Быстрое знакомство с Ruby

Андрей Васильев

2018

Ruby - объектно-ориентированный язык

- Объект - некоторый базовый элемент приложения, имеющий состояние и методы, использующие это состояние
- Для описания объектов вы определяете класс
- Для создания экземпляров классов (объектов) используется специальный метод new, конструктор

Музыкальный автомат

Создаём две переменных song1 и song2, записываем в них ссылки на созданные объекты

```
song1 = Song.new('Калинка малинка')
song2 = Song.new('За десантный батальон')
```

Объекты

- У каждого объекта есть уникальный идентификатор
- Объекты могут содержать переменные экземпляра, описывающие состояние конкретного объекта
- Методы объекта
  - Могут изменять переменные экземпляра
  - Могут быть публичными и приватными
  - Вызов метода - отправка сообщения объекту

Обработка сообщений объектом (вызов метода)

- Объект ищет метод по имени
- Когда метод найден, он вызывается

Метод puts

puts выводит переданные аргументы на стандартном потоке

```
puts "gin joint".length
puts "Rick".index("c")
puts 42.even?
puts sam.play(song)
```

- Часть кода до точки называется получателем
- После точки указывается имя метода
- Числа в Ruby содержат методы и могут их обрабатывать

```
num = -1234
num.abs
```

Простейший метод

```ruby
def say_goodnight(name)
  result = "Good night, " + name
  return result
end
# Вызываем метод
puts say_goodnight("John-Boy")
puts say_goodnight("Mary-Ellen")
```

- Выражения на разных линиях не надо разделять с помощью точки с запятой
- Комментарии начинаются с символа #
- При объявлении переменные инициализируются

Определение метода

- Определение начинается с ключевого слова `def`
- Затем следует имя метода, слова в названии которого разделены подчёркиваниями
- В круглых скобках перечисляются аргументы
- Тело метода завершается ключевым словом `end`
- Результат работы возвращается с помощью `return`

```
def say_goodnight(name)
  result = "Good night, " + name
  return result
end
```

Вызов метода

- Передавайте методу столько обязательных аргументов, сколько он ожидает от ввас
- При вызове метода можно не указывать круглые скобки:

```
puts say_goodnight("John-Boy")
puts(say_goodnight("John-Boy"))
```

Обычно скобки опускаются только в самых простых случаях, исключающих различные трактования:

- Если метод не имеет аргументов, не указывайте скобки
- Если метод имеет аргументы, указывайте скобки за исключением методов `puts` и `p`

Строки

- Для определения строк обычно используются литералы
- Литералы описываются с помощью одинарных и двойных кавычек
- Строки в одинарных кавычках не обрабатываются
- Обработка строк в двойных кавычках
  - Замена специальных символов (\n, \t)
  - Заполнение результатов выражений

```
def say_goodnight(name)
  result = "Good night, #{name.capitalize}"
  return result
end
```

Возвращаемое значение

- Для возвращения значения из метода используется ключевое слово `return`. Обычно используется для возвращения значения в начале метода.
- Результат вычисления последнего выражения является значением по умолчанию.

```ruby
def say_goodnight(name)
  "Good night, #{name.capitalize}"
end
```

Наименования элементов

Первый символ описывает назначение элемента языка

- Имена локальных переменных, параметров методов и имена методов должны начинаться со строчной буквы или подчёркивания
- Имена глобальных переменных начинаются со знака $
- Имена переменных экземпляра, начинаются со знака @
- Имена переменных класса, начинаются с @@
- Имена классов, модулей и констант начинаются с заглавной буквы
- Имена методов могу заканчиваться символами ?, !, =
- Слова в именах переменных разделяются подчёркиваниями
- Слова в названиях классов начинаются с заглавной буквы

Массивы и ассоциативные массивы

- Массивы являются индексируемыми коллекциями
  - Обычные массивы индексируются числами
  - Ассоциативные массивы - любым объектом
- Коллекции являются динамическими
- В коллекции могут находится любые объекты

Массивы

```
a = [1, 'cat' , 3.14]
puts "The first element is  #{a[0]}"
# Установим новое значение 3 элементу
a[2] = nil
puts "The array is now  #{a.inspect} "
```

Создание массивов

nil

- Во многих языках программирования есть понятие «отсутствующего значения», null
- nil - объект, обозначающий отсутствие значения

```
a = ['ant' , 'bee' , 'cat' , 'dog' , 'elk']
a[0] # => "ant"
a[3] # => "dog"
# Равнозначно
a = %w{ant bee cat dog elk}
a[0] # => "ant"
a[3] # => "dog"
```

Ассоциативные массивы

- Для создания литералов ассоциативных массивов используются фигурные скобки
- Элементами описания являются пары ключ-значение, разделённые =>: ключ => значение
- Ключи должны быть уникальными для массива
- Пары разделяются друг от друга запятыми

```
inst_section = {
  'cello' => 'string',
  'clarinet' => 'woodwind',
  'drum' => 'percussion'
}
print inst_section['cello']
```

Значение по умолчанию

- Если обратиться к несуществующему элементу, тогда будет возвращено значение `nil`
- Иногда желательно, чтобы массив возвращал осмысленное значение по умолчанию

```ruby
# Создаём массив со значением по умолчанию
histogram = Hash.new(0)
histogram['ruby'] # => 0
histogram['ruby'] = histogram['ruby'] + 1
histogram['ruby'] # => 1
```

Символы

Часто, при написании приложения, нам необходимо описать уникальные элементы и описывать их с помощью слов

```
NORTH = 1
EAST = 2
```

Символы в ruby описывают константы, которые не надо определять заранее и гарантировано будут уникальными

```
walk(:north)
def walk(direction)
  if direction == :north
    # ...
end
```

Символы как ключи хешей

Символы часто используются в качестве ключей хешей

```
inst_section = {
  :cello => 'string',
  :clarinet => 'woodwind'
}
```

Поддержка таких хешей добавлена в синтаксис языка

```
inst_section = {
  cello: 'string',
  clarinet: 'woodwind'
}
```

Управляющие конструкции

Ruby поддерживает знакомые вам конструкции

- `if` для описания условной логики
- `while` для описания циклических действий

```ruby
today = Time.now
if today.saturday?
  puts "Do chores around the house"
elsif today.sunday?
  puts "Relax"
else
  puts "Go to work"
end
```

Блок описания заканчивается ключевым словом `end`

Выражения в условиях

Любое выражение может являться условием. Ложными значениями являются только `false` и `nil`. Все остальные значения являются верными.

```
while weight < 100 && num_pallets <= 5
  pallet = next_pallet()
  weight += pallet.weight
  num_pallets += 1
end
```

При достижении конца файла `gets` возвращает `nil`

```
while line = gets
  puts line.downcase
end
```

Модификаторы выражений

Условные операторы могут выступать в роли модификатора выражения

```ruby
if radiation > 3000
  puts "Danger, Will Robinson"
end
puts "Danger, Will Robinson" if radiation > 3000
```

Аналогично для операторов цикла

```ruby
square = 4
while square < 1000
  square = square*square
end
square = square*square while square < 1000
```

Регулярные выражения

- Регулярное выражение - способ описать шаблон, которому должна соответствовать строка
- Реализованы как часть языка, не библиотека
- Регулярные выражения - объекты
- `/Perl|Python/` - либо строка целиком `Perl`, либо целиком `Python`
- `/ab+c/` - a, за которым следует несколько b, затем c
- Выражения можно сравнивать со строками с помощью `~=`
- Метод `sub` строки позволяет заменять подстроки

```
line = gets
newline = line.sub(/Perl/, 'Ruby')
newerline = newline.gsub(/Python/, 'Ruby')
```

Блоки и итераторы

- Вы можете ассоциировать некоторый блок кода с методом
- Блоки могут быть использованы для реализации
  - Обратных вызовов `callbacks`
  - Передачи управления в произвольных местах кода
  - Реализации итераторов

Примеры блоков

```
{ puts "Hello" } # Для одного выражения
do # Для нескольких выражений
  club.enroll(person)
  person.socialize
end
```

Ассоциация блока с методом

Блок всегда ассоциируется в конце вызова метода

```
greet { puts "Hi" }
verbose_greet("Dave") { puts "Hi" }
```

Вызов ассоциированного блока кода

Вызов осуществляется с помощью ключевого слова `yeild`

```
def call_block
  puts "Start of the method"
  yield
  puts "End of the method"
end
call_block { puts "In the block" }
```

Аргументы блоков

- Блокам могут быть переданы агрументы
- Аргументы описываются между вертикальными линиями

```ruby
def who_says_what
  yield ("Dave", "hello")
  yield ("Andy", "goodbye")
end
who_says_what {|person, phrase| puts " #{person}
```

Итераторы

Коллекции поддерживают множество итераторов

```ruby
animals = %w(ant bee cat dog)
animals.each { |animal| puts animal }
3.upto(6) { |i| print i }
('a' .. 'e').each { |char| print char }
```

Ввод и вывод в Ruby

- `puts` записывает свои аргументы на новой строке
- `print` не добавляет перевод на новую строку
  - `printf` вывод в форматированном виде, как в Си
- `gets` считывает строку с потока стандартного ввода

```
while line = gets
  print line
end
```

Обработка аргументов командной строки

Доступ к списку аргументов

Массив ARGV содержит список строк, которые были переданы в виде аргументов командной строки

```
puts "You gave  #{ARGV.size}  arguments"
p ARGV
```

Обработка файлов, переданных приложению

Если приложению необходимо обработать несколько файлов, то все они объединяются в специальный объект ARGF