

Ruby

Ruby

A Simple Lesson

What to expect

- A simple overview of the language
- A review of some of the cool features of Ruby
- Lots of examples
- These slides, as well as all code is available right now at <http://fadeover.org/ruby>

What NOT to expect

- Rails
- The ability to program really well in Ruby after this lecture
- A thorough review of ALL of Ruby's features... there are too many.

Hello World in Ruby

Hello World in Ruby

```
puts "Hello, World!"
```

Hello World in Ruby

```
puts "Hello, World!"
```

...That's it.

Ruby is (kind of) Like C

- You can define and set variables:
 - ▶ `css_is_pretty_cool = true`
 - ▶ `my_pi = 3.1415`
 - ▶ `my_string = "Ruby is cool!"`
 - ▶ `cool_stuff = ["Macs", 1337, "UWindsor"]`

Things to Notice

- No semicolons! Hooray!
- Dynamic typing - Ruby automagically figures out types.
- Heterogeneous arrays - Arrays can hold objects of different types.

Ruby is (kind of) Like C

- You can define functions:

```
▶ def doThatThing(param1)
    puts "I'm doing that thing to " + param1
end
```

Ruby is NOT Like C

- Unlike C, Ruby is completely object oriented. Everything, including strings, integers, booleans, and even nil have methods which you can call or send messages to.
- The syntax for message sending is a . (period)
- Comment lines with #

Ruby is NOT Like C

- ▶ `-5.abs`
 - ▶ `5`
- ▶ `"HELLO, World!".downcase`
 - ▶ `"hello world!"`
- ▶ `nil.nil?`
 - ▶ `true`

Ruby is NOT Like C

- Single pass interpreted language (“Scripting Language”)
- No need for compilation (“Nuclear war can ruin your whole compile. – Karl Lehenbauer”)
- Run apps with `ruby myapp.rb [arguments]`
- Or `chmod +x myapp.rb` and `./myapp.rb`

IRB is Your Friend

- irb (“Interpreted Ruby”) is a command line tool you can use to quickly test things and play around with the language.
- It comes with Ruby (<http://ruby-lang.org/>)
- If you plan on learning Ruby, **you want this.**

Quick IRB Demo

Blocks

- The ability to pass code itself as an argument
- Used everywhere in Ruby
- Can be given arguments
- Start and end with `do` and `end` or `{` and `}`

Blocks

Basic block structure

```
some_object.blockMethod do |blockArg1, blockArg2|  
  # This is where you do stuff with  
  # blockArg1 and blockArg2, if anything  
end
```

Simple Examples

```
def sayIntroduction
  puts "Hi, my name is Patrick."
end

3.times { sayIntroduction }
```

Output

```
Hi, my name is Patrick.
```

```
Hi, my name is Patrick.
```

```
Hi, my name is Patrick.
```

```
my_favourite_things = ["macbook",  
                        "digital SLR",  
                        "CSS"]
```

```
my_favourite_things.each do |thing|  
  puts "I love my " + thing + "!"  
end
```

Output

I love my macbook!

I love my digital SLR!

I love my CSS!

Blocks

Writing a method that uses blocks

```
def doTenTimes  
  10.times do { yield }  
end
```

Ruby as a Language

- Messages are used almost everywhere
- `2 + 2` is actually `2 .+(2)`
 - `+` is a method of the `fixnum` class, and the other number is the argument
- Ruby allows flexible (extended character set) method names:
 - `my_object.Heylo?!`

Creating Objects in Ruby

- Define classes with `class ... end`
- Define methods with `def ... end`
- `@` symbol specifies an instance variable, `@@` specifies a class variable, and `$` specifies a global variable
 - `@first_name`

Regular Expressions in Ruby

- Ruby has great regex support
- Create a regular expression object simply by enclosing text within forward slashes:
 - `/(P|p)atr1ck)/`
- Use the match operator `=~` to search within strings using regexs

A Real Example

ROT 13

- Simple Caesar cipher
- Rotates letters by 13 places (e.g. 'a' → 'n')
- The “Hold upside down to see answer” of USENET and web forums
- Totally symmetric – can be used to both encode and decode

```
def swapLetter(byte)
  if /[A-M]|[a-m]/ =~ byte.chr then byte += 13
  else byte -= 13 end
end

ARGV.shift.each_byte do |byte|
  byte = swapLetter(byte) if /[A-Z]|[a-z]/ =~ byte.chr
  print byte.chr.to_s
end
print "\n"
```

Hey, why do we have to learn so much Java?
Ruby seems like the way to go!

Url, jul qb jr unir gb yrnea fb zhpu Wnin?
Ehol vf frrzf yvyr gur jnl gb tb!

Hey, why do we have to learn so much Java?
Ruby seems like the way to go!

Files in Ruby

- Treated like objects, like everything else in Ruby
- Have basic Read/Write methods
- Have extremely convenient block methods, such as `each_byte` and `each_line`.

MyGrep.rb

- A simple app that does the work of the well known grep unix command line tool
- Written in Ruby in *4 lines of code, in less than 10 minutes*

MyGrep.rb

```
search_term = Regexp.new(ARGV.shift)
file = ARGV.shift
if file.nil? then file = STDIN else file = open(file) end
file.each_line { |line| puts line if line[search_term] }
```

Demo of MyGrep

Cool things you can do with Ruby

- RubyCocoa
- Allows you to use the Mac OS X API with ruby
- Many more bindings and libraries like this exist

A Real Example

rssGrowler, an RSS update notifier

```
require 'open-uri'
require 'osx/cocoa'
require 'growl'
require 'rubygems'
require 'simple-rss'

# Change these:
Site = 'http://rss.cbc.ca/topstoriesnews.xml'
Site_root = 'http://www.cbc.ca/'

favicon_string = OSX::NSString.stringWithString(Site_root +
'favicon.ico')
icon = OSX::NSImage.alloc.initWithContentsOfURL
(OSX::NSURL.URLWithString(favicon_string))
growl = GrowlNotifier.new('GRSS', ['rss'], nil, icon)
growl.register()
oldrss = nil; rss = nil
loop do
  oldrss = rss
  rss_source = open(Site)
  rss = SimpleRSS.parse(rss_source, false)
  growl.notify('rss', rss.channel.title.to_s, rss.items
[0].title.to_s) unless (oldrss == nil or oldrss.items == rss.items)
  sleep(300)
end
```

rssGrowler Demo

Ruby's speed

- MyGrep is fast, but what other applications?
- A test: prime number calculation

Primes Demo

C vs. Ruby

Advanced Ruby

- Introspection/Reflection
- Meta-programming (eval and send)
- Language Extension

Great things about Ruby

- Programming in Ruby is FUN
 - Think less about the constructs of the language, more about the problems you are actually trying to solve

Great things about Ruby

- Intelligent use of regular expressions
 - Ruby has great built-in regex support, including the `=~` operator ("match")
- Lack of function 'return's
 - Ruby functions/methods automatically returns the last accessed value

Great things about Ruby

- Late 'if' conditionals
 - Improves readability with one-liners
- 'each...' iterators
 - Ruby programming style stresses iteration instead of loops, as iterators eliminate "off-by-one" errors as well as improve readability and reduce memory cost

Great things about Ruby

- Easy file reading and writing
 - As natural as using strings
- Readability: code is easily understood, some even by non-programmers.
- Introspection and extension

Cool things you can do with Ruby

- Ruby TK
- Create GUI apps using in ruby
- Hello World:

```
require 'tk'
root = TkRoot.new { title "Ex1" }
TkLabel.new(root) do
  text 'Hello, World!'
  pack { padx 55 ; pady 55; side 'left' }
end
Tk.mainloop
```

Cool things you can do with Ruby

- RAA - Ruby Application Archive: <http://raa.ruby-lang.org/>
- RubyForge: <http://rubyforge.org/>
- Thousands of libraries and projects
- Artificial neural network, linguistic, gmail, and graphing libraries, just to name a few

Where to go from here

- <http://ruby-lang.org/> - GET RUBY
- <http://ruby-doc.org/> - Tutorials, Articles, Screencasts, and more.
- <http://ruby-doc.org/core/> - Ruby's core documentation. Info on every built in class, method, function, constants, etc.

A few last things...

- Again, these slides, as well as all programs and extra links are available at:

<http://fadeover.org/ruby>

- There are a lot of things I haven't had time to talk about, so check out the links and have fun!

Thanks!

~

Q and maybe A