электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
издатель ФГБОУ ВПО «Московский государственный технический университет им. Н.Э. Баумана»

Возможности использования языка Ruby в учебном процессе

77-48211/597851

08, август 2013

Самарев Р. С.

УДК 378

Россия, МГТУ им. Н.Э. Баумана
samarev@acm.org

Подготовка специалистов, имеющих отношение к вычислительным системам, подразумевает изучение современных технологий и получение практических навыков в использовании одного или нескольких языков программирования. Причем необходимо как изучение базовых алгоритмов, так и изучение перспективных направлений развития программной отрасли, количество которых становится всё больше. Сейчас уже невозможно найти один единственный язык программирования, который удовлетворил бы всем образовательным потребностям. Однако целью данной статьи является, предложить современный язык Ruby, который может стать основной для изучения современных программных технологий в рамках различных учебных курсов, хотя и не претендующий на роль первого изучаемого языка программирования.

Несмотря на то, что язык Ruby получил широкое распространение не так давно, этот язык разрабатывается уже более 20 лет и является достаточно стабильным, с большим открытым сообществом разработчиков, поддерживающих и развивающих как сам язык, так и множество библиотек к нему. Следует отметить, что в настоящее время существует две основные ветви Ruby – версия 1.8 и 1.9, не в полной мере совместимых друг с другом. При этом версия 1.8 в ближайшее время будет лишена поддержки, а недавно вышедшая версия 2.0 не требует переписывания программ, предназначенных для версии 1.9.

Язык Ruby разработан под влиянием таких языков как Perl, Python, Smalltalk. Основной акцент сделан на удобство написания для программиста и удобство последующего её восприятия. Для удобства программиста язык является мультипарадигменным. Программист имеет возможность писать программы как в стиле языков типа Pascal, C/C++ или в стиле скриптовых языков программирования типа Perl, Python. Однако возможности языка не исчерпываются тем, что он позволяет имитировать другие языки для человека, который впервые пытается написать на нём программу. Язык имеет развитую объектную модель, заимствованную от Smalltalk. Специальная конструкция языка (block) позволяет «маскировать» действия, описанные внутри блока. Помимо конструкций, похожих на циклы вида C/C++, имеется огромное количество методов, которые по смыслу сходны с циклами, однако название метода выражает вполне конкретное действие: each, find, find_all, any?, all? и пр. Кроме того, язык позволяет использовать кодировку UNICODE, поэтому программа может быть написана даже на русском языке, включая названия классов, методов, переменных. Таким образом, вытекает другой принцип языка – минимальное количество

кода при максимальной его выразительности. Код идеальном случае выглядит примерно как текст на естественном языке, хотя и сильно ограниченном.

Возможности языка

Язык Ruby является динамическим полностью объектным языком. Любое атомарное значение – это «объект», включая nil, true, false, 0, 1. Соответственно не бывает функций, реализованных вне какого-либо класса (т. е. «методов»), хотя синтаксически описать автономную функцию возможно. Любая операция над объектом приводит к созданию нового объекта, за исключением «методов», имя которых явно об этом сообщает.

Язык поддерживает строгую типизацию, но не на уровне переменных, а на уровне значений. Это значит, что корректными являются записи:

```
a = "строка"
a = 3
```

но некорректна (на этапе выполнения) следующая последовательность:

```
a = 3
a = a + "строка"
```

Интересной особенностью является не обязательное использование скобок при указании аргументов методов. Несмотря на то, что на традиционных примерах это может выглядеть неуместно:

```
foobar
foobar ()
foobar  a, b, c
foobar ( a, b, c )
```

В случае, если программа пишется с тем, чтобы текст выглядел как текст на естественном языке, то это становится оправданным:

`copy source, destination` вместо `copy (source, destination)`

В языке имеются строгие правила именования. Переменная должна начинаться со строчной буквы. Константа – с прописной. Атрибут класса всегда имеет префикс @, а если этот атрибут доступен всем экземплярам класса, то префикс @@. Глобальные переменные начинаются со знака $.

Имеются специальные суффиксы методов – знаки '?' и '!'. Знак '?' означает, что метод является предикатом, то есть утверждением, результат которого истина или ложь.

```
obj.empty? # объект пуст?
Numeric.nonzero? # число не нуль?
```

Знак '!' означает, что метод производит изменение данных, т.е. является деструктивным.

```
obj.empty! # очистить объект!!!
obj.truncate! # Обрезать что-то в самом объекте
obj.remove_name! # Удалить имя из объекта
```

Обратите внимание на то, что метод с именем obj.truncate без знака '!' не должен изменять объект obj, а должен возвращать измененную копию модифицируемых данных!

Язык унаследовал от Perl ряд специальных переменных: $!, $@, $&, $` и пр. Они доступны для использования аналогично Perl.

Доступны все традиционные для языков программирования простые типы данных. Однако интересным является тип Bignum, который представляет собой число без ограничения разрядности. То есть разрядность ограничена лишь размером оперативной памяти вычислительной системы.

Строки не имеют ограничения длины. Класс String имеет множество полезных методов, существенно сокращающих код программы даже, например, в сравнении со специальными библиотеками типа stl для C++.

Массивы и хэши строятся динамически и ограничений на размер не имеют.

Довольно интересны логические операции. Поскольку в Ruby всё есть объект, то традиционная для C/C++ конструкция if (0) в данном случае будет истинной, поскольку 0 – есть объект. То есть истинно всё, что есть объект, а ложно – false (специальный объект) или nil (нет объекта).
Это иллюстрирует следующий пример, в котором задан массив с различными объектами (обратите внимание, что массив не типизирован, а значения имеют разный тип), для каждого из которых проводится проверка на истинность:

```
[nil, 0, 1, true, false, '', '123'].each do |i|
      puts i.inspect + "\t is true" if i
end
```

Результат содержит только те значения, которые восприняты оператором if как истинные:

```
0        is true
1        is true
true     is true
""       is true
"123"    is true
```

Для удобства программиста в языке имеется несколько вариантов операторов ветвления. Это традиционный if (если):

```
if x < 10 then
  statement
end
```

Форма модификатора: `a=10 if b < c.`

А также имеются симметричные операторы unless.

Интересно многообразие циклов (использованы примеры из [4]). Имеются как традиционные конструкции, очень похожие на конструкции в языке Pascal:

```
i = 0
while i < list.size do
  print "#{list[i]} "
  i += 1
end
```

или

```
n = list.size - 1
```

```
for i in 0..n do
  print "#{list[i]} "
end
```

Имеются и циклы, реализованные в виде специальных методов:

```
# повторить количество раз, записанное в переменную n
n = list.size
n.times do |i|
  print "#{list[i]} "
end

# цикл по всем числам от 0 до значения n
n = list.size - 1
0.upto(n) do |i|
  print "#{list[i]} "
end
```

Кроме того имеется множество методов, по сути представляющих собой циклы. Пример представим в виде задачи, в которой необходимо найти гласные буквы, которые встречаются только во всех словах:

```
words = gets.split
result = ['a', 'e', 'i', 'o', 'u', 'y'].find_all do |c|
       words.all?{ |word| word.include? c }
end
```

Здесь #find_all – эквивалент циклу, формирующему массив, #all? – цикл с условием выхода при первом встреченном false, а #include? – цикл до первого совпадения элемента.

Эту же программу можно записать еще компактнее:

```
words = gets.split
result = %w(a e i o u y).find_all {|c| words.all?{|word| word.include? c }}
```

Возможность реализации таких циклов даёт конструкция языка, называемая блоком. Проиллюстрируем это следующим примером:

```
def test_func (x)
      for i in 0...x do
        yield i+1
      end
end
test_func(5) { |n| s="#{n}:";  n.times {s += "*"}; puts s; }
```

Результат:

```
1:*
2:**
3:***
4:****
5:*****
```

Разберём пример подробнее. Вызов метода test_func(5) содержит аргумент 5 и блок {|n| s="#{n}:"; n.times {s += "*"}; puts s;}. В тексте блока содержится декларация |n|, которая

обеспечивает передачу значения переменной из метода test_func. Остальная часть блока представляет собой код, который будет вызван из метода test_func. В самом методе test_func имеется специальная конструкция yield, которая как раз и обеспечивает передачу аргумента в вызывающий код и передачу управления к коду блока.
В результате подстановки получается следующий эквивалентный код:

```
def test_func (x)
      for i in 0...x do
        n=i+1; s="#{n}:";  n.times {s += "*"}; puts s;
      end
end
test_func(5)
```

По своей сути блок представляет собой способ формирования анонимного метода обратного вызова. А метод, который содержит конструкцию yield, может вызывать код, соответствующий yield, когда угодно или не вызывать его вообще, если это соответствует логике программы. И именно поэтому существует возможность реализовать на Ruby методы, которые синтаксически похожи на конструкции типа циклов или ветвлений других языков программирования, но являются управляемыми методами.

Нельзя не сказать об объектных возможностях языка. Ruby позволяет описывать классы в нескольких файлах. Методы могут быть динамически присоединены как к любому классу, так и к экземпляру класса. В первом случае все вновь создаваемые объекты получат новые методы, во втором – новые методы будут иметь лишь конкретные объекты, однако на вновь создаваемые это распространяться не будет. Более того, любые имеющиеся методы могут быть переопределены в своей программе. Это полезно в том случае, если необходимо оперативно изменить поведение ранее написанных классов, однако подход в целом является опасным и получил название «monkey patching». Поскольку если программист слишком увлекается подобным способом программирования, то логику программы становится невозможно понять.

В Ruby используется одиночное наследование, однако имеется понятие «примесь». Примесь – это модуль, реализующий набор методов, которые могут быть подключены к нескольким классам. Например реализуем модуль, считающий сумму:

```
module Sum
  def sum
    reduce(:+)
  end
end
```

И подключим его в стандартный класс массива:

```
class Array
  include Sum
end
```

Теперь любой созданный массив будет иметь возможность использовать этот метод:
`[1,2,3,4,5].sum #=> 15`

А если необходимо сделать метод, доступным не только для массивов, но и для хэшей, множеств и прочих сложных типов данных, достаточно расширить стандартный модуль Enumerable, который уже подключен в этих классах:

```
module Enumerable
  def sum
    reduce(:+)
  end
end
```

Отметим здесь также особенность Ruby, заключающуюся в том, что не обязательно писать служебное слово return для возврата значения из метода. В любом случае будет возвращено последнее вычисленное значение.

Здесь же приведем пример скорее демонстрационный, чем полезный, однако вполне работоспособный:

```
#coding: utf-8
class Integer
    def квадрат
       self*self
    end
end
puts 10.квадрат
puts 20.квадрат
```

Любое число после этого приобретает метод «квадрат», написанный именно на русском языке, который возвращает квадрат этого числа.

Ruby особенно эффективен в написании программ для обработки строк или для обработки массивов. При этом решение традиционных для студентов алгоритмических задач здесь превращается в решение, состоящее из двух-трёх строк. Пример: дан массив A(20). Определить максимум массива и поместить на место последнего отрицательного элемента.

Решение выглядит следующим образом:

```
p ar = Array.new(20) {rand(50)-10}
if idx = ar.rindex{|x| x<0}
  ar[idx] = ar.max
end
p a
```

Метод ‘p’ – это вывод содержимого объекта, включая его структуру. rindex – цикл с конца массива до первого возвращенного условием true.

DSL. Возможности языка для создания новых языков

Следствием гибкости языка Ruby является возможность создания на его основе специализированных языков для конкретной предметной области – DSL (Domain Specific Language). Возможность языка игнорировать скобки при вызове методов, а также

перечислять аргументы (но с некоторыми дополнительными усилиями) позволяет обеспечить возможность написания хотя и ограниченного, но связного текста на естественном языке. С использованием этой возможности были разработаны многочисленные DSL: RSpec, Capybara, Chef, God, Sinatra и др. О некоторых из них упомянем позже.

Рассмотрим более подробно создание DSL на языке Ruby. В статье [7] приводится следующий пример. В качестве учебного средства был разработан язык Quiz'em, который предназначен для написания тестов-опросников. Например следующий вопрос и варианты ответа:

```
Who was the first president of the USA?
1 - Fred Flintstone
2 - Martha Washington
3 - George Washington
4 - George Jetson
Enter your answer:
```

Язык Quiz'em предназначен для описания этого теста. Программа на этом языке выглядит следующим образом:

```
question 'Who was the first president of the USA?'
wrong 'Fred Flintstone'
wrong 'Martha Washington'
right 'George Washington'
wrong 'George Jetson'

question 'Who is buried in Grant\'s tomb?'
right 'U. S. Grant'
wrong 'Cary Grant'
wrong 'Hugh Grant'
wrong 'W. T. Grant'
```

Особенность Ruby здесь проявляется в том, что question, right, wrong являются не просто служебными словами, а полноценными методами, аргументами которых в данном случае являются строки. В статье [7] подробно рассматривается написание этого языка, поэтому здесь ограничимся лишь примером кода для проверки возможности DSL.

```
def question(text)
  puts "Just read a question: #{text}"
end

def right(text)
  puts "Just read a correct answer: #{text}"
end

def wrong(text)
  puts "Just read an incorrect answer: #{text}"
end

load 'questions.qm'
```

Последнее действие обеспечивает загрузку файла questions.qm, содержащего вопросы и ответы в приведенном выше формате. В результате работы этой программы получим распечатанный текст:

```
Just read a question: Who was the first president of the USA?
Just read an incorrect answer: Fred Flintstone
Just read an incorrect answer: Martha Washington
Just read a correct answer: George Washington
Just read an incorrect answer: George Jetson
…
```

Рассмотренный пример является примеров простейшей DSL, однако хорошо иллюстрирует простоту создания подобных языков на Ruby. Применительно к учебному процессу может использоваться как сама возможность написания DSL для разработки DSL студентами, так и реализация DSL под конкретные учебные задачи.

Области применения языка

Написание скриптов для администрирования

Поскольку Ruby в настоящее время разработан для операционных систем семейств Linux, BSD, Windows, Mac OS, этот язык может использоваться для написания служебных программ для контроля и управления операционной системой. Несмотря на широкое распространение скриптов на языках bash, awk для ОС Linux, скрипты на языке Ruby позволяют писать гораздо более понятный современным программистам код, не создавая при этом проблем с их использованием. Удобство Ruby здесь заключается в том, что легко обеспечить как работу с массивами или реализовать разбор текстовых файлов для формирования последовательности Linux-команд, так и проводить обработку результатов выполнения программ, формирующих вывод в консоль. Например запуск консольного процесса и получение результата его выполнения может быть записан в Ruby следующими способами (список не полный):

1. result = `ls -l`
2. f = open("|ls -l")

result = f.read()

3. result = IO.popen(["ls", "-l"]).read

Первый способ широко применяется в языках bash и Perl.

Для дальнейшей обработки строки, на которую ссылается result, доступны все средства Ruby.

Тестирование программ

Для тестирования Ruby-программ существует встроенный механизм Unit-тестов, суть которого заключается в том, что формируется программа тестов. Возможно описание действий, которые необходимо совершить до начала и после тестирования. А каждый тест содержит набор утверждений (методы, имя которых начинается на assert_). Эти утверждения позволяют определить критерий корректного выполнения программы.

Простота написания DSL способствовала распространению фреймворков для тестирования программ, написанных на Ruby. Наиболее известный из них – RSpec (http://rspec.info/). В первую очередь это фреймворк для тестирования Ruby-программ. В отличии от встроенных unit-тестов RSpec имеет развитые средства формирования отчетов о выполнении программ.

Однако главное отличие в том, что программа для тестирования выглядит как текст, написанный на английском языке. Приведем «фирменный» пример Rspec:

```
# Исходная программа bowling.rb
class Bowling
  def hit(pins)
 end

  def score
    0
  end
end
```

Программа для тестирования:

```
# bowling_spec.rb
require 'bowling'
describe Bowling, "#score" do
  it "returns 0 for all gutter game" do
    bowling = Bowling.new
    20.times { bowling.hit(0) }
    bowling.score.should eq(0)
  end
end
```

Обратите внимание на то, что описание теста начинается со слов describe и it "returns 0 for all gutter game" do. С точки зрения языка, describe и it являются методами, которые реализованы во фреймворке RSpec. Однако здесь describe определяет область тестов, а it … определяет содержимое конкретного теста. Текстовая строка после it является ничего не значащим для RSpec описанием сути теста, однако вместе они образуют вполне узнаваемое по структуре предложение. Утверждения в RSpec описываются специальными методами, которые динамически «прицепляются» к методам тестируемого класса. Один из таких методов – should в строке bowling.score.should.

Для тестирования веб-приложений используются как средства типа unit-test, RSpec, если тестируется логика приложения, написанного на языке Ruby, так и специальные библиотеки для тестирования через браузер, например Selenium Webdriver В последнем случае описание программы тестирования осуществляется с помощью RSpec или Capybara (https://github.com/jnicklas/capybara), который является надстройкой над интерфейсом Selenium. Тестирующая программа в последнем случае будет выглядеть следующим образом:

```
describe "the signup process", :type => :feature do
  before :each do
    User.make(:email => 'user@example.com', :password => 'caplin')
  end

  it "signs me in" do
    visit '/sessions/new'
    within("#session") do
      fill_in 'Login', :with => 'user@example.com'
      fill_in 'Password', :with => 'password'
    end
```

```
    click_link 'Sign in'
    page.should have_content 'Success'
  end
end
```

Отметим также, что существует возможность тестирования программ, написанных на иных языках. В случае веб-приложений для этого используется Selenium Webdriver, однако для и для настольных приложений возможно непосредственно подключить управление со стороны интерфейса пользователя через средства управления окнами операционной системы [6]. Кроме того, существует открытый проект http://www.sikuli.org/ для тестирования настольных приложений, причем для его программирования используется язык Java, а, следовательно, через jRuby, о котором будет сказано чуть позже, а также с помощью модуля gem sikuli возможно создания программы тестирования именно на Ruby.

Управление серверами

Для развёртывания и массовой одновременной настройки множества серверов в Linux-системах широко используется средство под названием Puppet (https://puppetlabs.com/). Популярность DSL, реализуемых на Ruby, привела к созданию специального средства Chef (http://www.opscode.com/chef/), которое по сути копирует идеологию Puppet, но пользователю доступен более удобный язык для описания программ развертывания и контроля, а также все средства Ruby в полном объеме, что не доступно пользователям Puppet. Основная идея Chef в том, что формируются так называемые «поваренные книги» или «cookbook» в которых описываются сценарии (в терминологии Chef называются рецептами) установки и настройки приложения (расположение файлов, параметры конфигурации, шаблоны). Chef Server обеспечивает хранение «поваренных книг» и выдает их клиентам.
Существует большая библиотека готовых «поваренных книг» для развертывание широко используемых приложений, например PostgreSQL, Apache, Nginx. Следует заметить, что Chef широко используется именно в тех областях, где необходимо быстро настроить десятки и сотни однотипных серверов.

Другим аспектом управления серверами является запуск и отслеживание состояния выполняемых процессов. Для этого может быть использован DSL под названием God (http://godrb.com/).

Простейшая программа контроля выглядит следующим образом:

```
God.watch do |w|
  w.name = "simple"
  w.start = "ruby /full/path/to/simple.rb"
  w.keepalive(:memory_max => 150.megabytes,
              :cpu_max => 50.percent)
end
```

Данный фрагмент иллюстрирует запуск процесса «ruby /full/path/to/simple.rb» и отслеживание состояния его выполнения. Однако если процесс превысит потребление ОЗУ более 150 Мбайт или превысит 50% потребления процессора, он будет остановлен.

God удобен тем, что по сути предоставляет типовой интерфейс для написания правил контроля. В рамках данного интерфейса могут быть написаны новые правила контроля, однако код, который при этом будет написан, будет сразу же структурно понятен программисту, знакомому с God.

Применительно к учебному процессу средство Chef может использоваться как для настройки учебных классов, так и удобное средство для проведения лабораторных работ по массовому конфигурированию серверов.

Создание веб-приложений

Одной из значительных задач программирования в наше время является создание веб-приложений. Следует заметить, что популярность Ruby в значительной степени обязана появлению фреймворка RubyOnRails (http://rubyonrails.org/), который и подчеркнул ключевые особенности языка Ruby, отсутствующие в других языках программирования. RubyOnRails представляет собой фреймворк, реализующий схему Model-View-Controller. Его особенностью является жёсткая структура проекта при которой отдельные директории отведены для контроллеров, моделей, представлений, вспомогательных модулей, а также предусмотрены тесты на все создаваемые контроллеры и модели. Для создания контроллеров и моделей имеются встроенные генераторы, которые создают необходимые файлы, необходимый код и формируют заготовки тестов с использованием Ruby unit-test или RSpec. Для обращения к СУБД используется специально разработанные средства объектно-реляционного преобразования (ORM). Причем реализацию ORM для RubyOnRails можно считать эталоном гибкости. Программист может вообще никогда не воспользоваться SQL-запросом, поскольку вся работа с данными происходит на уровне Ruby-объектов.
В настоящее время этот фреймворк очень хорошо документирован, поэтому может быть рекомендован для изучения в рамках курсов, связанных с интернет-программированием.

Другим широко распространенный фрейморком является DSL Sinatra (http://www.sinatrarb.com/). Например простейший код веб-сайта выглядит следующим образом:

```
require 'sinatra'

get '/hi' do
  "Hello World!"
end
```

По сути создание веб-приложения сводится к описанию маршрутов и их обработчиков. Например описание обработчиков для различных HTTP-запросов в форме фраз на английском языке:

```
get '/' do
  .. show something ..
end

post '/' do
  .. create something ..
```

```
end

put '/' do
  .. replace something ..
end
```

Дополнительно имеются средства, упрощающие написание шаблонов URL и назначение обработчиков на них. Этот фреймворк широко применяется в тех приложениях, в которых веб-интерфейс разрабатывается малым числом разработчиков или серверная часть веб-приложения вторична (в тех случаях, когда основная логика выполняется на браузере). Хорошо подходит в случаях, когда необходимо обеспечить создание учебных веб-приложений, а времени на изучение RubyOnRails недостаточно.

Создание настольных приложений

Несмотря на полное отсутствие средств для построения графических интерфейсов пользователя в самом языке Ruby, существует большое количество библиотек, позволяющих использовать возможности wxWidgets, GTK, Qt, ncurses и др. Однако если в случае wxWidget и GTK программирование сводится к манипулированию графическими примитивами также, как с этими библиотеками работают на языке C, то в случае Qt возможно использование Qt-объектов, однако возможности, доступные C++ по переопределению классов Qt доступными не будут.

Ruby может быть использован для написания не очень сложных графических приложений, либо приложений, основной графический интерфейс которых реализован на других языках программирования. Однако в учебных целях Ruby может быть использован как для ознакомления с основными библиотеками, так и для реализации учебных задач.

Java и Ruby

Существуют реализации Ruby на языке Java. Наиболее известная из них – jRuby (http://jruby.org/). Основной особенностью этой реализации по сравнению с Ruby-native является то, что программе на языке Ruby становятся доступны все имеющиеся Java-классы. А классы на языке Ruby могут быть использованы в Java-программе. Очевидно, что полной интеграции языков достичь не получится, однако веб-приложения на языке Ruby могут быть запущены под управлением Java-сервера приложений. Настольные приложения на Ruby могут использовать графические библиотеки Java, а для платформы Android возможно создание графических приложений при помощи средства Ruboto. Таким образом, в учебных целях знание Ruby и здесь может существенно облегчить восприятие материала студентами.

Прототипирование и отработка протоколов взаимодействия с периферийными устройствами

Для взаимодействия с периферийными устройствами обычно используются как низкоскоростные порты типа RS232, так и высокоскоростные типа USB. Для всех распространенных типов коммуникаций в ОС Linux существуют библиотеки, позволяющие реализовывать взаимодействие на пользовательском уровне операционной системы. Например взаимодействие с RS232-портами производится через специальные файлы, взаимодействие с USB – через библиотеку libusb.

Поскольку Ruby позволяет подключать динамические библиотеки, написанные на других языках, существует огромное количество модулей-переходников к там библиотекам. К примеру, существует модуль Ruby, который так и называется libusb. Пример кода подключения к устройству выглядит следующим образом:

```
require "libusb"

usb = LIBUSB::Context.new
device = usb.devices(:idVendor => 0x04b4, :idProduct => 0x8613).first
device.open_interface(0) do |handle|
    handle.control_transfer( :bmRequestType => 0x40, :bRequest => 0xa0,
                  :wValue => 0xe600, :wIndex => 0x0000, :dataOut => 1.chr)
end
```

Это позволяет реализовать удобный интерфейс для описания протокола, выявления проблем и отладки процесса взаимодействия. Гибкость Ruby позволяет подготовить код для переписывания на целевых языках программирования, однако во время отладки протокола можно использовать все синтаксические средства Ruby для лаконичного описания логики взаимодействия.

Заключение

Автор статьи читает курс «Языки Интернет-программирования». Одним из ключевых языков курса является Ruby. Несмотря на то, что этому языку не уделяется столько же времени, сколько отводится для изучения традиционных языков программирования типа Pascal или C++, студенты вполне смогли освоить программирование на нем до уровня, достаточного для написания несложных интерактивных консольных программ и веб-приложений с использованием Ruby on Rails. Уровень усвоения языка различался от минимального допустимого до понимания именно тех возможностей Ruby, которые дают возможность создавать лаконичный код.

В настоящее время ближайшим конкурентом Ruby является язык Python, поэтому нельзя не сказать несколько слов относительно того, почему именно Ruby лучше подходит для учебного процесса. Существует и активно используется несколько несовместимых версий Python, а наибольшее количество библиотек создано для его устаревших версий. В то же время язык Ruby развивается эволюционно, поэтому существенных проблем с поддержкой библиотек не возникало. Python использует отступ от начала строки в качестве операторного блока. Это с одной стороны способствует принудительному оформлению программы, с другой создаёт ненужные проблемы с кодом программы для начинающих. В целом, возможности Ruby и Python практически равные и обладают схожими синтаксическими конструкциями (для последних версий Python), что при необходимости позволяет легко освоить Python, уже зная Ruby. Тем не менее синтаксические возможности Ruby изначально шире и позволяют писать код, ориентированный на восприятие людьми, владеющими естественным английским языком, подбирая именно такие слова для реализации программных конструкций, которые наиболее подходят для описания совершаемых действий. Именно это и позволяет создавать лаконичные, хорошо читаемые программы, но теряющие выразительность на Python.

В целом, возможности языка Ruby и созданной с его помощью инфраструктуры таковы, что именно Ruby можно рекомендовать к обязательному изучению студентами технических

специальностей, поскольку может использоваться как в задачах, связанных с программированием сложной логики, так и в задачах администрирования, протипирования, быстрого создание веб или настольных приложений. В настоящее время на русском языке написано и переведено достаточное количество литературы [2-5], существуют методические пособия [1], а также созданы электронные учебники и справочники. Приведем также некоторые полезные ссылки для желающих ознакомиться с этим языком:

- http://ru.wikibooks.org/wiki/Ruby - учебник на русском языке.
- http://www.ruby-lang.org - основной сайт Ruby.
- http://www.ruby-doc.org - официальная документация Ruby на английском языке.
- http://rubymonk.com — сборник интерактивных учебников с возможностью написать мини-программу и проверить её работу. На английском языке.
- http://www.codecademy.com/ru/tracks/ruby — еще один сборник интерактивных учебников. На английском языке.
- http://rubygems.org - дополнительные библиотеки для ruby.

Список литературы

1. Роганов Е.А., Роганова Н.А. Программирование на языке Ruby. Учебное пособие.– Москва: МГИУ, 2008.–56 стр.
2. Сэм Руби, Дэйв Томас, Дэвид Хэнссон. Гибкая разработка веб-приложений в среде Rails. 4-е издание . Серия: Для профессионалов.- Питер: 2013.- 464 стр.
3. Д. Флэнаган, Ю. Мацумото. Язык программирования Ruby.– СПб.; Питер, 2011.– 496 стр.
4. Оби Фернандес. Путь Rails. Подробное руководство по созданию приложений в среде Ruby on Rails. -М. Символ-Плюс, 2009 г.– 768 стр.
5. Фултон Х. Программирование на языке Ruby.–М.:ДМК Пресс, 2007.-688 с.:ил.
6. Ian Dees. Scripted GUI Testing with Ruby. Pragmatic Programmers.– Pragmatic Bookshelf, 2008.–192 p.
7. Russ Olsen. Building a DSL in Ruby - Part I. [Электронный ресурс]. Режим доступа: http://jroller.com/rolsen/entry/building_a_dsl_in_ruby (дата обращения 01.06.2013)