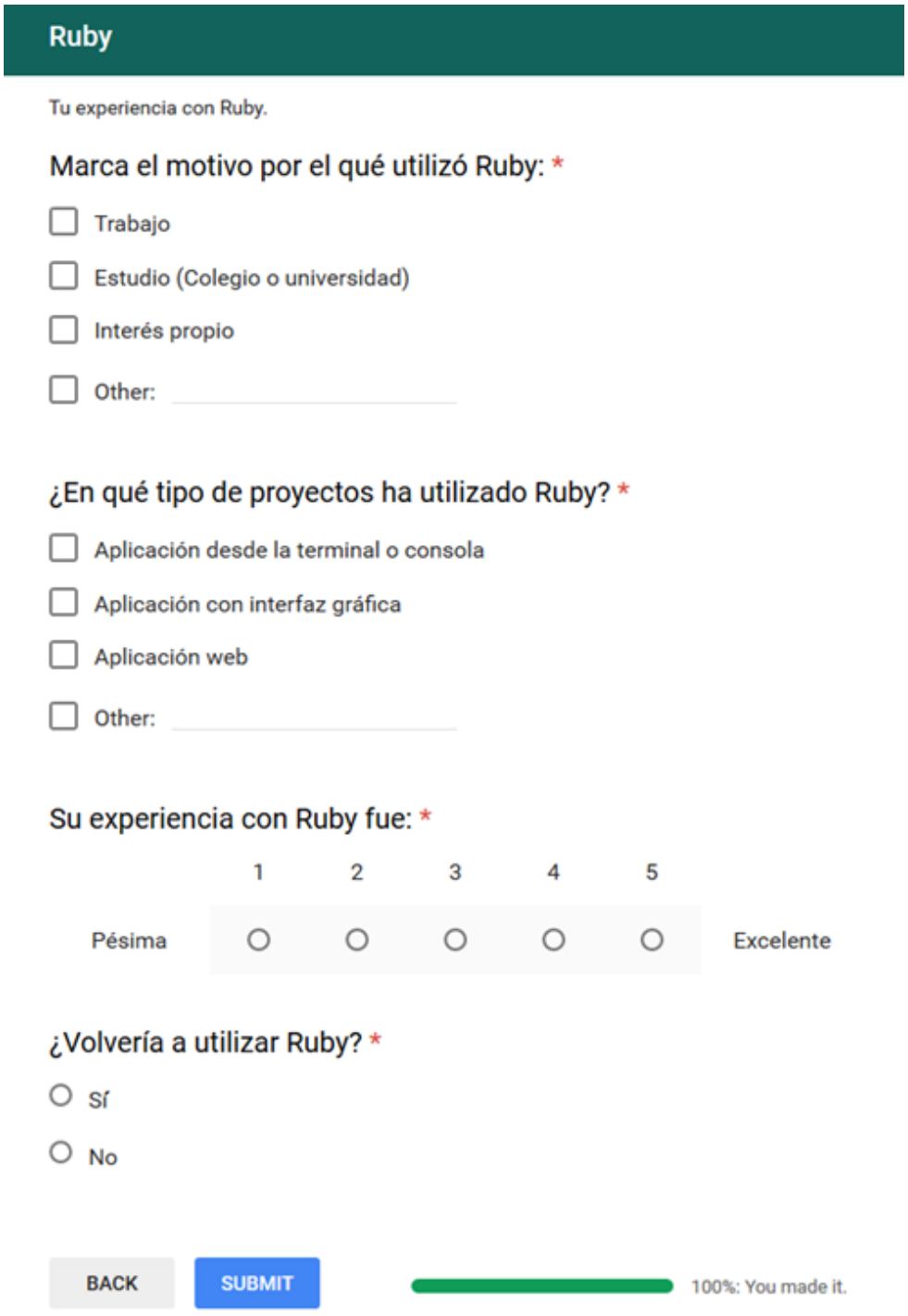
- [ ] C
- [ ] C++
- [ ] C#
- [ ] Java
- [ ] JavaScript
- [ ] Perl
- [ ] PHP
- [ ] Python
- [ ] Ruby
- [ ] Other: _____

Continuación apéndice 1.



Continuación apéndice 1.

¿Conoce el lenguaje de programación Ruby? Lo he utilizado.



Ruby

Tu experiencia con Ruby.

Marca el motivo por el qué utilizó Ruby: *

- [ ] Trabajo
- [ ] Estudio (Colegio o universidad)
- [ ] Interés propio
- [ ] Other: _____

¿En qué tipo de proyectos ha utilizado Ruby? *

- [ ] Aplicación desde la terminal o consola
- [ ] Aplicación con interfaz gráfica
- [ ] Aplicación web
- [ ] Other: _____

Su experiencia con Ruby fue: *

	1	2	3	4	5	
Pésima	○	○	○	○	○	Excelente

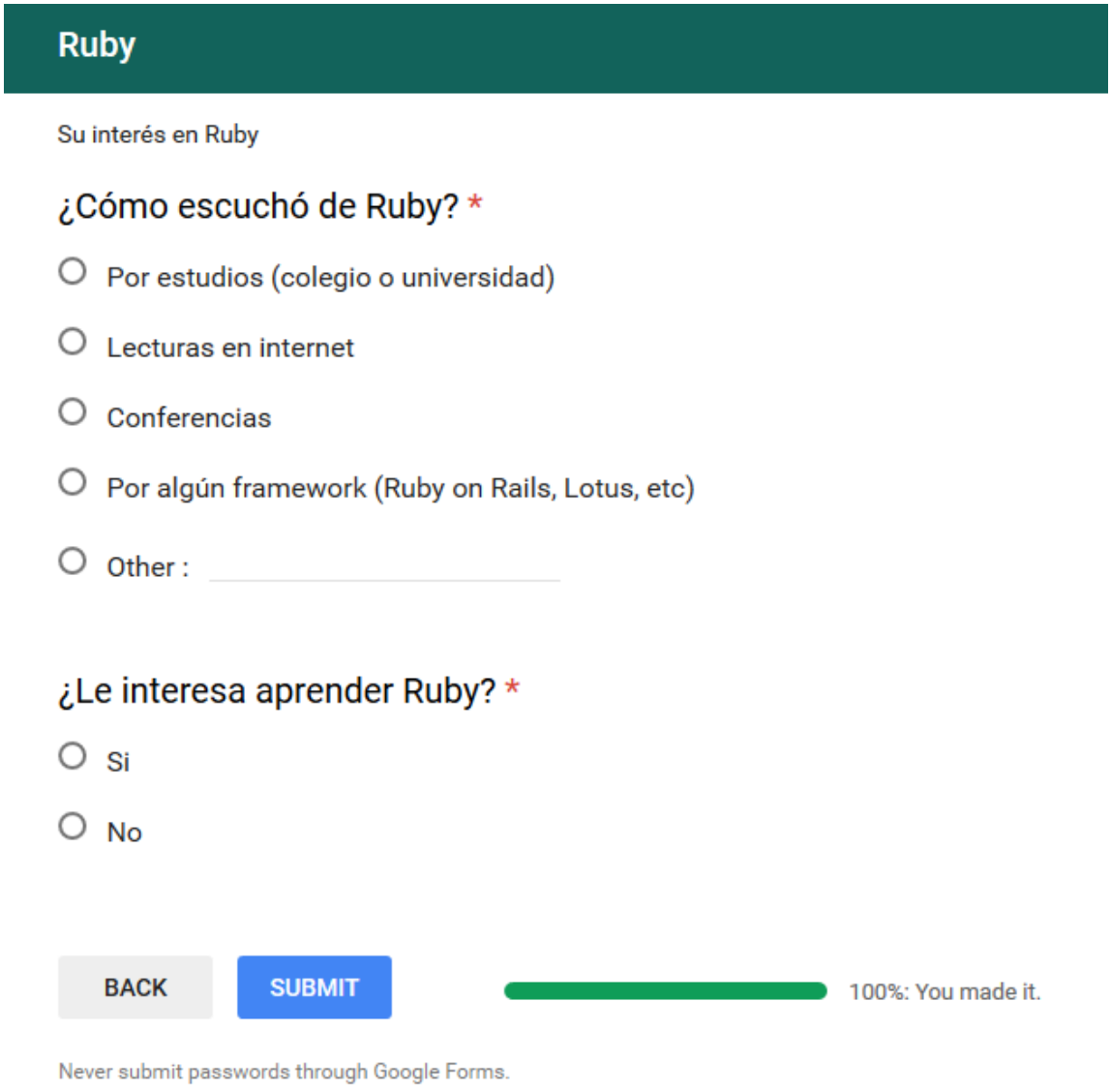
¿Volvería a utilizar Ruby? *

- ○ Sí
- ○ No

BACK SUBMIT 100%: You made it.

Continuación apéndice 1.

¿Conoce el lenguaje de programación Ruby? Lo he oído, pero no lo he utilizado



Fuente: elaboración propia, empleando Google forms.