Hello Ruby

History

Ruby History

- Yukihiro "matz" Matsumoto started in 1993
- 1.0 public release in 1995
- Progress...
- Rails 1.0 released in 2005 by David Heinemeier Hansson
- Now!

Where are we now?

Over 225,000 Ruby repositories!

Over 24,000 Gems!

Features

- Dynamically-typed
- Interpreted
- Smalltalk-style OO
- Large community

Ruby Basics

Using the REPL

- Open a terminal
- Use the 'irb' command
- You now have an interactive console!

Running a file

- Open a terminal
- Use the command 'ruby <file_name.rb>'
- You've now run a file!

ruby test.rb

Open a REPL

Expressions

```
1 + 1 #=> 2
16 / 2 #=> 8

"foo" * 3 #=> "foofoofoo"
"hello world".split(' ') #=> ["hello", "world"]

100.odd? #=> false
"1337".to_i #=> 1337
```

Hello World!

```
puts 'Hello World'
puts "Hello World"
puts('Hello World')
puts('Hello World');
```

Symbols

- Singleton
- Immutable
- Typically used as hash keys or in DSLs

```
:symbol

"crash".to_sym  # => :crash
```

Variables!

```
foo = "Hello World!"
puts foo

foo = 10
puts foo + 1
```

Truth Table

Value	Result
false	false
nil	false
Anything else	true

Logical Operators and Precedence

```
:: .
[]
**
-(unary) +(unary) ! ~
*  /  %
+  -
<<  >>
&
|  ^
>  >=  <  <=
<=> == === != =~ !~
&&
||
.. ...
=(+=, -=...)
not
and or
```

All of the operators are just methods except these:

`=, ::, ., .., ..., !, not, &&, and, ||, or, !=, !~`

In addition, assignment operators (`+=` etc.) are not user-definable.

if

```
if 1 + 1 == 2
  puts "Math ftw!"
end
```

else

```
foo = nil

if foo
  puts "Something improbable!"
else
  puts "Something far more reasonable!"
end
```

unless

- unless is equivelant to if not

```
unless 2 + 2 == 5
  puts "This gets printed!"
end
```

Conventions

Conventions

- Naming
- When to use parens
- method?
- method!

Naming - Classes

- Classes should be named using CamelCase
  - User
  - UserController
  - AliceTheCamelHasFiveHumps

Naming - Vars

- Variables should be named using snake_case
  - user
  - user_person
  - all_stretched_out

Naming - Constants

- Constants should be CAPITALIZED
  - PI
  - CATEGORIES
  - SPEED_OF_LIGHT
  - C

Parens

- Parens are optional on method call

```
puts "Hello World"
puts("Hello World")
```

Use Parens

- When using multiple arguments
- When doing multiple things on that line
- When it improves the readabliity

```
"Hello World".split(' ').first
```

Don't Use Parens

- When calling a method with no arguements
- When working with a DSL
- When it seems wrong

```
puts 'Hello World'
```

method?

- Methods ending in `?` should return a boolean

```
foo.nil?
```

method!

- This should mutate the state of the value

```
foo = "hello world"
foo.upcase          # =>  "HELLO WORLD"
foo                 # =>  "hello world"

foo.upcase!         # => "HELLO WORLD"
foo                 # => "HELLO WORLD"
```

Basic Data Types

Strings

```
foo = "Hello World"

foo.upcase #=> "HELLO WORLD"
foo.downcase #=> "hello world"

foo.split(' ') #=> ["Hello", "World"]
```

String Interpolation

- Use string interpolation instead of concatenation

```
foo = 'world'

"Hello #{foo}!" #=> Hello world!
```

Hashes

```
hash = {:foo => :bar, :hash => :rocket}

hash[:foo] #=> :bar

hash #=> {:foo=>:bar, :hash=>:rocket}
```

Arrays

```
foo = [:foo, :bar, :baz]

foo[1] #=> :bar

foo.first #=> :foo

foo.last #=> :baz
```

Arrays

```
range = (1..10).to_a

range[1..3] #=> [2, 3, 4]

range.each do |num|
  puts num
end
```

each?!

- Looping is a code smell!
- Iteration is the default
- `each` works by passing a block

```
[1, 2, 3, 4, 5].each do |num|
  puts num * num
end
```

Blocks

- Passing code as an arguement to a method
- Inline anonymous functions
- Used by each, map, reduce and similar higher order functions

each_with_index

- Iterates but also provides an index
- Useful for when you'd normally use a for loop

```
ranked_names.each_with_index do |name, i|
  puts "#{i}: #{name}"
end
```

map

- Iterates over things and returns the results of the iteration

```
squares = [1, 2, 3, 4, 5, 6, 7].map do |num|
  num * num
end
```

reduce

- Takes a list and reduces it to a single value

```
fifteen = [1, 2, 3, 4, 5].reduce(0)
  do |sum, num|
    sum + num
  end
```

Classes and Methods

Methods

```
def hello_world
  puts "Hello world"
end

def force(mass, acceleration)
  mass * acceleration
end
```

Optional Arguements

```
def greet_user(name = nil)
  unless name
    puts "What is your name?"
    name = gets
  end

  puts "Hello #{name}"
end
```

Named Arguements

```
def build_tag(tag, cont, options = {})
  attrs = options.map do |attr, val|
    "#{attr}='#{val}'"
  end.join(' ')

  "<#{tag} #{attrs}>#{cont}</#{tag}>"
end
```

Using Named Args

```
tag = build_tag("a", "a link",
   :href => "localhost", :class => "blue")

tag #=> <a href='localhost' class='blue'>a
link</a>

other_tag = build_tag("b", "Bold!")

other_tag #=> <b >Bold!</b>
```

Defining a Class

```
class HelloRuby
  def initialize(greeting)
    @greeting = greeting
  end

  def greet
    puts "#{@greeting} ruby!"
  end
end
```

Using a Class

```
hello_ruby = HelloRuby.new("Hello")

hello_ruby.greet
```

Class Methods

```
class HelloRuby
  def self.fix_string(str)
    str.gsub(/foo/, 'bar')
  end
end
```

Macros

Things to Cover

- attr_accessor
- Idea that methods can define methods
- eval is availible

Getters and Setters

```
class Bean
   attr_accessor :amount
   attr_reader :name

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

Getters and Setters cont

```
bean = Bean.new("Sumatra")

bean.amount # => nil
bean.amount = 10
bean.amount # => 10

bean.name # => "Sumatra"
bean.name = "Kona" # => Error!
```

What did we do?

- attr_accessor uses Ruby's metaprogramming features

```
attr_accessor :name
# generates
def name
  @name
end

def name_=(thing)
  @name = thing
end
```

How can we use this?

```
class Logger
  if MODE == 'dev'
    def self.log(msg)
      puts msg
    end
  else
    def self.log(msg)
      write_log(msg)
    end
  end
end
```