

Class 03: Ruby (2)

Held: Friday, 28 January 2011

Summary: We deepen our understanding of Ruby. In particular, we consider blocks, arrays, hashes, and the basics of classes.

Related Pages:

- EBoard.
- Lab: Ruby (2).
- Reading: Tate 2.3.

Notes:

- Apologies for not getting the reading link in proper place today.
- Lessons from yesterday's session?
- Reading for Monday (respond Sunday): the rest of Tate 2.
- I am perennially behind. I expect to get next week's assignment posted by this evening.

Overview:

- Leftover topics from day 1.
- New ideas from 2.3.
- Lab.

Leftover Topics from Ruby, Day 1

- Ruby is designed to encourage different ways of thinking. For example, when someone asked me whether ranges provided methods that lists don't provide, I ended up typing something like

```
((1..9).methods - [1,2].methods).sort
```

- I guessed that - gave set difference
- I know that most modern scripting languages give sort methods
- While blocks are sort of objects in Ruby, they seem to require a slightly different syntax for variables
 - You can't write `x = {puts "Hello"}`
 - You can't write `{puts "Hello"}.call`
 - You can't use a function name where you would normally use a block.
 - Why would someone design a language in which blocks are treated differently?

New (and not so new) Ideas from Tate 2.3

We'll quickly go through some of the ideas introduced in Tate 2.3.

- Ruby provides arrays with a kitchen sink of methods, including methods that suggest a linked-list perspective and methods that suggest an array perspective.
 - If you support this many methods, it is hard for the client to know which ones are efficient or inefficient.
- Like most modern scripting languages, Ruby provides a built-in dictionary data type that they call a "hash"
- Ruby includes *symbols*
 - *Atomic* values - They cannot be decomposed into parts
 - One primary operation: Equality checking
 - Provide useful mnemonic constants
 - Are more expensive than strings to parse, but much cheaper to compare
 - Are nice to use in hashes
- Ruby uses *blocks* to encapsulate code
 - Can be passed to functions
 - Typically called with `yield`
 - Can also be called with `blockvar.call`
 - Not quite first-class; treated differently in many ways.
- Ruby supports (encourages? classes)
 - You define classes with

```
class Name
  ...
```
 - You can also define classes with

```
class Name < Superclass
  ...
```
 - You can extend classes by adding new methods.
 - Object attributes get preceded by an `@` to distinguish them from regular variables.
 - Class attributes get preceded by an `@@` to distinguish them from regular variables and object variables.
 - There is nice syntactic sugar to indicate that you want to provide accessors and setters.
- You can make chunks of code that can be included in multiple classes
 - Like inheritance, in that all the classes now implement an interface
 - Called modules in Ruby or mixins more generally

We'll also look at some of your questions.

- Please explain mixins
- Please explain the class hierarchy

Lab

- Do the lab.
- Be prepared to reflect.