

Intro to Ruby



Aaron Bartell
abartell@kregeltech.com



Ruby... a dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.

Matz desired a language which he himself enjoyed using, by minimizing programmer work and possible confusion - enter Ruby.

- Est 1995 by **Yukihiro "Matz" Matsumoto**
- Mass acceptance 2006
- Very active and well organized community
- **October 2013, on IBM i with PowerRuby**



Features:

- variable declarations are unnecessary
- variables are dynamically and strongly typed
- syntax is simple and consistent
- everything is an object
- classes, methods, inheritance, etc.
- **NO SEMI COLONS!!!**

ruby-lang.org – Home website

ruby-doc.org - Formal documentation

codecademy.com - Learn Ruby in the browser for free

amzn.to/1apcrse - Metaprogramming Ruby: Program Like the Ruby Pros

Where can I use Ruby?

Web apps with Rails - rubyonrails.org

iOS with RubyMotion - rubymotion.com

Android with Ruboto - ruboto.org

Desktop (Mac, Linux, Windows) with Shoes - shoesrb.com



ShoesRB.com



RubyMotion



irb

irb (Interactive Ruby) is an interactive programming environment for Ruby that allows you to quickly test various coding ideas.

- Included with Ruby distribution.
- Symbolic link in `/usr/bin` exists for the `irb` binary
- **Great for learning Ruby** through quick tests vs. editing files, saving, and invoking.
- `irb` is the foundation for the `rails console`.
- `nil` in screenshot is the reality that every Ruby method (i.e. `puts`) returns a value.

```
2. ssh aaron@192.168.168.76 (ssh)
→ ~ ssh aaron@192.168.168.76
aaron@192.168.168.76's password:
-bash-4.2$ irb
irb(main):001:0> puts "hello"
hello
=> nil
irb(main):002:0> my_name = "Aaron"
=> "Aaron"
irb(main):003:0> puts "hello #{my_name}"
hello Aaron
=> nil
irb(main):004:0> puts "hello #{my_name}".length
11
=> nil
irb(main):005:0> 3 + 7
=> 10
irb(main):006:0> |
```

ruby-doc.org/stdlib-2.0/libdoc/irb/rdoc/IRB.html - Formal documentation

tryruby.org – Ruby code in the browser without having to install anything.

railscasts.com/episodes/48-console-tricks - Console tricks

stackoverflow.com/questions/123494/whats-your-favourite-irb-trick - Favorite irb tricks from community

Invoke Ruby program

Syntax:

```
ruby /ifs/folder/path/<program_name>.rb
```

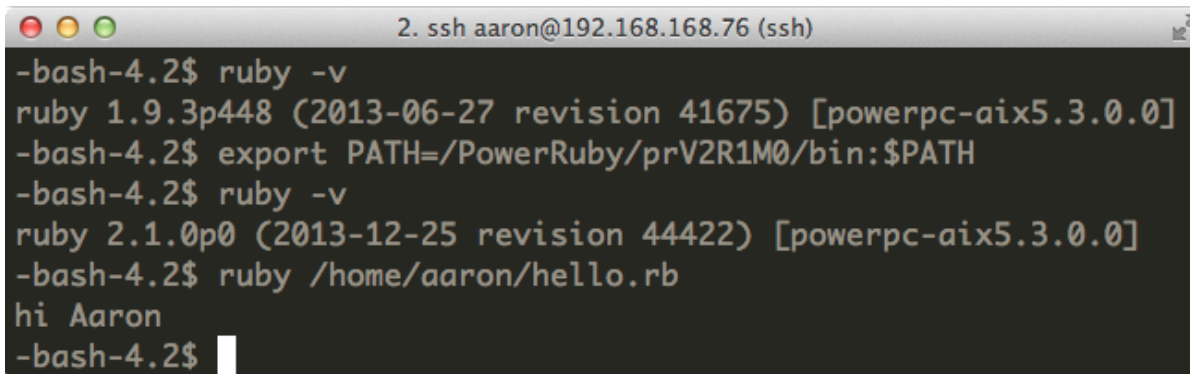
View active Ruby runtime version:

```
ruby -v
```

Alter Ruby runtime version (think TGTRLS):

```
export PATH=/PowerRuby/prV2R1M0/bin:$PATH
```

Symbolic link in /QOpenSys/usr/bin exists for the default ruby binary



```
2. ssh aaron@192.168.168.76 (ssh)
-bash-4.2$ ruby -v
ruby 1.9.3p448 (2013-06-27 revision 41675) [powerpc-aix5.3.0.0]
-bash-4.2$ export PATH=/PowerRuby/prV2R1M0/bin:$PATH
-bash-4.2$ ruby -v
ruby 2.1.0p0 (2013-12-25 revision 44422) [powerpc-aix5.3.0.0]
-bash-4.2$ ruby /home/aaron/hello.rb
hi Aaron
-bash-4.2$
```

/home/aaron/hello.rb

```
def say_hi name
  puts "hi #{name}"
end

say_hi "Aaron"
```

Comments

- Starts with `#` and continues to end of line
- `=begin` and `=end` allow for longer comments or commenting out a large portion of code

```
# This is a comment  
  
x = 1 # This is a comment  
  
=begin  
if customer_credit == "BAD"  
    stop_doing_business("NOW")  
end  
=end
```

Methods

```
def hi(name, age)
  puts "hello #{name}. Age: #{age}"
end
hi("Mr Ruby", 20)
```

```
def hi name, age
  puts "hello #{name}. Age: #{age}"
end
hi "Mr Ruby", 20
```

parentheses are optional for defining or calling a method.

Note: `puts` is a Ruby [kernel method](#)

Methods

```
def say_hello_to(name, age=99)
  return "hello #{name}. You are #{age}"
end
puts say_hello_to("Mr. Ruby")
```

default values can be defined with `age=99`

return value is last statement executed or use the explicit `return` keyword

```
def add_numbers(*numbers)
  total = 0
  numbers.each { |number| total += number }
  return total
end
puts add_numbers(1, 2, 3)
```

pass in a variable number of parameters using an asterisk (*).

Parameters are "pass-by-value", though you can choose to mutate strings.

Note: `puts` is a Ruby [kernel method](#)

Blocks

...used extensively in Ruby and Rails, though you don't need to intimately understand them to use them.

- ★ Pass a chunk of code to be invoked at a later time.
- ★ Similar to Javascript anonymous functions

Events that occur with a block

1. `File.open` will first create the file if it doesn't exist
2. It will `yield` to the block of code and pass an instance of the `file`.
3. `file.puts` is invoked and data is written to file
4. Control is returned to the `File.open` method and it issues a `close` to the `file`.

```
File.open('myfile.txt', 'w') do |file|  
  file.puts "Some data"  
end
```

Blocks

Another example...

```
['a', 'b', 'c'].each do |value|  
  puts "Value:#{value}"  
end  
# Value:a  
# Value:b  
# Value:c
```

Breaking it down...

The `each` portion is a method call

```
['a', 'b', 'c'].each
```

The `do` and `end` is the (*code*) block and `value` is the single parameter passed into the block from the `each` method.

```
do |value|  
  puts "Value:#{value}"  
end
```

Data Types

Dynamically Typed Language...

You don't have to declare your variables ahead of time. **They gain their data-type when first used.** Ruby interpreter looks at the type of value you are assigning to the variable and dynamically works out the variable type.

- ★ Boolean
- ★ Constants
- ★ Symbols
- ★ Hashes
- ★ Arrays
- ★ Strings
- ★ Numbers

Data Types - Constants

- ★ **Start with a capital letter**
- ★ You can change a constant's value but Ruby will issue a warning.
- ★ Ruby variables are case sensitive so two of the below variables that are spelled the same are actually different.

```
I_am_constant = "value"  
i_am_not_constant = "value"  
I_AM_constant = "value"
```

Data Types - Strings

- ★ Use single or double quotes
- ★ Interpolation with double quotes: "hello #{name}"
- ★ **Mutable**, especially with names ending with "!" (aka "bang method")
- ★ Unicode fully supported along with 90+ other character encodings.

Bang Methods (i.e. str.downcase!)

```
irb> name = "Aaron"  
=> "Aaron"  
irb> name.class  
=> String  
irb> name.encoding  
=> #<Encoding:UTF-8>  
irb> "hello #{name}"  
=> "hello Aaron"
```

```
irb> str = "A STRING"  
=> "A STRING"  
irb> str.downcase  
=> "a string"  
irb> puts str  
=> "A STRING"  
irb> str.downcase!  
=> "a string"  
irb> puts str  
=> "a string"
```

Data Types - Symbols

- Start with a colon
- Immutable (can't themselves be changed)
- Used for string values that don't change often to lessen CPU consumption.
- Same-named symbols are stored once in memory no matter how many times they are defined.

```
irb> 'my_string'.object_id
=> 70227383826500
irb> 'my_string'.object_id
=> 70227383742320
irb> :my_symbol.object_id
=> 927688
irb> :my_symbol.object_id
=> 927688
```

colon name

:my_symbol

```
irb> 'hello'.to_sym
=> :hello
```

Data Types - true, false and nil

- true, false, and nil are keywords in Ruby
- All are singleton classes (*only one instance in existence*)
- true and false are boolean values (*surprise!*)
- nil is the absence of a value (*surprise! part II*)
- Method nil? is available to all [objects](#)

```
irb> nil.class
=> NilClass
irb> "hello".nil?
=> false
```

```
irb> true.class
=> TrueClass
irb> 1 == 1
=> true
```

```
irb> false.class
=> FalseClass
irb> 1 > 2
=> false
```

Data Types - Numbers

- All numerics are immutable
- If using decimal point, digits must appear before and after (i.e. correct 0.12 incorrect .12)
- Use underscores for numeric literals: `population_max = 315_900_000`
- Standard operators: `+`, `-`, `*`, `/`, `%`, `**`

Numeric

|-->[Integer](#)

| |-->[Fixnum](#) - Up to 31 bits

| |-->[Bignum](#) - Everything over 31 bits. Converts from FixNum to BigNum automatically

|-->[Float](#) - Represent inexact real numbers using the native architecture

|-->[BigDecimal](#) - Most commonly used for amounts (i.e. monetary)

```
irb> 1.class
=> Fixnum
irb> 9_999_999_999_999_999_999.class
=> Bignum
irb> 12.34.class
=> Float
irb> BigDecimal.new('12.34').class
=> BigDecimal
```

```
irb> 1.odd?
=> true
irb> 2.times do
irb>   puts "hi"
irb> end
hi
hi
irb> 10.to_s
=> "10"
```


Data Types - Hashes

- Store a list of indexed key value pairs
- => is a *hash rocket*, separates keys from values
- Used A LOT in Ruby

```
my_hash = { :one => "value 1",  
            :two => "value 2",  
            :three => "value 3" }
```

```
puts my_hash[:one]    # value 1  
puts my_hash[:two]   # value 2  
puts my_hash[:three] # value 3
```

```
my_hash.delete(:three)
```

Hash rocket short-form

```
my_hash = { one: "value 1",  
            two: "value 2",  
            three: "value 3" }
```

```
my_hash = Hash.new  
my_hash[:one] = "value 1"  
my_hash[:two] = "value 2"  
my_hash[:three] = "value 3"
```

```
my_hash.each do |key, value|  
  puts value  
end
```

Produces:

```
# value 1  
# value 2  
# value 3
```

Data Types - Arrays

- A lot like Hashes except their keys are always consecutive numbers
- Start at an index of zero (0)
- The left shift operator (i.e. <<) will add an element to the array.

```
array1 = ["These", "are", "array", "elements"]
```

```
array2 = []
```

```
array2 << "Use" # index 0
```

```
array2 << "left" # index 1
```

```
array2 << "shift" # index 2
```

```
array2 << "to" # index 3
```

```
array2.push "add" # index 4, another way to add ele
```

```
my_str = array2.pop # returns "add", removes from array
```

Loops

Loops are an important part of any language, but you'll find Ruby leans much more towards *"blocks"*

Loop over a hash

```
hash1 = { name:"Larry", age:34,  
          job:"plumber" }  
  
for key, val in hash1  
  puts "#{key} is #{val}"  
end  
#=> name is Larry  
#=> age is 34  
#=> job is plumber
```

Loop over an array

```
array1 = ["Need", "more", "gum"]  
i = 0  
while array1[i]  
  puts array1[i]  
  i += 1  
end  
#=> Need  
#=> more  
#=> gum  
  
for item in array1  
  puts item  
end  
#=> Need  
#=> more  
#=> gum
```

Logic

Normal

```
if x =< 10
  x += 1
end
```

case statement

```
count = case
  when 1 == x
    "one"
  when 2 == x
    "two"
  else
    "none"
end
```

Single line

```
if x < 10 then x += 1 end
```

if, elsif, and else

```
if x < 10 || y == 10
  code
elsif x > 10 && y == 10
  code
else
  code
end
```

unless is the opposite of if

```
unless x == 10
  code
end
```

Conditional Assignment and Execution

Normal conditional assignment

```
x = nil
if x.nil?
  x = "default"
end
```

Value of `x` will be replaced with "default", but only if `x` is `nil` or `false`

```
x ||= "default"
```

Line will only execute if `x` is `nil`

```
x = "default" if x.nil?
```

Exception Handling

- Exceptions indicate something has gone wrong (hard errors)
- Custom exceptions can be created
- Ruby programs terminate when an exception is raised unless it is rescued
- Base class is `Exception`
- Use `raise` to cause an exception

```
begin
  sum / 0
rescue ZeroDivisionError => e
  puts 'Error: #{sum} / 0 Text: #{e}'
else
  # do this if no exception was raised
  # (optional)
ensure
  # do this regardless of whether an
  # exception was raised (optional)
end
```

End-of-line rescue with default return value

```
irb> 1 / 0
=> ZeroDivisionError:
divided by 0
irb> 1 / 0 rescue 0
=> 0
```

Classes

- Use when you need to instantiate to a variable or for inheritance.
- **Remember:** Everything in Ruby is an object (though they hide this complexity much better than other languages)
- Methods are public by default

`Barber.new` invokes `initialize`, the constructor

`@name` is a *class instance* variable which all start with `@` and are available to any method within the instance.

Everything below `private` keyword will be a private method.

bit.ly/1jjgUH – Formal docs

```
class Barber
  def initialize name
    @name = name
  end
  def cut_hair
    puts "Not that close #{@name}!"
    undo_haircut
  end
private
  def undo_haircut
    puts "Crap! Ctrl+Z is not working!"
  end
end

b = Barber.new "Aaron"
b.cut_hair
# Not that close Aaron!
# Crap! Ctrl+Z is not working!
```

Classes - accessors

- `attr_accessor` makes getters and setters dead simple (*big time saver*)
- `attr_reader` and `attr_writer` also exist for more fine-grained control
- **Note:** `attr_accessor` is actually a method call without parentheses and with `name` as the sole parameter. As seen here, some features are implemented using Ruby to enhance Ruby - brilliant!

short-form

```
class Barber
  attr_accessor :name
end
```

long-form

```
class Barber
  def name
    @name
  end
  def name=(value)
    @name = value
  end
end
```

Given the below code:

```
b = Barber.new
b.name = "Aaron"
```

The second line is actually invoking the `name=` method in the `Barber` class instance (`b`). The Ruby community calls this an "operator method" - a unique idea that requires mental acclimation for new-comers.

bit.ly/1jJlMdr – Formal docs

Variables

- Case sensitive
- Can contain letters, numbers and underscore but can't begin with a number
- *Snake casing* is the popular Ruby convention (i.e. `my_var_name` vs. `myVarName`)

Local variable - Begins with a lowercase letter. Scoped to the module, method or block.

```
local_var = ...
```

Instance variable - Scoped to the instance of a class.

```
@instance_var = ...
```

Class variable - Spans all instances of a class.

```
@@class_var = ...
```

Global variable - Visible anywhere in your program. Community says not good practice to use.

```
$global_var = ...
```

tutorialspoint.com/ruby/ruby_variables.htm – Good tutorial

Modules

- Collection of classes, methods, and constants
- Used for namespacing, similar to Java's package
- Supports "mixins"
- Procedural code allowed
 - you can ignore Class system as needed.
 - Mix and match object with procedural code

`def Foo.a2` and `def self.a3` accomplish the same thing as they both relate their methods back up to module `Foo`.

`def a1` is not able to be invoked in the current context. It first needs to be included into another class in a "mixin" scenario.

```
module Foo
  MY_CONSTANT = 100
  def a1
    puts "a1: hello"
  end
  def Foo.a2
    puts "a2: hello"
  end
  def self.a3
    puts "a3: hello"
  end
end

Foo::MY_CONSTANT
Foo.a2
Foo.a3
Foo.a1

# 100
# "a2: hello"
# "a3: hello"
# NoMethodError:
# undefined method `a1'
# for Foo:Module
```

Mixins

- Eliminate need for multiple inheritance
- Similar to `/copy` in RPG to bring in a chunk of code
- Use `include` to "*mix in*" a module within a class

```
module Foo
  def a1
    puts "a1: I am a #{self.class.name}"
  end
  def Foo.a2
    puts "a2: I am a #{self.class.name}"
  end
  def self.a3
    puts "a3: I am a #{self.class.name}"
  end
end

class Bar
  include Foo # Foo gets "mixed in"
  def run
    a1
    Foo.a2
    Foo.a3
  end
end

Bar.new.run
#=> a1: I am a Bar
#=> a2: I am a Module
#=> a3: I am a Module
```

bit.ly/1g8cTyp – Formal docs

Duck typing

Ruby is less concerned with the object type and more concerned with what methods can be called on it and what operations can be performed on it.

Walk like a duck? Quack like a duck? Must be a duck.

```
"Feathers".respond_to?(:to_str)
# => true

4.respond_to?(:to_str)
# => false

nil.respond_to?(:to_str)
# => false
```

ruby-doc.org/core-1.9.3/Object.html#method-i-respond_to-3F – respond_to? docs
en.wikipedia.org/wiki/Duck_typing – Concept
rubylearning.com/satishtalim/duck_typing.html – Formal docs

...classes are never closed. You can always add methods to an existing class - even at runtime.

Open Classes

- Ruby runtime allows the further defining of existing classes
- I'd argue to never change base Ruby classes as below, but this immediately shows the power.

```
irb> '0'.is_number?  
=> NoMethodError: undefined method `is_number?' for "0":String  
irb> class String  
irb>   def is_number?  
irb>     true if Float(self) rescue false  
irb>   end  
irb> end  
irb> '0'.is_number?  
=> true
```

```
class Customer < ActiveRecord::Base  
end
```

When the Rails application starts, it will query DB2 for the customer table metadata so it can dynamically insert getters and setters for column data.

vitarara.org/cms/ruby_metaprogramming_declaratively_adding_methods_to_a_class – Example
cantgrokwontgrok.blogspot.com/2012/01/ruby-open-classes.html – Example
amzn.to/1d9BxQu – Metaprogramming Ruby: Program Like the Ruby Pros

DSL - Domain Specific Language

... a computer language or syntax specialized to a particular application domain.

- An extension of Ruby syntax - methods look like keywords.
- Meant to create syntax for a particular problem that feels more natural
- You can create a custom DSL for your company's specific needs
- jQuery (*internal*) and SQL (*external*) are both DSLs.
- Often times it is used for a custom configuration file, like the `routes.rb` file of a Rails application, or to simplify the altering of a database.

app/db/migrations/*_create_products.rb

```
class CreateProducts < ActiveRecord::Migration
  def change
    create_table :products do |t|
      t.string :name
      t.text :description
      t.timestamps
    end
  end
end
```

app/config/routes.rb

```
Blog::Application.routes.draw do
  resources :posts
end
```

Both `create_table` and `resources` are actually a regular Ruby methods

en.wikipedia.org/wiki/Domain-specific_language – Computer Science definition
infoq.com/presentations/domain-specific-languages – Intro to DSL (Martin Fowler)
softwarebyjosh.com/2012/01/08/How-To-Write-Your-Own-DSL.html - Step by step

Rake

... a software task management tool similar to “make”. It allows you to specify tasks and describe dependencies as well as to group tasks in a namespace.

- **Est 2005 by Jim Weirich**
- Syntax is completely defined in Ruby
- “Prerequisites” allow for things like the Rails `:environment` be loaded before the task is invoked, giving full access to all models and other application code.
- One could think of it as **“The CL of Ruby”**

my_app/lib/tasks/set_pay_in_full.rake

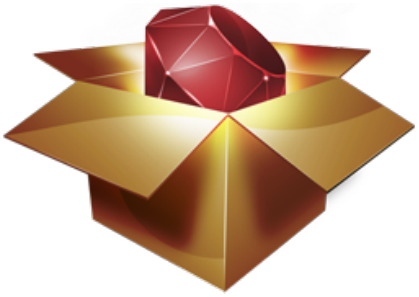
```
task :set_pay_in_full => :environment do
  Enrollment.all.each do |e|
    e.pay_in_full = e.term.pay_in_full
    e.save!
  end
end
```

Invoke via PASE command line:

```
$ rake set_pay_in_full
```

github.com/jimweirich/rake – Home website

railscasts.com/episodes/66-custom-rake-tasks – Video tutorial overview



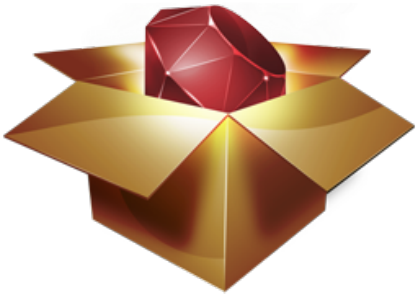
RubyGems... *simplify the process of installing, removing, updating and managing Ruby libraries and their dependencies.*

Ruby is a great language, but the Ruby community wanted to start modularizing and sharing code - enter RubyGems.

- Est 2001
- Server for hosting and distribution of gems, rubygems.org
- Manage gem dependencies
- Manage multiple versions of the same library easily
- Included in Ruby as of 1.9.3
- Rails and Bundler are gem themselves

“There’s a gem for that”
- every Ruby business programmer

```
-bash-4.2$ gem install will_paginate  
Fetching: will_paginate-3.0.5.gem (100%)  
Successfully installed will_paginate-3.0.5  
Installing ri documentation for will_paginate-3.0.5  
1 gem installed  
-bash-4.2$
```

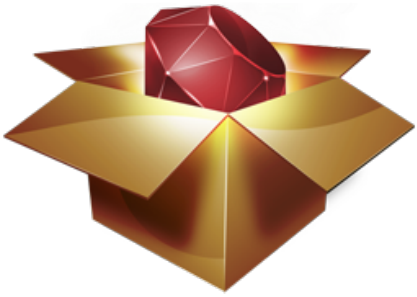
```
gem list                # List installed gems
gem environment         # Display RubyGems config info
gem install rails      # Install a named gem
gem update rails       # Update a named gem
gem update              # Update all installed gems
gem update --system    # Update RubyGems itself
gem uninstall rails    # Remove an installed gem
```

```
$ gem list
```

```
*** LOCAL GEMS ***
actionmailer (4.0.0)
actionpack (4.0.0)
activemodel (4.0.0)
activerecord (4.0.0)
activesupport (4.0.0)
atomic (1.1.14)
bigdecimal (1.2.0)
bootstrap-sass (3.0.3.0)
builder (3.1.4)
bundler (1.3.5)
coffee-rails (4.0.0)
coffee-script (2.2.0)
. . .
rails (4.0.0)
```

```
$ gem list -d
```

```
. . .
ibm_db (2.5.11)
  Author: IBM
  Homepage: rubyforge.
org/projects/rubyibm/
  Installed at:
    /PowerRuby/prV2R0M0/lib/ruby/gems/2.
0.0
  Rails Driver and Adapter
```



The `gem environment` command conveys a lot of important information about where Gems will be placed when `gem install` is run, what paths will be searched for Gems at runtime, and what site to look at to download new gems.

\$ gem environment

RubyGems Environment:

- RUBYGEMS VERSION: 2.1.9
- RUBY VERSION: 2.0.0 (2013-06-27 patchlevel 247) [powerpc-aix5.3.0.0]
- INSTALLATION DIRECTORY: /PowerRuby/prV2R0M0/lib/ruby/gems/2.0.0
- RUBY EXECUTABLE: /PowerRuby/prV2R0M0/bin/ruby
- EXECUTABLE DIRECTORY: /PowerRuby/prV2R0M0/bin
- SPEC CACHE DIRECTORY: /home/AARON/.gem/specs
- RUBYGEMS PLATFORMS:
 - ruby
 - powerpc-aix-5
- GEM PATHS:
 - /PowerRuby/prV2R0M0/lib/ruby/gems/2.0.0
 - /home/AARON/.gem/ruby/2.0.0
- GEM CONFIGURATION:
 - :sources => ["http://rubygems.org/"]
- REMOTE SOURCES:
 - http://rubygems.org/

yaml

- "YAML Ain't Markup Language" (abbreviated YAML)
- Use for human-friendly data serialization (i.e. config files)
- Used by many programming languages, including Ruby
- Ruby's implementation is done with the Psych Gem

Before YAML

```
{ 'development' => {  
  'adapter' => 'ibm_db',  
  'username' => 'A2222',  
  'password' => 'A2222',  
  'database' => '*LOCAL',  
  'schema' => A2222_D  
}
```

After YAML

```
development:  
  adapter: ibm_db  
  username: A2222  
  password: A2222  
  database: '*LOCAL'  
  schema: A2222_D
```

yaml.org – Home site

yaml.org/YAML_for_ruby.html – Ruby specific from home site

github.com/tenderlove/psych - Ruby's implementation

References

Matz Ruby Intro (tongue in cheek): slideshare.net/vishnu/the-top-10-reasons-the-ruby-programming-language-sucks/

Further Learning

TutorialsPoint.com/ruby

Codecademy.com/tracks/ruby

CodeSchool.com/paths/ruby

en.wikibooks.org/wiki/Ruby_Programming

TeamTreehouse.com

PowerRuby.worketc.com/kb - PowerRuby knowledge base

iwanttorearnruby.com - large collection of sites

THE END!

Aaron Bartell

abartell@kregeltech.com

www.MowYourLawn.com

twitter.com/aaronbartell

