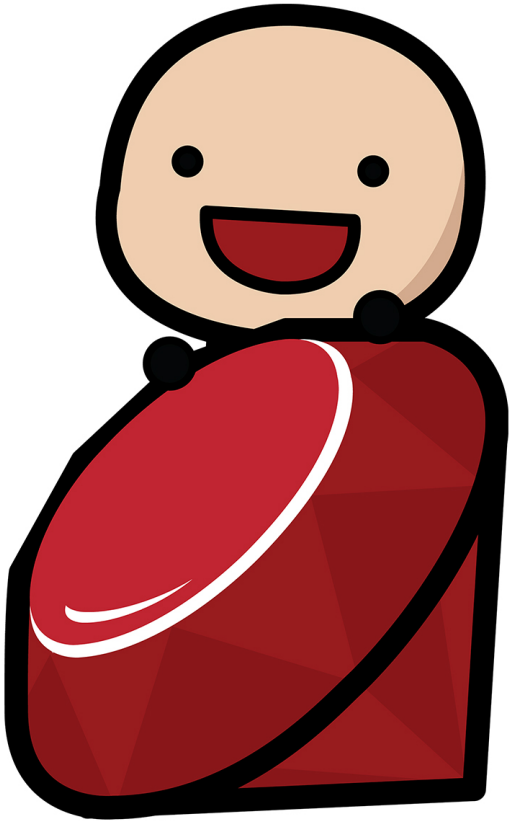
Ruby Kin

teach kids to code!



Ruby Kin

Teach Kids Ruby!

Doug Wright

This book is for sale at http://leanpub.com/rubykin

This version was published on 2014-04-15

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Doug Wright by spreading the word about this book on Twitter!

The suggested hashtag for this book is #rubykin.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#rubykin

To my wife with love

Contents

What is programming?

Programming is language. We use language to communicate with our friends. We use programming to communicate with our computers.

A program is nothing more than a set of instructions for the computer to execute. These instructions outline the plan or project, like a blueprint for a house. Without a blueprint, the builders wouldn't know what the architect wants, just like the computer doesn't know how to create what *you* want without a program.

Learning a programming language is a lot like learning any language. First you start with the basic building blocks–the A, B, C's. Then you start learning how to put those blocks into words, sentences, and eventually commands that will tell your computer what to do.

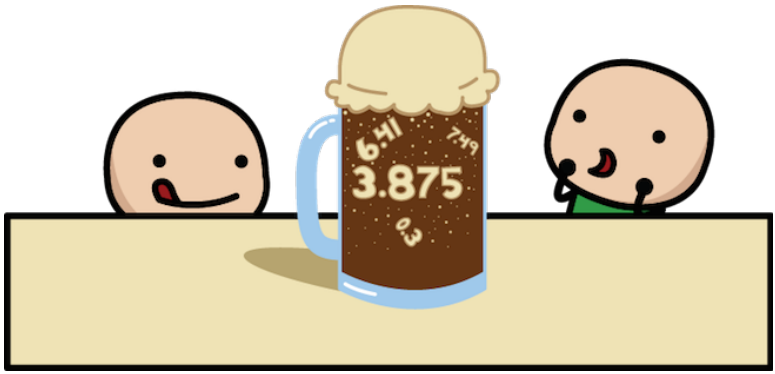
Why do we need to speak a strange language to communicate with computers?

Computers are a bit like dogs. They are great companions, but you need to give them commands to tell them what you want them to do. There are a lot of different programming languages, just like there are a lot of foreign languages. However, just as there are similarities in German, French or Spanish, there are similarities across programming languages, like JavaScript, Ruby or Python.

This book will help you learn the basics of programming, using the Ruby language. The building blocks you learn in this book will help you learn any other programming language.

Each chapter will go into more detail with basics at the beginning and more complicated material at the end. For now, let's get started with some simple ideas.

Integers, Floats and Strings...and maybe Ice Cream



Art by Vixuong Hong

Before we can understand how to write programs, we need to understand data. Data is simply information that you can input (give), output (get), store or manipulate with a computer. The two fundamental types of data used in almost every programming language are *numbers* and *strings*.

Numbers come in two tasty flavors. Integers, which are whole numbers without a decimal point, and floats, numbers that contain a decimal.

For example, these are integers:

```
1, 7, 0, 13, 2000
```

And these are floats:

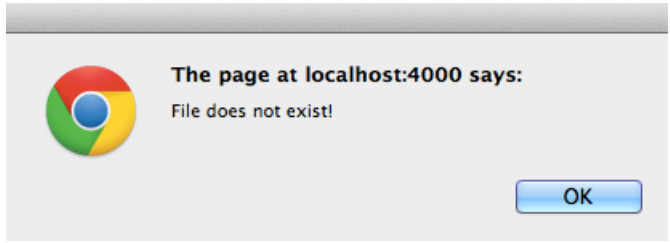
```
1.2, 3.14, 5.12345, 0.35
```

Every time you see a whole number like 8 or 19 you will know they are integers, and every time you see a number with a tasty sprinkle, or decimal, they will look like this: 3.4 or 8.1 and you will know they are (root beer) floats.

Simple, right? Ok, let's keep going.

The second type of data, or information given to a computer, are called *strings*. What's a string? "Anything between quotes is a string." Since that last sentence was inside of quotes, it was technically a string! You probably see strings all the time without even realizing it!

For example, have you ever seen an alert message on your computer saying something like this:



example alert message

Somewhere inside a program or web application, an engineer wrote the sentence "File does not exist!" and put it in quotes to create a string. When you were alerted with the pop up box, that string was printed to the screen.

Strings are a little bit like backpacks or lunch pails, they are great for storing all the stuff we care about in an easy-to-carry container. Except with a string, the straps are the quotes. We use strings to store words, sentences, and even files. Here are some examples of strings:

```
1 "I'm a string!"
2 "And_so_am_I"
3 "9"
4 "This long paragraph is even a string.\nAnd it has these
5 strange \n things that we'll explain later."
```

9 is such a joker. Did you notice we put the number nine as a string? This is very different than the actual number nine, but we will get to that later.

Now that you know the two most fundamental pieces of data the computer uses (numbers and strings), it's time to dive a bit deeper into each of these data types.

Numbers

Do you know basic math? Great, so does Ruby! Ruby uses the same adding, subtracting, multiplication and division that you do.

```
2 + 2 => 4
9 - 3 => 6
2 * 3 => 6
4 / 2 => 2
```

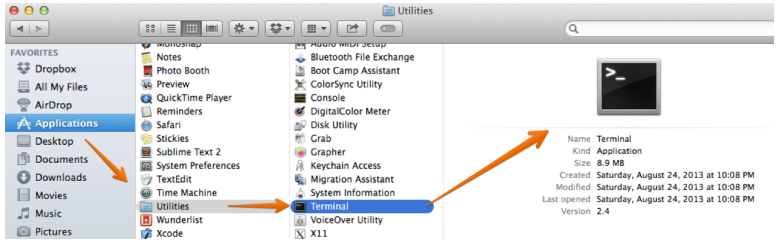
Ruby can also perform logical operations, such as greater than `>` and less than `<`.

```
4 > 2 => true
7 < 2 => false
3 >= 3 => true
0 <= 1 => true
```

You can try these simple commands yourself, but first you need to have Ruby installed on your computer.

Installing Ruby

If you're using Mac OS X, Ruby is already installed. Hooray! Just open your *Terminal* application. To find terminal you can click the magnifying glass in the top right corner of your screen, then type Terminal and click the first result. (Or you can navigate to Applications - Utilities - Terminal and double click).



to open Terminal on a Mac

If you're using Linux, open up a shell, type irb and hit enter. And if you're on Windows, open fxri from the Ruby section of your Start Menu.

As a third option, check out Repl.it (http://repl.it/languages/Ruby). This is a pretty fantastic tool that allows you to type code into your web browser. No installation required!

Next, type the letters `irb` into the terminal or shell screen and hit enter.

So, what is IRB anyway?

IRB stands for Interactive Ruby Shell. An Interactive Ruby Shell is like a little secret fort located inside your computer. You can use your IRB environment to play around with Ruby and learn different commands. Open up IRB (from a shell window or using www.repl.it/languages/Ruby and try typing some of these simple math calculations to see what the computer returns. Or try your own!

```
2 + 2
4 < 7
5 > 10
7 / 4
```

Curious about the => arrows? Ruby engineers like to call these hash rockets. You will see that typing `3 + 2` or any other thing into IRB will always return a value, signified by the pointer =>.

A bit more math

Programming languages like Ruby can perform a *lot* of mathematical equations and expressions. Now that we can see how Ruby performs addition, subtraction, multiplication and division, we'll see how she handles exponents and the oh-so-cool modulo!

Exponents

Exponents tell a number how many times it should be multiplied. For example, 2 times 2 equals 4, 4 times 2 equals 8. So, 2^3 means 2 times 2 times 2, or 8. Don't worry about exponents if they are unfamiliar to you. They are not necessary to learn programming! Ruby uses two stars to signify an exponent math expression:

```
2 ** 3 => 8
3 ** 2 => 9
10 ** 3 => 1000  #10 times 10 times 10!
# Also, the pound symbol is used for comments
# (and ignored by Ruby)
```

Modulo

In addition to these standard math operations, the computer has something called the modulo operator, which is represented using a percent symbol: %. The modulo's job is to find the remainder after dividing one number with another. The remainder is what is left over, or what remains when you divide one big number by a smaller number. Let's look at an example.

9 divided by 3 would result in a modulo of 0, because there is no remainder. Since 3 divides evenly into 9 (three times) there are no numbers left over, and that is why 9 modulo 3 is 0. If we try it again with a different set of numbers, say 9 modulo 2, we would have a remainder (or a modulo operator) of 1. Because 9 divided by 2 equals 8, leaving a remainder of 1. Try these examples

```
8 % 2 => 0
9 % 2 => 1
9 % 5 => 4
```

If exponents and modulos are too complex to understand right now, don't worry. Again, they are not a requirement for programming. It's important at this stage to simply understand that the computer can do simple math for you. Check out some examples below.

Practice

Try these problems in your head, or on paper. Then see how they work in the Ruby shell (IRB).

1) 2 + 3 + 5 2) 10 - 3 3) 9 / 3 4) 4 * 2 5) 4 ** 2

Now for some harder ones.

6) What is the result of 11 % 5 ?

7) What is 14 % 3 ?

8) A wizard carries two numbers (one even and one odd) in each hand. He won't open his hands to show you, but he will let you use modulo. Your task is to find out which hand holds an even number, and which holds the odd.

Hint: A number divided by two, with no remainder, is even.

Strings

Congratulations! You are halfway through learning the two most fundamental 'blocks' of programming code–Numbers and Strings.

Remember that a string is simply a piece of data (usually words) wrapped in quotes.

In Ruby, we can perform some math-like operations using our friends, the strings. For example, we can multiply a string like so:

```
1  "repeat " * 3
2  => "repeat repeat repeat "
```

In the above example we have a string: `"repeat "` that we multiply `*` by three. When Ruby performs the `* 3` method it actually just copies the string three times and pastes the result together. Sort of like this:

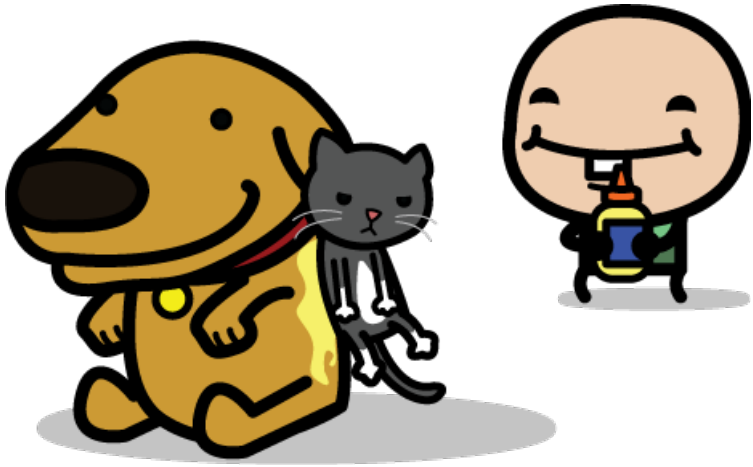
```
1  "repeat " + "repeat " + "repeat "
2  => "repeat repeat repeat "
```

In fact, you can write either of these lines of code and the result will be the same. Much in the same way that writing `3 + 3 + 3 = 9` yields the same result as `3 * 3 = 9`. When the computer adds strings together we call it concatenation.

```
1  "Cat and " + "Dog"
2  => "Cat and Dog"
```

In the example above, there are two strings, one is `"Cat and "` and the second string is `"Dog"`. By giving the command to add the two

strings +, we are able to join both strings together using the magic of concatenation. (Concatenation is just a fancy old Latin word for join together).



Art by Vixuong Hong

Ready to try it out?

Open Terminal again (or http://repl.it/languages/Ruby) and try multiplying and adding a few strings yourself to get the hang of it.

Now try adding a number with a string. It didn't work did it? Remember that joker `"9"` string from our previous example in chapter one? Ruby doesn't see this as the actual number 9, instead it sees a string.

To Ruby, anything that is inside the quote isn't just a word or a number anymore. Everything inside the quotes is a string. So when we try to add a string with a number, Ruby gives us an error:

```
1  "9" + 9
2  => TypeError: can't convert Fixnum into String
```

In the above example, we would see `"9" + 9` and think the answer is 18. But Ruby doesn't see it like that. Ruby sees `"STRING" + NUMBER`. And you can't add strings and numbers, because they are different *types* of data.

This doesn't mean we can't do this sort of equation, it just means we need to use a trick to make sure Ruby understands what we want. Of course, there are lots of interesting methods or *actions* we can perform to get Ruby to do what we want.

We'll look at some of those *actions* a bit later, but for now, you could solve the above problem by using the *to integer* method, or `to_i`. This would convert the string to a number (more specifically, an integer). And with two numbers, Ruby can do the math:

```
1  "9".to_i + 9
2  => 18
```

Remember those `\n` characters from the first chapter? This is what the computer uses to note a *new line* in your string. So the following string:

```
1  print "One line.\nAnother line.\nAnd another.\n"
```

Would print like this:

```
1  One line.
2  Another line.
3  And another.
```

In this way, Ruby can store sentences or even whole files inside just one string. Now that you have a better understanding of strings, check out some examples below.

Practice

What is the result of each of the following?

1) puts “What is the return result” + “ of “ + “this operation?”

2) “This string minus” - “That string”

3) `"1234.55".to_i`

4) `"1234.55".to_f`

5) `"Not a number".to_i`

6) `puts "1\n2\n3\n"`

7) Why might it be useful that the to_i (to integer) method return zero for strings that can’t be represented as numbers?

8) What do you think the length method does? `"Count".length`

9) How about the split method? `"Count".split("")`

10) What do you think slice does? `"Count".slice(2)`

Want to see more cool String methods? Just call *methods* on the String class! Type `String.methods` into your IRB:

```
IRB$ String.methods # some of the built-in Ruby methods for String
```

[:try_convert, :allocate, :new, :superclass, :freeze, :===, :==, :<⇒, :<, :<=, :>, :>=, :to_s, :included_modules, :include?, :name, :ancestors, :instance_methods, :public_instance_methods, :protected_instance_methods, :private_instance_methods, :constants, :const_get, :const_

and many more…