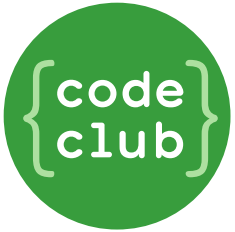
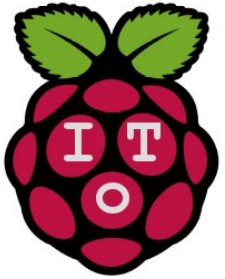


Internet of Things



Программирование на **Ruby**

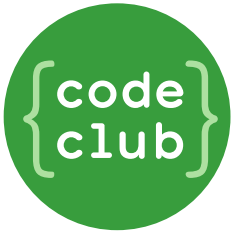
Шадринск
2018-2019

М. В. Шохирев

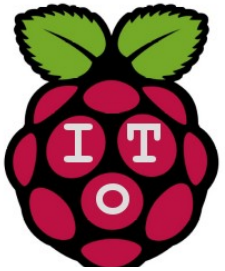
Язык Ruby

Название: **Ruby** (англ. ruby ['ru:bɪ] [ру́би]— рубин).
Создан: в 1995 году. Начало распространения вне Японии – с 1998 г.
Создатель: японец **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: реализация интерпретатора языка (для всех аппаратных платформ) является полностью свободной.
Похож на языки: Perl, Eiffel, Smalltalk, Python, Lisp, Dylan, Clu.
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.



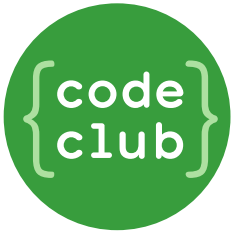
Программы на Ruby



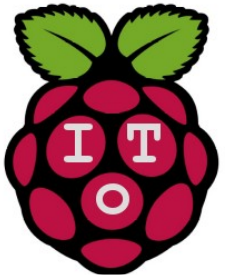
- Тексты программ сохраняются в файлах с суффиксом (расширением) **`.rb`**
- Команды (операторы) обычно записываются по одной в строке.
- Если нужно записать несколько команд в строке, они разделяются точкой-с-запятой (`;`).
- Все команды последовательно выполняются интерпретатором, даже объявления.
- Имена обычно состоят из латинских букв, цифр и знаков подчёркивания: **`red_led_on_pi_1`**.
- Имена должны начинаться с буквы: **`sensor25`**.
- Имена констант и классов начинаются с заглавной буквы, обычно пишутся без подчёркиваний, каждое слово с заглавной буквы: **`TemperatureSensor`**.
- Имена переменных и методов начинаются со строчной буквы, обычно разделяются подчёркиваниями: **`state = led.is_on?`**
- Переменные-атрибуты («внутри») объектов начинаются с символа (`@`): **`@pin = pin`**
- 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: **`led.on; led.off()`**
- Между именем класса или объекта и его методом ставится точка (`.`): **`r=LED.new(18);r.on`**
- Для выделения вложенности (программных конструкций) обычно делают отступ в 2 пробела.

Числа

```
1234567890                # целое число (в десятичной системе счисления)
+1234567890               # то же, положительное число
1_234_567_890             # то же, с разделителем разрядов «_»
-1234567890               # отрицательное число

0b1001_0110_1011          # целое число (в двоичной системе счисления)
076543210                 # целое число (в восьмеричной системе счисления)
0xFEDCBA9876543210        # целое число (в шестнадцатеричной системе счисления)

3.141592653               # дробное число

-123e4                    # -123000.0 — число в «научной» записи
+123E4                    # 123000.0    (в экспоненциальной нотации)
123E+4                    # 123000.0  = 123 * 10**4
123e-4                    # 0.0123    = 123 * 10**-4

((1+2)*(3-4/5.0)%6)**7    # арифметические операции = 0.027993600000000174
```



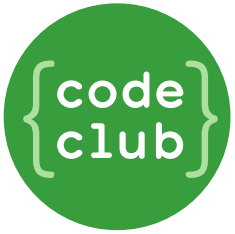
Диапазоны



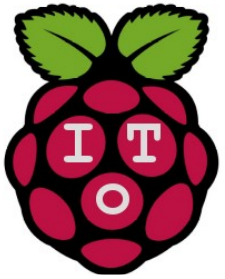

```
1..5              # числа от нижней до верхней границы включительно: 1,2,3,4,5

1...5             # числа от нижней до верхней границы, не включая её: 1,2,3,4

print "Буква среди строчных \n"   if ('a'..'z') === 'M' # false
print "Буква среди заглавных \n"  if ('A'..'Z') === 'M' # true

winter = case temp
  when -40..-25 then "морозно"
  when -24..-15 then "холодно"
  when -14..-5  then "не холодно"
  else "тепло"
end
```

Строки

```
'строка с кавычками (")'                # строка в (одинарных) апострофах
"строка с апострофом (')"               # строка в двойных кавычках
''                                      # пустая строка с нулевой длиной
' '                                     # строка из 1 пробела длиной 1
"строка со специальным символом\n"      # \n = переход на новую строку
"строка с #{2*2}"                       # строка подстановкой значения выражения
"строковый литерал  на нескольких
 строках в программе"                   # текст на нескольких строках как строка
"строка" + 'строка'                     # сцепление строк
"строка " * 3                           # 'строка строка строка ' – повторение
s = "Ru"; s << "by"                     # 'Ruby' – добавление в конец строки

'Ruby'[0]                               # 'R' – 1-й символ строки
'Ruby'[-1]                              # 'y' – последний символ строки
'Ruby'[1..2]                            # 'ub' — символы с № 1 по № 2
'Ruby'[-2..-1]                          # 'by' — с предпоследнего по последний
'Ruby'[2,2]                             # 'by' — 2 символа, начиная с № 2

`исполняемая команда ОС`                # строка в в обратных апострофах
```



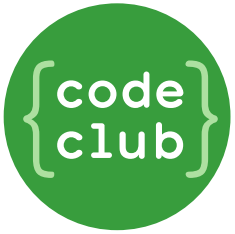
Символы (неизменяемые имена)



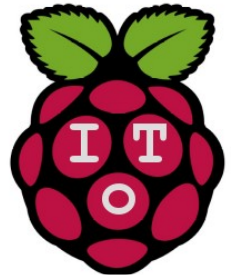

```
:symbol       # символ :symbol != строка "symbol"
:символ       # можно и по-русски (в UTF-8)

rgb = {red: 17, green: 27, blue: 22} # хэш {:red=>17, :green=>27, :blue=>22}

# ПРЕИМУЩЕСТВА использования «символов»:
# + занимают меньше памяти
# + работа с ними быстрее, чем со строками
```

Условия и ветвления (if)

```
if n > 0                                    # проверка условия
  print "Число положительное\n"             # выполняется, если условие == true
elsif n < 0                                 # проверка условия, если 1-е == false
  print "Число отрицательное\n"             # выполняется, если 2-е условие == true
else                                        # иначе – выполняется,
  print "Нуль.\n"                           # если ни одно условие не удовлетворено
end

print "Цифра" if (n >= 0) and (n <= 9) # условное выражение

# операции сравнения:
#   == (равно), != (не равно)
#   > (больше), >= (больше или равно)
#   < (меньше), <= (меньше или равно)

# логические операции:
#   !  (НЕ),  not (НЕ)
#   && (И),   and (И)
#   || (ИЛИ), or  (ИЛИ)
```



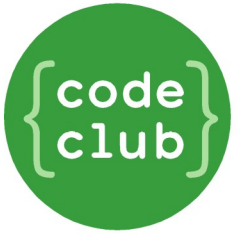
Условия и ветвления (unless)



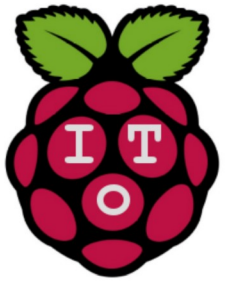

```
unless n > 0                                  # проверка условия
  print "Число не положительное\n"    # выполняется, если условие == false
else                                          # иначе — выполняется,
  print "Число положительное или 0\n" # выполняется, если условие == true
end

print "Цифра" unless (n < 0) and (n > 9) # условное выражение
```

Повторения: циклы

```
loop do                          # бесконечный цикл
  value = sensor.measure()       # команды в теле цикла
  break if value > THRESHOLD     # выход из цикла по условию
end

while t < T_MAX do               # цикл с проверкой продолжения в начале
  p "Температура нормальная"     # команды в теле цикла
end                              # конец цикла

until t == T_MAX do              # цикл с проверкой окончания в начале
  p "Температура нормальная"     # команды в теле цикла
end

begin                            # цикл с проверкой окончания в конце
  p "Температура нормальная"     # команды в теле цикла
end until t == T_MAX

for n in 1..100 do               # цикл с перебором значений последовательности
  print(n)
end
```



Повторители: итераторы




```ruby
100.times { |n| print(n) }              # итератор «повторить» с блоком в {}

100.times do |n|                        # тот же итератор с блоком do ... end
  print(n)
end


0.upto n do |i|                         # итератор «увеличивать до»
  print(i)
end

100.downto 0 do |j|                     # итератор «уменьшать до»
  print(j)
end

(1..100).each { |n| print(n) }          # «каждый» (элемент последовательности)
[25, true,"Ruby"].each { |e| print(e) }
```

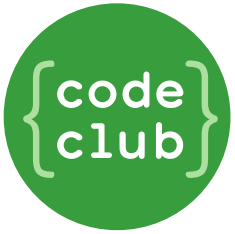
Списки значений: массивы

```ruby
list = Array.new              # создать объект класса Array «массив» (пустой)
empty = []                    # пустой массив длиной 0 элементов
array = [25, true,"Ruby"]     # массив = список из 3-х элементов с №№ от 0 до 2

print array[0]                # 25 (1-й элемент с № 0)
print array[-1]               # "Ruby" (последний элемент = 1-й с конца)

array[1] = 3.14               # заменить значение элемента № 1 с true на 3.14
array << "IoT"                # добавить элемент в конец списка

print array.size              # 4 (размер массива = 4 элемента)
print array.length            # 4 (длина массива = 4 элемента)

matrix = [[11,12],[21,22],[31,32]] # массив массивов = двумерный массив
r1 = matrix[0]                # [11,12] (1-й подмассив = 1-я строка таблицы)
e11 = matrix[0][0]            # 11 (1-й элемент 1-й строки таблицы)
e12 = matrix[0][1]            # 12 (2-й элемент 1-й строки таблицы)
r2 = matrix[1]                # [21,22] (2-я строка таблицы)
e32 = matrix[2][1]            # 32 (2-й элемент 3-й строки таблицы)
```

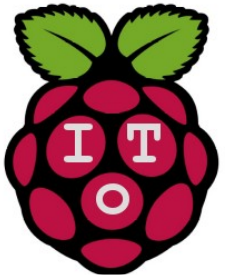
Очень похоже на строки!

Ассоциативные массивы: хэши

```ruby
associative_array = Hash.new                # создать пустой объект класса Hash
empty = {}                                  # пустой хэш длиной 0 пар
hash = {"Ruby" => 1995,"Perl" => 1987}# создать хэш с 2 парами ассоциаций
hash["C++"[ = 1983                          # связать ключ "C++" со значением 1983
print hash["Ruby"]                          # выбрать значение по ключу

hash['Java'[ = 1995                         # добавить в хэш пару ключ=>значение
hash.delete("Java")                         # удалить пару ключ=>значение

hash.each_key do |key|                      # перебрать все ключи в хэше
  print "Язык "+key+" создан в "+hash[key]" году."
end

# 2 способа создать хэш с ключами-символами:
# 1) обычная запись :ключ => значение
rgb = {:red => 17, :green => 27, :blue => 22}
# 2) сокращённая запись ключ : значение
rgb = {red: 17, green: 27, blue: 22}  # {:red=>17, :green=>27, :blue=>22}
```



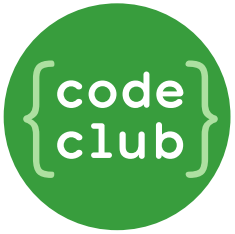
Преобразования



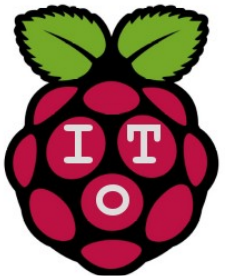

```
number = "3.14".to_f            # строка "3.14" --> число 3.14
number = "3.14".to_i            # строка "3.14" --> число 3

string = 3.14.to_s              # число 3.14 --> строка "3.14"
string = [1,2,3].join(';')      # массив [1,2,3] --> строка "1,2,3"

string = :symbol.to_s           # символ :symbol --> строка "symbol"
symbol = "symbol".to_sym        # строка "symbol" --> символ :symbol
sym = 25.5.to_s.to_sym          # число --> строка --> символ :"25.5"

(1..10).to_a                    # диапазон --> массив [1,2,3,4,5,6,7,8,9,10]
'1;2;3'.split(';')              # строка "1;2;3" --> массив [1, 2, 3]
'Ruby'.split ''                 # строка "Ruby" --> массив ['R','u','b','y']
```

Методы (подпрограммы)

```
def method(p1, p2)        # описание метода с двумя параметрами
  result = p1 + p2        # команды в теле метода
  return result           # вернуть результат действий
end                       # конец описания метода

method(40, 2)             # вызвать метод (возвращаемое значение отбросить)
r = method(40, 2)         # вызвать метод (возвращаемое значение поместить в r)

class LED
  def dot                 # описание метода без параметров для класса LED
    on_for(0.25)          # включить светодиод на 1/4 секунды
    off_for(0.25)         # выключить светодиод на 1/4 секунды
  end                     # конец описания метода
end

led = LED.new(18)         # создать объект led класса LED с параметром 18
3.times { led.dot }       # 3 раза вызвать для объекта led метод dot()
```



Справочник и учебник по Ruby

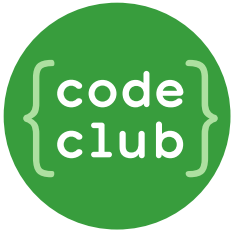


Краткий справочник по синтаксису языка — в файле:

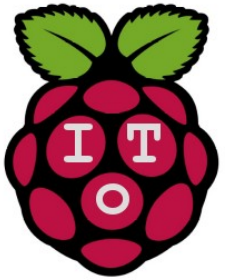
`IoT-Ruby_syntax.pdf`

Учебник для начинающих по языку — в файле:

`~/Documents/books/Learn_To_Program-Ch.Pine-ru.pdf`



Что нужно ещё?



Подумайте,
каких конструкция языка
вам не хватает,
чтобы запрограммировать свои идеи?