

# Ruby, язык программирования.

Ruby – объектно-ориентируемый динамический интерпретируемый язык программирования высокого уровня.

Ruby – мощный, практичный и элегантный язык.

Ruby создан в 1995 году Юкихиро Матцумото (Япония) под влиянием Smalltalk, Eiffel и Perl.

# 1. Вкратце о языке.

# ОСНОВНОЙ СИНТАКСИС.

```
class SpaceMan < Dreamer
  attr_reader :name
  attr_accessor :rocket
  def initialize(name)
    @name = name
  end
  def where_i_am?
    @current_place.to_s
  end
end
```

# Основной синтаксис.

```
class SpaceMan < Dreamer
  attr_reader :name
  attr_accessor :rocket
def initialize(name)
  @name = name
end
def where_i_am?
  @current_place.to_s
end
end
```

## Классы. Объявление и наследование.

**class** – ключевое слово для объявления класса.

Символ < используется для наследования.

Объявление в классе заканчивается ключевым словом **end**, как любой другой блок кода.

Множественное наследование запрещено.

Доступ к родительскому классу осуществляется с помощью ключевого слова **super**.

# Основной синтаксис.

```
class SpaceMan < Dreamer
  attr_reader :name
  attr_accessor :rocket
  def initialize(name)
    @name = name
  end
  def where_i_am?
    @current_place.to_s
  end
end
```

**Конструктор, методы и члены класса.**

Объявление метода в классе начинается с ключевого слова **def**, затем следует имя метода, и параметры.

Метод-конструктор класса должен называться **initialize**.

Любая переменная, имя которой начинается с одного символа **@** - член класса. С двух символов **@** - статический член класса.

# Основной синтаксис.

```
class SpaceMan < Dreamer
  attr_reader :name
  attr_accessor :rocket
  def initialize(name)
    @name = name
  end
  def where_i_am?
    @current_place.to_s
  end
end
```

## Вызов метода.

Метод вызывается, как и в большинстве языков, через точку, скобки с перечислением параметров после вызова можно опустить, если это не вызывает недоразумений.

# Основной синтаксис.

```
class SpaceMan < Dreamer
  attr_reader :name
  attr_accessor :rocket
def initialize(name)
    @name = name
end
def where_i_am?
    @current_place.to_s
end
end
```

## Ruby-символы.

Идентификатор, и символ двоеточия в начале – это специальный объект в ruby, ruby-символ.

В большинстве случаев можно считать, что это ссылка на строку. Точнее, что-то, что представляет строку или имя.

Два ruby-символа с одинаковым именем – это один и тот же объект.

# Основной синтаксис.

```
class SpaceMan < Dreamer
  attr_reader :name
  attr_accessor :rocket
def initialize(name)
    @name = name
end
def where_i_am?
    @current_place.to_s
end
end
```

Весь код в объявлении класса начинается как только интерпретатор видит его.

**attr\_reader** и **attr\_accessor** – вызов метода класса Module, добавляющего в класс методы для чтения и доступа к членам класса **@name** и **@rocket** соответственно.

# ОСНОВНОЙ СИНТАКСИС.

```
class Array
  def from_place(place)
    self.select do |s|
      s.where_i_am == place
    end
  end
end

flyers << mike
flyers.from_place('Mars').each do |s|
  print s
end
```

# Основной синтаксис.

```
class Array
  def from_place(place)
    self.select do |s|
      s.where_i_am == place
    end
  end
end
```

```
flyers << mike
flyers.from_place('Mars').each do |s|
  print s
end
```

Классы в ruby открыты и свободны для дополнения.

Здесь в стандартный класс **Array** добавляется собственный метод, который будет виден только на время действия этого кода.

# Основной синтаксис.

```
class Array
  def from_place(place)
    self.select do |s|
      s.where_i_am == place
    end
  end
end
```

```
flyers << mike
flyers.from_place('Mars').each do |s|
  print s
end
```

## Ruby-блоки.

Ruby-блоки – специальная конструкция языка. Код, объявленный внутри **do..end** выполняется внутри метода, с которым используется блок.

**select** и **each** – стандартные методы для выбора по условию и перебора всей коллекции соответственно.

# Основной синтаксис.

```
class Rocket
  def travel_to(place)
    planet = fly_to place
    yield planet if block_given?
    fly_back
  end
end
end
```

```
mike = SpaceMan.new('mike')
mike.rocket = Rocket.new
mike.rocket.travel_to(:mars).do |planet|
  mike.conquer! planet
end
```

Код, написанный внутри блока, выполняется внутри метода, принимающего его. Этот метод рассматривает блок, как функцию, в которую можно передать какие-то параметры.

Вызов блока и передача параметров происходит с помощью ключевого слова **yield**.

## 2. Ruby и Python.

Ruby и Python очень похожи.

И Ruby и Python – это объектно-ориентированные высокоуровневые динамические языки программирования. Оба позволяют использовать метапрограммирование и, частично, функциональный стиль программирования.

Пожалуй, можно найти лишь несколько существенных отличий.

# Синтаксис.

Основным синтаксическим отличием Ruby до Python 2.5 были блоки, как способ использования сопрограмм.

Нововведения в Python 2.5 (измененный `yield`, `with`) позволяют писать код, который будет предоставлять те же возможности, что и ruby-блоки.

# Использование и производительность

Python позиционируется как язык общего назначения, в то время как для ruby пока что больше подходит роль языка для обработки строк и обёртки над C-библиотеками.

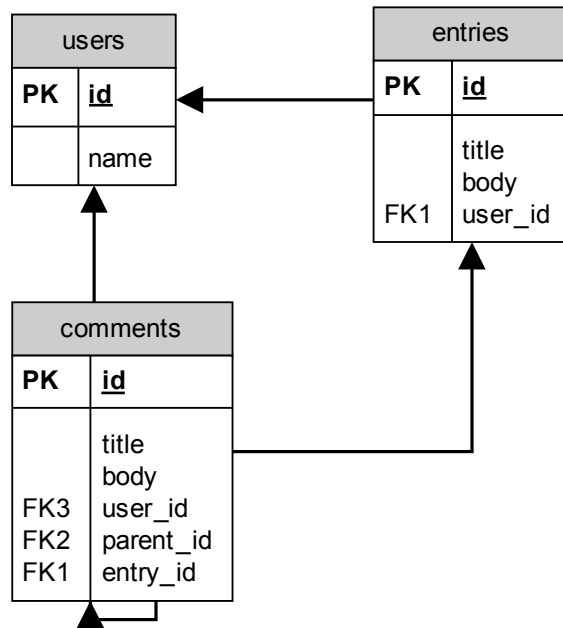
В первую очередь на это влияет скорость интерпретатора. Программы на Python работают значительно быстрее, чем программы на ruby.

### 3. Некоторые скучные повседневные задачи с ruby.

Базы данных, ORM.  
ActiveRecord.

Задача: реализовать классы – обертки таблиц в базе данных, модели пользователей системы блогов.

Описываемые таблицы:



# ActiveRecord.

```
require 'rubygems'  
require 'activerecord'
```

```
class User < ActiveRecord::Base  
  has_many :entries  
end  
class Entry < ActiveRecord::Base  
  has_many :comments  
  belongs_to :user  
end  
class Comment < ActiveRecord::Base  
  belongs_to :entry  
  belongs_to :user  
  acts_as_tree  
end
```

ActiveRecord – библиотека, позволяющая связать таблицу в базе данных и класс в ruby.

Объект класса – наследника

**ActiveRecord::Base** – имеет эксплицитно все методы для доступа к полям в таблице.

**has\_many** и **belongs\_to** – отношения между таблицами в реляционной базе данных.

# ActiveRecord.

```
user = User.find_by_name('Kei Korte')
user.entries.each do |entry|
  entry.comments each do |comment|
    if comment.user.name == 'Katy'
      comment.destroy
    end
  end
end
End
```

```
user.name = 'Ryan'
user.save
```

Прямой доступ к таблице в базе данных объект имеет лишь в методах **find**, **save**, **destroy**.

**find\_by\_name** – один из динамических методов для поиска в базе данных.

Генерация и парсинг XML.  
Builder и REXML.

# Builder.

Генерация xml – несложная задача, не зависимо от языка. В простых случаях легко можно обойтись и без внешних библиотек.

```
require 'rubygems'
require 'builder'
xml = Builder::XmlMarkup.new
xml.instruct!
xml.customers do
  customers.each do |customer|
    xml.customer(id = customer) do
      xml.name {customer.name}
      xml.email(customer.email)
    end
  end
end
end
```

Библиотека Builder позволяет просто генерировать xml-файлы, при помощи прокси-объекта, который создает очередной элемент xml исходя из названия вызываемого псевдо-метода.

# customers.xml

```
<customers>
  <customer id="100">
    <name>Christian Newman</name>
    <email>newman.mr@gmail.com</email>
  </customer>
  <customer id="114">
    <name>Charlie Newman</name>
    <email>newman.jr@site.ru</email>
  </customer>
</customers>
```

# REXML

```
require 'rexml/document'
```

```
include REXML
```

```
xml_tree = Document.new(File.new('customers.xml'))
```

```
puts "N:" + xml_tree.root.elements.size.to_s
```

```
XPath.match(xml_tree, '//customer').each do |c|
```

```
  puts c.name+" #" + c.attributes['id']
```

```
  c.elements.each do |element|
```

```
    puts "\t" + element.name+": "+element.text
```

```
  end
```

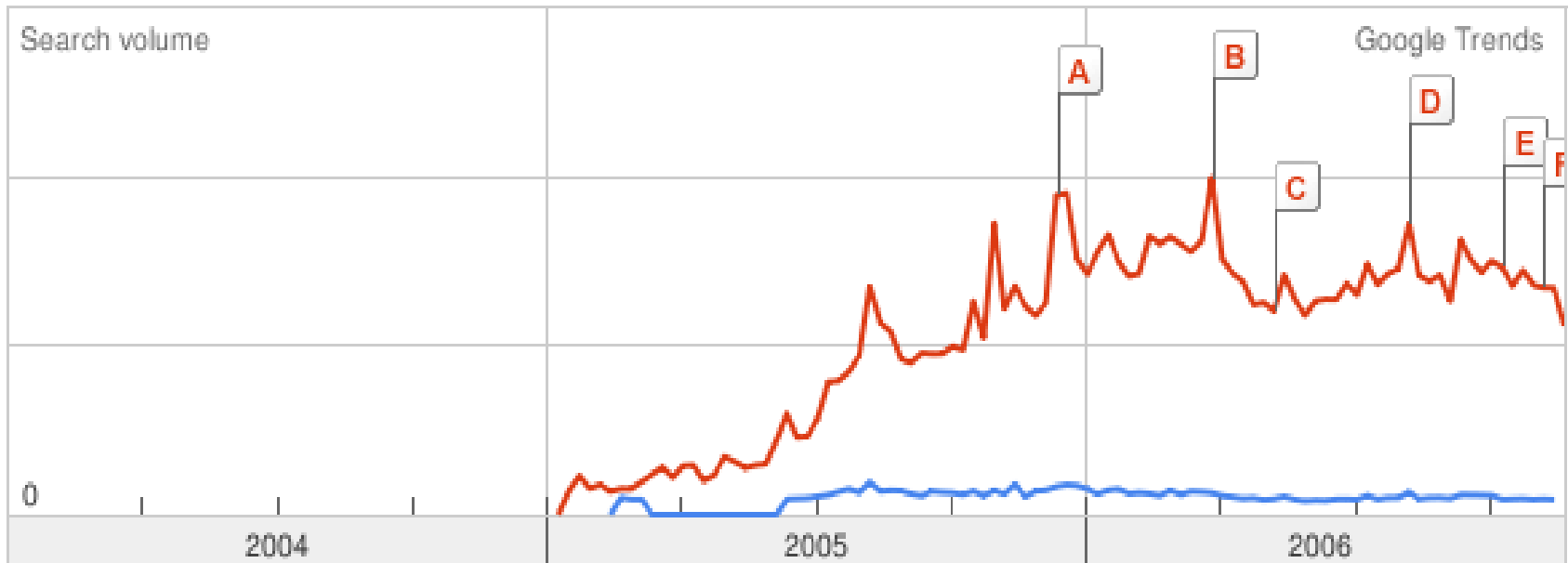
```
end
```

Разбор xml-документов - всегда, наверное, не самая интересная задача. Здесь используется модуль REXML из стандартной библиотеки. REXML – это не единственный (и не самый производительный) метод для разбора XML в Ruby.

## 4. Недостатки.

Несмотря на то, что язык существует с середины 90х, общую известность он получил 2 года назад. Значимые недостатки в языке практически отсутствуют, но есть недостатки, возникшие из-за молодости языка.

● ruby language ● ruby on rails



- Хорошие глянцевые IDE,
  - всегда переведенная на английский качественная документация,
  - быстрый интерпретатор –
- все это нет.

Что-нибудь ещё?

[ruby-lang.org](http://ruby-lang.org)