Numbers

Chapter 1

Now that you've gotten everything setup, let's write a program! Open up your favorite text editor and type in the following:

```
puts 1 + 2
```

Save your program (yes, that's a program!) as calc.rb (the .rb is what we usually put at the end of programs written in Ruby). Now run your program by typing ruby calc.rb into your command line. It should have put a 3 on your screen. See, programming isn't so hard, now is it?

Introduction to puts

So what's going on in that program? I'm sure you can guess what the 1+2 does; our program is basically the same as:

```
puts 3
```

puts simply writes onto the screen whatever comes after it.

Integer and Float

In most programming languages (and Ruby is no exception) numbers without decimal points are called integers, and numbers with decimal points are usually called floating-point numbers, or more simply, floats.

Here are some integers:

```
5
-205
99999999999999999999999
0
```

And here are some floats:

```
54.321
0.001
-205.3884
0.0
```

In practice, most programs don't use floats; only integers. (After all, no one wants to look at 7.4 emails, or browse 1.8 webpages, or listen to 5.24 of their favorite songs...) Floats are used more for academic purposes (physics experiments and such) and for 3D graphics. Even most money programs use integers; they just keep track of the number of pennies!

Simple Arithmetic

So far, we've got all the makings of a simple calculator. (Calculators always use floats, so if you want your computer to act just like a calculator, you should also use floats.) For addition and subtraction, we use + and -, as we saw. For multiplication, we use *, and for division we use /. Most keyboards have these keys in the numeric keypad on the far right side. If you have a smaller keyboard or a laptop, though, you can just use Shift 8 and / (same key as the ? key). Let's try to expand our calc.rb program a little. Type in the following and then run it.

```
puts 1.0 + 2.0
puts 2.0 * 3.0
puts 5.0 - 8.0
puts 9.0 / 2.0
```

This is what the program returns:

```
3.0
6.0
-3.0
4.5
```

(The spaces in the program are not important; they just make the code easier to read.) Well, that wasn't too surprising. Now let's try it with integers:

```
puts 1 + 2
puts 2 * 3
puts 5 - 8
puts 9 / 2
```

Mostly the same, right?

```
3
6
-3
4
```

Uh... except for that last one! But when you do arithmetic with integers, you'll get integer answers. When your computer can't get the "right" answer, it always rounds down. (Of course, 4 is the right answer in integer arithmetic for 9/2; just maybe not the answer you were expecting.)

Perhaps you're wondering what integer division is good for. Well, let's say you're going to the movies, but you only have $9. Here in Portland, you can see a movie at the Bagdad for 2 bucks. How many movies can you see there? 9/2... 4 movies. 4.5 is definitely not the right answer in this case; they will not let you watch half of a movie, or let half of you in to see a whole movie... some things just aren't divisible.

So now experiment with some programs of your own! If you want to write more complex expressions, you can use parentheses. For example:

```
puts 5 * (12 - 8) + -15
puts 98 + (59872 / (13 * 8)) * -52
```

```
5
-29802
```

A Few Things to Try

Write a program which tells you:

how many hours are in a year? how many minutes are in a decade? how many seconds old are you? how many chocolates do you hope to eat in your life? Warning: This part of the program could take a while to compute! Here's a tougher question:

If I am 1245 million seconds old, how old am I? When you're done playing around with numbers, let's have a look at some letters.