More about Ruby

This module focussed on **object-oriented programming**, and we used Ruby as the vehicle language.

Here we look briefly at other aspects of Ruby: things you should be aware of.

Functional Programming

In **functional programming**, computation is seen as stateless function evaluation.

By contrast **imperative programming** relies on storing values in variables, i.e., maintaing program state.

In functional programming, the value returned by a function (method) depends solely on its arguments.

In imperative programming, a function may return different values depending on the program state when it is invoked.

Well-known functional programming languages include:
Lisp, Scheme, Erlang, Haskell, ...

Does Ruby support Functional Programming?

To an extent, yes.

We've already seen some of the functional elements of Ruby.

We saw how to use iterators like **map** and **inject** to apply a function (block) to a collection.

A block can be passed to a method to be invoked by yield.

Taking this further, methods/blocks can be made into true **first-class objects** using Ruby **procs** and **lambdas**.

(In a programming language, a **first-class object** is an entity that can be passed as a parameter, returned from a subroutine, or assigned into a variable.)

Functional Programming: take home message

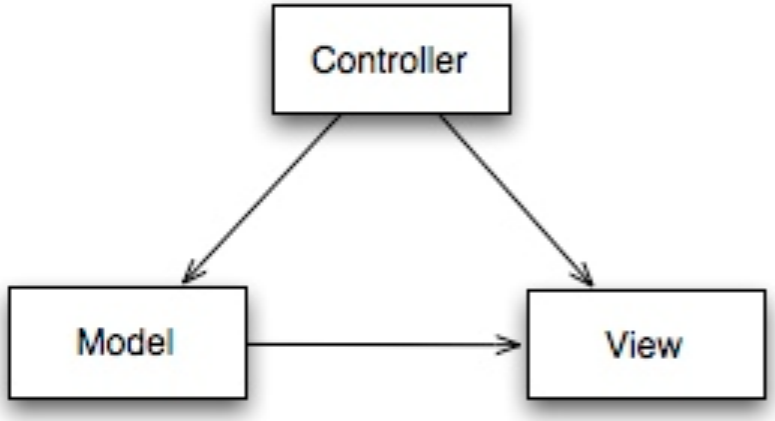
Ruby is not a functional language as such, but it supports functional programming quite well.

The functional aspects of Ruby blend well with the object-oriented aspects.

Ruby on Rails

Rails is a framework for developing web applications. It is **written in Ruby**, and is one of the reasons Ruby has become so popular.

Rails is based on the architectural pattern, **Model-View Controller** (MVC). This makes it easy to design the application well.



The Controller is repsonsible for the control flow of the application.

The Model contains business data and behaviour.

The View renders Models to the user, triggered by the controller.

Using Ruby on Rails

If you're new to web application development, there's no faster way to develop than using RoR.

Most people learn Ruby and Rails at the same time. It's a great advantage to have learned Ruby first.

Performance

"More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason - including blind stupidity."
-- William Wulf

"97% of the time: premature optimization is the root of all evil."
-- Donald Knuth

"The First Rule of Program Optimization: Don't do it. The Second Rule of Program Optimization (for experts only!): Don't do it yet."
-- Michael A. Jackson

Is performance in Ruby a problem?

If microseconds count, don't use Ruby, obviously.

Microseconds seldom count.

In general a compiled language (like Java) will outperform an interpreted language (like Ruby)...

...but not by an order of magnitude.

In most cases, the ease of development compensates for this lower run-time performance.

Twitter and Performance

The best-known Ruby on Rails application is Twitter.

As Twitter grew, performance issues unsurprisingly emerged.

These were resolved by reimplementing the backend in Scala, another dynamic language, but one that has lightweight static typing and runs on the JVM.

For front-end web development, Twitter still uses Rails.

Reflection

Reflection is where a program examines its own state and structure.

We saw some simple examples of reflection before. For example, when writing:

```
...
puts obj.class
```

this involves simple reflection.

Other simple reflexive methods available include (for variable **`obj`**):

```
obj.instance_of? c
obj.respond_to? m
```

Reflection

Ruby provides much more in terms of reflection

Some of things you can do include:

- add new methods to an object
- execute a string of Ruby code (`Kernel.eval`)
- iterate through all existing objects of class X
- see what objects variables are bound to
- add/remove instance variables to an object
- define actions to be performed when e.g. a class is subclassed or a new method is added to a class...
- etc.

Don't use these facilities in the normal course of programming!

However, when you need them, they're invaluable.

Regular Expressions

Ruby provides built-in support for **regular expressions**.

A regular expression is a concise way of performing pattern-matching on strings.

This is a big topic that has books devoted to it. Here are some examples:

`line1 =~ /Hello(.*)/`	true iff line1 starts with 'Hello'
`/[Rr]uby/`	Match "Ruby" or "ruby"
`/[aeiou]/`	Match any one lowercase vowel
`/\d{3}/`	Match exactly 3 digits

Regex for IPv6 addresses

Sample IPv6 addresses:
fe80:0000:0000:0000:0204:61ff:fe9d:f156
fe80:0:0:0:204:61ff:fe9d:f156 fe80::204:61ff:fe9d:f156
fe80:0000:0000:0000:0204:61ff:254.157.241.86
fe80:0:0:0:0204:61ff:254.157.241.86 fe80::204:61ff:254.157.241.86

... and a Ruby regex to match one (this is to convince you of the complexity of regexes):

```
/^\s*(([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]{1,4}|:))|(([0-9A-Fa-f]{1,4}:){6}
(:[0-9A-Fa-f]{1,4}|((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|
1\d\d|[1-9]?\d)){3})|:))|(([0-9A-Fa-f]{1,4}:){5}(((:[0-9A-Fa-f]{1,4})
{1,2})|:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?
\d)){3})|:))|(([0-9A-Fa-f]{1,4}:){4}(((:[0-9A-Fa-f]{1,4}){1,3})|((:[0-9A-
Fa-f]{1,4})?:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|
[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){3}(((:[0-9A-Fa-f]{1,4}){1,4})|((:
[0-9A-Fa-f]{1,4}){0,2}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2
[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){2}(((:[0-9A-Fa-f]
{1,4}){1,5})|((:[0-9A-Fa-f]{1,4}){0,3}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)
(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){1}(((:
[0-9A-Fa-f]{1,4}){1,6})|((:[0-9A-Fa-f]{1,4}){0,4}:((25[0-5]|2[0-4]\d|1\d\d|
[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(:(((:[0-9A-Fa-f]
{1,4}){1,7})|((:[0-9A-Fa-f]{1,4}){0,5}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)
(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:)))(%.+)?\s*$/
```

Regular Expressions: the take-home message

If you have to perform some pattern matching on strings, don't write the code yourself, look up

Writing complex regular expressions is a skill in its own right. If you need to do this, take the time to learn.

Dynamic Languages

The term **dynamic language** is used loosely to mean an interpreted language that does at runtime what is traditionally done at compile time

What does this involve? Some examples:
- type checking
- adding new methods to an object
- reflection
- higher-order functions
- ...

Well-known dynamic languages: Ruby, Python, Perl, PHP, Javascript, etc.

What if I miss static typing?!

Static typing, as used in Java and C++ prevents a whole family of run-time errors from occurring, e.g.,
- trying to multiply a network socket by a file manager
- trying to give a window a salary increase
- ... and so on

Isn't Ruby's dynamic typing inherently more risky?

In theory yes, in practice, it's not a commonly-reported problem.

Automated unit testing is a great help -- which is why it's so important in Ruby (and why this course emphasizes it).

There are great advantages to not having a burdensome, static typing system as well (less fingerwork, more flexible code).

The Future of Dynamic Languages

Dynamic languages have been around for a long time, but have come into industrial practice more in recent times.

They are still not taken as seriously as C++, C#, Java. You'll find plenty of Python in Google, probably not so much in the AIB IT department.

Programming language evolution has been in the direction of more dynamism, later binding, less emphasis on performance, so expect to hear more about dynamic languages.

In this context, **Scala** is an interesting new development. It's dynamic, but supports static typing in a non-intrusive way.

The future is definitely multilingual so don't get married to one language -- plan to be a polyglot.