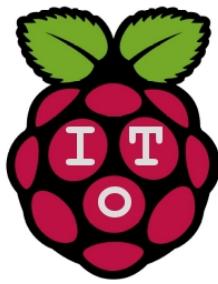


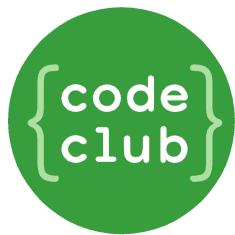
Internet of Things



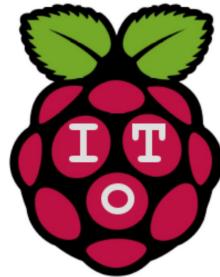
Методы и блоки в **Ruby**

Шадринск
2018-2019

M. B. Шохирев



Подпрограммы



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

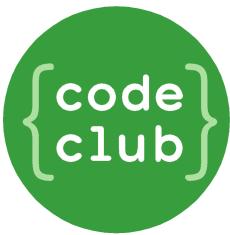
```
procedure() # вызов подпрограммы по имени
```

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

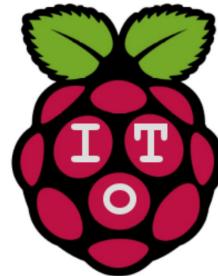
```
def procedure()
# ...
end # описание подпрограммы без параметров
# полезные действия в подпрограмме
# конец описания подпрограммы
```

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

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

def function(prm1, prm2)
    return 0
end

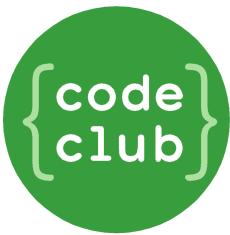
procedure("1-й аргумент", argument2)      # вызов процедуры с фактическими
                                              # значениями параметров

result = function(arg1, 2.0)                  # вызов функции с фактическими
                                              # значениями параметров
                                              # и присвоением результата

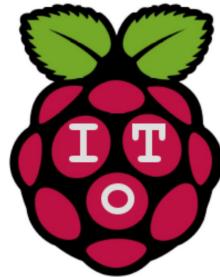
# описание процедуры с параметрами
# полезные действия в подпрограмме
# конец описания подпрограммы

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

В **Ruby** все методы возвращают значение по **return** или значение последнего выражения.



Подпрограммы: методы



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

```
class SomeClass
  def method_1()
    # ...
  end

  def method_2(p1)
    # ...
    return expression
  end
end

some_object = SomeClass.new

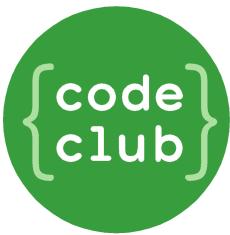
some_object.method_1()
result = some_object.method_2(100)
```

в описании класса
описание метода без параметров
полезные действия
конец описания метода

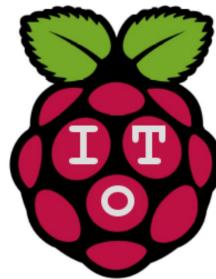
описание метода с параметром
полезные действия
возврат результата функции
конец описания метода

создать объект класса

выполнить метод объекта
выполнить другой метод объекта



Методы объектов класса

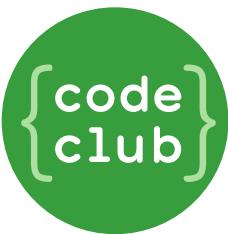


```
def method(p1, p2)      # описание метода с двумя параметрами (в классе Object)
    result = p1 + p2    # команды в теле метода
    return result        # вернуть результат действий
end                      # конец описания метода

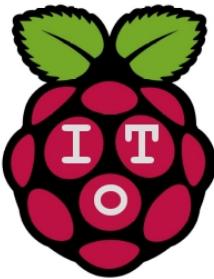
method(40, 2)            # вызвать метод (возвращаемое значение отбросить)
r = method(40, 2)         # вызвать метод (возвращаемое значение поместить в r)

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()
```



Выполнение метода



```
# Описание метода
def method(parameter)
    print(parameter)
    return 3.14
end
```

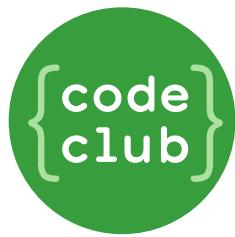
3

1 # команда перед вызовом метода
berry = "Raspberry"

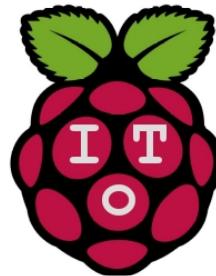
2 # вызов метода = его выполнение:
pi = method(berry)

4 # команда после вызова метода
print(pi) # 3.14

Выполнение метода



Классы: пример разработки



Мы хотим посчитать количество «счастливых» билетов в катушке с номерами от 0 до 999999 и напечатать номера этих билетов.

Для этого нужно: (1) описать класс с требуемыми характеристиками (диапазон номеров билетов) и (2) определить необходимые действия (сформировать список билетов со «счастливыми» номерами).

```
class RollOfTickets
# ...
end

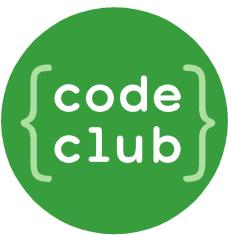
roll = RollOfTickets.new(0, 999999)

lucky_tickets = roll.lucky

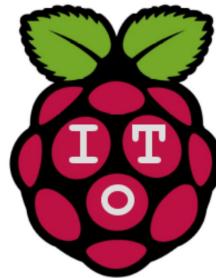
print lucky_tickets.size

print lucky_tickets
```

описание класса «КатушкаБилетов»
описание методов методов
действия по достижению цели:
новая катушка из 1_000_000 билетов
метод lucky находит счастливые билеты
вывести количество счастливых билетов
вывести список счастливых билетов



Классы: метод-инициализатор



В описании класса «КатушкаБилетов» предусмотреть требуемые атрибуты: нижнюю и верхнюю границы диапазона номеров для билетов в катушке.

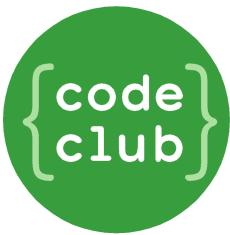
```
class RollOfTickets
  def initialize(lo=0, hi=999999)
    @lo = lo
    @hi = hi
  end

  # ...
end

roll_1 = RollOfTickets.new(0, 999)           # новая катушка из 1_000 билетов

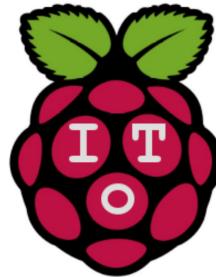
roll_2 = RollOfTickets.new                   # новая катушка из 1_000_000 билетов
                                              # (по умолчанию: lo=0, hi=999999)

roll_3 = RollOfTickets.new(0, 99999)         # новая катушка из 100_000 билетов
```



Классы:

Открытые методы

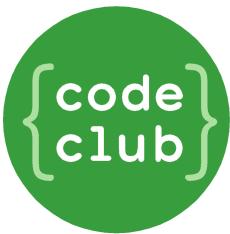


В описании класса «КатушкаБилетов» определить необходимый метод: сформировать список билетов со «счастливыми» номерами.

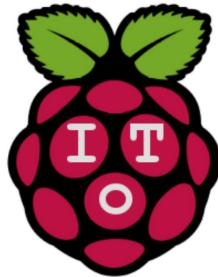
```
class RollOfTickets # в описание класса добавить

  public # public - «открытые» методы для вызова без ограничений
    def lucky() # метод поиска «счастливых билетов»
      tickets = [] # массив для найденных билетов
      for i in (@lo..@hi) # искать от нижней до верхней границы
        if (self.lucky?(i)) # метод проверки на «счастливость»
          tickets << i # добавить найденный билет массив
        end
      end
      return tickets # вернуть список найденных билетов
    end
  end

  roll = RollOfTickets.new(0, 999) # roll - новая катушка из 1000 билетов
  lucky_tickets = roll.lucky # метод lucky находит счастливые билеты
```



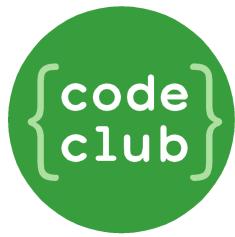
Классы: закрытые и защищённые методы



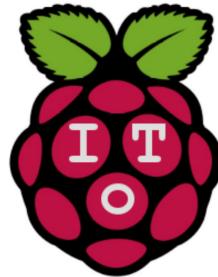
```
class RollOfTickets          # в описание класса добавить
protected      # protected - «захищённые» методы для вызова из объектов класса
def lucky?(i)           # метод проверки на «счастливость»
  h1, h2, h3 = head(i)    # разбить «голову» целого на 3 числа
  t1, t2, t3 = tail(i)    # разбить «хвост» целого на 3 числа
  return ((h1+h2+h3) == (t1+t2+t3))
end

private        # private - «закрытые» методы для вызова только в этом объекте
def head(i)           # 123456
  h = (i/1000).to_i
  return (h/100).to_i, ((h % 100)/10).to_i, ((h % 100)%10).to_i # 1,2,3
end

def tail(i)           # 123456
  t = i % 1000
  return (t/100).to_i, ((t % 100)/10).to_i, ((t % 100)%10).to_i # 4,5,6
end
end
```



Классы: методы класса



В описании класса можно определять методы для управления поведением класса, а не отдельных объектов: например, добавочный конструктор объектов класса.

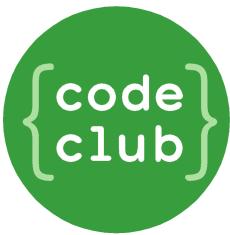
```
class RollOfTickets # в описание класса добавить

  def RollOfTickets.new_roll(n) # для создания катушки по количеству
    RollOfTickets.new(0, n-1) # вызываем конструктор с параметрами
  end

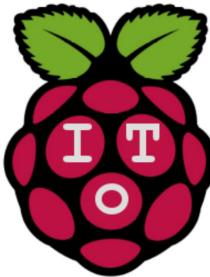
end

roll = RollOfTickets.new_roll(1000000) # новая катушка из 1000000 билетов

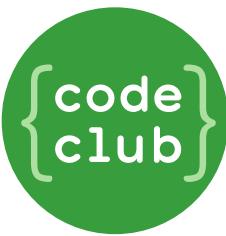
print roll.lucky().size() # найти количество счастливых билетов
```



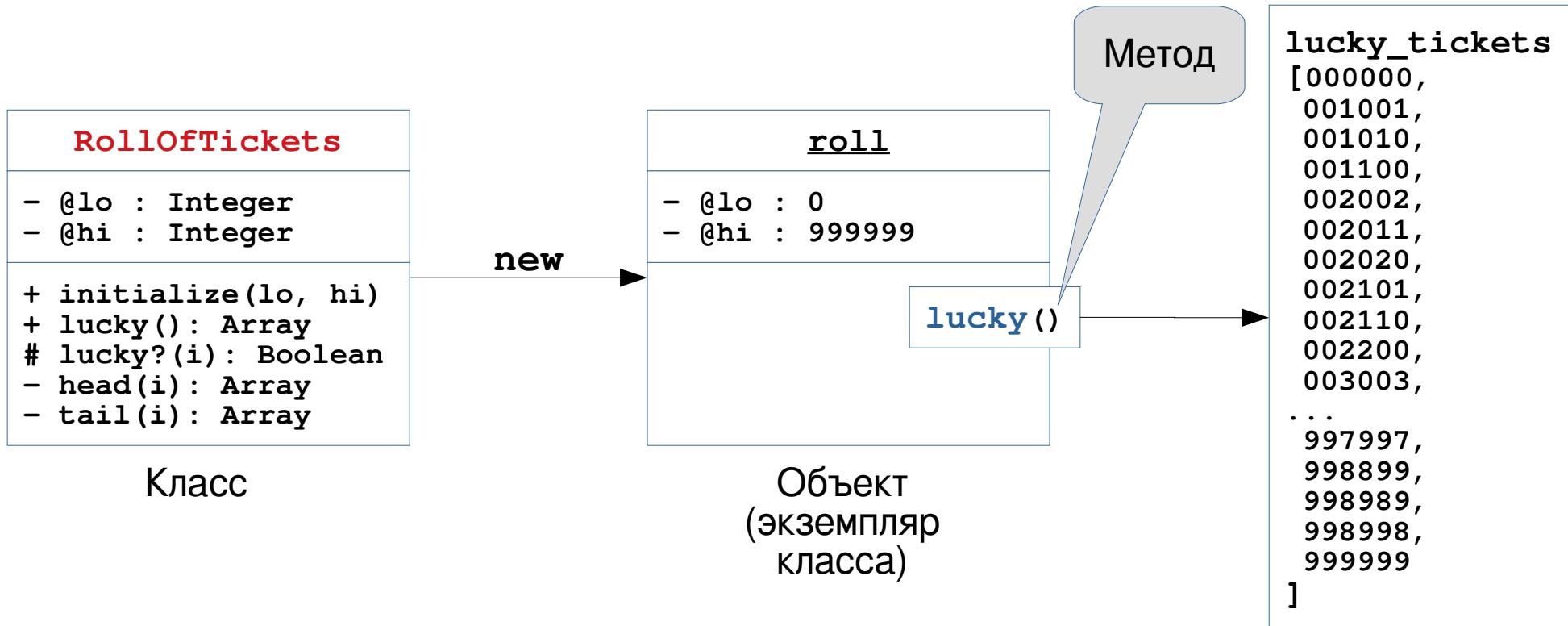
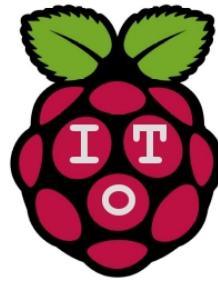
ООП: разработка «сверху вниз»

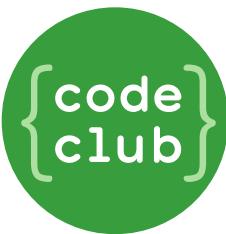


1. Придумать класс (с подходящим именем), в котором будут описаны требуемые характеристики и необходимые действия.
2. В классе описать атрибуты (переменные объектов), в которых будут храниться значения требуемых характеристик (свойств объектов).
3. Описать метод-инициализатор (`initialize`), который при создании каждого объекта класса будет задавать начальные значения атрибутов (и выполнять другие начальные действия).
4. Описать открытые методы, которые будет выполнять полезные действия с каждым из объектов, изменять значения его атрибутов (состояние этого объекта).
Набор открытых методов представляет из себя интерфейс объекта, через который можно управлять поведением каждого из объектов класса.
5. Описать защищённые и закрытые методы (при необходимости), которые будут выполнять скрытые действия по изменению состояния объекта или другие вспомогательные действия.
6. Создать нужное количество объектов этого класса с помощью конструктора `new`, при необходимости передавая ему параметры для инициализации объектов.
7. Управлять созданными объектами с помощью методов, чтобы получить требуемый результат.

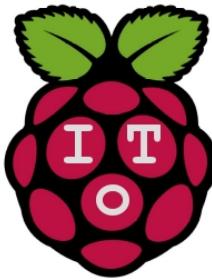


ООП: моделирование реальных объектов





Выполнение блока



Описание метода

```
def method
```

```
    yield # выполнение блока
```

```
end
```

Выполнение метода

2

1

3

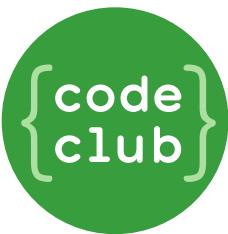
Выполнение блока

```
method
```

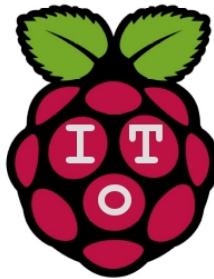
```
{ p "Блок 1.1" } # блок
```

вызов метода с параметром-блоком

команда после вызова метода



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
```

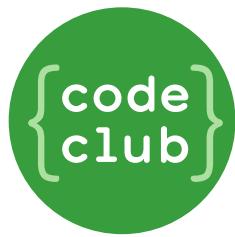
В методе параметр передаётся блоку так:

```
yield(parameter)
```

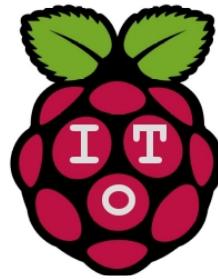
```
# метод с параметром
def method_2(parameter)          # начало описания метода
  print "> Начало работы метода 1\n" # команды в теле метода
  yield(parameter)                # выполнить блок с параметром
  print "< Конец работы метода 1\n" # команды в теле метода
end                             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

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

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



Справочник и учебник по Ruby

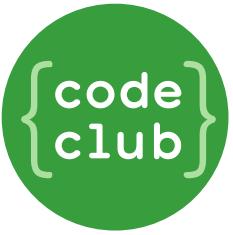


Краткий справочник по синтаксису языка — в файле:

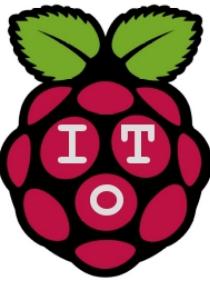
IoT-Ruby_syntax.pdf

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?