Introduction to Ruby

Mark Armstrong

October 9, 2020

Contents

1 Introduction

These notes were created for, and in some parts **during**, the lecture on October 9th and the following tutorials.

2 Motivation

So far, we have been investigating

- the functional paradigm, using Scala,
  - which happens to also be a pure object-oriented language, and
- the logical paradigm, using Prolog.

We now investigate an *imperative* pure object-oriented language, Ruby.

Ruby's syntax is also heavily influenced by Lisp, and the final language we will investigate later in the course is a Lisp. So Ruby acts as a stepping stone to that point.

3 Background on Ruby

Ruby is an almost purely object-oriented language, heavily inspired by Smalltalk, Lisp and Perl.

It's creator, Yukihiro "Matz" Matsumoto, has documented these inspirations in a post on ruby-talk.

> Ruby is a language designed in the following steps:
>
> - take a simple lisp language (like one prior to CL).
> - remove macros, s-expression.
> - add simple object system (much simpler than CLOS).
> - add blocks, inspired by higher order functions.
> - add methods found in Smalltalk.
> - add functionality found in Perl (in OO way).

Ruby was a language born out of Matz's interests and love of certain language features. He's said about its creation

> Well, Ruby was born on February 24 1993. I was talking with my colleague about the possibility of an object-oriented scripting language. I knew Perl (Perl4, not Perl5), but I didn't like it really, because it had smell of toy language (it still has). The object-oriented scripting language seemed very promising.
>
> I knew Python then. But I didn't like it, because I didn't think it was a true object-oriented language OO features appeared to be add-on to the language. As a language manic and OO fan for 15 years, I really wanted a genuine object-oriented, easy-to-use scripting language. I looked for, but couldn't find one.

So, I decided to make it. It took several months to make the interpreter run. I put it the features I love to have in my language, such as iterators, exception handling, garbage collection.

Then, I reorganized the features of Perl into a class library, and implemented them. I posted Ruby 0.95 to the Japanese domestic newsgroups in Dec. 1995.

4 "Purely object-oriented"

What does it mean when I have said, both about Ruby and Scala, that they are purely object oriented?

Quite simply, that all *data* is represented as an *object, and all operations are /methods*!

4.1 Integers are objects, operations are methods

For instance, consider integers, which in many languages are a *basic*, *builtin* type that do not have an implementation within the language. This is not the case in a pure language! In both Ruby and Scala, integers are objects, and operations on them are methods.

So code such as

```
5 + 2
```

```
5 + 2
```

could instead be written

```
5.+(2)
```

```
5.+(2)
```

that is, `+` is a method of the first argument, being passed the second argument as a parameter. The form `x` $\oplus$ `y` is just *syntactic sugar*.

4.2 Integers are objects, operations are *messages*

In Ruby, we can move one level of abstraction higher: all data are objects, and all operations are *messages between objects*. This harkens back to SmallTalk, one of the founding languages of the object oriented paradigm.

```
5.send("==", 3)
```

So even the form `x.⊕(y)` is syntactic sugar!

In Scala, moving to this message passing abstraction is not possible, aat least not easily; why? :TODO:

5 Postfix forms

:TODO:

6 Method naming conventions

By convention, methods ending with a `?` are predicates. They do not *necessarily* return a boolean, but should return a "truth value" of some kind. :TODO:

Methods ending with a `!` are *descructive*; they modify the receiver. :TODO:

Methods ending with a `=` indicate an *assignment* method. :TODO:

7 Defining classes

7.1 The basics

A class declaration in Ruby for a class named `Name` is begun by simply saying

```
class Name
```

Instance variables (whose value is unique per object of the class) begin with a `@`. We do not declare explicitely declare variables in Ruby, but you can initialise them to "declare" them (you don't need to though; they can be initialised inside a constructor or other methods.)

Class variables (whose value is shared by each object of the class) begin with a `@@`.

It is common to want to define *getters* and *setters* for instance variables in OO programming. For example,

```
class MyContainer

  def initialize(thing=nil) @thing = thing end

  def thing; @thing end
```

```
  # Assignment method syntax
  def thing=(thing) @thing = thing end
end

container = MyContainer.new()
container.thing = 5
puts(container.thing)
```

Because this is so common, there is a shorthand to avoid declaring these methods.

```
class MyContainer
  attr_accessor :thing   # :thing is a symbol; essentially in
  ↪  interned string
  # attr_reader   provides only the getter
  # attr_writer   provides only the setter
```

:TODO: initialize

7.2 Inheritance

:TODO:

```
class C1

class C2 < C1
```

7.3 TODO Mixins

:TODO: