6. Управление выполнением

Аааaа, управление выполнением... Вот где всё соединяется воедино. И хотя эта глава короче и легче, чем глава о методах, она откроет вам целый мир программных возможностей. После этой главы мы сможем писать по-настоящему интерактивные программы; до этого мы создавали программы, которые *выводили* разные вещи в зависимости от вашего ввода с клавиатуры, но после этой главы они также будут действительно *делать* разные вещи. Но прежде, чем мы сможем сделать это, нам нужно иметь возможность сравнивать объекты в наших программах. Нам нужны...

Методы сравнения

Давайте быстро пробежимся по этому разделу, чтобы можно было перейти к следующему разделу, **"Ветвления"**, где будут происходить самые классные вещи. Так, чтобы узнать, что один объект больше или меньше другого, мы применяем методы `>` и `<`, вот так:

```
puts 1 > 2
puts 1 < 2
```

```
false
true
```

Ничего сложного. Подобным же образом мы можем выяснить, что объект больше или равен другому (или меньше или равен) с помощью методов `>=` и `<=`.

```
puts 5 >= 5
puts 5 <= 4
```

```
true
false
```

И наконец, мы можем выяснить, равны ли два объекта или нет, используя `==` (что обозначает "они равны?") и `!=` (что обозначает "они различны?"). Важно не путать `=` с `==`. `=` используется, чтобы заставить переменную указывать на объект (присваивание), а `==` используется для ответа на вопрос: "Эти два объекта равны?"

```
puts 1 == 1
puts 2 != 1
```

```
true
true
```

Конечно, мы также можем сравнивать строки. Когда сравниваются строки, они сопоставляются в лексикографическом порядке, что обычно означает словарную упорядоченность. **`cat`** в словаре идёт перед **`dog`**, поэтому:

```
puts 'cat' < 'dog'
```

```
true
```

Однако, здесь таится ловушка: обычно в компьютерах устроено так, что

устанавливается такой порядок букв, в котором заглавные располагаются перед строчными буквами. (Так хранятся буквы в шрифтах, например: сначала все заглавные буквы, а за ними — строчные.) Это означает, что компьютер будет думать, что **'Zoo'** располагается перед **'ant'**, так что, если вы хотите выяснить, какое слово идёт первым в настоящем словаре, обязательно примените метод **downcase** (или **upcase** или **capitalize**) к обоим словам прежде, чем вы попытаетесь их сравнить.

Последнее замечание перед **"Ветвлением"**: методы сравнения не возвращают нам строки **'true'** и **'false'**; они возвращают нам специальные объекты **true** и **false**. (Конечно, **true.to_s** вернёт нам **'true'**, именно поэтому **puts** напечатал **'true'**.) **true** и **false** всегда используются для...

Ветвления

Ветвление — это простое, но очень мощное понятие. Фактически, оно настолько простое, что, ей-Богу, мне совсем даже и не нужно объяснять его; я просто покажу его вам:

```
puts 'Привет, как Вас зовут?'
name = gets.chomp
puts 'Привет, ' + name + '.'
if name == 'Chris'
  puts 'Какое милое имя!'
end
```

```
Привет, как Вас зовут?
```

```
Chris
```

```
Привет, Chris.
Какое милое имя!
```

Но если мы введём другое имя...

```
Привет, как Вас зовут?
```

```
Chewbacca
```

```
Привет, Chewbacca.
```

Вот это и есть ветвление. Если то, что находится после **if**, имеет значение **true**, мы выполняем код между **if** и **end**. Если то, что находится после **if**, имеет значение **false**, то не выполняем. Ясно и просто.

Я выделил отступом код между **if** и **end** просто потому, что, полагаю, таким образом легче отслеживать ветвление. Почти все программисты делают так, независимо от того, на каком языке они программируют. В этом простом примере может показаться, что от этого не слишком много пользы, но когда программа становится более сложной, это существенно меняет дело.

Часто нам бы хотелось, чтобы программа выполняла что-то, если выражение равно **true**, и нечто другое, если оно равно **false**. Для этого имеется **else**:

```
puts 'Я - предсказатель судьбы.  Скажите мне своё имя:'
```

```
name = gets.chomp
if name == 'Chris'
  puts 'Я предвижу у Вас в будущем великие дела.'
else
  puts 'Ваше будущее... О Боже!  Посмотрите-ка на часы!'
  puts 'На самом деле мне пора уходить, извините!'
end
```

```
Я предсказатель будущего.  Скажите мне своё имя:
```

```
Chris
```

```
Я предвижу у Вас в будущем великие дела.
```

А теперь давайте попробуем другое имя...

```
Я предсказатель будущего.  Скажите мне своё имя:
```

```
Ringo
```

```
Ваше будущее... О Боже! Посмотрите-ка на часы!
На самом деле мне пора уходить, извините!
```

Ветвление — это как будто мы подошли к развилке в коде: мы выберем дорожку к людям, чьё имя — Крис (**`name == 'Chris'`**), или же (**`else`**) мы выберем другой путь.

И совсем как ветви на дереве, у вас могут быть ветвления, которые сами содержат ветвления:

```
puts 'Здравствуйте и добро пожаловать в 7-й класс на урок английского.'
puts 'Меня зовут миссис Габбард.  А тебя зовут...?'
name = gets.chomp
if name == name.capitalize
  puts 'Садись, пожалуйста, ' + name + '.'
else
  puts name + '?  Ты имел в виду: ' + name.capitalize + ', не так ли?'
  puts 'Ты что, даже не знаешь, как пишется твоё имя??'
  reply = gets.chomp

  if reply == 'да'
    puts 'Хммм!  Ну хорошо, садись!'
  else
    puts 'УБИРАЙСЯ ВОН!!'
  end
end
```

```
Здравствуйте и добро пожаловать в 7-й класс на урок английского.
Меня зовут миссис Габбард.  А тебя зовут...?
```

```
chris
```

```
chris?  Ты имел в виду: Chris, не так ли?
Ты что, даже не знаешь, как пишется твоё имя??
```

```
да
```

```
Хммм!  Ну хорошо, садись!
```

Прекрасно, напишу его с заглавной буквы...

```
Здравствуйте и добро пожаловать в 7-й класс на урок английского.
Меня зовут миссис Габбард.  А тебя зовут...?
Chris
Садись, пожалуйста, Chris.
```

Иногда можно запутаться, пытаясь "вычислить", куда же ведут все эти **if**-ы, **else**-ы и **end**-ы. Я делаю так: пишу **end** *сразу же*, когда напишу **if**. Так, когда я писал приведённую выше программу, сначала она выглядела вот так:

```
puts 'Здравствуйте и добро пожаловать в 7-й класс на урок английского.'
puts 'Меня зовут миссис Габбард.  А тебя зовут...?'
name = gets.chomp
if name == name.capitalize
else
end
```

Затем я вставил комментарии — те вещи в коде, которые компьютер проигнорирует:

```
puts 'Здравствуйте и добро пожаловать в 7-й класс на урок английского.'
puts 'Меня зовут миссис Габбард.  А тебя зовут...?'
name = gets.chomp
if name == name.capitalize
  #  Она говорит вежливо.
else
  #  Она постепенно свирепеет.
end
```

Всё, что стоит после знака #, считается *комментарием* (конечно, если только он не находится внутри строки). После этого я заменил комментарии работающим кодом. Некоторым нравится оставлять комментарии; лично я думаю, что хорошо написанный код обычно говорит сам за себя. Раньше я применял больше комментариев, но чем более "бегло" я пишу на Ruby, тем меньше я их использую. На самом деле я нахожу их в большинстве случаев отвлекающими внимание. Но это каждый выбирает для себя сам; и у вас сложится свой собственный (обычно развивающийся) стиль. И вот, мой следующий шаг выглядел так:

```
puts 'Здравствуйте и добро пожаловать в 7-й класс на урок английского.'
puts 'Меня зовут миссис Габбард.  А тебя зовут...?'
name = gets.chomp
if name == name.capitalize
  puts 'Садись, пожалуйста, ' + name + '.'
else
  puts name + '?  Ты имел в виду: ' + name.capitalize + ', не так ли?'
  puts 'Ты что, даже не знаешь, как пишется твоё имя??'
  reply = gets.chomp

  if reply == 'да'
```

```
  else
  end
end
```

Снова я написал **if**, **else** и **end** одновременно. Это на самом деле помогает мне отслеживать, "где я нахожусь" в тексте программы. К тому же от этого задача кажется немного легче, поскольку я могу сфокусироваться на небольшой части программы — на вставке кода между **if** и **else**. Ести и другое преимущество от того, что я поступаю именно так. Оно состоит в том, что компьютер может понять программу на любом этапе разработки. Каждая из незаконченных версий программы, которые я показывал вам, будет выполняться. Они не были закончены, но это были работающие программы. Таким образом, я мог проверять её по ходу её написания, что помогало увидеть, как обстоят дела, и где она всё ещё нуждалась в доработке. Когда она прошла все проверки, я и выяснил, что я её закончил!

Эти подсказки помогут вам писать программы с ветвлением, но они также помогут освоить другой основной тип управления выполнением:

Циклы

Довольно часто вам может захотеться, чтобы ваш компьютер выполнял одно и то же снова и снова — в конце концов, предполагается, что именно этим компьютеры занимаются лучше всего.

Когда вы говорите вашему компьютеру повторять что-либо, вам также нужно сказать ему, когда остановиться. Компьютеру никогда не может наскучить, поэтому если вы не скажете ему остановиться, он и не остановится. Мы можем быть уверены, что подобное не случится, если скажем компьютеру повторять отдельные части программы пока (**while**) определённое условие является истинным. Это работает очень похоже на то, как работает **if**:

```
command = ''
while command != 'пока'
  puts command
  command = gets.chomp
end
puts 'Приходите ещё!'
```

```
Ау?
Ау?
Привет!
Привет!
Рад познакомиться.
Рад познакомиться.
О... как приятно!
О... как приятно!
пока
Приходите ещё!
```

Вот это и есть цикл. (Вы, наверное, заметили пустую строку в в начале вывода; она — от первого **puts**, перед первым **gets**. Как бы вы изменили программу, чтобы избавиться от этой пустой строки? Проверьте её! Она работает *в точности*, как программа выше, не считая первой пробельной строки?)

Циклы позволяют вам делать самые разные интересные вещи, какие только можно себе вообразить. Они, однако, могут также вызвать и неприятности, если вы сделаете ошибку. Что если ваш компьютер попадётся в ловушку бесконечного цикла? Если вы думаете, что именно так, возможно, и случилось, то просто, удерживая клавишу `Ctrl`, нажмите на клавишу `C`. [Это прервёт выполнение вашей программы. — *Прим. перев.*]

Но прежде, чем мы начнём забавляться с циклами, давайте изучим несколько вещей, которые облегчат нам жизнь.

Чуть-чуть логики

Давайте снова взглянем на нашу первую программу с ветвлением. Что если моя жена пришла домой, увидела эту программу, попробовала выполнить её, и она не сказала ей, какое милое имя *у неё*? Я не хотел бы обидеть её чувства (и спать на кушетке), так что давайте перепишем её:

```
puts 'Привет, как Вас зовут?'
name = gets.chomp
puts 'Привет, ' + name + '.'
if name == 'Chris'
  puts 'Какое милое имя!'
else
  if name == 'Katy'
    puts 'Какое милое имя!'
  end
end
```

```
Привет, как Вас зовут?
```

```
Katy
```

```
Привет, Katy.
Какое милое имя!
```

Ну да, она работает... но это не очень-то красивая программа. Почему не очень? Ну, потому, что лучшее правило, которое я когда-либо узнал в программировании, — это правило DRY (Don't Repeat Yourself), то есть "Не повторяйся!" Возможно. я бы мог написать небольшую книгу только о том, почему это такое хорошее правило. В нашем случае, мы повторили строку **puts 'Какое милое имя!'**. Почему это так важно? Ладно, а что если я сделал опечатку, когда переписывал её? Что если я хотел изменить с **'милое'** на **'прекрасное'** в обеих строках? Я же ленивый, помните? По сути, если я хочу, чтобы программа делала одно и то же, когда она получает **'Chris'** или **'Katy'**, тогда она должна действительно *делать одно и то же*:

```
puts 'Привет, как Вас зовут?'
```

```
name = gets.chomp
puts 'Привет, ' + name + '.'
if (name == 'Chris' or name == 'Katy')
  puts 'Какое милое имя!'
end
```

```
Привет, как Вас зовут?
```

```
Katy
```

```
Привет, Katy.
Какое милое имя!
```

Гораздо лучше. Чтобы заставить это заработать, я применил **or**. Другие *логические операции* это **and** и **not**. ["ИЛИ", "И" и "НЕ" — *Прим. перев.*] И когда работаешь с ними, лучшим решением будет всегда использовать скобки. Давайте посмотрм, как они работают:

```
iAmChris  = true
iAmPurple = false
iLikeFood = true
iEatRocks = false
puts (iAmChris  and iLikeFood)
puts (iLikeFood and iEatRocks)
puts (iAmPurple and iLikeFood)
puts (iAmPurple and iEatRocks)
puts
puts (iAmChris  or iLikeFood)
puts (iLikeFood or iEatRocks)
puts (iAmPurple or iLikeFood)
puts (iAmPurple or iEatRocks)
puts
puts (not iAmPurple)
puts (not iAmChris )
```

```
true
false
false
false
true
true
true
false
true
false
```

Единственная из них, которая может обмануть вас, — это операция **or**. В английском языке мы часто используем "or" в значении "один или другой, но не оба". Например, ваша мама могла бы сказать: "На десерт ты можешь съесть торт или пирожное". Она *не* предполагала, что вы можете съесть и то и другое! Компьютер, напротив, использует **or** в значении "один или другой или оба". (Другой способ выразить это: "по крайней мере, один из них — истинный".) Вот почему с компьютерами гораздо

веселее, чем с мамами.

Попробуйте ещё кое-что

• *"99 бутылок пива на стене..."* Напишите программу, которая печатает стихи этой излюбленной классической походной песни: "99 бутылок пива на стене".
• Напишите программу "Глухая бабуля". Что бы вы ни говорили бабуле (чтобы вы ни вводили с консоли), она должна отвечать: АСЬ?! ГОВОРИ ГРОМЧЕ, ВНУЧЕК!, если только вы не кричите ей (вводите слова заглавными буквами). Если вы кричите, она вас слышит (или по крайней мере думает, что слышит) и вопит в ответ: НЕТ, НИ РАЗУ С 1938 ГОДА! Чтобы сделать вашу программу *действительно* правдоподобной, пусть бабуля кричит каждый раз другой год; например, любой случайный год между 1930 и 1950. (Эта часть необязательная, и вам будет гораздо легче, если вы прочтёте раздел о генераторе случайных чисел в Ruby в конце главы о методах.) Вы не можете остановить разговор с бабулей, пока не прокричите `ПОКА`.

Подсказка: *Не забывайте о* **`chomp`***!* **`'ПОКА'`** *с символом Enter это не то же самое, что* **`'ПОКА'`** *без него!*
Подсказка 2: *Попробуйте обдумать, какие части вашей программы должны происходить снова и снова. Все они должны находиться внутри цикла* **`while`**.

• Улучшите вашу программу "Глухая бабуля": Что если бабуля не хочет, чтобы вы уходили? Когда вы кричите `ПОКА`, она может притвориться, что не слышит вас. Измените вашу предыдущую программу так, чтобы вам нужно было прокричать `ПОКА` три раза *в одной строке*. Удостоверьтесь в правильности вашей программы: если вы прокричите `ПОКА` три раза, но не в одной строке, вы должны дальше разговаривать с бабулей.
• Високосные годы. Напишите программу, которая будет спрашивать начальный год и конечный год, а затем выдавать с помощью **`puts`** все високосные годы между ними (и включая их, если они также високосные). Високосные годы — это годы, нацело делящиеся на 4 (как 1984 и 2004). Однако, годы, нацело делящиеся на 100, — *не* високосные (как 1800 и 1900) ***если только*** они не делятся нацело на 400 (как 1600 и 2000, которые действительно были високосными). *(Да, это всё довольно запутанно, но не настолько запутанно, как если бы июль был в середине зимы, что иногда случалось бы.)* [если бы не было високосных годов. — *Прим. перев.*]

Когда вы это закончите, сделайте перерыв! Вы уже многое изучили. Поздравляю! Вас удивляет, сколько разных вещей вы можете заставить делать компьютер? Ещё несколько глав, и вы сможете запрограммировать почти всё, что угодно. Серьёзно! Только взгляните на все эти штуки, что вы можете делать сейчас и которые не смогли бы сделать без циклов и ветвления.

А теперь давайте изучим новую разновидность объектов, которые отвечают за списки других объектов: массивы.