8. Пишем свои методы

Как мы уже видели, циклы и итераторы позволяют нам делать одно и то же (выполнять тот же самый код) снова и снова. Однако, иногда мы хотим сделать одно и то же несколько раз, но в разных частях программы. Например, мы бы разрабатывали, скажем, программу опроса для студента-психолога. Судя по разговорам со знакомыми студентами-психологами и по опросам, которые они мне предлагали, она, наверное, должна быть примерно такой:

```
puts 'Здравствуйте! И спасибо, что Вы нашли время, чтобы'
puts 'помочь мне в этом исследовании.  Мое исследование'
puts 'связано с изучением того, как люди относятся к'
puts 'мексиканской еде. Просто думайте о мексиканской еде'
puts 'и попробуйте отвечать на все вопросы честно,'
puts 'только словами "да" или "нет".  Моё исследование'
puts 'не имеет ничего общего с ночным недержанием мочи.'
puts
#  Мы задаём эти вопросы, но игнорируем ответы на них.
goodAnswer = false
while (not goodAnswer)
  puts 'Вам нравится есть такос?'
  answer = gets.chomp.downcase
  if (answer == 'да' or answer == 'нет')
    goodAnswer = true
  else
    puts 'Пожалуйста, отвечайте "да" или "нет".'
  end
end
goodAnswer = false
while (not goodAnswer)
  puts 'Вам нравится есть бурритос?'
  answer = gets.chomp.downcase
  if (answer == 'да' or answer == 'нет')
    goodAnswer = true
  else
    puts 'Пожалуйста, отвечайте "да" или "нет".'
  end
end
#  Мы, однако, обращаем внимание на *этот* вопрос.
goodAnswer = false
while (not goodAnswer)
  puts 'Вы мочитесь в постель?'
  answer = gets.chomp.downcase
  if (answer == 'да' or answer == 'нет')
```

```
    goodAnswer = true
    if answer == 'да'
      wetsBed = true
    else
      wetsBed = false
    end
  else
    puts 'Пожалуйста, отвечайте "да" или "нет".'
  end
end
goodAnswer = false
while (not goodAnswer)
  puts 'Вам нравится есть чимичангас?'
  answer = gets.chomp.downcase
  if (answer == 'да' or answer == 'нет')
    goodAnswer = true
  else
    puts 'Пожалуйста, отвечайте "да" или "нет".'
  end
end
puts 'И ещё несколько вопросов...'
goodAnswer = false
while (not goodAnswer)
  puts 'Вам нравится есть сопапиллас?'
  answer = gets.chomp.downcase
  if (answer == 'да' or answer == 'нет')
    goodAnswer = true
  else
    puts 'Пожалуйста, отвечайте "да" или "нет".'
  end
end
#  Задайте много других вопросов о мексиканской еде.
puts
puts 'ПОЯСНЕНИЕ:'
puts 'Спасибо за то, что Вы нашли время, чтобы помочь'
puts 'этому исследованию.  На самом деле, это исследование'
puts 'не имеет ничего общего с мексиканской едой.  Это'
puts 'исследование ночного недержания мочи.  Мексиканская еда'
puts 'присутствовала только затем, чтобы усыпить Вашу бдительность'
puts 'в надежде, что Вы будете отвечать более'
puts 'правдиво.  Ещё раз спасибо.'
puts
puts wetsBed
```

```
Здравствуйте! И спасибо, что Вы нашли время, чтобы
```

```
помочь мне в этом исследовании.  Моё исследование
связано с изучением того, как люди относятся к
мексиканской еде.  Просто думайте о мексиканской еде
и попробуйте отвечать на все вопросы честно,
только словами "да" или "нет".  Моё исследование
не имеет ничего общего с ночным недержанием мочи.
Вам нравится есть такос?
да
Вам нравится есть бурритос?
да
Вы мочитесь в постель?
никогда!
Пожалуйста, отвечайте "да" или "нет".
Вы мочитесь в постель?
нет
Вам нравится есть чимчангас?
да
И ещё несколько вопросов...
Вам нравится есть сопапиллас?
да
ПОЯСНЕНИЕ:
Спасибо за то, что Вы нашли время, чтобы помочь
этому исследованию.  На самом деле, это исследование
не имеет ничего общего с мексиканской едой.  Это
исследование ночного недержания мочи.  Мексиканская еда
присутствовала только затем, чтобы усыпить Вашу бдительность
в надежде, что Вы будете отвечать более
правдиво.  Ещё раз спасибо.
false
```

Это была довольно длинная программа со многими повторениями. (Все разделы программы с вопросами о мексиканской еде были одинаковыми, а вопрос о недержании мочи отличался совсем немного.) Повторение — это нехорошая штука. И всё же, мы не можем поместить его в один большой цикл или итератор, поскольку иногда нам нужно кое-что сделать между вопросами. В подобных ситуациях лучше всего написать метод. Вот так:

```
def sayMoo  #  скажи: "Му"
  puts 'мууууууу...'
end
```

Ээ... наша программа не выполняет **sayMoo**. Почему же? Потому что мы не сказали ей это делать. Мы сказали ей, *как* мычать методом **sayMoo**, но мы фактически так и не сказали ей *сделать* это. Давайте попытаемся по-другому:

```
def sayMoo  #  скажи: "Му"
  puts 'мууууууу...'
```

```
end
sayMoo
sayMoo
puts 'куан-куан'
sayMoo
sayMoo
```

```
мууууууу...
мууууууу...
куан-куан
мууууууу...
мууууууу...
```

Ааа, гораздо лучше. (Если вы не говорите по-французски, поясняю: в середине программы была французская утка. Во Франции утки говорят: *"куан-куан"*.)

Итак, с помощью **def** мы определили метод **sayMoo**. (Имена методов, как и имена переменных, начинаются со строчной буквы. Есть, однако, несколько исключений таких, как + или ==.) Но разве методы не должны всегда ассоциироваться с объектами? Ну да, должны, и в этом случае (как и в случаях с **puts** и **gets**) метод просто ассоциируется с объектом, представляющим всю программу. В следующей главе мы увидим, как добавлять методы к другим объектам. Но сначала...

Параметры метода

Вы, должно быть, заметили, что некоторые методы (такие, как **gets**, **to_s**, **reverse**...) просто вызываются у какого-нибудь объекта. Однако, другие методы (такие, как **+**, **-**, **puts**...) принимают параметры для указания объекту, как выполнять метод. Например, вы не скажете просто **5+**, правда? Этим вы говорите числу **5** прибавить, но не говорите ему *что* прибавить.

Чтобы определить параметр для метода **sayMoo** (скажем, количество мычаний), мы должны сделать так:

```
def sayMoo numberOfMoos
  puts 'мууууууу...'*numberOfMoos
end
sayMoo 3
puts 'хрю-хрю'
sayMoo  #  Это должно вызвать ошибку, потому что параметр отсутствует.
```

```
мууууууу...мууууууу...мууууууу...
хрю-хрю
#<ArgumentError: wrong number of arguments (0 for 1)>
```

[#<Ошибка аргументов: неверное число аргументов (0 вместо 1)> — *Прим. перев.*]

numberOfMoos — это переменная, которая указывает на параметр, переданный в метод. Я повторю это ещё раз, хотя это всё равно звучит немного запутанно: **numberOfMoos** — это переменная, которая указывает на параметр, переданный в

метод. Так, если я напишу **sayMoo 3**, то параметр равен **3**, а переменная **numberOfMoos** указывает на **3**.

Как видите, параметр теперь *обязателен*. В конце концов, каким образом **sayMoo** должен повторять **'myyyyyyy...'**, если вы не передадите ему параметр? Ваш бедный компьютер не сообразит.

Если объекты в Ruby подобны существительным в английском языке, а методы подобны глаголам, то вы можете думать о параметрах, как о наречиях (как в случае с **sayMoo**, где параметр говорит нам *как* выполнить **sayMoo**) или иногда, как о прямом дополнении (как в случае с **puts**, где параметр — это то, *что* выводится через **puts**).

Локальные переменные

В следующей программе имеется две переменные:

```
def doubleThis num
  numTimes2 = num*2
  puts num.to_s+' дважды будет '+numTimes2.to_s
end
doubleThis 44
```

```
44 дважды будет 88
```

Переменные — это **num** и **numTimes2**. Обе они расположены внутри метода **doubleThis**. Эти (и все другие переменные, которые вы видели до сих пор) являются локальными переменными. Это означает, что они "живут" внутри метода и недоступны снаружи. А если вы попытаетесь выполнить следующий код, вам будет выдана ошибка:

```
def doubleThis num
  numTimes2 = num*2
  puts num.to_s+' дважды будет '+numTimes2.to_s
end
doubleThis 44
puts numTimes2.to_s
```

```
44 дважды будет 88
#<NameError: undefined local variable or method `numTimes2' for
#<StringIO:0x82ba924>>
```

[#<Ошибка имени: неопределённая локальная переменная или метод `numTimes2' для #<StringIO: 0x82ba924>> — *Прим. перев.*]

Неопределённая локальная переменная... Фактически, мы *определили* эту локальную переменную, но она не является локальной там, где мы попытались её использовать; она локальная внутри метода.

Это может показаться неудобным, но на самом деле это очень даже приятно. Хотя это означает, что у вас нет доступа к переменным внутри методов, это также означает, что у них нет доступа к *вашим* переменным, и, таким образом, их нельзя

испортить:

```
def littlePest var
  var = nil
  puts 'ХА-ХА!  Я уничтожил твою переменную!'
end
var = 'Ты не можешь даже притронуться к моей переменной!'
littlePest var
puts var
```

```
ХА-ХА!  Я уничтожил твою переменную!
Ты не можешь даже притронуться к моей переменной!
```

Фактически, в этой маленькой программе *две* переменные **var**, а именно: та, что внутри метода **littlePest**, и та, что вне его. Когда мы вызвали **littlePest var**, мы в действительности просто передали строку из одной переменной **var** в другую так, что обе указывали на одну и ту же строку. Затем в методе **littlePest** его собственная *локальная* **var** стала указывать на **nil**, но это не повлияло на **var** вне метода.

Возвращаемые значения

Должно быть, вы заметили, что некоторые методы возвращают вам что-нибудь, когда вы их вызываете. Например, **gets** возвращает строку (ту строку, что вы ввели с клавиатуры), а метод **+** в выражении **5+3**, (а это на самом деле **5.+(3)**) возвращает **8**. Арифметические методы чисел возвращают числа, а арифметические методы строк возвращают строки.

Важно понять отличие между методами, возвращающими значение туда, где этот метод был вызван, и выводом вашей программой информации на экран монитора, как это делает **puts**. Заметьте, что **5+3** возвращает **8**; он **не** выводит 8.

А что же тогда *возвращает* **puts**? Мы никогда до этого не задумывались об этом, но давайте взглянем сейчас:

```
returnVal = puts 'Это вернул метод puts:'
puts returnVal
```

```
Это вернул метод puts:
nil
```

Итак, первый **puts** вернул **nil**. И хотя мы этого не проверяли, второй **puts** вернул то же; **puts** всегда возвращает **nil**. Каждый метод должен возвращать что-нибудь, даже если это просто **nil**.

Прервитесь ненадолго и напишите программу, чтобы выяснить, что же возвращает **sayMoo**.

Вы удивлены? Хорошо, вот как это всё работает: значение, возвращаемое из метода, — это просто значение последней строки метода. В случае с **sayMoo** это означает, что он возвращает **puts 'мууууууу...'*numberOfMoos**, то есть просто **nil**, поскольку **puts** всегда возвращает **nil**. Если бы мы хотели, чтобы все наши

методы возвращали строку **'жёлтая подводная лодка'**, нам бы просто нужно было поместить *это* в конце каждого из них:

```
def sayMoo numberOfMoos
  puts 'мууууууу...'*numberOfMoos
  'жёлтая подводная лодка'
end
x = sayMoo 2
puts x
```

```
мууууууу...мууууууу...
жёлтая подводная лодка
```

Итак, давайте снова вернёмся к нашему психологическому исследованию, но на этот раз мы напишем метод, который будет задавать для нас вопросы. Нужно, чтобы он принимал вопрос в качестве параметра и возвращал **true**, если ответ был `да`, или **false**, если ответ был `нет`. (Даже если в большинстве случаев мы просто игнорируем ответ, всё равно это неплохая идея, чтобы наш метод возвращал результат. Поступая так, мы сможем использовать его и для вопроса о ночном недержании.) Я также собираюсь сократить приветствие и пояснение просто для того, чтобы программу было легче читать:

```
def ask question  #  задать вопрос
  goodAnswer = false
  while (not goodAnswer)
    puts question
    reply = gets.chomp.downcase

    if (reply == 'да' or reply == 'нет')
      goodAnswer = true
      if reply == 'да'
        answer = true
      else
        answer = false
      end
    else
      puts 'Пожалуйста, отвечайте "да" или "нет".'
    end
  end

  answer  #  Это то, что мы возвращаем (true или false).
end
puts 'Здравствуйте! И спасибо, что Вы...'
puts
ask 'Вам нравится есть такос?' # Мы игнорируем возвращаемое значение.
ask 'Вам нравится есть бурритос?'
wetsBed = ask 'Вы мочитесь в постель?' # Мы сохраняем возвращаемое значение.
ask 'Вам нравится есть чимичангас?'
```

```
ask 'Вам нравится есть сопапиллас?'
ask 'Вам нравится есть тамалес?'
puts 'И ещё несколько вопросов...'
ask 'Вам нравится пить хорчата?'
ask 'Вам нравится есть флаутас?'
puts
puts 'ПОЯСНЕНИЕ:'
puts 'Спасибо за то...'
puts
puts wetsBed
```

```
Здравствуйте! И спасибо, что Вы...
Вам нравится есть такос?
да
Вам нравится есть бурритос?
да
Вы мочитесь в постель?
никогда!
Пожалуйста, отвечайте "да" или "нет".
Вы мочитесь в постель?
нет
Вам нравится есть чимичангас?
да
Вам нравится есть сопапиллас?
да
Вам нравится есть тамалес?
да
И ещё несколько вопросов...
Вам нравится пить хорчата?
да
Вам нравится есть флаутас?
да
ПОЯСНЕНИЕ:
Спасибо за то...
false
```

Неплохо, а? Мы смогли добавить больше вопросов (ведь добавлять вопросы теперь *легко*), но наша программа к тому же ещё и стала немного короче! Это значительное усовершенствование — мечта каждого ленивого программиста.

Ещё один большой пример

Думаю, было бы очень полезно привести ещё один пример метода. Мы назовём его **`englishNumber`**. Он будет принимать число, например, **`22`**, и возвращать его

английское название (в данном случае, строку **'twenty-two'**). Для начала, пусть он работает только с целыми числами от **0** до **100**.

> *(**ПРИМЕЧАНИЕ:** Этот метод использует новую хитрость для преждевременного возврата из метода с помощью ключевого слова **return**, а также представляет новый поворот в ветвлении: **elsif**. Из контекста должно быть ясно, как они работают.)*

```
def englishNumber number
  #  Нам нужны только числа в диапазоне 0-100.
  if number < 0
    return 'Пожалуйста, введите число ноль или больше.'
  end
  if number > 100
    return 'Пожалуйста, введите число 100 или меньше.'
  end

  numString = ''  #  Эту строку мы будем возвращать.

  #  "left" - сколько от числа нам ещё осталось вывести.
  #  "write" - часть числа, которую мы выводим сейчас.
  #  write и left... поняли?  :)
  left  = number
  write = left/100          #  Сколько сотен осталось вывести?
  left  = left - write*100  #  Вычтем эти сотни.

  if write > 0
    return 'one hundred'
  end

  write = left/10          #  Сколько десятков осталось вывести?
  left  = left - write*10  #  Вычтем эти десятки.

  if write > 0
    if write == 1  #  Охо-хо...
      #  Поскольку мы не можем вывести "tenty-two" вместо "twelve",
      #  нам нужно сделать особые исключения для них.
      if    left == 0
        numString = numString + 'ten'
      elsif left == 1
        numString = numString + 'eleven'
      elsif left == 2
        numString = numString + 'twelve'
      elsif left == 3
        numString = numString + 'thirteen'
```

```
    elsif left == 4
      numString = numString + 'fourteen'
    elsif left == 5
      numString = numString + 'fifteen'
    elsif left == 6
      numString = numString + 'sixteen'
    elsif left == 7
      numString = numString + 'seventeen'
    elsif left == 8
      numString = numString + 'eighteen'
    elsif left == 9
      numString = numString + 'nineteen'
    end
    #  Поскольку уже мы позаботились о цифре для единиц,
    #  нам не осталось ничего для вывода.
    left = 0
  elsif write == 2
    numString = numString + 'twenty'
  elsif write == 3
    numString = numString + 'thirty'
  elsif write == 4
    numString = numString + 'forty'
  elsif write == 5
    numString = numString + 'fifty'
  elsif write == 6
    numString = numString + 'sixty'
  elsif write == 7
    numString = numString + 'seventy'
  elsif write == 8
    numString = numString + 'eighty'
  elsif write == 9
    numString = numString + 'ninety'
  end

  if left > 0
    numString = numString + '-'
  end
end

write = left  #  Сколько единиц осталось вывести?
left  = 0     #  Вычтем эти единицы.

if write > 0
  if    write == 1
```

```
      numString = numString + 'one'
    elsif write == 2
      numString = numString + 'two'
    elsif write == 3
      numString = numString + 'three'
    elsif write == 4
      numString = numString + 'four'
    elsif write == 5
      numString = numString + 'five'
    elsif write == 6
      numString = numString + 'six'
    elsif write == 7
      numString = numString + 'seven'
    elsif write == 8
      numString = numString + 'eight'
    elsif write == 9
      numString = numString + 'nine'
    end
  end

  if numString == ''
    #  Только в одном случае "numString" может быть пустой -
    #  если "number" равно 0.
    return 'zero'
  end

  #  Если мы дошли досюда, то у нас имеется число где-то
  #  между 0 и 100, поэтому нам нужно вернуть "numString".
  numString
end
puts englishNumber(  0)
puts englishNumber(  9)
puts englishNumber( 10)
puts englishNumber( 11)
puts englishNumber( 17)
puts englishNumber( 32)
puts englishNumber( 88)
puts englishNumber( 99)
puts englishNumber(100)
```

```
zero
nine
ten
eleven
seventeen
```

```
thirty-two
eighty-eight
ninety-nine
one hundred
```

И всё-таки, определённо имеется несколько моментов в этой программе, которые мне не нравятся. Во-первых, в ней слишком много повторений. Во-вторых, она не обрабатывает числа больше 100. В-третьих, в ней слишком много особых случаев, слишком много возвратов по **return**. Давайте используем несколько массивов и попробуем её немного подчистить:

```
def englishNumber number
  if number < 0  #  Без отрицательных чисел.
    return 'Пожалуйста, введите неотрицательное число.'
  end
  if number == 0
    return 'zero'
  end

  #  Больше нет особых случаев!  Больше нет возвратов по return!

  numString = ''  #  Эту строку мы будем возвращать.

  #  единицы
 onesPlace = ['one',     'two',       'three',    'four',     'five',
               'six',     'seven',     'eight',    'nine']
  #  десятки
  tensPlace = ['ten',     'twenty',    'thirty',   'forty',    'fifty',
               'sixty',   'seventy',   'eighty',   'ninety']
  teenagers = ['eleven',  'twelve',    'thirteen', 'fourteen', 'fifteen',
               'sixteen', 'seventeen', 'eighteen', 'nineteen']

  #  "left" - сколько от числа нам ещё осталось вывести.
  #  "write"  - часть числа, которую мы выводим сейчас.
  #  write и left... поняли?  :)
  left  = number
  write = left/100          #  Сколько сотен осталось вывести?
  left  = left - write*100  #  Вычтем эти сотни.

  if write > 0
    #  Вот здесь действительно хитрый фокус:
    hundreds  = englishNumber write
    numString = numString + hundreds + ' hundred'
    #  Это называется "рекурсия".  Так что же я только что сделал?
    #  Я велел этому методу вызвать себя, но с параметром "write" вместо
    #  "number".  Помните, что "write" это (в настоящий момент) число
```

```
  #  сотен, которые нужно вывести.  Прибавив "hundreds" к "numString",
  #  мы добавляем после неё строку ' hundred'.  Так, например, если
  #  мы сначала вызвали englishNumber с 1999 (т.е. "number" = 1999),
  #  затем в этой точке "write" будет равен 19, а "left" равен 99.
  #  Наиболее лениво в этом месте было бы заставить englishNumber
  #  вывести нам 'nineteen', а затем мы выведем ' hundred',
  #  и потом оставшаяся часть englishNumber выведет 'ninety-nine'.

  if left > 0
    #  Так, мы не выводим 'two hundredfifty-one'...
    numString = numString + ' '
  end
end

write = left/10          #  Сколько десятков осталось вывести?
left  = left - write*10  #  Вычтем эти десятки.

if write > 0
  if ((write == 1) and (left > 0))
    #  Поскольку мы не можем вывести "tenty-two" вместо "twelve",
    #  нам нужно сделать для них особую обработку.
    numString = numString + teenagers[left-1]
    #  "-1" здесь потому, что teenagers[3] это 'fourteen', а не 'thirteen'.

    #  Поскольку уже мы позаботились о цифре для единиц,
    #  нам не осталось ничего для вывода.
    left = 0
  else
    numString = numString + tensPlace[write-1]
    #  "-1" потому, что tensPlace[3] это 'forty', а не 'thirty'.
  end

  if left > 0
    #  Так, мы не выводим 'sixtyfour'...
    numString = numString + '-'
  end
end

write = left  #  Сколько единиц осталось вывести?
left  = 0     #  Вычтем эти единицы.

if write > 0
  numString = numString + onesPlace[write-1]
  #  "-1" потому, что onesPlace[3] это 'four', а не 'three'.
```

```
  end

  #  А теперь мы просто возвращаем "numString"...
  numString
end
puts englishNumber(  0)
puts englishNumber(  9)
puts englishNumber( 10)
puts englishNumber( 11)
puts englishNumber( 17)
puts englishNumber( 32)
puts englishNumber( 88)
puts englishNumber( 99)
puts englishNumber(100)
puts englishNumber(101)
puts englishNumber(234)
puts englishNumber(3211)
puts englishNumber(999999)
puts englishNumber(1000000000000)
```

```
zero
nine
ten
eleven
seventeen
thirty-two
eighty-eight
ninety-nine
one hundred
one hundred one
two hundred thirty-four
thirty-two hundred eleven
ninety-nine hundred ninety-nine hundred ninety-nine
one hundred hundred hundred hundred hundred hundred
```

Aaaaa.... Это гораздо, гораздо лучше. Программа довольно компактная, вот почему я добавил в неё так много комментариев. Она даже работает с большими числами... хотя не совсем так хорошо, как можно было надеяться. Например, я полагаю, для последнего числа было бы гораздо лучше вернуть значение **'one trillion'**, или даже **'one million million'** (хотя все три значения правильные). В сущности, вы можете сделать это прямо сейчас...

Попробуйте ещё кое-что

• Доработайте **englishNumber**. Во-первых, добавьте тысячи. Так она должна возвращать **'one thousand'** вместо **'ten hundred'**, а также **'ten thousand'** вместо **'one hundred hundred'**.

• Ещё доработайте **englishNumber**. Теперь добавьте миллионы, чтобы вам возвращалось **'one million'** вместо **'one thousand thousand'**. Затем попробуйте добавить миллиарды и триллионы. Насколько далеко вы сможете зайти?

• А как насчёт **weddingNumber**? Она должна работать почти также, как **englishNumber**, но только она должна вставлять повсюду слово "and", возвращая что-то наподобие **'nineteen hundred and seventy and two'**, или как там это должно выглядеть в приглашениях на свадьбу? Я бы привёл вам больше примеров, но я сам не совсем это понимаю. Вам возможно, понадобится обратиться за помощью к устроителю свадеб.

• *"Девяносто девять бутылок пива..."* Используя **englishNumber** и вашу старую программу, напечатайте стихи этой песни, на этот раз *правильно*. Накажите ваш компьютер: пусть она начнётся с 9999. (Однако не задавайте слишком большое число, так как выводить всё это на экран займёт у компьютера достаточно много времени. Сто тысяч бутылок пива занимает приличное время; а если вы зададите миллион, вы накажете и себя тоже!

Мои поздравления! К этому моменту, вы стали настоящим программистом! Вы изучили всё, что нужно, чтобы составлять огромные программы с чистого листа. Если у вас есть идеи, какие бы программы вы сами хотели бы написать, попробуйте воплотить их!

Конечно, составлять всё с чистого листа может оказаться довольно медленным процессом. Зачем же тратить время на написание кода, который кто-то уже написал? Вы бы хотели, чтоб ваша программа оправляла электронную почту? Вы бы хотели сохранять и загружать файлы на свой компьютер? А как насчёт генерирования web-страниц для учебника, где примеры кода в самом деле выполняются каждый раз, когда загружается страница? ;) В Ruby есть много различных видов объектов, которые мы можем использовать и которые помогут нам писать программы лучше и быстрее.