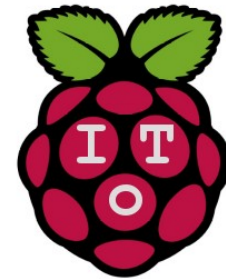




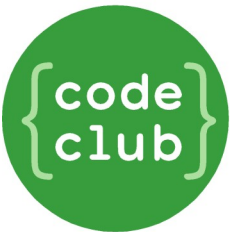
Internet of Things



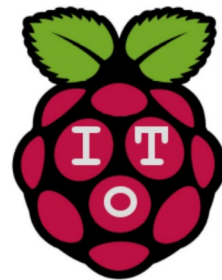
Программирование на **Ruby**

Шадринск
2018-2019

М. В. Шохирев



Язык Ruby



Название: **Ruby** (англ. ruby ['ru:bi] [ру́би]— рубин).

Создан: в 1995 году. Начало распространения вне Японии – с 1998 г.

Создатель: японец **Юкихиро «Matz» Мацумото** (松本行弘 = まつもとゆきひろ).

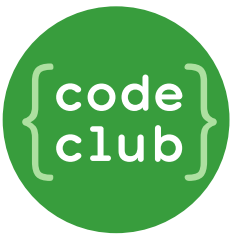
Характеристика: интерпретируемый, динамический, сценарный, высокоуровневый полностью объектно-ориентированный (мультипарадигменный) язык программирования.

Лицензия: реализация интерпретатора языка (для всех аппаратных платформ) является полностью свободной.

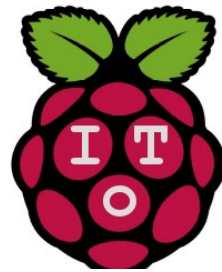
Похож на языки: Perl, Eiffel, Smalltalk, Python, Lisp, Dylan, Clu.

Принципы:

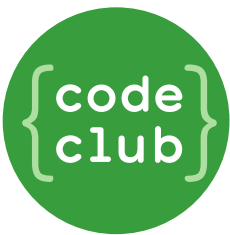
- Ориентация на разработчика, на то, как ему лучше решать программистские задачи.
- *POLS* = Principle Of Least Surprise = «Правило наименьшего удивления» — программные конструкции должны быть логичными, интуитивно понятными и ожидаемыми.
- *TIMTOWTDI* [Tim Toady] = There Is More Than One Way To Do It ~ «Есть не один способ это сделать» — один и тот же результат можно получить несколькими различными способами.



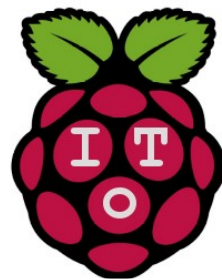
Программы на Ruby



- 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
+1234567890
1_234_567_890
-1234567890
```

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

```
0b1001_0110_1011
076543210
0xFEDCBA9876543210
```

```
# целое число (в двоичной системе счисления)
# целое число (в восьмеричной системе счисления)
# целое число (в шестнадцатеричной системе счисления)
```

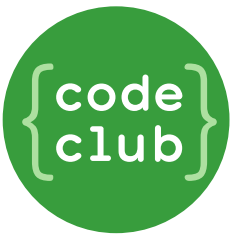
```
3.141592653
```

```
# дробное число
```

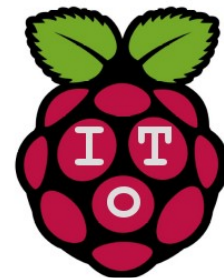
```
-123e4
+123E4
123E+4
123e-4
```

```
# -123000.0 — число в «научной» записи
# 123000.0      (в экспоненциальной нотации)
# 123000.0  = 123 * 10**4
# 0.0123     = 123 * 10**-4
```

```
((1+2) * (3-4/5.0) % 6) ** 7 # арифметические операции = 0.0279936000000000174
```



Диапазоны

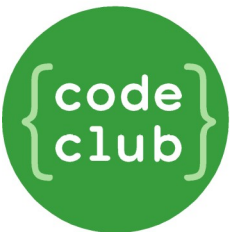


(1..5)

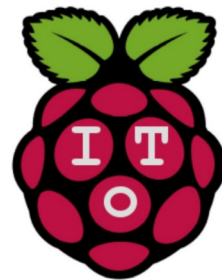
числа от нижней до верхней границы включительно: 1,2,3,4,5

(1...5)

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



Строки



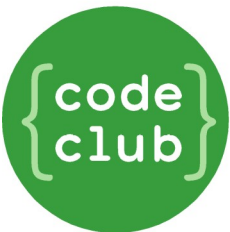
```
'строка с кавычками (") '  
"строка с апострофом (') "  
' '  
' '
```

```
"строка со специальным символом\n"  
"строка с #{2*2}"  
"строковый литерал на нескольких  
строках в программе"  
"строка" + 'строка'  
"строка " * 3  
s = "Ru"; s << "by"
```

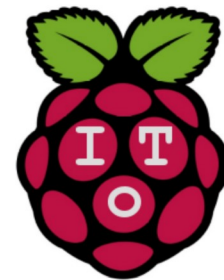
```
'Ruby' [0]  
'Ruby' [-1]  
'Ruby' [1..2]  
'Ruby' [-2..-1]  
'Ruby' [2, 2]
```

```
`исполняемая команда ОС`
```

```
# строка в (одинарных) апострофах  
# строка в двойных кавычках  
# пустая строка с нулевой длиной  
# строка из 1 пробела длиной 1  
# \n = переход на новую строку  
# строка подстановкой значения выражения  
  
# текст на нескольких строках как строка  
# сцепление строк  
# 'строка строка строка ' - повторение  
# 'Ruby' - добавление в конец строки  
  
# 'R' - 1-й символ строки  
# 'y' - последний символ строки  
# 'ub' - символы с № 1 по № 2  
# 'by' - с предпоследнего по последний  
# 'by' - 2 символа, начиная с № 2  
  
# строка в в обратных апострофах
```



Условия и ветвления (if)



```
if n > 0
    print "Число положительное\n"
elif n < 0
    print "Число отрицательное\n"
else
    print "Нуль.\n"
end
```

```
# проверка условия
# выполняется, если условие == true
# проверка условия, если 1-е == false
# выполняется, если 2-е условие == true
# иначе — выполняется,
# если ни одно условие не удовлетворено
```

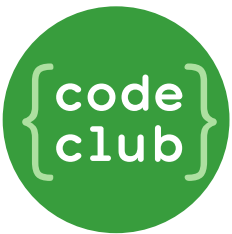
```
print "Цифра" if (n >= 0) and (n <= 9) # условное выражение
```

операции сравнения:

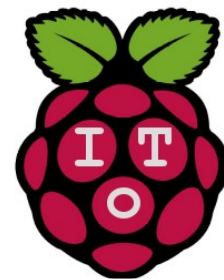
```
# == (равно), != (не равно)
# > (больше), >= (больше или равно)
# < (меньше), <= (меньше или равно)
```

логические операции:

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

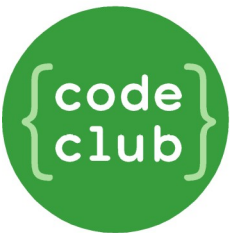


Условия и ветвления (unless)

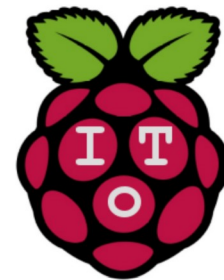


```
unless n > 0                                # проверка условия
  print "Число не положительное\n"          # выполняется, если условие == false
else                                         # иначе — выполняется,
  print "Число положительное или 0\n"       # выполняется, если условие == true
end

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

Повторения : циклы



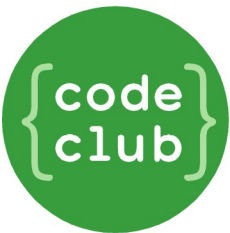
```
loop do                                     # бесконечный цикл
  value = sensor.measure()                 # команды в теле цикла
  break if value > THRESHOLD               # выход из цикла по условию
end

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

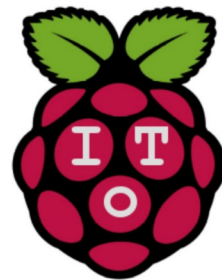
until t == T_MAX do                        # цикл с проверкой окончания в начале
  p "Температура нормальная"              # команды в теле цикла
end

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



```
100.times { |n| print(n) }
```

итератор «повторить» с блоком в {}

```
100.times do |n|  
  print(n)  
end
```

тот же итератор с блоком do ... end

```
0.upto n do |i|  
  print(i)  
end
```

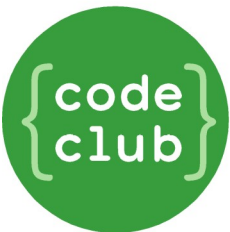
итератор «увеличивать до»

```
100.downto 0 do |j|  
  print(j)  
end
```

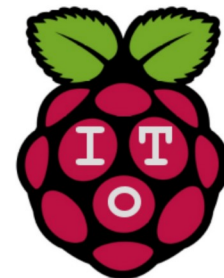
итератор «уменьшать до»

```
(1..100).each { |n| print(n) }  
[25, true, "Ruby"].each { |e| print(e) }
```

«каждый» (элемент последовательности)



Списки значений: массивы



```
list = Array.new  
empty = []  
array = [25, true, "Ruby"]
```

```
# создать объект класса Array «массив» (пустой)  
# пустой массив длиной 0 элементов  
# массив = список из 3-х элементов с № от 0 до 2
```

```
print array[0]  
print array[-1]
```

```
# 25 (1-й элемент с № 0)  
# "Ruby" (последний элемент = 1-й с конца)
```

```
array[1] = 3.14  
array << "IoT"
```

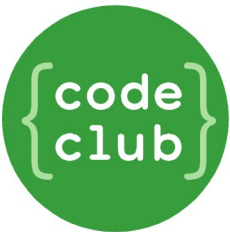
```
# заменить значение элемента № 1 с true на 3.14  
# добавить элемент в конец списка
```

```
print array.size  
print array.length
```

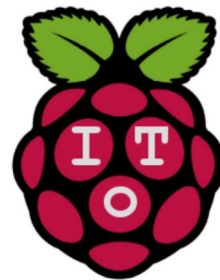
```
# 4 (размер массива = 4 элемента)  
# 4 (длина массива = 4 элемента)
```

Очень похоже
на строки!

```
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)
```



Ассоциативные массивы: ХЭШИ



```
assoc_array = Hash.new
empty = {}
hash = {"Ruby" => 1995, "Perl" => 1987}
hash["C++"] = 1983

print hash["Ruby"]

hash.each_key do |key|
  puts "Язык "+key+" создан в "+hash[key].to_s+" году."
end

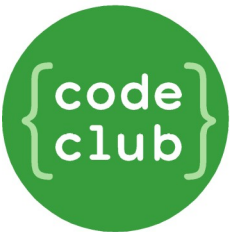
key = "Perl"
value = hash.delete(key)
```

создать пустой объект класса Hash
пустой хэш длиной 0 пар
создать хэш с 2 парами ассоциаций
связать ключ "C++" со значением 1983

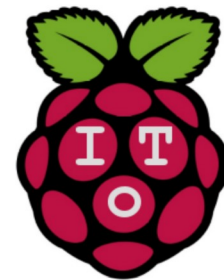
выбрать значение по ключу

перебрать в хэше все ключи

удалить по ключу,
вернуть удалённое значение



Методы (подпрограммы)



```
def method(p1, p2)
  result = p1 + p2
  return result
end
```

```
# описание метода с двумя параметрами
# команды в теле метода
# вернуть результат действий
# конец описания метода
```

```
method(40, 2)
r = method(40, 2)
```

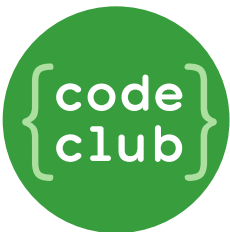
```
# вызвать метод (возвращаемое значение отбросить)
# вызвать метод (возвращаемое значение поместить в r)
```

```
class LED
  def dot
    on_for(0.25)
    off_for(0.25)
  end
end
```

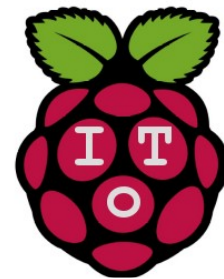
```
# описание метода без параметров для класса LED
# включить светодиод на 1/4 секунды
# выключить светодиод на 1/4 секунды
# конец описания метода
```

```
led = LED.new(18)
3.times { led.dot }
```

```
# создать объект led класса LED с параметром 18
# 3 раза вызвать для объекта led метод dot()
```



Справочник и учебник по **Ruby**

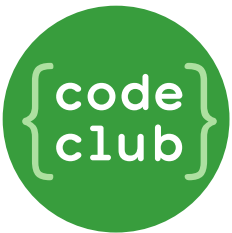


Краткий справочник по синтаксису языка — в файле:

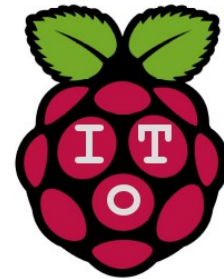
`IoT-Ruby_syntax.pdf`

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?*