Intro to Ruby and Rails

By Mark Menard

Enable Labs

Enable Labs

What is a dynamic language?

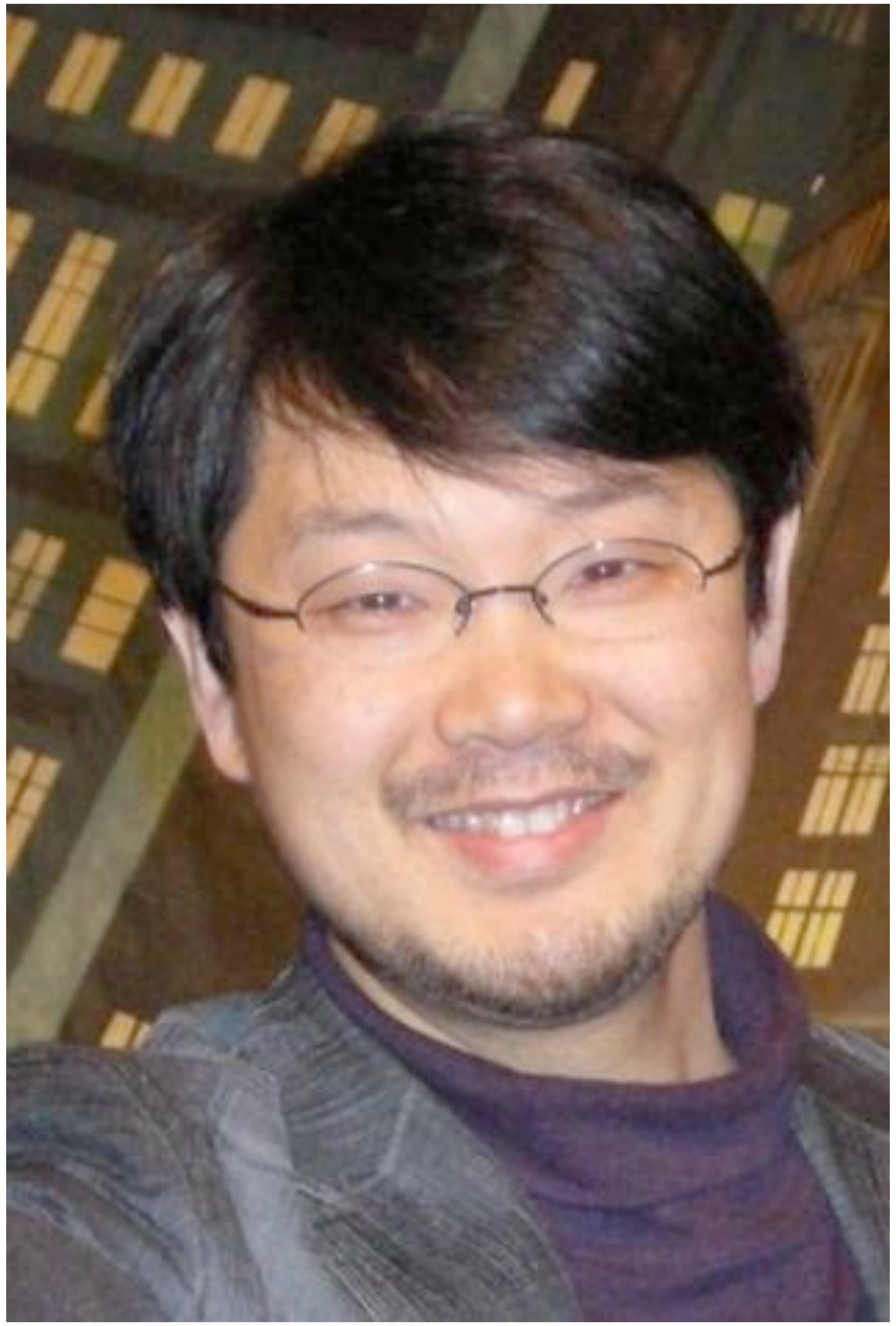
"…a class of high-level programming languages that execute at runtime many common behaviors that other languages might perform during compilation, if at all. These behaviors could include extension of the program, by adding new code, by extending objects and definitions, or by modifying the type system, all during program execution."

-Dynamic Programming Languages from Wikipedia

Enable Labs

Ruby

- Yukihiro Matsumoto
  - “Matz”
  - Released Ruby 1.0 in 1996
  - Focus on Human, not Machines
  - Making the work pleasurable for the programmer
  - Principle of least astonishment (POLA)
    - Array.each
    - Object.persisted?



REPL (Read-eval-print-loop): irb

- Access to the entire language and functionality in an interactive environment
- Prototype new functionality
- Explore functionality of new libraries
- Manually exercise your code
- Interact with your application while it is running

```
~ $ irb
ruby-1.9.2-p290 :001 >
> a = 1
=> 1
> b = 2
=> 2
> a + b
=> 3
```

Enable Labs

Primitives are not primitive!

- Everything in Ruby is an Object!
- Fixnum
  - 1, 2, 32358
  - Automatically grows as needed to satisfy assignment
- Floats
  - decimal numbers
  - currency
- Strings
  - “I am a string!”
- Boolean
  - True
  - False
- Nil

Enable Labs

Array, Hash and Regex in Ruby

```
# Create an array
array = [ 1, 2, 3, 4, 5 ]

> array
[1,2,3,4,5]
```

```
# Create a hash
hash = { :a => 'a', :b  => 'b', 1 => 1, "a" => "string a" }

> hash
{ :a => 'a', :b  => 'b', 1 => 1, "a" => "string a" }
```

```
# Using a Regular Expression

email_regex = /[a-z\-\.]@[a-z]+\.(com|org)/i

"john.doe@example.org" =~ email_regex ? "match" : "no match"
=> "match"

"john.doe@example.edu" !~ email_regex ? "no match" : "match"
=> "no match"
```

Enable Labs

Methods

- The Ruby standard is to use snake_case_for_method_names
- Method names should be
  - descriptive
  - start with a lower case letter
- parameters should be between parentheses
- parameters can include a default
- methods always return the last evaluated expression or you can use return
- parentheses are not required when calling a method

```
def turn_on_windshield_wipers (speed = WindshieldWiper::Low)
   wiper_speed = speed
end

> turn_on_windshield_wipers (3)
=> 3
> turn_on_windshield_wipers 1
=> 1
```

Enable Labs

Variables

- Do not need to be declared, just use it and it springs into existence.
  - Instance variables start with @ (ie: @name)
    - Method variables are just the name (ie: name =)
  - Class variables start with @@ (ie: @@class_var = 1)
  - Constants start with a capital letter (A-Z)
  - Global variables begin with $
  - Method scoped variables do not use a special character

```
class Mondial
  @@make = "Ferrari"

  def initialize
    @model = "Mondial"
  end
  def make_and_model
    "#{@@make} #{@model}"
  end
end
```

Enable Labs

Control Structures

```
if car.lights_on?
  car.lights_off!
elsif car.in_drive?
  car.shutdown_now!
else
  car.toot!
end
```

```
while car.speed > LEGAL_LIMIT
  car.decelerate(5)
  sleep 5
end
```

```
puts “I have something to say” if car.off?
```

```
case car.color
  when Car::Grey
    puts “I’m slow”
  when Car::Red, Car::Blue
    puts “Arrest me. I’m red”
  else
    puts “Keep on keepin’ on!”
end
```

```
puts “Whoa, slow down!” unless car.off?
```

Enable Labs

Iterators

```
def door_open?
  result = false
  for door in doors do
    result = true if door.open?
  end
  result
end
```

```
def door_open?
  doors.each do |door|
    return true if door.open?
  end
  false
end
```

```
def door_open?
  doors.any? { |door| door.open? }
end
```

```
def open_doors
   doors.collect { |door| door.location }
end
```

```
def open_doors
   doors.map { |door| door.location }
end
```

```
def open_doors
   doors.collect(&:location?)
end
```

```
def open_door_count
   doors.inject(0) { |sum, door| sum += 1 if door.open? }
end
```

Enable Labs

Classes

- Starts with the keyword class
- Class names should be CamelCase, but must start with a capital letter
- Sub-classed by using the “less-than” operator <
- Contains
  - variables
  - constants
  - class methods
  - public/private/protected methods
- The constructor method is known as initialize

Enable Labs

Methods in a class

```
class Car
  def make=(make)
    @make = make
  end
  def make
    @make
  end
end
```

```
class Car
  def initialize(make)
    @make = make
  end
  def make
    @make
end
```

```
class Car
  attr_writer :make
  attr_reader :make
end
```

```
class Car
  attr_accessor :make
end
```

```
> car = Car.new("VW")
> puts car.make
VW
=> nil
> car.make= "BMW"
> puts car.make
BMW
=> nil
```

Inheritance in Ruby

Class Definition

```
class Foo
  def self.what_am_i
    puts “I am a Foo”
  end
  def do_something
    puts "Foo#do_something"
  end
end

> Foo.what_am_i
I am a Foo
=> nil
> foo = Foo.new
=> #<Foo:0x007dfca520>
> foo.do_something
Foo#do_something
=> nil
```

Inheritance or Sub-Classing

```
class Bar < Foo
  def do_something
    super
    puts "Bar#do_something"
  end
end

> Bar.what_am_i
I am a Foo
=> nil
> bar = Bar.new
=> #<Bar:0x008cdca41a>
> bar.do_something
Foo#do_something
Bar#do_something
=> nil
```

Enable Labs

Mixins in Ruby

```
module Logging
  def log (msg)
    puts msg
  end
end
```

```
class OrderDispatcher
  include Logging

  def dispatch_international_order (order)
    destination = DestinationService.find(order.foreign_party)
    log("Sending #{order.number} to #{destination.name}")
    ...
  end
end
```

```
class Car
  include Logging

  def accelerate (by)
    speed += by
    log “Increased speed to #{speed}”
  end
end
```

Enable Labs

Duck Typing

- An object's type is defined by the messages it responds to
- Interfaces are not programmatically enforced
- An object is about what it can do, not what it is!

"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck."
- James Whitcomb Riley

Enable Labs

Duck Typing in Ruby

```
class Cat
  def talk
    puts "Meow"
  end
end
```

```
class Dog
  def talk
    puts "Woof"
  end
end
```

```
class Person
  def talk
    puts "Hi"
  end
end
```

```
class Duck
  def talk
    puts "Quack!"
  end
end
```

```
>[Cat.new,Dog.new,Duck.new,Person.new].each do |ob|
  puts ob.talk
end
Meow
Woof
Quack
Hi
```

Enable Labs

Terse / Low Ceremony

- Semi-colons optional
- Hashes are created with { key: value }
- Arrays created with [value1, value2]
- Hashes passed as last parameter in a method need no braces {}
- A code block is implicitly created as the last argument to a function
- No variable type declarations
- Last evaluated expression of a method is automatically returned

See the solution not the noise

Enable Labs

Terse / Low Ceremony - Example Exceptions

Java Style Exceptions

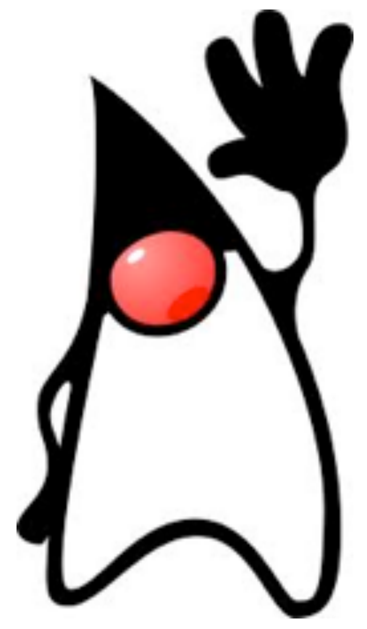
```
void foo throws FooException {
  throw new FooException();
}

void bar throws FooException {
  foo();
}

void bar {
  try {
    foo
  } catch (FooException e) {

  } finally {

  }
}
```



Ruby Style Exceptions

```
def foo
  raise FooException
end

def bar
   foo
end

def bar
   foo
   rescue => e
   ensure
end
```

Enable Labs

Git - Source Code Control - Commands for Labs

- Cloning

```
> git clone git://github.com/EnableLabs/rails_training_feb_2013.git
```

- Checkout

```
> git checkout <step_n>
```

git - Commands for Development

- Initialize a repo

```
> git init
```

- Get the status of your repo

```
> git status
```

- Add a changed file

```
> git add <filename>
```

- Create and use a new branch

```
> git checkout -b <branch>
```

- Commit changes

```
> git commit -m “commit message”
```

- Push files to a central repository

```
> git push origin master
```

Enable Labs

Helpful Sites for Continued Learning of Ruby/Rails

Ruby News

- Rubyflow - http://www.rubyflow.com
- Ruby Inside - http://www.rubyinside.com
- Ruby Weekly - http://www.rubyweekly.com
- Ruby Lang - http://www.ruby-lang.org

Podcasts

- Ruby 5 - http://ruby5.envylabs.com
- Ruby Rogues - http://rubyrogues.com

Videos/Learning

- Confreaks - http://www.confreaks.com
- Railscasts - http://www.railscasts.com
- Code School - http://www.codeschool.com

Enable Labs

Local Resource

- Tech Valley Ruby Brigade
  - http://www.techvalleyrb.org
  - Meets the 4th Wednesday of every month
  - Local and out of town speakers on various aspects of web development. Past topics include
    - Writing API's
    - Creating your own gems
    - Security exploits
    - Testing
    - Development Techniques
    - Developer Tools
  - Free beer and pizza!

tech valley RUBY BRIGADE

Enable Labs

Lab

```
> git clone git://github.com/EnableLabs/rails_training_feb_2013.git
> cd rails_training_feb_2013
> git checkout lab1
> bundle install
> rspec spec
```

Make the tests pass!

Enable Labs

Bonus Slides

Enable Labs

gems - Sharing Ruby Libraries

- Require a gem

```
require ‘<gem name>’
```

- Requiring a specific version

```
require ‘rubygems’
gem ‘active_support’, ‘~> 3.0’
```

- Listing installed gems

```
gem list
```

- If you want to add functionality check out http://rubygems.org
  - Chances are someone has already done it as there are over 50,000 gems
  - One source for documentation and code
- Check out http://github.com

Enable Labs

Open Classes

- Classes are open for modification at runtime
  - Methods can be added
  - Methods can be redefined
  - Methods can be added to an instance
  - Methods can be redefined on an instance
- Allows you to closely adapt the language to your problem domain
  - It’s like sculpting in clay instead of stone
- Mocking and stubbing become trivial

Enable Labs

Open Classes

```
class String

  def url_decode
    CGI::unescape(self)
  end


  def url_encode
    CGI::escape(self)
  end

end
```

Enable Labs

Open Classes in Ruby

```
class Foo
  def escape
    puts "Whee! I'm free!"
  end
end

foo = Foo.new
foo.escape #=> prints "Whee! I'm free!"

class Foo
  def escape
    puts "Not so fast!"
  end
end

foo.escape #=> prints ”Not so fast!"
```

Enable Labs

Closures in Ruby

```
def create_closure (name)

  # Create a closure closing over the scope which contains 'name'
  lambda do |job|
    puts "#{name} has a new job doing #{job}."
  end
end

closure = create_closure("Mark")
closure.call("web development") #=> Mark has a new job doing web development.
closure.call("goat milking") #=> Mark has a new job doing goat milking.
```

Enable Labs

Method Missing in Ruby

```
class MethodMissing
  def method_missing (name, *args)
    puts "Oops! Method #{name} does not exist."
  end
end

mm = MethodMissing.new
mm.foo #=> prints “Oops! Method foo does not exist.
```

Enable Labs

Metaprogramming in Ruby

```
class MethodMissing
  def method_missing (name, *args)
    puts "method_missing called the first time."
    puts "Defining #{name} method."
    instance_eval %Q{
      def #{name.to_s} (args)
        puts "Inside the dynamically defined foo method."
      end
    }
    send(name, args)
  end
end

mm = MethodMissing.new
mm.foo(nil)
mm.foo(nil)
```

Enable Labs

Operator Overloading in Ruby

```
class Person
  def initialize (name)
    @name = name
  end

  def + (other)
    "#{@name} and #{other.to_s} have gotten together"
  end

  def to_s
    @name
  end
end

mark = Person.new("Mark")
sylva = Person.new("Sylva")

puts mark + sylva #=> "Mark and Sylva have gotten together"
```

Intro to Ruby - Summary

- Dynamic Object Oriented Language
  - Everything is an object
- Duck typed
- Has open classes
- Uses dynamic method dispatch
- Supports method_missing functionality
- Support meta-programming
- Executable Class Definitions
- REPL
- Mixins
- Everything is an expression
- Closures
- Literal arrays, hashes and regexes
- Operator overloading
- Runs on Windows, Linux, *nix, OS X, Java, and .Net

Enable Labs

Examples of Dynamic Languages

php

JavaScript

Perl

LiSP

Groovy

python

small talk

the programming language Lua

Enable Labs

Dynamic vs. Static

Dynamic Languages	Static Languages
Variable types do not have to be specified	Variable types generally have to be specified
Class definitions can be modified at run time	Class definitions are frozen at time of specification
Standard library can be augmented	Standard library is frozen
Generally terse	Generally more verbose
Frequently used for scripting	Generally not used for scripting
Usually aren’t compiled	Compiled
Usually interpreted, but can be run in a VM	Can run on a VM or on bare metal
Typically support reflection	Can support reflection

Enable Labs