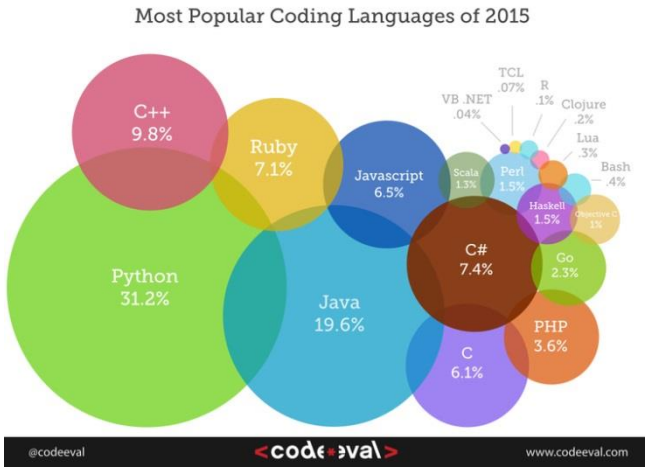
Ruby on Rails

Overview

- Ruby programming language
  - History and philosophy
  - Features
  - Syntax
- Ruby on Rails
  - History and philosophy
  - Rails architecture
  - Directory structure

Ruby: History & philosophy

- History
  - 1993, Yukihiro Matsumoto
  - General purpose language
    - Best known for web programming



"I was talking with my colleague about the possibility of an object-oriented scripting language. I knew Perl (Perl4, not Perl5), but I didn't like it really, because it had the smell of a toy language (it still has). The object-oriented language seemed very promising. I knew Python then. But I didn't like it, because I didn't think it was a true object-oriented language — OO features appeared to be add-on to the language. As a language maniac and OO fan for 15 years, I really wanted a genuine object-oriented, easy-to-use scripting language. I looked for but couldn't find one. So I decided to make it."

–Yukihiro Matsumoto

"Often people, especially computer engineers, focus on the machines. They think, 'By doing this, the machine will run faster. By doing this, the machine will run more effectively. By doing this, the machine will something something something.' They are focusing on machines. But in fact we need to focus on humans, on how humans care about doing programming or operating the application of the machines. We are the masters. They are the slaves."

–Yukihiro Matsumoto

Ruby: features

- Interpreted: Matz's Ruby Interpreter (MRI)
  - 1.9+, Yet Another Ruby Virtual Machine (YARV)
    - Automatically compiles to byte-code, no separate tool
    - Also other byte-code compilers, e.g. Rubinius
- Dynamic *and* strongly typed
  - Var type determined at runtime, type can change
    - But strict about what you can do with types

      Perl: `print "5" + 3 → 8`

      Javascript: `alert("5" + 3) → "53"`

      Ruby: `puts "5" + 3 → runtime error`

      `puts "5".to_i + 3 → 8`
  - Uppercase variable name = constant

Ruby: features

- Object-oriented
  - Everything is an object
  - Single inheritance
    - But mixins allow shared methods
  - Duck typing: if it walks like a duck, talks like a duck, then treat it like a duck
- Everything is an expression
- RubyGems
  - Gem provides a library or plug-in
  - Package manager like apt-get
  - "There's a gem for that"

```ruby
199.abs
5.times { print "Hello world! " }
```

```ruby
def plus_one_to_y(x)
  @y = x + 1
  # puts "blah"
end

puts plus_one_to_y(10)
puts @y
```

Collections

```ruby
a = [1, 'hi', 3.14, 1, 2, [4, 5]]

a[2]              # => 3.14
a.[](2)           # => 3.14
a.reverse         # => [[4, 5], 2, 1, 3.14, 'hi', 1]
a.flatten.uniq    # => [1, 'hi', 3.14, 2, 4, 5]
```

```ruby
hash = { :water => 'wet', :fire => 'hot' } puts hash[:fire] #
prints "hot"

hash.each_pair do |key, value|   # or: hash.each do |key, value|
  puts "#{key} is #{value}"
end
# returns {:water=>"wet", :fire=>"hot"} and prints:
# water is wet
# fire is hot

# deletes the pair :water => 'wet' and returns "wet" hash.delete
:water

# deletes the pair :fire => 'hot' and returns {}
hash.delete_if {|key,value| value == 'hot'}
```

Classes, constructors, instance vars

```ruby
class Sample
  def hello
    puts "Hello Ruby!"
  end
end

object = Sample. new
object.hello
```

```ruby
class Person
  def initialize name
    @name = name
  end

def get_name
    @name
  end
end

person = Person.new "Jane"
puts person.get_name
```

Getters and setters

```ruby
class Person

  attr_reader :name   # Create getter
  attr_accessor :age  # Create getter and setter

  def initialize(name, age)
    @name, @age = name, age
  end

  def to_s
    "#{name} (#{age})"
  end

end

bob = Person.new("Bob", 33)

puts(bob.name)      # Prints "Bob"
puts(bob.age)       # Prints 33
bob.age = 30        # Changes age to 30
bob.name = "Alice"  # Runtime exception
```

OOP: inheritance

```ruby
class Being
  @@count = 0

  def initialize
    @@count += 1
  end

  def show_count
    "There are #{@@count} beings"
  end
end

class Animal < Being
  def initialize
    super
    puts "Animal is created"
  end
end

class Dog < Animal
  def initialize
    super
    puts "Dog is created"
  end
end
```

Duck typing

```ruby
class Duck
  def quack
    'Quack!'
  end

  def swim
    'Paddle paddle paddle...'
  end
end

class Goose
  def honk
    'Honk!'
  end
  def swim
    'Splash splash splash...'
  end
end

def make_it_swim(duck)
  duck.swim
end

puts make_it_swim(Duck.new)
puts make_it_swim(Goose.new)
```

Modules and mixins

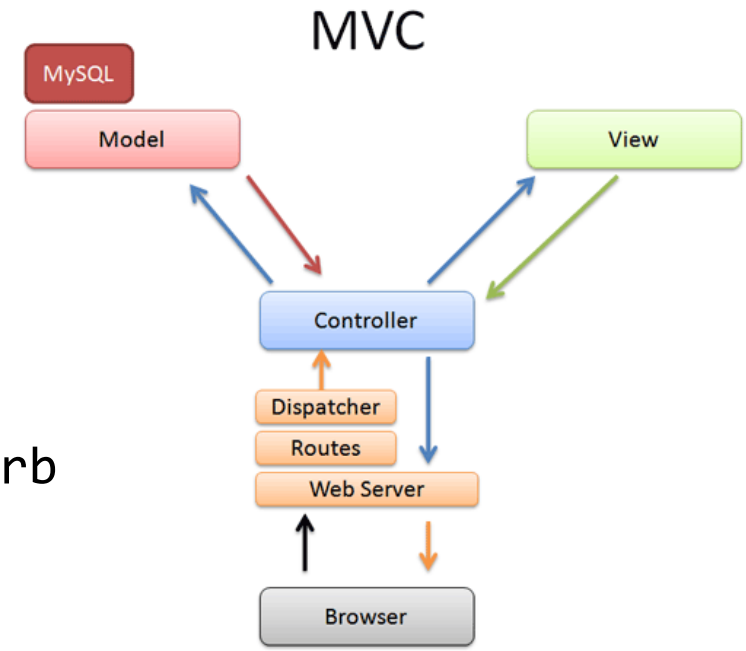
```ruby
module A
  def a1
    puts "a1 is the best!"
  end
end

module B
  def b1
    puts "seriously, what about me: b1?!?"
  end
end

class Sample
  include A
  include B
  def s1
    puts "s1 is king of the hill!"
  end
end

samp = Sample.new
samp.a1
samp.b1
samp.s1
```

Ruby on Rails

- History
  - 2005, David Heinemeier Hansson
    - Working on Basecamp project management tool
    - Extracted Rails from the project
- Philosophy:
  - Convention over configuration
  - Don't Repeat Yourself (DRY)
  - Rails is opinionated
    - Makes assumptions about the "best" way to do things
    - Designed to encourage you to do it that way

Rails architecture

- Model View Controller
  - Model
    - Article class: a blog entry
    - Table `articles` in a database
    - Ruby file: `app/models/article.rb`
  - View
    - Handles presentation to the user
    - Template: `app/views/articles/show.html.erb`
    - The controller's minions
  - Controller
    - Parses user requests, queries/updates models
    - Ruby file: `app/controllers/articles_controller.rb`



http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/

Getting started: Rails tutorial

http://railsforzombies.org/

Ruby on Rails

- Ruby: the programming language
  - Object-oriented scripting language
  - Dynamic and strongly typed
  - Strong developer community
  - "There's a gem for that"
- Ruby on Rails: web framework
  - Convention over configuration
  - Don't Repeat Yourself (DRY)
  - Model-View-Controller design pattern