Ruby.learn{ |stuff| }

- What is Ruby?
- What features make it interesting to me (and maybe you)?
- A quick, idiosyncratic tour of the Ruby ecosystem.
What is it?

- a dynamic, object-oriented, open source programming language...
- with a uniquely (but not too uniquely), expressive syntax
- dynamically or “duck” typed
- influenced by Perl, Smalltalk, Eiffel, Ada and Lisp
- has aspects of functional and imperative programming styles
History

- created by Yukihiro “Matz” Matsumoto on Feb 24 1993, released to public in 1995
- “Ruby” coined in comparison to “Perl”
Philosophy (the “Ruby Way”)

- emphasize programmer needs over computer needs
- encourage good design, good APIs
- Principle of Least Surprise (POLS)
  - *the language should behave in such a way as to minimize confusion for experienced users*
- “Everything should be a simple as possible, but no simpler.” - Einstein
- orthogonality
- change at runtime is not to be feared
Interesting Stuff...

- **Is Not** a Ruby tutorial
  - Programming Ruby: The Pragmatic Programmer's Guide
  - our just google it
  - you want something totally different?
    - [http://poignantguide.net/ruby/](http://poignantguide.net/ruby/)

- **Is** what I find interesting/different/well done about Ruby.
Everything's an Object

- no exceptions
Falsiness

- only false and nil are falsey. *Not 0.*

```ruby
puts "false is falsey" if !false
# => false is falsey

puts "nil is falsey" if !nil
# => nil is falsey

puts "0 is falsey" if !0
```
Symbols

- labels
- lightweight strings with no behaviour
- often used as hash keys

```ruby
name = {
  :first => 'David',
  :middle => 'Ian',
  :last => 'Gasson'
}

puts "#{name[:first]} #{name[:middle]} #{name[:last]}"
# => David Ian Gasson
```
Blocks

- another name for a Ruby block is a “closure”
- closure [kloh-zher] – noun. A function that is evaluated in an environment containing one or more bound variables.
Procs

- anonymous subroutines or closures with a life of their own

```ruby
def gen_times(factor)
  return Proc.new { |n| n*factor }
end

times3 = gen_times(3)
times5 = gen_times(5)

times3.call(12) #⇒ 36
	times5.call(5) #⇒ 25
	times3.call(times5.call(4)) #⇒ 60
```
Adding Methods to Classes

- You can add methods to an object at runtime in several ways...

```ruby
class Dog
  def eat
    puts "Yum!"
  end

  def sleep
    puts "Snore!"
  end

  def bark
    puts "Bow wow!"
  end
end
```
Adding Methods: Classes are Open

- you can “reopen” a class

```ruby
# Dogs wag their tails, too!
class Dog
  def wag_tail
    puts "Wag wag!"
  end
end

doggie = Dog.new
doggie.eat     # => Yum!
doggie.sleep   # => Snore!
doggie.bark    # => Bow wow!
doggie.wag_tail # => Wag wag!
```
Adding Methods: Mix-ins

- you can “mix-in” extra methods from modules

```ruby
# But wait, some dogs like to retrieve things.
module Retriever
  def fetch
    puts "Again!"
  end
end

class LabradorRetriever < Dog
  include Retriever
end

lab = LabradorRetriever.new
lab.respond_to?(:eat)   # => true
lab.respond_to?(:fetch) # => true
lab.fetch               # Again!
```
Adding Methods: Extending Instances

- you can mix-in methods *only* for a particular instance

```ruby
# But my lab is a bit weird, he likes to herd sheep, too,
# although that's atypical (work with me) for the breed.

module Herder
  def herd
    puts "Get moving!"
  end
end

lab.extend(Herder)
lab.respond_to?(:herd)           # => true
lab.herd                        # => Get moving!
```
Enumerable

- suppose I have a collection of objects
- it has an #each method that allows me to iterate through each member of a collection
- I can mix-in the Enumerable module and get a whole range of useful methods for free:
  - #find, #find_all, #detect, #reject, #grep
  - #map, #inject
  - #sort, if the shuttle operator (<=>) is defined
  - + more...
Enumerable: Example

class IntegerSequence
  include Enumerable

  attr_accessor :start_with, :end_with

  def initialize(end_with, start_with=1)
    self.end_with = end_with
    self.start_with = start_with
  end

  def each
    self.start_with.upto(self.end_with) do |i|
      yield(i)
    end
  end

  seq = IntegerSequence.new(10)
  puts seq.find_all { |i| i % 2 == 0 }

  # => 2
  # => 4
  # => 6
  # => 8
  # => 10
Regular Expressions

- supports Perl-like constructs
- all the usual Perl character classes and what not are supported

```ruby
# String matching
'abcdef' =~ /d/ # => 3 (i.e. d is the 4th letter)
'aaaaaa' =~ /d/ # => nil

# String substitution
"hello".gsub(/[aeiou]/, '*') # => "h*ll*
```
Attribute Accessors

- spend a lot of time writing trivial accessors (getters/setters)
- Ruby can help with the drudgery
Stuff I haven't mentioned (for no good reason)

- exception handling (raise, catch, throw)
- ranges
- a sweet switch statement
- threads/processes
- writing domain specific languages
- lots of other stuff...
The Wide Wide World...

- how I use Ruby regularly
- Ruby is in no way restricted by my imagination—this is not exhaustive!
- Rubyforge.org ([http://rubyforge.org/](http://rubyforge.org/)) is your friend
Read CSV Files: csv

- in core library
- slow on large files
- slightly awkward API for certain things
  - try FasterCSV (http://fastercsv.rubyforge.org/)

```ruby
require 'csv'
CSV.open('csvfile.csv', 'r') do |row|
  puts "Column 1: #{row[0]}, Column 2: #{row[1]}"
end
```
Access a Database

- huge range of choices
- I'll demonstrate just a few
- I normally use ActiveRecord because it plays nice w/ Rails
Access a Database: Sequel

- strengths
  - thread safety
  - connection pooling

```ruby
require 'rubygems'
require 'sequel'

DB = Sequel("postgres://postgres:postgres@localhost/geoinfo")
middle_east = DB[:countries].filter(:region => 'Middle East')

# Find the average GDP for the five biggest countries in the middle east region.
avg_gdp = middle_east.reverse_order(:area).limit(5).avg(:GDP)

# Print out the name of each middle east country.
middle_east.order(:name).each { |r| puts r[:name] }
```
Access a Database: ActiveRecord

- part of Rails, but can be used separately

```ruby
require 'rubygems'
require 'active_record'

ActiveRecord::Base.establish_connection({
  :adapter => "sqlite",
  :dbfile => "db/database.sqlite"
})

class Investigator < ActiveRecord::Base; end

class ScienceJustification < ActiveRecord::Base; end

class Proposal < ActiveRecord::Base
  has_many :investigators
  has_one :science_justification
end

proposal = Proposal.find(1)
proposal.investigators.each do |i|
  puts i.name
end

puts proposal.science_justification.body

Proposal.find(:all, :conditions => ['submitted > ?', DateTime.now])
```
Access a Database: DataMapper

- http://datamapper.org/
- fast
- new kid on the block
- borrows a lot of ActiveRecord syntax
- excellent alternative if you're not using Rails
Access a Database: DataMapper

```ruby
require 'rubygems'
require 'data_mapper'

DataMapper::Database.setup({
  :adapter => 'sqlite',
  :database => 'db/database.sqlite'
})

class Investigator < DataMapper::Base
  property :name, :string
end

class ScienceJustification < DataMapper::Base
  property :body
end

class Proposal < DataMapper::Base
  has_many :investigators
  has_one :science_justification
  property :submitted, :datetime
end

proposals = Proposal.all(submitted.gt => DateTime.now)
```
### Parse HTML: Hpricot

- [http://code.whytheluckystiff.net/hpricot/](http://code.whytheluckystiff.net/hpricot/)
- a flexible HTML parser
- access elements through a CSS selector-like syntax

```ruby
require 'rubygems'
require 'hpricot'
require 'open-uri'

# load the RedHanded home page
doc = Hpricot(open("http://redhanded.hobix.com/index.html"))

# change the CSS class on links
(doc/"span.entryPermalink").set("class", "newLinks")

# remove the sidebar
(doc/"#sidebar").remove

# print the altered HTML
puts doc
```
Parse XML: rexml

- in standard library
- nice API
- slow on large XML

```ruby
require 'rexml/document'

doc = REXML::Document.new(File.new('mydoc.xml'))

doc.root.each('//investigator').each do |investigator|
  puts investigator # => <investigator><name>David Gasson</name></investigator>
end
```
Parse XML: ruby-libxml

- wrapper around C library libxml2
- screaming fast even on large XML
- somewhat screwy namespace support
- slightly awkward API sometimes
Build an App: Rake

- [http://rake.rubyforge.org/](http://rake.rubyforge.org/)
- a simple ruby build program with capabilities similar to make or ant
Build an App: Rake
Automate Remote Tasks: Capistrano

- [http://www.capify.org/](http://www.capify.org/)
- tool for automating tasks on one or more remote servers
- not only good for Ruby/Rails tasks
Automate Remote Tasks: Capistrano

```bash
> cap search_libs -f Cap.rb
  * executing 'search_libs'
  * executing "ls -x1 /usr/lib | grep -i xml"
servers: ["www", "haedi"]

Password:

[www] executing command
  ** [out :: www] libbsdxml.a
  ** [out :: www] libbsdxml.so
  ** [out :: www] libbsdxml_p.a
[haedi] executing command
  ** [out :: haedi] TclxmlConfig.sh
  ** [out :: haedi] libxml2.2.dylib
  ** [out :: haedi] libxml2.dylib
  ** [out :: haedi] libxml2.la
  ** [out :: haedi] tclDOMxmlConfig.sh
  ** [out :: haedi] xml2Conf.sh
command finished
```
Remote Method Invocation

- many options
- outline the most common
Remote Method Invocation: soap4r

- http://dev.ctor.org/soap4r
- complicated, but relatively complete SOAP 1.1 implementation
- in standard library, but old. Use gem version.

```ruby
require 'rubygems'
gem 'soap4r'
require 'soap/wsd1Driver'

wsdl = 'http://nvogre.physt.pitt.edu:8080/axis/services/WesixTest?wsdl'
driver = SOAP::WSDLDriverFactory.new(wsdl).create_rpc_driver
driver.return_response_as_xml = true

vot = driver.wsextractorURL1VO('http://nvogre.physt.pitt.edu:8080/wesix/testr.fits', 0)
```
Remote Method Invocation: xmlrpc

- [http://www.fantasy-coders.de/ruby/xmlrpc4r/](http://www.fantasy-coders.de/ruby/xmlrpc4r/)
- in the standard library
- lightweight compared to soap4r
- nice and simple

```ruby
require 'xmlrpc/client'

# Make an object to represent the XML-RPC server.
server = XMLRPC::Client.new( "xmlrpc-c.sourceforge.net", "/api/sample.php")

# Call the remote server and get our result
result = server.call("sample.sumAndDifference", 5, 3)

sum = result["sum"]
difference = result["difference"]

puts "Sum: #\{sum\}, Difference: #\{difference\}"
```
Remote Method Invocation: DRb

- in the standard library
Aspect Oriented Programming: Aquarium

- [http://aquarium.rubyforge.org/](http://aquarium.rubyforge.org/)

```ruby
require 'rubygems'
require 'aquarium'

# trace all invocations of the public, instance methods in all classes
# whose names end with "Service"
class ServiceTracker
  include Aquarium::Aspects::DSL::AspectsDSL

  around(:calls_to => :all_methods, :in_types => /Service$/) do |join_point, object, *args|
    log "Entering: #{join_point.target_type.name}#{join_point.method_name}: object=#{object}, args=#{args.inspect}"
    result = join_point.proceed
    log "Leaving: #{join_point.target_type.name}#{join_point.method_name}: object=#{object}, args=#{args.inspect}"
    result # block needs to return the result of the "proceed"
  end
end
```
Manipulate Images: RMagick

- wrapper around ImageMagick
  - [http://www.imagemagick.org/](http://www.imagemagick.org/)
- therefore requires ImageMagick to be installed

```ruby
require 'rubygems'
require 'RMagick'

clown = Magick::ImageList.new("clown.jpg")
face = clown.crop(50, 15, 150, 165)

white_bg = Magick::Image.new(clown.columns, clown.rows)
clown = white_bg.composite(face, 50, 15, Magick::OverCompositeOp)

clown.write('crop.jpg')
```
Manipulate Images: RMagick
Testing: Test::Unit

- in core library
- typical unit testing framework

```ruby
require 'test/unit'
require 'voruby/resolver/sesame'
include VORuby::Resolver

class SesameTest < Test::Unit::TestCase
  assert_nothing_raised do
    sesame = Sesame.resolve('m51', :all)
    assert_kind_of Sesame, sesame

    resolver = sesame.resolvers.find { |r| r.name == 'Simbad' }
    assert_equal 'M 51', resolver.oname
  end
end

# Then just run the file. Any other subclasses of
# Test::Unit::TestCase will also be run.
```
Testing: RSpec

- http://rspec.info/
- Behaviour Driven Development framework
- includes:
  - story framework
  - spec framework
  - mock object support
Testing: RSpec

- Story Framework

```ruby
# Story: transfer from savings to checking account
# As a savings account holder
# I want to transfer money from my savings account to my checking account
# So that I can get cash easily from an ATM
#
# Scenario: savings account has sufficient funds
# Given my savings account balance is $100
# And my checking account balance is $10
# When I transfer $20 from savings to checking
# Then my savings account balance should be $80
# And my checking account balance should be $30

steps_for(:accounts) do
  Given("my $account_type account balance is $amount") do |account_type, amount|
    create_account(account_type, amount)
  end
  When("I transfer $amount from $source_account to $target_account") do |amount, source_account, target_account|
    get_account(source_account).transfer(amount).to(get_account(target_account))
  end
  Then("my $account_type account balance should be $amount") do |account_type, amount|
    get_account(account_type).should have_a_balance_of(amount)
  end
end
```
Testing: RSpec

- Spec Framework

```ruby
# bowling.rb
#
# Just enough code to pass.
#
class Bowling
  def hit(pins)
    end
  
  def score
    0
  end
end

# bowling_spec.rb
require 'bowling'

describe Bowling do
  before(:each) do
    @bowling = Bowling.new
  end

  it "should score 0 for gutter game" do
    20.times { @bowling.hit(0) }
    @bowling.score.should == 0
  end
end

# on the command line
#
> spec bowling_spec.rb --format specdoc
# Bowling
# - should score 0 for gutter game
#
# Finished in 0.007534 seconds
# #
# 1 example, 0 failures
```
Numerical Computing: Ruby/GSL

- http://rb-gsl.rubyforge.org/
- wraps the GNU Scientific Library
- vectors, matrices, linear algebra, fourier transforms, wavelet transforms, etc. etc.
Query the VO: VORuby

- http://rubyforge.org/projects/voruby/
- use voruby2 (currently in “preview”)

```ruby
require 'rubygems'
require 'voruby/votable/votable'
require 'voruby/wesix/wesix'
include VORuby

# Query WESIX
wesix = Wesix::Service.new
votable = wesix.extract('http://nvogre.physt.pitt.edu:8080/wesix/testr.fits')

# Get the first resource and table.
table = votable.resources.first.tables.first

# See the name of each column
puts table.fields.collect{|field| field.name}.join(' | ')

# --

# Query the Sesame name resolver
ra, dec = Sesame.resolve_position('m51')
```
Read FITS Files

- http://rubyforge.org/projects/rfits
- wrapper around cfitsio
Wrap Java: JRuby

- [http://jruby.codehaus.org/](http://jruby.codehaus.org/)
- alternate implementation of Ruby 1.8.6 under the JVM

```ruby
require 'java'
include_class 'java.util.TreeSet'

set = TreeSet.new
set.add "foo"
set.add "Bar"
set.add "baz"

set.each do |v|
  puts "value: #{v}" end
```
Wrap Java: JRuby

```ruby
require 'java'
include_class 'cds.savot.model.SavotSet'

# SavotSet is the base for Savot's array-like objects.
# It behaves similarly to a native Ruby array, but not
# exactly. With the simple addition of an #each
# method, it can become Enumerable in the Ruby sense.
class SavotSet
  include Enumerable

  # required for Enumerable
  def each
    (0..self.getItemCount()).each do |i|
      yield self.getItemAt(i)
    end
  end

  # map some of the existing constructs into more
  # ruby-like semantics
  def []=(i); self.getItemAt(i) end
  def <<(item); self.addItem(item) end
  def size; self.getItemCount() end
  def delete_at(i); self.removeItemAt(i) end
  def clear; self.removeAllItems() end
end
```
Wrap Java: JRuby

- a simple Savot Java program
- not bad, pretty compact
- all those loops are bit distracting, though
Wrap Java: JRuby

- equivalent in Ruby makes the domain more obvious
Web Applications: Ruby on Rails

- full-stack framework for developing database-backed web applications according to the Model-View-Control pattern
- has its own extensive ecosphere
- http://www.rubyonrails.org/
Web Applications: Ruby on Rails

- philosophy
  - *convention over configuration*
    - eschew complicated XML configuration files
  - *don't repeat yourself (DRY)*
  - *less software*
    - write less code
- easiest thing is just to show you an example...
Web Applications: Other

- Merb
  - http://www.merbivore.com/
- Nitro
  - http://www.nitroproject.org/
- Camping
  - http://code.whytheluckystiff.net/camping/
Resources

- About the language:
  - http://www.ruby-lang.org/ (canonical)
  - http://www.ruby-doc.org/ (documentation)

- Libraries/plugins/applications:
  - http://rubyforge.org/
  - http://raa.ruby-lang.org/