Краткий справочник
по языку программирования Ruby

О языке:

- Название: **Ruby** (англ. ruby ['ru:bɪ] [ру́би]— рубин). Название навеяно языком Perl, многие особенности синтаксиса и семантики из которого заимствованы в Ruby: англ. pearl — «жемчужина», ruby — «рубин».
- Когда создан: 1995 год.
- Создатель: японский программист **Юкихиро «Matz» Мацумо́то** (松本行弘 = まつもとゆきひろ).
- Характеристика: интерпретируемый, динамический, сценарный, высокоуровневый полностью объектно-ориентированный (мультипарадигменный) язык программирования.
- Лицензия: Ruby License. Кросс-платформенная реализация интерпретатора языка является полностью свободной.
- Испытал влияние: Perl 5, Smalltalk, Eiffel, Ada, Lisp, Python, Dylan, CLU, C++, Lua.
- Повлиял на: Groovy, Amber, CoffeeScript, Perl 6, Elixir, Crystal.
- Принципы:
  - Ориентация на разработчика, на то, как ему лучше решать программистские задачи.
  - POLS = Principle Of Least Surprise = «Правило наименьшего удивления» — программные конструкции должны быть логичными, интуитивно понятными и ожидаемыми.
  - TIMTOWTDI [Tim Toady] = There Is More Than One Way To Do It ~ «Есть не один способ это сделать» — один и тот же результат можно получить несколькими различными способами.
  - Потребность программиста создавать полезные и красивые программы как причина программирования.

Формат программы:

- Тексты программ сохраняются в файлах с суффиксом (расширением) **`.rb`**
- Команды (операторы) обычно записываются по одной в строке.
- Если нужно записать несколько команд в строке, они разделяются точкой-с-запятой (;).
- Все команды последовательно выполняются интерпретатором, даже объявления.
- Имена обычно состоят из латинских букв, цифр и знаков подчёркивания.
- Имена констант и классов начинаются с заглавной буквы, обычно пишутся без подчёркиваний.
- Имена переменных и методов начинаются со строчной буквы, обычно разделяются подчёркиваниями.
- Пробелы можно не вставлять, если это не меняет смысл выражения, но лучше вставлять их для удобства чтения человеком.
- Круглые скобки при вызове метода можно не писать.
- Для выделения вложенности (программных конструкций) обычно делают отступ в 2 пробела.

Запуск программы:

Программу на **Ruby** можно выполнить в терминальном окне, для чего его нужно запустить интерпретатор командой:

```
ruby /путь/к/файлу/с/исходным/текстом/программы.rb
```

В ОС Unix / Linux / *BSD / MacOS программу на **Ruby** можно выполнить по имени с автоматическим запуском интерпретатора:

```
/путь/к/файлу/с/исходным/текстом/программы.rb
```

если в 1-й строке исходного текста указан `#!/путь/к/интерпретатору/ruby` и файл с программой помечен как «исполняемый» командой

```
chmod a+x /путь/к/файлу/с/исходным/текстом/программы.rb
```

При этом у файла с исходной программой на **Ruby** не обязательно должен быть суффикс **.rb**, он может быть вовсе без «расширения».

В ОС MS Windows можно настроить автоматический запуск интерпретатора при двойном нажатии мышью на файл с исходным текстом программы на **Ruby**. Для этого нужно ассоциировать расширение файла "**.rb**" с интерпретатором **ruby**. 1-й способ это сделать — выбрать в проводнике файлов "Инструменты | Параметры папки" и настроить ассоциацию на вкладке "Типы файлов". 2-й способ — ввести в командной строке команды, которые создадут эту ассоциацию:

```
assoc .rb=RubyScript
ftype RubyScript=ruby.exe %1 %*
```

Интерактивный интерпретатор:

Любые команды языка **Ruby** можно выполнить в диалоговом интерпретаторе, для чего его нужно запустить в терминальном окне командой:

```
irb
```

После ввода команда сразу выполняется и выдаётся результат её работы или сообщение об ошибке.

Константы, литералы, переменные

Описание	Обозначение	Пример	Пояснения
Литералы			
целое число (в десятичной системе счисления)	*цифры* **+***цифры*	1234567 +1234567	положительное число
	-*цифры*	-1234567	отрицательное число
	десятичные_цифры	1_234_567	подчёркивание игнорируется
целое число (в восьмеричной системе счисления)	**0***восьмеричные_цифры*	04553	цифры: 01234567
целое число (в шестнадцатеричной системе счисления)	**0x***шестнадцатеричные_цифры*	0x96b	цифры: 0123456789ABCDEF
целое число (в двоичной системе счисления)	**0b***двоичные_цифры*	0b1001_0110_1011	цифры: 01
дробное число (в десятичной системе счисления)	*цифры***.***цифры*	3.141592653 0.1234567	точка отделяет дробную часть
дробное число в «научной» записи (экспоненциальной нотации)	*цифры***E+***цифры* *цифры***e-***цифры*	123e4 123e+4 +123E4 123e-4	123000.0 123000.0 123000.0 0.0123
строка	**'***строка с кавычками (')* **'** **"***строка с апострофом (')***"** **%Q{***строка***}**	'строка в апострофах' "строка в кавычках" '' '@' %Q{The Ruby language}	без подстановки #{v} с подстановкой #{v} пустая строка с длиной == 0 строка из 1 символа "The Ruby language"
символ	**:***символ*	sensor = :DS18B20	:DS18B20 != 'DS18B20'
логические (булевы) значения	**true** # истина **false** # ложь	flag = true positive = (x > 0)	
диапазоны	**(***число***..***число***)** **(***число***...***число***)**	(1..5) (1...5)	1,2,3,4,5 1,2,3,4
Константы			
присваивание значения	*Константа***=***значение*	PI = 3.141592653	имя — с заглавной буквы
использование значения	*Константа*	print(PI)	

логические «константы»	**true** # истина **false** # ложь	fairy_tale = false documentary = true	
неопределённое значение	**nil**		Воспринимается как false
Переменные			
присваивание значения	*переменная***=***значение*	pi = PI	имя — с маленькой буквы
использование значения	*переменная*	print(pi)	имя: латинские буквы, цифры и подчёркивания
Методы (подпрограммы: процедуры и функции)			
описание процедуры	**def** *имя_процедуры*(*параметры*) *команда 1* . . . *команда N* **end**	def exclame(word) print(word + "!") end	Возвращает значение последнего выражения, вычисленного в процедуре
вызов процедуры	*имя_процедуры*(*аргументы*)	exclame("Ruby")	"Ruby!"
описание функции	**def** *имя_функции*(*параметры*) *команда 1* . . . *команда N* **return** *значение* **end**	def s_to_a(string) return s.split("") end	Возвращает значение, указанное в команде return и завершает выполнение
вызов функции	*переменная* = *имя_функции*(*аргументы*)	array = s_to_a("Ruby")	["R", "u", "b", "y"]
функция с логическим результатом	**def** *имя_функции***?**(*параметры*) . . . **return** *логическое значение* **end**	def is_empty?(string) return string.strip == "" end	
вызов функции с логическим результатом	*переменная = имя_функции***?**(*аргументы*)	print "No" if is_empty?(line)	

Операции

Описание	Обозначение	Пример	Пояснения
Присваивание			
присваивание переменной	**=**	x = 1	переменная

		x = 'Ruby'	
параллельное присваивание		x, y, z = 100, 200, 500	x = 100; y = 200; z = 500
цепочка присваиваний		a = b = c = 3	c = 3; b = c; a = b
			результат – присвоенное значение
Арифметические операции			
сложить, вычислить сумму	**+**	123.4 + 56.7 + 89	269.1
вычесть, вычислить разность	**-**	n - 1	n, уменьшенная на 1
умножить, найти произведение	*****	2 * PI * r	длина окружности
разделить, вычислить частное от деления	/	42 / 5 42 / 5.0 42.0 / 5	8 8.4 8.4
вычислить остаток от деления	**%**	42 % 5	2
возвести в степень	**	2**5	32
изменение значения переменной	**+=** **-=** ***=** **/=** **%=** ****=**	n += 1 n -= 2 n *= 3 n /= 4 n %=5 n **= 6	увеличить на 1 уменьшить на 2 увеличить в 3 раза уменьшить в 4 раза остаток от n/5 возвести n в 6-ю степень
Операции сравнения:			
равно	**==**	2*2 == 4	true
не равно	**!=**	2*2 != 4	false
больше	**>**	2*2 > 4	false
больше или равно	**>=**	2*2 >= 4	true
меньше	**<**	2*2 < 4	false
меньше или равно	**<=**	2*2 <= 4	true
Логические операции:			
И	**&&** **and**	true && true true and false	true false

ИЛИ	|| **or**	true || true true or false	true true
НЕ	**!** **not**	! true not false	false true
изменение с присваиванием	**&&=** **||=**	variable &&= value variable ||= value	
Условная операция	*условие* **?** *команда1* **:** *команда2*	z = (y != 0) ? x/y : 0	Если y не равен 0, то z = x/y, иначе z = 0
Операции со строками:			
повторение строки	*строка* ***** *число*	print('Ура! ' * 3)	Ура! Ура! Ура!
сцепление строк	*строка* **+** *строка*	n=1; 'Значение N=' + n.to_s	'Значение N=1'
дополнение	*строка* **<<** *строка*	s = "Ru"; s << "by"	s == "Ruby"
подстановка в строку значения выражения	**" #{***выражение***} "**	v=2; print "#{v}x#{v}=#{v*v}"	"2x2=4"
вычленение символа	*строка***[***индекс***]**	word[0] word[-1]	1-й символ строки последний символ строки
выделение подстроки	*строка***[***индекс,длина***]** *строка***[***диапазон***]**	word = 'Ruby' word[0, 1] word[1..2] 'Ruby'[-2, 2] 'Ruby'[-4..-3]	# нумерация с 0 'R' 'ub' 'by' 'Ru'
замена символа	*строка***[***индекс***]=***символ*	word[0] = 'r' word[-1]='ies'	'ruby' 'rubies'
замена подстроки	*строка***[***строка***]=***новая_строка*	word['R'] = 'ies'	'Rubies'
длина строки	*строка***.size** *строка***.length**	'Ruby'.size 'Ruby'.length	4 4
Преобразование в массив по разделителю	*массив* = *строка***.split(***разделитель***)**	'1;2;3'.split(';') 'Ruby'.split ''	[1, 2, 3] ["R", "u", "b", "y"]
Операции побитовые:			
побитовое И	**&**	bits & mask	
побитовое ИЛИ	**|**	bits | mask	
побитовое исключительное ИЛИ	**^**	bits ^ mask	

побитовое дополнение	**~**	bits ~ mask	
побитовый сдвиг влево	**<<**	bits << mask	
побитовый сдвиг вправо	**>>**	bits >> mask	
побитовое изменение с присваиванием	**&=** **|=** **^=** **<<=** **>>=**	bits &= mask bits |= mask bits ^= mask bits <<= mask bits >>= mask	
Преобразование значений:			
преобразовать в строку	**.to_s**	42.to_s (25.5).to_s :symbol.to_s	"42" "25.5" "symbol"
преобразовать в целое	**.to_i**	(25.5).to_i 'один'.to_i	25 0
преобразовать в дробное	**.to_f**	25.to_f "1".to_f	25.0 1.0
преобразовать в тип «символ»	**.to_sym**	"symbol".to_sym (25.5).to_s.to_sym	:symbol :"25.5"
преобразовать в массив	**.to_a**	(1..12).to_a	[1,2,3,4,5,6,7,8,9,10,11,12]

Управляющие конструкции

Описание	Обозначение	Пример	Пояснения
Комментарии			

однострочный	**#** *текст*	# Программа: led.rb
		led = LED.new # светодиод
			Весь текст от # и до конца строки

многострочный	**=begin**
	комментарии
	комментарии
	комментарии
	=end
		=begin
		  Программа: led.rb
		  Автор: ШМВ
		  Описание: «мигалка»
		=end
			=begin
			  Многострочные комментарии
			  используются для
			  документирования
			  программы
			=end

специальный (shebang): имя программы-интерпретатора для выполнения	**#!**/*путь/к/интерпретатору/ruby*	#!/usr/bin/ruby	Для запуска исполняемых программ на Ruby на

			выполнение по имени файла
Последовательность	*команда1* *команда2*	print "Привет, " print "Ruby!\n"	Привет, Ruby!
	команда1; *команда2*	s="рубины!"; puts s	; заменяет перевод строки
Блоки			
Блок без параметров	{ *команда1; команда2* } **do** *команда1* *команда2* **end**	3.times { print("Ruby! ") }	Обычно между { } записываются команды однострочного блока, а между do и end — команды многострочного блока.
Блок с параметром	{ |*параметр*| *команда1; команда2* } **do** |*параметр*| *команда1* *команда2* **end**	sorted = array.sort do |a, b| a <=> b end	Между | | записываются параметры (переменные), значения которых нужно обработать в блоке
Ветвление			

Ветвление по условию истинности	**if** *условие1*
	команды1
	eslif *условие2*
	команды2
	else
	команды3
	end
		t = sensor.temperature()
		if (t > low_t) && (t < high_t)
		fan.switch_off()
		elsif (t > high_t)
		fan.switch_on()
		elsif (t > max_t)
		system('poweroff')
		else
		log.write(t)
		end
			В одном if может быть сколько угодно elsif и не больше одного else.

	if *условие1* **then** *команды1* **else** *команды2* **end**	if (t > max_t) then system('poweroff') end	then писать не обязательно
Ветвление по условию ложности	**unless** *условие1* *команды1* **else** *команды2* **end**	unless (command=='stop') send(command) else stop_sending() end	
	unless *условие1* **then** *команды1*	unless y == 0 then	

	else *команды2* **end**	z = x / y end	
Ветвление по значению выражения	**case** *выражение* **when** *значение1* *команды1* **when** *значение2* *команды2* **else** *команды3* **end**	t = sensor.temperature() case t when t < high_t log.write(t) when t > high_t fan.switch_on() when t > max_t system('poweroff') end	
Повторение			
Цикл с условием продолжения (повторять, пока условие истинно)	**while** *условие* **do** *команды* **end**	n = 0 while n < 50 do n += 10 end	
	begin *команды* **end while** *условие*	n = 0 begin n += 10 end while n < 50	
Цикл цикл с условием окончания (повторять, пока условие ложно)	**until** *условие* **do** *команды* **end**	n = 0 until n == 50 do n += 10 end	
	begin *команды* **end until** *условие*	n = 0 begin n += 10 end until n == 50	
Бесконечный цикл с оператором прерывания повторения	**loop do** *команды* **break if** *условие* **end**	i = 0 loop do i += 1 break if i == 10 end	
Цикл перебора элементов списка	**for** *переменная* **in** *список* **do** *команды* **end**	for i in 0..9 { print(i) } # или for j in 0..9 do	0123456789

		print(j) end	
Итераторы	*объект.**итератор** блок*		
выполнить заданное количество раз	*число.**times** блок*	n.times { led.on; led.off }	Выполнится n раз
выполнять с увеличением от начального до конечного значения	*число.**upto** число блок*	0.upto n do |i| print(i) end	Выполнится с увеличением i от 0 до n
выполнять с уменьшением от начального до конечного значения	*число.**downto** число блок*	n.downto 0 do |i| print(i) end	Выполнится с уменьшением i от n до 0
	*объект.**each { |элемент| команды }***	(1..5).each { print('Ruby') } [1,2,3].each { |e| print(e) }	Выполнится 5 *раз* Печать *каждый* элемент массива
Модификаторы команд:			
Выполнение команды при условии	*команда **if** условие*	system('halt') if (t > max_t)	
Выполнение команды не при условии	*команда **unless** условие*	print('OK') unless t > max_t	
Выполнение команды пока условие	*команда **while** условие*	work while dow != 'суббота'	
Выполнение команды до условия	*команда **until** условие*	work until dow == 'суббота'	

Агрегаты:

Описание	Обозначение	Пример	Пояснения
Массивы (списки)			
Литерал	*[элемент0, элемент1, элемент2]*	empty = [] list = [0, 'один', [2], :three]	Пустой массив Массив со значениями
Создание массива	*массив=**Array.new(**размер,значение**)***	empty = Array.new array = Array.new(3) a = Array.new(3, rand())	[] [nil, nil, nil] Массив с 3-мя случайными
Добавление элемента в конец массива	*массив **<<** значение*	a = []; a << 3.14; a << 2.718	[3.14, 2.718]
Изменение значения одномерного массива	*массив**[**индекс**]** **=** значение*	list[4] = 'IV' list[100] = 99+1	

Выборка значения одномерного массива	*значение = массив*[*индекс*]	one = list[1] print list[0]	
Создание многомерного массива (массива массивов)	*куб* = [*массив1, массив2, массив3*]	cube = [[11,12], [21,22], [31,33]]	
Изменение значения многомерного массива	*массив*[*i1*][*i2*] = *значение*	matrix[i][j] = rand()	Значение элемента в строке i и столбце j двумерной матрицы
Выборка значения многомерного массива	*значение=массив*[*i1*][*i2*]	cell = matrix[i][j]	cube[x][y][x] для трёхмерного массива и так далее
Перебор элементов массива	*массив*.**each do** |*элемент*| *команды* **end**	[1, 2, 3].each do |e| print "#{e} " end	1 2 3
Перебор индексов массива	*массив*.**each_index do** |*индекс*| *команды* **end**	[1, 2, 3].each_index do |i| print "a[#{i}]=#{a[i]}; " end	a[0]=1; a[1]=2; a[2]=3;
Перебор элементов массива с индексами	*массив*.**each_with_index do** | *элемент,индекс*| *команды* **end**	[4, 5, 6].each_with_index do |e, i| print "arr[#{i}]=#{e}; " end	a[0]=4; a[1]=5; a[2]=6;
Преобразование элементов массива в строку с разделителем	*строка = массив*.**join**(*разделитель*)	s = [1,2,3].join(';')	'1;2;3'
Хэши (ассоциативные массивы, словари)			
Литерал	{*ключ1*=>*значение1,* *ключ2*=>*значение2*} {*ключ1*:*значение1,* *ключ2*:*значение2*}	hash = {'Ruby' => 1995, :author => 'Matz', 0.95 => '1995-12-21'}	точка отделяет дробную часть
Создание объекта класса «Хэш»	*хэш*=**Hash.new**	hash = Hash.new	hash = {}
Изменение значения: ассоциация значения с ключом	*хэш*[*ключ*] = *значение*	hash[1.0] = '1996-12-25'	{1.0 => '1996-12-25' }
Выборка значения	*значение = хэш*[*ключ*]	released = hash[1.0]	Значение, ассоциированное с ключом или nil, если указанного ключа не существует

Удаление пары по ключу	*хэш*.**delete** *ключ*	hash.delete 'Python'	

Модули, классы, методы

Описание	Обозначение	Пример	Пояснения
Классы			
Описание класса	**class** *ИмяКласса* *атрибуты* *методы* **end**	class Sensor attr :value def read return @value end end	Класс описывает для объектов набор характеристик (свойства) и набор действий (методы).
Наследование	**class** *ПодКласс* **<** *НадКласс* **end**	class DHT11 < Sensor end	Подкласс наследует все описания надкласса и добавляет свои.
Атрибуты объектов			
атрибут (свойство) объекта	*@имя_атрибута*	class Sensor def initialize @value = 0.0 end end	У всех объектов одного класса один и тот же набор атрибутов, но у каждого объекта своё состояние (набор значений) атрибутов.
объявления атрибута с возможностью только чтения значения	**attr_reader :***имя_атрибута* **attr :***имя_атрибута* # устаревшее	class C attr_reader :ro_value end	class C def ro_value @ro_value end end
объявления атрибута с возможностью только изменения значения	**attr_writer :***имя_атрибута*	class C attr_writer :wo_value end	class C def wo_value=(new_value) @ wo_value = new_value end end
объявления атрибута с возможностью чтения и изменения значения	**attr_accessor :***имя_атрибута*	class C attr_accessor :rw_value end	class C attr_reader :rw_value attr_writer :rw_value end

Атрибуты класса			
атрибут (свойство) класса	*@@имя_атрибута_класса*	class ESPeasy @@port = 80 end	Одно значение для всех объектов класса
Методы объекта			

Метод объектов	**def** *имя_метода*(*параметры*)
	  команды
	end
		class DHT11
		  def read()
		  @value = read_dht11()
		  end
		end
			Методы — это органы управления объектом.

Инициализатор для конструктора объекта	**def initialize**(*параметры*)
	end
		def initialize(pin=18)
		  @pin = pin
		  @value = read()
		end
			initialize() вызывается при создании объекта, ему передаются параметры Класс.new()

Вызов одноимённого метода надкласса	**super**	def method() super end	
Создание объекта класса	*объект* **=** *Класс*.**new**(*аргументы, для, initialize*)	sensor = Sensor.new(17)	
Вызов метода объекта	*объект*.***метод***(*аргументы*)	t = sensor.read()	() писать не обязательно
Методы класса			

Описание метода класса	**def** ***имя_класса.****имя_метода*
	(*параметры*)
	  команды
	end
		class Tickets
		  def Tickets.new_roll(n)
		  Tickets.new(0, n-1)
		  end
		end	

Вызов метода класса	***класс.****метод*(*параметры*)	roll=Tickets.new_roll(10000)	
Модули			
Описание модуля	**module** *ИмяМодуля* **end**	module TemperatureSensor end	
Наследование	**module** *ПодМодуль* **<** *НадМодуль* **end**	module DigitalSensor < Sensor end	

Ввод-вывод: файлы, каталоги

Описание	Обозначение	Пример	Пояснения
Файлы			
Вывод в STDOUT (на экран)	**print** *список, выражений*	print "Перевод строки\n"	
Вывод в STDOUT (на экран) с новой строки	**puts** *список, выражений*	puts 'Ошибка чтения!'	
Отладочный вывод в STDOUT (на экран)	**p** *список, выражений*	p object	
Форматированный вывод в STDOUT	**printf "***формат вывода***"**, *список, выражений*	printf "T=%5.2f°C \n", temp	
Вывод в STDERR (на экран)	**STDERR.***команда сообщение*	STDERR.print(error_msg)	
Чтение строки из STDIN (с клавиатуры)	*переменная* = **gets**	mode = gets()	

Чтение всех строк из STDIN	**while** (*переменная* = **STDIN.gets**)
	do
	  команды
	end
		while (line = STDIN.gets) do
		  puts line
		end
			Читает по одной строке до конца файла

Чтение («Read») из файла	**File.open(***имя_файла*, **'r').each do** |*переменная*|
	  команды
	end
		File.open('in.txt', 'r').each do |s|
		  puts s
		end
			Прочитает из существующего файла

Запись («Write») в файл	**File.open(***имя_файла*, **'w') do |f|**
	  f.print *выражение*
	  f.puts *выражение*
	  f.write *выражение*
	end
		File.open("out.txt", "w") do |f|
		  5.times do
		  f.write rand(100)
		  end
		end
			Создаст файл и запишет в него с начала

Дозапись («Append») в файл	**File.open(***имя_файла*, **'a') do |f|**
	  f.print *выражение*
	  f.puts *выражение*
	  f.write *выражение*
	end
		File.open("add.txt", "a") do |f|
		  5.times do
		  f.printf("%d\n", rand(100))
		  end
		end
			Создаст файл или дозапишет строки в конец существующего файла

Проверка существования файла	*логическое_выражение* = **File.exist?(***имя_файла***)**	File.exist?("text.txt").	true или false

Имя принадлежит файлу?	**File.file?**(*имя_файла*)	File.file?(file)	→ true или false
Имя принадлежит каталогу?	**File.directory?**(*имя_файла*)	File.directory?(file_name)	→ true или false
Только имя каталога (путь), где находится файл	*каталог* = **File.dirname** (*полное_имя_файла*)	File.dirname("/home/pi/x.rb")	"/home/pi"
Основное имя файла без пути (и суффикса)	*файл* = **File.basename** (*полное_имя_файла*)	File.basename("/tmp/x.rb") File.basename("/tmp/x .rb", ".rb") File.basename("/tmp/x.rb", ".*")	"x.rb" "x" "x"
Суффикс имени файла	*суффикс* = **File.extname** (*полное_имя_файла*)	File.extname("/home/pi/x.rb")	".rb"
Полное имя (путь) файла с правильными разделителями каталогов	*полное_имя_файла* = **File.join** (*путь, к, файлу*)	File.join("home", "pi", "x.rb")	"/home/pi/x.rb" в Linux "\home\pi\x.rb" в MS Windows
Размер файла в байтах	*размер* = **File.size**(*имя_файла*)	File.size(file_name)	Размер файла

Учебник по Ruby для начинающих:
`/home/pi/Documents/books/Learn_To_Program-Ch.Pine-ru.pdf`

<https://ru.wikipedia.org/wiki/Ruby>
<http://www.ruby-lang.org/ru/>
<http://ruby-doc.org/>