4. Собираем всё вместе

Мы рассмотрели несколько различных видов объектов (числа и буквы), мы создавали переменные, указывающие на них; а следующее, что мы хотим сделать — заставить их всех дружно работать вместе.

Мы уже знаем, что если мы хотим, чтобы программа напечатала 25, то следующий код ***не*** *будет* работать, поскольку нельзя складывать числа и строки:

```
var1 = 2
var2 = '5'
puts var1 + var2
```

Частично трудность заключается в том, что ваш компьютер не знает, пытаетесь ли вы получить 7 (**`2 + 5`**), или вы хотите получить 25 (**`'2' + '5'`**).

Прежде чем мы сможем сложить их вместе, нам нужно каким-то образом получить строковую версию значения **`var1`** или целочисленную версию значения **`var2`**.

Преобразования

Чтобы получить строковую версию объекта, мы просто записываем после него **`.to_s`**:

```
var1 = 2
var2 = '5'
puts var1.to_s + var2
```

```
25
```

Подобным же образом **`to_i`** возвращает целочисленную версию значения объекта, а **`to_f`** возвращает плавающую версию. Давайте взглянем на то, что эти три метода делают (и что *не* делают) чуть более пристально:

```
var1 = 2
var2 = '5'
puts var1.to_s + var2
puts var1 + var2.to_i
```

```
25
7
```

Обратите внимание, что даже после того, как мы получили строковую версию **`var1`**, вызвав **`to_s`**, переменная **`var1`** продолжает постоянно указывать на **`2`**, и никогда — на **`'2'`**. До тех пор, пока мы явно не переприсвоим значение **`var1`** (что требует применение знака **`=`**), она будет указывать на **`2`** до конца работы программы.

А теперь давайте испробуем несколько более интересных (и несколько просто-таки странных) преобразований:

```
puts '15'.to_f
puts '99.999'.to_f
puts '99.999'.to_i
puts ''
puts '5 - это моё любимое число!'.to_i
puts 'Кто вас спрашивал о 5 или о чём-нибудь подобном?'.to_i
puts 'Ваша мамочка.'.to_f
puts ''
puts 'строковое'.to_s
puts 3.to_i
```

```
15.0
99.999
99
5
0
0.0
строковое
3
```

Итак, это, возможно, вызвало у вас некоторое удивление. Первый пример весьма обычен и даёт в результате 15.0. После этого мы преобразовали строку **'99.999'** в число с плавающей точкой и в целое. Плавающее получилось таким, как мы и ожидали; целое было, как всегда, округлено.

Далее, у нас было несколько примеров, когда несколько ... *необычных* строк преобразовывались в числа. **to_i** игнорирует всё, начиная с первой конструкции, которая не распознана как число, и далее до конца строки. Таким образом, первая строка была преобразована в **5**, а остальные, поскольку они начинались с букв, были полностью проигнорированы, так что компьютер вывел только ноль.

Наконец, мы увидели, что последние два наших преобразования не изменили ровным счётом ничего — в точности, как мы и предполагали.

Другой взгляд на puts

Есть что-то странное в нашем любимом методе... Посмотрите-ка вот на это:

```
puts 20
puts 20.to_s
puts '20'
```

```
20
20
20
```

Почему эти три вызова puts выводят одно и то же? Ну, положим, последние два так и должны выводить, поскольку **20.to_s** *и есть* **'20'**. А как насчёт первого, с целым числом **20**? Собственно говоря, что же это значит: написать *целое число* 20? Когда вы пишете *2*, а затем *0* на листе бумаги, вы записываете строку, а не целое. *Число* 20 — это количество пальцев у меня на руках и ногах; это не *2* с последующим *0*.

Что ж, у нашего знакомого, **puts**, есть большой секрет: прежде, чем метод **puts** пытается вывести объект, он использует **to_s**, чтобы получить строковую версию этого объекта. Фактически, *s* в слове **puts** обозначает *string*; **puts** на самом деле значит put string ["вывести строку" — *Прим. перев.*].

Сейчас это может показаться не слишком впечатляющим, но в Ruby есть много, *много* разнообразных объектов (вы даже научитесь создавать собственные объекты!), и было бы неплохо узнать, что произойдёт, если вы попытаетесь вывести с помощью **puts** действительно причудливый объект, например, фотографию вашей бабушки, музыкальный файл или что-нибудь ещё. Но об этом позже...

А тем временем, у нас есть для вас ещё несколько методов, и они позволят нам писать разнообразные забавные программы...

Методы gets и chomp

Если **puts** обозначает put string, уверен, что вы догадаетесь, что обозначает **gets**. И так же, как **puts** всегда "выплёвывает" строки, **gets** считывает только строки. А откуда он берёт их?

От вас! Ну, по крайней мере, с вашей клавиатуры. Так как ваша клавиатура может производить только строки, всё это прекрасно работает. В действительности происходит вот что: **gets** просто сидит себе и считывает всё, что вы вводите, пока вы не нажмёте `Enter`. Давайте попробуем:

```
puts gets
```

```
Здесь есть эхо?
```

```
Здесь есть эхо?
```

Конечно же, что бы вы ни вводили с клавиатуры, будет просто выведено вам обратно. Выполните пример несколько раз и попробуйте вводить разные строки.

Теперь мы можем делать интерактивные программы! Например, эта, когда вы введёте своё имя, поприветствует вас:

```
puts 'Приветик, ну и как Вас зовут?'
name = gets
puts 'Вас зовут ' + name + '?  Какое прекрасное имя!'
puts 'Рад познакомиться с Вами, ' + name + '.  :)'
```

Ммм-да! Я только что выполнил её — я ввёл своё имя, и вот что получилось:

```
Приветик, ну и как Вас зовут?
```

```
Chris
```

```
Вас зовут Chris
?  Какое прекрасное имя!
Рад познакомиться с Вами, Chris
.  :)
```

Хммм... похоже, когда я ввел буквы "`K`", "`p`", "`и`", "`c`" и затем нажал на `Enter`, **gets** воспринял все буквы моего имени *и* символ `Enter`! К счастью, имеется метод как раз

для подобных случаев: **chomp**. Он убирает все символы Enter, которые "болтаются" в конце вашей строки. Давайте снова проверим эту программу, но на сей раз призовём на помощь **chomp**:

```
puts 'Приветик, ну и как Вас зовут?'
name = gets.chomp
puts 'Вас зовут ' + name + '?  Какое прекрасное имя!'
puts 'Рад познакомиться с Вами, ' + name + '.  :)'
```

```
Приветик, ну и как Вас зовут?
```

```
Chris
```

```
Вас зовут Chris?  Какое прекрасное имя!
Рад познакомиться с Вами, Chris.  :)
```

Гораздо лучше! Обратите внимание, что поскольку **name** указывает на **gets.chomp**, нам не требуется писать **name.chomp**; значение **name** уже было **chomp**-нуто.

Попробуйте ещё кое-что

• Напишите программу, которая спрашивает у человека имя, затем отчество, затем фамилию. В результате она должна поприветствовать человека, называя его полным именем.

• Напишите программу, которая спрашивает у человека его любимое число. Пусть ваша программа прибавит единицу к этому числу, а затем предложит результат в качестве *большего и лучшего* любимого числа. (Однако будьте при этом тактичными.)

После того, как вы закончите эти две программы (и любые другие, которые вы пожелаете попробовать), давайте изучим ещё несколько методов (и узнаем о них ещё что-нибудь).