

CS 152: *Programming Language Paradigms*

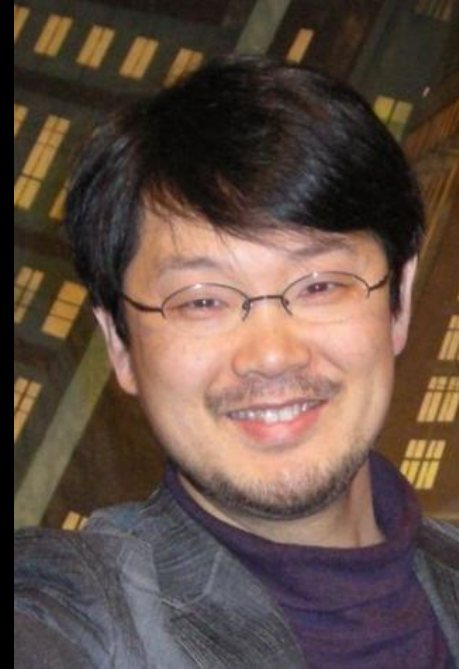
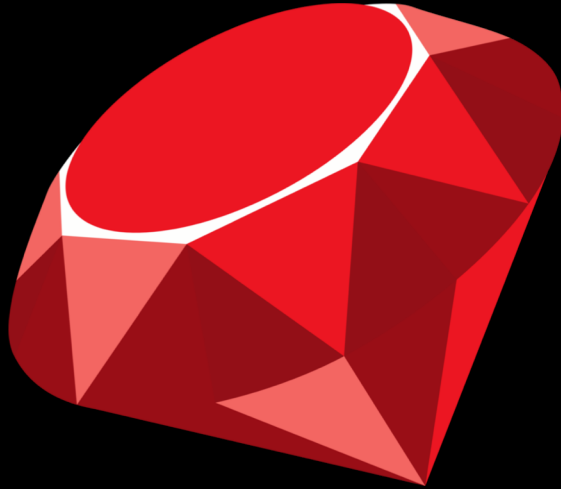


Ruby

Prof. Tom Austin

San José State University

Introduction to Ruby



Created by
Yukihiro Matsumoto
(known as "Matz")

Ruby influences



Smalltalk

- everything is an object
- blocks
- metaprogramming

Perl

- regular expressions
- function names



Ruby on Rails

- Ruby's "killer app"
 - lightweight web framework
 - "convention over configuration"
- David Heinemeier Hansson (DHH)
 - initial framework was PHP
 - abandoned PHP for Ruby

Hello World in Ruby

```
puts 'Hello world!'
```

Working with data structures

```
a = [1, 2, 3]
```

```
m = { 'a' => "Apple",  
      'b' => "Banana",  
      'c' => "Cantalope" }
```

```
puts a[0]
```

```
puts m['a']
```

Ruby is object-oriented

"I was talking with my colleague about the possibility of an object-oriented scripting language. [...] 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." --Matz 1999

```
class Person
  def initialize name # Constructor
    @name = name
  end

  def name            # Getter
    return @name
  end

  def name= newName  # Setter
    @name = newName
  end

  def say_hi          # Method
    puts "Hello, my name is #{@name}."
  end
end
```

The @ indicates an object's field

The = in the method name (by convention) indicates assignment

Generating getters and setters

Powerful
metaprogramming

```
class Person
  attr_accessor :name
  def initialize name # Constructor
    @name = name
  end
  def say_hi          # Method
    puts "Hello, my name is #{@name}."
  end
end
```

Using a class in Ruby

```
p = Person.new "Joe"  
puts "Name is #{p.name}"  
p.say_hi
```

Inheritance in Ruby

(in-class)

Parent Class

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

  def speak
    puts "#{@name} says bark"
  end
end

rex = Dog.new('Rex')
rex.speak
```

Child Class

```
class GuardDog < Dog
  attr_reader :breed
  def initialize(name, breed)
    super(name)
    @breed = breed
  end
  def attack
    puts "Growl"
  end
end
```

Equivalent to
extends in Java

```
satan = GuardDog.new('Satan', 'Doberman')
puts "Satan is a #{satan.breed}"
satan.speak
satan.attack
```

Mixins

- Allow user to add features to a class
- Similar to interfaces in Java, but programmer can specify functionality.

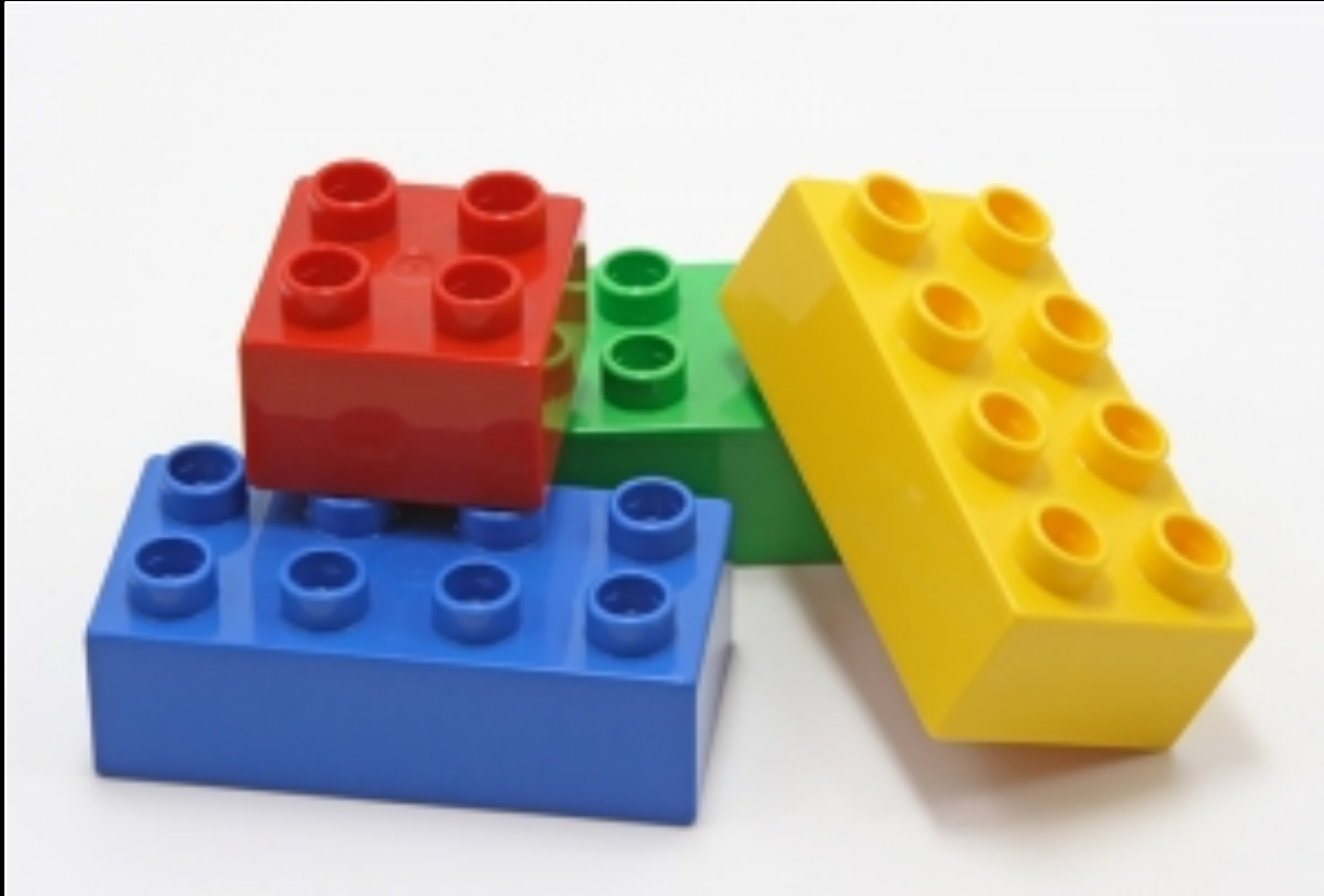
```
class Person  
  include Comparable  
end
```

```
module RevString
  def to_rev_s
    to_s.reverse
  end
end
```

```
class Person # Re-opening class
  include RevString
  def to_s
    @name
  end
end
```

```
p.to_rev_s # p defined previously
```

Blocks in Ruby



Blocks in Ruby

- Superficially similar to blocks in other languages.
- Can create custom control structures.
- (We'll discuss in depth another day).

File I/O Example

(in class)

File I/O

```
file = File.open(  
    'temp.txt', 'r')  
file.each_line do |line|  
    puts line  
end  
file.close
```

File I/O with blocks

```
File.open('file', 'r') do |f|  
  f.each_line { |ln| puts ln }  
end
```

Dynamic code evaluation



eval

- Executes dynamically
- Typically, `eval` takes a string:
`eval "puts 2+3"`
- Popular feature
 - especially in JavaScript
 - Richards et al. *The Eval that Men Do*, 2011
- Source of security problems

Additional Ruby eval methods

- `instance_eval`
 - evaluates code within object body
- `class_eval`
 - evaluates code within class body
- Take a string or a block of code
 - block of code more secure

String Processing



Regular Expressions in Ruby

```
s = "Hi, I'm Larry; this is my" +  
    " brother Darryl, and this" +  
    " is my other brother Darryl."  
s.sub(/Larry/, 'Laurent')  
puts s  
s.sub!(/Larry/, 'Laurent')  
puts s  
puts s.sub(/brother/, 'frère')  
puts s.gsub(/brother/, 'frère')
```

Regular Expression Symbols

- `/./` - Any character except a newline
- `/\w/` - A word character (`[a-zA-Z0-9_]`)
- `/\W/` - A non-word character (`[^a-zA-Z0-9_]`)
- `/\d/` - A digit character (`[0-9]`)
- `/\D/` - A non-digit character (`[^0-9]`)
- `/\s/` - A whitespace character: `/[\t\r\n\f]/`
- `/\S/` - A non-whitespace char: `/[^ \t\r\n\f]/`
- `*` - Zero or more times
- `+` - One or more times
- `?` - Zero or one times (optional)

References for Ruby

- "Programming Ruby: The Pragmatic Programmer's Guide", <http://ruby-doc.com/docs/ProgrammingRuby/>
- "Why's Guide to Ruby", <http://mislav.uniqpath.com/poignant-guide/> (unusual, but entertaining reference).
- David Black, "Ruby for Rails", 2006.

Lab: Eliza in Ruby

Use Ruby to model a psychiatrist.

<http://en.wikipedia.org/wiki/ELIZA>

Download eliza.rb from the course website and extend it. Note that if you call ``ruby eliza.rb -test``, you will get some cases to consider.