

Человек создан для творчества, и я всегда знал, что люблю творить. Увы, я обделён талантом художника или музыканта. Зато умею писать программы.

Я хочу, чтобы компьютер был моим слугой, а не господином, поэтому я должен уметь быстро и эффективно объяснить ему, что делать.

Юкихиро Мацумото

Занятие 1.

Тема 1. Введение в язык Ruby.

История создания.

Первая версия классической реализации появилась в далеком 1995 году.. Потом долго добирался до Европы и США, так как не было переводов документации. В 2006 получил-таки всеобщее признание. А с приходом веб-фреймворка Rails стал ещё более популярным.

Последняя версия на данный момент Ruby 2.2.1

Официальный сайт: <https://www.ruby-lang.org/>

Онлайн-документация: <http://www.ruby-doc.org/>

Какой же он, этот Ruby?

- **динамический**: позволяет определять типы данных в момент присваивания значения и осуществлять синтаксический анализ и компиляцию «на лету», на этапе выполнения программы.

Динамические языки удобны для быстрой разработки приложений.

- **сильно типитизированный**: несмотря на динамическую типитизацию у него нет автоматического преобразования типов. Просто сложить число и строку не получится (в отличие, например, от JavaScript).

- **интерпретируемый**: анализирует и тут же построчно выполняет (собственно интерпретация) программу по мере поступления её исходного кода на вход интерпретатора.

Достоинством такого подхода является мгновенная реакция.

Недостаток — такой интерпретатор обнаруживает ошибки в тексте программы только при попытке выполнения команды (или строки) с ошибкой.

- **объектно-ориентированный**: Ruby — полностью объектно-ориентированный язык. В нём все данные являются объектами, в отличие от многих других языков, где существуют примитивные типы (это означает, что и числа, и строки, и даже “ничто” в виде nil - всё это объекты определённого класса). Каждая функция — метод.

- **рефлексивный**: может отслеживать и модифицировать собственную структуру и поведение во время исполнения (определять и переопределять методы и классы во время исполнения). Это один из видов метапрограммирования.

На практике означает крайнюю гибкость и изменяемость прямо во время выполнения программы.

- **высокоуровневый**: абстрагирован от реализации самих структур данных и операций над ними.

Более понятен человеку и прост в использовании, но в то же время это сказывается на возможностях языка: прошивки на кофеварки и операционные системы на нём написать не получится.

- **обладает своим сборщиком мусора**: самостоятельно автоматически управляет используемой программой памятью.

Специальный процесс, называемый сборщиком мусора, периодически освобождает память, удаляя объекты, которые уже не будут востребованы программой.

Сравнение с другими языками:

Схожесть с Python:

- наличие интерактивной консоли - *irb*
- наличие системы документации в командной строке - *ri*
- объекты String могут хранить многострочные значения (аналог тройных кавычек в Питоне)
- размер массивов увеличивается автоматически
- объекты динамически строго типитизированы
- всё есть объект и переменные лишь ссылки на объекты
- возможности для документации прямо в тексте программы - *rdoc*
- ...

Отличие от Python:

- объекты строк - изменяемые
- наличие констант (переменных, значение которых не планируется изменять)
- некоторые принятые условия по наименованию (например классы с большой буквы, локальные переменные - с маленькой)
- только одна реализация списка - массив, и он изменяемый
- двойные кавычки обрабатывают управляющие символы (`\n`, `\t`..)
- весь доступ к атрибутам объектов - через вызов метода, напрямую нельзя
- скобки в методах, как правило, не обязательны
- отсутствие множественного наследования. Данная функциональность реализована через примеси модулей
- возможности изменения методов даже стандартной библиотеки
- только `false` и `nil` - ложь. `0`, `0.0` - истина
- ...

Интерпретаторы Ruby. Советы по установке.

Существующие реализации:

- **Классический MRI** - написан на **C**

- JRuby - использует виртуальную машину Java
- IronRuby - под .NET
- MacRuby — реализация для Mac OS
- Rubinius - сам на себе (Ruby)
- mruby - облегченная версия для встраивания в другие программы

Экосистема: Ruby = interpreter + rubygems

Rubygems - хранилище библиотек.

<https://rubygems.org/>

Установка.

Так как в реальном мире 99% приложений на Ruby/Rails hostятся на операционных системах семейства linux Вам придётся подружиться с одной из них.

- проверяем наличие в системе так как часто уже предустановлен

\$ ruby -v

Возможные варианты:

- как системный
 - + стабильный и проверенный
 - + легко установить
 - часто сильно устаревшие версии
 - менее гибок (нет возможности сборки со специальными параметрами для своей операционной системы)
 - условно только один на всей системе

\$ yum install ruby ruby-devel ruby-irb

- из исходных кодов (<https://github.com/ruby/ruby>)
 - + гибок, можно собрать специально под архитектуру своего компьютера

- сложен в установке
- сложно обновлять версии
- условно только один на всей системе

- **с помощью Ruby Version Manager - RVM**

- + возможность использования разных версий на одной машине
- + возможность использования разделенных наборов библиотек для каждой из версий или даже проекта
- + легко обновлять, удалять
- при установке возможны конфликты с системными библиотеками (решаемо)

официальный сайт <http://rvm.io/>

Установка:

```
$ yum install curl
```

```
$ gpg --keyserver hkp://keys.gnupg.net --recv-keys D39DC0E3
```

```
$ \curl -sSL https://get.rvm.io | bash -s stable
```

```
$ source ~/.rvm/scripts/rvm (или переоткрыть терминал)
```

```
$ rvm requirements (доустановит необходимые зависимости под вашу систему)
```

Проверяем установку:

```
$ rvm -v
```

Устанавливаем Ruby:

```
$ rvm install 2.2.1 (или 2.2.0)
```

```
$ rvm use 2.2.0 --default
```

```
$ ruby -v
```

- **Полезные команды:**

```
$ rvm list known
```

```
$ rvm list / rvm use 2.2.0
```

\$ rvm gemset list / rvm gemset use trololo --default

Работа с помощью irb. Документация ri.

По-умолчанию Ruby устанавливается вместе с двумя полезными утилитами.

Интерактивная консоль-песочница: **irb**

Вход:

\$ irb

Далее можно проверять любой Руби-код

\$ puts "Hello World!"

...

Выход:

\$ exit

или **Ctrl+D**

Поиск по документации в консоли: **ri**

Если ставили Руби через RVM:

Генерируем локальную документацию:

\$ rvm docs generate

Вход:

\$ ri

Далее можно писать название метода, класса и в ответ получать их описание на английском.

\$ Hash

\$ String.split

...

Выход:

Ввести пустую строку или **Ctrl+D**

Где используется Ruby?

Отдельно Ruby чаще всего используется как скриптовый язык для написания программ для конфигурации чего-либо, запуска тестов, разворачивания виртуальных машин в облаке, для расширения возможностей программ и тд..
Вместе с Rails - уже в качестве полноценных веб-приложений, сайтов и сервисов. Некоторые компании использующие Ruby или Rails.

- NASA, NOAA (национальная администрация по океану и атмосфере)
- Motorola
- Amarok - музыкальный плеер
- SketchUp
- Inkscape — скрипты для обработки векторных изображений
- Metasploit
- Chef, Puppet — системы управления конфигурациями
- Redmine — багтрекер
- XChat
- Vagrant
- Netmri - ситема автоматизации сетей и управления DNS
- Heroku, Nitrous, Digital Ocean - облачный хостинг
- Eram, Amazon - внутренние проекты
- Github - крупнейший портал для разработчиков и хранения исходных кодов
- Coub - русскоязычный видео-сервис
- 37signals - система управления проектами
- Shopify - облачная система создания интернет магазинов
- Travis - система тестирования и сборки проектов
- ...
- + хостинг провайдеры, мелкий бизнес, простые сайты, стартапы..

Также на Ruby можно разрабатывать и мобильные приложения, с помощью оболочек:

- Titanium Studio — среда разработки мобильных приложений на HTML5, CSS3,

Javascript, Ruby, Rails, Python, PHP

- Ruboto — среда разработки Android приложений на Ruby
- RubyMotion — среда разработки iOS приложений на Ruby

Список литературы для самостоятельной работы:

<https://ru.wikipedia.org/wiki/Ruby> - статья на вики на русском

<https://www.ruby-lang.org/en/documentation/quickstart/> - краткий экскурс в язык, на английском

https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F - сравнение с другими языками, по-русски

<https://pragprog.com/book/ruby4/programming-ruby-1-9-2-0> - библия Рубистов, основная книга (английская версия)

<http://dl.dropbox.com/u/306877/mtn/Ruby.pdf> - книга о Ruby, на русском

http://dl.dropbox.com/u/306877/mtn/ruby_book.pdf - книга по Rails, есть вводные главы по Ruby, на русском.

https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5 - немного об ООП, все должны разбираться

Тема 2. Синтаксис языка.

Запуск программ:

- Через интерактивную консоль irb. Только для проверки чего-либо, данные не сохраняются после выхода.
\$ irb
- Через прямое выполнение интерпретатором:
\$ ruby -e 'variable = a; puts a'
- Код в файле, передача файла интерпретатору. Файл находится в той же папке, где мы и вызываем:
\$ ruby ./file.rb
- Исполняемый Ruby-скрипт
Код в файле, первой строкой идёт волшебный комментарий. Добавляем права на запуск и вызываем напрямую.
Волшебный комментарий: `#!/usr/bin/env ruby`
\$ chmod +x ./file.rb
\$./file.rb

Руководство по синтаксису.

Комментарии и вывод:

В Ruby знаком начала комментария служит `#`. Всё, что между ним и концом строки пропускается. Пример:

```
puts 2 + 2 # это комментарий
```

```
puts "Привет!" # тоже комментарий
```

Многострочные комментарии помещаются между словами `=begin` и `=end`

```
=begin
```

```
Это длинный комментарий
```

```
Очень длинный
```

```
=end
```

```
puts "Привет!"
```

Названия переменных:

- начинаются с буквы или `_`
- состоят из латинских букв, цифр или знака подчёркивания
- кроме зарезервированных слов

```
ruby # ok
ruby_tort # ok
ruby_tort666 # ok
_6ruby # ok
___ # и даже это ок
666ruby # а это уже нет
```

Переменные хранят ссылку на объект, а не сам объект.
Разберем в классе.

Вывод на кран:

`printf` – форматированный вывод
`p` – переводит на новую строку, вызывает `.inspect` (выводит в кавычках реальное содержимое переданного объекта, не обрабатывая управляющие символы)
`puts` – переводит на новую строку, вызывает `.to_s` (приводит к строке если может, обрабатывает управляющие символы, вывод в удобочитаемом виде)
`print` - то же, что и `puts`, но без новой строки.

Простейшая программа вывода приветствия на экран:

```
puts 'Hello, World!'
```

; в конце строки - необязательна

отступы во вложенных конструкциях: два пробела (не `tab`)

`=` оператор присваивания

`!` оператор отрицания. Есть ещё `not`, у него слабее приоритет, лучше не использовать.

`==` оператор проверки на равенство

`===` расширенная проверка на равенство или включение, с приведением типов
(переопределен только на некоторых классах)

`>`, `<=`, `!=` операторы сравнения

`+`, `-`, `/`, `*` - простейшая математика

`**` возведение в степень

`%` остаток от деления

`&&` дополнительное условие (и). Есть ещё `and`, у него слабее приоритет, лучше не использовать.

|| дополнительное условие (или) Есть ещё `or`, у него слабее приоритет, лучше не использовать.

- **&&** – выполняют второе условия, только, если первое истина

- *and слабее &&*

- `||or` – выполняют второе условия, только, если первое ложно

- or слабее ||

!! приведение к Boolean

+= короткая версия инкрементации на заданную величину:

$$a = 1$$

a += 1 #=> 2

a += 2 #=> 4

$a += 1$ равносильно $a = a + 1$

множественное присваивание $a, b = b, a$

Инструкции для самостоятельной работы:

p "1 + 1"

```
puts 1 + 1
```

p "10 / 3"

```
puts 10 / 3
```

p "10.0 / 3"

```
puts 10.0 / 3
```

p "10.class"

```
puts 10.class
```

p "(10*99999999999999999999999999999999).class"

puts (10*9999999999999999999999999999999999).class

```
p "float to integer: 123.0.to_i"
```

```
puts 123.0.to_i.class
```

p "integer to complex: 123.to_c"

```
puts 123.to_c
```

```
p "2 ** 3"  
puts 2 ** 3
```

```
p "2 ** -3"  
puts (2 ** -3).class
```

```
p "2 ** 0.5"  
puts 2 ** 0.5
```

Управляющие структуры. Условия.

- позволяют регулировать процесс выполнения программы, задавать условия и разветвления исполнения кода.
- конструкция if-else-end выводит результат последнего выражения
- [then] - для однострочников
- Ruby: всё что может вывести – выводит. Тобешь фактически вызов = выражение.
- всё что не nil и не false – есть истина.
- 0, "", [], {} - всё истина :)

```
If < conditions > [ then ]  
  < code >  
elsif < conditions > [ then ]  
  < code >  
elsif < conditions > [ then ]  
  < code >  
else  
  < code >  
end
```

```
unless < conditions > [ then ]  
  < code >  
else
```

```
< code >  
end
```

Например:

```
number = 10  
if number > 10 then puts '>5'  
elsif (number > 8 && number < 20)  
  puts 'bigger than 8'  
  puts 'lest than 20'  
else  
  puts 'less than 8'  
end
```

Пример использования результата:

```
result = if 5 > 10  
  'first'  
elsif 5 > 2  
  'second'  
else  
  'third'  
end  
puts result # => second
```

Краткая форма:

```
< code > if < conditions >
```

```
< code > unless < conditions >
```

```
< code-условие > ? что делать если true : что делать если false
```

Примеры:

```
before = 1  
puts before # => 1  
before = 5 if true  
puts before # => 5
```

```
word = 'word'
```

```
word = 'new word' unless 5 > 10 && 5 > 20
```

puts word # => new word

puts 10 > 0 ? 'big' : 'small' # => big

Домашнее задание #1:

Теория:

- прочесть заметки лекции ещё раз, два, три...
- пройти по всем указанным ссылкам
- составить список вопросов
- выделить одну интересную и запомнившуюся особенность/метод/факт

связанный с Ruby

Практика:

- установить системный Ruby через yum
 - установить новые библиотеки-гемы 'pry', 'sqlite3' (возможно понадобится установка дополнительных dev-пакетов)

- удалить гемы

- удалить системный Ruby

- установить Ruby 2.2.1 / 2.2.0 через RVM
- создать новый gemset mtn, сделать его дефолтным при заходе в баш
- установить в нем библиотеки-гемы 'pry', 'sqlite3'

*** поиграть в игру-курс <http://tryruby.org/levels/1/challenges/0>

- написать исполняемые скрипты на Ruby, решающие следующие задачи:

- а. Есть 4 ведра, каждое своего цвета: красное, зеленое, синее, жёлтое. В каждом лежат шары, определённого количества: 50, 100, 30, 60 соответственно.
 - Вывести на экран цвет ведра с максимальным количеством шаров.
 - Вывести на экран ответ на вопрос: больше ли шаров в зеленом ведре чем в жёлтом или в красном и синем вместе?
- б. Курс продажи доллара 11170 + 30% сбор.
 - Вывести на экран, сумму в белорусских рублях, необходимую для покупки 270 долларов

- Дать ответ на вопрос: Если у человека есть 5.600.000 белорусских рублей, сколько долларов он может на них купить, и хватит ли оставшихся рублей на мороженное (оно стоит 10.000)

Домашку присылать по адресу: aliaksandr_buhayeu@epam.com с темой письма: MTN:L_1:ИМЯ_ФАМИЛИЯ.

Домашка - это один Ruby-скрипт, в котором сверху в многострочном комментарии описаны команды для решения первых задач (по установке и решению возможных проблем), а после решения задач по работе с числами (уже вне комментариев).

На этом всё, жду вас на следующем занятии!