

# О языке программирования

# Ruby



*расскажет М. В. Шохирев*

в шадринском Клубе программистов

2020

# Ruby: пробуйте примеры по ходу рассказа

TryRuby — интерактивный интерпретатор с редактором в web-браузере для знакомства с языком

[ruby.github.io/TryRuby/](http://ruby.github.io/TryRuby/)

**IRB** (Interactive Ruby Shell) — интерактивная командная оболочка для программирования на Ruby

**> irb**

*«The beauty of Ruby is found in its balance between simplicity and power.»*



# Язык программирования Ruby

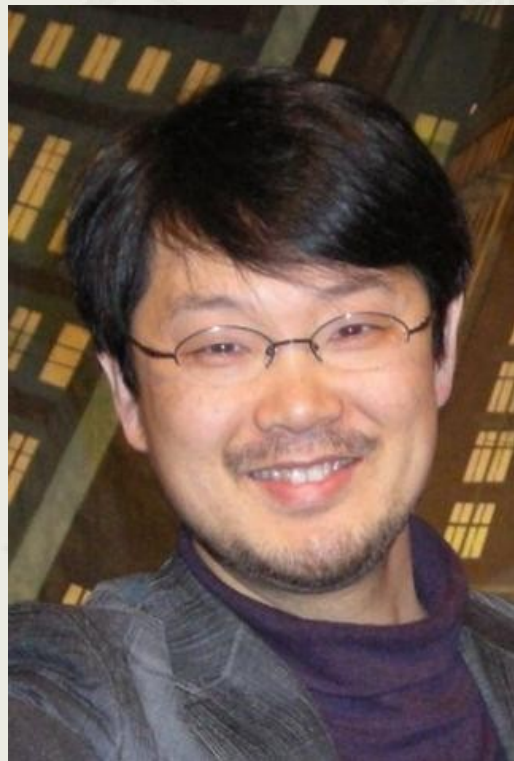
**Ruby** — интерпретируемый динамический полностью объектно-ориентированный высокоуровневый язык программирования. Разработан под влиянием языков Perl, Smalltalk, Eiffel, Ada и Lisp, а также C++, CLU, Dylan, Lua и Python.

**Ruby** поддерживает несколько парадигм программирования: **процедурную** (определение подпрограмм и переменных вне классов), **объектно-ориентированную** (всё является объектами), **функциональную** (анонимные функции, замыкания, возврат значения всеми инструкциями, возврат методом последнего вычисленного значения, функции высшего порядка), **аспектно-ориентированную** (AOP).

В нём есть мощные средства интроспекции (reflection), реализована независимая от ОС многопоточность и эффективный сборщик мусора, имеются мощные средства мета-программирования.



# История создания ルビー



Название языка навеяно языком `Perl`, многие особенности синтаксиса и семантики из которого заимствованы в `Ruby`.

Одним из источников вдохновения для Ю. Мацумото при разработке `Ruby` был научно-фантастический роман «Вавилон-17» Сэмюэля Дилэни, основанный на гипотезе лингвистической относительности Сепира — Уорфа (язык определяет мышление, и, соответственно, лингвистические категории ограничивают и определяют когнитивные категории).

Целью разработки было создание настоящего объектно-ориентированного интерпретируемого мощного и удобного языка программирования, лёгкого в изучении, который приятно использовать, на котором можно быстро разрабатывать.

Matz создал сбалансированный гармоничный язык с ясным удобным синтаксисом, сочетающий преимущества объектно-ориентированного программирования с достоинствами других парадигм.

**Matsumoto** «Matz» Yukihiro ( まつもとゆきひろ = 松本行弘 )



# Хронология

**1993** — Matz начал разработку нового языка.

**1993.02.24** — выбрано имя "**Ruby**", а не "Coral" в online-беседе между Matsumoto Yukihiro и Keiju Ishitsuka.

**1995** — публикация языка **Ruby** в японском списке рассылки **ruby-list**.

**с 1995 по 1999** — известен только в Японии.

**1997** — 1-я статья о **Ruby** на английском опубликована в WWW.

**1997** — Matz нанят в open source-компанию **netlab.jp** на полный рабочий день для работы над Ruby.

**1998** — создан RAA (Ruby Application Archive) и домашняя страница для Ruby на английском языке.

**1999** — 1-й список рассылки **ruby-talk** на английском.

**1999.10** — 1-я книга «オブジェクト指向スクリプト言語 Ruby» (The Object-oriented Scripting Language Ruby).

**начало 2000-х** — в Японии опубликовано около 20 книг.

**2000.09** — 1-я книга на английском: «Programming Ruby» (**PickAxe**), авторы: Dave Thomas & Andy Hunt.

**к 2002** — количество сообщений в англо-язычном **ruby-talk** превысило объём **ruby-list** на японском языке.

**2005.12.13** — выходит web-framework **Ruby on Rails**, сильно повлиявший на web-разработку.

**2006** — **Ruby** стал Programming Language of the Year @ TIOBE Index.

**2007** — «Программирование на языке Ruby», перевод 2-го издания «The Ruby Way» (Hal Fulton).

**2011** — Matz поступил на должность Chief Architect of Ruby @ **Heroku** (США).

**2011** — Matz получил награду от Free Software Foundation (FSF) For Advancement of Free Software.

**2011** — Японский национальный промышленный стандарт JIS X 3017:2011 по языку Ruby

**2012** — Международный стандарт **ISO/IEC 30170:2012** по языку Ruby.

**2016.05** — Ruby #8 @ TIOBE Index (← #39@2002.01).

**2020.02** — Ruby #15 @ TIOBE Index (← #16@2019.02).



# Версии

1995.12.21 — Ruby 0.95 (начальная версия)  
1996.12.25 — Ruby 1.0 (@ MRI)  
1998.12 — 1.2  
1999.08 — 1.4  
2000.09 — 1.6  
2003.08.04 — Ruby 1.8 (основная версия ветки 1.x)  
2007.12.25 — 1.9 (переходная версия к 2.x @ YARV)  
2013.02.24 — Ruby 2.0 (новая эволюционная ступень)  
2013.12.25 — 2.1  
2019.04.01 — 2.4 (сопровождаемые версии)  
2017.12.25 — 2.5  
2018.12.25 — 2.6 (JIT)  
2019.12.25 — Ruby 2.7 (текущая стабильная версия)  
2020 — Ruby 3.0 (увеличение скорости в 3 раза по проекту 3x3)

*Начиная с первых разрабатываемых версий в **Ruby** присутствовали основные возможности, знакомые по более поздним версиям языка, включая объектно-ориентированный подход, примеси (mixin), итераторы, замыкания (closure), обработка исключений и сборка мусора.*

# Реализации

## Реализации интерпретатора:

- **Основная:**  
**CRuby/MRI** (Matz's Ruby Interpreter) → **YARV** (Yet Another Ruby VM)
- **Альтернативные:**  
**JRuby**@JVM, **Rubinius** @ LLVM, **TruffleRuby** @ GraalVM
- **Особые:**  
**mruby** (embeddable), **RGSS** (Ruby Game Scripting System),  
**MagLev** (Smalltalk@GemStone/S VM)
- **Устаревшие:**  
**MacRuby/RubyMotion** (Mac OS X/iOS), **IronRuby** (@.NET), **Cardinal** (@ParrotVM),  
**REE** (Ruby Enterprise Edition)
- **Экзотические:**  
**Topaz** (@Python), **Opal** (@JavaScript), **HotRuby** (@JavaScript & ActionScript)

Реализации языка **Ruby** – свободно распространяемые (под лицензиями Ruby License, GPLv2, 2-clause BSD License).

Операционные системы: NEWS-OS, SunOS, SVR4, Solaris, NEC UP-UX, HP-UX, NeXTstep, BSD, **Linux**, Raspbian, MacOS, iOS, BeOS, DOS, **MS Windows**, Windows Phone, Windows CE, Symbian OS, AIX, IBM i.

# Принципы

- **Язык для программиста, а не для компьютера:**  
*приоритетны удобство, минимизация трудозатрат и производительность программиста.*
- **Просто, но не слишком просто:**  
*упрощение — благо, но не самоцель, которая может вредить конечному результату.*
- **Принцип наименьшей неожиданности (POLS / POLA):**  
*программные конструкции означают именно то, что ожидает программист.*
- **Ортогональность важна, но естественность важнее (TIMTOWTDI):**  
*избыточность допустима, если она удобна.*
- **Производительность разработки важнее эффективности выполнения:**  
*следует предпочитать элегантность и мощь эффективности, когда она не критична.*
- **Не бояться изменений во время выполнения:**  
*динамические средства языка во время исполнения дают очень эффективные возможности.*
- **Следовать простым и строгим правилам, но не доходить до педантизма:**  
*если отступление от принятых правил и соглашений логично и понятно, оно оправдано.*
- **«Не нужно с этим бороться»:**  
*если ваши ожидания о языке оказываются неверны, это нужно просто принять и использовать.*

*"Я надеюсь увидеть, как Ruby помогает каждому программисту в мире стать плодотворным, наслаждаться программированием и быть счастливым. Это и есть основное назначение языка Ruby."*

**Matz**





# Особенности

- Полностью **объектно-ориентированный**: inheritance, polymorphism, incapsulation, mixins, metaclasses
- Поддерживает несколько парадигм программирования: **imperative, object-oriented, functional, aspect-oriented**
- **Строгая динамическая типизация и duck typing**
- **Всё является выражениями** (даже предложения)
- **Все описания исполняются** (даже объявления)
- **Краткий и гибкий синтаксис** минимизирует syntactic noise, служит основой для DSLs
- **Динамическая интроспекция и изменение** объектов во время выполнения для мета-программирования
- Лексические **замыкания, итераторы, генераторы** с блочным синтаксисом
- Нотация для литералов: arrays, hashes, regular expressions, symbols
- Встраивание кода в строки (**интерполяция выражений**)
- Аргументы по умолчанию, именованные параметры, список параметров переменной длины в методах
- **4 уровня областей видимости переменных** (обозначаемых через *sigils*): `$global`, `@@class`, `@instance`, `local`
- Сборщик мусора (Compaction GC)
- Строгие правила приведения boolean: всё является `true`, кроме `false` и `nil`
- Обработка исключений (Exception handling)
- **Перегрузка операций**: `=`, `+`, `-`, `*`, `/`, `<=>` и т. д.
- Встроенная поддержка рациональных и комплексных чисел
- **Арифметические вычисления произвольной точности** с автоматическим преобразованием `Fixnum` ↔ `Bignum`
- Пользовательская диспетчеризация: `method_missing`, `const_missing`
- **Независимая от ОС многопоточность**: **native threads, cooperative fibers** (@1.9/YARV)
- **Полноценные продолжения** (First-class continuations), currying = partial application of functions
- API на C для разработки plugin-ов
- Интерактивная командная оболочка: Interactive Ruby Shell (`irb` / REPL)
- Централизованное управление пакетами: **RubyGems**
- **Большая стандартная библиотека**: YAML, JSON, XML, CGI, OpenSSL, HTTP, FTP, RSS, curses, zlib, Tk
- Поддержка **Unicode** и кодировок с многобайтовыми символами
- Реализован для всех основных платформ



# Синтаксис: пример

```
#!/usr/bin/ruby
while true do                                     # на основе примера из «The Ruby Way»
  print "Введите температуру и шкалу (C или F): "
  STDOUT.flush
  string = STDIN.gets(); string.chomp!
  exit if string.nil? || string.empty?
  temp, scale = string.split(" ")
  if temp !~ /-?\d+/ then puts "#{temp} - некорректное число."; next; end
  temp = temp.to_f
  case scale
  when "C", "c" then
    f = 1.8*temp + 32; c = nil
  when "F", "f"
    c = (5.0/9.0)*(temp-32); f = nil
  else
    puts "Укажите шкалу C или F через пробел от числа!"; next
  end
  if f.nil?
    puts "#{temp}°F - это #{sprintf('%5.1f', c)} градусов C"
  else
    print "#{temp}°C == #{f}°F \n"
  end
end
end
```

# Ruby: КЛЮЧЕВЫЕ СЛОВА

**alias**  
**and**

**BEGIN**  
**begin**  
**break**

**case**  
**class**

**def**  
**defined?**  
**do**

**else**  
**elsif**  
**END**  
**end**  
**ensure**

**false**  
**for**

**if**  
**in**

**module**

**next**  
**nil**

**not**  
**or**

**redo**  
**rescue**  
**retry**  
**return**

**self**  
**super**

**then**  
**true**

**undef**  
**unless**  
**until**

**when**  
**while**

**yield**

**\_\_ENCODING\_\_**  
**\_\_END\_\_**  
**\_\_LINE\_\_**  
**\_\_FILE\_\_**

По ключевым словам языка можно сделать предварительный вывод о том, на какие другие языки похож синтаксис этого языка.

Похоже, синтаксис **Ruby** напоминает и **Pascal** (**begin**, **end**, **nil**) и **Perl** (**BEGIN**, **END**, **undef**, **unless**) а чем-то **Python** (**def**).

Большинство ключевых слов типичны для многих популярных языков: **break**, **class**, **else**, **elsif**, **for**, **if**, **self**, **super**, **true**, **while**, ...

Хорошо, что в **Ruby** немного необычных ключевых слов: на первый взгляд всё более или менее привычно.

# Синтаксис: знакомый

```
array = [0, true, 2.0, 1+2, 0.4e1, '五', [6]] # массив
hash = {2=>[0,0], 'пять'=>5.0, true=>'истина', []=>nil} # ассоциативный массив

if (array[1] > array[0]) then # ветвление
  print("По возрастанию.\n")
else
  puts("По убыванию.")
end

n = 0 # целое
radius = 10.0 # дробное
while (radius <= 50) do # цикл пока истинно
  circle_length = 2 * Math::PI * radius # присваивание
  printf("%d R=%f, L=%f\n", n, radius, circle_length)
  radius += 10; n += 1
end

until (radius > 50) do # цикл до истины
  # тело цикла
end

x += 1 # нет операций ++ и --
```

# Синтаксис: странный

# Диапазоны чисел (класс Range):

```
range1 = 1..5  
range2 = 1...5
```

# включающий: 1, 2, 3, 4, 5  
# исключающий: 1, 2, 3, 4

# Символы (класс Symbol) - уникальные неизменяемые значения для именования:

```
symbol_is_not_a_string = :value # скаляр  
colors = {red: 0xFF0000, yellow: 0xFFFF00, green: 0x00FF00} # ключи в хэше
```

# Подстановка в строку #{значения выражения}:

```
puts("Число  $\pi$  = #{Math::PI}")
```

```
unless (array[1] > array[0]) then  
  print("По убыванию.\n")  
end
```

# ветвление: unless == if not

```
a <=> b  
(1..100) === 25  
array[-1]  
"Да!" * 2  
[a, b].min  
puts "OK!" if "Ruby" =~ /by/
```

# сравнение: -1, 0, +1  
#== (1..100).include?(25)  
# 1-й с конца == последний элемент  
#=> "Да!Да!"  
# метод объекта типа Array  
# !~ отрицание =~

# Синтаксис: особенный

```
Constant; ClassName                                # константы, в т. ч. имена классов
$global_variable; $NOT_CONSTANT                   # глобальные переменные
@object_variable; @instance_attribute             # переменные объекта (атрибуты)
@@class_variable; @@class_property               # статические переменные класса
variable_local                                     # локальные переменные
method_name()                                     # методы

# Методы, оканчивающиеся на ? (предикаты), возвращают true или false
array.empty?
number.kind_of?(Numeric)

# Методы, оканчивающиеся на !, изменяют значение объекта (побочный эффект)
string = " Ruby "; string.strip!                  #=> "Ruby"
a = ["a", nil, "b", nil, "c"]; a.compact!          #=> ["a", "b", "c"]

# операция добавления в конец
list << element                                    #== list.push(element)
string << character                                #== string += character
STDERR << "Ошибка!"                               #== STDERR.print("Ошибка!")

# Арифметические вычисления произвольной точности: Fixnum → Bignum
number = 1_000_000_000_000*1_000
```

# Синтаксис: удобный

# Условные модификаторы:

z = x / y if y != 0

z = x / y unless y == 0

x = x \* 2 until x > 100

# если ...

== if not

== while not

a, b, c = 1, 2, 3

b, a = a, b

# параллельное присваивание

# обмен значениями

# оператор безопасной навигации &. для обработки значений nil

if robot&.motors[n]&.on? == robot && robot.motors[n] && robot.motors[n].on?

# Множественный выбор case

case expression

when 'something concrete' then ...

when SomeClass then

when /matches RegExp/ then ...

when (range1...range2)

...

when some\_expression >= some\_value then ...

else ...

end

# Массивы: удобные методы

```
a = [1, 2, 3, 4, 5]
b = Array.new [3, 4, 6, 7, 8, 9]
# Массивы как множества
# + объединение (union) без удаления дубликатов
p a + b # [1, 2, 3, 4, 5, 3, 4, 6, 7, 8, 9]
# | объединение (union) с удалением дубликатов
p a | b # [1, 2, 3, 4, 5, 6, 7, 8, 9]
# & пересечение (intersection)
p a & b # [3, 4]
# - разность (difference), отрицание (negation) или дополнение (complement)
p a - b # [1, 2, 5]
p b - a # [6, 7, 8, 9]
# |= накопление (accumulate): a = a | b
p (a |= b) # [1, 2, 3, 4, 5, 6, 7, 8, 9]
# &= сужение (narrow): a = a & b
p (a &= b) # [3, 4, 6, 7, 8, 9]

a.push(42)[-1] == a.pop()
first = a.shift
a[2,3] == a.slice(2,3)
a[2..4] == a.slice(2..4)
a.sample

# массив как Stack
# массив как Queue
# подмассив 3-х элементов со 2-го
# подмассив со 2-го по 4-й
# a.shuffle.first, a[rand(a.size)]
```



# Методы: удобные параметры

# Параметры по умолчанию:

```
def ping(host="127.0.0.1", times=3)
```

```
  `ping -c #{times} #{host}`
```

```
end
```

```
# `` выполнение внешней программы
```

# Список параметров переменной длины: последние сжимаются (splat) в массив:

```
def variable_args(head, *tail)
```

```
  print tail.size, tail, "\n"
```

```
end
```

```
variable_args(1, 2, 3) #=> 2 [2,3]
```

```
variable_args("ichi", "ni", "san", "shi", "go") #=> 4 ["ni","san","shi","go"]
```

# Именованные параметры:

```
def deliver(from: "A", to: nil, via: "e-mail")
```

```
  "Отправить из #{from} в #{to} по #{via}."
```

```
end
```

```
deliver(to: "B") #=> "Отправить из А в В по e-mail."
```

```
deliver(via: "факсу", from: "B", to: "A") #=> "Отправить из В в А по факсу."
```

# Синтаксис: функциональный

```
# любая конструкция возвращает значение
UltimateQuestion(); 42;                                     # выражения
var1 = var2 = 25                                           #== var2 = 25; var1 = var2
puts(if var1 > var2 then "Так" else "Иначе" end)           #=> Иначе

# все методы - это функции
# возвращаемое значение метода - это последнее вычисленное выражение
def AnswerToTheUltimateQuestionOfLifeTheUniverseAndEverything(question)
  answer = findTheAnswerToThe(question)
  42                                                         #== return 42
end

# цепочки вызовов функций
print 'А роза упала на лапу Азора'.upcase.split('').reverse.join('')
```

# Синтаксис: лёгкий

Несмотря на некоторые необычности  
(которые совсем не обязательно использовать)  
синтаксис **Ruby** – простой, понятный,  
лёгкий для изучения и понимания,  
часто даже без пояснений.

**Ruby** хорошо подходит для обучения,  
в том числе, как первый язык программирования.



# Синтаксис: поэтический (*poetry style*)

```
expression1; expression2
```

# ; не обязательны в конце строки

```
if condition  
  action  
end
```

# then можно не писать в if, unless, when  
# () не обязательны в if, unless, while, until

```
while condition  
  action  
end
```

# do можно не писать в while и until

```
print list unless list.empty?
```

# () не обязательны при вызове методов

```
p = Language.new name: "Ruby"
```

# {} не обязательны, если параметр - hash

```
array.size == array.length  
p(x) == puts(x.inspect)  
puts(x) == print(x, "\n")
```

# есть синонимы методов, можно добавлять:  
# alias :new\_name :old\_name

# иногда много скобок и других знаков затрудняют чтение исходника

# DSL = Domain-Specific Language

При помощи «поэтического стиля»  
и других конструкций языка  
разрабатываются удобные прикладные  
***языки описания предметной области.***

Фактически **DSL** – это набор методов  
с удачно выбранными именами  
и параметры, записанные без лишних знаков препинания.



# DSL @ Tello quadcopter

# DSL для квадрокоптера Tello: @ <https://github.com/blacktm/tello>

connect

takeoff

speed 75

up 150; down 50

left 250; right 75

forward 500; backward 200

cw 90

cw 360

ccw 360\*10

if height > 50

    flip :left; flip :right

    flip :forward; flip :backward

end

go 25, 35, 45, 55

land if battery < 10

# взлететь

# установить скорость 75 см/сек.

# подняться на 1.5 м; опуститься на 50 см

# лететь налево 250 см; направо 75 см

# лететь вперёд 5 м; назад 2 м

# повернуть по часовой на 90°

# повернуться по часовой на полный круг

# вращаться против часовой 10 раз

# если высота дрона > 50 см

# перекувырнуться налево и направо

# перекувырнуться вперёд и назад

# лететь в точку x, y, z со скоростью 55

# приземлиться при уровне заряда < 10%

# DSL на русском :-)

если ("Главный Вопрос Жизни, Вселенной и Вообще.") { ответить 42 }

def ответить(x); print x; end

def если(condition); yield if condition; end

# DSL @ Rails: СВЯЗИ В ORM

```
# модели @ Ruby on Rails
class Department < ActiveRecord::Base
  belongs_to :organization # 1:1
  belongs_to :leader, class_name: 'Worker', foreign_key: 'leader_id' # 1:1
  has_many :workers # 1:N
end

class Worker < ActiveRecord::Base
  belongs_to :department # N:1
  belongs_to :boss, class_name: 'Worker', foreign_key: 'boss_id' # N:1
  has_many :project_members # 1:N
  has_many :projects, through: :project_members # 1:N
end

class ProjectMember < ActiveRecord::Base # N:M
  belongs_to :project
  belongs_to :worker
end

class Project < ActiveRecord::Base
  has_many :project_members # 1:N
  has_many :workers, through: :project_members # 1:N
end
```

# DSL @ RSpec: описание тестов

```
describe "Заказ" do
  describe "Добавление заказа" do
    before do
      @book = Book.new title: "The Ruby Way", price: 25.5
      @customer = Customer.new
      @order = Order.new @customer, @book
      @order.submit
    end
    describe "order" do
      it "is marked as complete" do
        expect(@order).to be_complete
      end
      it "is not yet shipped" do
        expect(@order).not_to be_shipped
      end
    end
  end
  describe "Заказчик" do
    it "помещает заказанную книгу в свой список заказов" do
      expect(@customer.orders).to include(@order)
      expect(@customer.ordered_books).to include(@book)
    end
  end
end
end
end
```



# Блок = nameless function

Блок – последовательность команд между { }, которую можно передать в метод для выполнения (после списка параметров метода).

*Так можно дополнять функциональность метода разными действиями, не изменяя сам метод.*

```
method1(param1) { sequence; of; expressions } # передать блок методу
```

```
def method1(param1)  
  yield  
end
```

```
# выполнить переданный блок
```



# Блок = nameless function

У блока может быть свой список параметров,  
который заключается между `|` `|` .

Многострочный блок удобно заключать между `do` и `end`.

```
method2(param2) do |block_param|  
  perform; something; with(block_param)  
end
```

# передать блок методу

```
def method2(param2)  
  block_parameter = process(param2);  
  result = yield(block_parameter)  
end
```

# выполнить блок с параметром



# Блоки и методы

# Методу после списка параметров можно передать {блок команд} для выполнения

```
def calc_numbers_in_range(number1, number2)
  result = 0.0
  for i in (number1..number2) do
    result += yield(i) if block_given?
  end
  print result
end
```

# действия до блока  
# цикл в пределах диапазона  
# **выполнить блок с параметром**  
# действия до блока

```
calc_numbers_in_range(1,5) { |x| x*x } # 1 + 4 + 9 + 16 + 25 == 55
calc_numbers_in_range(1,5) { |x| x**3 } # 1 + 8 + 27 + 64 + 125 == 225
calc_numbers_in_range(1,5) { |x| x/2.0 } # 0.5 + 1.0 + 1.5 + 2.0 + 2.5 == 7.5
```

# Блок, переданный в open(): транзакция между открытием и закрытием файла

```
File.open('block.rb', 'r') do |file|
  while (line = file.gets) do
    puts line
  end
end
```

# Итераторы и блоки

```
array = ['perl', 'ruby', 'python']  
array.map! { |item| item.capitalize } # ['Perl', 'Ruby', 'Python']  
array.each { |language| printf "Я изучаю %s.\n", language }
```

```
hash = {'Ruby' => 1995, 'Perl' => 1987, 'Python' => 1990}  
hash.each_pair do |language, year|  
  printf "Язык %s создан в %d году.\n", language, year  
end  
hash.each_key { |language| printf "Я изучаю %s.\n", language }
```

```
5.times { |n| puts n } # 0 1 2 3 4  
1.upto(5) { |n| puts n } # 1 2 3 4 5  
5.downto(1) { |n| puts n } # 5 4 3 2 1  
(20..50).step(10) { |n| puts n } # 20 30 40 50  
20.step(50, 10) { |n| print n, " " } # 20 30 40 50
```

```
array.each_with_index { |e, i| printf "%d %s\n", i, e } #== with_index  
"John\nPaul\nGeorge\nRingo\n".each_line { |line| print line }  
%w[John Paul George Ringo].reverse_each { |word| print word }
```

# Proc: ИСПОЛНЯЕМЫЙ ОБЪЕКТ

```
def calc_numbers_in_range(n1, n2, &block)
  result = 0.0
  for i in (n1..n2) do
    result += block.call(i)
  end
  print result, "\n"
end
calc_numbers_in_range(1,5) { |x| x*x }
```

```
square_it = Proc.new { |x| x * x }
puts square_it.call(5)
```

```
def power(exponent)
  Proc.new { |base| base**exponent }
end
square = power(2)
cube = power(3)
b = square.call(4)
c = cube.call(8)
```

# (Почти) синонимы создания Proc-ов:

```
# Proc.new { |x| p x }, proc { |x| p x }, lambda { |x| p x }, ->(x) { p x }
```

# блок → объект Proc (closure)

# вызвать Proc с параметром

# result = 1+4+9+16+25 == 55

# конструктор объекта Proc  
#=> 25

# возвращает объект Proc  
# (anonymous function)

#

#

#=> 16

#=> 512

# Процедурный стиль: скрытый ООП

```
def fibonacci_sequence(n)
  first, second = 0, 1
  fibonacci = 0
  (0..n).each do |number|
    fibonacci = if (number <= 1)
      number
    else
      first, second = second, first + second
    end
    print fibonacci, " "
  end
  print "\n"
end
```

```
@limit = 12                                # атрибут объекта main
fibonacci_sequence(@limit)                 # 0 1 1 2 3 5 8 13 21 34 55 89 144
p self, self.class                         #=> main Object
p self.private_methods.include?(:fibonacci_sequence) #=> true
p self.instance_variables                   #=> [:@limit]
```

# ООП в чистом виде

# даже литералы – это объекты, у которых можно вызывать методы:

```
-42.abs           #=> 42
"Рубин".length    #=> 5
'Ruby'.downcase    #=> "ruby"
[1,2,3,4,5].size   #=> 5
'Ruby'.index('u')  #=> 1
3.141592653.class  #=> Float
```

# операции – это методы (кроме =, .., ..., not, &&, and, ||, or, ::):

```
25./(2.0)         #=> 12.5
2.+(5)            #=> 2+5
```

# можно описать методы наподобие ==, [], []=, также +@ и -@ для унарных + и -

# вызов метода == отправка объекту сообщения с параметрами: 'метод', аргументы

```
2.send "+", 5      #=> 2.+(5)
```

# ООП: это всё объекты!

```
# Регулярные выражения – это объекты класса RegExpr
/^\\d/.class                               #=> Regexp

# Классы – тоже объекты, экземпляры класса Class
EmptyClass = Class.new                     # вызов конструктора класса Class
p EmptyClass.class                         #=> Class
anonymous = Class.new(EmptyClass)         # ссылка на безымянный подкласс
p anonymous, anonymous.superclass         #=> #<Class:0x57f3657c>, EmptyClass

# Анонимные функции – это тоже объекты, экземпляры класса Proc
proc = Proc.new { |x| x + x }
proc.class                                #=> Proc
proc.arity                                #=> 1 == число параметров
proc.parameters                           #=> [[:opt, :x]] == имя параметра

# Методы – это объекты, экземпляры класса Method
method = "string".method(:length)         #
method.class                              #=> Method
method.inspect                            #=> "#<Method: String#length>"
method.call                               #=> 6
```



# ООП: наследование

```
class MultiCopter
  def initialize(no_of_motors)
    @motors = no_of_motors
  end

  def motors; return @motors; end

  def motors=(new_value)
    @motors = new_value
  end
  def rotors(); self.motors(); end
end
```

```
class QuadCopter < MultiCopter
  attr :model
  def initialize(model)
    super(4)
    @model = model
  end
end
```

```
copter = QuadCopter.new('DJI Mavic Air')
puts copter.motors, copter.model
```

```
# "инициализатор" для конструктора
# переменная объекта: всегда private
```

```
# getter motors() для @motors
```

```
# setter для @motors
```

```
#== alias :rotors :motors
```

```
# QuadCopter наследует от MultiCopter
# создаёт getter и setter атрибута
```

```
# вызов инициализатора в надклассе
```

```
# вызов конструктора
# вызов методов
```

# ООП: модули

Модули используются:

- 1) как **интерфейсы** для множественного наследования поведения;
- 2) как средства объединения исходников в пакеты;
- 3) определения иерархии видимости имён (namespace).

```
module Api
  module V1
    class Connector
    end
  end
end

c = Api::V1::Connector.new
```



# ООП: модули

```
module Greetings
  def hello; puts "Hello!"; end
  def bonjour; puts "Bonjour!"; end
  def hola; puts "¡Hola!"; end
  def privet; puts "Привет!"; end
end
```

```
class User
  include Greetings
  def initialize(name, language=:en)
    @name = name; @language = language
  end
  def greet
    case @language
    when :fr then bonjour
    when :es then hola
    when :ru then privet
    else hello
    end
  end
end
print User.new("Pablo", :es).greet
print User.new("Paul").greet
```

```
# набор методов модуля
# образует интерфейс,
# который можно подключить
# к нужному классу
```

```
# модуль подключается
# (mixed in)
```

```
# методы модуля
# становятся
# методами экземпляров
```

```
#=> ¡Hola!
#=> Hello!
```

# ООП: примеси (mixins)

```
# при включении модуля в класс методы модуля становятся методами экземпляров
class Person
  include Comparable                                # подключив модуль и
  attr :name
  def initialize(name)
    @name = name
  end

  def <=>(other)                                     # реализовав 1 метод сравнения,
    @name <=> other.name
  end
# получаем из Comparable операторы <, <=, ==, !=, >=, > и методы between?, clamp
end

ruby = Person.new 'Matsumoto, Yukihiro'
perl = Person.new 'Wall, Larry'
java = Person.new 'Gosling, James'
smalltalk = Person.new 'Kay, Alan'
lua = Person.new 'Ierusalimschy, Roberto'

p ruby > lua                                         # true
p ruby.between? smalltalk, perl                    # true
p [ruby, perl, java, smalltalk, lua].sort           # для sort нужен <=>
```

# Строгая динамическая типизация

```
# Переменная не имеет типа, а её значение – имеет!  
# Она хранит ссылку на объект конкретного класса  
n = 3  
n.class                               #=> Integer  
n.class.superclass                    #=> Numeric  
n.instance_of?(Integer)               #=> true  
n.kind_of?(Numeric) == n.is_a? Numeric #=> true
```

```
n + '0.14'      # String can't be coerced into Integer (TypeError)  
n + '0.14'.to_f # явное преобразование строки во Float
```

```
n = Math::PI  
n.class                               #=> Float  
n.is_a?(Numeric)                      #=> true  
  
n = 'Число Пи'                        #=> String
```

# Duck Typing

```
# "If it looks like a duck,  
# swims like a duck  
# and quacks like a duck,  
# then it must be a duck."
```

```
# Duck Typing: на практике поведение важнее, чем тип (класс)  
list_of_objects.each do |object|  
  object.some_method() if object.respond_to?(:some_method)  
end
```

```
# "Proc" Duck-Typed Objects: отзываются на метод call()  
list_of_proc_objects.each do { |callable| callable.call(arguments) }
```

# Метапрограммирование

```
# Получение информации об объектах при выполнении: introspection / reflection
object.class                # класс объекта
object.class.superclass    # его надкласс
object.class.ancestors     # его классы-предки
object.class.instance_methods(false) # методы объекта (без родительских)
class_name.public_methods(false)

# Все описания выполняются
class Robot                # определение класса
  print self              # self указывает на текущий класс
  def self.produce()      # метод класса: Robot.produce()
    return self.new()
  end
  def big_motor(port=1, command=:on)
  end
end
# Monkey patching: можно внести изменение в описание класса
class Robot                # открыть класс снова
  def middle_motor(port=2, command=:off) # и добавить ещё метод
  end
end
```

# Метапрограммирование

```
# Динамическое определение методов: на примере DSL для коптера Tello
# в цикле вызывается define_method с именем метода и блоком кода
class Tello
  [:up, :down, :left, :right, :forward, :backward].each do |cmd|
    define_method cmd do |cm|
      Tello::Client.return_bool(send("#{cmd.to_s} #{cm}"))
    end
  end
end
# def up(cm); Tello::Client.return_bool(send("up #{cm}")); end
end

# Реакция на обращение к отсутствующему методу
class SomeClass
  def method_missing(method_name, *arguments, &block)
    # если обработка не требуется, можно вызвать super или проигнорировать
    # можно обработать обращение к несуществующему методу
  end
end

# Средства динамического изменения объектов при выполнении
remove_instance_variable(:@var)
```



# Metaprogramming @ Rails

```
# по метаданным из СУБД динамически создаются методы доступа к атрибутам
class Person < ActiveRecord::Base # из базового класса наследуется динамика
end

# по названиям колонок таблицы генерируются методы доступа:
professor = Person.new(
  first_name: 'Владислав', middle_name: 'Юрьевич', family_name: 'Пирогов'
) => #<Person id: nil, first_name: "Владислав", middle_name: "Юрьевич",
family_name: "Пирогов", birthday: nil, pseudonym: nil>
professor.save

# из имени метода генерируется SQL-запрос к БД:
professor = Person.find_by_family_name_and_first_name("Пирогов", "Владислав")
# SELECT "people".* FROM "people" WHERE "people"."family_name" = ? AND
"people"."first_name" = ? LIMIT 1  [["family_name", "Пирогов"],
["first_name", "Владислав"]]

professor.pseudonym = 'Председатель Клуба Программистов'
professor.save
=> #<Person id: 5116, first_name: "Владислав", middle_name: "Юрьевич",
family_name: "Пирогов", birthday: nil, pseudonym: "Председатель Клуба
Программистов">
```

# RubyGems

**RubyGems** — менеджер библиотечных пакетов для **Ruby**

[RubyGems.org](https://rubygems.org) — хранилище дистрибутивов пакетов

[guides.rubygems.org](https://guides.rubygems.org) — документация

**gem** — (1) упакованный дистрибутив стандартизованного библиотечного пакета программ;  
(2) программа управления такими пакетами

```
gem install raspberry_pi_iot
gem list rails
gem update net-http
```

```
require 'net/http'
```

```
uri = URI.parse("http://10.36.0.36:3000")
```

```
http = Net::HTTP.new(uri.host, uri.port)
```

```
get_request = Net::HTTP::Get.new(uri.request_uri)
```

**bundler** — программа управления наборами библиотечных пакетов, требуемые версии которых описаны в конфигурационном файле **Gemfile** в каталоге проекта.

```
bundle install
```

```
# установить все нужные версии
```

# RVM = Ruby Version Manager

```
curl -sSL https://get.rvm.io | bash -s stable
rvm list known
# MRI Rubies
. . .
[ruby-]2.6[.5]
[ruby-]2.7[.0-preview1]
ruby-head
# JRuby
. . .
jruby[-9.2.8.0]
jruby-head
. . .
rvm install 2.6
rvm --default use 2.6.5
rvm list
=* ruby-2.6.5 [ x86_64 ]
# =* - current && default
rvm use ruby-1.8.6
rvm rvm upgrade ruby-2.6 ruby-2.7
```

**RVM** — программа (rvm.io) управления несколькими системами программирования **Ruby** (различных реализаций и разных версий), установленных параллельно, включая интерпретаторы и библиотечные пакеты, с возможностью переключаться между ними и согласованно обновлять.

# Учебные материалы

## Сайты:

- <https://www.ruby-lang.org/ru/documentation/>
- <https://ru.wikibooks.org/wiki/Ruby>
- <https://ru.wikibooks.org/wiki/Ruby/Справочник>
- <https://github.com/rubocop-hq/ruby-style-guide>
- <https://ru.wikipedia.org/wiki/Ruby>
- <http://www.shokhirev.com/mikhail/ruby/ltp/title.html>
- <https://ruby.github.io/TryRuby/>

# Учебные материалы

## Книги на русском:

1. Макгаврен Дж. Head First. **Изучаем Ruby** = Head First Ruby / Пер. с англ. — СПб.: Питер, 2016. — 528 с.
2. Метц С. **Ruby . Объектно-ориентированное проектирование** = Practical Object-Oriented Design in Ruby: An Agile Primer / Пер. с англ. — СПб.: Питер, 2017 — 304 с.
3. Мацумото Ю. **Ruby — руководство пользователя** = Ruby User's Guide / пер. на русский А. Мячков на OpenNET — 2005—2008.
4. Пайн К. **Учись программировать** = Learn to Program / пер. на русский М. Шохирев — 2006-2007 — 96 с.
5. Роганов Е. А., Роганова Н. А. **Программирование на языке Ruby**. Учебное пособие — М.: МГИУ, 2008. — 56 с.
6. Симдянов И. В. **Самоучитель Ruby** — СПб.: БХВ-Петербург, 2020 — 656 с.
7. Стюарт Т. **Теория вычислений для программистов** / Пер. с англ. — М: ДМК Пресс, 2016. — 384 с.
8. Фитцджеральд М. **Изучаем Ruby** = Learning Ruby / пер. с англ. Н. Гаврилова. — 1-е изд. — СПб.: БХВ-Петербург, 2008. — 336 с.
9. Флэнаган Д., Мацумото Ю. **Язык программирования Ruby** = The Ruby Programming Language / пер. с англ. Н. Вильчинский. — 1-е изд. — СПб.: Питер, 2011. — 496 с.
10. Фултон Х. **Программирование на языке Ruby** = The Ruby Way / пер. с англ. А. Слинкин. — 2-е изд. — М.: ДМК Пресс, 2007. — 688 с.
11. Фултон Х., Арко А. **Путь Ruby** = The Ruby Way / пер. с англ. — 3-е изд. — М.: ДМК Пресс, 2015.— 660 с.

*Мною собрано >70 книг по Ruby, в основном на английском. И публикуются и переводятся всё новые...*

# Почему мне *очень* нравится Ruby

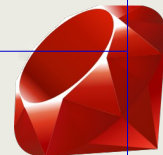
- Это элегантный гармоничный выразительный язык, на котором приятно программировать.
- У него понятный предсказуемый логичный и удобный синтаксис.
- В нём много мощных языковых конструкций и средств, облегчающих разработку.
- Программы получаются короткие, в них легко искать баги, их просто рафакторить.
- Начать писать программы – просто, а потом можно постепенно углубляться в полезные тонкости.
- На нём можно легко и быстро разрабатывать сложные программы.
- Средствами мета-программирования можно эффективно автоматизировать разработку.
- Есть библиотечные пакеты (gems) «на все случаи жизни», которые легко установить, а очень многие уже входят в стандартный набор.

*"Ruby – programmer's best friend."*



# Моё мнение об основных скриптовых языках

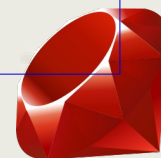
Язык	Плюсы	Минусы
Ruby	Чистое ООП + FP. Просто изучить. Быстро разрабатывать. Мощный.	Некоторые критикуют "monkey-patching". Многие не знают, что <b>Ruby &gt; Rails</b> .
Perl 5	Интересный. Повлиял на другие. Похож на <b>sh</b> .	Криптографический синтаксис. ООП только имитируется. Не такой уж и быстрый.
Raku (Perl 6)	Реализовано много интересных инновационных идей.	Не широко распространён. Пока скорее экспериментальный.
Python	Широко распространён (модный благодаря Google). Много разработок.	Нелогично спроектирован. Странный синтаксис: отступы. Несовместимость версий. Имитация ООП. Глобальные функции.
JavaScript	Реализован в браузерах для DOM. Синтаксис похож на C. Формат JSON удобен.	Прототипирование вместо ООП. Только <b>Node.js</b> на сервере – не лучшее решение.
PHP	<i>"I have absolutely no idea how to write a programming language...."</i> <b>Rasmus Lerdorf</b>	Сумбурно сделанный нелогичный язык, провоцирующий неряшливую разработку программ.
Lua	Хочу изучить. Встраиваемый для микроконтроллеров.	Имитация ООП.



# Анекдот

*Какой из интерпретаторов **доброжелательнее** к программисту?*

Язык	Python	Ruby
Интерактивная оболочка	<code>python</code> или <code>python3</code>	<code>irb</code>
Завершение работы	<pre>\$ python3 Python 3.7.3 (default, Apr 3 2019, 05:39:12) [GCC 8.3.0] on linux Type "help", "copyright", "credits" or "license" for more information. &gt;&gt;&gt; quit Use quit() or Ctrl-D (i.e. EOF) to exit &gt;&gt;&gt; exit Use exit() or Ctrl-D (i.e. EOF) to exit &gt;&gt;&gt; exit()</pre>	<pre>\$ irb 2.6.5 :001 &gt; quit  \$ irb 2.6.5 :001 &gt; exit</pre>





# Вопросы?



# Ruby: *дополнительные примеры*

- Скорость
- Синтаксис: пример программы (с методом)
- Функциональный стиль
- Ассоциативные массивы
- Исключения
- Особенности
- Литералы
- Идиомы
- Идиомы в ООП
- Странности
- Класс с нуля (во время выполнения)
- DSL @ Cucumber: BDD
- Приёмы создания DSL



# Ruby: скорость

Мой тест / время, сек.	Python v3.7.3	Ruby v2.6.5	Perl v5.28.1	PHP v7.0.33
Нахождение простых чисел до 100_000_000 / 50_000_000 методом “решета Эратосфена”	17.281235 8.837612	6.945963 4.038316	42.081436 20.797865	<i>integer overflow</i> 6.602159
Суммирование целых / плавающих чисел от 1 до 1_000_000_000	175.676209 1	21.804367	139.641206	33.930488
Перебор целых чисел от 1 до 1_000_000_000	94.619167	14.636823	82.411183	11.866565
Создание 25_000_000 объектов и вызов 1 метода у каждого из них	23.212306	10.174832	36.872394	8.198939

# Синтаксис: пример

```
#!/usr/bin/ruby                                     # на основе примера из «The Ruby Way»
def convert(temperature, scale)
  case scale
  when "C", "c" then
    f = 1.8*temperature.to_f + 32; c = nil
  when "F", "f"
    c = (5.0/9.0)*(temperature.to_f-32); f = nil
  end
  [c, f]
end

while true do
  print "Введите температуру и шкалу (C или F): "
  STDOUT.flush
  str = STDIN.gets; str.chomp!
  exit if str.nil? || str.empty?
  temp, scale = str.split(" ")
  if temp !~ /-?\d+/ then puts "#{temp} - некорректное число."; next; end
  if scale !~ /[cCfF]/ then p "Укажите C или F через пробел от числа!"; next; end
  c, f = convert(temp, scale)
  puts "#{temp}°F - это #{sprintf('%5.1f', c)} градусов C" if f.nil?
  print "#{temp}°C == #{f}°F \n" if c.nil?
end
```

# Стиль: функциональный

```
# Флэнаган Д., Мацумото Ю. "Язык программирования Ruby"  
# гл.6.8: Функциональное программирование  
# вычислить среднее и стандартное отклонение по массиву чисел  
mean = array.inject { |x, y| x + y } / array.size  
sumOfSquares = array.map{ |x| (x - mean)**2 }.inject{ |x, y| x + y }  
standardDeviation = Math.sqrt(sumOfSquares / (array.size-1))  
  
# 6.8.2 Составление функций = Composing Functions  
# 6.8.3 Частично применяемые функции = Partially Applying Functions  
# 6.8.4 Функции, обладающие мемоизацией = Memoizing Functions  
# 6.8.5 Классы Symbol, Method и Proc = Symbols, Methods, and Procs
```

# Ассоциативные массивы

```
# Ассоциативные массивы == хэши == словари
# ключи - любые, например, строки:
hash1 = {'Ruby' => 1995, 'Perl' => 1987, 'Python' => 1990}
# удобно, если ключи - символы:
hash2 = {:Ruby => 1995, :Perl => 1987, :Python => 1990}
# сокращённый вариант записи ключей-символов, в т. ч. для параметров:
hash2 = {Ruby: 1995, Perl: 1987, Python: 1990}
```

```
# экзотические ключи и значения
```

```
hash = Hash.new
```

```
hash[1995] = 'Ruby'
```

```
hash['п'] = Math::PI
```

```
hash[Math::E] = 'e'
```

```
hash[:Ruby] = ['Matsumoto, Yukihiro', :Japan]
```

```
hash[{'Perl'=>'1987'}] = 'Larry Wall'
```

```
hash[['Python',1991]] = 'Guido van Rossum'
```

```
hash[true] = 0
```

```
hash[Array] = []
```

```
hash[/[aeiou]/] = :vowels
```

```
hash[Person.new('Matz')] = ''
```

```
# целое
```

```
# строка в Unicode
```

```
# дробное
```

```
# Symbol
```

```
# Hash
```

```
# Array
```

```
# логическая величина
```

```
# класс
```

```
# Regexp
```

```
# экземпляр класса
```

# Исключения

```
# Возбудить исключение  
raise SomeException, 'message'
```

```
# синоним: fail
```

```
# Обработать исключения  
begin  
  # опасное действие  
  rescue Exception1, Exception2 => e  
    # при определённых ошибках  
    retry if condition  
  rescue  
    # при остальных ошибках  
  else  
    # без ошибок  
  ensure  
    # несмотря на ошибки  
end
```

```
# повтор
```

```
# неявный begin в теле метода  
def method  
  # основная обработка  
  rescue  
    # обработка исключений  
end
```

# Ruby: особенности

```
# Истина и ложь
true          # истина - всё, что не false и не nil (не как в C и Perl)

# Переменная хранит ссылку на значение объекта
s1 = "Ruby"    #=> значение "Ruby" связывается с именем s1
s2 = s1        #=> "Ruby" - переменная s2 получает ссылку на ту же строку
s1[0] = 'r'    #=> "ruby" - изменяется строка, на которую ссылаются s1 и s2
puts s2        #=> "ruby" - т. к. s2 ссылается на изменённое значение объекта
s3 = s1.dup    # в новый объект s3 скопировано значение из s1

variable = 0    # переменная не имеет типа, а значение имеет тип Integer
1 + 'строка'   # 1:in `+': String can't be coerced into Integer (TypeError)
```



# Ruby: литералы-скаляры

```
# Пустое значение: nil
```

```
# Строки:
```

```
'non-interpolated with \' and \\'
```

```
"\t #{'interpolated'} \n"
```

```
"con" "cat" "en" "at" "ed"
```

```
"Multi-line  
strings"
```

```
here_doc = <<DELIMITER
```

```
This would contain
```

```
Several lines of specially formatted text.
```

```
DELIMITER
```

```
# Числа:
```

```
Binary = [0b11110000, 0B1111_0000]
```

```
octal = [0777, 0o777, 0O777]
```

```
decimal = [12345, 0d12345, 0D12345]
```

```
integer = [1234567, 1_234_567]
```

```
float = 1_234.56789
```

```
scientific = [-1.234E1, 1.234e-1]
```

```
hexadecimal = [0xABCD, 0xAB_CD, 0Xabcd]
```

# Ruby: литералы

```
# Логические (Boolean): true; false
# Символы (Symbol): :name; : "weird string"

# Массивы:
['each', 'word', 'is', 'element']
%w(each word is element)

# Хэши:
{:key => 'value'}
{key: 'value'}

# Регулярные выражения:
/hello/
%r{hello/path}
%r{hello#{'interpolated'}}
```

```
# Блоки:
{ block; literal }
```

# Ruby: ИДИОМЫ (*rubyisms*)

```
variable ||= default # variable = default unless variable
boolean = true if boolean.nil? # для boolean
something &&= something.change() # изменить значение, если существует

# Вызов метода для каждого элемента массива
['Perl', 'Python', 'Ruby'].map(&:upcase) #=> ["PERL", "PYTHON", "RUBY"]
(1..5).map(&:to_f) #== (1..5).map { |n| n.to_f }
[1, 2, 3].select(&:even?) #== [1, 2, 3].select { |n| n.even? }

# Неявное преобразование (splat) массива в список параметров блока:
[[1,2], [3,4], [5,6]].each { |one, two| print "#{one}, #{two}} " }

# предикаты
[].any? #=> false
puts "Одно из многих!" if [v1,v2,v3].include?(value)

SOME_CONSTANT = "value" unless defined?(SOME_CONSTANT)
p x #== puts x.inspect
```

# Ruby: ИДИОМЫ В ООП

# Альтернативное объявление

```
class Person
  class << self
    def m
      "Class method"
    end
  end
end
```

# Eigenclass

```
object = Array.new
class << object
  def m
    "Singleton method"
  end
end
```

# Обычное объявление метода класса

```
class Person
  def self.m
    "Class method"
  end
end
```

# Обычное объявление singleton-метода

```
object = Array.new
def object.m
  "Singleton method"
end
```

# Ruby: странности

[#https://www.rubyguides.com/2017/02/weird-ruby/](https://www.rubyguides.com/2017/02/weird-ruby/)

```
input = <<-IN.split
```

Эти строки будут  
преобразованы в массив строк.  
IN

```
"abc"::size
```

```
[1,2,3,4,5]::length
```

```
a = []
```

```
a << 1 << 2 << 3
```

```
#=> 3
```

```
#=> 5
```

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

# Ruby: класс с нуля

```
c = Class.new          # Анонимный (безымянный) класс, подкласс класса Object
p c
['model', 'motor', 'height', 'battery'].each do |variable|
  c.class_eval "@#{variable} = nil"
end
p c.instance_variables, c.instance_variable_defined?('@motor')

init = -> (model) { @model = model; @motor = :off; @height = 0; @battery = rand(100) }
c.send :define_method, :initialize, init
p c.public_methods(false), c.private_methods(false)

method = Proc.new do |height|
  if @battery > 10
    @motor = :on
    @height = height
  end
  [@model, @battery, @motor, @height]
end
c.send(:define_method, :takeoff, method)
p c.public_method_defined?(:takeoff)

Copter = c          # присвоить имя
tello = Copter.new 'Tello EDU'
p tello.takeoff(12)
```

```
class Copter
  def initialize(model)
    @model = model; @motor = :off
    @height = 0; @battery = rand(100)
  end
  def takeoff
    if @battery > 10
      @motor = :on
      @height = height
    end
    [@model, @battery, @motor, @height]
  end
end
```

# DSL @ Cucumber: BDD

```
# BDD = Behaviour-Driven Development
# Feature: Добавление товара
#   As покупатель
#   I want добавить товар в корзину
#   So that этот товар был добавлен к моему заказу
#   Scenario: Пользователь добавляет товар
#     Given я на странице со списком товаров
#     When я нажимаю на кнопку добавления
#     Then я должен увидеть подтверждение, что товар добавлен
```

```
Given(/^I am on the products page$/) do
  visit product_list_path
end
```

```
When(/^I click the add button$/) do
  click_on "add-item"
end
```

```
Then(/^I should see the confirmation message$/) do
  expect(page).to have_content("Item added")
end
```

# Приёмы создания DSL

```
# Блок для инициализации объекта
class Recipe
  attr :ingredients

  def initialize(&block)
    @ingredients = []
    instance_eval &block
  end

  def ingredient(name, quantity)
    @ingredients << [name, quantity]
  end
end

r = Recipe.new do
  ingredient "Eggs", 3
  ingredient "Cheese", "100g"
end
```