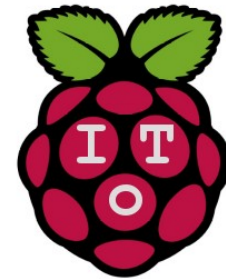




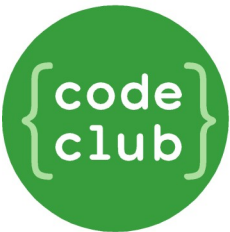
Internet of Things



Методы и блоки в **Ruby**

Шадринск
2018-2019

М. В. Шохирев



Подпрограммы

Подпрограммы — наборы команд, имеющие название (*имя подпрограммы*), которые можно многократно выполнять, *ВЫЗЫВАЯ* их по имени.

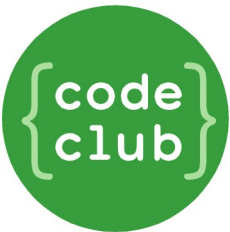
```
procedure () # вызов подпрограммы по имени
```

Подпрограммы обычно описываются в начале программы (или размещаются в отдельном исходном файле, который подключается к основной программе).

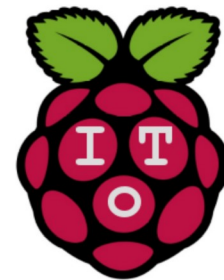
```
def procedure()      # описание подпрограммы без параметров
    # ...           # полезные действия в подпрограмме
end                 # конец описания подпрограммы
```

В описании подпрограммы можно предусмотреть *параметры* — переменные, значения которых можно определять при вызове подпрограммы для обработки в подпрограмме.

```
def function(prm1, prm2)      # описание функции с параметрами
    return 0                  # возврат результата функции
end                            # конец описания функции
```



Подпрограммы: функции и процедуры



Подпрограммы делятся на 2 вида: *функции*, которые возвращают вычисленный результат, и *процедуры*, которые выполняют действия, но не возвращают результата.

```
def procedure(prm1, prm2)
    # ...
end
```

```
# описание процедуры с параметрами
# полезные действия в подпрограмме
# конец описания подпрограммы
```

```
def function(prm1, prm2)
    return 0
end
```

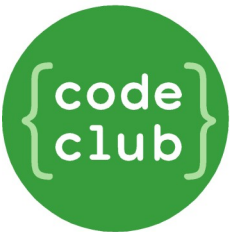
```
# описание функции с параметрами
# возврат результата функции
# конец описания функции
```

```
procedure("1-й аргумент", argument2)
```

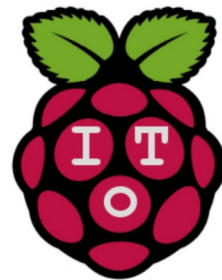
```
# вызов процедуры с параметрами
```

```
result = function(arg1, 2.0)
```

```
# вызов функции с параметрами
```



Подпрограммы: методы



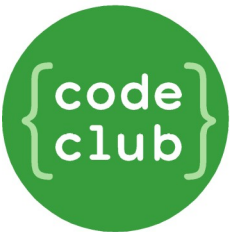
Подпрограммы, описанные в классах для выполнения действий с объектами этих классов, называются *методами*.

```
class SomeClass
  def some_method_1()      # описание метода без параметров в классе
    # ...                 # полезные действия
  end                     # конец описания метода

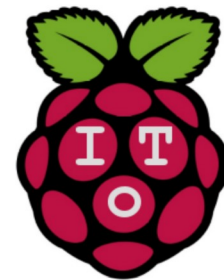
  def some_method_2()      # описание метода с параметрами
    # ...                 # полезные действия
    return expression      # возврат результата функции
  end                     # конец описания метода
end

some_object = SomeClass.new # создать объект класса

some_object.some_method_1() # выполнить метод объекта
result = some_object.some_method_2() # выполнить другой метод объекта
```



Методы объектов класса



```
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)
```

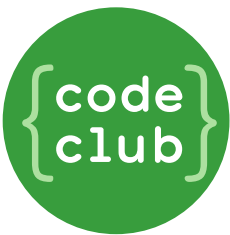
```
# вызвать метод (возвращаемое значение отбросить)
# вызвать метод (возвращаемое значение поместить в r)
```

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

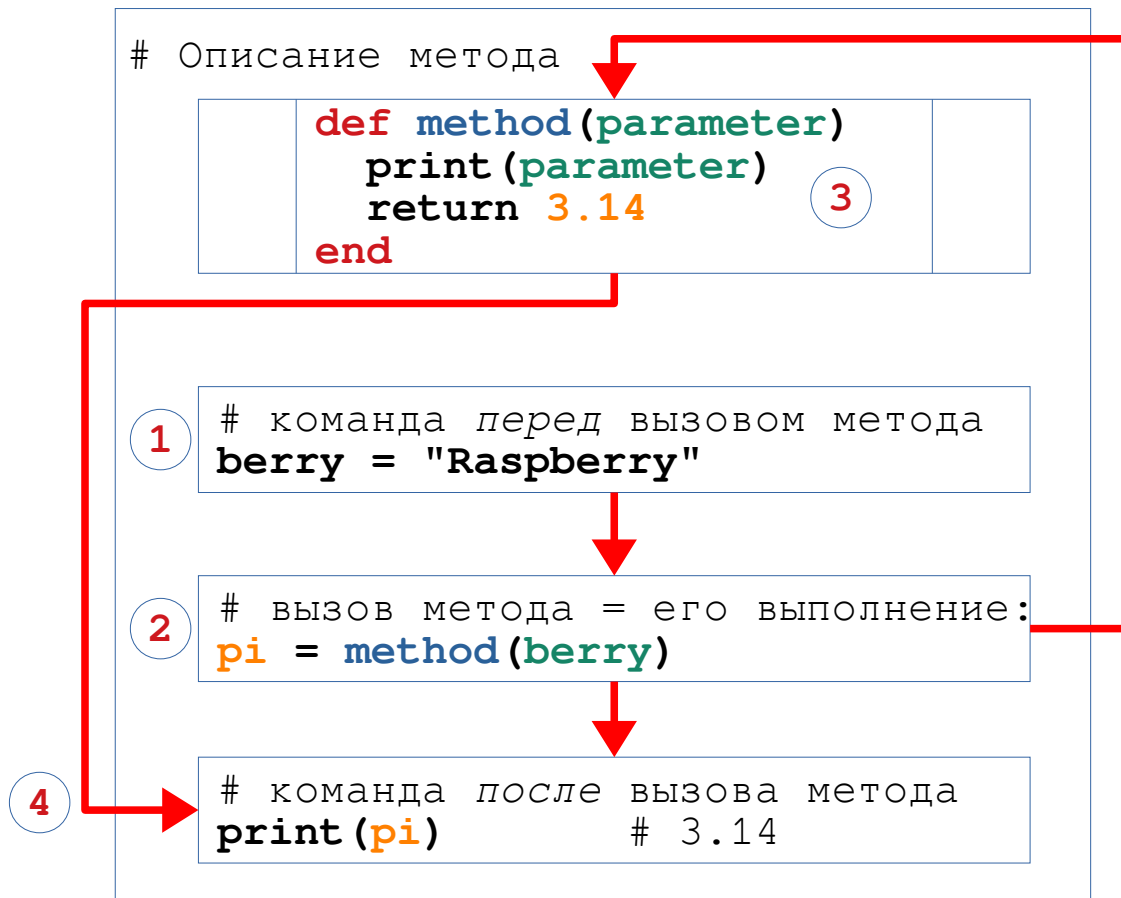
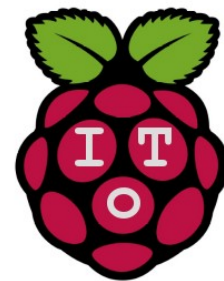
```
# описание метода без параметров для класса LED
# включить светодиод на 1/4 секунды
# выключить светодиод на 1/4 секунды
# конец описания метода
```

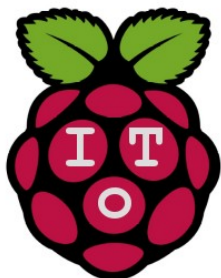
```
led = LED.new(18)
3.times { led.dot }
```

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



Выполнение метода





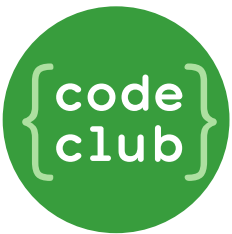
Блок — это программный код, который можно передать любому методу последним аргументом внутри фигурных скобок { } или ключевых слов **do** и **end**.

```
# метод, которому можно передавать блок как параметр
def method_1 # начало описания метода
  print "> Начало работы метода 1\n" # команды в теле метода
  yield if block_given? # блок выполняется только по yield
  print "< Конец работы метода 1\n" # команды в теле метода
end # конец описания метода

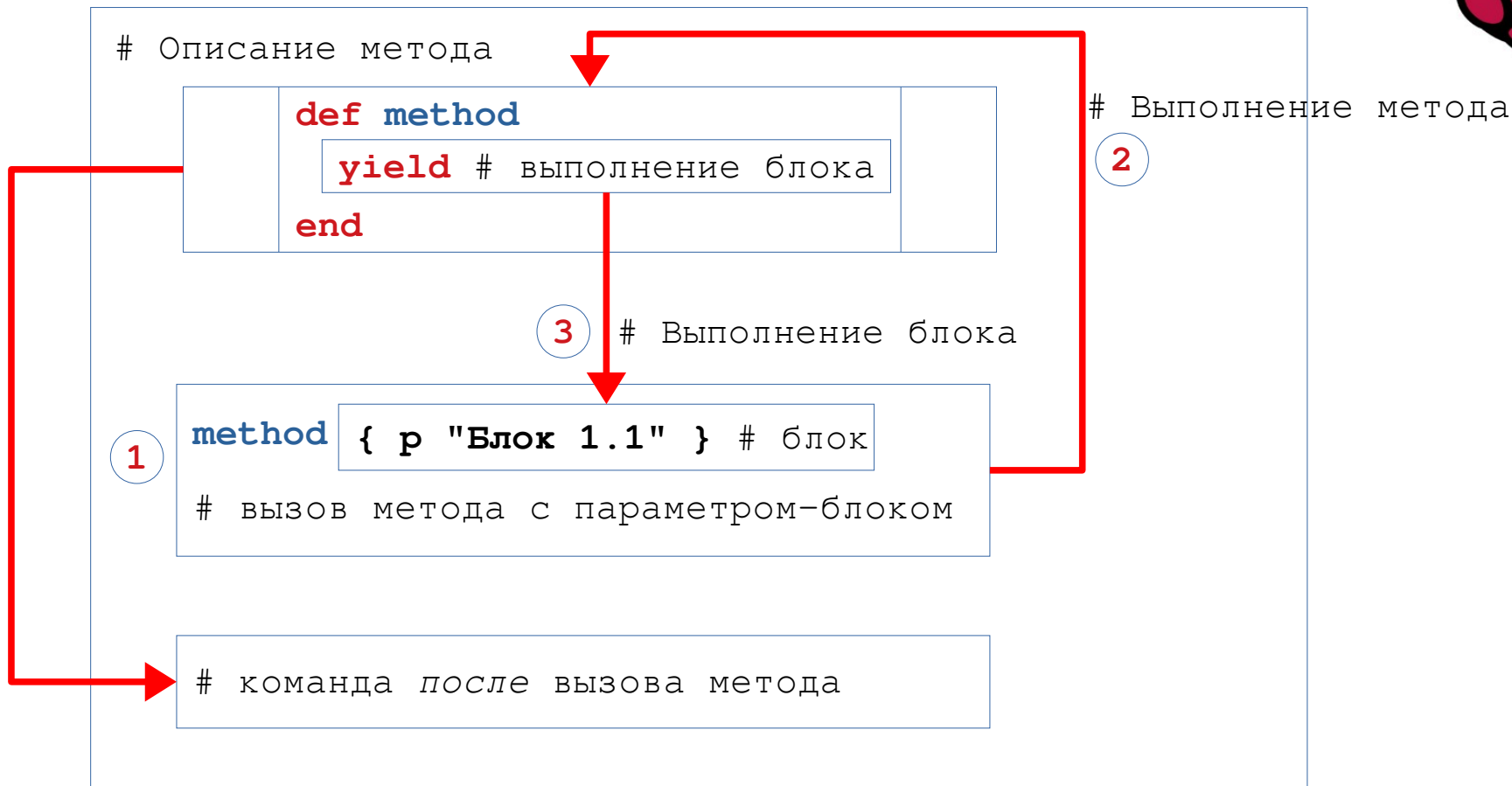
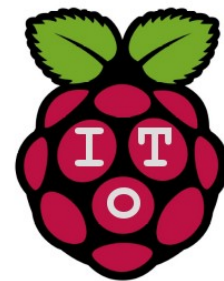
method_1 # вызов метода без блока

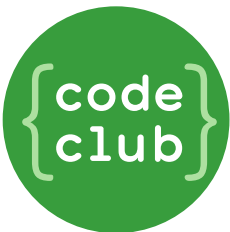
method_1 { p "Выполняется блок 1.1" } # вызов метода с блоком между { и }

# вызов метода с блоком между do и end
method_1 do # начало блока
  print "~ Выполняется блок 1.21\n" # команды в теле блока
end # конец блока
```

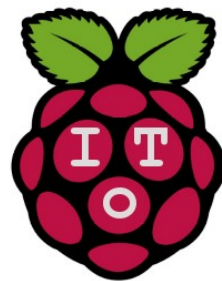


Выполнение блока





Блоки с параметрами



При вызове метода блоку можно передавать значения параметров (аргументы) так:

```
method { |parameter| do_something_with(parameter) }
```

или так:

```
do |parameter|  
  do_something_with(parameter)  
end
```

В методе параметр передаётся блоку так:

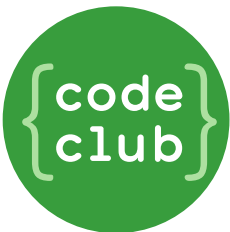
```
yield(parameter)
```

метод с параметром

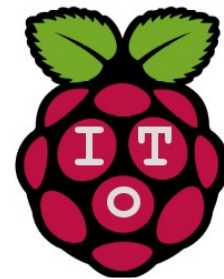
```
def method_2(parameter) # начало описания метода  
  print "> Начало работы метода 1\n" # команды в теле метода  
  yield(parameter) # выполнить блок с параметром  
  print "< Конец работы метода 1\n" # команды в теле метода  
end # конец описания метода
```

```
method_2("Блок 2.1") { |p1| print "---"+p1+"---" }
```

```
method_2("Блок 2.2") { |p2| print "==="+p2+"===" }
```



Справочник и учебник по **Ruby**



Краткий справочник по синтаксису языка — в файле:

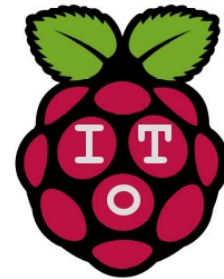
`IoT-Ruby_syntax.pdf`

Учебник для начинающих по языку — в файле:

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



Что непонятно?



*Какие конструкции языка
для вас сложны?*

Что нужно объяснить дополнительно?