

Введение в Ruby

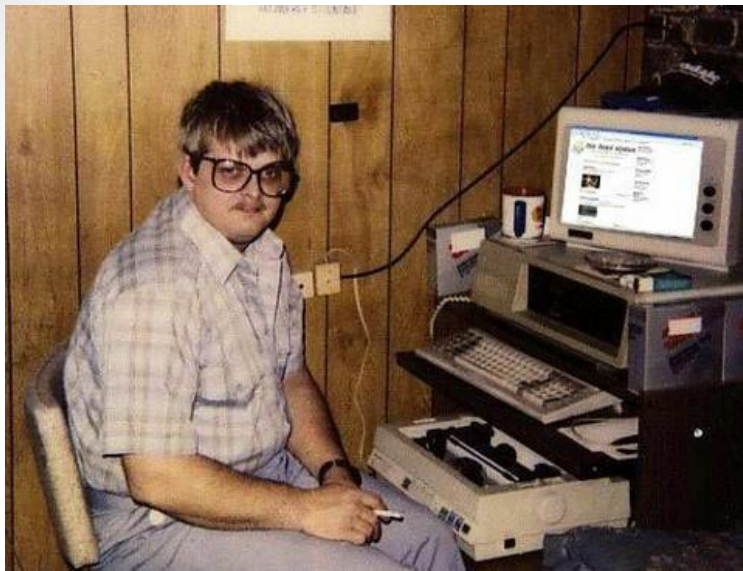


Общие сведения

- Высокоуровневый мультипарадигменный интерпретируемый язык программирования
- Автор – Юкихиро Мацумото
- Первая версия – 1995 г
- Основной упор сделан на ООП
- Цель создания Ruby – язык для быстрой разработки простых и понятных программ с хорошей читаемостью кода

Философия

- Любой результат можно достичь несколькими способами (один из принципов Perl)
- Закон наименьшего удивления – код должен работать так, чтобы вызывать у программиста как можно меньше удивления
- Дружелюбность предпочитается избыточности, например:
 - `size ~ length`
 - `indices ~ indexes`
- Последовательность и единообразие
 - Добавление символа «?» в конец имени предиката
 - Добавление символа «!» в конец имени метода, изменяющего состояние
- Не будьте рабом производительности, предпочитайте элегантность эффективности



Программист на Perl смотрит на тебя как на `$_[0]`

Программист на Haskell поднимает штангу именем теории категорий



Ruby-программисты – заботливые отцы, а значит их любят девушки

Важные особенности

- Нет примитивных типов, все типы – объекты
- Мощная реализация ООП
- Очень гибкая система итераторов
- Поддержка регулярных выражений, методы использования сходны с Perl
- Автоматически расширяемые числа
- Части кода являются объектами
- Поддержка замыканий
- Очень развитая работа с потоками
- Любой оператор возвращает значение, в т.ч. и управляющие структуры языка (if, case, ...)
- Слабая строгость синтаксиса, множество syntax sugar-фич

Основы синтаксиса

Комментарии и документация

- Комментарии начинаются с “#”

```
x = y + 10 # Комментарий
```

```
# Еще раз комментарий
```

```
puts "# Уже не комментарий"
```

- Документация пишется в блоках `=begin ... =end`

```
=begin
```

```
Этот метод метод делает всем хорошо
```

```
=end
```

Переменные и константы

- Имена
 - Локальных переменных начинаются со строчной буквы
 - Глобальные – со знака доллара “\$”
 - Константы – с заглавной буквы
 - Пример:

```
alpha = 3 # локальная переменная
$duTy = 1000 # глобальная переменная
ObjectForSitting = "Stump" # Константа
```
- Тип имеют не переменные, а объекты, на которые они ссылаются

Условный оператор

- Условный оператор существует в двух формах:
 - if – else
 - unless - else

Форма с if	Форма с unless
<pre>if x < 5 then statement1 end</pre>	<pre>unless x >= 5 then statement1 end</pre>
<pre>if x < 5 then statement1 else statement2 end</pre>	<pre>unless x >= 5 then statement1 else statement2 end</pre>
<pre>statement1 if y == 3</pre>	<pre>statement1 unless y != 3</pre>
<pre>x = if a > 0 then b else c end</pre>	<pre>x = unless a <= 0 then b else c end</pre>

Оператор выбора

- Оператор **case** в Ruby позволяет не только проверять объект на равенство, но и задавать различные условия, в т.ч. и в виде regex

```
case "Это одна строка символов"  
  when "одно значение"  
    puts "Ветвь 1"  
  when "другое значение"  
    puts "Ветвь 2"  
  when /СИМВ/  
    puts "Ветвь 3"  
  else  
    puts "Ветвь по умолчанию"  
  end
```

- Для чисел возможна проверка на вхождение в диапазон (например: 2..5)

ЦИКЛЫ

```
i = 0
while i < 5
  #code...
end
```

```
for x in list do
  #code...
end
```

```
loop do
  #code...
  break if condition
end
```

```
n.times do |i|
  #code...
end
```

```
for i in 0..10 do
  #code...
end
```

```
i = 0
until i == 5
  #code...
end
```

```
list.each do |x| #обход
  итератором
  #code...
end
```

```
loop do
  #code...
  break unless
  antiCondition
end
```

```
0.upto(n) do |i|
  #code...
end
```

```
list.each_index do |i|
  #code...
end
```

Числа

- Целые числа представлены двумя классами **Fixnum** и **BigNum**
 - **FixNum**: $-2^{30} + 1 \dots 2^{30} - 1$
 - **BigNum**: если число превышает диапазон FixNum
 - FixNum автоматически преобразуется в BigNum при выходе из диапазона и наоборот.
- Система счисления указывается в префиксе записи числа:
 - Десятичные – по умолчанию, префикса нет
 - Восьмеричные – **0**, например **046732**
 - Шестнадцатеричные – **0x**, например **0xac12**
 - Двоичные – **0b**, например **0b11011101**
- Для удобства записи в числах можно использовать подчеркивание, оно игнорируется:
 - **1_000_124**

Числа

- Числа с плавающей точкой представлены классом **Float**
 - 25.65 – обычная форма
 - 45.23e2 – экспоненциальная форма
- **Complex** и **Rational** (библиотека `mathn`)
 - Если **Complex** теряет мнимую часть, то число автоматически приводится к `FixNum`, `BigNum` или `Float`
 - Если у **Rational** знаменатель становится равным 1, то он преобразуется к `FixNum` или `BigNum`

Строки

- Строки представлены классом **String**
- Простые строки могут быть записаны в одиночных либо в двойных кавычках
 - Строки в одиночных кавычках воспринимаются буквально, в качестве управляющих символов в них распознаются только символы «\» и «\\», например:

```
s = 'Простая строка'
```

- Строки в двойных кавычках могут содержать и другие управляющие последовательности, а также интерполяционные выражения:

```
a = 5
```

```
b = 3
```

```
s = "#{a} + #{b} равно \t #{a+b}"
```

- Вместо символов кавычек можно использовать %q или %Q и символы ограничителя, например

```
s1 = %q[Это строка с "кавычками"]
```

```
s2 = %Q:Это тоже строка со 'спец' \t\n  
символами:
```

Строки

- Строки в обратных кавычках – особый тип строк (аналог обратных кавычек в bash)
- Строка в обратных кавычках посылается ОС в качестве команды, результат записывается обратно в строку
- Пример:

Код

```
s = `ls -a`  
puts s
```

Выведет

```
.  
..  
.idea  
qwe.rb
```

Встроенные документы

- Если использовать обычные строки для хранения многострочных текстов, то в них будут храниться все отступы
- Для хранения многострочных текстов используются встроенные документы
- Формат:

```
<<концевой_маркер  
куча_строк  
концевой_маркер
```


Встроенные документы

- Пример:

```
str = <<EOF
```

```
Всякие строки тут
```

```
По несколько штук за раз
```

```
Лезут и лезут
```

```
EOF
```

- Пример:

```
someMethod(<<str1, <<str2, <<str3)
```

```
Кусок
```

```
текста
```

```
str1
```

```
Второй кусок текста
```

```
str2
```

```
Третий
```

```
кусок текста
```

```
str3
```

Массивы

- Массивы в Ruby представлены одним классом – **Array**
- Особенности массивов в Ruby
 - Нет ограничений на размер
 - Гетерогенность – возможность хранить объекты произвольных типов
 - Возможность использования итераторов позволяет не использовать циклы
 - Операции над массивами записываются очень быстро и наглядно

Массивы

- Создание массивов

```
a = Array.[](1,2,3) #[1,2,3]
```

```
b = Array[1,2,3] #[1,2,3]
```

```
c = [1,2,3] #[1,2,3]
```

```
d = Array.new # пустой массив
```

```
e = Array(3) # [nil,nil,nil]
```

```
f = Array(3, "Test") # ["Test", "Test", "Test"]
```

```
h = ["Дверь", "запили", [1000, "Очень мало"]]
```

- Доступ и присваивание

```
a[0] = 5
```

```
b[2] = a # [1,[1,2,3],3] вложенный массив
```

```
c[-1] = 5 # [1,2,5] отрицательные индексы нумеруют с конца
```

```
d = a[1..2] # [2,3] выделили из a подмассив
```

```
a[1..2] = b[0..2] # заменяем подмассив из a подмассивом из b
```

Массивы

- Метод **at** получает ссылку на элемент массива
`x = a.at(2)`
- Метод **values_at** возвращает подмассив по списку индексов
`x = a.values_at(2..5, 7, 9)`
- **first** и **last** – получение первого и последнего элементов
- **size** и **length** – длина массива
- **nitems** – длина без учета nil-элементов
- Диапазоны можно задавать двумя способами
 - Две точки: `n..m` – диапазон `[n, m]`
 - Три точки: `n...m` – диапазон `[n, m-1]`

Если вы хотите продать **кошку**
специалисту по компьютерам, скажите,
что она **объектно-ориентированная**.

Роджер Кинг

ООП

Обзор

- Все сущности – объекты

`3.succ`

`"abc".upcase`

`[1, 4, 5, 2, 3].sort`

`obj.method1`

- Нет множественного наследования
- Вместо него используются примеси (mixin)
- Все объекты являются потомками класса **Object**

Классы

- Создание класса

```
class MyClass  
  #...  
end
```

- Имя класса – глобальная константа, ссылающаяся на объект типа Class
- Сущности класса:
 - Переменные класса (начинаются с @@) – аналог статических полей
 - Переменные экземпляра (начинаются с @) – обычные поля класса
 - Константы класса (начинаются с заглавной буквы) – аналог статических констант класса
 - Методы экземпляра
 - Методы класса (начинаются с названия класса + ‘.’)

Классы. Пример

```
class MyClass
  NAME = "My Class" # константа класса
  @@count = 0 # переменная класса
  def initialize # инициализатор
    @@count += 1
    @myVar = 500 # переменная объекта
  end
  def MyClass.getCount # метод класса - геттер
    @@count
  end
  def getMyVar # метод объекта - геттер
    @myVar
  end
  def setMyVar(val) # метод объекта - сеттер
    @myVar = val
  end
  def myVar=(val) # сеттер объекта в другой форме
    @myVar = val
  end
end
```


Управление доступом

- Все переменные всегда закрыты. Доступ только через методы
- Для методов есть модификаторы:
 - **public** – самый обыкновенный public, кто его не знает!
 - **private** – метод может вызываться только внутри класса или подклассов, и только в функциональной форме от имени **self**
 - **protected** – почти private, только не требует **self**
- После модификатора перечисляются методы, предваряемые двоеточием
- Если двоеточие опущено, модификатор действует на все последующие определения

Управление доступом. Пример

```
class MyClass
  def method1
    #...
  end
  def method2
    #...
  end
  def method3
    #...
  end
  private :method1, :method2 #закрытые
  protected :method3 # защищенный
  public # открыто все, что ниже
  def method4
    #...
  end
end
```

Акцессоры

- Акцессоры – механизм, облегчающий доступ к полям объекта – аналог автосвойств в C#

<code>attr_reader :name</code>	<code>def name @name end</code>
<code>attr_writer :name</code>	<code>def name=(val) @name = val end</code>
<code>attr_accessor :name</code>	<code>attr_reader :name attr_writer :name</code>
<code>attr_accessor :name, :age</code>	<code>attr_accessor :name attr_accessor :age</code>

Наследование

- Синтаксис

```
class InterestingPerson < Person
  def sawTheDoor
    #...
  end
end
```

- При совпадении имен методы перекрываются
- Если требуется вызвать перекрытый метод, используется ключевое слово **super**

Модули

- Модули – некоторый аналог классов, за исключением:
 - модуль не может иметь представителей
 - модуль не может иметь подклассов
- Класс **Module** является суперклассом класса **Class**
- Модули используются для двух целей
 - хранение методов и констант (аналог статических классов или пространств имен)
 - создание миксинов (mixin)

Миксины (примеси)

- Примесь – механизм расширения класса, заменяющий множественное наследование

```
module AvailableForSitting
  @@price = 1000
  def sit
    puts "You have to pay #{@@price}"
  end
end

class Stump
  include AvailableForSitting
  #...
end

x = Stump.new
x.sit
```

Замыкания

Блоки кода

- Замыкание – функция, определенная в теле другой функции и имеющая доступ к локальным переменным внешней функции
- В Ruby есть несколько типов замыканий, один из которых – **блоки кода**
- Блок кода – кусок кода, окруженный {...} или **do ... end**
- Блоки кода могут принимать параметры, например

```
y = 1000  
list = [1,2,3]  
list.each { |x| puts x, puts y }
```


Блок как объект

- Блок кода можно обернуть в объект, используя класс **Proc**

```
myProc = Proc.new { |a| puts a }  
myProc.call(500)
```

- Объект класса **Proc** можно передать методу, принимающему блок, поставив перед именем переменной амперсанд

```
myProc = Proc.new { |a| puts a }  
(1..3).each(&myProc)
```

Интересности

Операторы

- Оператор **тройного равенства** (**===**) используется в выражении **when** конструкции **case**. Его можно перегрузить, например, для проверки вхождения числа в диапазон
- Оператор **космический корабль** (**<=>**) сравнивает объекты и возвращает:
 - **-1**, если левый меньше правого
 - **0**, если они равны
 - **1**, если, левый больше правого
- Параллельное присваивание
 - `a, b, c = 10, 20, 30`

Поэтический режим

- Термин «Поэтический режим» означает, что можно опускать многие знаки препинания и лексемы

- При вызове функций можно опускать скобки

```
method(1, 2, 3)
```

```
method 1, 2, 3
```

- При определении методов также можно опускать скобки

```
def method(a, b, c) #...
```

```
def method a, b, c #...
```

- Можно опускать **then** в конструкциях **if** (не всегда)
- Можно опускать скобки при каскадном вызове функций

```
method1(method2(method3(x)))
```

```
method1 method2 method3 x
```

- Не стоит злоупотреблять, интерпретатор может понять неправильно

Что читать

- (!!!)Хэл Фултон «Программирование на языке Ruby»
- Учебник на википедии
- Несколько толковых статей на opennet.ru
- www.rubyinside.com



Bce!